中国科学院大学

University of Chinese Academy of Sciences

学士学位论文

一种高性能敏捷开发智能网卡的设计与实现

作者姓名：黄天羿

指导教师：陈明宇 研究员 中国科学院计算技术研究所

学位类别：工学学士

专　　业：计算机科学与技术

学院（系）：计算机科学与技术学院

2023 年 6 月

Design and implementation of a high-performance agile-developed smart NIC

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Bachelor of Engineering

in Computer Science and Technology

By

Huang Tianyi

Supervisor: Professor Chen Mingyu

School of Computer Science and Technology, University of Chinese Academy of Science

June, 2023

中国科学院大学
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的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对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或致谢。

作者签名：

日　　期：

中国科学院大学
学位论文授权使用声明

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：　　导师签名：

日　　期：　　日　　期：

摘 要

随着数据中心运营规模和强度的扩大，对高速网络的要求也逐渐提高。为了减少软件处理的性能开销与处理延迟，常使用智能网卡进行包处理分流的硬件卸载工作。目前已有的基于固定场景的高性能智能网卡设计中，存在功能开发难度高、定制灵活性不足等缺点。本文介绍了一种基于 FPGA 的可支持 100Gbps 带宽的智能网卡设计，使用基于 Chisel 的敏捷硬件开发方法对该智能网卡的功能模块进行实现。该智能网卡同时具有高性能与灵活可定制的优点，可作为扩展框架进行定制功能开发，除了高带宽的收发功能外，实现了包 TCP / IP 校验和计算、RSS 负载均衡、字段比较与搜索、正则表达式匹配与 AES 加解密等扩展功能。

关键词：高性能，可定制，敏捷开发，智能网卡，Chisel

Abstract

With the expansion of scale and intensity of data center operations, the requirements for high-speed networks have also gradually increased. In order to save the performance overhead of software processing, smart NICs are often used to process and distribute packets on hardware. The existing scene-oriented high-performance smart NIC have shortcomings such as high function-developing difficulty and poor customizing flexibility. This paper introduces an FPGA-based smart NIC design that can support 100Gbps bandwidth, and implements its functioning module using Chisel-based agile hardware development methods. The smart NIC combines high performance with customizing flexibility, which can be used as a framework for customized function development. In addition to behaving as a high-bandwidth transceiver, the smart NIC is implemented with extensive functions such as packet TCP / IP checksum generation and verification, RSS Load balancing, char matching / searching, regular expression matching, and AES encrypting / decrypting.

Keywords: high performance, customizable, agile development, smart NIC, Chisel

目 录

图形列表

表格列表

第 1 章　引论

在当前发达的互联网应用场景下，数据中心作为服务端，需要同时对大量用户的请求进行服务处理，其运营规模和运算强度也日益提升。为了保证高服务并发数与低服务延迟，同时维持集群在高速处理中的数据一致性，数据中心对高速网络的要求也逐渐提高。在数据中心高速网络中，为减小软件处理的性能开销与处理延迟，最大限度提升请求处理速度，常使用智能网卡实现包处理、包分流等各项工作的硬件卸载。

1.1　智能网卡简介

智能网卡 (Smart NIC) 是具有硬件加速能力的网卡。图1.1所示的是智能网卡的定义，除具有普通网卡的所有功能外，其还拥有独立的运算逻辑，能在硬件层面完成填充校验和、包分流等原本由软件运算逻辑完成的工作，大幅降低包处理过程中的 CPU 资源消耗与处理延迟，提高应用的性能。

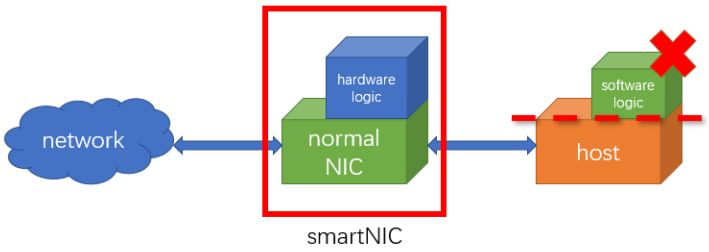


图 1.1 智能网卡的定义

Figure 1.1 The difinition of Smart NIC

智能网卡可基于 ASIC、FPGA 或 MP 进行实现[1]，这三种实现方式的优缺点与代表性商用产品如表1.1所示。下面对这三种实现进行分别介绍。

ASIC 实现使用传统的固化电路并流片的方式提供具有固定功能的智能网卡，该实现商用技术成熟，在量产时成本最低，性价比最高，但如果需要修改逻辑则需要经过漫长的设计开发与重新流片过程，灵活性最低。虽然 ASIC 定制化的逻辑对于技术成熟的应用场景已经能够提供显著的性能提升，但是随着新的应用场景与功能需求不断出现，采用 ASIC 实现智能网卡并不总是一个合适的选择。目前，仅基于 ASIC 实现的智能网卡种类较少，ASIC 技术在智能网卡中主要

表 1.1 智能网卡实现方式比较

Table 1.1 Comparison between implement method of smart NIC

智能网卡	ASIC	FPGA	MP
优点	成本低，性价比高 适合批量生产	兼具高性能与高灵活性 低延迟，低功耗	在网卡上集成 SoC 灵活性与可编程性最高
缺点	设计门槛高，周期长 可编程性低，迭代缓慢	设计复杂度高 开源生态不完善	高并发时性能较弱
代表产品	Amazon® Nitro Card NVIDIA® ConnectX-7	Intel® SmartNIC C5020X Xilinx® Alevo U25	NVIDIA® BlueField DPU Broadcom® NetXtreme

用于实现网络控制器部件，其中以 NVIDIA® 收购 Mellanox® 生产的 ConnectX 系列最为出名，其最新一代产品为 ConnectX-7，可进行可编程化的数据处理与硬件加速，支持虚拟化、SDN (Software Defined Networking，软件定义网络) 等技术，为敏捷高性能的网络解决方案提供助力。

FPGA 实现通过在具有网络通信能力的 FPGA 板卡上烧写用户定义的硬件逻辑实现智能网卡功能。该实现较为平衡，相比于 ASIC 实现具有更低的迭代成本与更高的定制性，而相比于 MP 实现具有更高的处理性能，可利用流水线结构并行处理数据，降低处理延迟并提高吞吐量；缺点是硬件逻辑设计复杂度高与开源生态受限，且开发者需要从厂商购买 IP 核，存在购买与开发成本。该实现的主要商用产品有 Intel® 的 SmartNIC 系列板卡与 Xilinx® 的 Alevo 系列板卡等；在学术界，使用 FPGA 实现智能网卡硬件逻辑的研究以 Microsoft® 的 Catapult 架构[2] 为代表，其将 FPGA 网络与数据中心网络融合，提出了针对可重构数据中心云服务器的加速方案。

MP 实现为使用片上系统 (system on chip, SoC) 的实现方案，采用片上多核的方式进行进行包处理与流控制，用户可以通过控制片上系统简单地实现智能网卡功能，具有最高的灵活性与可编程性，但在高并发场景下 SoC 的处理开销可能成为传输速率的瓶颈。在 MP 实现中使用的处理器核可以是专用的网络处理器，也可以是通用处理器，在使用专用处理器时性能相对更强。MP 实现的代表商用产品为目前热门的 NVIDIA® BlueField DPU，其将 ConnectX 系列智能网卡与多个可编程的 ARM 核组合，采用 ASIC 与 MP 的混合实现，在可编程性、性能与成本间实现平衡，通过卸载、加速和隔离高级网络、存储和安全服务，为任

何环境中的工作负载提供安全、高速与软件定义的基础架构。

1.2　基于 FPGA 的智能网卡发展现状

关于基于 FPGA 实现的智能网卡相关工作，可主要分为两类：

1.2.1　面向固定功能的智能网卡

可以利用 FPGA 本身较强的可开发性，将网卡与自编写的硬件算子结合，实现单一固定功能下的包处理硬件卸载，实现防火墙、科学计算等功能。下面以负载均衡功能与特征筛选功能为例说明。

1.2.1.1　负载均衡

在高速传输网络中，接收端多处理器核一对一绑定网卡接收队列并行处理接收包，此时智能网卡在收包时可以根据包的内容和处理权重等信息进行计算，并结合当前系统负载情况将包放入合适的接收队列，实现在多处理器核情景下的的负载均衡调度。常用的负载分流策略是使用接收端缩放 (Receive Side Scaling，RSS) 技术[3] 和对称哈希计算[4]，用特定的种子对包的源和目的 IP 地址与端口号进行 Toeplitz 哈希算法[5] 下的哈希计算，并根据结果查找表与分配队列，从而实现负载均衡。RSS++[6]、FGLB[7] 和 AlNiCo[8] 使用更为复杂精细的策略实现负载分流。

1.2.1.2　特征筛选

智能网卡可以提取接收包的特征并硬件上对其进行筛选，从而节省软件处理的 I/O 和计算开销，减小性能损耗。NIC-QF[9] 在分布式数据库系统里用于存储节点和计算节点之间的查询筛选操作，减少发送不需要的数据造成的网络开销。HyPaFilter[10] 和 HyPaFilter+[11]，通过设置硬件筛选器实现具有复杂灵活规则的高速网络防火墙。Hayashi 等[12] 将带有筛选器的网卡用于筛选传感器大数据集，仅接受存在异常模式的数据。

1.2.2　面向固定场景的智能网卡

在面向固定功能的基础上，可以开发面向固定场景的智能网卡，此时，智能网卡可以在同一场景下的实现多个自定义功能，通过组合多个功能实现协议栈处理、数据中心应用等复杂场景下的硬件卸载，通用性与可扩展性得到增强。下

面以 CPU - 网络场景与 GPU - 网络场景为例说明。

1.2.2.1 CPU - 网络场景

赛灵思 (Xilinx®) 官方的开源项目 OpenNIC[13]，在 FPGA 板卡上构建包含两个关键 IP 核 (在第3章介绍) 的硬件工程，以实现面向 CPU - 网络场景的智能网卡功能；在达到 100Gbps 传输带宽的同时，保留多个基于 AXI-Stream 总线协议的扩展盒，可以作为高性能高扩展性智能网卡的基础框架。

1.2.2.2 GPU - 网络场景

FpgaNIC[14] 是面向 GPU - 网络场景的高性能智能网卡，传输带宽可达 100Gbps，同时使用高层次综合 (HLS) 技术进行开发，支持载入用户自定义的硬件算子，同时实现 GPU 通信控制的硬件卸载，在大规模并行计算场景下有较好的应用。

1.3 本文工作

本文希望基于 FPGA 设计实现一种面向 CPU - 网络场景的，同时具有高性能与高扩展性的智能网卡框架，一方面支持用户进行针对特定需求的增量开发，并实现硬件框架和扩展功能的解耦合，用户在开发时不需知道底层硬件细节；另一方面使用更为高效便捷的硬件开发方法，减小用户在需要添加功能扩展时的编码工作量，降低硬件逻辑设计复杂度，有利于功能模块的高效开发与快速迭代。

本文智能网卡与 FpgaNIC、OpenNIC 项目的对比如表1.2所示。在可移植性上，FpgaNIC 基于特定 FPGA 板卡开发，没有考虑框架可移植性问题；OpenNIC 需要使用 Vivado tcl 脚本进行项目生成，且存在苛刻的板卡种类限制，如果用户没有使用特定型号的 FPGA 板卡，则无法使用该框架，此外，框架与 IP 核紧密捆绑，没有进行上层封装，用户仍需要了解硬件工程的设计细节。本文工作自主搭建框架，考虑到所需的两个 IP 核均为软核，理论上对于具有足够板上资源的 FPGA 板卡 (可参考第5章中提到的资源占用情况)，在提供相应约束文件后均可以烧写使用；对于功能模块，一方面部分屏蔽底层硬件细节，用户开发成本更小；另一方面实现标准化，只需具有相应的总线协议规范与传输模式，就可以在不同的硬件工程之间迁移。在主要开发语言上，OpenNIC 全部使用 SystemVerilog 语言开发，没有使用高级硬件设计语言进行设计，用户需要自定义功能时编码成本较高；FpgaNIC 的功能模块使用基于 C++ 的 HLS 技术进行开发，虽然已实现高

层次硬件设计，但 HLS 仍然存在资源使用大、综合速度慢、对编译器依赖大导致细节优化困难等缺点。本文工作使用 Chisel 作为功能开发语言，一方面实现高层次硬件设计，另一方面比 HLS 更加轻量化与贴近底层硬件实现，用户的易用性与开发自由度更高。

表 1.2 智能网卡工作比较

Table 1.2 Comparison of smart NIC works

智能网卡项目	FpgaNIC	OpenNIC	本文智能网卡
应用场景	GPU - 网络场景	CPU - 网络场景	CPU - 网络场景
硬件卸载内容	网络控制/运算模型	不包含，仅为框架	TCP/IP 包处理分流
可移植性	低	低	高
主要开发语言	基于 C++ 的 HLS	SystemVerilog	Chisel

本文工作可以分为三个部分，即：搭建不具备扩展功能的基础框架硬件工程；使用 Chisel 进行功能模块搭建与扩展功能开发；对智能网卡进行上板实现与测试。在第3章、第4章与第5章中，将分别对这三部分工作进行详细介绍。

1.4　主要贡献

本文设计实现一种基于 FPGA 的可支持 100Gbps 带宽的智能网卡，该智能网卡同时具有高性能与灵活可定制的优点，可作为扩展框架进行定制功能开发。本文工作搭建包含两个关键 Xilinx IP 核的高性能网卡基础框架以实现高带宽收发功能，并使用基于 Chisel 的敏捷硬件开发方法对智能网卡的功能模块进行开发；使用流水线结构进行包处理，并保留流水线的扩展功能接口，在包处理流水线上实现包 TCP / IP 校验和计算、RSS 负载均衡、字段匹配、正则表达式匹配与 AES 加解密功能作为示例，用户可参照这些功能的设计，实现符合自身需求的自定义功能。在用户侧使用 Data Plane Development Kit (DPDK) 框架对智能网卡进行控制，最大化提升用户程序的包处理能力与网络性能。在完成设计实现工作后，使用 IXIA 测试仪与自编写的 SimpleTxRx 软件等工具对智能网卡进行简单测试，测试结果显示，该智能网卡具有预期的高性能与高扩展性，在包长度为 1024 字节的情况下收发极限速率可达到约 100Gbps。各项扩展功能实现正确，除较复杂的正则表达式匹配与 AES 加解密功能外，其他功能不影响包传输速率。通过使用 Ping 软件进行测试，证明该智能网卡具有实际情景下的通用性。

第 2 章　背景

本章节的主要目标是介绍本文工作中使用的数种协议与算法的相关技术背景。

2.1　AXI 协议总线

AXI (Advanced eXtensible Interface) 协议[15] 是 ARM® 公司的 AMBA 微控制器总线系列的一部分，该系列在 2003 年公布的 AMBA 3.0 中包含第一个版本的 AXI 协议，并在之后的 2010 年公布的 AMBA 4.0 中包含 AXI 协议的第二个主版本，即 AXI4 协议。AXI4 协议具有高效性、高灵活性和高通用性，在 Xilinx® 的产品中被广泛使用。AXI4 协议总线有三个类型，即用于高性能内存映射的标准 AXI4 总线、用于简单低带宽内存映射的 AXI4-Lite 协议总线和用于高速数据传输的 AXI4-Stream 总线。

AXI4-Stream 总线协议的基础部分包括：

- tdata: 用于传输数据内容。
- tvalid: 代表发送端该拍数据有效。
- tlast: 代表该拍为当前数据包传输的最后一拍。
- tready: 接收方的反馈信号，代表接收端可以接收数据。tvalid 和 tready 同时置起时握手成功，此时进行数据传输。

此外，还有一系列扩展信号可供使用，在第3章中将会提及。

2.2　TCP / IP 校验和计算

TCP / IP 校验和计算包含校验和生成与校验和检查两部分。生成 IP 校验和时，将 20 字节的 IP 包头除校验和 (2 字节) 字段之外的所有部分以 2 字节为单位依次相加，得到一个 32 位的数，将该数的高 16 位和低 16 位相加[1]并将结果取反，得到 16 位的最终校验和计算结果。TCP 校验和的生成流程基本一致，但进行依次相加的范围为 TCP 伪首部 (由源 IP 地址、目的 IP 地址、传输层协议号、TCP

[1]在某些情况下相加后结果长度仍然超过 16 位，此时需要进行两次加法。对于所有 TCP / IP 校验和计算，高低位相加最多进行两次，就可以得到长度在 16 位以内的结果。

报文长度构成，这些信息可以从 IP 包头中得到) 和 TCP 报文 (报头和数据)，并将 TCP 校验和字段除外。检查校验和的过程和生成校验和的过程也基本相同，此时不需要排除校验和字段，完成依次相加后取反，如果得到的最终结果为 0，则校验和正确，否则校验和出错。

2.3 RSS 技术与 Toeplitz 算法

RSS 技术[3] 是一种网络驱动技术，主要的作用是在多处理器系统上进行网络接收包的高效分发，将当前接收包根据其特性分发到不同的接收队列，从而供不同的处理器核进行包处理，实现多处理器的负载均衡。经典的包分发策略是使用 Toeplitz 哈希算法，将 TCP / IP 包的源 IP 地址、源端口、目的 IP 地址与目的端口组成 96 位长的四元组，并对其进行哈希计算，根据计算的结果查找跳转表并确定放入的接收队列。

Toeplitz 哈希算法的原理如图2.1所示，对包的 {源 IP 地址，源端口，目的 IP 地址，目的端口} 四元组逐位进行遍历。对于第 n 位，如果为 0，则其对应的中间结果为 0；如果为 1，则其对应的中间结果为哈希种子左移 $(95 - n)$ 位后的最高 32 位。对所有的中间结果逐个进行异或计算，从而得到最终的哈希计算结果。对该算法设定特定的哈希种子[4]，可以实现对称哈希计算，即将四元组中的源 IP 地址 - 源端口和目的 IP 地址 - 目的端口对调时，哈希计算结果不变，以此将属于相同 TCP 传输流的包分发至相同的接收队列，方便与该队列绑定的处理器核进行专门处理，提升处理性能。

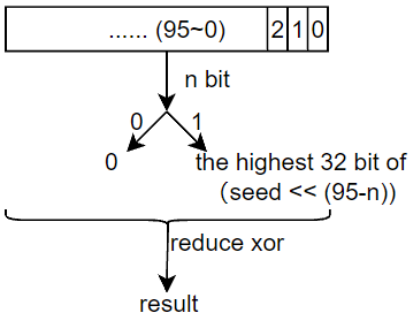


图 2.1 Toeplitz 哈希算法原理

Figure 2.1 Principle of Toeplitz hashing algorithm

2.4　AES-128-ecb 加解密算法

AES (Advanced Encryption Standard，高级加密标准) 算法[16] 是一种常见的分组对称加密算法，由美国国家标准技术研究所在 1997 年征集并评选得出，具有较高的安全性与流行性。AES 算法的工作方式为将原文拆成 128 位的加密块，每块进行多轮 4 步的加密操作，并将加密后的加密块拼接形成最终的密文。解密步骤和加密步骤基本相同，但每步的操作上存在差异。本文工作使用 AES-128-ecb 算法，128 代表密钥长度为 128 位，此时进行 10 轮加解密操作；“ecb”代表电子密码本模式 (Electronic Code Book)，即每块单独加解密，块与块之间没有关联性，这样能最大程度保证处理的并行性，便于硬件实现。

AES-128-ecb 算法的加解密步骤如图2.2所示, 左图为加密步骤, 其中第 1 - 9 轮进行蓝色箭头 (虚线) 步骤，第 10 轮进行红色箭头 (实线) 步骤；右图为解密步骤，其中第 1 轮进行虚线箭头步骤，第 2 - 10 轮进行实线箭头步骤。

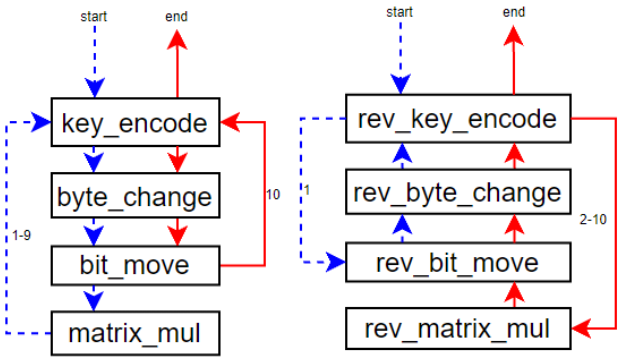


图 2.2 AES-128-ecb 算法加解密步骤

Figure 2.2 Encryption and decryption steps of AES-128-ecb algorithm

加解密时，对当前块的 16 字节，从 0 字节开始，将其从左上角开始以纵列从左到右放置在一个 4x4 的矩阵中，如图2.3所示。在加解密完成后，同样按照该规则将数据变回线性。

由图2.2，加密操作共有 4 种：

1. key_encode: 轮密钥加操作，与当前轮的轮密钥取异或。

2. byte_change: 字节代换操作。对矩阵中的每个字节，前 4 位作为行值，后 4 位作为列值，在算法给出的 S 盒 (代换表) 中查询，并将当前字节代换为查询结果。

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

图 2.3 AES-128-ecb 算法矩阵顺序

Figure 2.3 Matrix order in AES-128-ecb algorithm

3. bit_move: 行移位操作，对当前矩阵进行循环左移操作，第一行不操作，第二行左移 1 字节，第三行左移 2 字节，第四行左移 3 字节，得到本步的计算结果。

4. martix_mul: 列混合操作。将当前矩阵在阶为 256 的有限域 (GF(256)) 下与如图2.4所示的固定矩阵相乘，得到本步的计算结果。

0x2	0x3	0x1	0x1
0x1	0x2	0x3	0x1
0x1	0x1	0x2	0x3
0x3	0x1	0x1	0x2

图 2.4 列混合固定矩阵

Figure 2.4 The fixed matrix in mix-column process

同样地，解密操作共有 4 种：

1. rev_key_encode: 逆轮密钥加操作，与第 $(10-n)$ 轮的轮密钥取异或，n 为当前轮数。

2. rev_byte_change: 逆字节代换操作，与字节代换操作规则相同，但使用算法给出的逆 S 盒查询。

3. rev_bit_move: 逆行移位操作，与行移位操作规则相同，但使用右移。

4. rev_martix_mul: 逆列混合操作，与列混合操作规则相同，但固定矩阵变为图2.5所示矩阵。

由图2.2，AES-128-ecb 算法中，加解密步骤均需进行 10 轮共 40 步操作，可看作有特殊情况的循环: 加密步骤进行轮密钥加 - 字节代换 - 行移位 - 列混合的

0xE	0xB	0xD	0x9
0x9	0xE	0xB	0xD
0xD	0x9	0xE	0xB
0xB	0xD	0x9	0xE

图 2.5 逆列混合固定矩阵

Figure 2.5 The fixed matrix in rev-mix-column process

循环操作，但第 40 步更换为轮密钥加；解密步骤进行逆列混合 - 逆行移位 - 逆字节代换 - 逆轮密钥加的循环操作，但第 1 步更换为逆轮密钥加。所以，轮密钥加 / 逆轮密钥加操作共进行 11 次，而列混合 / 逆列混合操作共进行 9 次；首次轮密钥加 / 逆轮密钥加操作可以视为第 0 轮的操作，整个加解密步骤共需要第 0 - 10 轮共计 11 个 128 位的轮密钥。

对于 AES-128-ecb 轮密钥，初始 (第 0 轮) 轮密钥由参数给出，在第 1 - 10 轮中，第 n 轮的轮密钥 round_key(n) 可由如图2.6中给出的伪代码算法计算。图中，轮密钥以字节大端序放置[2]。s_substitute 函数代表使用 S 盒代换，两个参数为行与列，返回 S 盒代换的结果；get_round_con 函数代表获取一个和轮数相关的常量，在当前算法中，该常量在 $n \leq 8$ 时为 $1 << (n-1)$，在 $n = 9$ 时为 0x1b，在 $n = 10$ 时为 0x36。

```
round_key(n)[31:0] = round_key(n-1)[31:0] ^ {
    s_substitute(round_key(n-1)[103:100],round_key(n-1)[ 99: 96]), //0, 103:96
    s_substitute(round_key(n-1)[127:124],round_key(n-1)[123:120]), //3, 127:120
    s_substitute(round_key(n-1)[119:116],round_key(n-1)[115:112]), //2, 119:112
    s_substitute(round_key(n-1)[111:108],round_key(n-1)[107:104])  //1, 111:104
    } ^ get_round_con(n);
round_key(n)[ 63:32] = round_key(n-1)[ 63:32] ^ round_key(n)[31: 0];
round_key(n)[ 95:64] = round_key(n-1)[ 95:64] ^ round_key(n)[63:32];
round_key(n)[127:96] = round_key(n-1)[127:96] ^ round_key(n)[95:64];
```

图 2.6 AES-128-ecb 轮密钥生成算法

Figure 2.6 Round key generation algorithm in AES-128-ecb algorithm

[2]硬件上用大端序存放轮密钥，主要是出于需要在网络序 (大端序) 的数据包中进行轮密钥加的考虑。

第 3 章　智能网卡基础框架

本章节的主要目标是设计一个具有高带宽收发能力的 FPGA 网卡硬件工程，并将其作为基础框架，为第4章中的模块开发与功能实现打下基础。

3.1　基础框架结构设计

本文智能网卡硬件工程的整体结构设计如图3.1所示。

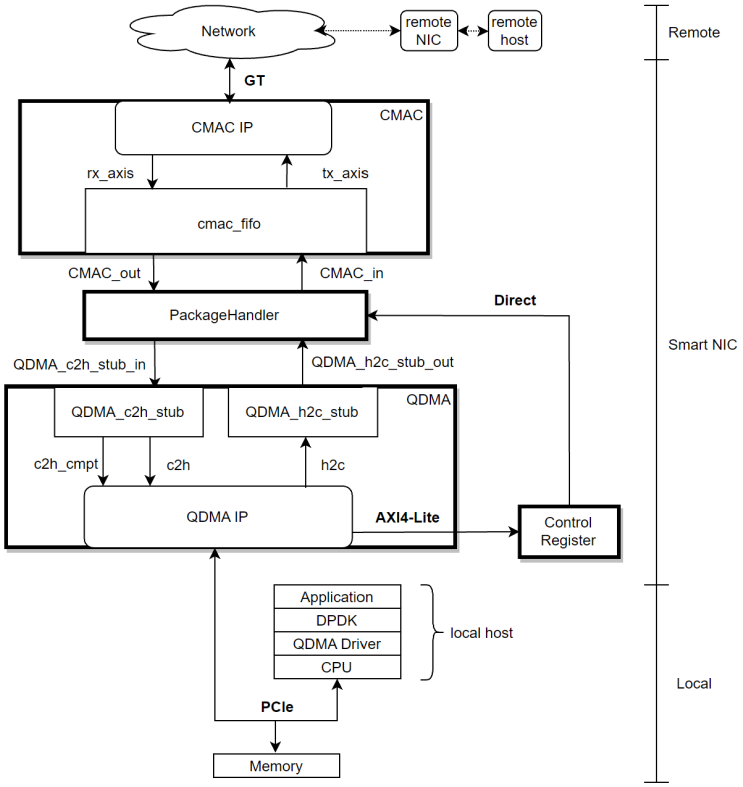


图 3.1 硬件工程整体结构设计

Figure 3.1 Overall design of hardware project

本地主机上的应用通过 DPDK 组件提供的驱动接口控制 CPU，使用 PCIe 协议总线与智能网卡通信；智能网卡使用 GT 协议总线连接板卡的物理网络接口，从而在网络上收发报文，与远端主机进行通信。由图3.1中可见，本文智能网卡设计主要包含四个功能组件[1]：

1. 主机端控制组件，用于管理与本地主机的交互，在图中标识为 QDMA。

[1]对下文中的“CMAC”和“QDMA”，除非专门指出其为 IP 核，否则均指代整个控制组件。

2. 主功能模块，用于实现包处理的硬件卸载，在图中标识为PackageHandler。
3. 网络端控制组件，用于管理与网络接口的交互，在图中标识为CMAC。
4. 控制寄存器堆，用于保存控制参数等内容，在图中标识为Control Register。

3.2 基础框架组件介绍

在图3.1的组件中，PackageHandler 是实现包硬件处理的主功能模块，也是本文工作的开发重点，所有扩展功能均在其上实现，其设计在第4章中介绍；除PackageHandler 外的组件设计是本章的主要介绍内容。下面对这些组件分别进行介绍。

3.2.1 模块间互联协议

图3.1中，特殊总线格式使用粗体标识，除被标识的部分总线外，其余总线均为 AXI4-Stream (AXIS) 协议总线。在本文工作中，AXI4-Lite 协议总线仅用于QDMA IP 核在驱动程序控制下和控制寄存器堆的交互，各模块间通信主要使用的是 AXIS 协议总线。需要注意的是，根据情境的不同，共使用 4 种 AXIS 协议总线，在基本的 AXIS 协议基础上加入与情境相关的扩展内容，这些总线的格式名称与构成如表3.1所示。4 种 AXIS 协议总线格式分别用于 QDMA 传输 (QDMA_ h2c_ stub_ out 与 QDMA_ c2h_ stub_ in)、CMAC 传输 (CMAC_in 与 CMAC_out；tx_axis 与 rx_axis) 和 PackageHandler 内部的发送 (Tx) /接收流水线 (Rx) 处理传输。

表 3.1 硬件工程中 AXIS 总线格式名称与构成

Table 3.1 AXIS bus format's name and composition in hardware project

IOBundle	tdata/tvalid/tready/tlast	tuser	tkeep	extern_config	info
QDMAAxisIO	√	√			
CMACAxisIO	√	√	√		
TxPipelineAxisIO	√			√	tx_info
RxPipelineAxisIO	√	√		√	rx_info

框架中 AXIS 总线具有 512 位的数据 (tdata) 位宽，其扩展内容包括：

- tkeep: 代表 tdata 中的有效部分，每 1 位与 tdata 的 8 位对应。

- tuser: AXIS 协议中规定的自定义字段。对于 QDMA 代表该拍是否为 QDMA 包头；对于 CMAC 代表该拍数据是否出现传输错误。
- extern_config (结构体) : 用于从控制寄存器堆中传入软件设置的参数，包括一个操作数 (op) 和 16 个参数 (arg)。通过改变传入参数，可以在软件上控制包处理行为，不需反复烧写硬件比特流。
- tx_info / rx_info (结构体) : 用于传输流水线处理上下文中需要随拍保存的内容，包括计算的 TCP/IP 中间校验和；对于 tx_info 结构体还包含从尾部算起的包最后一拍的无效字节数 (mty) ，对于 rx_info 结构体还包含包长度 (tlen) 和接收队列 (qid) 。

3.2.2　主机端控制组件

主机端控制组件包含 QDMA IP 核与 QDMA_ h2c (c2h) _ stub 模块。

3.2.2.1　QDMA IP 核

QDMA IP 核为 Xilinx® 给出的 IP 核，其全称为“QDMA Subsystem for PCI Express”[17]，是带有队列特性的直接存储器访问模块 (Direct Memory Access，DMA) ，通过 PCIe 接口与主机的 CPU 和内存通信，提供异步高速数据传输，可以减少向内存传输数据时的 CPU 控制开销；同时将数据包分发到多个硬件队列中供软件读取，从而实现包的分流处理。本文工作中，QDMA 与 DPDK 软件的 QDMA 驱动配合，负责主机侧的包收发与控制工作，如按照包的队列 ID (包含在 QDMA 包头中) 进行多队列分流、根据用户的驱动函数调用执行相应的操作等。QDMA IP 核是实现包收发功能的主机端硬件基础。

在传输数据时，整个以太网帧作为数据包，在 AXIS 总线的 tdata 信号上传输，使用 tlast 信号进行数据包的界定。在识别包时，对于 QDMA IP 核，由于需要将包与指定队列关联，以及进行异步传输控制，需要额外的控制信息，该信息以 QDMA 包头的形式传输。该包头使用 tuser 信号标识，tuser 置 1 时本拍为 QDMA 包头。QDMA 包头在每次传输以太网帧的前一拍传输，其中包含包长度与队列 ID 等信息，QDMA 包结构如图3.2所示，方框上方数字为拍序号，一拍 pld (数据载荷，payload) 包括 64 字节 (512 位) 数据，所有 pld 的数据组成一个以太网帧。进出主机端控制组件的包必须满足 QDMA 包的格式，而进出网络端控制组件的包不应具有 QDMA 包头，其中的控制由 PackageHandler 实现。

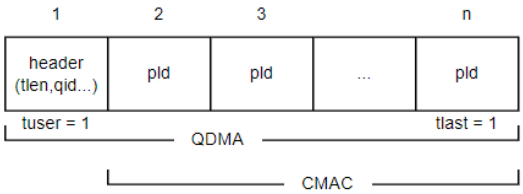


图 3.2 QDMA 包结构

Figure 3.2 QDMA packet's Structure

3.2.2.2 QDMA_ h2c (c2h) _ stub

QDMA_ h2c_ stub 和 QDMA_ c2h_ stub 模块是 QDMA IP 核的外围模块，用于生成与消费 QDMA 包头。模块名字中的“c2h”和“h2c”分别表示“card to host”和“host to card”，即对于主机的接收 (Rx) / 发送 (Tx) 。在上面几个部分中提到的使用 QDMAAxisIO 总线格式的“QDMA 总线”，实际上指的是图3.1中的 QDMA_ h2c_ stub_ out 和 QDMA_ c2h_ stub_ in，而与 QDMA IP 核相关的 h2c、c2h 和 c2h_cmpt 使用另外的特殊 AXIS 总线标准，其中的格式转换由这两个模块完成。

QDMA_ h2c_ stub 的结构如图3.3所示，图中实线箭头代表 QDMA 包头 (cmp[2]) 通路，虚线箭头代表 pld 通路，点状线代表传递包状态信息。QDMA_ h2c_ stub 的输入为 QDMA IP 核发出的数据。QDMA IP 核在 h2c 方向仅有一条总线，用于传输包数据 (h2c，即图中的 QDMA_ h2c_ stub_ in)，控制信息通过经过加宽的 tuser 信号传输。该模块包括一个小的 input FIFO，对每个包根据第一拍 pld 的 tuser 信号附带的控制信息生成一拍 QDMA 包头，并在该包的所有 pld 前发出。in_context 是模块内保存的控制信息上下文，该上下文在包的第一拍被填入，在整个包的传输过程中均有效。

QDMA_c2h_stub 的结构如图3.4所示。QDMA_ c2h_ stub 的输入为 Package-Handler 构造的 QDMA 包。QDMA IP 核在 c2h 方向上具有两条总线，分别用于传输 QDMA 完成信息 (c2h_cmpt) 和包数据 (c2h) ，由 QDMA_ c2h_ stub 进行分流。该模块包含 3 个小 FIFO，入口为一个 input FIFO，在该 FIFO 的出口处根据 tuser 信号进行分流，把 tuser 为 0 的数据拍放入 payload FIFO，以向 QDMA IP 核

[2]在本节介绍中，cmp 和 cmpt 均代指 completion 信息，即完成信息；该信息可以视为与 QDMA 包头等价，用于 QDMA IP 核在完成接收传输时向 CPU 通知，可以简单地理解为控制信息。(发送时不使用完成信息，仅使用该名称代指 QDMA 包头)

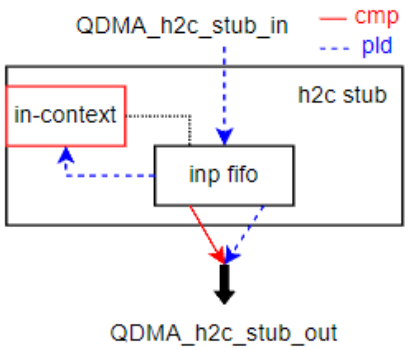


图 3.3 QDMA_h2c_stub 模块设计

Figure 3.3 Design of QDMA_h2c_stub module

的 c2h 通道传输数据；把 tuser 为 1 的 QDMA 包头拍放入 completion FIFO，并填充 in_context 控制信息上下文，以提供 c2h 通道的 tuser 信号包含的 QDMA IP 核控制信息。completion FIFO 储存 QDMA 包头，用于 QDMA IP 核的异步传输控制。当一个数据包的内存写入工作完成时，QDMA IP 核会在 c2h_cmpt 通道拉高一拍的 tready 信号，代表本次数据传输已经完成，此时 completion FIFO 内储存的该包的 QDMA 包头生成 QDMA IP 核需要的完成信息并通过 c2h_cmpt 通道传输，QDMA IP 核根据该信息通过 PCIe 总线向 CPU 告知当前包接收完成，此时软件得知接收完成的信息。

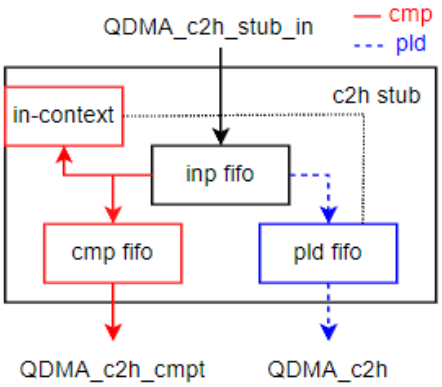


图 3.4 QDMA_c2h_stub 模块设计

Figure 3.4 Design of QDMA_c2h_stub module

3.2.3　网络端控制组件

网络端控制组件包含 CMAC IP 核与 cmac_fifo 模块。

3.2.3.1 CMAC IP 核

CMAC IP 核为 Xilinx® 给出的 IP 核，其全称为“UltraScale+ Integrated 100G Ethernet Subsystem”[18]，通过 GT 接口与 FPGA 板卡的百 G 网口相连，用于在网络界面进行包的发送与接收。CMAC IP 核是实现包收发功能的网络端硬件基础。

在工程设计中，考虑到 CMAC IP 核与前述的 QDMA IP 核还存有较为复杂的周边信号规则与模式设置要求，将其封装为 Vivado 内的 Block Design 实现。在 Block Design 内部添加两个 IP 核，并根据手册要求规范连接两个 IP 核的时钟、复位、常量设定等信号，最终其对外暴露的只有两个 IP 核的总线接口，即 CMAC IP 核的 GT 总线与 AXIS 总线 (tx_axis、rx_axis)、QDMA IP 核的 PCIe 总线、AXI4-Lite 总线与 AXIS 总线 (QDMA_h2c、QDMA_c2h、QDMA_c2h_cmpt) 。

3.2.3.2 cmac_fifo

cmac_fifo 模块的作用是进行 QDMA IP 核和 CMAC IP 核之间的时钟频率转换，以及对包进行缓冲。本文工作中除 CMAC IP 核和 cmac_fifo 模块，其他部分均运行在 QDMA IP 核时钟域内。QDMA IP 核的默认时钟域 (以下简称为“QDMA 时钟域”) 为 250MHz，而 CMAC IP 核的默认时钟域 (以下简称为“CMAC 时钟域”) 为约 322.25MHz[3]，两者存在不一致，为保证 IP 核间的协同运行，需要进行时钟域间的转换。此外，虽然表3.1中显示 CMAC 总线包括 tready 信号，但是实际上由于接收数据时无法对网络远端进行总线信号握手，从 CMAC 接收数据的总线 (即图3.1中的 rx_axis 总线) 中不包括 tready 信号，CMAC IP 核默认每拍 tvalid 都能被接收，如果接收方无法接收，则数据直接丢失。为减少丢包，需要为 CMAC IP 核的接收端准备一个缓冲区，暂时存放等待被处理的包。cmac_fifo 模块的设计需要同时解决这两个问题。

cmac_fifo 模块的设计如图3.5所示。cmac_fifo 模块由 3 个子模块构成，这 3 个子模块均为 Xilinx® 的“AXI4-Stream Data FIFO”IP 核，使用 CMACAxisIO 总线格式，在包模式[4]下运行。tx_fifo 和 rx_fifo 使用 FPGA 板上的 BRAM 资源，这两个 FIFO 容量较小，但是具有转换时钟域的能力，可以在 QDMA 时钟域和 CMAC 时钟域间转换 (图中实线箭头为 QDMA 时钟域，虚线箭头为 CMAC 时钟域) 。rx_uram_fifo 使用 FPGA 板上的 URAM 资源，这个 FIFO 容量较大，用于

[3]精确值为 322.265625MHz。

[4]Packet Mode；相比于传统 FIFO 的以拍为单位，在该模式下以 tlast 界定的包为单位缓冲。

对 CMAC 接收端进行缓冲。考虑到 rx_axis 没有 tready 信号的反馈机制，设置接近满 (prog_full) 信号与配套的处理逻辑 (点状线箭头)，以在 FIFO 可能无法接收包时停止本次接收，防止出现收包溢出导致接收错乱的情况[5]。

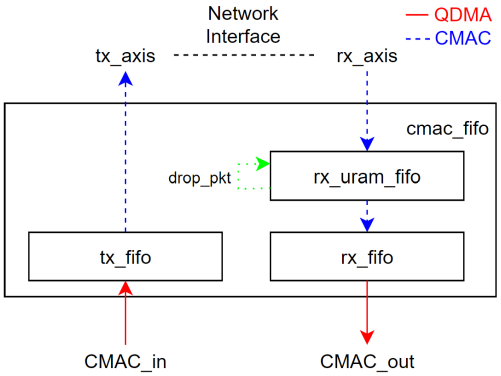


图 3.5 cmac_fifo 模块设计

Figure 3.5 Design of cmac_fifo module

3.2.4　控制寄存器堆

控制寄存器堆 (Control Register) 用于对 QDMA IP 核的模式与状态进行控制，在软件上可以通过 DPDK 框架的 QDMA 驱动函数根据地址直接对其进行读写，QDMA IP 核通过 AXI4-Lite 总线与寄存器堆进行交互。考虑到 PackageHandler 模块在实现软件控制功能时也需要传参，可以在该模块中增加一些新的地址与对应寄存器，并直接将这些寄存器的输出引入 PackageHandler 模块，从而实现软件向硬件的参数传递。该模块的部分地址映射如表3.2所示。寄存器堆中，0x100 - 0x10c 地址对应的是发送和接收方向的包总数计数器和错误计数器，用于在硬件层面统计收发包信息，其中 c2h_err_counter 除一般的长度溢出情况外，还在 TCP / IP 校验和检查功能开启时统计校验和错误的情况。这 4 个硬件计数器集成在 PackageHandler 模块内实现，寄存器堆只提供地址映射与读取功能。0x110 地址对应计数器重置寄存器，通过向这个寄存器先写 1 后写 0，可以实现 4 个硬件计数器的数值重置。0x114 - 0x154 地址的对应寄存器是 op 和 16 个 arg，这些参数寄存器控制整个 PackageHandler 模块的行为，它们的具体作用在介绍 PackageHandler 模块时会具体介绍。

[5]区分 rx_fifo 和 rx_uram_fifo 的原因是，BRAM 资源通用性好，在硬件工程的各个模块中使用频率高，可减少使用以节约资源，优化布局布线；URAM 资源在板上丰富，但不能转换时钟域，可用于大容量缓存。

表 3.2 寄存器堆部分地址映射表

Table 3.2 Address mapping table of Control Register (partly)

地址	寄存器名	注释
0x100	h2c_pack_counter	发送方向包总数计数器
0x104	h2c_err_counter	发送方向包错误计数器
0x108	c2h_pack_counter	接收方向包总数计数器
0x10c	c2h_err_counter	接收方向包错误计数器
0x110	reset_counter	计数器重置寄存器
0x114	op	PackageHandler 操作数
0x118 - 0x154	arg0 - arg15	PackageHandler 参数

通过将上面介绍的组件组合在一起，可以得到智能网卡的高性能基础框架，该框架具有数据位宽大、运行时钟频率高、使用支持高速率的 IP 核和物理接口等优点，实现了高性能网络传输。在第4章中，将对 PackageHandler 模块进行框架设计，并添加多项扩展功能，以实现智能网卡的包处理硬件卸载功能。

第 4 章　智能网卡功能模块

本章节的主要目标是对基础框架中的 PackageHandler 模块进行开发，使用基于 Chisel 的硬件敏捷开发方法构建高层次框架，在实现屏蔽硬件细节的同时，在框架内搭建收发包处理流水线结构，实现各基础扩展功能。使用 JetBrain® 推出的 IntelliJ IDEA 作为 IDE，配合 Chisel 官方给出的设计模板 chisel - template[19] 进行 PackageHandler 模块的设计与实现；使用的 Chisel 版本为 3.5.1。

4.1　Chisel 敏捷硬件开发方法

Chisel[20] 是一种基于 Scala 的开源硬件描述语言，具有参数化硬件生成器和设计重用等高级语言特性，可以用于更加高效的数字逻辑设计。开发者在使用 Chisel 设计硬件结构时，同时具有高效性和高兼容性：一方面，可以利用 Scala 的面向对象高级特性，更加简练优化地描述硬件结构；另一方面，完成设计后，通过编译能够直接生成对应的 Verilog 代码，从而兼容所有的经典 EDA 软件，不需要厂商额外进行适配。使用 Chisel 进行开发时，由于修改硬件结构的成本小于使用 Verilog 开发，可以付出很低的时间与开发成本即完成硬件功能的改进迭代，实现类似软件开发的敏捷硬件开发。

PackageHandler 模块是本文智能网卡的主功能模块，其主要作用是在 QDMA 和 CMAC 之间进行包的硬件处理工作，是智能网卡中的自定义硬件逻辑部分。PackageHandler 模块是用户在自定义功能时需要直接进行修改的模块，与硬件工程的耦合度相较于基础框架部分更小，更新频率更高，且需要实现的功能更加复杂。考虑到 Chisel 的前述优点，使用 Chisel 进行该模块的开发工作。

4.2　PackageHandler 结构设计

在上一章图3.1的基础上，PackageHandler 模块的顶层结构设计如图4.1所示。PackageHandler 对包的处理分发送 (Tx) 与接收 (Rx) 两个方向进行，发送方向接收从 QDMA_ h2c_ stub 模块传出的数据包，进行发送相关的硬件包处理后发送给 cmac_fifo 模块，最终通过 CMAC 在 FPGA 板卡的网口上发出；接收方向接收从 cmac_fifo 模块传出的数据包，进行接收相关的硬件包处理后发送给 QDMA_

c2h_ stub 模块，最终通过 QDMA IP 核向内存中传输包数据并告知主机包接收已经完成。除上面提到的 4 条用于传输数据的总线外，还需要引入参数寄存器值[1]，并连接与硬件计数器相关的信号 (引入重置计数器值，引出 4 个硬件计数器值) 。

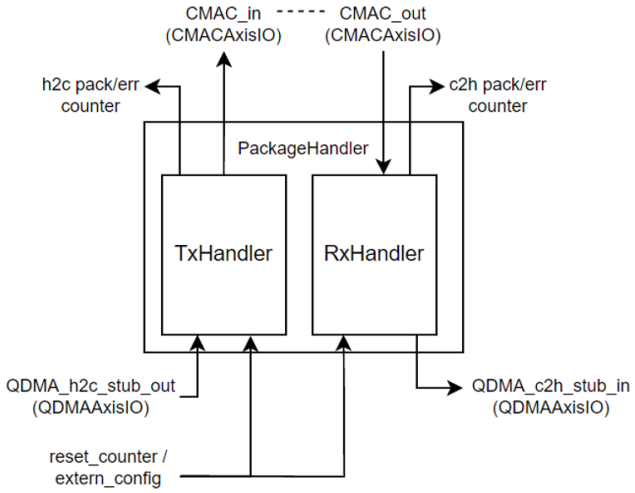


图 4.1 PackageHandler 模块顶层设计

Figure 4.1 Top-level design of PackageHandler module

4.3 PackageHandler 组件介绍

考虑到两个方向的包处理工作是相对独立的，在 PackageHandler 模块内切分两个功能子模块，称为 TxHandler 和 RxHandler，各进行一个传输方向的总线格式转换和包处理流程。进一步展开 TxHandler 和 RxHandler，其结构设计如图4.2所示。

由图4.2可见，TxHandler 和 RxHandler 均由三个子模块组合而成：

- Converter: 转换器，进行总线格式转换、包长度统计等预操作。
- Pipeline: 可扩展的包处理流水线，通过插入用户定义的 PipelineHandler 子模块对包进行逐级硬件处理。
- BufferFIFO: ping - pong Buffer，对流出流水线的包进行缓冲，并根据其携带的上下文信息进行汇总处理。

对于 QDMA 包头的处理，由于发送包时 QDMA 包头包含的信息不被使用，在发送方向上，需要丢弃从 QDMA_ h2c_ stub 模块发出的数据包的 QDMA 包头；

[1]参数寄存器实现在寄存器堆内，但实际代码中在 PackageHandler 模块的顶层还设置一组暂存参数寄存器，从而优化综合时的布线时序。

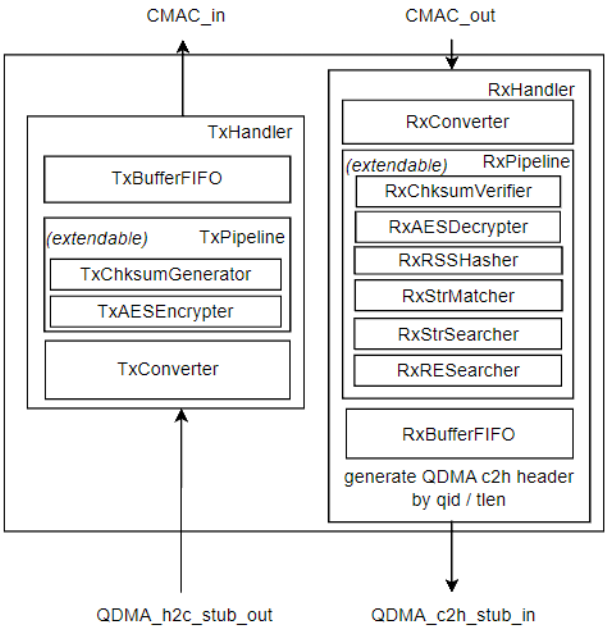


图 4.2 TxHandler / RxHandler 模块设计

Figure 4.2 Design of TxHandler / RxHandler module

在接收方向上，需要生成一个新的 QDMA 包头，此时需要包长度 (tlen) 和包队列 ID (qid) 信息，包长度在 RxConverter 内统计，而包队列 ID 则在 RxPipeline 中处理时生成，以实现根据包内容分发到不同接收队列的分流效果。最终在数据包可以流出 RxBufferFIFO 时，由 RxHandler 根据相应信息生成 QDMA 包头，并在数据包前一拍传输给 QDMA_ c2h_ stub 模块。

下面对 Converter、Pipeline 和 BufferFIFO 三个组件逐一进行介绍。

4.3.1　Converter

Converter 的主要作用是进行总线格式转换，将 QDMA / CMAC 使用的总线格式转化为 TxPipeline / RxPipeline 使用的总线格式。对于 TxConverter，其在 QDMAAxisIO 总线格式的基础上丢弃 tuser 为 1 的拍 (即 QDMA 包头)，初始化 tx_info 结构体，并和外部传入的 extern_config 结构体一同构成 TxPipelineAxisIO 总线格式；根据 QDMA 包头中含有的包长度信息，对包长度非 64 字节对齐情况下的包内容进行处理 (清空包尾边界外部分，以免计算错误)，并将包长度信息转化为 mty 字段，填入 tx_info 结构体中。对于 RxConverter，不需要丢弃拍，CMAC 的 tuser 信号留到 RxBufferFIFO 进行处理。除将 CMACAxisIO 总线格式转换为 RxPipelineAxisIO 总线格式外，还根据其 tkeep 和 tlast 信号统计当前数据

包的长度，并将最终结果放置在数据包最后一拍的 rx_info 结构体中。

4.3.2 Pipeline

Pipeline 是一个搭载 PipelineHandler 的框架，进出 Pipeline 的数据包使用对应的 PipelineAxisIO 总线格式，用户只需要遵循这一总线格式，就可以在该框架上开发任意功能的 PipelineHandler 并插入 Pipeline 中，从而实现用户自定义的包硬件处理功能。得益于 Chisel 的使用，一方面可以使用模块连接 (“<>”操作符) 简单地连接具有一致总线格式的不同 PipelineHandler，避免重复接线造成的冗余工作量与代码；另一方面，可以使用 Chisel 语言的继承特性，开发一个 PipelineHandler 基底模块，包含流水级寄存器以及常用的控制逻辑，具有基本的流水级握手功能。如果用户需要进一步开发自己的 PipelineHandler，只需要了解该基底模块并对在新开发的模块中使用继承即可，省去开发过程中总线握手逻辑调试的时间，使开发自定义功能更加简单，代码也更简洁。

在本文工作中，在实现 TxPipeline / RxPipeline 和对应的 PipelineHandler 基底模块的同时，也在该框架上实现如下扩展功能：

- TCP/IP 校验和计算与检查
- RSS 负载均衡
- 字段匹配，包括指定位置字段比较与字段搜索
- 正则表达式匹配
- AES 加密与解密

这些扩展功能对应的 PipelineHandler 在 Pipeline 中按照图4.2所示的顺序逐级连接。在“PackageHandler 扩展功能”章节中，将逐一介绍这些扩展功能的设计实现，用户可以其为参考，自行设计扩展功能组件。

4.3.3 BufferFIFO

BufferFIFO 是以 ping - pong 模式工作的 Buffer，它具有 2 个包缓存单元，通过向这 2 个包缓存单元轮流读写数据[2]，可以实现无需等待的高速数据流处理。BufferFIFO 也使用 Chisel 进行实现，一个包缓存单元支持最大 2048 字节的包。具有基本的缓存功能、包总数统计、包错误 (长度溢出) 统计并丢弃等功能。

[2]ping - pong Buffer 的理想工作模式是放入数据和取出数据完全错开，2 个包缓存单元处于一个正在读取，一个正在写入的状态。

根据传输方向和 Pipeline 功能的不同，BufferFIFO 也具有不同的独特功能：

对于 TxBufferFIFO，在启用 TCP / IP 校验和生成功能时，如果当前包为 TCP / IP 包，则需要在输出时根据当前包最后一拍 tx_info 结构体附带的 TCP / IP 校验和计算结果，在包内容的对应字段插入 TCP / IP 校验和。此外，在包长度超过一拍 (64 字节) 时，根据 tx_info 结构体中附带的 mty 信息，在包的最后一拍生成非全 1 的 tkeep 信号，从而发出非 64 字节对齐大小的包。由于 CMAC IP 核支持处理的最小包长度为 64 字节，如果包长度小于 64 字节，则在包尾填充 0 生成 64 字节的包。

对于 RxBufferFIFO，在启用 TCP / IP 校验和校验功能时，如果当前包为 TCP / IP 包，则需要在输出时检查当前包最后一拍 rx_info 结构体附带的 TCP / IP 校验结果。如果校验结果不为 0，则该包的 TCP / IP 校验和出现错误，直接丢弃包并计入包错误硬件计数器。除此之外，RxBufferFIFO 还有两个功能，一是丢弃 CMAC 传输错误 (tuser 为 1) 的包并计入包错误硬件计数器，二是在包流出之前向 RxHandler 顶层的 QDMA 包头生成逻辑传递保存在包最后一拍的 rx_info 结构体中的包长度和包队列 ID 信息，在发出第一拍 QDMA 包头后才能允许包流出。

4.4　PackageHandler 扩展功能

在 TxPipeline / RxPipeline 的扩展框架上，设计一系列扩展功能，并基于基底模块实现对应功能的 PipelineHandler，这些 PipelineHandler 在图4.2中也有所体现。可以使用 extern_config 结构体中的 op 与 arg 对它们的工作模式进行控制，控制规则如图4.3所示。图中，左侧列代表 op 的位号 (第 9 - 15 位保留)，中间列代表位号控制开关的对应 PipelineHandler (第 0 - 2 位是 RxStrMatcher 的子操作数，代表其匹配的功能)，右侧列代表在该位启用时 16 个参数寄存器的用处。

通过设置这些参数寄存器，可以通过软件设置寄存器值的方式改变智能网卡的功能，从而实现高度功能集成与更高的灵活性。可以在智能网卡设计时集成所有的功能，而用户在需要使用不同的功能或调整参数时，不需要修改硬件设计并重新生成比特流与烧写，只需要调用 QDMA 驱动函数，根据参数规则调整对应的寄存器即可，大幅提高智能网卡的实用性。

下面对扩展功能与对应的 PipelineHandler 逐个进行介绍。

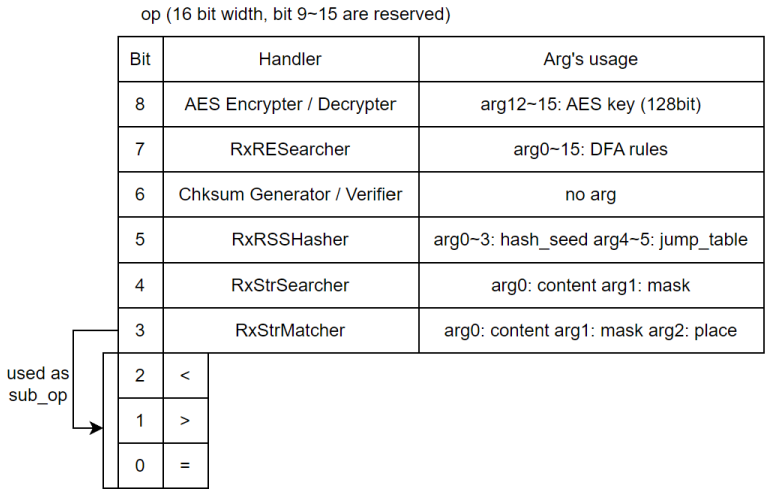


图 4.3 参数寄存器控制规则

Figure 4.3 Control rules of parameter register

4.4.1 TCP / IP 校验和处理

TxChksumGenerator 和 RxChksumVerifier 用于在收发包时对类型为 IPv4 的数据包生成 / 检查 IP 校验和，进一步地，对类型为 TCP 的数据包生成 / 检查 TCP 校验和。将 op[6] 置位可以开启该功能。在该功能开启时，软件在发出 TCP / IP 包时可以将校验和字段置 0，当该包离开网口时，已经完成正确校验和的计算与填充；从网络端接收到的包如果是 TCP / IP 包且校验和错误，则将其丢弃并统计。

在实现上，由于 TCP 校验和的生成和检查需要将整个 TCP 报文进行分段并依次相加，最终的校验和计算结果存放在包最后一拍的 tx_info 结构体中，在 BufferFIFO 模块对该信息进行判断与操作。考虑到高性能处理对减少操作拍数的要求，为最大限度发挥硬件并行性优势并优化时序，使用 Chisel 的 ReduceTree 方法生成加法归约树进行依次相加的操作，并进行切分优化，最终在一拍内即可完成当前拍传输的 64 字节数据的相加操作，在流水线中不会降低包的传输速度。在其他功能中，当需要进行类似的数据批量操作时，也通过生成计算归约树的方法优化时序与减小延迟。

4.4.2 RSS 负载均衡

RxRSSHasher 在收包时以 arg0 - 3 为哈希种子，对包中的源与目的信息进行 Toeplitz 哈希算法下的哈希计算，并将计算结果在 arg4 - 5 的跳转表中查找，将

结果作为当前包的接收队列 ID，从而实现包的分流。将 op[5] 置位可以开启该功能。在流水线中，该模块不会降低包的传输速度。

实际实现时，使用归约树进行逐个取或计算。考虑到进行 TCP / IP 包计算时需要 128 位的哈希种子，用 arg0 到 arg3 这 4 个 32 位的参数寄存器按 0 - 1 - 2 - 3 顺序拼接得到最终使用的 128 位的哈希种子。此外，将 arg4 与 arg5 两个寄存器值按 4 - 5 顺序拼接成的 64 位值从高位到低位视为一个跳转表数组，每 4 位代表一个跳转值，如 arg4 值为 0x01234567，代表哈希结果为 0x0 - 0x7 时，由与哈希结果相同的接收队列接收。在计算出哈希结果后，取其最低 4 位并查找对应下标的跳转表值，从而确定接收队列号。该实现支持 16 接收队列、哈希掩码为 0xf (即哈希结果只取最低 4 位) 的哈希分流。

4.4.3　字段匹配

RxStrMatcher 和 RxStrSearcher 用于对接收包的内容字段进行匹配，根据匹配结果确定该包放入的接收队列 ID: 如果匹配成功则接收队列 ID 为 1，否则接收队列 ID 为之前流水级传递的接收队列 ID (默认为 0) 。在流水线中，这两个模块均不会降低包的传输速度。RxBufferFIFO 综合包的整体信息时只使用包最后一拍中的 rx_info 结构体内的信息，所以在这两个模块进行匹配时，如果匹配成功，在该拍之后的所有当前包的数据拍内保持接收队列 ID (值为 1) 不变。

4.4.3.1　指定位置比较

RxStrMatcher 用于对接收的包进行指定位置的字段比较。arg0 中存放比较内容 (content) ，arg1 中存放比较掩码 (mask) ，arg2 中存放比较位置 (place) ；在接收包时，选取从包的第 place 个字节开始的 4 个字节作为比较窗口，与 content 字段进行 mask 掩码下的比较。mask 掩码的设置可以使比较的粒度降低。将 op[3] 置位可以开启该功能，op 的第 0 - 2 位是该模块的子操作数，在开启该功能时，用于设置比较方式：第 0 位代表等于，第 1 位代表大于，第 2 位代表小于，如果三位都不置位代表不等于；这些子操作数可以同时置位，以实现大于等于和小于等于的比较，例如，op 为 0x000b 时，最低 4 位的二进制表示为 1011，代表开启指定位置比较功能，并进行大于等于的比较，此时如果包中指定字段大于等于 content 字段，则比较成功。

实际实现时，由于对于一个包只需要一个比较窗口，不需要使用规约树。例

化一个计数寄存器代表本拍的数据在当前包中的字节位置，并使用移位计算抽出对应的 4 字节包内容进行比较即可，对于比较窗口跨拍的情况需要单独进行处理。

4.4.3.2 搜索

RxStrSearcher 用于对接收的包进行字段搜索。arg0 中存放比较内容 (content)，arg1 中存放比较掩码 (mask)。与上一节的指定位置比较不同的是，本模块中不需给出指定位置，而将整个包的内容进行 4 字节窗口的 mask 掩码下的搜索，如果包中搜索到 content 字段，则搜索成功。将 op[4] 置位可以开启该功能。

实际实现时，由于字段可能出现在包内的任意位置，为不造成流水线阻塞，希望能够在一拍内完成当前拍数据的内容搜索，这表示对包内的每个连续的 4 字节组合，都需要一个搜索窗口。具体实现如图4.4所示。对于一拍数据 (64 字节)，从 0 - 60 字节中每个字节开始的连续 4 字节都是一个窗口，拍内有 61 个窗口；如果当前拍不为包第一拍，则还需加上前一拍的 61 - 63 字节开始的跨拍窗口，共有 64 个窗口。将所有搜索窗口的比较结果逐个取或 (使用规约树)，即可得到当拍数据的搜索的结果；对包内的每拍数据都进行该操作，则得到整个包的搜索结果。

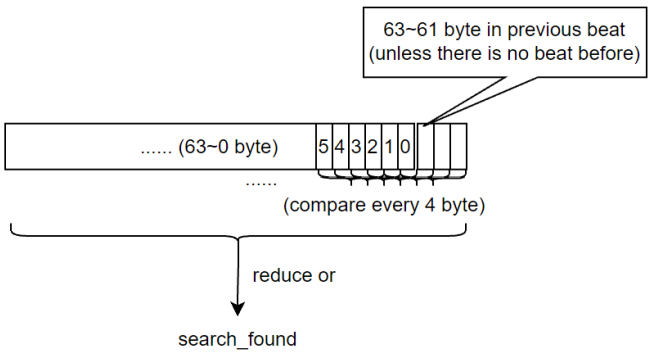


图 4.4 RxStrSearcher 模块搜索原理

Figure 4.4 Principle of searching in RxStrSearcher module

4.4.4 正则表达式匹配

RxRESearcher 用于对接收的包进行正则表达式的匹配。正则表达式匹配与字段匹配的区别在于，字段匹配的长度有限制，可以通过设置窗口的方式实现，而正则表达式是模式匹配，符合正则表达式的字段不定长，且有多种满足匹配条

件的方式，需要使用状态机进行实现。正则表达式匹配的参数为正则表达式对应的 DFA 状态机规则，对接收包的内容逐个字节输入状态机并进行状态跳转，如果状态机跳转到接收状态则认为匹配成功，包的接收队列 ID 为 1，否则认为匹配失败，接收队列 ID 为之前流水级传入的 ID (默认为 0) 。将 op[7] 置位可以开启该功能，arg0 - 15 存放输入的 DFA 状态机规则[3]，本模块支持 16 个状态 (对应 16 进制状态号 0 - f，使用状态号代表状态) ，16 条规则的状态机跳转，该状态机默认状态 0 为起始状态，状态 f 为接受状态，如果输入字符没有对应的状态机规则，在非接受状态下则跳至起始状态，在接受状态下则不做状态改变。由于状态机是逐个字节比较的，后一个字节的状态跳转依赖于前一个字节的状态跳转结果，在该模块中，为得到一拍数据的匹配结果，需要进行多拍的匹配操作，这会造成流水线阻塞，降低包的传输速度。

本模块参数中的状态机规则格式如图4.5所示。一条规则的长度为 32 位，用一个参数寄存器存储，其中字段从高到低为：下一个状态 (4 位) ，当前状态 (4 位) ，保留字段 (6 位) ，是否取非 (1 位) ，是否为范围跳转 (1 位) ，字符 2 (8 位，ASCII 码) ，字符 1 (8 位，ASCII 码) 。当范围跳转位开启时，符合状态机规则的字符集为 ASCII 码大于等于字符 1 且小于等于字符 2 的字符，否则仅为字符 1 和字符 2 两个字符；当取非位开启时，符合状态机规则的字符集为字符 1 和字符 2 考虑范围跳转位后代表的字符集整体取非得到的结果。例如，一条规则的值为 0xf0036361，其中‘a’的 ASCII 码为 61，‘c’的 ASCII 码为 63，该条规则标识的含义为：在当前状态为状态 0 (即起始状态) 时，如果输入的字符不是‘a’到‘c’的字符 (即‘a’、‘b’、‘c’)，则跳转到状态 f (即接受状态) 。当参数中仅有该规则时，匹配的正则表达式为“![a-c]”。通过引入范围跳转和取非功能，本模块有能力对具有变长和多字符匹配规则 (如包含‘-’、‘+’、‘*’、‘|’等符号) 的简单正则表达式进行匹配。

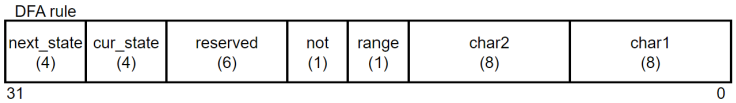


图 4.5 RxRESearcher 模块状态机规则格式

Figure 4.5 Format of DFA rule in RxRESearcher module

[3]本模块中，所有 DFA 状态机规则都必须是互斥的，否则匹配结果无法确定。

实际实现时，采用逐级构建跳转状态机的方法，如图4.6所示。首先构建 REHandlerUnit 子模块。该模块仅由组合逻辑构成，输入一个字符 (8 位数) 、当前状态和有参数设定的所有 DFA 规则，根据输入字符和输入状态匹配 DFA 规则，并输出下一个状态。未匹配到规则则输出 0，当前状态为 f 则输出 f。接着，将多个 REHandlerUnit 进行串联组成 REHandler 子模块，并在子模块输出处增加使能信号与一个结果寄存器。REhandlerUnit 的串联个数 (图中的 step) 在使用 Chisel 开发时已经进行参数化处理，在时序允许时可以调节该参数，以在一拍内比较多个字节，提高正则表达式匹配速度。在 REHandler 外围增加一系列控制逻辑以考虑不同匹配情况，使得完成正则表达式匹配时对当前包的剩余拍不再进行匹配操作，以提高包传输速率；同时增加保存当前状态的寄存器，匹配过程中将 REHandler 的输出作为其下一拍的输入，提高其重用性，最终构成 RxRESearcher 模块。在本文工作中，RxRESearcher 模块每拍进行 1 个字节的跳转，对于一拍数据，最多需要花费 64 拍才能完成匹配。

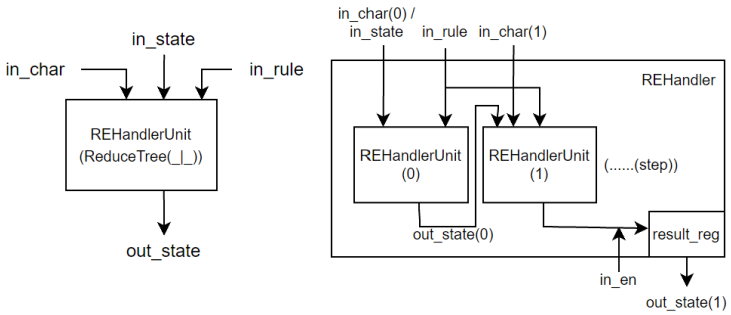


图 4.6 RxRESearcher 模块子组件

Figure 4.6 Sub-assembly of RxRESearcher module

4.4.5 AES 加解密

TxAESEncrypter 和 RxAESDecrypter 用于对包进行指定密钥下的 AES 算法加解密。开启加解密功能时，发送与接收方向上的包如果长度大于一拍 (64 字节) ，则对其第二拍开始的 payload 根据指定密钥进行 AES-128-ecb 算法加解密操作。加解密操作按 16 字节对齐, 不足的部分在后面补 0。将 op[8] 置位可以开启该功能，arg12 - 15 储存 AES 加解密操作的密钥，将这四个寄存器按 12 - 13 - 14 - 15 顺序拼接，可以得到最终的 128 位 AES 密钥。将密钥放入最后 4 个参数寄存器，是为方便其与前面的分流功能模块共同运作。在该模块中，对一拍数据需要进行

多拍的加解密操作，这会造成流水线阻塞，降低包的传输速度。

实际实现时，每拍数据需要进行 40 拍的加解密步骤，每拍进行 1 步操作；对于每次新设置 AES 密钥的第一拍数据，需要先进行 10 拍的轮密钥生成过程，将生成的轮密钥存储于寄存器中待用，之后的加解密过程中，如果没有设置新的 AES 密钥，则之前生成的轮密钥可以复用，不需要重新生成。对于列混合 / 逆列混合操作的矩阵乘法，考虑到有限域运算的性质，可以使用异或和移位递归进行实现；硬件寄存器上的数据存放顺序写法（大端序）与算法和软件的数据存放顺序写法（小端序）有较大差别，在实现时需要特别注意；操作中的矩阵变换操作在实现时也同样需要先映射到寄存器上。考虑到 AES-128-ecb 算法加解密的数据块需要 16 字节对齐，为不影响 TCP / IP 包头，从包的第二拍数据开始加解密操作，实际使用时需要将以太网帧内的包头总长 (以太网帧首部、IP 包头、TCP 包头长度之和) 填充到 64 字节，且在包长度非 16 字节对齐的情况下，包尾被填充 0 直到 16 字节对齐；由于 TCP / IP 校验和生成 / 检查功能设计上在 AES 加解密功能的外层进行，当这两个功能同时开启时，基于 AES 加密后的包进行 TCP / IP 校验和的生成与检查，此时对于软件接收到的包，其数据已经完成解密，会造成软件上出现 TCP 校验和错误的提示，但该提示可以忽略不计，硬件上已经确认校验和的正确性。

至此，已经完成 PackageHandler 模块的搭建与扩展功能的开发，从而完成智能网卡的硬件设计。在下一章中，将对智能网卡进行上板实现，并对其进行功能与性能上的简单测试。

第 5 章　智能网卡实现测试

本章节的主要目标是对智能网卡的硬件设计进行综合与实现，并在实际的硬件环境中进行扩展功能与极限性能上的简单测试，以对智能网卡开发成果进行评估。

5.1　开发环境

本文工作使用的开发环境如表5.1所示。

表 5.1 开发环境

Table 5.1 Development environment

开发环境		作用
硬件环境	带有 FPGA 板卡的服务器一台	智能网卡设计与调试
	高性能交换机（100Gbps）一台	连接智能网卡与外部测试机器
	IXIA 测试仪（发包机）一台	进行性能与功能测试
	外部服务器一台	进行网卡间连通性测试
软件环境	CentOS 7	服务器操作系统
	Vivado 2020.1	硬件工程设计与调试
	IntelliJ IDEA 2022.3.2	进行基于 Chisel 的硬件开发
	DPDK stable 19.11.6	通过驱动与智能网卡交互

5.2　实现情况

按照第3章和第4章中的设计，使用 Xilinx® 推出的 Vivado 2020.1 软件对硬件工程进行搭建与调试，使用的 FPGA 板卡为 Sugon Netfirm，子型号为 xczu19eg-ffvc1760-2-e。

在搭建硬件工程时，对于 PackageHandler 模块，采用第4章所述的 Chisel 敏捷硬件开发方法进行开发。对于 PackageHandler 模块总体与各扩展功能 (PipelineHandler) 模块，其 Chisel 实现代码行数与使用编译器生成的 Verilog 代码行数如表5.2所示 (对于各个 PipelineHandler 的 Chisel 代码，未统计基底模块、总线定义与规约计算树组件等部分) 。对于同一硬件模块，使用 Chisel 描述相比使用

Verilog 描述抽象程度与复用程度更高，代码行数可降至 10%，相应地，开发工作量与迭代成本大幅降低，体现出 Chisel 硬件敏捷开发方法的优势。

表 5.2 实现代码行数比较

Table 5.2 Comparison of line count in implement code

扩展功能模块		Chisel 代码行数	Verilog 代码行数
TCP/IP 校验和	生成	36	679
	检查	35	689
RSS 负载分流		34	768
字段匹配	比较	45	256
	搜索	40	692
正则表达式匹配		80	712
AES 加解密	加密	74	2262
	解密	74	3233
PackageHandler 模块总计		1248	13472

在调试硬件工程时，一方面可以使用硬件调试核 (ILA 核)，对调试中的硬件工程进行综合与实现并烧写上板后，使用软件进行收发包操作，并抓取指定信号的实际的硬件波形，从而排查问题的可能原因；另一方面，由于 Vivado 综合与实现硬件工程所需的时间较长，可以在硬件工程基础上修改，搭建不包含 CMAC IP 核和 QDMA IP 核的回环简单仿真工程，使用仿真激励文件输入 AXIS 协议总线信号，从而实现对 PackageHandler 模块的初步功能验证，快速发现问题。

完成调试后，对功能正确的硬件工程进行综合与实现，时序约束情况如图5.1所示，板上资源占用情况如图5.2所示。由图可见，硬件工程满足时序约束条件 (即对于触发器置起与保持的最低时序余量均大于 0ns)，硬件设计在板上不存在触发器介稳态的干扰，能够得到可靠性高的实现；板上资源占用合理，在满足时序约束的前提下，仍然有供用户进行自定义功能的扩展空间。

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.030 ns	Worst Hold Slack (WHS):	0.010 ns	Worst Pulse Width Slack (WPWS):	0.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	387755	Total Number of Endpoints:	387703	Total Number of Endpoints:	152669

All user specified timing constraints are met.

图 5.1 硬件工程时序约束情况

Figure 5.1 Timing constraints of hardware project

Summary

Resource	Utilization	Available	Utilization %
LUT	125732	522720	24.05
LUTRAM	12964	161280	8.04
FF	127048	1045440	12.15
BRAM	202.50	984	20.58
URAM	20	128	15.63
DSP	3	1968	0.15
IO	1	512	0.20
GT	20	48	41.67
BUFG	14	940	1.49

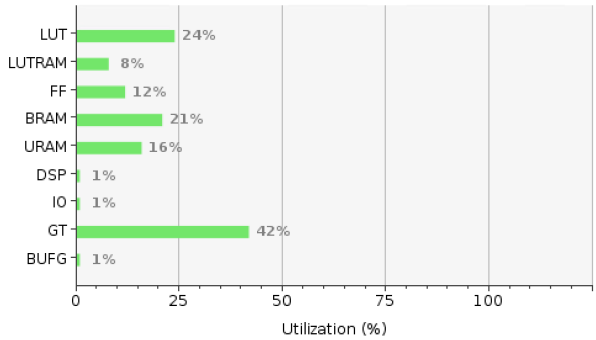


图 5.2 硬件工程资源占用情况

Figure 5.2 Resource utilization of hardware project

5.3 测试环境

测试的硬件环境上，使用 IXIA 测试仪 (发包机) 作为外部设备，对智能网卡进行收发包测试，使用支持 100Gbps 传输带宽的高性能交换机作为中介；为保证智能网卡的处理通用性，进行网卡连通性测试，此时将发包机与交换机的连接断开，并使用带有通用网卡的外部服务器连接交换机。搭建的测试环境如图5.3所示。

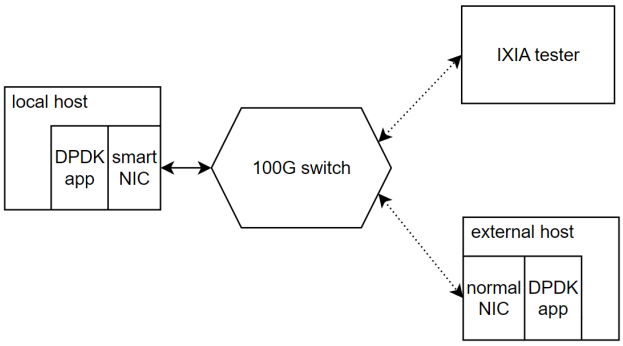


图 5.3 测试环境

Figure 5.3 Test Environment

测试的软件环境上，基于 DPDK stable 19.11.6 套件进行测试。DPDK[21] 是 Intel® 开发的数据平面开发套件，通过使用 UIO 技术和精心设计的库函数绕过 Linux 内核协议栈，在用户态对数据进行收发与处理。使用的测试程序由 DPDK 套件内提供的 l2fwd 样例程序修改而成，该程序可以在 DPDK 套件目录下的 examples 文件夹中找到。l2fwd 程序的原本功能是监听接收端口并对接收到的包进行数据链路层转发，只有 1 个收包队列，同时也只使用 1 个处理器核进行收包-处理-发包的过程，这一方面无法达到高速传输的处理性能要求，另一方面也不利于对网卡硬件分流功能的测试。在其基础上修改，并将修改后的软件称为 SimpleTxRx。该软件具有如下功能：

- 不再进行转发，而软件直接构造 TCP / IP 包进行高速率发送。（参考发包机构造的 TCP / IP 包）
- 使用 16 个接收队列进行收包，1 个队列进行发包，每个队列绑定专属的处理器核进行处理。
- 支持包 IP-端口四元组和包内容的 dump 功能。
- 支持初始化软件环境时设置智能网卡参数。

- 打印更加详细的收发包信息 (如每个队列的接收包个数、以及硬件计数器的当前值)，使用专门的处理器核打印信息。

在与发包机交互的测试中，使用 SimpleTxRx 软件进行测试[1]；在网卡连通性测试中，选用 DPDK 套件下 ping 软件的服务端作为测试软件进行测试。

5.4　测试方法与结果

对智能网卡的测试包含如下几个方面：

1. 不同包长度下的的极限收发速率测试。
2. TCP / IP 校验和的生成 / 检查正确性测试。
3. RSS 负载分流均匀性与 Toeplitz 算法实现正确性测试。
4. 字段匹配 (指定位置比较 / 搜索) 功能正确性测试。
5. 正则表达式匹配极限速率测试与功能正确性测试。
6. AES 加解密极限速率测试与功能正确性测试。
7. 网卡连通性测试 (Ping 软件) 。

下面对测试的具体方法和结果分别进行介绍。

5.4.1　极限收发速率测试

为测得指定包大小下的智能网卡极限收发速率，在测量发送速率时，使用 SimpleTxRx 持续以最高速率发送该大小的包，在发包机控制软件的数据统计界面中统计接收速率；在测量接收速率时，首先使用发包机以极限发送速率向智能网卡发送 10 亿个该大小的包[2]，并检查 SimpleTxRx 的收包情况，根据其收包比率确定智能网卡此时的极限接收速率。使用该方法测得的智能网卡极限收发速率如表5.3所示。在包长度在 512B 以下时，智能网卡极限接收速率小于极限发送速率，这可能与软件处理小包的速度不足有一定关系；包长度达到 1024B 后，发包机能够以极限发送速率运行，智能网卡极限接收速率基本达到 100Gbps。在之后的 TCP / IP 校验和、RSS 负载分流和字段匹配测试中，均采用 1024B / 极限发送速率 (此时为 98.09Gbps) 作为包大小与发包机发送速率。

[1]在实际测试中，由于高速传输时软件处理速度和实际硬件环境条件的限制，在接收包时仍然会有极少量丢包的情况，可以忽略不计。

[2]考虑到发包机发包时的包长度计入以太网帧结尾的 4 字节 CRC 校验码，而其在智能网卡硬件和 SimpleTxRx 软件处理时均已丢弃，在测试时，发包机发送的包长度应设为待测包长度 + 4。

表 5.3 智能网卡收发极限速率

Table 5.3 Sending and receiving limit rate of Smart NIC

包大小 (B)	极限发送速率 (Gbps)	极限接收速率 (Gbps)
64	20.97	14.63
128	38.17	28.27
256	65.73	53.87
512	93.76	91.75
1024	98.09	98.09
1450	98.64	98.64

5.4.2 TCP / IP 校验和测试

对 TCP / IP 校验和相关功能的测试分为两个部分，即校验和生成测试与校验和检查测试。

5.4.2.1 校验和生成测试

使用 SimpleTxRx 构建 1024B 大小的 TCP / IP 包，对每个 TCP / IP 包，使用随机数据填充源 IP 地址字段、目的 IP 地址字段以及 TCP 报文的数据部分，不填充 TCP / IP 校验和字段。向发包机以极限发送速率发送 10 亿个包，发包机接收包时，将检测该包的 TCP / IP 校验和是否出现问题并统计。发包机上统计的实验结果如图5.4所示。发包机接收到的包 TCP / IP 校验和均没有出现错误，说明 TCP / IP 校验和生成功能正常。

IPv4 Packets Received	1,000,000,000	0
UDP Packets Received	0	0
TCP Packets Received	1,000,000,000	0
IPv4 Checksum Errors	0	0
UDP Checksum Errors	0	0
TCP Checksum Errors	0	0

图 5.4 TCP / IP 校验和生成测试结果

Figure 5.4 Test result of TCP / IP checksum generation

5.4.2.2 校验和检查测试

使用发包机以极限发送速率发送 10 亿个具有随机源 IP 地址字段、目的 IP 地址字段以及 TCP 报文数据部分的 1024B 大小的 TCP / IP 包，分别将 IP 和 TCP 校验和设置为无效 (即发包机随机填充无效的校验和)，并进行尝试，SimpleTxRx

得到的结果均如图5.5所示。IP 校验和与 TCP 校验和中任意一个出现错误，均会导致当前数据包被丢弃并统计入接收方向的包错误硬件计数器。

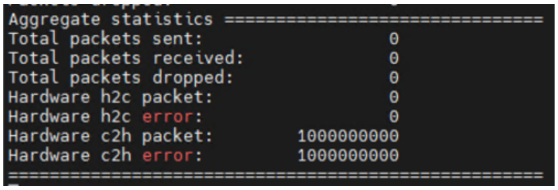


图 5.5 TCP / IP 校验和检查测试结果 (错误校验和)

Figure 5.5 Test result of TCP / IP checksum verification (wrong checksum)

如果 IP 校验和与 TCP 校验和均设置为正确，则 SimpleTxRx 得到的结果如图5.6所示，此时包能够正常被软件接收。

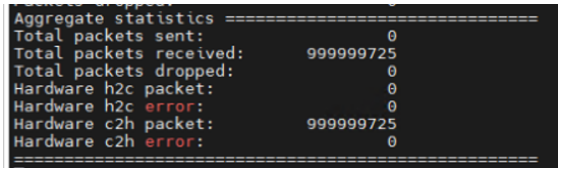


图 5.6 TCP / IP 校验和检查测试结果 (正确校验和)

Figure 5.6 Test result of TCP / IP checksum verification (right checksum)

5.4.3　RSS 负载分流测试

对 RSS 负载分流功能的测试分为两个部分，即分流均匀性测试与 Toeplitz 算法实现正确性测试。

5.4.3.1　分流均匀性测试

使用发包机以极限发送速率发送 10 亿个具有随机源 IP 地址字段与目的 IP 地址字段的 1024B 大小的 TCP / IP 包，开启 RSS 负载分流功能，设置 128 位的随机哈希种子，使用 16 个接收队列接收包，并将跳转表设置为哈希结果与接收队列直接一一对应，即将 arg4 与 arg5 分别设置为 0x01234567 与 0x89abcdef。此时分流结果如图5.7所示。接收到的数据包均匀分配在 16 个接收队列中[3]。

如果将 arg4 与 arg5 分别设置为 0x00002244 与 0x6688aacc，则分流结果如图5.8所示。接收到的包被分配到 0 / 2 / 4 / 6 / 8 / 10 / 12 号接收队列中，其中 0 号队列接收到的包是其他队列的两倍，行为符合预期，跳转表功能正确运行。

[3]考虑到使用随机功能，接收队列内的数据包数目存在小幅浮动。

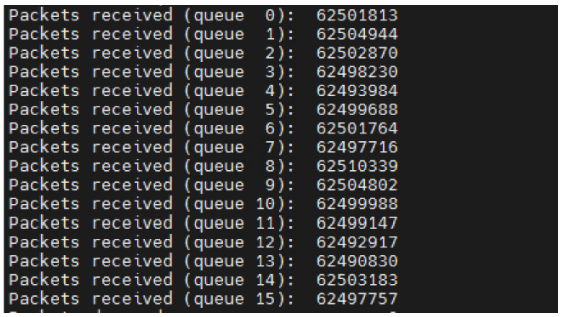


图 5.7 RSS 分流均匀性测试结果 (1)

Figure 5.7 Test result of RSS balancing uniformity (1)

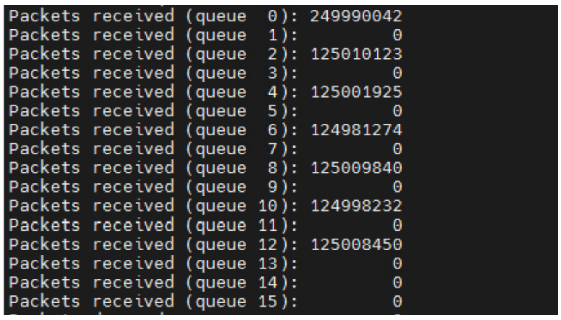


图 5.8 RSS 分流均匀性测试结果 (2)

Figure 5.8 Test result of RSS balancing uniformity (2)

5.4.3.2 Toeplitz 算法实现正确性测试

在分流均匀性测试中第一个实验的情况下，使用软件对 Toeplitz 算法的实现正确性进行测试。使用 SimpleTxRx 的 dump 功能，在 16 个接收队列上进行 1 / 10000 的包 IP-端口四元组抽样，并使用软件在同样的哈希种子下对被抽样的四元组进行哈希计算，将结果与当前被分配到的接收队列号进行比对，如果存在不一致则测试失败。软件实现的 Toeplitz 算法代码如图5.9所示，其原理在第2章中已有所介绍。其中, 数组 s 包含 96 位的四元组，数组 hash_seed 包含 128 位的哈希种子，均使用 2 个 64 位数存储 (不足 128 位时末尾填充 0) 。

该测试的结果如图5.10所示，软件计算结果和硬件分流到的接收队列号完全一致，测试通过，说明 Toeplitz 算法实现正确。

5.4.4 字段匹配测试

对字段匹配功能的测试分为两个部分，即指定位置比较测试与字段搜索测试。

```c
unsigned int sw_hash(unsigned long int s[],unsigned long int hash_seed[]){
    unsigned int sw_result = 0;
    unsigned long int test[2],seed[2];
    test[0] = s[0];
    test[1] = s[1];
    seed[0] = hash_seed[0];
    seed[1] = hash_seed[1];
    int i;
    for (i=0;i<96;i++){
        if (test[0] >> 63) sw_result ^= (seed[0] >> 32);
        test[0] = (test[0] << 1) | (test[1] >> 63);
        test[1] = test[1] << 1;
        seed[0] = (seed[0] << 1) | (seed[1] >> 63);
        seed[1] = seed[1] << 1;
    }
    return sw_result & 0xf;
}
```

图 5.9 Toeplitz 算法软件实现代码

Figure 5.9 Software implement code of Toeplitz Algorithm

```
[root@localhost simpleTxRx]# ./toeplitz_test
queue  1's test result success!
queue  5's test result success!
queue  4's test result success!
queue  3's test result success!
queue  6's test result success!
queue  7's test result success!
queue  8's test result success!
queue  9's test result success!
queue 11's test result success!
queue 13's test result success!
queue 12's test result success!
queue 10's test result success!
queue 14's test result success!
queue 15's test result success!
queue  2's test result success!
queue  0's test result success!
[Overall] test success!
```

图 5.10 Toeplitz 算法软件测试结果

Figure 5.10 Software Test result of Toeplitz Algorithm

5.4.4.1 指定位置比较测试

选取源 IP 字段作为指定位置比较测试的目标字段，该字段在包内位置为 26-29 字节 (从 0 字节开始计算) 。使用发包机以极限发送速率发送 10 亿个 1024B 大小的 TCP / IP 包，其中 5 亿个包具有小于 128.0.0.0 的随机源 IP 地址，另外 5 亿个包具有大于等于 128.0.0.0 的随机源 IP 地址，将参数设置为该字段大于等于 128.0.0.0 时匹配成功。测试结果如图5.11所示。0 号队列和 1 号队列平分所有包，功能实现正确。

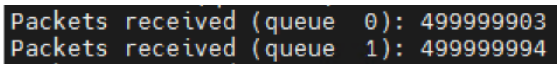


图 5.11 指定位置比较测试结果 (1)

Figure 5.11 Test result of specified location comparison (1)

如果将设置调整为 7 亿个包具有不等于 128.0.0.0 的随机源 IP 地址，另外 3 亿个包具有等于 128.0.0.0 的源 IP 地址，将参数设置为该字段等于 128.0.0.0 时匹配成功，此时测试结果如图5.12所示。0 号队列与 1 号队列的收包数目比为 7 : 3，功能实现正确。

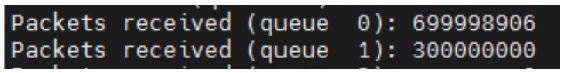


图 5.12 指定位置比较测试结果 (2)

Figure 5.12 Test result of specified location comparison (2)

5.4.4.2 字段搜索测试

使用发包机以极限发送速率发送 10 亿个 1024B 大小的 TCP / IP 包，这 10 亿个包按5.4中所示的 10 个包的队列循环生成。(包的 TCP 数据中其余位置用 0 填充)

此时在包中搜索“test”字段，测试结果如图5.13所示。0 号队列与 1 号队列的收包比为 2 : 8，对应未插入字段与插入“test”字段的包个数的比值，功能实现正确。

5.4.5 正则表达式匹配测试

对正则表达式匹配功能的测试分为两个部分，即匹配极限速率测试与匹配功能测试。

表 5.4 字段搜索测试包循环列表

Table 5.4 Circular list of packets in field search test

包编号	插入字段	插入位置 (B)
1	“test”	100
2		150
3		300
4		450
5		600
6		750
7		900
8		1000
9 / 10	不插入	——

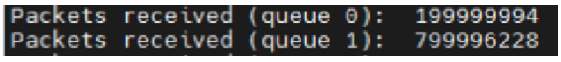


图 5.13 字段搜索测试结果

Figure 5.13 Test result of field search

5.4.5.1　匹配极限速率测试

在开启正则表达式匹配功能时，对于一拍数据需要进行多拍的匹配过程，这会阻塞收包流水线，降低包接收速率；当完成正则表达式匹配时，对后续内容不再进行匹配而直接通过，可以提升匹配成功后的包接收速率。所以，对于正则表达式匹配功能，当包进入的第一拍即完成匹配，后续内容不进行匹配，正则表达式匹配速率最高；当包中不含有匹配内容，此时对于每个数据拍，都需要花费 64 拍进行匹配，正则表达式速率最低。在开启正则表达式功能时，将 arg1 - arg15 设置为 0x0，分别将 arg0 设置为 0xf0000000[4]与 0x0，前者确定在第一拍匹配成功，后者在任何情况下均不可匹配成功，此时采用测得表5.3中数据的方法测试两种状态下的极限接收速率 (包大小为 1024B)。测试结果如表5.5所示。

由表5.5可见，当第一拍完成匹配时，极限接收速率没有受到影响，可以认为此时正则表达式匹配不降低包接收速率；当不含有匹配内容时，极限接收速率为

[4]该规则代表如果包第一个字符数值为 0 则由状态 0 跳转至状态 f，使用发包机发送的包被设定满足这一条件。

表 5.5 正则表达式匹配极限速率

Table 5.5 Regex matching limit rate of Smart NIC

arg0	极限接收速率 (Gbps)
0xf0000000	98.08
0x0	1.98

2.01Gbps。考虑到 PackageHandler 位于 QDMA 时钟域内，时钟频率为 250MHz，可以计算，若不降低速率时每拍都能完成一个数据拍的传输，则不含匹配内容时的理论极限接收速率为

$$\frac{512\times10^{-9}\ \text{Gb}}{\frac{1}{2.5\times10^{8}\ \text{Hz}}}\times\frac{1}{64}=2\ \text{Gbps}$$

实际测试结果符合理论极限接收速率的计算结果。

5.4.5.2 匹配功能测试

使用如下两个正则表达式测试匹配功能：

1. a(e|i)*o_u，其中“_”符号代表匹配任意字符 0 或多次。
2. <uid=(0-9)+>

根据正则表达式构造相应的 DFA，如图5.14所示。(未匹配时回到状态 0 和匹配成功后留在状态 f 的跳转没有显示)

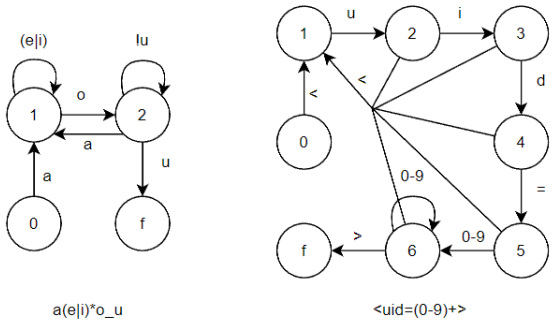


图 5.14 正则表达式匹配测试 DFA 构造图

Figure 5.14 DFA construction in regex matching test

将图5.14中的所有跳转转化为 DFA 规则参数并输入。对两个正则表达式，分别使用测试字段搜索时使用的方法，构造由 5 个包构成的发送包循环列表 (包大小为 1024B，TCP 数据的其余部分用 0 填充)，如表5.6所示。

表 5.6 正则表达式匹配测试包循环列表

Table 5.6 Circular list of packets in regex matching test

a(eli)*o_u			
包编号	插入字段	插入位置 (B)	预期匹配结果
1	“aaeiorsu”	100	√
2	“aaou”	500	√
3	“aooo”	500	×
4	“aeeiioou”	1000	√
5	不插入	——	×
<uid=(0-9)+>			
包编号	插入字段	插入位置 (B)	预期匹配结果
1	“<uid=1234567890>”	100	√
2	“<uid=0>”	500	√
3	“<uid=>”	500	×
4	“<uid=abcdef>”	1000	×
5	不插入	——	×

对每个正则表达式，使用发包机在 1Gbps 速率下按照表5.6的循环队列发送 100 万个包，测试结果分别如图5.15和图5.16所示。可见，两个正则表达式的测试结果均符合预期 (对第一个正则表达式，0 号队列与 1 号队列的收包数目比为 2:3；对第二个正则表达式，收包数目比为 3:2)，正则表达式匹配功能运行正常。

```
Packets received (queue 0):    4000000
Packets received (queue 1):    6000000
```

图 5.15 正则表达式匹配功能测试结果 (1)

Figure 5.15 Test result of regex match (1)

```
Packets received (queue 0):    6000000
Packets received (queue 1):    4000000
```

图 5.16 正则表达式匹配功能测试结果 (2)

Figure 5.16 Test result of regex match (2)

5.4.6 AES 加解密测试

对 AES 加解密功能的测试分为两个部分，即极限加解密速率测试与加解密功能测试。

5.4.6.1 加解密极限速率测试

与正则表达式匹配不同的是，除需要初始化 AES 密钥的数据拍花费额外的 10 拍外，对于每个数据拍，均需要 40 拍进行加解密步骤；对于包头拍，由于不需要加解密，故直接通过，不降低收发速率。在开启 AES 加解密功能时，采用测得表5.3中数据的方法测试极限收发速率 (包大小为 1024B)。测试结果如表5.7所示。

表 5.7 AES 加解密极限速率

Table 5.7 AES encryption / decryption limit rate

极限发送速率 (Gbps)	极限接收速率 (Gbps)
3.27	3.32

可以计算，若不降低速率时每拍都能完成一个数据拍的传输，则理论极限收发速率趋近于

$$\frac{512\times10^{-9}\ \text{Gb}}{\frac{1}{2.5\times10^{8}\ \text{Hz}}}\times\frac{16}{1+40\times15}\approx3.41\ \text{Gbps}$$

实际测试结果符合理论极限收发速率的计算结果。

5.4.6.2 加解密功能测试

可以按照如下步骤测试 AES 加解密功能的正确性：

1. 准备实现 AES-128-ecb 加解密算法的软件。(由于加解密步骤较复杂，此处不给出示例代码)

2. 使用 SimpleTxRx 软件发送 TCP 数据使用随机数填充的 TCP / IP 包，大小为 1024B，在软件的发送端 dump 包内容 (包 1) 。

3. 在发包机上接收包并抓取内容 (包 2) ，对包 2 使用同密钥进行软件解密，与包 1 进行对比。(也可将软件加密后的包 1 与包 2 进行对比，效果一致)

4. 发包机将包 2 发回，在 SimpleTxRx 软件的接收端 dump 包内容 (包 3) 。

5. 将包 1 与包 3 进行对比。

使用随机的 AES 密钥，按照步骤进行测试后，使用 VSCode 进行包 1 与解密后的包 2，以及包 1 与包 3 的内容对比，结果分别如图5.17与图5.18所示。对于包 1 与解密后的包 2，两个包的内容没有区别；对于包 1 与包 3，包内只有 TCP / IP 校验和字段存在区别，这是因为测试时硬件开启 TCP / IP 校验和生成与检查功能。综上所述，实验结果符合预期，AES 加解密功能正确运行。

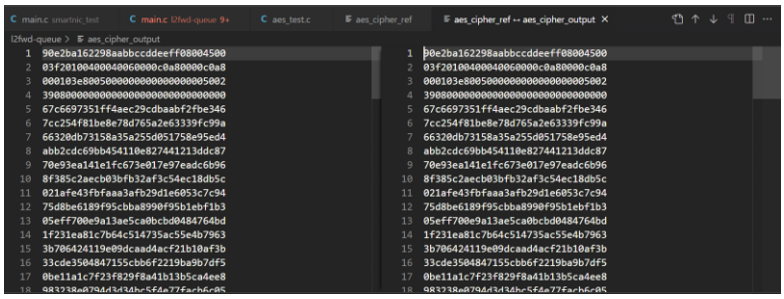


图 5.17 AES 加解密测试对比结果 (包 1 与解密后的包 2)

Figure 5.17 Comparison results of AES test (packet 1 and decrypted packet 2)

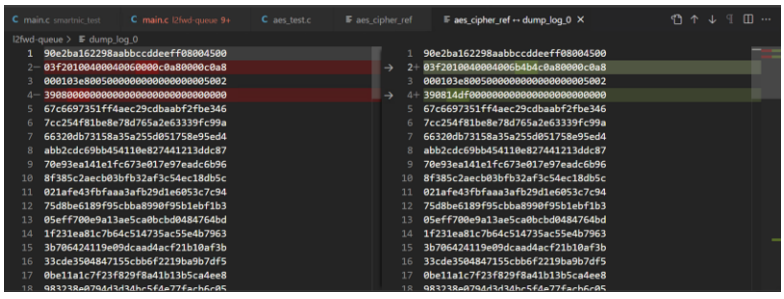


图 5.18 AES 加解密测试对比结果 (包 1 与包 3)

Figure 5.18 Comparison results of AES test (packet 1 and packet 3)

5.4.7　网卡连通性测试

使用基于 DPDK 框架的 Ping 服务端对网卡连通性进行测试。此时，智能网卡的对端不再是发包机，而是普通外部服务器上的通用网卡。断开发包机与交换机的连接，并将外部服务器连接交换机。配置外部服务器的 IP 地址 (19.19.1.137) 与智能网卡的 IP 地址 (19.19.1.170)，在智能网卡端运行 Ping 服务端，并在外部服务器上对 19.19.1.170 地址执行 ping 命令，结果如图5.19所示。

使用 tcpdump 命令抓取外部服务器的收发包记录，部分结果如图5.20所示。外部服务器发出的每个 ping 请求都接到智能网卡端的应答，且序列号正确。对包长度非 64 字节对齐 (98 字节) 的 ICMP 请求包，可以正确返回相同长度且数据

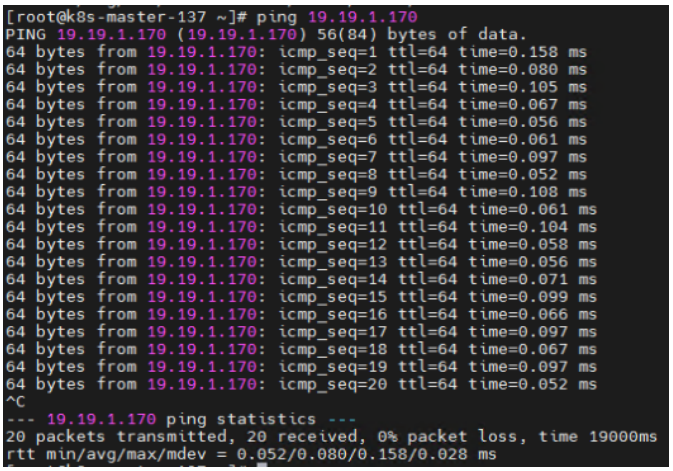
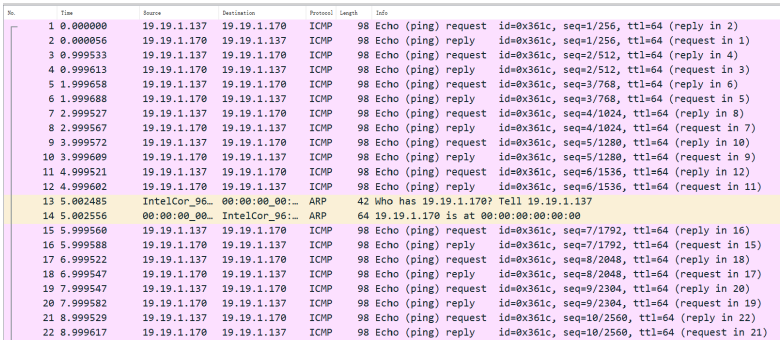


图 5.19 网卡连通性测试软件结果

Figure 5.19 Software result of NIC connectivity test

正确的 ICMP 应答包；对包长度小于 64 字节 (42 字节) 的 ARP 请求包，可以返回长度为 64 字节的 ARP 应答包。可以认为通用网卡与智能网卡能够正常连通。



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=1/256, ttl=64 (reply in 2)
2	0.000056	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=1/256, ttl=64 (request in 1)
3	0.999533	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=2/512, ttl=64 (reply in 4)
4	0.999613	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=2/512, ttl=64 (request in 3)
5	1.999658	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=3/768, ttl=64 (reply in 6)
6	1.999688	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=3/768, ttl=64 (request in 5)
7	2.999527	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=4/1024, ttl=64 (reply in 8)
8	2.999567	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=4/1024, ttl=64 (request in 7)
9	3.999572	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=5/1280, ttl=64 (reply in 10)
10	3.999609	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=5/1280, ttl=64 (request in 9)
11	4.999521	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=6/1536, ttl=64 (reply in 12)
12	4.999602	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=6/1536, ttl=64 (request in 11)
13	5.002485	IntelCor_96…	00:00:00_00:…	ARP	42	Who has 19.19.1.170? Tell 19.19.1.137
14	5.002556	00:00:00_00…	IntelCor_96:…	ARP	64	19.19.1.170 is at 00:00:00:00:00:00
15	5.999560	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=7/1792, ttl=64 (reply in 16)
16	5.999588	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=7/1792, ttl=64 (request in 15)
17	6.999522	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=8/2048, ttl=64 (reply in 18)
18	6.999547	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=8/2048, ttl=64 (request in 17)
19	7.999547	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=9/2304, ttl=64 (reply in 20)
20	7.999582	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=9/2304, ttl=64 (request in 19)
21	8.999529	19.19.1.137	19.19.1.170	ICMP	98	Echo (ping) request id=0x361c, seq=10/2560, ttl=64 (reply in 22)
22	8.999617	19.19.1.170	19.19.1.137	ICMP	98	Echo (ping) reply id=0x361c, seq=10/2560, ttl=64 (request in 21)

图 5.20 网卡连通性测试抓包结果

Figure 5.20 Packet capture result of NIC connectivity test

通过对智能网卡硬件设计进行上板实现与测试，得到能够在实际硬件环境中运行的智能网卡，实现其高性能和高扩展性的特性，并完成多个扩展功能的正确性验证。可以认为本文的智能网卡设计达到预期目标。

第 6 章　总结与展望

在第3章中，对本文智能网卡的基础框架进行设计搭建，通过组合 2 个关键 IP 核与 cmac_fifo、QDMA_ h2c(c2h)_ stub 等功能组件，得到智能网卡的高性能基础框架。在第4章中，对本文智能网卡的主功能模块，即 PackageHandler 模块框架与子组件进行设计，并使用基于 Chisel 的敏捷硬件开发方法进行代码编写与实现。随后，在 PackageHandler 模块的包处理流水线结构中对 TCP / IP 校验和处理、RSS 负载分流、字段匹配、正则表达式匹配与 AES 加解密，共 5 种示例扩展功能进行探索性实现。将 PackageHandler 模块嵌入基础框架中，得到同时具有高性能与高扩展性的智能网卡。在第5章中，使用发包机与 SimpleTxRx 软件等工具对本文智能网卡的各项功能与性能进行了简单测试，完成功能正确性与极限性能的验证，并通过使用实际应用在本文智能网卡与通用网卡间进行通信，验证了本文智能网卡的通用性。至此，本文智能网卡的开发与设计实现工作已经完成。

关于本文智能网卡的改进工作，可以在如下方面进行。

6.1　时钟域切分

在第3章中提到，本文智能网卡共使用两个时钟域，即 250MHz 的 QDMA 时钟域与约 322.25MHz 的 CMAC 时钟域，而 PackageHandler 模块在 QDMA 时钟域下运行。考虑到 QDMA 时钟域时钟频率高，满足其时序要求较为困难，如果在 PackageHandler 模块内设计复杂的功能，即使 FPGA 板上仍有大量空闲资源，也会由于布局布线拥塞与时序违例而难以实现。为了充分利用板上资源，支持复杂功能实现，一个可行的办法是为 PackageHandler 模块切分单独的时钟域，将该时钟域频率降低，并通过增加可进行时钟域转换的 IP 核，在 PackageHandler 模块的入口与出口进行时钟域转换。此时 PackageHandler 模块的时序要求更为宽松，便于扩展功能的进一步设计。

6.2 多发射并行处理

在完成时钟域切分后，PackageHandler 模块可能由于运行频率过低而成为整体传输速率的瓶颈。考虑到数据包是相对独立的个体，可以设计类似 CPU 中指令多发射机制的包处理多发射机制。在 PackageHandler 模块内设计重排序逻辑并例化多组处理部件，同时并行处理多个数据包，即可实现 PackageHandler 模块降频后包处理速率不降低。

6.3 阻塞功能优化

在完成时钟域切分与多发射设计的基础上，可以优化本文智能网卡设计中的两个需要状态机处理的功能 (即正则表达式匹配与 AES 加解密)，以进一步提升包处理速率。对于正则表达式匹配组件，一方面可以在时序宽松的基础上一拍完成多个字节的状态跳转，从而减少匹配所需的拍数，另一方面可以利用多发射机制同时对多个包进行正则表达式匹配，提升处理速度；对于 AES 加解密组件，除了可以进行上述优化外，还可以在多个包并行处理时将加解密步骤流水化，进一步提升组件利用率。

6.4 更多扩展功能的开发

可以根据用户的具体需求，进一步改善现有的扩展功能与开发新的扩展功能，如对协议栈处理的硬件卸载等；同时，对各扩展功能进行更为详细的测试。此外，也可以在 DPDK 框架内对智能网卡的驱动接口进行封装优化，以适配更多上层应用，提升智能网卡的通用性。

参考文献

[1] 马潇潇, 杨帆, 王展, 等. 智能网卡综述 [J]. 计算机研究与发展, 2022.

[2] Chiou D. The microsoft catapult project [C]//2017 IEEE International Symposium on Workload Characterization (IISWC). IEEE Computer Society, 2017: 124-124.

[3] Microsoft. Introduction to Receive-Side Scaling [EB/OL]. 2022. https://learn.microsoft.com/en-us/windows-hardware/drivers/network/introduction-to-receive-side-scaling.

[4] Woo S, Park K. Scalable TCP session monitoring with symmetric receive-side scaling [J]. KAIST, Daejeon, Korea, Tech. Rep, 2012: 163-169.

[5] Krawczyk H. New hash functions for message authentication [C]//Advances in Cryptology—EUROCRYPT'95: International Conference on the Theory and Application of Cryptographic Techniques Saint-Malo, France, May 21–25, 1995 Proceedings 14. Springer, 1995: 301-310.

[6] Barbette T, Katsikas G P, Maguire Jr G Q, et al. RSS++ load and state-aware receive side scaling [C]//Proceedings of the 15th international conference on emerging networking experiments and technologies. 2019: 318-333.

[7] Huang X, Guo Z, Song M. FGLB: A fine-grained hardware intra-server load balancer based on 100G FPGA SmartNIC [J]. International Journal of Network Management, 2022, 32(6): e2211.

[8] Li J, Lu Y, Wang Q, et al. AlNiCo: SmartNIC-accelerated Contention-aware Request Scheduling for Transaction Processing [C]//2022 USENIX Annual Technical Conference (USENIX ATC 22). 2022: 951-966.

[9] Zhan J, Jiang W, Li Y, et al. NIC-QF: A design of FPGA based Network Interface Card with Query Filter for big data systems [J]. Future Generation Computer Systems, 2022, 136: 153-169.

[10] Fiessler A, Hager S, Scheuermann B, et al. HyPaFilter: A versatile hybrid FPGA packet filter [C]//Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems. 2016: 25-36.

[11] Fiessler A, Lorenz C, Hager S, et al. Hypafilter+: Enhanced hybrid packet filtering using hardware assisted classification and header space analysis [J]. IEEE/ACM Transactions on Networking, 2017, 25(6): 3655-3669.

[12] Hayashi A, Tokusashi Y, Matsutani H. A line rate outlier filtering FPGA NIC using 10GbE Interface [J]. ACM SIGARCH Computer Architecture News, 2016, 43(4): 22-27.

[13] Xilinx. AMD OpenNIC Project [EB/OL]. 2022. https://github.com/Xilinx/open-nic.

[14] Wang Z, Huang H, Zhang J, et al. FpgaNIC: An FPGA-based Versatile 100Gb SmartNIC for GPUs [C]//2022 USENIX Annual Technical Conference (USENIX ATC 22). 2022: 967-986.

[15] Xilinx. Vivado Design Suite: AXI Reference Guide (UG1037) [EB/OL]. 2017. https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide.

[16] Daemen J, Rijmen V. The design of rijndael: volume 2 [M]. Springer, 2002.

[17] Xilinx. QDMA Subsystem for PCI Express Product Guide (PG302) [EB/OL]. 2022. https://docs.xilinx.com/r/en-US/pg302-qdma.

[18] Xilinx. UltraScale+ Devices Integrated 100G Ethernet Subsystem Product Guide (PG203) [EB/OL]. 2022. https://docs.xilinx.com/r/en-US/pg203-cmac-usplus.

[19] freechipsproject. Chisel Project Template [EB/OL]. 2022. https://github.com/freechipsproject/chisel-template.

[20] chipsalliance. Chisel 3: A Modern Hardware Design Language [EB/OL]. 2022. https://github.com/chipsalliance/chisel3.

[21] Intel. Home - DPDK [EB/OL]. 2022. https://www.dpdk.org/.

致 谢

本文工作由于需要从零开始搭建智能网卡，对于一个科研经验不足的本科生而言具有较大的难度。其能够得以完成，首先，需要感谢陈明宇老师与张文力老师对我的耐心指导，为整体开发方向与具体工作方式提供了重要的指引，让我对实际的科研流程与规范化工程开发有了更为详细的理解，也提供了开发所需的硬件环境。其次，需要感谢张旭、张钊等各位师兄提供的技术支持，在搭建软硬件环境、理解硬件工程搭建方式和 IP 核约束、进行 debug 与论文撰写等多个方面均提供了具体详实的建议与教学，在毕设过程中，我学习到了 Chisel、FPGA 开发与 DPDK 开发等以前不甚熟悉的内容，在知识积累与编程能力上收获颇丰。然后，感谢国科大各位老师四年来的培养教育，带领我走上计算机体系结构的学习研究之路。最后，感谢刘梓穆、葛宇涛等各位同学朋友与我的父母在开发过程中提供的精神支持与帮助 (有时也有技术援助)，正是有了他们的一路陪伴，我才能一直坚持到现在。“长风破浪会有时，直挂云帆济沧海”，探索未知固艰苦，然亦有无穷乐趣。希望今后能够在全面发展的同时，在计算机体系结构方向上进一步学习深造，提升自我，取得更加辉煌的成就。