密级：_____

中国科学院大学
University of Chinese Academy of Sciences

博士学位论文

面向通用通信基带处理器的专用协处理器研究与设计

作者姓名：赵旭莹

指导教师：王东琳　研究员

中国科学院自动化研究所

学位类别：工学博士

学科专业：计算机应用技术

培养单位：中国科学院自动化研究所

2017 年 10 月

密级：_____

中国科学院大学
University of Chinese Academy of Sciences

博士学位论文

面向通用通信基带处理器的专用协处理器研究与设计

作者姓名：赵旭莹

指导教师：王东琳　研究员

中国科学院自动化研究所

学位类别：工学博士

学科专业：计算机应用技术

培养单位：中国科学院自动化研究所

2017 年 10 月

Research and Design on the Specific CoProcessor of Universal Communication Baseband Processor

By

Xuying Zhao

A Dissertation Submitted to

University of Chinese Academy of Sciences

In partial fulfillment of the requirement

For the degree of

Doctor of Engineering

Institute of Automation, Chinese Academy of Sciences

October, 2017

独创性声明

本人声明所提交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确地说明并表示了谢意。

签名：赵旭东 日 期：2017.12.4

关于论文使用授权的说明

本人完全了解中国科学院自动化研究所有关保留、使用学位论文的规定，即：中国科学院自动化研究所有权保留送交论文的复印件，允许论文被查阅和借阅；可以公布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论文。

（保密的论文在解密后应遵守此规定）

签名：赵旭东 导师签名：王琳东印 日 期：2017.12.4

摘要

主流通信基带处理器大多采用协处理器对某些实时性要求高，但不适于矢量处理的复杂算法进行加速。随着通信技术发展和日益增长的数据速率需求，协处理器中加速引擎种类和数目不断增加。加速引擎互联关系和协处理器调度模式直接影响处理器整体性能，成为一个研究热点。研究和设计具有自主知识产权的高性能、低功耗和高可靠性芯片是一个巨大挑战。

论文对无线通信系统中不适于矢量处理的信道译码算法进行研究，首先在通用通信基带处理器现有体系结构基础上，提出了一种新型二维可配置协处理器架构；然后对协处理器中 turbo 译码器、viterbi 译码器和 polar 译码器进行了设计及优化，包括：提出了一种基于二阶差分辅助的 CRC 校验停止准则，改善 turbo 译码器在信号质量差或突发错误下无用功迭代、设计了一种支持多标准的高性能可配置 viterbi 译码器，以及提出了一种路径扩展优化方法和新型路径删减策略，有效降低了 polar 译码延迟。具体包括以下工作和创新点：

1. 提出了一种新型二维可配置协处理器架构，极大降低了互连网络功耗和总线带宽占用比。针对主流协处理器架构存在互连网络功耗大、协处理器调度频繁等问题，提出了一种面向通信处理器的新型二维可配置协处理器架构。通过将加速引擎分簇，并以特定工作模式重新编程加速引擎内部联结关系，使协处理器在灵活度和可靠性方面达到平衡。第一维配置为工作模式和协处理器公共参数配置，由主处理器发起，协处理器实时响应；第二维配置为加速引擎私有参数配置，由主处理器离线完成。通过功耗评估模型，总线互连网络功耗仅为主流通信处理器架构的 1/3；对于无线通信标准数据帧处理，总线带宽占用比由 6.88%降到 2.05%。新型协处理器架构的提出为通信处理器低功耗、低复杂度设计提供了有益探索。
2. 提出了一种基于二阶差分的 CRC 校验停止准则，有效降低了 turbo 译码器在信号质量差或突发错误下无用功迭代次数。针对在传输环境较差或发生突发错误时，turbo 译码器迭代多次而译码结果不理想的问题，提出了一种基于二阶差分的 CRC 校验提前退出迭代方法。该方法通过对传递信息进行二阶差分值计算，可以提前感知信道情况并及时退出迭代。仿真实验表明：与

常规 CRC 校验停止准则相比，该方法在信道恶劣情况下，turbo 译码器平均迭代次数下降约 20%。

3. 针对目前多标准 viterbi 译码器吞吐不高的问题，设计了一种支持多标准的高性能可配置 viterbi 译码器，适用于不同场合的卷积码译码。该译码器支持编码约束长度为 5~9，码率为 1/2,1/3,1/4，支持零结尾和咬尾。译码器峰值吞吐为 1.15Gbps@6144bit,600MHz。主流商用 viterbi 译码器 VCP2，数据处理能力为 9.5Mbps@40bit,333MHz，本文中译码器数据处理能力为 32.173Mbps@40bit,333MHz，性能提升约 3.3 倍，可满足日益增长的数据量处理需求。
4. 提出了一种路径扩展优化方法和新型路径删减策略，有效降低了 polar 译码延迟。针对连续消除列表（Successive Cancellation List，SCL）算法译码延迟比较大的问题，提出了一种路径扩展优化方法，避免冗余路径分裂，有效降低了译码延迟，同时理论证明该优化方法在译码性能方面没有任何损失。此外，提出了一种基于置信区间的新型删减策略，降低了 SCL 译码复杂度。仿真表明路径扩展优化方法可以有效降低路径分裂数目，最高可达 49%；在性能损失可忽略情况下，新型删减策略可以降低搜索路径数目，在中 SNR 区间可以降低 60%，高 SNR 区间可以降低 80%。

关键词：协处理器架构；硬件加速引擎；停止迭代；多标准 viterbi 译码器；Polar 译码器

Abstract

Most of the mainstream communication baseband processors adopt coprocessors to accelerate complex algorithms which are not suitable for the vector processing but with high requirement of real-time. With the development of communication technology and the increasing demand of data rate, the types and numbers for acceleration engines in the coprocessors continuously increase. The interconnection of acceleration engines and the coprocessor scheduling technology directly affect the overall performance of the processor, becoming a hot research topic. The research and design of high performance, low power consumption, and high reliability chips with independent intellectual property rights is a huge challenge.

In this paper, we focus on the channel decoding algorithms which have the high computational complexity in wireless communication system. Firstly, based on the present architecture of the general purpose communication processor, a novel two level reconfigurable CSCP architecture is proposed; then, the acceleration engines in the CSCP such as turbo decoder, viterbi decoder, and polar decoder are designed and optimized, including: proposing a second order difference aided CRC check stopping criterion to reduce the iteration numbers when the transmission environment is bad, designing a reconfigurable high performance viterbi decoder which supports multiple standards, and proposing an optimized path expansion method and a novel tree-pruning scheme to reduce the polar decoding latency. The main work and contributions are as follows.

We propose a novel two level reconfigurable CSCP architecture, reducing the power consumption of the bus interconnection network and the bandwidth utilization ratio. Based on the shortcomings of the current commercial communication digital signal processor architecture, a novel two level reconfigurable coprocessor architecture for communication processors is proposed, and of which the internal connections of the acceleration engines are reprogrammed in a specific work mode by clustering. The first level configuration includes coprocessor work mode and coprocessor common parameters. The coprocessor is initiated by the main processor and responds in real time.

The second level configuration includes the private parameters for each acceleration engine and it is accomplished by the main processor offline. The power consumption of bus interconnection network is equivalent to a third of the typical communication processor architecture under a power evaluation model. The bus bandwidth utilization ratio falls to 2.05% from 6.88% for a standard data frame processing by clustering the acceleration engines. The novel coprocessor architecture has provided a useful exploration for low power and low complexity design of communication processors.

We propose a second order difference aided CRC check stopping criterion to improve the iteration numbers of turbo decoders. When the transmission environment is bad, the receiving data cannot be decoded correctly even the decoder keeps working all the time. To solve this problem, second order difference aided CRC check stopping criterion is proposed. Based on the second order difference of soft information and/or hard-bit information, such bad scenarios are detected before CRC check stopping conditions are satisfied and the decoders stop iteration. Simulation results show that compared with the conventional CRC stopping criterion, the average iterative number of turbo decoder is reduced by about 20% in the case of poor channel conditions.

We design a high performance multi-standard viterbi decoder. The throughput for the present multi-standard viterbi decoder is relatively low, we propose a high performance decoder which supports polynomial reconfiguration, constraint length of 5~9, and code rate of 1/2, 1/3, 1/4. Moreover, both tail-biting and zero trellis terminating modes are supported. Simulation results show that the maximum data throughput is 1.15Gbps under a clock frequency of 600MHz. The data processing capacity of commercial viterbi decoder VCP2 is 9.5Mbps@40bit, 333MHz, and the proposed decoder is 32.173Mbps@40bit,333MHz. The data throughput is improved about 3.3 times and it can meet the increasing demand of data processing.

To reduce the decoding latency of polar decoder under successive cancellation list (SCL) decoding, an optimized path expansion method is proposed and it is proved that the optimization method has no degradation in decoding performance. In addition, a novel tree-pruning scheme is proposed based on the confidence interval. Studies show that the optimized path expansion method has a number of split paths which are up to 49% lower than the conventional SCL algorithm for each bit estimation and the average

searching path number with pruned-SCL falls almost 60% at moderate SNR region and 80% at high SNR region.

Keywords: Coprocessor Architecture; Hardware Acceleration Engine; Stop Iteration; Multi-Standard Viterbi Decoder; Polar Decoder

目录

图目录

表目录

第1章 绪论

1.1 研究背景和意义

近年来，我国在 CPU、操作系统等核心技术方面有了很大发展，但尚未构成自己的技术体系和生态系统，尤其在芯片制造方面存在短板。我国民用通信设备所需要的基带处理芯片仍基本依赖国外芯片厂商提供。无线通信系统基带处理所需的高性能数字信号处理器的关键技术主要掌握在国外公司手中，极大的制约了我国通信产业的发展，尤其美国对中兴禁运芯片事件，也使我们更进一步看到了发展国产芯片的严峻性和紧迫性。

2014 年，国务院印发《国家集成电路产业发展推进纲要》，明确了集成电路产业发展目标：力争 2020 年，移动智能终端、网络通信、云计算、物联网等重点领域，集成电路设计技术达到国际领先水平，到 2030 年，集成电路产业链主要环节达到国际先进水平[1]。

中科院自动化所国家专用集成电路中心在先导专项“代数处理器芯片（课题编号：XDA06011000）”的支持下开始研制一款具有自主知识产权体系结构和高性能运算能力的代数处理器芯片，并在 2016 年流片。其高效的能耗比达到国际领先水平。通用通信处理器 UCP（Universal Communication Processor）旨在将此具有创新体系结构和高效能耗比的处理器应用于无线通信领域。通过在民用通信领域推广，将形成技术先进、性能可靠的数字基带信号处理器系列化产品和应用技术方案，对推动无线通信行业快速发展和支援国民经济建设具有重要意义。

1.1.1 通用通信处理器面临的问题

无线通信技术的发展，给数字基带处理带来了很大挑战。目前基带处理器主要是以 GPP（General Purpose Processor，通用处理器）、FPGA（Field-Programmable Gate Array，现场可编程门阵列）和 DSP（Digital Signal Processor，数字信号处理器）为主的实现方案。虽然目前基于 Intel CPU 的网络功能虚拟化（Network Function Virtualization，NFV）[3]研究比较火热，但对于物理层高实时、高密集型数据处理，通用处理器表现得仍然很吃力；FPGA 属于半定制芯片，工程师在给定资源下做硬件开发，就单个芯片的成本而言，FPGA 是不占优势的。此外，功耗问题也是制约

FPGA 产业快速发展的一个因素[4,5]；可编程 DSP 以其高性能、低功耗特性在实时通信系统中的地位举足轻重，全球 DSP 大厂，如德州仪器（Texas Instruments，TI）、CEVA、飞思卡尔等，纷纷推出了一系列基于 DSP 的无线通信解决方案[6,7]。

无线通信系统的数据处理既有高密集数据运算的特点，如信道估计[8]，MIMO 检测[9]等；又有小位宽数据交织的特性，如速率匹配、信道交织等；还有一些特定复杂算法的加入，如 turbo 译码[10]、viterbi 译码[11]等。传统通用通信处理器一方面要满足高吞吐量的需求，势必会增加运算部件复杂度，带来芯片面积过大、功耗过高等问题；另一方面通用通信处理器对某些复杂算法适应度不够，使得复杂算法的运算成为制约处理器性能的瓶颈。

1.1.2 UCP 芯片介绍

通用通信处理器 UCP 是在代数处理器[2]按照“同心圆”层次进行体系结构优化的方法学及自主创新的 AppAISArc™ 指令体系结构基础上针对无线通信领域衍生的处理器系列。AppAISArc™ 同心圆模型如图 1-1 所示。

该处理器在硬件、算法和编程模型三个层面对应用进行优化，采用一种新型的多粒度并行存储系统和 SIMD（Single Instruction Multiple Data，SIMD）指令集。通过基于状态机的编程模型，将应用软件与硬件之间建立联系，并在应用软件性能和编程复杂度之间建立一个完美平衡，有很强的通用适应性。

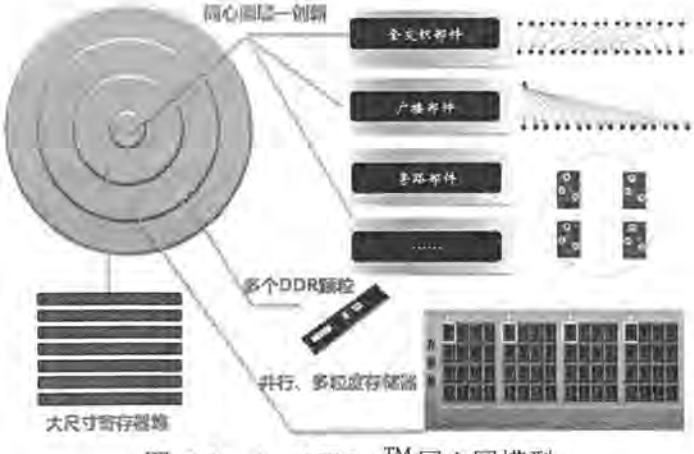

图 1-1：AppAISArc™ 同心圆模型

UCP 处理器采用双流水线结构，UCP 核端代码分为两层：标量层代码和代数指令层代码，其编程模型与体系结构之间的关系如图 1-2 所示。代数指令流水线的执行是通过标量层代码进行调用实现的，标量层程序负责配置代数流水线参数，控制代数流水线的启停等工作。此外，在无线通信领域，存在一小部分算法，无法充分发挥代数指令的高效性。因此，标量层程序也用于实现数据量小但控制流复杂的算法。代数指令层是基于同心圆优化思想的全新指令集体系结构，是一种自适应、高效的代数指令，可动态重构流水线结构。支持该指令系统的处理器在流片生产后，程序员仍可根据无线通信算法特点对处理器指令集进行重定义。重定义后的处理器指令集体系结构更加契合无线通信算法特征，从而能提高处理器在该类应用中的处理性能。重定义过程不受处理器硬件制约，也不影响汇编器、编译器等开发工具的使用。

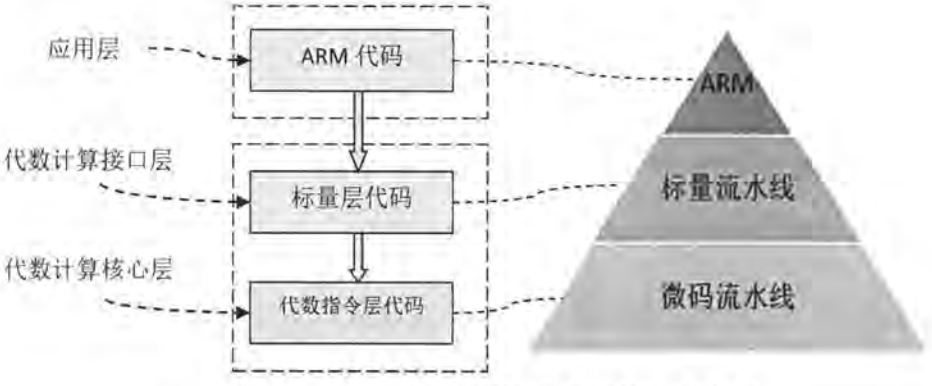

图 1-2：UCP 编程模型与体系结构抽象模型

UCP 代数处理引擎（又称为代数流水线）如图 1-3 所示。在代数处理引擎中，包含大量基本运算单元，主要有数据加载/存储单元、数据处理单元和旁路网络。这些基本单元并行运行，其运行时的具体动作由微码指令决定。微码指令统一存放在微码存储器中，同一时刻发射的微码存放在存储器中的同一行。微码程序员通过微码序列，规定数据加载单元、数据计算单元、数据旁路网络在每个时刻的动作，组织功能单元的级联，形成代数指令的流水线。

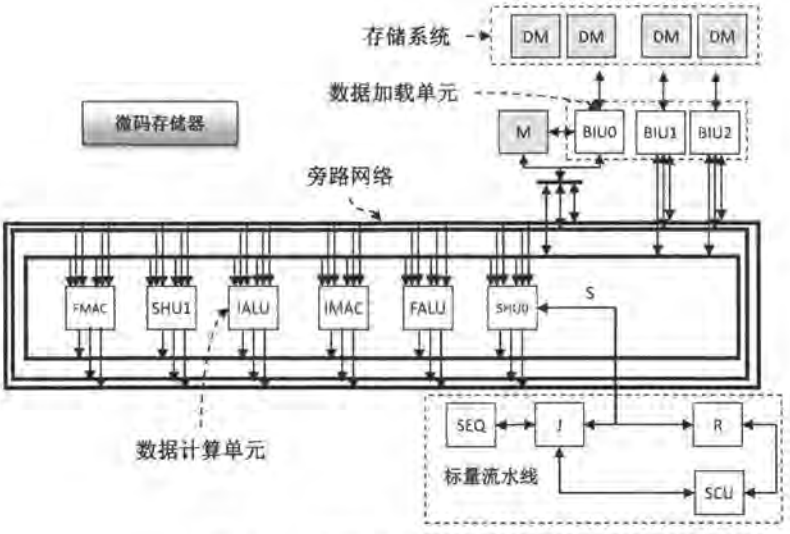

图 1-3：代数流水线结构示意图

无线通信系统的物理层处理分为符号级和比特级两部分[12,13]，符号级处理的特点是数据量大、数据处理比较规整，判断和跳转语句不多，非常适合高并行度的矢量处理器来实现；比特级处理，尤其是接收端软比特处理，运算比较复杂，运算量比较大，这部分数据处理通常采用协处理器来实现。

结合无线通信算法及通用处理器现有体系结构，本文面向通用通信基带处理器进行协处理器研究和设计，对信道译码等不适于矢量处理器实现的复杂算法进行硬件加速。该研究对自研芯片应用推广和芯片国产化意义深远。

1.2 国内外研究现状

针对通用通信处理器面临的问题，本文研究了通信专用协处理器架构以及协处理器中运算复杂度比较高的信道译码加速引擎：turbo 译码器、viterbi 译码器和 polar 译码器。下面分别介绍其国内外研究现状。

1.2.1 通信专用协处理器架构

协处理器是一种与主处理器协同工作、辅助其完成特定计算任务的专用处理芯片或器件。协处理器的功能通常在硬件中实现，替代纯软件算法与指令，设计时不必过多地考虑通用性，因而往往会拥有很好的特定计算性能。协处理器作为一种常规加速机制，可以采用专用硬件 ASIC 实现[14-16]，也可以采用软件可编程芯片 DSP[17-19]、硬件可编程芯片 FPGA 来实现[20,21]。在 x86 领域，最早的协处理器

是 8087[22]，它是 Intel 设计的第一个数学协处理器。后期协处理器类型不断演进，功能更加复杂，不再是单纯的代数运算逻辑，而是针对复杂算法特殊设计的加速引擎[79-81]。

目前，全球领先的半导体公司 TI、飞思卡尔和信号处理知识产权（Intellectual Property，IP）授权公司 CEVA 均推出了一系列商用通信数字信号处理器。商用通信 DSP 中嵌入了一系列硬件加速引擎作为协处理器，对复杂算法进行加速。Lasse Harju[82]针对 WCDMA/OFDM 移动终端系统提出了一种协处理器架构，三个加速引擎同主处理器一起挂接到总线，如图 1-4 所示。文献[83]提出了一种灵活的协处理器架构，控制单元以控制字形式对加速引擎进行调度，协处理器可以实现并行流水。

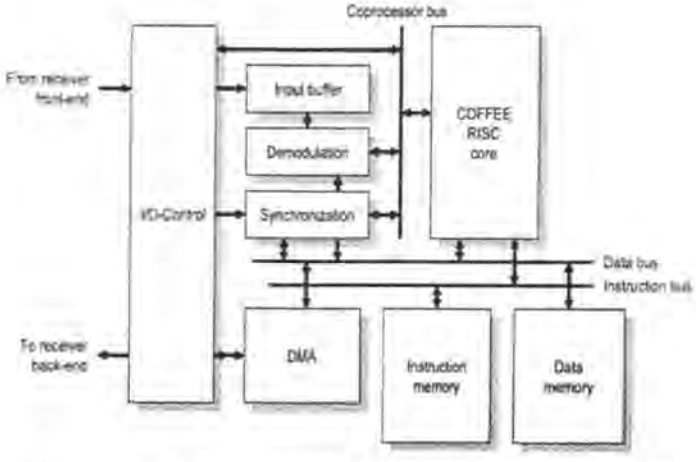

图 1-4：针对 WCDMA/OFDM 移动终端协处理器架构

随着通信技术发展和日益增长的数据速率要求，通信专用协处理器类型和数目不断增加。例如，TI 早期的 TMS320C64x 系列 DSP 中只有 turbo、viterbi 解码协处理器，后来的 TMS320C66x DSP 中又增加了 FFT、BCP 等加速引擎。下面以 TI 66x、飞思卡尔 8157 和 CEVA-XC 为例，介绍通信专用协处理器架构。

1. TI TMS320C6670

TI 的 TMS320C6670 DSP 在基站设备中应用比较广泛，支持的标准有 WCDMA/HSPA/HSPA+，TD-SCDMA，GSM，TDD-LTE，FDD-LTE 和 WiMAX，工作主频为 1.2GHz，其功能模块架构如图 1-5 所示[23]。TMS320C6670 中的协处理器包括 2 个 RAC（Receive Acceleration Coprocessors）、1 个 TAC（Transmit Acceleration Coprocessor）、3 个 FFTC（Fast Fourier Transform Coprocessor）[24]、3

个 TCP3d（Turbo Decoding Coprocessor）[25]、4 个 VCP2（Viterbi Decoder）[26]、1 个 TCP3e（Turbo Encoding Coprocessor）[27]、和 1 个 BCP（Bit Rate Coprocessor）[28]。这些协处理器与主处理器、外围设备共同挂接在 TeraNet 总线上。

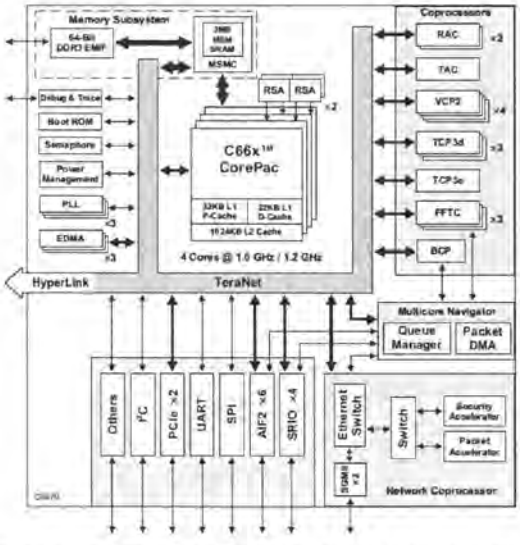

图 1-5：TI TMS320C6670 DSP 功能模块架构图

TMS320C6670 中 TeraNet 数据连接如图 1-6 所示，TeraNet 为 DSP 核、协处理器、存储器和外围设备提供高速率、并行正交通信。

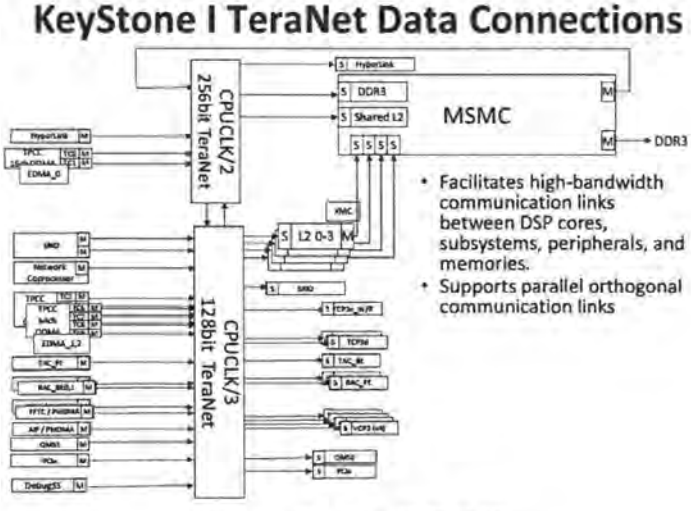

图 1-6：TeraNet 数据连接

2. 飞思卡尔 MSC8157

飞思卡尔 MSC8157 主要用于基站中，支持的标准有 3G-LTE（FDD、TDD），HSPA+， LTE-Advanced、WiMAX，工作频率为 1GHz，其功能框图如图 1-7 所示[29]。协处理器 MAPLE-B2 包括 eTVPE、eFTPEs、DEPE、EQPE、CRPE、CRCPE、CGPE、CONVPE 等加速引擎。DSP 核、MAPLE-B2、存储器和其它外围设备挂接在 CLASS（Chip Level Arbitration and Switching System，芯片级仲裁切换系统）总线。CLASS 总线为主从设备提供全互连、非阻塞通信，其中主设备有 15 组，从设备有 10 组。MAPLE-B2 有四组 master 接口，用于加载数据；一组 slave 接口，用于接收配置。

飞思卡尔协处理器 MAPLE-B 系列不断演进，目前已演进到了第三代。第一代协处理器 MAPLE-B 功能框图如图 1-8 所示。MAPLE-B 中包含一组仲裁转换总线，将 RISC（Reduced Instruction Set Computing）处理器、数据存储器和加速引擎进行桥接。第三代协处理器 MAPLE-B3 功能框图如图 1-9 所示[30]。协处理器中增加了加速引擎种类和数量，RISC 处理器和对外总线端口数也相应增加。

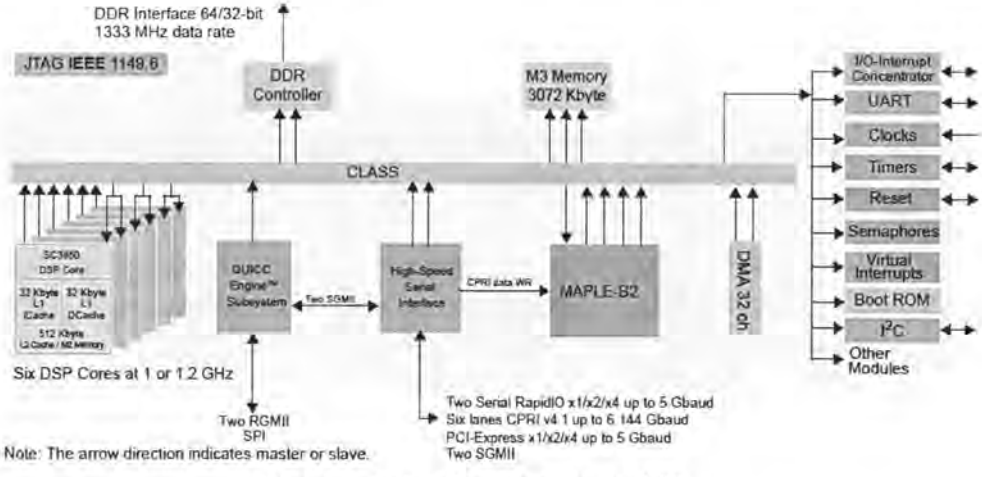

图 1-7：飞思卡尔 MSC8157 结构图

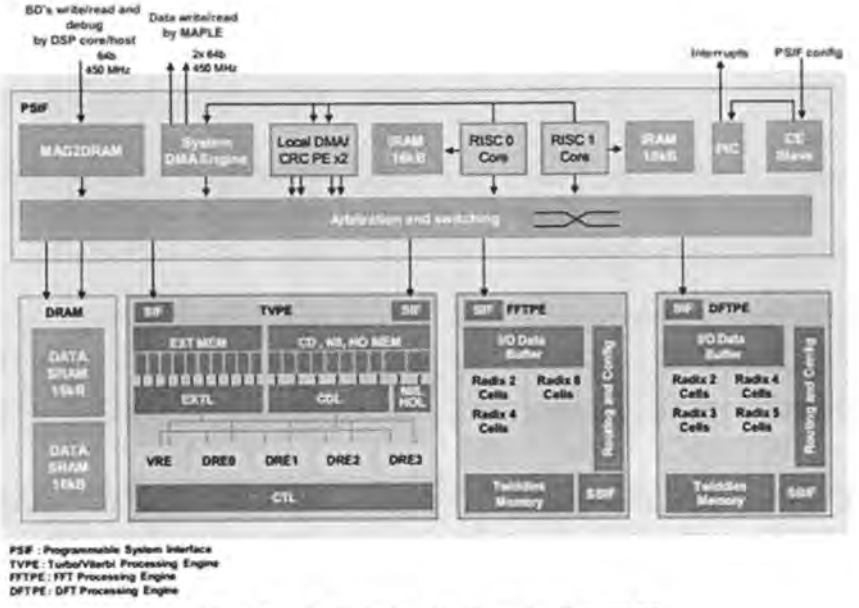

图 1-8：飞思卡尔 MAPLE-B 功能框图

图 1-9：飞思卡尔 MAPLE-B3 功能框图

3. CEVA-XC 系列

CEVA-XC4500 矢量 DSP 一般用于基站中，由 CEVA-XC4000 演进而来，支持的标准有 LTE、TD-LTE、HSPA+、LTE-A、TD-SCDMA、GSM 等，工作频率为 1.3GHz，采用 28nm 工艺。CEVA-XC4000 和 CEVA-XC4500 DSP 架构如图 1-10、图 1-11 所示[31,32]。CEVA 协处理器 TCE 中包括可编程的多维最大似然译码器（Maximum Likely Decoder，MLD）、FFT-DFT、viterbi、HARQ 合并、FHT 等加速引擎。从图中可以看出，TCE 中加速引擎数量不断增加。协处理器与外部通信通过 FIC（Fast

InterConnect）总线，FIC 可以为主从设备提供高带宽、低延迟连接，用户可以根据需要配置 FIC 设备数目和总线位宽。

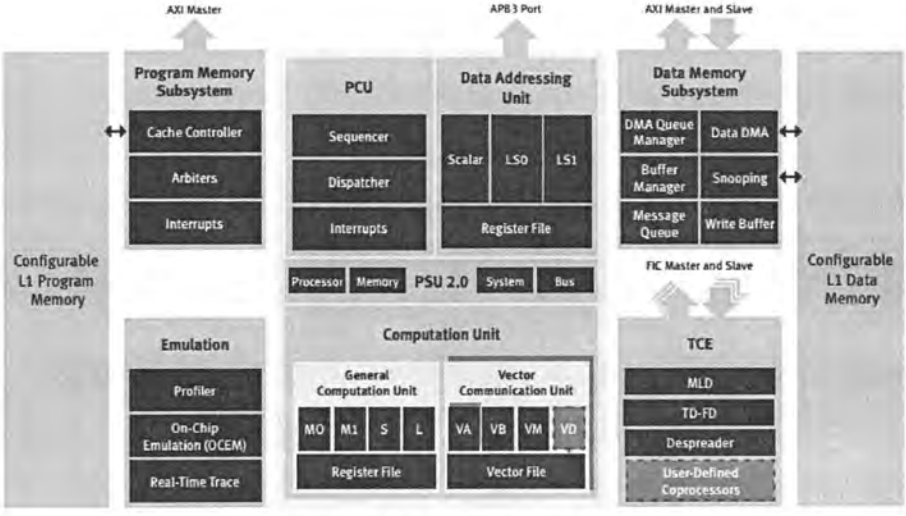

图 1-10：CEVA-XC4000 结构框图

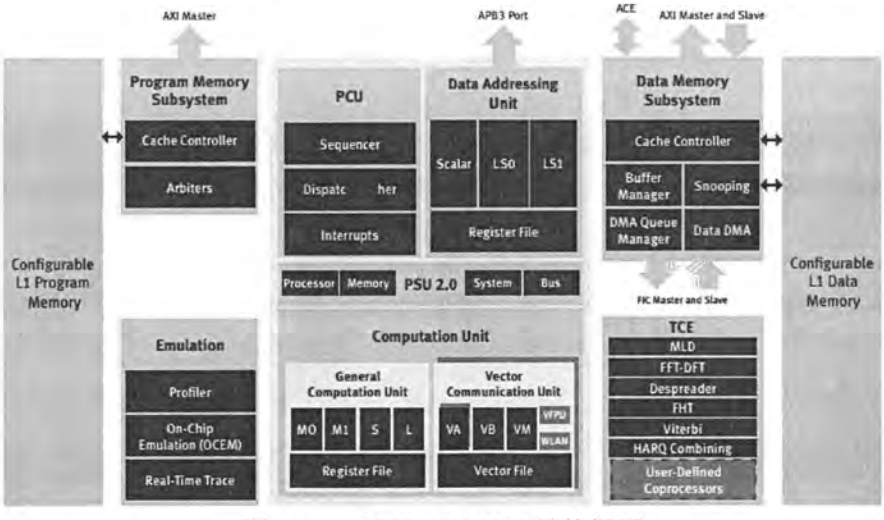

图 1-11：CEVA-XC4500 结构框图

CEVA 处理器中协处理器工作机制是基于动态任务调度，每个加速引擎对应一组任务队列和数据队列，如图 1-12 所示，主处理器与协处理器桥接采用 AXI/FIC 总线，如图 1-13 所示。

图 1-12：CEVA 协处理器工作机制

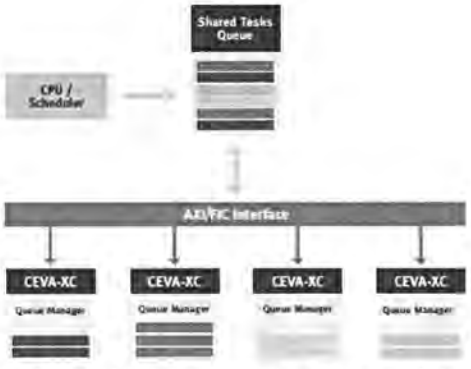

图 1-13：CEVA 主处理器与协处理器桥接关系

1.2.2 信道译码加速引擎

1. Turbo 译码器退出迭代停止准则

Turbo 码又称并行级联卷积码[33]，是将两个简单的分量编码器通过伪随机交织器并行级联来得到具有伪随机特性的长码，由 Claude Berrou 在 1993 年 IEEE 会议上提出[10]。Turbo 码由于其接近香农限的译码性能，在当时的通信领域引起了轰动，随后被广泛应用于各种通信系统。

Turbo 码表现出了接近香农限的优异性能，主要是由于采用迭代译码思想。S.ten Brank 基于软输入软输出（Soft In/Soft Output，SISO）译码器中外信息传递特性（Extrinsic Information Transfer，EXIT），分析了 turbo 迭代译码收敛行为，并利用 EXIT 图，跟踪子译码器间外信息传递轨迹，预测 turbo 译码迭代次数[34]。EXIT 图为分析 turbo 迭代译码提供了理论依据。

Turbo 译码随着迭代进行，译码性能不断趋于稳定，通常会设置一个最大迭代

次数，来避免无休止迭代，这种方法称为固定迭代次数（Fixed-Iteration-Number，FIN）停止准则。FIN 停止准则存在这样的问题：在信道环境比较好的情况下，迭代一两次即可接收正确信息，但由于没有提前退出迭代机制，译码器仍需迭代多次才会结束译码，这对通信系统实时性要求很不利。

针对 FIN 停止准则不足，很多其他提前退出迭代算法被提出。Joachim Hagenauer 提出一种交叉熵（Cross-Entropy，CE）方法[35]，该方法可以在译码损失很小的情况下降低迭代次数，但是计算复杂度比较大，占用大量存储器资源。基于 CE 方法，出现了两种简化的停止迭代准则[36]：符号变化率（Sign-Change-Ratio，SCR）和辅助硬判决（Hard-Decision-Aided，HDA）。SCR 准则是通过计算两次迭代间的外信息符号变化情况，并与一定阈值做比较，得到退出迭代条件。相似的，HDA 准则是通过计算外信息的硬判决结果来得到退出迭代条件，如果两次迭代硬判决符号相同，则退出迭代。针对 CE 算法数据存储比较大的问题，符号差分比（Sign-Difference-Ratio，SDR）[37]和改进型 HDA（Improved HDA，IHDA）[38]方法被提出，一定程度上降低了数据存储面积。随后，N. Yul Yu 在降低迭代次数和保证提前退出迭代对译码性能损失降到最小方面进行了探索。提出了两个判决阈值，即在一次迭代中，通过两次阈值比较，来判断是否提前退出迭代[39]。基于 SNR 估计的停止迭代准则由文献[40]提出，该方法使误比特率（Bit Error Rate，BER）性能提升了 0.3dB，并在一定程度上降低了译码延迟和计算时间。近年来，一些新的提前退出迭代方法被提出，例如置信度测量[41]、奇偶校验停止准则[42]、外信息方差判决准则[43]等。

与以上停止准则相比，CRC 校验准则具有较好的译码性能和较低的迭代次数[44]。CRC 校验需在原始比特后面添加部分冗余比特。当码块长度较小情况下，传输带宽会受到一定影响。但冗余比特的增加，使译码器迭代次数在 BER=10^{-6} 时降低 75%，因此 CRC 停止准则在 3G、4G 通信系统中应用比较广泛。文献[45]提出一种 CRC 辅助的 Log-MAP turbo 译码器架构，在译码性能几乎无损情况下，该译码器迭代次数由 5 降到 3（BER=10^{-7} 处）。文献[46]提出三种 turbo 译码器提前退出迭代和错误检查准则，通过对译码器输出的对数似然比（Log-Likelihood Ratio，LLR）求取平均值，得到基于平均值的停止准则。为了进一步提高译码器性能，又提出基于平均值符号变化的 MSC（Mean-Sign-Change）和 MSC-CRC*eb* 准则。Carlo Condo 等[47]针对 3GPP LTE/LTE-A 中 24 比特 CRC 序列，提出一种并行度可变的 CRC 电路结构，可使译码器适用于多种数据长度的检错和提前退出迭代。Hyeji Kim 等[48,49]提出了两种分布式 CRC 结构用于提前停止迭代。该结构可用于高并行度

turbo 译码器中，支持不同的数据长度和并行度，可以与子译码器并行工作，降低译码延迟。

2. 可配置 viterbi 译码器

Viterbi 算法于 1967 年提出，最初用于卷积译码[11]。Omura[50]从状态空间角度解释了 viterbi 算法相当于通过加权图，动态规划求解最短路径问题。Sun Ping[51]等在广义 viterbi 算法基础上，提出一种简化 viterbi 译码器架构，称为 PSS((Probability Selecting States）型架构。并实现了针对码率为 1/2，约束长度为 7 的卷积码译码，该架构比传统译码器硬件开销有所降低。Claude Berrou[52]等通过理论证明，得出了低复杂度软输出 viterbi 译码器架构。相比硬比特译码，译码性能得到了极大改善。文献[53]提出一种人工神经网络译码器。该译码器采用全并行架构，译码速度大幅提升。文献[54]提出两种方法对 viterbi 译码器进行加速，一种方法是提高译码基数，每次操作对多个时刻的状态转移进行估计；另一种方法是基于交织的思想，将输入码元交织成多个码块，分别送入独立的编码器，接收端采用多个译码器并行处理独立编码块。两种方法均可极大提升译码器数据吞吐，为后续译码器研究提供了理论基础。

Xun Liu 等[55]针对 IS95 数字蜂窝标准设计了一个（3,1,9）viterbi 译码器，在 0.25 μm 标准单元库下，该译码器数据吞吐为 20Mbps，功耗为 450mW。文献[56]基于修正的 T-algorithm 和回溯方法，设计了一个（2,1,7）viterbi 译码器，该译码器在 0.25 μm 标准单元库下，数据吞吐为 200Mbps，功耗为 185mW。文献[57]基于向量处理器（Co-Vector Processor，CVP）设计了两条针对 GSM 和 UMTS 标准的 viterbi 译码指令，该指令的数据吞吐仅为 Kbps 量级。文献[58]基于 XCV1000-04 FPGA，设计了一个支持约束长度为 4~14 的自适应 viterbi 译码器，该译码器峰值吞吐可达到 61.7Kbps。Fei Sun 等[59]在自适应 viterbi 译码器基础上进行改进，提出一种松弛自适应 viterbi 算法并进行 VLSI（Very Large Scale Integration）实现，该译码器支持 1/2 码率，约束长度为 7~9。在 0.13 μm 标准单元库下，数据吞吐为 200Mbps，与文献[60]中的寄存器交换法译码相比，功耗降低了 81%；与文献[61]中的回溯法译码相比，功耗降低了 48%。

此后，关于高性能 viterbi 译码器研究不断涌现[62]。文献[63]设计了一个支持 1/3 码率、约束长度为 7 的 viterbi 译码器，采用基四算法，在 Altera STRATIX III EP3SE80F1152C2 设备上，时钟主频可达到 68.56MHz，峰值吞吐为 274Mbps。文献[64]设计了一种基于硬判决的 viterbi 译码器并进行了 VLSI 实现，支持码率 1/2，约

束长度 7，基四算法下峰值吞吐为 714Mbps。在 90nm 标准单元库下，采用基二算法，功耗为 204.3mW。基于 FPGA 设计的 viterbi 译码器，由于主频限制，数据吞吐一般不是很高，文献[65]和文献[66]中的 viterbi 译码器，时钟主频分别为 64.516MHz 和 157.149MHz，译码器难以达到较高的数据吞吐。Narayan V Sugur 等[67]在 2014 年设计出一个针对 1/2 码率、约束长度为 3 的 viterbi 译码器，该译码器在 65nm 标准单元库下，时钟主频可达到 250MHz，数据吞吐为 1Gbps。

近年来，viterbi 译码器在多个领域得到了关注。文献[68,69]研究了 viterbi 译码器用于光通信中载波相位估计；He Wang 等[70]首次将 viterbi 算法用于超高频视频识别（Ultra-High Frequency Radio Frequency Identification，UHF RFID）接收机中，取得了很好的误码性能；Mahender Veshala 等[71]研究了 viterbi 译码器在 Wi-Fi 接收机中的应用；Hiromasa Kato 等[72]研究了其在物联网（Internet of Things，IoT）传感设备中的应用，研究重点偏向于小型化、低功耗。

以上研究中，viterbi 译码器多是支持单一标准的，灵活性和扩展性比较差。文献[73]针对无线通信协议，提出了一种高性能可配置译码器，可用于 Wi-Max，WLAN，3GPP2，GSM 和 LTE 标准中。该译码器基于 Xilinx FPGA 实现，功耗为 78mW。

3. 基于连续消除列表的 polar 译码器

2009，Arikan[125]提出 polar 码。由于 Polar 码在二进制输入离散无记忆信道（Binary-input Discrete Memoryless Channel，B-DMC）中可以达到香农极限，一经提出便引起了学者关注。Polar 码以其低复杂度的编译码结构，在 2016 年 3GPP RAN1 87 次会议上被选为 5G 控制信道增强型移动宽带（Enhance Mobile Broadband，eMBB）场景数据编码方案。

Arikan 提出的连续消除（Successive Cancellation，SC）译码算法，复杂度为 $O(N\log N)$。该算法具有 FFT 结构，可以实现资源共享，因此比其它译码算法，例如置信传递（Belief Propagation，BP）算法[126]，更适合硬件实现。尽管 polar 码可以达到香农极限，但 SC 算法中错误节点可以传递到下一节点，且该错误在中短码长译码中难以被消除，这严重影响了 SC 译码性能。为了解决这个问题，提高有限长度 polar 码译码性能，连续消除列表（Successive Cancellation List，SCL）算法被提出，该算法与最大似然算法性能相当[127,130]。

SCL 算法保留 L 条候选路径，所有路径同时进行 SC 译码，可看作是基于码树的路径搜索，译码复杂度随搜索路径数目增加而增加。2012 年，Bin Li[128]提出一种 CRC 辅助的自适应 SCL 译码器，搜索路径初始化为 1，译码结束后若 CRC 校验

不通过，则搜索路径数目加倍，重新进行译码。该方法在信道环境恶劣情况下迭代译码数目过高，译码延迟比较大。2014 年，Balatsoukas Stimming[129]基于对数似然值进行了 SCL 译码器硬件实现，在 L=4 时数据吞吐为 124Mbps。随后，在 2015 年进行了基于 LLR 值的译码器设计[131]。同年，多比特低延迟 SCL 译码器由 Yuan B[130]提出，SCL 算法中 60%以上的延迟集中在译码最后几个阶段，降低最后几个阶段的译码延迟对提升整体吞吐有重大意义。该文献提出 2^kb-rSCL 算法，将最后 k 个阶段合并，2^k 个比特可以同时得到译码结果。该算法随着 k 增大，候选路径数目 2^{2^k} 越大，带来的硬件开销也随之增长。2016 年，Lin J[132]设计一种高吞吐 SCL 译码器，该译码器基于简化 SC（Simplified Successive-Cancellation，SSC）算法[133]，对于 Rate-1 节点设置参数阈值，选择可靠度比较高的 L 条候选路径，在码长 N=1024，L=4 时，数据吞吐为 570Mbps。后续基于 SSC 算法衍生了一系列高速 polar 译码器，该算法也是目前实现低延迟高吞吐 polar 译码器最有前景的算法之一。2013 年提出的 ML-SSC 算法[134]，对 Rate-R 节点进行优化，在 200MHz 主频，码长为（32768,29491）时，数据吞吐为 1.82Gbps。2014 年提出的快速 polar 译码器[135]，对 Rate-R 节点进行划分，提出三种新型节点类型：SPC（Single-Parity-Check-code）节点、REP（REPetition-code）节点、REP-SPC 节点。新型节点的提出，打破了 SC 算法天然串行特定，难以并行实现的瓶颈，是在算法层次提高 SCL 译码器数据吞吐的一次飞跃。该算法硬件实现中[136]，长度为 N=1024，L=4 时，数据吞吐为 1.146Gbps。

SCL 算法为了避免搜索路径数目指数增长，当达到最大搜索路径数目时，对搜索路径进行删减，只保留 L 条候选路径。一般来说，SCL 路径分裂和删减运算量比较大，关于路径分裂和删减是一个研究热点。目前已知文献中，快速简化 SCL 算法是针对 Rate-1 节点进行，在没有任何性能损失情况下，可以在 $\min(L\text{-}1,n_v)$周期内将长度为 n_v 的 Rate-1 节点译出，而传统 SCL 需要 $3n_v$-2 周期，SSCL 算法（Simplified SCL，SSCL）需要 n_v 周期[137]。文献[138]提出一种低复杂度 SCL 译码算法，在路径搜索过程中，度量值比最大度量值小的路径被删除。文献[139]提出一种基于阈值的删减策略提高低 SNR 区间误块率（Block Error Rate，BLER）性能。以上两种方法均是基于后验概率（A Posteriori Probability，APP）。文献[140]提出一种降低路径分裂数目的 SCL 译码算法，该算法中设置一个记录路径连续分裂次数计数器，当该计数器值小于某阈值时，该条路径将被删除。该算法经过若干译码阶段后，仅会留下一条候选路径，因此译码性能不是很理想。SCL 算法关于信息比特路径分裂和删减仍在不断研究和探索中。

1.3 主要工作和内容安排

1.3.1 主要工作

本课题对面向通用通信基带处理器的专用协处理器进行研究与设计。首先分析了当前主流通信协处理器架构，针对其不足，提出了一种二维可配置协处理器架构；在确定协处理器架构后，对协处理器内部信道译码加速引擎进行研究设计，具体工作如下：

协处理器架构研究与设计。协处理器架构影响芯片整体性能发挥。不同协处理器架构，其工作机制有所不同。本文研究了协处理器在 UCP-SOC（System On Chip）系统中的地位，以及协处理器启停机制。针对加速引擎种类和数目，并结合无线通信算法，提出了一种新型二维可配置协处理器架构。本文对无线通信接收端协处理器进行了寄存器传输级（Register Transfer Level，RTL）设计和实现，包括协处理器内部控制逻辑和存储系统设计，并给出了 DC（Design Compiler）综合结果。

Turbo 译码加速引擎研究与设计。作为无线通信接收端协处理器中重要的加速引擎之一，turbo 译码器是 LTE 系统中复杂度最高、耗时最多的模块。本文研究了 turbo 译码器并行化算法，并对译码器并行度和滑窗长度进行了算法仿真。针对 turbo 译码器在信道环境比较差或遇突发错误情况下，译码器迭代次数过高，性能增益几乎为零的问题，本文提出了一种二阶差分辅助的 CRC 校验停止迭代准则。实验证明该准则可有效改善信道环境比较差或突发错误情况下，译码器迭代次数过高问题。本文还对高性能 turbo 译码器进行了设计和实现，并给出了 DC 综合结果。

Viterbi 译码器加速引擎研究与设计。作为无线通信接收端协处理器中重要的加速引擎之一，本文设计和实现了一款支持多标准的高性能可配置 viterbi 译码器，该译码器支持码率 1/2、1/3、1/4，约束长度 5~9，生成多项式任意配置，支持零结尾和咬尾，具有一定的灵活性和扩展性。本文完成了可配置 viterbi 译码器的算法设计和仿真，进行了 RTL 设计实现和完备性验证，最后给出了 DC 综合结果。

Polar 译码器路径分裂和路径删减优化。SCL 算法由于路径分裂和路径删减运算量比较大，导致译码延迟比较大，尤其在搜索路径数目比较大的情况下更加明显。本文提出了一种路径扩展优化方法，避免冗余路径分裂，降低译码延迟；同时，理论证明该方法对译码性能没有任何损失；此外，提出了一种基于置信区间的新型删减策略，降低了 SCL 译码复杂度。本文对优化方法进行了仿真实验，对 polar 译码器 BLER 性能和运算复杂度进行了对比和分析。

1.3.2 内容安排

第 1 章为绪论部分，介绍课题的研究背景和意义，对通信专用协处理器架构和信道译码加速引擎国内外研究现状进行介绍，指出当前研究内容的问题与不足，引出本课题研究内容，最后概述本课题的主要工作和内容安排。

第 2 章为二维可配置协处理器架构研究与设计。通过分析主流通信专用协处理器架构特点及不足，提出一种面向通用通信处理器的协处理器架构。对新型协处理器架构在互连网络功耗、总线带宽占用比、软件调度频率方面进行了性能评估。对协处理器内部控制逻辑、存储系统设计与实现进行了介绍，并给出了综合结果。

第 3 章为基于二阶差分辅助的 CRC 校验停止准则 turbo 译码器研究。针对 CRC 校验停止准则在信道环境比较差或译码器遇到不可纠正错误情况下，译码器迭代次数比较高的问题进行了优化。介绍了 turbo 迭代译码算法和基于二阶差分的 CRC 校验停止准则，通过仿真实验，对该准则下 turbo 译码器性能进行了评估。最后，对高性能 turbo 译码器设计和实现进行了介绍，并对数据吞吐、面积和功耗进行评估。

第 4 章为高性能可配置 viterbi 译码器研究与设计。介绍了基于滑窗流水的后向回溯法，并对兼容多标准的 viterbi 译码器可配置性进行了分析。对基于滑窗流水的后向回溯算法进行了浮点仿真和定点化设计，并对基四算法下加比选结构进行优化，最后对译码器数据吞吐、面积和功耗进行评估。

第 5 章为基于 SCL 的 polar 译码器路径扩展和路径删减优化。介绍了 polar 码和 SCL 译码算法，对路径扩展优化方法进行了理论证明；分析了路径删减引起的错误概率，在误差允许范围内，提出了一种基于置信区间的路径删减策略，并使用仿真实验验证方法有效性。最后，对优化方法进行复杂度分析和性能对比。

第 6 章为总结与展望。总结本文的主要内容，同时展望下一步的工作计划。

第2章 二维可配置协处理器架构

2.1 引言

目前，主流通信处理器均采用多核架构。通过任务划分，可以在特定时间执行多个任务，在实时通信系统中得到了广泛应用。但主流通信专用协处理器架构仍然存在一些不足，例如总线挂接设备多，互连网络功耗比较大；协处理器一维配置，调度比较频繁；协处理器多元化配置，软硬件开销大等。

本文通过分析主流通信专用协处理器架构特点和不足，提出了一种面向通用通信基带处理器的新型二维可配置协处理器架构。该架构将加速引擎分簇，并以特定工作模式重新编程加速引擎内部联结关系。第一维配置为工作模式和协处理器公共参数配置，由主处理器发起，协处理器实时响应；第二维配置为加速引擎私有参数配置，由主处理器离线完成。通过功耗评估模型，该架构下总线互连网络功耗仅为主流通信处理器架构的 1/3；通过对协处理器加速引擎分簇，一个标准数据帧处理，总线带宽占用比由 6.88%降到 2.05%；此外，该新型架构在加速引擎调度频率方面也进行了优化。最后，本文对面向基站的无线通信接收端协处理器进行了设计实现，给出了基于 Synopsys Design Compiler 工具的综合结果。

2.2 主流通信专用协处理器架构特点及不足

1. 总线挂接设备多，互连网络功耗大

根据国际半导体技术发展线路图（International Technology Roadmap for Semiconductors，ITRS）预测，随着工艺技术的升级换代，加速引擎的数量将成指数增长，如图 2-1 所示。加速引擎直接挂接到总线上，属于经典协处理器架构，以 TI TMS320C6670 和 CEVA-XC 处理器为代表。经典协处理器架构通常包括主处理器单元、存储器、加速引擎阵列（Processing Engine，PE）和一系列外围设备，如图 2-2 所示。

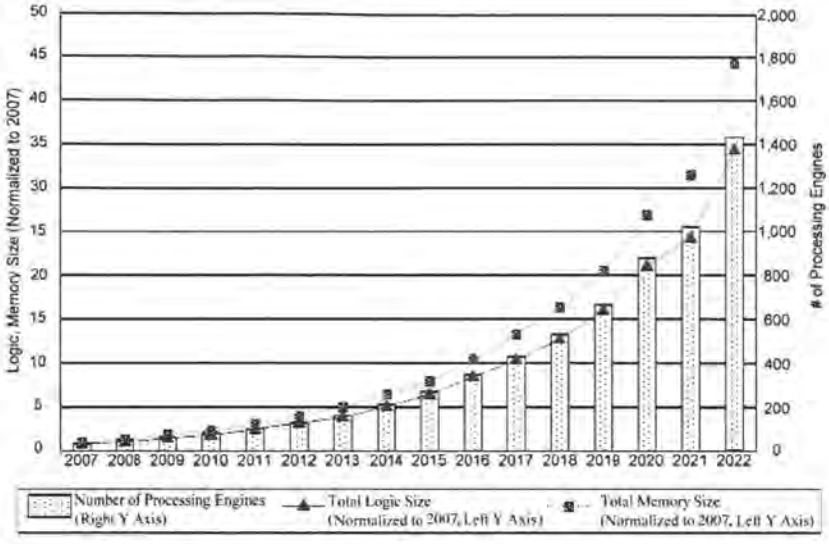

图 2-1：便携设备 SOC 设计复杂性趋势

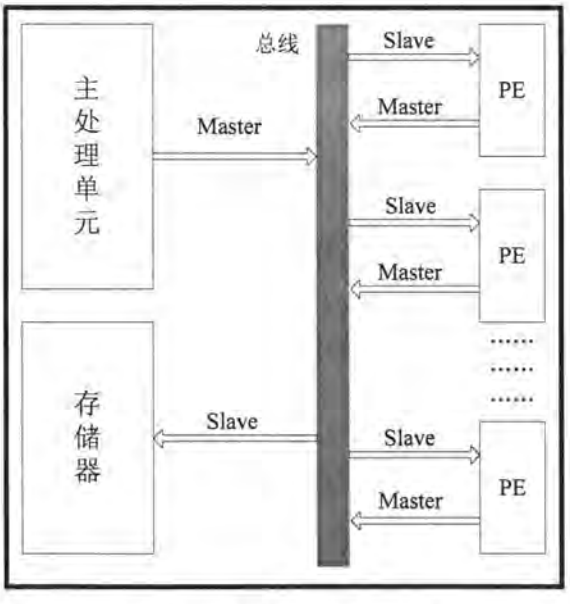

图 2-2：经典协处理器架构

经典协处理器架构中，片上系统互连网络随着接入节点数目增加，其面积、功耗、延迟也会急剧增加。文献[74]基于分析模型提出了一种片上互连建模工具，并利用 40nm CMOS 电路仿真结果进行校准。该工具得出了功耗评估模型，如公式(2.1)

所示，其中 N 表示节点数目，DW 表示总线数据位宽，C_{wire} 是输入信号、输出信号和内部转换信号的线电容总和，C_{mux} 表示多路复用器转换一个数据位时所有门电路电容和扩散电容之和。

$$power = N \times DW \times (C_{wire} VDD^2 + C_{mux} VDD^2) \tag{2.1}$$

由功耗评估模型可以看出，总线上接入节点数目越多，总线电容负载越大，功耗也越大。文献[75]通过 90nm CMOS 设计，得出了互连网络功耗关于 N 和 DW 的函数模型。当 $N<32$ 时，功耗为 $O(N^2DW)$，并附加 25%~50%额外功耗开销；当 $N \geq 32$，功耗为 N 的超二次函数，并附加 40%~80%额外功耗开销。研究表明，互连网络功耗约占总导线功耗的 90%[76]，随着总线位宽的增加，总线功耗占比会越来越大。

根据以上分析可以看到主流商用 DSP 架构的明显缺点：加速引擎阵列全部挂接在总线上。当总线接入节点数量较多时，互连网络功耗比较大。

2. 协处理器一维配置，调度频繁

经典协处理器架构中加速引擎只有一维配置，如 MorphoSys 协处理器[77]，加速引擎阵列通过“Context Word”进行配置；REMARC 架构[78]，主处理器将配置指令写入协处理器内部指令存储器，协处理器接收配置，执行指令。一维配置下，主处理器根据任务队列，对加速引擎进行调度；加速引擎工作结束后发出中断通知主处理器；主处理器接收中断后，对下一加速引擎进行调度。

协处理器一维配置下调度比较频繁，主处理器对配置总线访问频率比较高，寄存器配置、加载时间开销比较大；同时，协处理器的频繁调度会增加主处理器的时序控制复杂度。

3. 协处理器多元化配置，软硬件开销大

飞思卡尔 MSC8157 协处理器架构与经典架构不同，协处理器中除了加速引擎和数据存储器外，还有若干 RISC 处理器以及相应的指令存储器。以 MAPLE-B 为例，每个 RISC 处理器对应 16KB 指令存储器。RISC 处理器主要承担配置寄存器解析、任务分解、模块调度以及数据流控制等任务。MAPLE-B 采用基于 Buffer Descriptor 的编程模型，参数信息经两级配置到达期望寄存器，配置过程如图 2-3 所示。配置参数存储介质经历了外部 SRAM（Static Random Access Memory，静态随机存取存储器）、内部 SRAM，最后到达期望寄存器。MAPLE-B 中内部 SRAM 大小为 12KB。

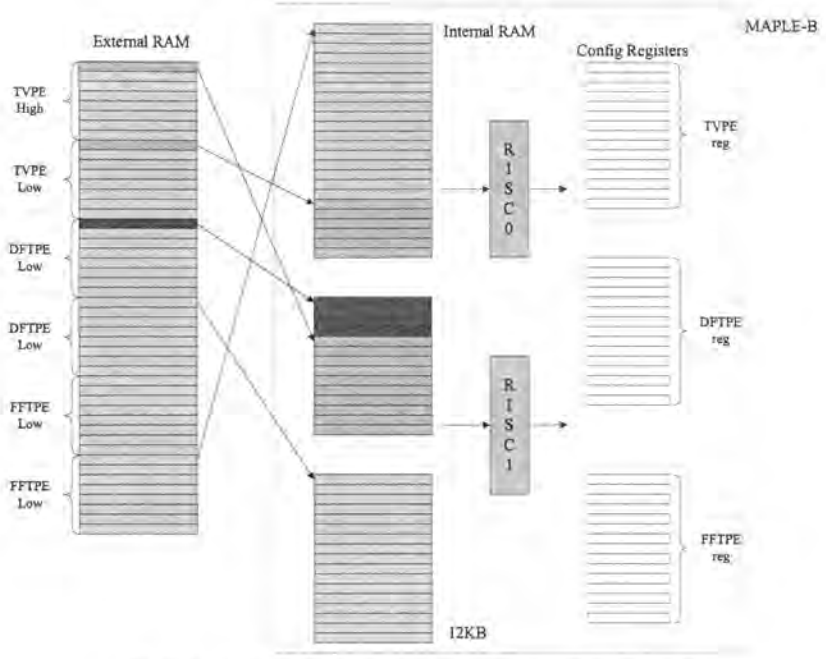

图 2-3：MAPLE-B Buffer Descriptor 参数配置流

MAPLE-B 协处理器配置编程模型包括三元配置：参数初始化配置、基于 RAM 的 Buffer Descriptor 配置、加速引擎寄存器配置。以 TVPE 加速引擎配置为例，参数配置如表 2-1 所示。从配置编程模型可以看出，三类配置寄存器均为加速引擎私有参数，可以归为一元配置，减少不必要的软件开销。此外，参数配置流可以通过基于 RAM 的寄存器编址，直接由外部存储器到达期望寄存器，减少协处理器内部缓存，降低硬件开销。

表 2-1：TVPE 加速引擎参数配置

配置类型	配置参数
初始化配置	Turbo 提前退出迭代的 CRC 开关、后验信息质量开关、HARQ 使能开关、调度子译码器个数开关等
基于 RAM 的 Buffer Descriptor 配置	Viterbi 译码器生成多项式、约束长度、结尾方式，turbo 译码算法、最大/最小迭代次数、软/硬判决开关、码块长度、数据加载/写回地址等
加速引擎寄存器配置	Turbo 译码器退出迭代的后验信息阈值、数据输出顺序等

2.3 新型二维可配置协处理器架构

针对目前主流通信专用协处理器架构存在不足，并结合无线通信数据处理流程和自研芯片 UCP 体系结构，提出一种面向通用通信基带处理器的新型二维可配置协处理器架构。如图 2-4 所示。

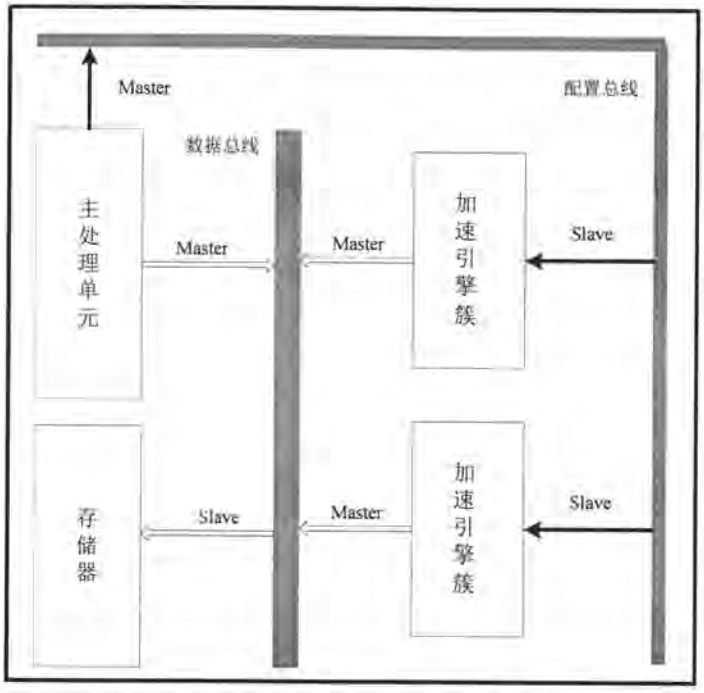

图 2-4：面向通用通信处理器的新型二维可配置协处理器架构

将协处理器中的加速引擎分簇。分簇后的加速引擎阵列分时共享簇内一组总线接口。协处理器配置编程模型采用二维配置，减少了主处理器与协处理器交互频率。其中，第一维配置为工作模式和协处理器公共参数配置，由主处理器发起，协处理器实时响应；第二维配置为加速引擎私有参数配置，由主处理器离线完成。通过功耗评估模型，总线互连网络功耗仅为主流通信处理器架构的 1/3；对于无线通信标准数据帧处理，总线带宽占用比由 6.88%降到 2.05%。

2.3.1 面向 UCP 的协处理器架构

无线通信数据处理流程示例如图 2-5、图 2-6 所示。UCP 擅长大密集数据处理，且标量处理器和向量处理器之间的互动比较紧密。若协处理器频繁发出中断，会打

乱 UCP 处理时序，增加标量处理器时序控制复杂度，这是不希望看到的结果。

基于无线通信数据处理流程和 UCP 的体系结构，得出无线通信系统级解决方案：符号级数据处理由 UCP 负责，比特级数据处理由协处理器负责。比特级数据处理包括发送端硬比特处理和接收端软比特处理。本文中将发送端协处理器称为 TxCP（Transmission CoProcessor，TxCP），接收端协处理器称为 RxCP（Receiver CoProcessor，RxCP）。RxCP 与 TxCP 统称为通信专用协处理器（Communication Specific CoProcessor，CSCP），CSCP 在 UCP 片上系统中的架构如图 2-7 所示。

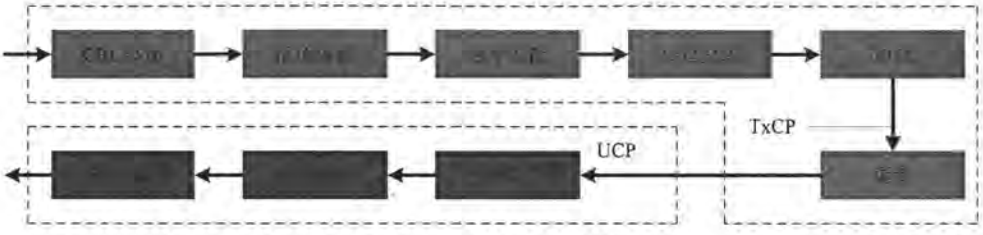

图 2-5：LTE-R9 PDSCH 信道数据处理流程

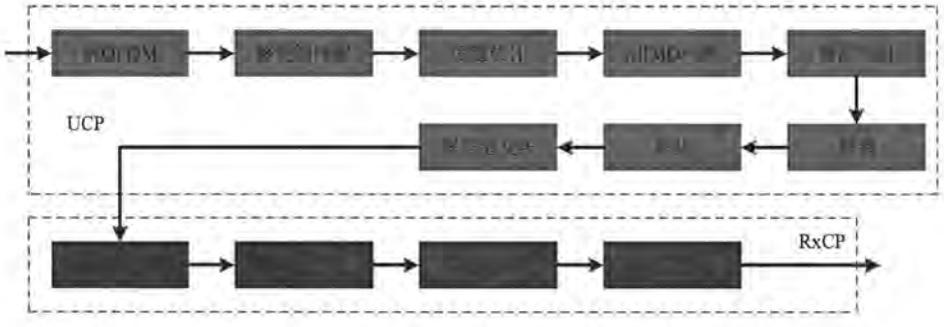

图 2-6：LTE-R9 PUSCH 信道处理流程

TxCP 协处理器包含的加速引擎有 CRC、turbo 编码协处理器（Turbo Encoding Coprocessor，TEC）、卷积码编码器（Convolutional Code Encoding，CCE）、速率匹配（Rate Matching，RM）、信道交织模块（Channel Interleaver Module，CIM）和加扰调制协处理器（Scrambling Modulation Coprocessor，SMC）。

RxCP 协处理器包含的加速引擎有解速率匹配（DeRate Matching，DeRM）、turbo 译码协处理器（Turbo Decoder Coprocessor，TDC）、viterbi 译码协处理器（Viterbi Decoder Coprocessor，VDC）和 CRC。

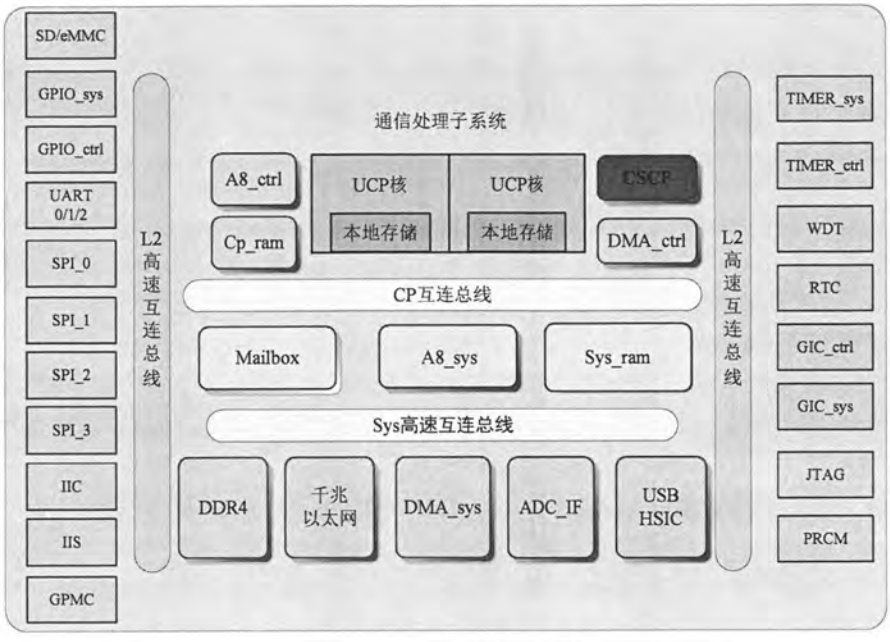

图 2-7：UCP SOC 架构

协处理器中各加速引擎功能及特性说明如下：

1. CRC
   - ✓ 8 比特并行校验
   - ✓ 支持 CRC24A、CRC24B、CRC16、CRC8
   - ✓ 支持掩码校验
2. TEC
   - ✓ 8 比特并行编码
   - ✓ 支持 1/3 码率、188 种码块长度编码
3. CCE
   - ✓ 8 比特并行编码
   - ✓ 支持 1/2、1/3、1/4 码率
   - ✓ 支持约束长度 5~9
   - ✓ 支持零结尾和咬尾结尾
   - ✓ 支持可配置生成多项式

4. RM
   - ✓ 支持数据信道和控制信道速率匹配
   - ✓ 内嵌码块级联功能
   - ✓ 支持非对齐输出
5. CIM
   - ✓ 支持所有长度信道质量指示（Channel Quality Indicator，CQI）数据处理
   - ✓ 支持确认字符（Acknowledgement，ACK）、秩指示（Rank Indication，RI）、CQI 压缩输入
   - ✓ 支持常规/扩展循环前缀（Cyclic Prefix，CP）配置
   - ✓ 支持探测参考信号（Souding Reference Signal，SRS）传输
6. SMC
   - ✓ 支持正交相移键控（Quadrature Phase Shift Keying，QPSK）、16QAM（Quadrature Amplitude Modulation，QAM）、64QAM 调制
   - ✓ 支持可配置 X、Y 占位符
   - ✓ 内嵌伪随机序列生成逻辑
7. DeRM
   - ✓ 支持数据信道和控制信道解速率匹配
   - ✓ 支持 HARQ 合并
   - ✓ 支持四种重传版本数据合并
8. TDC
   - ✓ 基四算法
   - ✓ 支持四路或八路并行译码
   - ✓ 支持 Log-MAP 算法和 Max-Log-MAP 算法
   - ✓ 支持 188 种码块长度译码
9. VDC
   - ✓ 支持 1/2、1/3、1/4 码率
   - ✓ 支持约束长度 5~9
   - ✓ 支持零结尾和咬尾结尾

✓ 支持可配置生成多项式

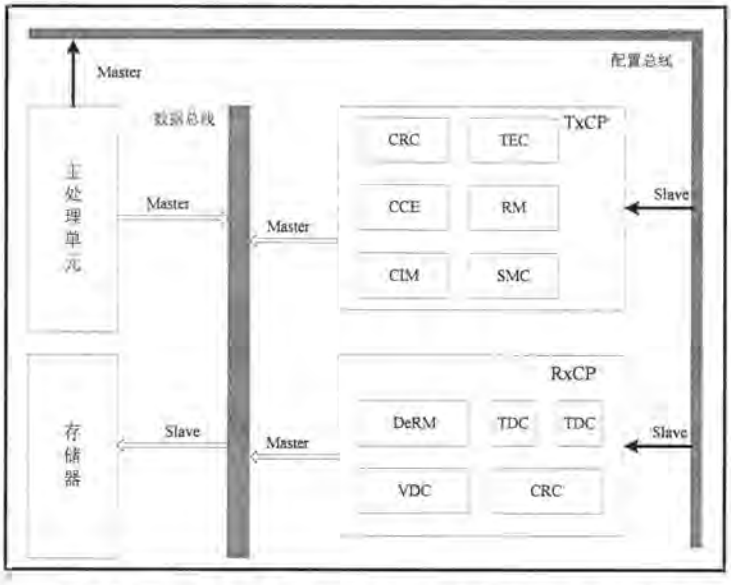

图 2-8：面向 UCP 的新型二维可配置协处理器架构

将加速引擎阵列依据发送和接收分离的准则进行分簇，分簇后协处理器与主处理器之间的关系如图 2-8 所示。经过分簇后的加速引擎阵列，分时共享簇内一组总线接口。协处理器配置编程模型采用二维配置，减少了主处理器与协处理器交互频率。其中，第一维配置为工作模式和协处理器公共参数配置，由主处理器发起，协处理器实时响应；第二维配置为加速引擎私有参数配置，由主处理器离线完成。配置寄存器为两元配置，配置流直接由外部 SRAM 到达期望寄存器，降低了软硬件开销。

2.3.2 新型二维可配置协处理器架构性能分析

1. 互连网络功耗

随着芯片加速引擎集成数目增加，片上互连网络消耗大量能量[84]。降低互连网络功耗对提升系统整体性能意义重大。

文献[74]对互连网络功耗进行了理论建模，模型重点关注了节点数目、总线位宽和开关电容电流。并在 40nm CMOS 工艺下，通过电路级后仿，对理论模型进行校准。互连网络功耗模型如公式(2.1)所示。对于特定位宽的总线，当相同节点设备

接入总线时，互连网络功耗与接入节点数量成线性增长关系。

文献[85]通过网络仿真工具 OPNET 对 2D-Mesh 和 2D-Torus 网络拓扑结构进行了互连网络功耗评估，仿真方案采用均匀流量的网络通信模式，即节点间等概率发生数据传输。不同网络节点数目下，两种网络拓扑结构的互连网络功耗仿真结果如图 2-9、图 2-10 所示。从图中可以看出，随着互连网络节点数目增加，网络功耗也随之增加，并呈现出线性增长关系。当节点数达到某个程度后，互连网络功耗保持不变。这是由于网络中数据包传输出现阻塞，没有通过路由传输所致。

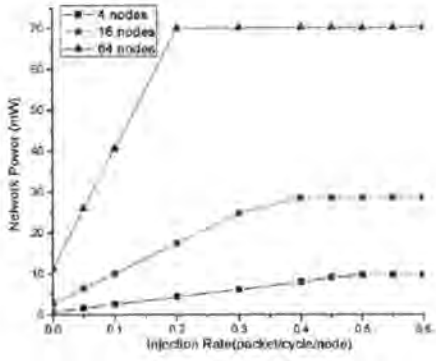

图 2-9：2D-Mesh 不同节点功耗仿真结果

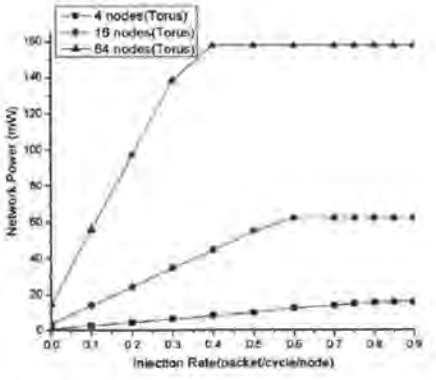

图 2-10：2D-Torus 不同节点功耗仿真结果

接下来基于互连网络功耗模型和电路级仿真结果，对面向 UCP 的二维可配置协处理器架构进行互连网络功耗评估。

假设协处理器中加速引擎个数为 N，经典架构下互连网络功耗为 $P_{typical}$。新型二维可配置协处理器将加速引擎分簇。假设分簇后节点数目为 n，则互连网络功耗

如公式(2.2)所示。其中，用 $a=n/N$ 表征总线分簇比。可以看出：总线分簇比越低，互连网络开销越小，功耗也越低。

$$P_{novel} = aP_{typical} \tag{2.2}$$

接下来以无线通信系统中标准数据帧处理为例来说明新型架构下互连网络动态功耗。无线通信接收端部分数据处理流程如图 2-11 所示，对应的加速引擎有 DeRM、TDC 和 CRC。

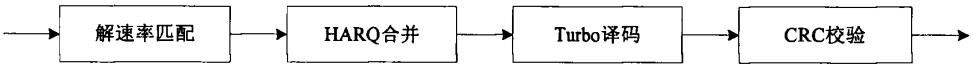

图 2-11：无线通信接收端部分处理流程

单位数据通过互连网络传输，产生的功耗一般包括三部分：交换开关功耗、链路功耗和时钟树功耗，如公式(2.3)所示。若节点 i 到节点 j 之间传输数据包个数为 M，互连网络节点数目为 N，则节点 i 到节点 j 之间的互连网络功耗和网络总功耗分别如公式(2.4)和(2.5)所示，其中，时钟树功耗抽象为网络节点数目 N 的函数。互连网络功耗与节点间传输数据包长度以及节点数目有关，可以进一步将互连网络功耗用公式(2.6)表示，其中 BW 表示总线访问带宽，表征访问节点数目以及节点间传输的数据包长度。

$$P_{network} = P_{switch} + P_{path} + P_{clocktree} \tag{2.3}$$

$$P_{i\to j} = \sum_{k=0}^{M-1}(P_{switch}(k) + P_{path}(k)) + P_{clocktree} \tag{2.4}$$

$$P_{total} = \sum_{s=0}^{N-1}\sum_{k=0}^{M-1}(P_{switch}(s,k) + P_{path}(s,k)) + P_{clocktree}(N) \tag{2.5}$$

$$P_{total} = (P_{switch} + P_{path}) * BW + P_{clocktree} \tag{2.6}$$

假设接收机 1ms 内处理的一个标准数据帧长为 75376，分割成的码块长度为 5824，码块个数为 13。经典协处理器架构下，1ms 内图 2-11 所示数据处理流对总线带宽访问如图 2-12 所示，带宽访问量为

$$(14400*8+2*5828*3*8+5824*2)*13 = 5285696(\text{bit/ms}) \tag{2.7}$$

新型架构下，协处理器对总线带宽访问如图 2-13 所示。协处理器带宽访问量

计算如公式(2.8)所示，不足经典协处理器架构的 1/3。

$$(14400*8+5824)*13=1573312(bit/ms) \tag{2.8}$$

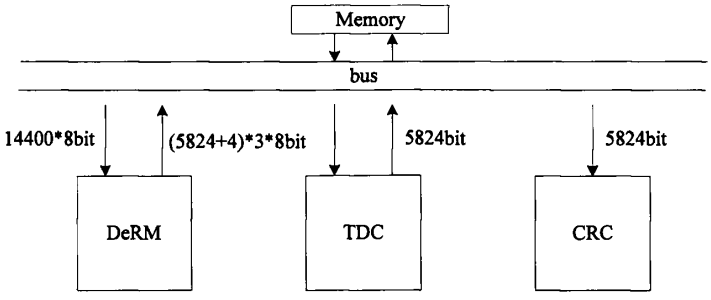

图 2-12：经典协处理器架构总线带宽访问量示意图

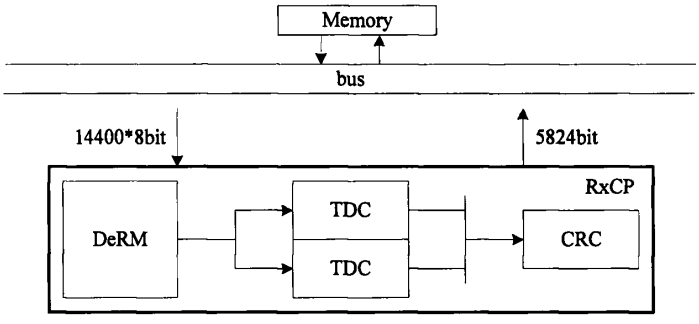

图 2-13：新型协处理器架构总线带宽访问量示意图

系统功耗包括簇内功耗和总线互连功耗，簇内模块产生的结果数据可以直接送入下一模块进行消费，降低了数据流动。信号翻转引起的电容充放电是动态功耗的主要来源。通过分簇，可以降低数据搬运次数，降低互连网络功耗，进而降低系统整体功耗。

2. 总线带宽占用比

配置总线占用的带宽比较小，这里只讨论数据总线带宽占用比。本文用 BW_{load}、BW_{store} 表示总线带宽占用，则经典协处理器架构下，协处理器占用的总线带宽可以表示为

$$BW_{typical-total}=BW_{0\_load}+BW_{0\_store}+BW_{1\_load}+BW_{1\_store}+...+BW_{N-1\_load}+BW_{N-1\_store} \tag{2.9}$$

对加速引擎分簇后，簇内某些加速引擎产生的结果数据可以立即被下一加速引擎消耗，无需通过总线传递。图 2-11 所示无线通信系统接收端处理流程，协处理器内部根据工作模式配置，可以将 DeRM 的输出结果直接写入 TDC 缓存区，TDC

的输出结果直接写入 CRC 缓存区，如图 2-14 所示。此种工作模式下，协处理器占用总线带宽如公式(2.10)所示。

$$BW_{novel-total} = BW_{0\_load} + BW_{N-1\_store} \tag{2.10}$$

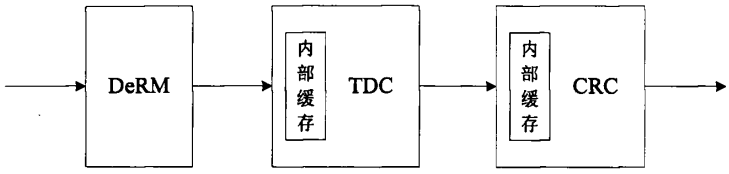

图 2-14：协处理器内部数据处理流

下面仍以标准数据帧长 75376 为例讨论协处理器总线带宽占用比。协处理器处理一个码块时总线带宽访问量如图 2-12 所示。假设总线位宽为 128 比特，时钟频率为 600MHz，则协处理器架构优化前总线带宽占用比如公式(2.11)所示，大约为 6.88%。

$$r_{优化前} = \frac{(14400*8+2*5828*3*8+5824*2)*13*1000}{600*128*10^6} = 0.0688 \tag{2.11}$$

协处理器架构优化后总线带宽占用比如公式(2.12)所示，大约为 2.05%。

$$r_{优化后} = \frac{(14400*8+5824)*13*1000}{600*128*10^6} = 0.0205 \tag{2.12}$$

相比传统通信处理器架构，新型二维协处理器架构总线带宽占用比由 6.88%降到 2.05%，大大缓解了总线压力。

3. 软件调度频率

任务调度是系统架构设计中的经典问题[86]，尤其当前处理器朝着异构多核方向发展，体系结构设计时任务划分和任务调度不可忽视[87,88]。本文中对加速引擎分簇，利用协处理器内部控制逻辑单元分担部分加速引擎调度任务，减少协处理器与主处理器交互频率，从而降低软件开销。

无线通信系统中，会出现频繁调度某个加速引擎的情况。例如信道译码是分码块进行的，分的码块越多，对译码加速引擎调度越频繁。无线通信接收端多码块信道译码流程如图 2-15 所示。假设一个传输块分为 n 个码块，传统架构下主处理器对 DeRM、TDC、CRC 加速引擎调度次数为 $f_{typical}=2n+1$。二维可配置协处理器架构，将加速引擎以某种工作模式进行联结，可以使调度次数成倍降低。例如将 DeRM 与 TDC 进行联结，则信道译码处理流程如图 2-16 所示，此种工作模式下协处理器调

度次数为 $f_{DeRM+TDC}=n+1$。二维可配置协处理器还可以将 DeRM、TDC 与 CRC 进行联结，作为另一种工作模式进行调度。此种模式下，主处理器只需调度一次协处理器即可实现多码块信道译码。

新型协处理器架构，通过增加一维工作模式配置，可以大大降低主处理器对协处理器调度频率。尤其在高性能计算机[2]中，主处理器本身承担了大量数据处理任务，通过对协处理器进行二维配置，可以将主处理器从频繁调度中解放出来。

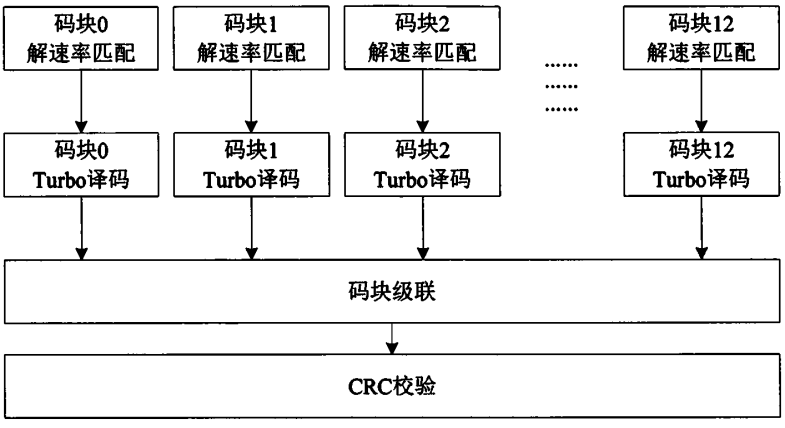

图 2-15：无线通信接收端多码块信道译码流程

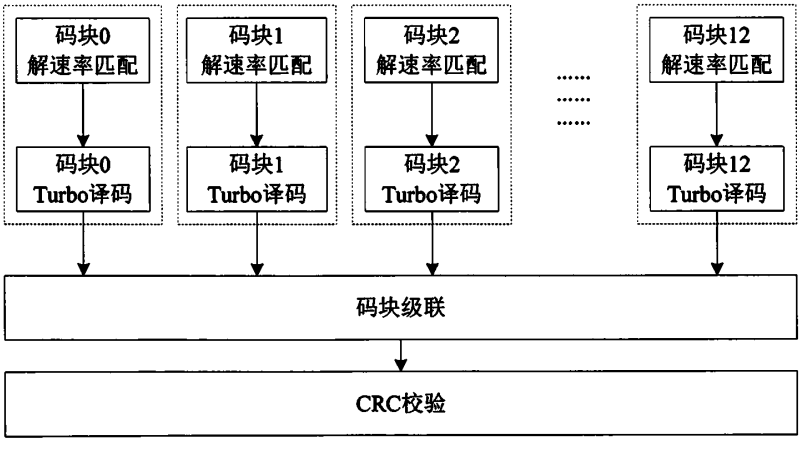

图 2-16：DeRM 与 TDC 联结后数据处理流程

2.3.3 协处理器工作机制

协处理器架构确定后，下面对其工作机制进行研究，主要从协处理器与主处理

器工作时序、协处理器配置管理、协处理器工作模式三方面来进行。

1. 协处理器与主处理器工作时序

LTE 系统中最小数据传输时间 TTI（Transmission Time Interval）为 1ms，即在 1ms 内基站做一次用户调度，处理完一个子帧的数据。一个子帧最多传输两个传输块数据，以下行发送端和上行接收端处理为例，每个传输块处理流程如图 2-5、图 2-6 所示。根据“UCP 处理符号级，协处理器处理比特级”划分原则，协处理器与主处理器的工作时序如图 2-17、图 2-18 所示，分别为物理层下行发送端工作时序和物理层上行接收端工作时序。基于符号级与比特级处理分离原则，可以实现主处理器与协处理器流水工作，进一步提高数据处理能力。

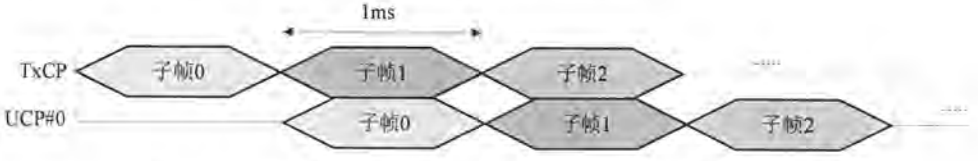

图 2-17：协处理器与主处理器物理层下行发送端工作时序

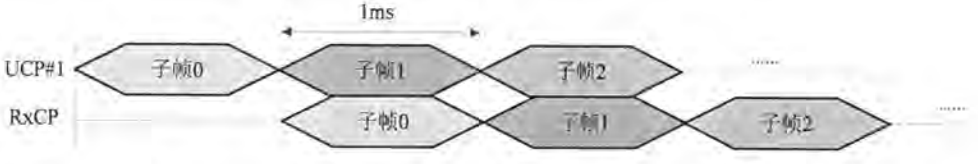

图 2-18：协处理器与主处理器物理层上行接收端工作时序

2. 协处理器配置管理

协处理器配置由 UCP 承担：UCP 接收处理任务，对处理任务进行分解，同时，对需要启动的硬件加速引擎工作参数进行计算。然后对协处理器工作模式和工作参数进行配置。配置类型分为两类：工作模式和公共参数配置、基于 RAM 的加速引擎私有参数配置。主处理器将待处理任务所需的数据和配置信息按一定格式存储于指定位置，然后激活协处理器。协处理器根据工作模式配置，加载该任务下数据激励和配置激励，完成处理任务。下面以接收端协处理器 RxCP 工作机制来说明 UCP 对协处理器的配置管理。

（1） UCP 接收从 MAC（Media Access Control）层下发的数据处理任务。首先进行符号级处理，处理流程为“解 OFDM→解资源映射→信道估计→

MIMO 检测→解层映射→解调解扰→解信道交织”。处理完毕后，将数据写入 UCP 与 RxCP 的共享缓存区。

（2） UCP 计算 DeRM、TDC 和 CRC 加速引擎工作参数，并将工作参数以配置寄存器堆的形式写入共享缓存区指定位置。

（3） UCP 查询 RxCP 工作状态。若 RxCP 处于空闲状态，对 RxCP 进行工作模式配置。此过程称为“初始配置字写入”。初始配置字位宽为 32 比特。

（4） RxCP 接收初始配置字，根据工作模式启动相应的子状态机。

（5） RxCP 工作过程中，UCP 开始处理下一子帧数据。RxCP 工作结束后，将处理结果写入共享缓存区，并发出中断。

（6） UCP 响应中断，下发新的处理任务。

UCP 对协处理器调度通过配置寄存器实现。配置寄存器分为两类，具体如下：

A. 第一类配置寄存器

第一类配置寄存器称为初始配置字（Initial Configuration Word，ICW），初始配置字说明如表 2-2 所示。每个子帧最多有两个传输块进行处理。由于主处理器与协处理器流水工作，需要预留两个传输块的初始配置字地址空间。

表 2-2：RxCP 初始配置字地址空间分配及说明

地址偏移	读/写	名称	说明
0x00	读/写	RxCP_TB0_CFG_BASE	传输块#0 配置寄存器堆基地址
0x04	读/写	RxCP_TB0_RX_CMD	传输块#0 RxCP 上报信息
0x08	读/写	RxCP_TB0_CMD	传输块#0 RxCP 工作模式配置
0x0C	读/写	RxCP_TB1_CFG_BASE	传输块#1 配置寄存器堆基地址
0x10	读/写	RxCP_TB1_RX_CMD	传输块#1 RxCP 上报信息
0x14	读/写	RxCP_TB1_CMD	传输块#1 RxCP 工作模式配置

B. 第二类配置寄存器

第二类配置寄存器存储在共享缓存区。RxCP 通过 Master 总线接口加载配置寄

存器堆。一个码块的配置寄存器堆地址空间分配及说明如表 2-3 所示。

表 2-3：RxCP 配置寄存器堆地址空间分配及说明

地址偏移	读/写	名称	说明
0x0	读/写	RxCP_T0_C0_DeRM_CFG00	解速率匹配配置寄存器 0
0x4	读/写	RxCP_T0_C0_DeRM_CFG01	解速率匹配配置寄存器 1
0x8	读/写	RxCP_T0_C0_DeRM_CFG02	解速率匹配配置寄存器 2
0xC	读/写	RxCP_T0_C0_DeRM_CFG03	解速率匹配配置寄存器 3
0x10	读/写	RxCP_T0_C0_DeRM_CFG04	解速率匹配配置寄存器 4
0x14	读/写	RxCP_T0_C0_DeRM_CFG05	解速率匹配配置寄存器 5
0x18	读/写	RxCP_T0_C0_DeRM_CFG06	解速率匹配配置寄存器 6
0x1C	读/写	RxCP_T0_C0_TDC_CFG00	Turbo 译码器配置寄存器 0
0x20	读/写	RxCP_T0_C0_TDC_CFG01	Turbo 译码器配置寄存器 1
0x24	读/写	RxCP_T0_C0_TDC_CFG02	Turbo 译码器配置寄存器 2
0x28	读/写	RxCP_T0_C0_TDC_CFG03	Turbo 译码器配置寄存器 3
0x2C	读/写	RxCP_T0_C0_TDC_CFG04	Turbo 译码器配置寄存器 4
0x30	读/写	RxCP_T0_C0_TDC_CFG05	Turbo 译码器配置寄存器 5
0x34	读/写	RxCP_T0_C0_TDC_CFG06	Turbo 译码器配置寄存器 6
0x38	读/写	RxCP_T0_C0_TDC_CFG07	Turbo 译码器配置寄存器 7
0x3C	读/写	RxCP_T0_C0_TDC_CFG08	Turbo 译码器配置寄存器 8
0x40	读/写	RxCP_T0_C0_TDC_CFG09	Turbo 译码器配置寄存器 9
0x44	读/写	RxCP_T0_C0_TDC_CFG10	Turbo 译码器配置寄存器 10
0x48	读/写	RxCP_T0_C0_TDC_CFG11	Turbo 译码器配置寄存器 11
0x4C	读/写	RxCP_T0_C0_TDC_CFG12	Turbo 译码器配置寄存器 12
0x50	读/写	RxCP_T0_C0_TDC_CFG13	Turbo 译码器配置寄存器 13

0x54	读/写	RxCP_T0_C0_TDC_CFG14	Turbo 译码器配置寄存器 14
0x58	读/写	RxCP_T0_C0_TDC_CFG15	Turbo 译码器配置寄存器 15
0x5C	读/写	RxCP_T0_C0_TDC_CFG16	Turbo 译码器配置寄存器 16
0x60	读/写	RxCP_T0_C0_TDC_CFG17	Turbo 译码器配置寄存器 17
0x64	读/写	RxCP_T0_C0_TDC_CFG18	Turbo 译码器配置寄存器 18
0x68	读/写	RxCP_T0_C0_TDC_CFG19	Turbo 译码器配置寄存器 19
0x6C	读/写	RxCP_T0_C0_TDC_CFG20	Turbo 译码器配置寄存器 20
0x70	读/写	RxCP_T0_C0_TDC_CFG21	Turbo 译码器配置寄存器 21
0x74	读/写	RxCP_T0_C0_TDC_CFG22	Turbo 译码器配置寄存器 22
0x78	读/写	RxCP_T0_C0_TDC_CFG23	Turbo 译码器配置寄存器 23
0x7C	读/写	RxCP_T0_C0_TDC_CFG24	Turbo 译码器配置寄存器 24
0x80	读/写	RxCP_T0_C0_TDC_CFG25	Turbo 译码器配置寄存器 25
0x84	读/写	RxCP_T0_C0_TDC_CFG26	Turbo 译码器配置寄存器 26
0x88	读/写	RxCP_T0_C0_TDC_CFG27	Turbo 译码器配置寄存器 27
0x8C	读/写	RxCP_T0_C0_TDC_CFG28	Turbo 译码器配置寄存器 28
0x90	读/写	RxCP_T0_C0_TDC_CFG29	Turbo 译码器配置寄存器 29
0x94	读/写	RxCP_T0_C0_TDC_CFG30	Turbo 译码器配置寄存器 30
0x98	读/写	RxCP_T0_C0_TDC_CFG31	Turbo 译码器配置寄存器 31
0x9C	读/写	RxCP_T0_C0_TDC_CFG32	Turbo 译码器配置寄存器 32
0xA0	读/写	RxCP_T0_C0_VDC_CFG00	viterbi 译码器配置寄存器 0
0xA4	读/写	RxCP_T0_C0_VDC_CFG01	viterbi 译码器配置寄存器 1
0xA8	读/写	RxCP_T0_C0_VDC_CFG02	viterbi 译码器配置寄存器 2
0xAC	读/写	RxCP_T0_C0_VDC_CFG03	viterbi 译码器配置寄存器 3
0xB0	读/写	RxCP_T0_C0_VDC_CFG04	viterbi 译码器配置寄存器 4

0xB4	读/写	RxCP_T0_C0_VDC_CFG05	viterbi 译码器配置寄存器 5
0xB8	读/写	RxCP_T0_C0_VDC_CFG06	viterbi 译码器配置寄存器 6
0xBC	读/写	RxCP_T0_C0_CRC_CFG00	CRC 配置寄存器 00
0xC0	读/写	RxCP_T0_C0_CRC_CFG01	CRC 配置寄存器 01
0xC4	读/写	RxCP_T0_C0_CRC_CFG02	CRC 配置寄存器 02

3. 协处理器工作模式

根据兼容多种无线通信标准，并支持一定扩展应用为原则，对发送端协处理器 TxCP 与接收端协处理器 RxCP 进行工作模式研究。以 RxCP 为例，其工作模式有七种。表 2-4 列出了 RxCP 工作模式及其应用场景。

表 2-4：RxCP 工作模式及应用场景

序号	工作模式	应用场景	工作流程
模式 1	DeRM+TDC+CRC	PDSCH、PCH、PMCH、PUSCH	解速率匹配（HARQ）→ turbo 译码→CRC24A
模式 2	DeRM+VDC+CRC	PBCH、PDCCH、UCI	解速率匹配→viterbi 译码→CRC16/CRC8
模式 3	DeRM+TDC	PDSCH、PCH、PMCH、PUSCH	解速率匹配（HARQ）→ turbo 译码
模式 4	DeRM	PDSCH、PUSCH	解速率匹配（HARQ）
模式 5	TDC	为其他解决方案预留	Turbo 译码
模式 6	VDC	为其他解决方案预留	Viterbi 译码
模式 7	CRC	为其他解决方案预留	CRC 校验

2.4 二维可配置协处理器实现

本节以接收端协处理器 RxCP 的设计与实现来说明二维可配置协处理器设计思想及实现细节。该协处理器主要面向基站数字基带处理。基站侧对数据吞吐要求比

较高，通常需集成多个 TDC 模块，本文根据系统要求和 TDC 性能指标，设置 TDC 个数为 2。

RxCP 顶层模块接口如图 2-19 所示，除时钟和复位信号外，还包括两组 Master 接口和一组 Slave 接口。Master 接口分别对应从共享缓存区和 DDR 加载数据接口。Slave 接口是 UCP 对 RxCP 初始配置字写入接口。

RxCP 整体架构如图 2-20 所示。除加速引擎 DeRM、TDC、VDC、CRC 外，还包括控制逻辑单元 Controller、数据接口 data interface、HARQ 接口 HARQ interface、配置寄存器堆加载单元 Cfg_Loader、码块分割单元 Code_Blk_Seg，以及五块内部存储器 RAM1~RAM5。其中，数据接口和 HARQ 接口均支持数据加载和写回操作。

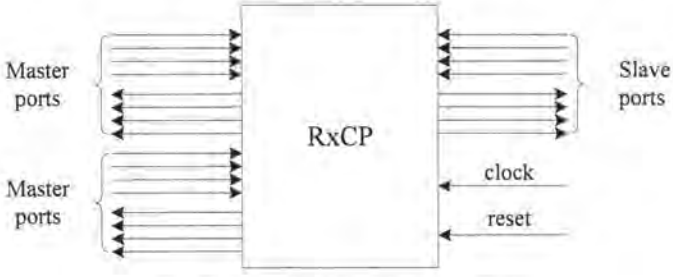

图 2-19：RxCP 顶层模块接口

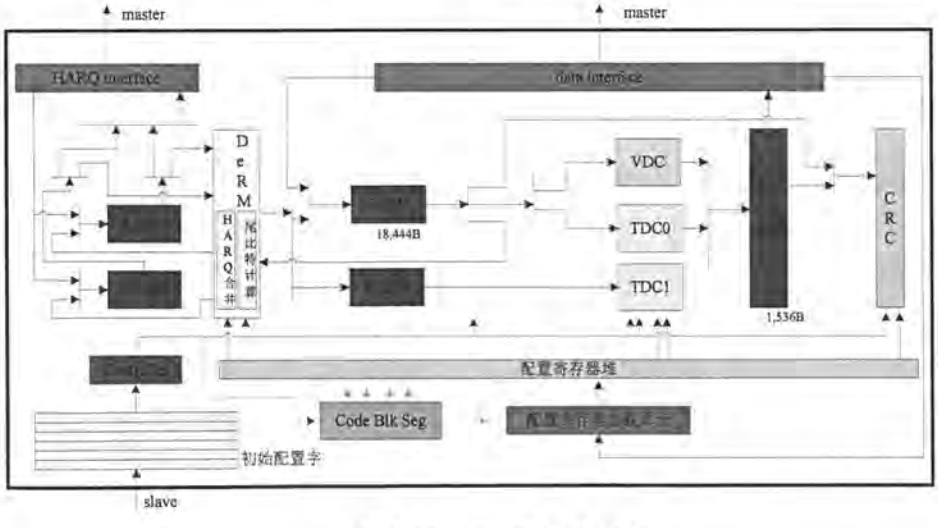

图 2-20：RxCP 整体架构

下面重点介绍协处理器 RxCP 的控制逻辑和存储系统设计与实现，最后给出协处理器综合结果。

2.4.1 协处理器控制逻辑设计与实现

二维可配置协处理器和主流商用协处理器相比，对加速引擎进行了分簇，簇内增加控制逻辑，协处理器可以自动实现特定配置下的工作模式。

RxCP 控制逻辑单元 Controller 处理流程如图 2-21 所示，其中 *cb_num* 表示码块个数，*cb_idx* 表示码块索引。

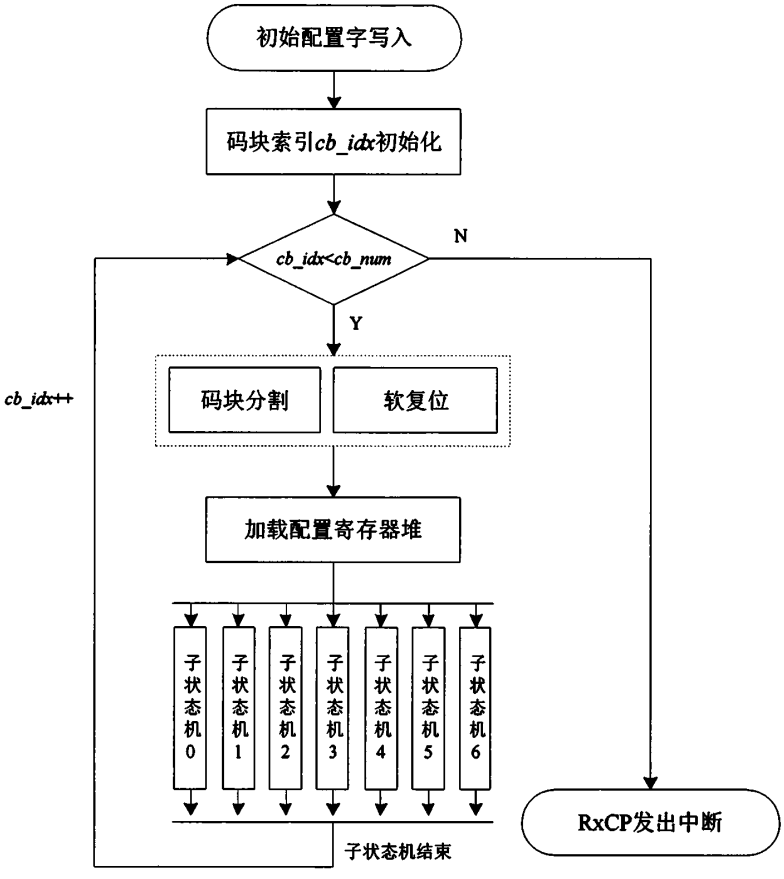

图 2-21：RxCP 控制逻辑处理流程图

RxCP 接收初始配置字后，解析出工作模式、码块个数、配置寄存器堆地址等宏观配置参数。将码块索引寄存器初始化为零，如果码块索引小于码块个数，进行码块分割，并发出软复位信号。码块分割主要用于计算每个码块的配置寄存器堆地址，软复位信号用于对加速引擎等逻辑单元进行初始化。得到码块配置寄存器堆地址后，RxCP 从共享缓存区加载一个码块的配置寄存器，得到每个加速引擎的工作

参数。根据工作模式配置，RxCP 启动相应的子状态机进行处理。子状态机结束后，开始处理下一码块。所有码块处理完毕后，RxCP 发出中断。

RxCP 中包含七个子状态机，分别对应七种工作模式。以表 2-4 中工作模式 1 对应的子状态机为例，说明子状态机工作流程，如图 2-22 所示。子状态机启动后，协处理器由空闲状态进入数据加载状态。同时，若 HARQ 使能（非首次重传），且速率匹配长度小于循环缓冲区长度，协处理器加载 HARQ 合并次数历史信息。数据信息加载结束后，协处理器进入解速率匹配状态；HARQ 合并次数加载结束后，协处理器继续加载 HARQ 重传数据，若速率匹配长度大于循环缓冲区长度，则协处理器直接由空闲状态进入加载 HARQ 重传数据状态。HARQ 合并次数加载结束后，进行 HARQ 合并次数更新。HARQ 合并次数更新结束后，将更新后的合并次数写回 DDR 指定位置。解速率匹配结束后，协处理器进入 turbo 译码状态，同时将 HARQ 更新后的重传数据输出到 DDR 指定位置。Turbo 译码结束后，进行 CRC 校验。CRC 校验结束后，将译码结果输出到共享缓存区指定位置。

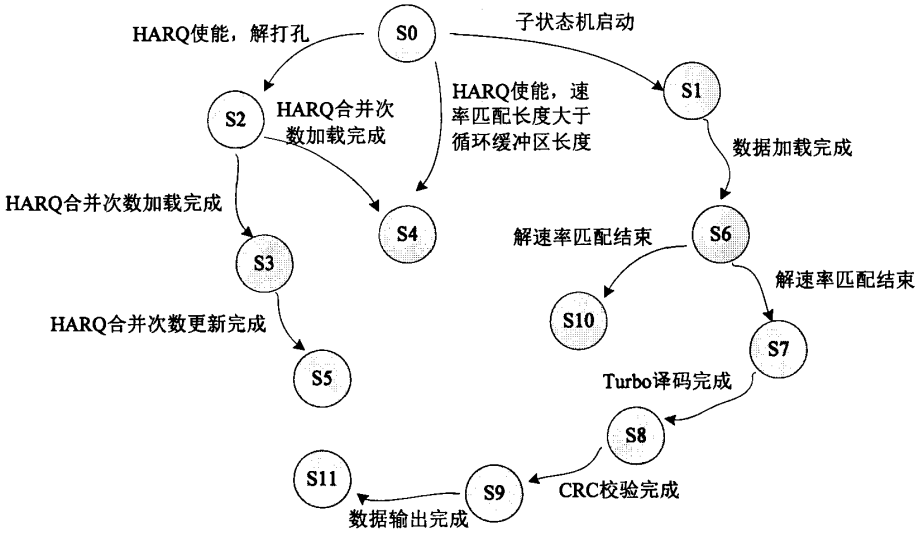

图 2-22：工作模式 1 对应的子状态机

子状态机状态说明如下：

S0：空闲

S1：加载数据

S2：加载 HARQ 合并次数历史信息

S3：HARQ 合并次数更新

S4：加载 HARQ 重传数据历史信息

S5：HARQ 合并次数输出

S6：解速率匹配

S7：turbo 译码

S8：CRC 校验

S9：译码结果输出

S10：HARQ 重传数据输出

S11：结束

2.4.2 协处理器存储系统研究与设计

协处理器 RxCP 中存储系统有三类：常规 SRAM 存储器、按字节寻址大位宽存储器、列进行出存储单元。

1. 常规 SRAM 存储器

此类存储器可直接由内存编译器（Memory Compiler）生成。

2. 按字节寻址大位宽存储器

RxCP 中解速率匹配加速引擎，需将位宽为 128 比特的数据写入按字节寻址的存储单元中。如果按照多周期写入，一次数据加载时间为 16 个周期，时间开销比较大。本论文采用的解决方案是将 16 块 8 比特位宽的 SRAM 拼接，加入 SRAM 地址、数据解析逻辑，打包成一块 128 位宽、按字节寻址的存储单元，可在一周期内将 128 位宽数据写入和读出。

假设输入存储单元的地址为 i，j 为要访问的 SRAM 编号，$addr_j$ 表示第 j 块 SRAM 的片内地址，$data_j$ 表示第 j 块 SRAM 数据。则每块 SRAM 的片内地址和片内数据计算公式分别如(2.13)和(2.14)所示。

$$\begin{aligned} addr_j &= addr_{(i+k)\bmod 16} \\ &= (i+k)/16 \qquad (k=0,1,2,...,15) \end{aligned} \tag{2.13}$$

$$\begin{aligned} data_j &= data_{(i+k)\bmod 16}[7:0] \\ &= data\_in[(8k+7):8k] \qquad (k=0,1,2,...,15) \end{aligned} \tag{2.14}$$

按字节寻址的大位宽存储单元结构如图 2-23 所示。

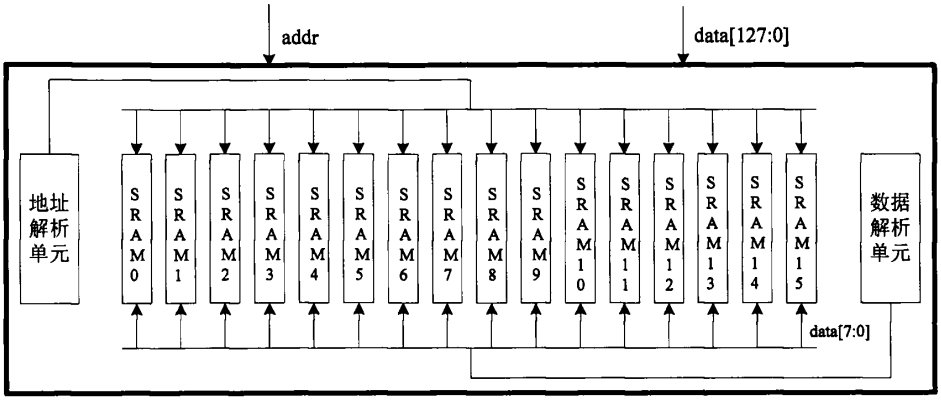

图 2-23：按字节寻址的 128 位宽存储单元

存储示例：

假设存储单元的写地址为 1，写数据为 *data*[127:0]，写入示例如图 2-24 所示。

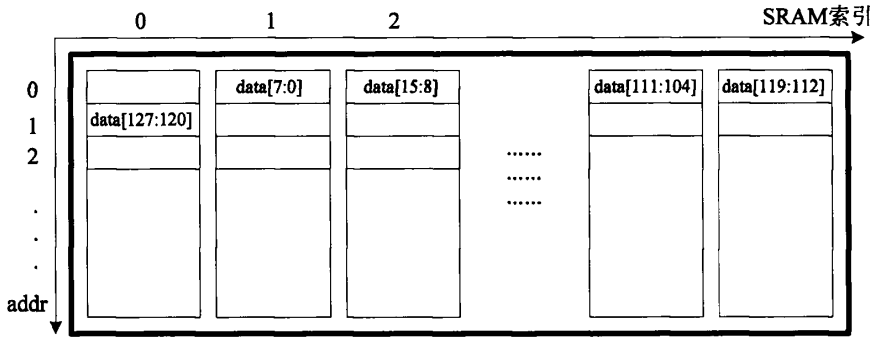

图 2-24：字节寻址大位宽存储单元访存示例

3. 列进行出存储单元

RxCP 解速率匹配加速引擎，在解子块交织模块中，包含一块“按列写入，按行读出”矩阵，如图 2-25 所示。传统 SRAM 难以实现矩阵交织，本文采用 32 块 8 比特 SRAM 进行拼接，加入存储控制逻辑，巧妙实现了矩阵交织。下面介绍列进行出存储单元的读写逻辑设计。

A. 写逻辑

假设某时刻向矩阵第 i 列 j 位置写入数据 $data[255:0]$，写入的 SRAM 索引用 m 表示。SRAM 索引计算如公式(2.15)所示， SRAM 片内地址计算如公式(2.16)所示。

$$m_{SRAM}=(i+j+k)\bmod 32 \qquad (k=0,1,2,...,31) \tag{2.15}$$

$$waddr_m=j+k \qquad (k=0,1,2,...,31) \tag{2.16}$$

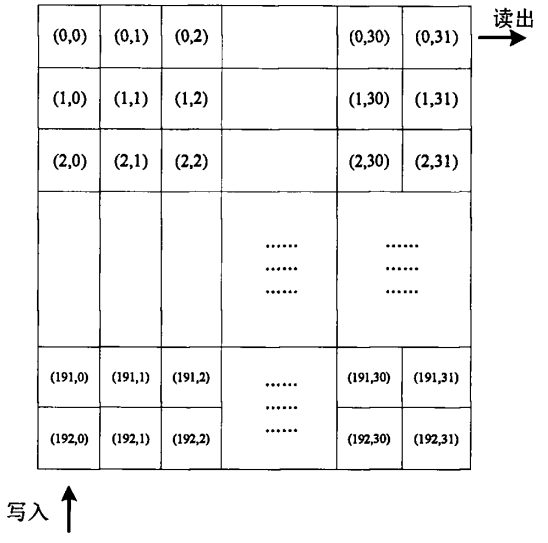

图 2-25：子块交织矩阵

B. 读逻辑

假设某时刻从矩阵第 i 行 j 位置读出 $data[255:0]$，访问的 SRAM 索引用 m 表示。则 SRAM 索引计算如公式(2.17)所示，对应 SRAM 片内地址计算如公式(2.18)所示。

$$m_{SRAM}=(i+j+k)\bmod 32 \qquad (k=0,1,2,...,31) \tag{2.17}$$

$$raddr_m=i \qquad (k=0,1,2,...,31) \tag{2.18}$$

存储示例：

以 8*8 交织矩阵为例，数据存储形式如图 2-26 所示。

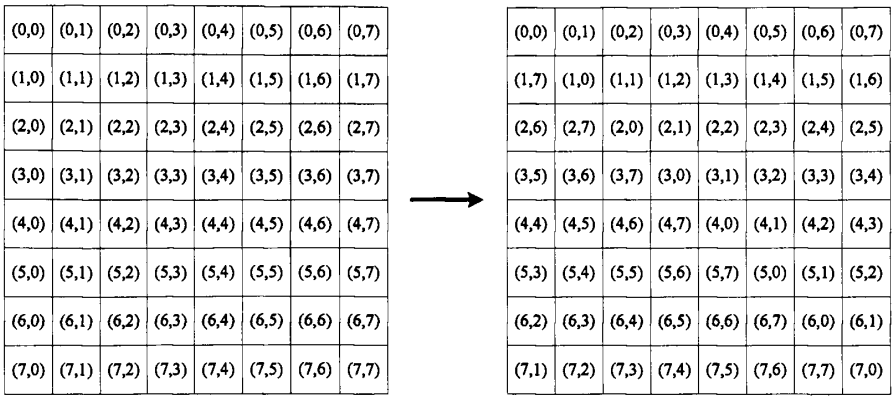

图 2-26：列进行出存储单元示例

2.4.3 协处理器完备性验证

随着集成电路规模不断增大，且流片费用昂贵，验证成为集成电路设计中一项非常重要的工作，约占芯片设计周期的 70%。本节设计了一种高效、自动化的验证平台，对协处理器进行了完备性验证。

1. 验证方法

本文中协处理器验证采用的是自底至上的验证方法，首先对协处理器内部加速引擎进行验证，然后对协处理器整体功能进行验证。在协处理器设计过程中，首先根据规格说明书，对各个模块，尤其是加速引擎模块和协处理器整体架构，进行设计说明，撰写设计文档。然后依据设计文档制定验证计划。验证计划中很重要的一项是列写验证功能点。以接收端协处理器 RxCP 为例，首先对验证功能点分类，然后对各个功能点根据属性不同进行细分，最终得出验证计划。RxCP 验证功能点如图 2-27 所示。

协处理器验证分为两大类：单模块验证和工作模式验证。单模块验证又分为接口模块和功能模块验证。其中，接口模块根据数据长度随机和数据基地址随机原则进行完备性验证；功能模块根据配置激励在约束范围内随机、数据激励随机进行单码块和多码块验证。单模块验证基础上，对协处理器工作模式进行验证。

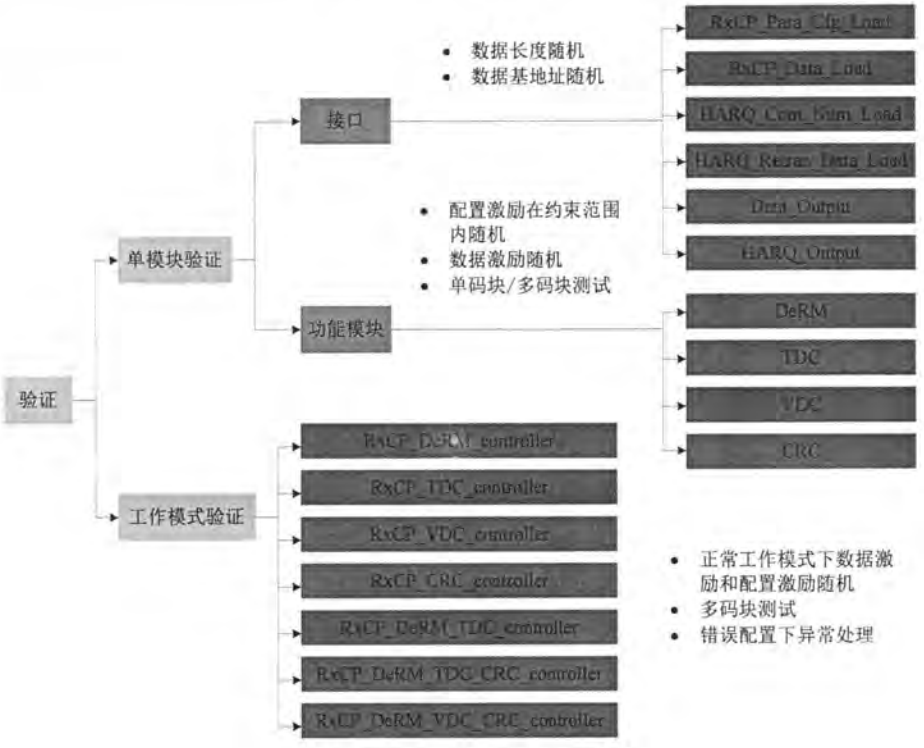

图 2-27：RxCP 验证功能点

2. 验证平台

基于黑盒测试和白盒测试相结合的思想，本文验证平台如图 2-28 所示。验证平台包括测试激励生成模块、Testbench、参考模型、功能检查模块和脚本文件。

测试激励生成模块：协处理器中加速引擎验证采用数据随机、配置参数受限的大规模仿真，经过迭代设计，使加速引擎鲁棒性更强；对协处理器整体验证，遍历通信协议要求的所有功能项，使验证尽可能完备、高效。加速引擎验证，测试激励生成模块可以由随机函数和约束条件产生；协处理器验证需借助实验室已有的通信链路平台，为 DUT（Design Under Test）提供测试激励。

Testbench：包括时钟信号、复位信号、DUT 及其相关的总线接口。由于该验证平台是基于文件接口的数据传输，所以 Testbench 中还需增加文件读写逻辑，如图 2-29 所示。

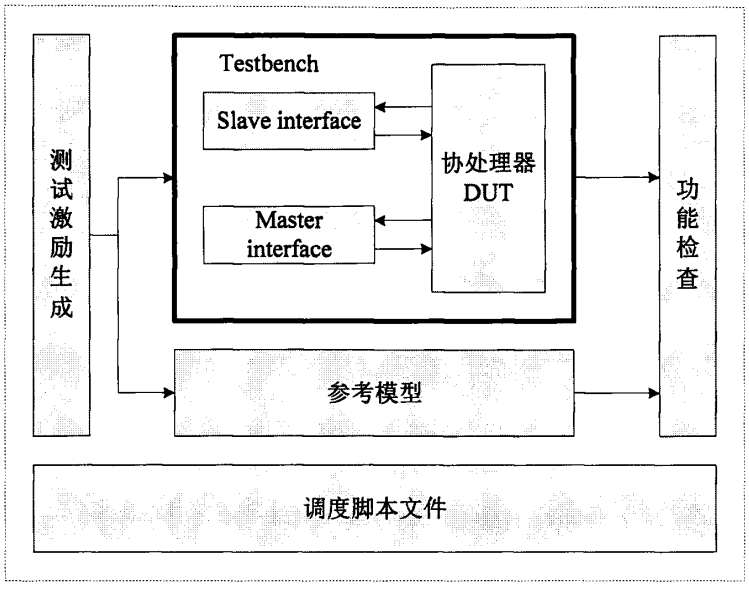

图 2-28：协处理器验证平台

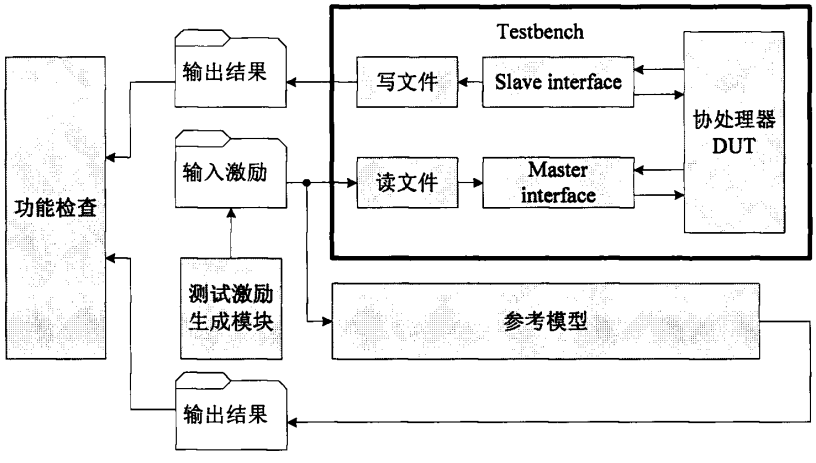

图 2-29：基于文件读写的验证平台

参考模型：验证采用的参考模型来自定点链路平台。RxCP 的验证结果与参考模型一致，可以认为其硬件设计达到了规格要求。为了 DUT 调试方便，通常会在参考模型中添加监视器。当验证系统处于 debug 模式下，监视器会将某些关键信息输出，用于程序员进行代码调试。

功能检查：可以在 DUT 输出结果时调用 task 任务来执行，也可以在 DUT 工作结束后，通过脚本文件读取输出结果进行比对。当 DUT 输出结果与参考模型不匹

配时，打印错误报告，提醒程序员进行调试。

控制脚本：调度仿真平台中各个模块，是用户与验证平台交互的窗口。本文中控制脚本主要有两类：Makefile 和批处理脚本。Makefile 脚本主要是对 DUT 和参考模型进行编译、执行；批处理 shell 脚本主要用于自动产生满足规格要求的测试向量。根据规格说明书要求的所有测试例，编写 Shell 脚本，可以使协处理器验证平台更高效、自动化，给验证工作带来了很大便利。

3. 验证过程

协处理器验证过程如图 2-30 所示。首先用 C 语言搭建激励生成模型，产生符合规格要求的配置激励和数据激励。配置激励主要包括工作模式配置、加速引擎工作参数等。Testbench 中数据接口完成后，编译、执行 DUT，产生输出结果。编译、执行参考模型代码，产生标准输出结果。读取两个文件中的输出结果，对数据进行比对。若全部匹配，则测试例通过；否则，复现该测试例，对参考模型和 DUT 加入监视器，对 DUT 进行调试。调试成功后，进行下一功能点测试，直到完成所有验证。

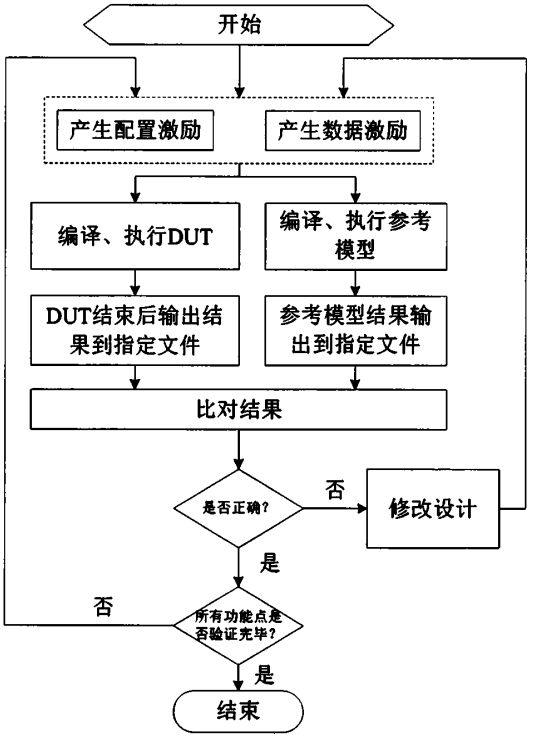

图 2-30：协处理器验证过程

4. 验证结果

本文对协处理器覆盖率报告进行了分析，以 viterbi 译码器覆盖率报告为例进行说明，如图 2-31 所示。对覆盖率报告中不足 100%的行覆盖、翻转覆盖、状态机覆盖、条件覆盖、分支覆盖等进行了分析；对覆盖率比较低的模块增加测试例进行完备性测试。

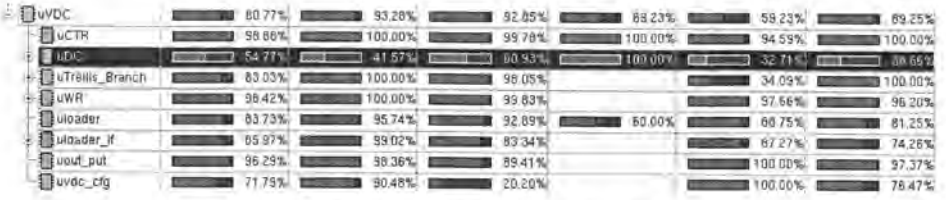

图 2-31：viterbi 译码器覆盖率报告

本文对协处理器各个模块进行完备性验证后，又对系统功能进行了充分验证。规格说明书中要求的所有测试向量，在自动化验证平台下，协处理器全部验证通过，可以认为该协处理器验证是充分的。

2.4.4 协处理器综合结果

本论文完成了基站侧 RxCP 协处理器 RTL 设计和实现，利用 Synopsys Design Compiler 进行综合，综合结果如表 2-5 所示。

表 2-5：RxCP 协处理器综合结果

类别	综合结果
工艺	28nm
主频	600MHz
面积	1.98mm^2
功耗	490.4081mW

2.5 小结

本章分析了当前主流通信协处理器架构，针对协处理器架构总线挂接设备多，互连网络功耗大，以及配置寄存器多元化问题，提出了一种面向通用通信基带处理

器的新型二维可配置协处理器架构。该架构通过将加速引擎分簇，并以特定工作模式重新编程加速引擎内部联结关系，使协处理器在灵活度和可靠性方面达到平衡。第一维配置为工作模式和协处理器公共参数配置，由主处理器发起，协处理器实时响应；第二维配置为加速引擎私有参数配置，由主处理器离线完成。通过功耗评估模型，该架构总线互连网络功耗仅为主流通信处理器架构的 1/3；通过对协处理器加速引擎分簇，一个标准数据帧处理，总线带宽占用比由 6.88%降到 2.05%。

本章还以基站侧无线通信接收端协处理器 RxCP 为例，介绍了协处理器工作机制、整体架构，以及协处理器内部控制逻辑和存储系统实现，并对协处理器进行了完备性验证，最后给出了 DC 综合结果。

无线通信系统中，传输数据存在衰减、失真、干扰和噪声等问题，导致接收到的数据出现一些错误[89]，通常采用信道编译码来进行检错、纠错。信道编译码器运算量比较大，一般采用加速引擎进行算法加速。本文接下来的章节对协处理器中信道译码加速引擎进行研究设计，分别包括：turbo 译码器提前退出迭代研究、高性能可配置 viterbi 译码器设计和实现、基于连续消除列表的 polar 译码器路径分裂和路径扩展优化。

第3章 基于二阶差分的 CRC 校验停止准则 turbo 译码器

3.1 引言

绪论中提到的 turbo 提前退出迭代 CE[35]、SCR[36]、SDR[37]、MSC[46]等准则，阈值通常由仿真得到，在译码纠错性能和译码延迟之间取折中，因此该阈值与 SNR 和码块交织长度强相关。 CRC 停止准则虽然对数据传输带宽有一定影响，但能有效改善迭代次数，提高 turbo 译码性能[44]。

虽然 CRC 停止准则对译码性能有改善，但也存在以下问题：当 SNR 比较低或者译码器遇到突发错误时，CRC 校验无法通过，译码器持续迭代计算，直到到达最大迭代次数停止工作。仿真表明，该情形下译码器输出结果与迭代少次的结果几乎相同，译码器多次迭代的时间开销并没有带来性能增益。

本文提出了一种二阶差分辅助的 CRC 校验退出迭代方法。首先计算两个子译码器在每次迭代中外信息相关值，称为一阶差分计算；然后计算相邻迭代中，译码器外信息相关值差值，称为二阶差分计算。译码器根据二阶差分值符号判断是否退出迭代。仿真表明，该方法在性能损失可接受范围内，能显著降低译码器在恶劣环境下的平均迭代次数。与常规 CRC 校验停止准则相比，turbo 译码器平均迭代次数下降约 20%。

3.2 二阶差分辅助的 CRC 校验停止准则

3.2.1 CRC 校验停止准则

Turbo 译码器中采用 CRC 校验停止准则，其 BLER 性能接近 Genie-aided 算法。Genie 算法在数据帧没有任何错误情况下停止迭代。本论文以码块长度 768 为例，仿真了 CRC 校验停止准则与 Genie 算法 BLER 性能，如图 3-1 所示。

从图 3-1 可以看出，CRC 校验提前退出迭代，对译码器 BLER 性能几乎没有影响。另一方面，我们也可以看到：在 SNR 比较低的情况下，误块率接近 100%。此情形下 CRC 停止准则迭代次数依然很高，迭代次数几乎等于预设最大迭代次数。接下来本文将解决 SNR 比较低或在突发错误下，译码器迭代次数比较高的问题。

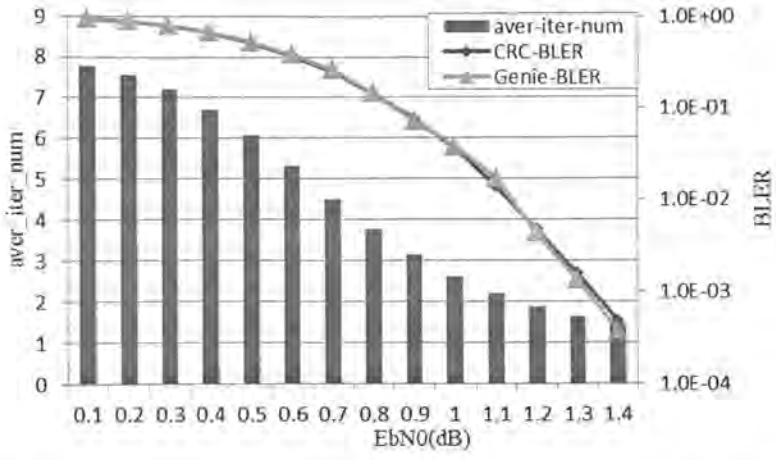

图 3-1：CRC 停止准则与 Genie 算法 BLER 性能对比及平均迭代次数

3.2.2 二阶差分辅助的 CRC 校验停止准则

在离散函数中，两个连续相邻值之差称为一阶差分。如果我们定义 $x(k)$，那么 $y(k)=x(k+1)-x(k)$ 表示一阶差分，$y(k)$ 指示离散函数的单调性。二阶差分定义为 $z(k)=y(k+1)-y(k)$，$z(k)$ 指示离散函数的变化速率。

对于 turbo 码，本文分析了译码器内部数据流变化规律。通过跟踪大量中间过程信号变化情况，得到以下发现：随着迭代次数增加，两个子译码器之间外信息差值越来越小。论文选取了某一测试例，两子译码器间对数似然比 LLR 相关值随着迭代计算的变化情况如表 3-1 所示。从表中可以看出，译码器正常工作时，随着迭代进行，两个子译码器间的 LLR 一阶差分值越来越小，呈现趋同现象。 二阶差分值总是为正数，表征两子译码器“接近”趋势。一般来说，当迭代次数越来越大时，一阶差分值越来越小，且差分下降速率越来越快，对应二阶差分符号为正，且越来越大。当译码器处于异常情况时，随着迭代进行，一阶差分值越来越大，二阶差分值变为负数，如表 3-2 所示。在第四次迭代中，LLR 一阶差分相关值比第三次迭代中的-1.674656 要大，表明译码器遇到某些突发错误无法纠正。此时二阶差分值为负值-0.062031。

当译码器遇到某些突发错误或不可纠正错误时，这种“错误”一定程度上会通过迭代进行传递，导致不必要的运算开销，甚至会使译码性能更糟糕。根据以上分析和研究，提出了一种二阶差分辅助的 CRC 校验停止准则，可以在信道环境恶劣情况下或译码器遇到不可纠正错误时，及时退出迭代，减少不必要的时间开销。

表 3-1：随着迭代进行子译码器间 LLR 相关值变化规律

迭代次数	一阶差分值	二阶差分值
0	-0.47484	--
1	-0.791836	0.316997
2	-0.981215	0.189379
3	-1.200226	0.219011
4	-1.375428	0.175202
5	-1.681867	0.306439
6	-1.958921	0.277054
7	-2.659726	0.700806

表 3-2：异常情况下子译码器间 LLR 相关值变化规律

迭代次数	一阶差分值	二阶差分值
0	-0.564011	--
1	-0.9835	0.419489
2	-1.244495	0.260995
3	-1.674656	0.430161
4	-1.612625	-0.062031

二阶差分辅助的 CRC 校验停止迭代过程如下：

（1） 初始化迭代计数变量 i=0。

（2） 在第 i 次迭代中，计算两个子译码器外信息相关值，并进行存储，此相关值即为一阶差分值。同时，对译码器进行 CRC 校验，如果 CRC 校验通过，停止迭代，否则进入步骤（3）。

（3） 在第 i+1 次迭代中，计算两个子译码器外信息相关值，并与第 i 次迭代中的相关值做二阶差分。如果二阶差分值符号为正，继续进行 CRC 校验，并将一阶差分值进行存储，否则停止迭代。如果 CRC 校验通过，停止迭代，否则 i=i+1，进入步骤（4）。

（4） 如果 i 小于预设最大迭代次数，进入步骤（3），否则停止迭代。

3.3 仿真实验和系统复杂度分析

随着通信系统对传输速率的要求不断增大，如何降低译码延迟，提高数据吞吐，并且使译码性能损失降低到最小，成为一个很重要的课题。

提高系统吞吐量最直接的方法是增加并行度[90]。把一个码块分成若干个子块，利用若干个子译码器并行进行译码，能大大提高系统吞吐量，是一种典型的以“空间换时间”的方法。对于子块内部的译码，为了降低硬件实现复杂度，减少存储面积，可以采用滑窗技术[91]。将一个子块分割成若干个滑窗，若干个滑窗串行进行译码，能大大缓解硬件资源压力。本文在 turbo 译码器并行化设计中，对子译码器并行度和滑窗长度进行了算法仿真，并对基于二阶差分的 turbo 译码器迭代次数和 BLER 性能进行了仿真和分析。

3.3.1 Turbo 子译码器算法仿真

仿真条件设置：在 AWGN 信道下，采用 BPSK 调制方式，最大迭代次数设置为 8 次，采用 Max_Log_MAP 算法进行译码。

1. 子译码器并行度确定

假设一个码块分成 N 个子块，除第一个子块的 α 和最后一个子块的 β 初始状态已知外，其余子块的 α、β 初始值设为等概。以后每次迭代中每个子块的 α 初始值来自上次迭代中上一子块的 α 初始值，每个子块的 β 初始值来自上次迭代中下一子块的 β 初始值。这种子块初始值的获得称为“置信传递”[92,93]。

在单纯只有分块情况下，评估子译码器并行度对系统性能的影响，仿真结果如图 3-2、图 3-3 所示。从图中可以看出，译码器并行度越高，系统的译码性能越差。同时，小码块由于长度较短，不能采用较高的并行度，否则性能损失比较严重。基于算法性能和硬件开销，本文中子译码器并行度设置如表 3-3 所示。

表 3-3：Turbo 子译码器并行度设置

码块范围	子译码器并行度
40~768	1
784~1536	2
1568~6144	4

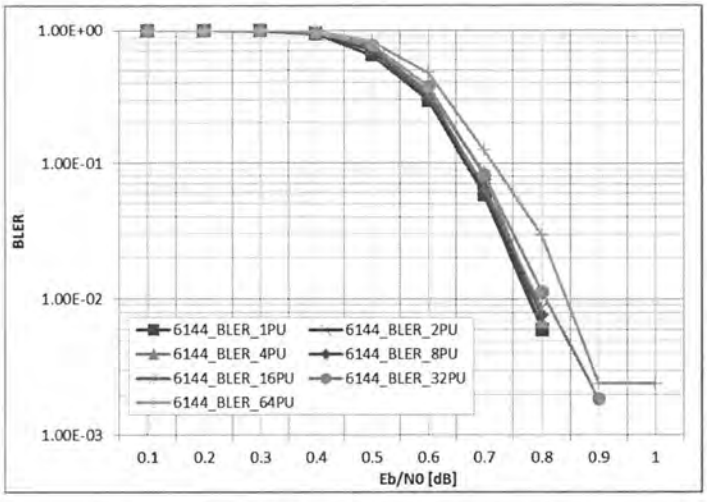

图 3-2：码块 6144 不同并行度下 BLER 性能

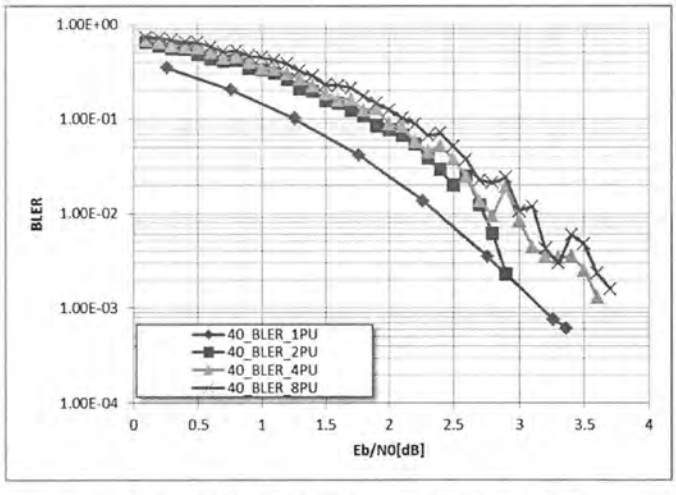

图 3-3：码块 40 不同并行度下 BLER 性能

2. 滑窗长度确定

子块内部以滑窗为单位进行译码，可以降低硬件资源消耗，尤其是缓解内存压力。每个滑窗（子块内第一个滑窗除外）的 α 初始值可以由上一个滑窗得到，每个滑窗（子块内最后一个滑窗除外）的 β 初始值由上一次迭代中下一个滑窗递推得到，这里采用的也是“置信传递”的思想。第一次迭代时每个滑窗的 β 初始值设为等概。

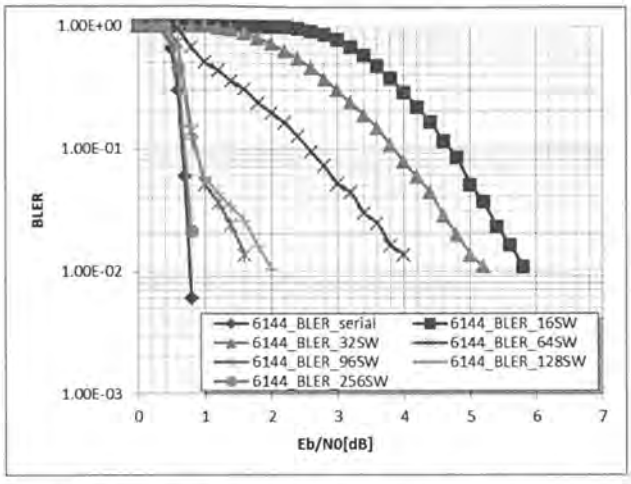

图 3-4：码块 6144 不同滑窗长度 BLER 性能

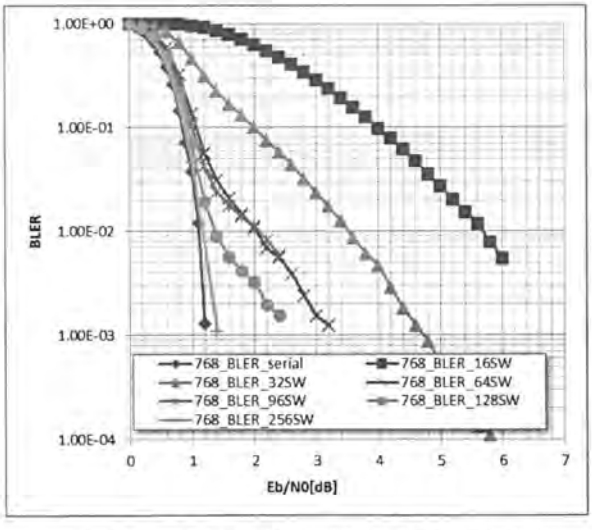

图 3-5：码块 768 不同滑窗长度 BLER 性能

本文仿真了不同滑窗长度对译码性能的影响，仿真结果如图 3-4、图 3-5 所示。从仿真结果可以看出：滑窗长度越小，turbo 译码性能越差；滑窗长度越大，turbo 译码性能越接近最优算法性能。滑窗长度的增加会带来较大的内存单元消耗，所以滑窗长度需在内存和译码性能方面取一个折中。由图 3-4、图 3-5 可以看出，滑窗长度为 256 时，译码性能几乎没有损失。

3.3.2 二阶差分辅助的 CRC 校验停止准则性能分析

一阶差分计算有多种方案，可以采用每个子译码器输出的软信息 LLR 计算相关值，也可以采用判决之后的硬比特计算相关值，还可以采用软信息和硬比特的混合计算求相关值。一定程度上，基于软信息的译码性能要比硬比特好很多，这是因为硬判决对数据精度有损失，属于“粗放型”译码。但是软信息计算对硬件资源要求比较高，时间开销比较大。考虑到 turbo 译码器中已采用 CRC 校验模块，而 CRC 校验是基于硬比特进行的。所以本文基于现有译码器结构，采用一种软信息和硬比特相结合的二阶差分停止准则。

软信息和硬比特计算相关值的过程相当于一个二路选通器，如图 3-6 所示。相比较定/浮点乘法器，硬件开销大大降低。

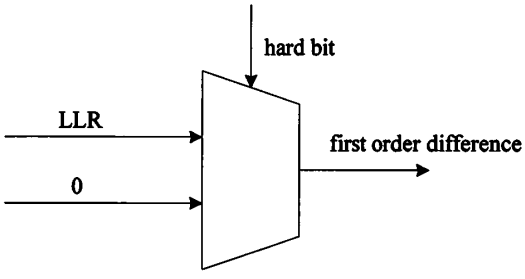

图 3-6：软信息和硬比特相关计算模块

为了简化，本文将软信息和硬比特结合的二阶差分 CRC 校验停止准则称为 CRC-SHA（CRC-Soft and Hard Aided），将单纯硬比特二阶差分 CRC 校验停止准则称为 CRC-HARD。由于单纯软信息二阶差分硬件开销比较大，这里不予讨论。

基于 CRC-SHA 的 turbo 译码器结构如图 3-7 所示，需额外增加一块存储器，存储当次迭代第一个子译码器产生的 LLR。第二个子译码器产生判决比特后，与第一个子译码器产生的 LLR 进行一阶差分值计算。完成整个码块的一阶差分值计算后，与上次迭代中存储的一阶差分值做减法，得到二阶差分值。根据二阶差分值符号判断译码器是否提前退出迭代。

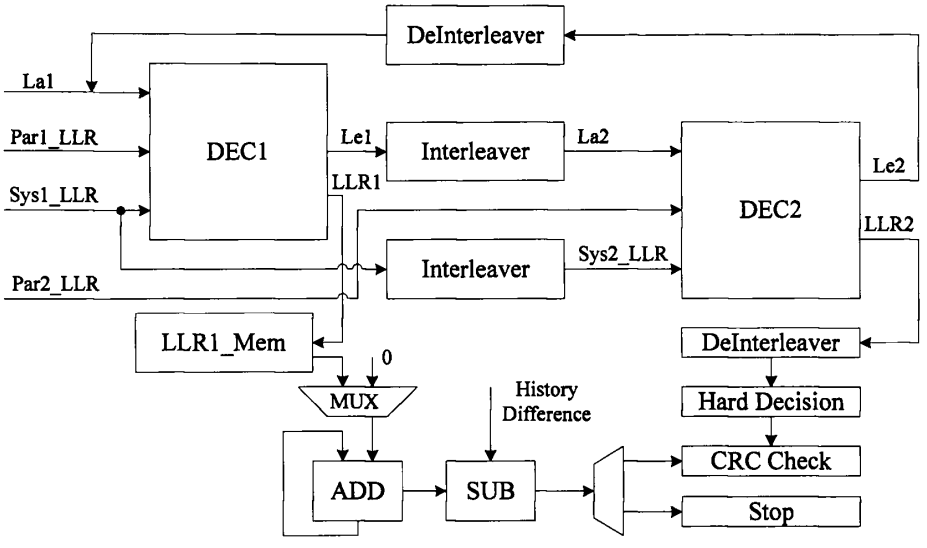

图 3-7：基于 CRC-SHA 的 turbo 译码器架构

当译码器处于异常工作状态时，译码器本身具有一定的“修复”能力得到正确结果。如果基于二阶差分的 CRC 校验停止准则检测到译码器异常，直接退出迭代，一定程度上会降低译码性能。基于这个问题，本文在译码器中加入错误检测变量，用于记录译码器每次迭代中异常出现次数。

本文仿真了异常出现次数为 1、2 时，译码器退出迭代后 BLER 性能和译码器平均迭代次数，仿真结果如图 3-8 和图 3-9 所示。由图 3-8 可以看出，单纯采用 CRC 校验停止准则，BLER 译码性能是最优的，但是译码器平均迭代次数最高，如图 3-9 所示。当检测到译码器出现两次异常，进而退出迭代，CRC-SHA 停止准则下 BLER 性能与单纯 CRC 校验停止准则相当，性能接近最优，且平均迭代次数下降约 20%。当检测到译码器出现一次异常，进而退出迭代，CRC-SHA 停止准则下 BLER 性能损失约 0.1dB，CRC-HARD 停止准则下 BLER 性能损失约 0.2dB，SNR 在-4.9dB~-4.4dB 区间，平均迭代次数约为 CRC 校验准则的 1/2。CRC-SHA 和 CRC-HARD 停止准则可以有效解决信道质量差或突发错误下译码器迭代次数较高的问题。考虑到 CRC-HARD 停止准则对译码性能损伤比较大，本文选取 CRC-SHA 停止准则作为基于二阶差分的 turbo 迭代译码器候选方案。

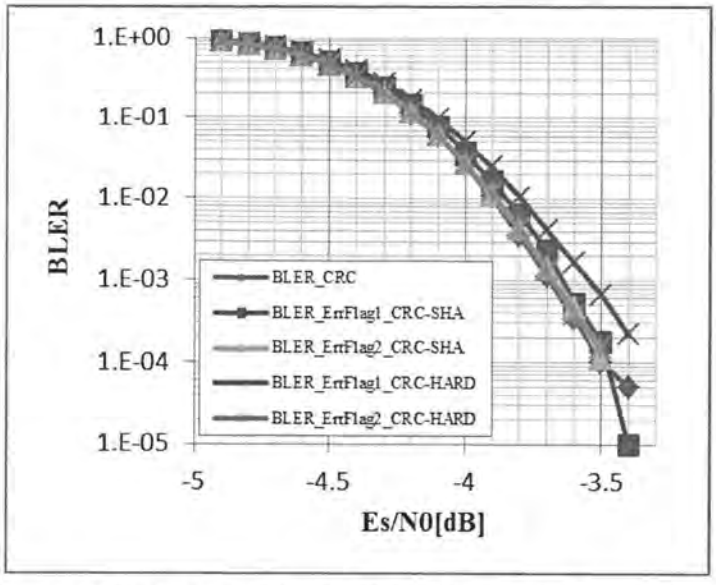

图 3-8：CRC-SHA 和 CRC-HARD 在不同异常次数下 BLER 性能对比

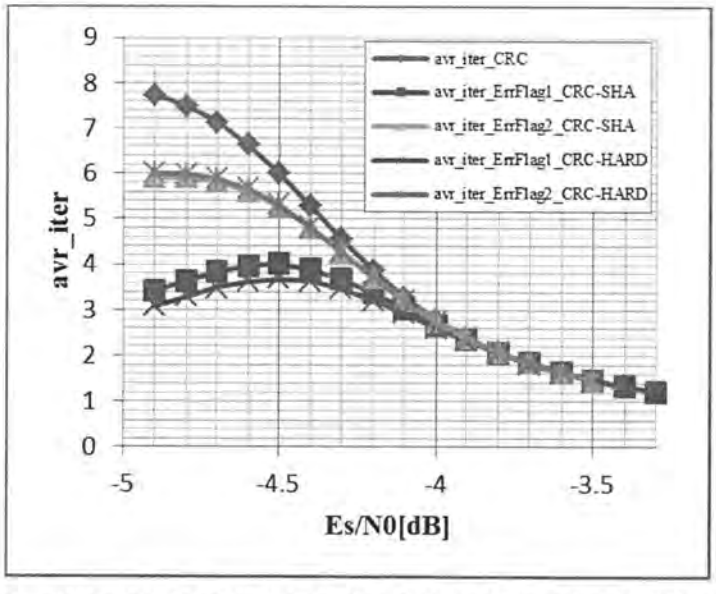

图 3-9：CRC-SHA 和 CRC-HARD 在不同异常次数下平均迭代次数对比

3.4 高性能 turbo 译码器实现

本文中的 turbo 译码器采用了分块译码和滑窗技术，并支持 Log_MAP 算法和

Max_Log_MAP 算法，数据峰值吞吐为 123Mbps@6144bit,600MHz，满足 LTE 物理层上行数据处理要求 75Mbps（64QAM）或 50Mbps（16QAM）。国际领先的半导体公司 TI 的 turbo 译码器峰值吞吐为 147Mbps@6144bit,600MHz，该译码器与国际领先水平相当。下面详细介绍高性能 turbo 译码器实现细节。

3.4.1 Turbo 译码器整体架构

Turbo 译码器整体架构如图 3-10 所示，包括四个子译码器、一个交织器、一个硬判决单元和若干内部缓存单元。

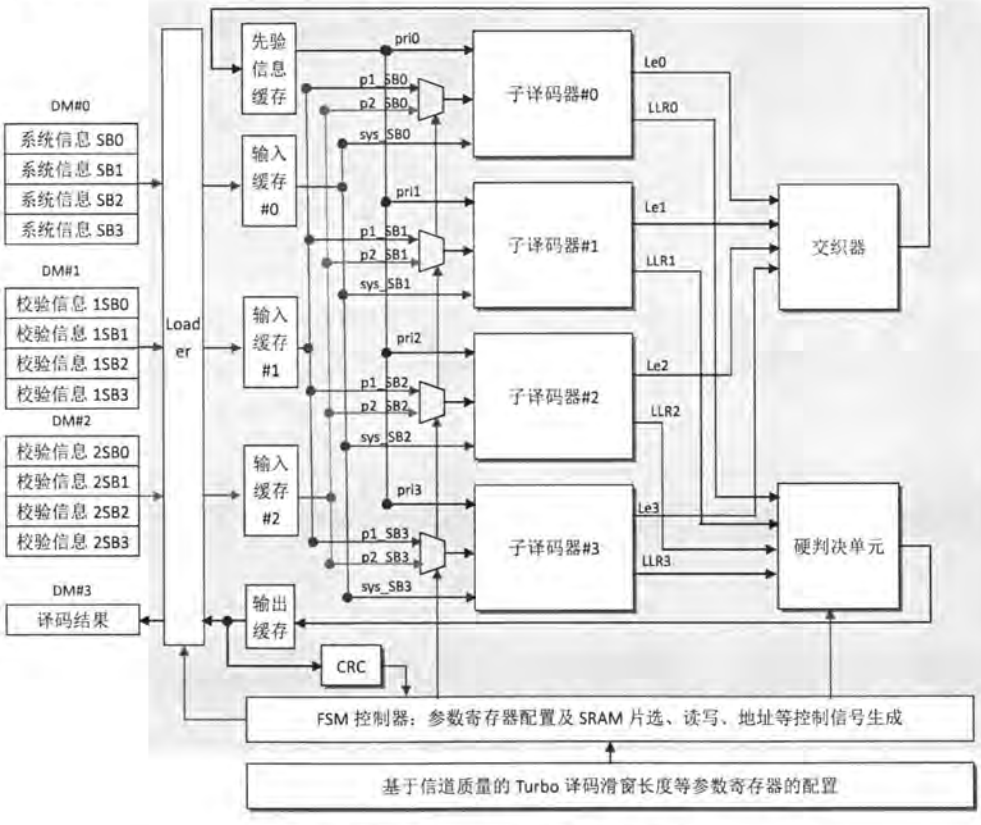

图 3-10：Turbo 译码器整体架构

Turbo 译码器工作流程如图 3-11 所示。其中，初始化工作包括：

1. 数据存储及译码器参数配置
   - 根据码块长度 K，确定子块分割数目 M 和子块大小 L，从而确定数据读写首地址。

- 根据信道质量确定滑窗长度 N_{sw}。

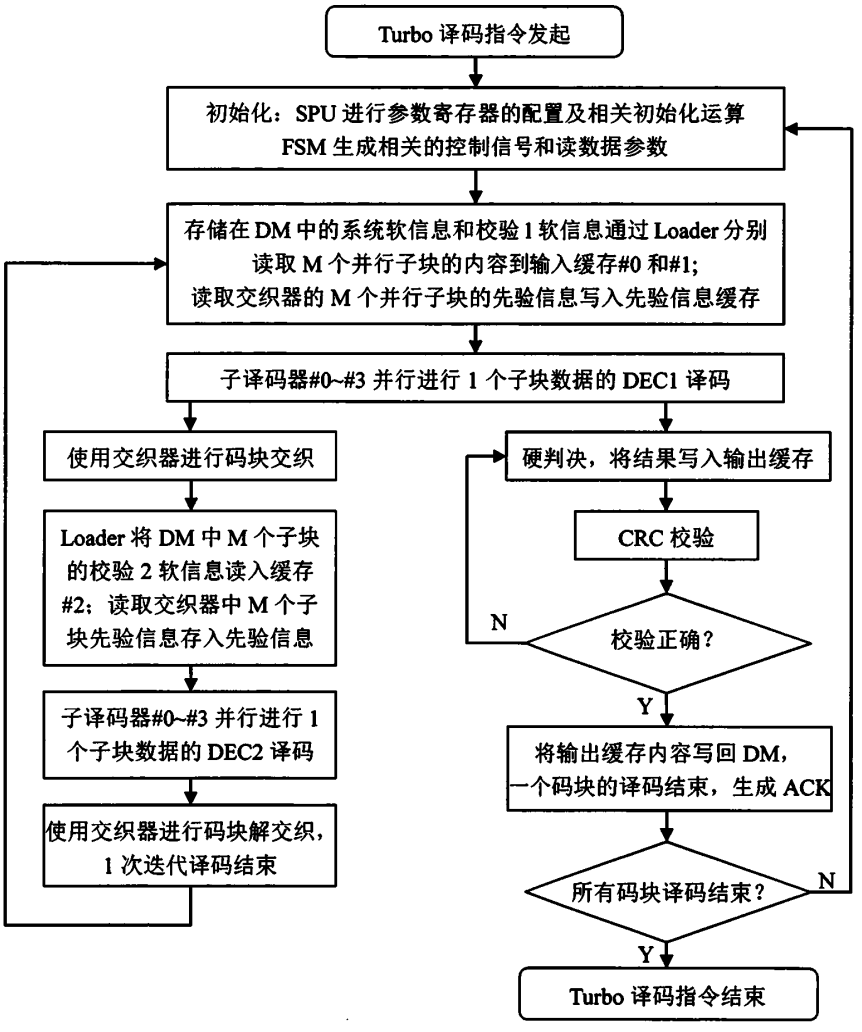

图 3-11：Turbo 译码器工作流程

2. 子译码器初始化

- 配置子译码器开关。
- 配置最大译码迭代次数。
- 配置迭代译码算法：Log_MAP 算法还是 Max_Log_MAP 算法。
- 配置 Turbo 译码运算单元中的 CRC 类型。
- 根据解速率匹配输出的尾比特软信息计算码块最后软比特的后向度量

β 的初始值。

- 对每个子块的边缘 α 和 β 进行初始化。
- 配置 LLR 计算单元所需的伸缩因子值。

3. 交织器初始化

- 初始化交织地址计算所需参数：f_1, f_2, e_1, e_2，并计算 $p(0)$、$g(0)$、$as(0)$和 $ts(0)$等交织所需参数。
- 初始化交织地址译码单元所需参数：addr_bank[M]。
- 解交织器内存储单元清零。

Turbo 译码器启动后，加载 M 个并行子块的系统信息和校验 1 信息到内部缓存#0 和内部缓存#1；加载交织器的 M 个并行子块的先验信息写入先验信息缓存区。子译码器#0~#3 并行工作，进行 M 个子块的前半次迭代译码。前半次迭代结束后，对子译码器输出的外信息进行交织运算，存入先验信息缓存区；加载 M 个子块的校验 2 信息到内部缓存#2。子译码器#0~#3 并行工作，进行 M 个子块的后半次迭代译码。后半次迭代结束后，对子译码器输出的外信息进行解交织运算，存入先验信息缓存区。然后开始第二次迭代，以此类推，直到达到最大迭代次数停止译码。其中，子译码器前半次迭代结束后，对子译码器输出的对数似然比 LLR 进行硬判决，将结果写入输出缓存。对硬判决结果进行 CRC 校验，若校验通过，则提前停止迭代，将译码结果输出；若校验不通过，则继续进行迭代译码。Turbo 译码器整体时空图如图 3-12 所示。

图 3-12：Turbo 译码器时空图

3.4.2 子译码器和交织器

本节对 turbo 译码器中关键模块：子译码器和交织器设计进行说明。

1. 子译码器

子译码器是 turbo 译码器中重要的运算单元，其整体架构如图 3-13 所示。包括分支度量γ计算单元、前向度量α计算单元、后向度量β计算单元和 LLR 计算单元，以及若干内部缓存单元。其中，分支度量γ、前向度量α、后向度量β和 LLR 计算逻辑分别如公式(3.1)(3.2)(3.3)和(3.4)所示。

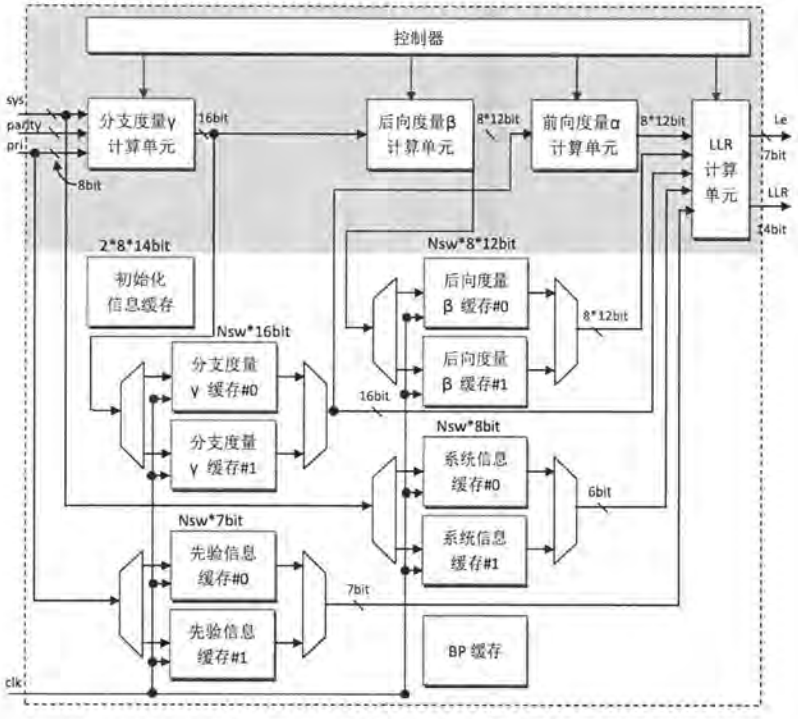

图 3-13：子译码器整体架构

$$\begin{aligned}\bar{\gamma}_{a,k}(m',m) &= \ln(\gamma_{a,k}(m',m)) \\ &\propto \left[\frac{1}{2}x_s(k)\left(L\left(x_s(k)\right)+L_c y_s(k)\right)+\frac{1}{2}L_c x_p(k)y_p(k)\right]\end{aligned} \tag{3.1}$$

$$\begin{aligned}\bar{\alpha}_k(m) &= \ln(\alpha_k(m)) \\ &= \ln(\sum_{m',a} e^{\bar{\alpha}_{k-1}(m')} + e^{\bar{\gamma}_{a,k}(m',m)})\end{aligned} \tag{3.2}$$

$$\begin{aligned}\bar{\beta}_{k-1}(m') &= \ln(\beta_{k-1}(m')) \\ &= \ln(\sum_{m,a} e^{\bar{\gamma}_{a,k}(m',m)} + e^{\bar{\beta}_k(m)})\end{aligned} \tag{3.3}$$

$$
\begin{aligned}
&L(\hat{d}(k)) \\
&= \ln \frac{\sum_{m',m} \alpha_{k-1}(m')\gamma_{1,k}(m',m)\beta_k(m)}{\sum_{m',m} \alpha_{k-1}(m')\gamma_{-1,k}(m',m)\beta_k(m)} \\
&= \ln \frac{\sum_{m',m} \alpha_{k-1}(m')\gamma_{1,k}^{(e)}(m',m)\beta_k(m)\exp\left(\frac{1}{2}\left(L\left(x_s(k)\right)+L_c y_s(k)\right)\right)}{\sum_{m',m} \alpha_{k-1}(m')\gamma_{-1,k}^{(e)}(m',m)\beta_k(m)\exp\left(-\frac{1}{2}\left(L\left(x_s(k)\right)+L_c y_s(k)\right)\right)} \\
&= L_c y_s(k) + L\left(x_s(k)\right) + \ln \frac{\sum_{m',m} \alpha_{k-1}(m')\gamma_{1,k}^{(e)}(m',m)\beta_k(m)}{\sum_{m',m} \alpha_{k-1}(m')\gamma_{-1,k}^{(e)}(m',m)\beta_k(m)}
\end{aligned}
\tag{3.4}
$$

子译码器进行迭代译码的工作流程如图 3-14 所示。其中，第一次迭代时各个滑窗（除第一个和最后一个滑窗）的初始值设为等概，且第一次迭代不输出 LLR；之后的迭代，滑窗的初始值通过置信传递，从上一次迭代时的边界值获取。子译码器工作时空图如图 3-15 所示。

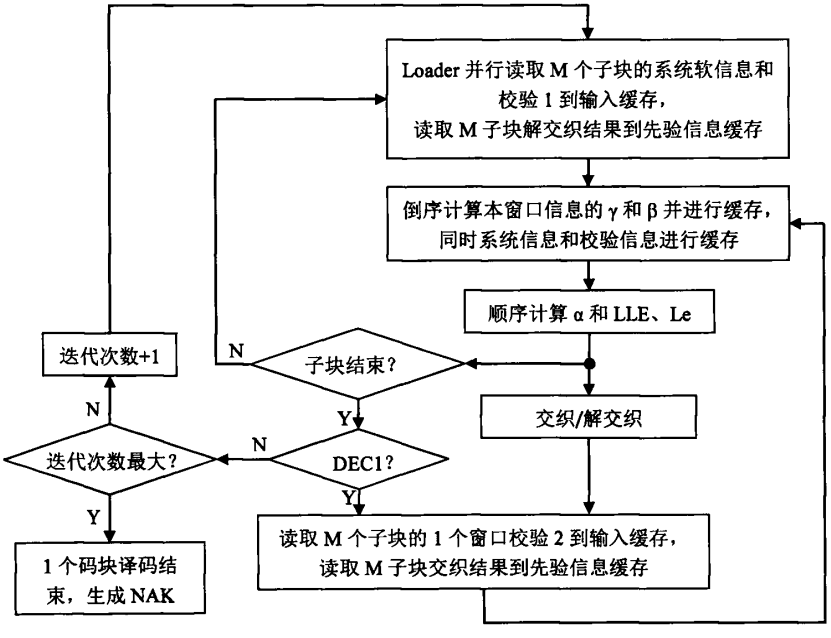

图 3-14：子译码器工作流程

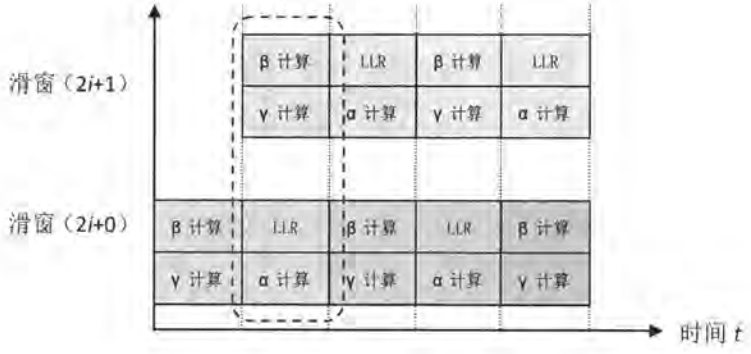

图 3-15：子译码器时空图

2. 交织器

交织器影响 turbo 码的距离特性，采用交织器后，整个码接近随机编码特性。对于一个子译码器中不可纠正的差错事件，在另一个子译码器中被分开，成为可纠正差错。交织器一方面提高了译码纠错性能，另一方面也增加了编译码延迟。

LTE 系统中的交织器具有如下特性：M 个子块的数据经交织后，仍会无冲突地分别处于 M 个不同的子块，且块内地址偏移相同。具体地，在进行交织操作时，地址生成单元生成读地址，通过将数据“顺序写入，乱序读出”实现交织；在进行解交织操作时，地址生成单元生成写地址，通过将数据“乱序写入，顺序读出”实现解交织。交织器结构如图 3-16 所示。

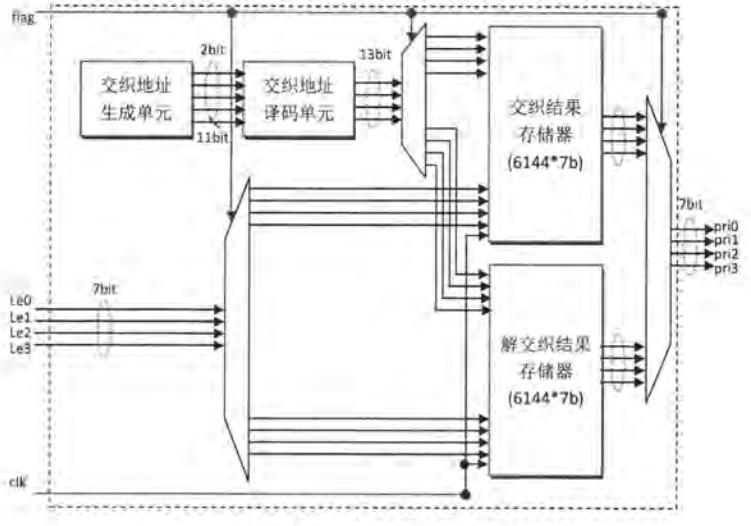

图 3-16：交织器结构图

3.4.3 Turbo 译码器性能评估

1. 数据吞吐

假设码块长度为 K，分割成的子块长度为 L，子块数目为 M，滑窗大小为 N_{sw}，则单个子译码器单次迭代的译码延时为：

- M=1 时，$L = K <= 768$，则译码延时为 $2*N_{sw}+L<= 1024$ *cycle*；
- M=2 时，$L=K/2$，$768<K<=1536$，则译码延时为 $2* N_{sw} +L <= 1024$ *cycle*；
- M=4 时，$L=K/4$，$1536<K<=3072$，则译码延时为 $2* N_{sw} +L <= 1024$ *cycle*；
- M=4 时，$L=K/4$，$3072<K<=6144$，则译码延时为 $2* N_{sw} +L <= 1792$ *cycle*；

假设最大迭代次数为 8 次，最坏情况下译码器迭代 7.5 次输出结果，其译码最大延迟为 2*1792*7.5=26880 *cycle*。

采用 VCS 工具对 RTL 代码进行验证。通过波形仿真，观测到码块 6144 迭代 7.5 次输出结果所需的时钟周期数为 29839。在 600MHz 时钟主频下，其实际吞吐为

$$\frac{6144*600}{29839}*10^6=123.54\text{Mbps} \tag{3.5}$$

2. 综合结果

本文采用 TSMC 标准单元库进行综合，综合结果如表 3-4 所示。

表 3-4：Turbo 译码器综合结果

类别	综合结果
工艺	28nm
主频	600MHz
面积	0.576mm^2
功耗	223.7277mW

3.5 小结

CRC 校验停止准则在信道环境比较差或译码器遇到不可纠正错误下，译码器迭代次数较高，译码性能却不理想。针对此问题，本章提出了一种基于二阶差分的 CRC 校验停止准则。通过实验仿真，在译码损失可接受范围内，能极大降低译码器迭代

次数。与常规 CRC 校验停止准则相比，平均迭代次数下降约 20%。

本章还对 turbo 子译码器并行度和滑窗长度进行了实验仿真，在有限硬件资源下，得出了不同码长最优并行度，并对 turbo 译码器进行了设计实现。首先描述了 turbo 译码器整体架构和工作过程。然后介绍了译码器中的关键运算模块：子译码器和交织器的实现细节。最后对译码器数据吞吐进行了评估，并给出了 DC 综合结果。

第4章 高性能可配置 viterbi 译码器

4.1 引言

Viterbi 译码器广泛应用于无线通信系统和多媒体设备中进行前向纠错，例如GSM、3G、4G、IEEE 802.16e/m、数字音频广播(Digital Audio Broadcasting，DAB)、数字视频广播(Digital Video Broadcasting，DVB)等。作为信道编码技术，viterbi 译码器还用于卫星通信系统中[94]。近年来，关于 viterbi 算法在物联网(Internet of Things，IoT)传感设备[95]和光通信载波相位估计[96]中的应用是一个研究热点。

随着通信技术和无线设备发展，单个设备中集成的功能越来越多。通信设备根据应用场景和用户需求，切换到不同的通信标准，成为通信设备发展趋势。因此，本文采用 ASIC 技术，设计一种可配置 viterbi 译码器，可支持多种通信标准，并具有一定扩展性。

Viterbi 译码器主要用于卷积编码。卷积码一般采用码率、约束长度、生成多项式来表征。卷积码应用场景示例如表 4-1 所示。

表 4-1：卷积码应用场景

应用场景	码率	约束长度
IS-95[55]	1/3	9
DVB[56]	1/2	7
UMTS[57]	1/2,1/3	9
MB-OFDM UWB[63]	1/3	7
802.11a/n/ac/ah[95]	1/2	7
GSM[97]	1/2	5
WCDMA	1/2	9
CDMA2000	1/4	9
LTE	1/3	7

本文根据常用通信标准，设计的 viterbi 译码器支持码率 1/2、1/3、1/4，约束长度 5~9，生成多项式可任意配置。此外，由于卷积码结尾方式有所不同，例如 LTE 系统中采用咬尾，其它系统大多为零结尾，本文中 viterbi 译码器支持两种结尾方式。

Viterbi 译码器在宽带和移动通信系统中要求的数据速率比较高，通常要达到 250Mbps 数据吞吐。在光学 M 阵列脉冲调制系统中，译码速率甚至达到 Gbps[67]。开发高性能并具有一定灵活性的 viterbi 译码器在软件定义无线电(Software Defined Radio，SDR)和认知无线电(Cognitive Radio，CR)中有很大需求[98]。

目前，关于 viterbi 译码器研究，大多是针对单一标准[55][63][99-101]。2008 年，Mark A 等[102]在 90nm CMOS 工艺下设计了一种可配置 viterbi 译码器，支持约束长度 5~9。该译码器在 3.8GHz 主频，32 状态下数据吞吐速率为 1.9Gbps，功耗为 358mW，面积为 1.9mm^2。其中，回溯单元和八个加比选单元所占面积为 0.53mm^2，动态漏电流功耗为 44mW。该译码器较高的数据吞吐是以高主频为代价的，随之带来的面积和功耗开销也比较大。2016 年，Khloud Mostafa 等[73]基于 Zynq-7000 FPGA，设计一种支持 Wi-Max、WLAN、3GPP2、GSM 和 LTE 系统的可配置 viterbi 译码器。该译码器采用基二算法，数据处理速度不高，最优性能下功耗为 78mW。同年，夏飞等[103]基于 FPGA 设计一种支持 GPRS、WiMAX、LTE、CDMA、3G 等通信标准的可配置 viterbi 译码器。该译码器采用基四算法，主频为 270.5MHz，峰值吞吐为 541Mbps。

本文设计的可配置 viterbi 译码器在 600MHz 主频下，峰值吞吐为 1.15Gbps。在 Synopsys 28nm 标准单元库下进行综合，其功耗为 46mW，面积为 0.211mm^2。半导体公司 TI 商用 DSP TMS320C6670 中 viterbi 译码器 VCP2，数据处理能力为 9.5Mbps@40bit,333MHz，本文中译码器数据处理能力为 32.173Mbps@40bit,333MHz，性能提升约 3.3 倍。

4.2 Viterbi 算法概述

Viterbi 算法是在无记忆噪声干扰下，对有限状态离散马尔科夫序列的最大后验概率估计[104]。无记忆离散信道的一般模型如图 4-1 所示，其中$c=(c_0,c_1,...,c_{K-1})$表示待编码序列，K为编码前数据长度，$y=(y_0,y_1,...,y_{L-1})$表示编码后序列，L表示编码后数据长度，$r=(r_0,r_1,...,r_{L-1})$表示经过无记忆信道后的接收序列，$u=(u_0,u_1,...,u_{K-1})$为译码后得到的估计序列。

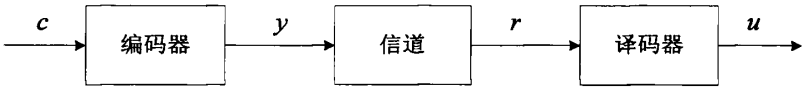

图 4-1：无记忆离散信道模型

对于无记忆离散信道，噪声对每一位发送码元的影响是独立不相关的，从而条件概率 $p(r_i|y_i)$ 就等于每个独立同分布接收码元条件概率的乘积，即长度为 L 的接收序列 r 的似然函数如公式(4.1)所示。

$$p(r|y)=\prod_{i=0}^{L-1}p(r_i|y_i) \tag{4.1}$$

为了简化似然函数计算复杂度，通常采用对数似然函数来表示，如公式(4.2)所示。

$$p(r|y)=\sum_{i=0}^{L-1}\log(p(r_i|y_i)) \tag{4.2}$$

为了简化对数函数求和运算，定义码元度量 $M(r_i|y_i)$，如公式(4.3)所示。

$$M(r_i|y_i)=a[\log(p(r_i|y_i))+b] \tag{4.3}$$

其中，常数 a 和 b 的选取应尽量使得码元度量值尽可能接近某个小的正整数。

Viterbi 算法通常采用欧式距离来计算码元度量。对于高斯加性白噪声信道 $n\sim N(0,N_0/2)$，似然函数可以用高斯概率密度函数来表示，如公式(4.4)所示。其中，E_b 为接收码元平均功率，接收码元服从均值为 $\sqrt{E_b}y_i^{(J)}$、方差为 $N_0/2$ 的高斯分布。

$$p(r_i^{(J)}|y_i^{(J)})=\frac{1}{\sqrt{\pi N_0}}\exp\{-\frac{(r_i^{(J)}-\sqrt{E_b}y_i^{(J)})^2}{N_0}\} \tag{4.4}$$

假设发送序列长度为 L，对数似然函数表示为公式(4.5)，

$$\begin{aligned}
p(r|y)&=\sum_{i=0}^{L-1}\sum_{j=0}^{n_0-1}\log p(r_i^{(J)}|y_i^{(J)})\\
&=\sum_{i=0}^{L-1}\sum_{j=0}^{n_0-1}[-\frac{(r_i^{(J)}-\sqrt{E_b}y_i^{(J)})^2}{N_0}-\frac{1}{2}\log\pi N_0]\\
&=-\frac{1}{N_0}\sum_{i=0}^{L-1}\sum_{j=0}^{n_0-1}[(r_i^{(J)}-\sqrt{E_b}y_i^{(J)})^2]-\frac{LN_0}{2}\log\pi N_0\\
&=-\frac{1}{N_0}\sum_{i=0}^{L-1}\sum_{j=0}^{n_0-1}[r_i^{(J)2}-2\sqrt{E_b}r_i^{(J)}y_i^{(J)}+E_b y_i^{(J)2}]-\frac{LN_0}{2}\log\pi N_0\\
&=C_1\sum_{i=0}^{L-1}\sum_{j=0}^{n_0-1}r_i^{(J)}y_i^{(J)}+C_2
\end{aligned} \tag{4.5}$$

式中，$y_i^{(j)2}=1$，C_1和C_2是与y无关的项。从而可以定义码元度量如公式(4.6)所示。

$$M(r_i^{(j)}|y_i^{(j)})=r_i^{(j)}y_i^{(j)} \tag{4.6}$$

基于相关度量的软判决算法可以看做是对欧式距离软判决算法的化简。欧式距离码元度量如公式(4.7)所示，其中n_0是码率的倒数。

$$M(r_i|y_i)=\sum_{j=0}^{n_0-1}(r_i^j-y_i^j)^2 \tag{4.7}$$

离散时间有限状态马尔科夫过程如图 4-2 所示，以四状态移位寄存器为例来说明，如图 4-3 所示。

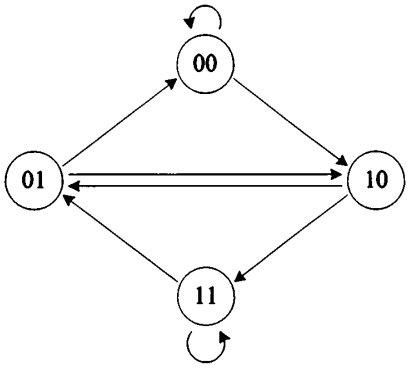

图 4-2：四状态移位寄存器状态转移图

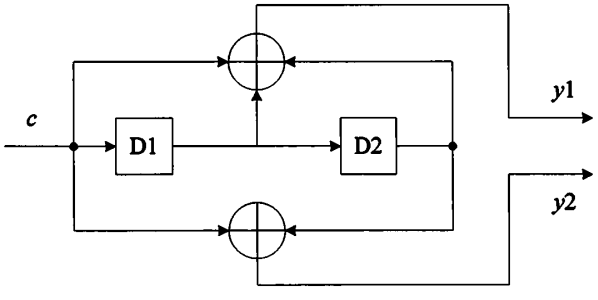

图 4-3：四状态卷积编码器

以基于相关度量的 viterbi 译码算法为例来说明译码过程，用$s_{k,t}$表示编码网格图中第t时刻的状态s_k。

（1） t=0 时刻：给网格图中每个状态指定一个度量值 $V(s_{k,t})$ 并初始化。若编码器初始时刻状态未知，则每个状态等概赋予度量值。若初始时刻状态已知，则已知状态度量值赋值为 0，其余状态赋值为无限大。

（2） $t \to t$+1 时刻：计算在 t+1 时刻到达 s_k 状态的所有路径的部分路径度量值。首先计算 t+1 时刻的分支度量 $M(r_{t+1}|y_{t+1}) = \sum_{j=0}^{n_0} M(r_{t+1}^{(j)}|y_{t+1}^{(j)})$ ，通过计算 $r_{t+1}^{(j)}$ 和 $y_{t+1}^{(j)}$ 的相关函数 $\sum_{j=0}^{n_0} r_{t+1}^{(j)} y_{t+1}^{(j)}$ 来完成；然后计算 t+1 时刻的部分路径度量如公式(4.8)所示，可以通过 $V(S_{k,t}) + M(r_t|y_t)$ 来实现。

$$M^{t+1}(r_{t+1}|y_{t+}) = \sum_{t=0}^{t+1} \sum_{j=0}^{n_0} M(r_t^{(j)}|y_t^{(j)}) \tag{4.8}$$

（3） 将 $V(S_{k,t})$ 设置为 t 时刻到达 s_k 状态的最好部分路径度量。如果有多个最好部分路径度量，可以选择其中任意一个。最好部分路径度量一般选择欧式距离或者汉明距离最小的部分路径度量。

（4） 存储最好的部分路径度量及其相应的幸存路径和状态路径。

（5） 若 $t < L$，返回（2）；否则读出幸存路径作为译码输出。

综上，viterbi 算法是在网格图中进行递归计算，不断更新状态度量值，最后选取状态度量值最好的幸存路径作为译码输出。

通常，viterbi 算法采用寄存器交换法（Register Exchange, RE）[106-109]和回溯法（Trace Back，TB）实现译码[110-112]。寄存器交换法一般应用于状态数比较少的译码器中，这是由于寄存器交换法对内存中每一比特会反复读写，随状态数增加，功耗线性增加。状态数较多的译码器通常采用回溯法，但回溯法译码延迟比较大。下面具体介绍寄存器交换法和回溯法译码。

4.2.1 寄存器交换法

寄存器交换法，顾名思义是在源状态与目的状态之间一系列的“寄存器交换”操作，以图 4-3 所示（2,1,2）卷积编码器为例来说明寄存器交换法的译码过程。

假设发送序列 C 为全 0，接收序列 $r = [10,00,01,00,00,00,00,...]$，译码器初始状态为 0。网格图[105]递归计算过程如图 4-4 所示。节点表示给定时间编码器所处状态，分支表示当输入为“0”或“1”时由当前状态转移到下一时刻某个新状态。其中，实线表示当输入为“0”时转移分支，虚线表示当输入为“1”时转移分支。表 4-2 以汉明距离为例详细说明了网格图状态转移及幸存路径更新过程。

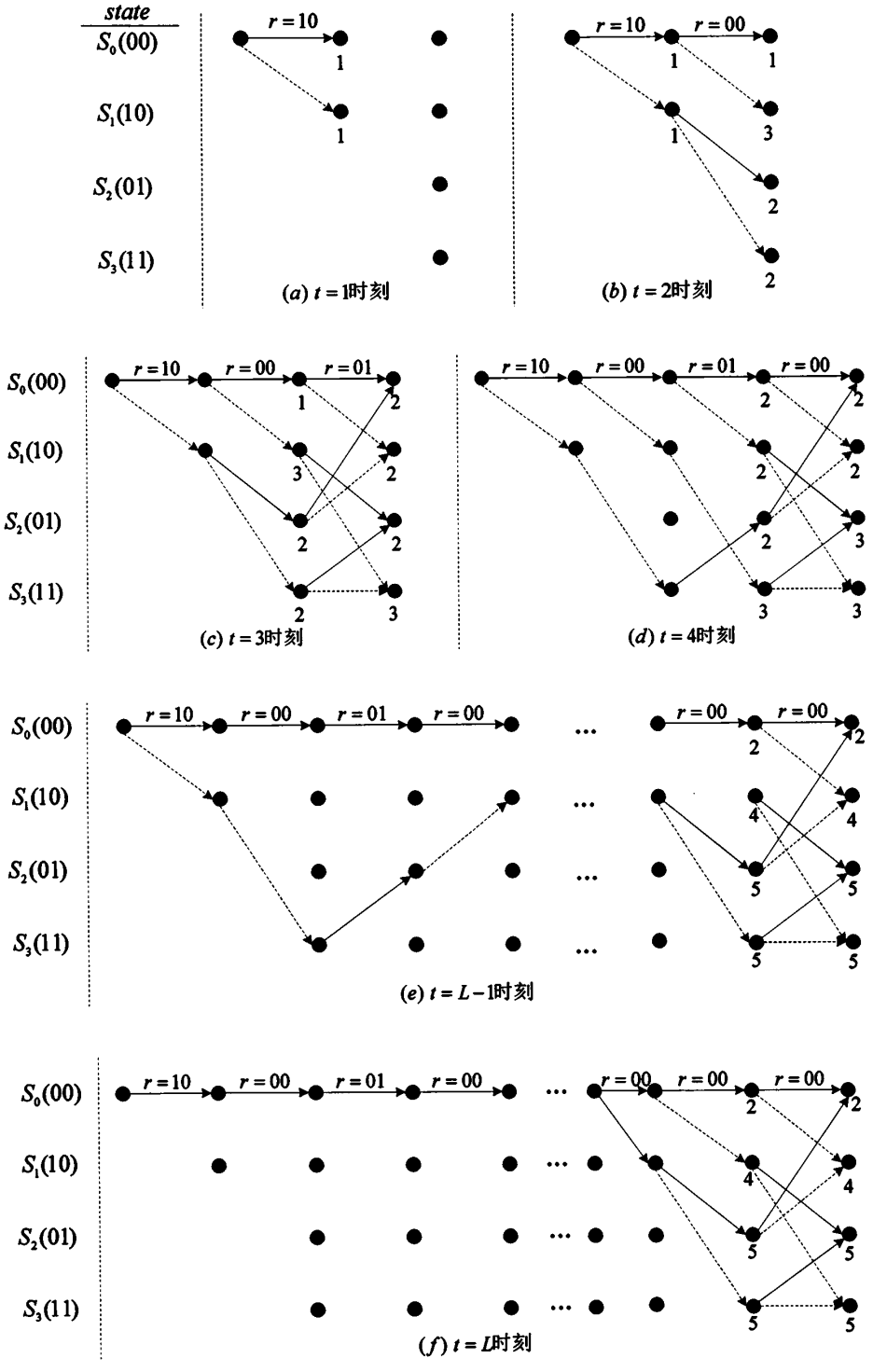

图 4-4：Viterbi 网格图递归计算过程

表 4-2：网格图状态转移及幸存路径更新过程

时刻	到达状态	起始状态	幸存路径	历史累加距离	转移分支	分支距离	累加距离	距离更新	幸存路径更新
t=1	S_0	S_0		0	00	1	1	1	0
	S_1	S_0		0	11	1	1	1	1
t=2	S_0	S_0	0	1	00	0	1	1	00
	S_1				11	2	3	3	01
	S_2	S_1	1	1	10	1	2	2	10
	S_3				01	1	2	2	11
t=3	S_0	S_0	00	1	00	1	2	2	000
		S_2	10	2	11	1	3		
	S_1	S_0	00	1	11	1	2	2	001
		S_2	10	2	00	1	3		
	S_2	S_1	01	3	10	2	5	2	110
		S_3	11	2	01	0	2		
	S_3	S_1	01	3	01	0	3	3	011
		S_3	11	2	10	2	4		
t=4	S_0	S_0	000	2	00	0	2	2	0000
		S_2	110	2	11	2	4		
	S_1	S_0	000	2	11	2	4	2	1101
		S_2	110	2	00	0	2		
	S_2	S_1	001	2	10	1	3	3	0010
		S_3	011	3	01	1	4		
	S_3	S_1	001	2	01	1	3	3	0011
		S_3	011	3	10	1	4		
…									
t=L	S_0	S_0	00000000	2	00	0	2	2	000000000
		S_2	00000010	5	11	2	7		
	S_1	S_0	00000000	2	11	2	4	4	000000001
		S_2	00000010	5	00	0	5		
	S_2	S_1	00000001	4	10	1	5	5	000000010
		S_3	00000011	5	01	1	6		
	S_3	S_1	00000001	4	01	1	5	5	000000011
		S_3	00000011	5	10	1	6		

寄存器交换情况如图 4-5 所示。每个译码时刻，寄存器中的信息需要刷新一遍，且需要一份寄存器副本，用来进行寄存器交换。寄存器数目与译码器状态数目相等，寄存器长度与待译码数据长度相关。译码器状态数目 n 计算如公式(4.9)所示，其中 K 为编码器约束长度。寄存器刷新产生的功耗比较大，所以当编码器约束长度比较大时，不适合采用寄存器交换法进行译码。

$$n = 2^K \tag{4.9}$$

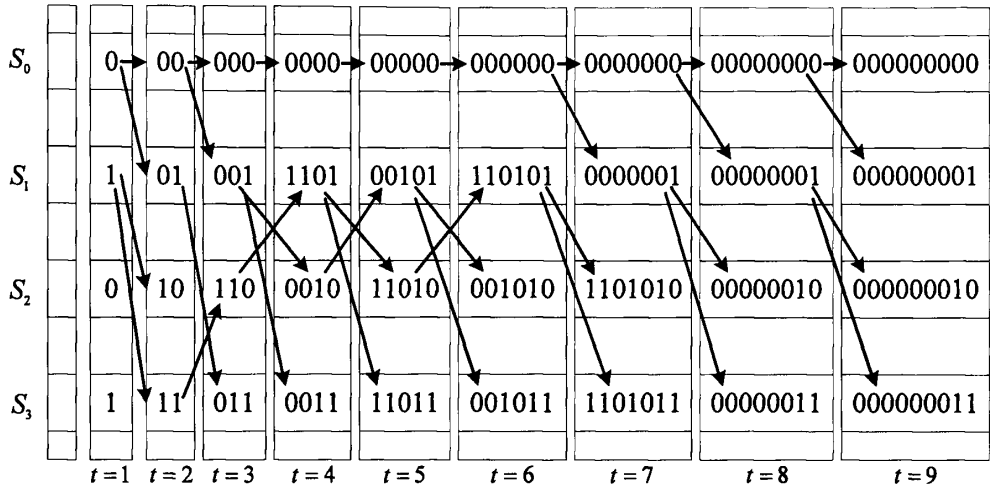

图 4-5：寄存器交换

4.2.2 后向回溯法

寄存器交换法在递归计算过程中，需要存储每个状态的幸存路径，且每个时刻各状态得到的幸存路径均是最优幸存路径，最后通过比较各个状态的路径度量值，从中选出路径度量值最小的状态对应的幸存路径作为译码比特。后向回溯法也需要存储每个状态的幸存路径，但是每个状态的幸存路径更新无需寄存器交换，只是将每个时刻的幸存比特依次添加在该状态的幸存路径后面，最后通过比较各个状态的路径度量值，从中选出路径度量值最小的状态开始后向回溯。

后向回溯过程如下：假设t时刻待回溯状态s_t对应的幸存比特为d_t，则$t-1$时刻的状态s_{t-1}如公式(4.10)所示；得到$t-1$时刻的状态s_{t-1}后，再利用$t-1$时刻的状态s_{t-1}对应的幸存比特继续向前回溯，直到到达 0 时刻停止回溯。回溯过程中各个时刻回溯状态的最高比特位即为译码比特。

$$s_{t-1} = \{s_t << 1, d_t\} \tag{4.10}$$

后向回溯法没有频繁的寄存器刷新，但是前向递归计算结束后，不能像寄存器交换法那样直接得到译码序列，需再进行一遍后向回溯，所以其译码延迟是寄存器交换法的二倍，通常在译码器状态数目比较多的情况下使用。

4.3 高性能可配置 viterbi 译码器算法设计与仿真

本文 viterbi 译码器采用基四算法[123,124]。相比基二算法，基四算法在一个时钟周期内可以完成两个时刻网格图递归计算和回溯译码，其数据处理速度是基二算法的两倍。一般来说，“时间”和“空间”难以两全，基四算法获得的“时间”增益也是以“空间”为代价的。表 4-3 统计了 viterbi 译码器在基二和基四算法下“时间”和“空间”开销。由统计结果可以看出，基四算法处理速度约是基二算法的两倍，运算器开销也约是基二算法的两倍。高性能 viterbi 译码器设计，是基于当前通信技术的发展和对数据速率的要求。未来的物联网时代对实时性要求很高，数据传输用户体验速率为“Gbps”。在面积和功耗满足要求的情况下，提高数据处理速度为 viterbi 译码器在高容量网络通信应用中提供了更多可能性。

表 4-3：Viterbi 译码器基二/基四算法比较

功能部件\类别	运算器数量		处理时间	
	基二	基四	基二	基四
分支度量计算单元	加法器（M）	加法器（2M）	2T	T
路径度量更新单元	加比选部件（M）	加比选部件（2M）	2T	T
后向回溯单元	移位寄存器（1）	移位寄存器（1）	2T	T

本文采用一种基于滑窗流水的后向回溯法，克服了寄存器交换法的大功耗和传统后向回溯法的大延迟，有很出色的译码和吞吐性能。接下来详细介绍该算法，并给出仿真结果。

4.3.1 基于滑窗流水的后向回溯法

传统后向回溯法译码延迟比较大，如何对此问题进行优化是高性能 viterbi 译码器设计不可回避的问题。

通过大量实验仿真，发现后向回溯具有这样一条规律：某一时刻所有状态回溯 $X \geq X'$ 时刻后，它们均到达同一状态。这条规律通常被称为数据一致性[115]。

根据数据一致性规律，在初始时刻，对每个状态进行编号{0,1,2...}。网格图前向递归计算过程中，记录每个时刻的更新状态来自前面时刻哪个状态，并继承前面时刻对应状态存储的状态编号，作为当前更新状态的状态编号，如图 4-6 所示。依次递归，经过 X' 时刻后，所有更新状态存储的状态编号均相同，即得到了 $t-X'$ 时

刻的后向回溯起始状态。$t+1$时刻继续前向递归计算，同时可以开始$t-X'$时刻的后向回溯[117,118]。X'通常称为最小分割长度。最小分割长度一般通过经验得来，通常设置为约束长度的5~6倍[116]。

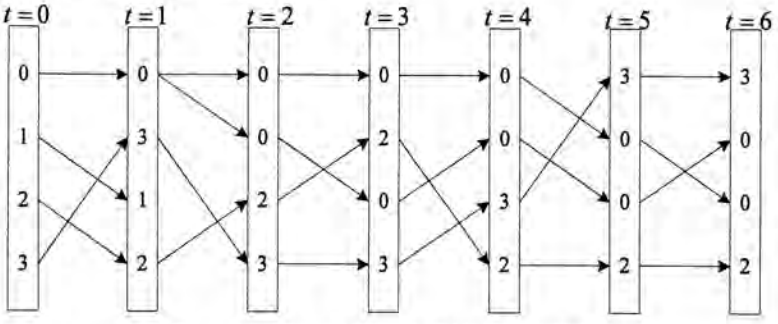

图 4-6：网格图递归计算状态标号更新

结合数据一致性规律和后向回溯算法，可以将待译码数据分滑窗进行流水处理。这里，网格图前向递归计算产生幸存路径过程称为SPG（Survivor Path Generation），后向回溯过程称为TB（Trace Back）。实际上，译码之初还有数据加载过程，称为LD（Load）。译码器工作时序如图 4-7 所示。从图中可以看出，SPG 和 TB 同时进行，处理的数据间隔一个滑窗。SPG 对幸存路径存储单元进行写操作，TB 对幸存路径存储单元进行读操作。为了避免存储器读写冲突，幸存路径存储单元分为三块bank，每块bank深度为滑窗长度L。幸存路径存储单元读写时序如图 4-8 所示。

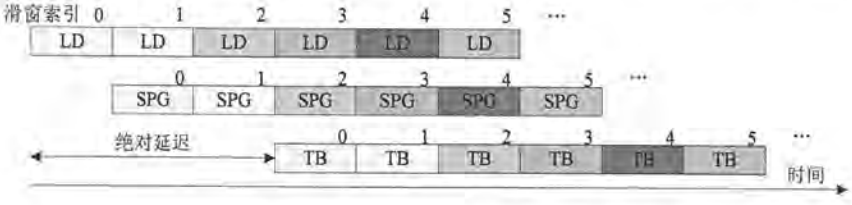

图 4-7：Viterbi 译码器工作时序

图 4-8：幸存路径存储单元读写时序

4.3.2 可配置性实现

本文中 viterbi 译码器支持的配置形式如表 4-4 所示。下面详细介绍 viterbi 译码器对各配置项的实现思路。

表 4-4：Viterbi 译码器配置形式

约束长度	码率			结尾方式	
	1/2	1/3	1/4	零结尾	咬尾
5	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓
7	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓
9	✓	✓	✓	✓	✓

卷积码一般用(n_0,k_0,m)来表示，其中，k_0表示每一时刻送入编码器的输入码元数，n_0表示相应的编码器输出码元数，m为编码器中移位寄存器个数，同时也表示输入码元在编码器中存储时间单位数。卷积码的输出实际上是k_0个输入码元与编码寄存器中存储的m个信息元线性组合的结果，因此，这样的卷积码又称为线性卷积码[119]。除非特别说明，本文只讨论线性卷积码。

1. 约束长度

通常，viterbi 译码器复杂度随约束长度的增加指数增长，实际应用中约束长度一般不会大于 15[113]（3~9 应用比较普遍[114]）。约束长度为编码器中移位寄存器个数m加 1，表示编码过程中互相约束的码段数。约束长度在可配置译码器中主要影响译码器状态数目。状态数目对译码器的影响主要体现在两个方面：存储面积和执行时间。

存储面积。Viterbi 译码器需要存储滑窗内所有状态的幸存路径，用于回溯译码。所以，约束长度比较大时，开辟的存储面积也相应增加。本文中支持的约束长度为 5~9，对应的译码器状态数目分别为 16~256。假设滑窗长度为L，则幸存路径存储单元为$256*L$比特。

执行时间。假设 viterbi 译码器并行度为N，当译码器状态数目较小时，译码器

可以在一个时钟周期内处理完所有状态更新。当译码器状态数目 n 与译码器并行度具有公式(4.11)所示关系时，其中 $k=\{2,3,4\}$，此时，译码器状态更新需要 k 个时钟周期才能完成。译码器处理一个滑窗的数据所需时间与状态数目线性相关。

$$n=k*N \tag{4.11}$$

2. 生成多项式

生成多项式表征了编码器输出码元与输入码元之间的线性关系。生成多项式的物理意义可以在网格图中用网格分支来表示，如图 4-9 所示。其中，网格图状态转移示例如图 4-10 所示。网格分支表示在 $t\rightarrow t+1$ 时刻，当输入码元为“0”（或者“1”）时，对应的输出码元。网格图递归计算过程中，接收序列与网格分支对应的输出码元计算分支度量。分支度量累加值作为最终幸存路径选择依据。网格分支对应的输出码元由输入码元和生成多项式共同决定，对于支持生成多项式任意配置的 viterbi 译码器，需增加一个网格分支计算（Trellis Branch Compute，TBC）单元。根据生成多项式配置，得到相应网格分支，用于后续译码。

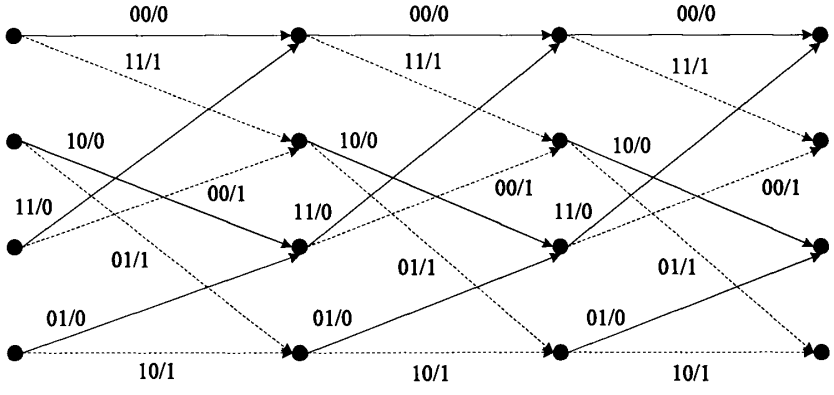

图 4-9：生成多项式在网格图中的体现

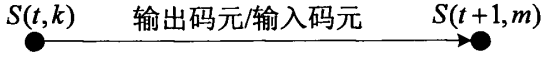

图 4-10：网格图状态转移示例

3. 码率

编码速率简称码率，码率 $R=k_0/n_0$，是衡量卷积编码效率的重要参数。码率对译码器的影响体现在分支度量计算，分支度量计算如公式(4.12)所示。当码率不同时，

分支度量计算模块所需的加法器数目不同。基二和基四算法下所需加法器数目如表 4-5 所示。

$$M(r_k|y_k)=\sum_{j=0}^{n_0-1}M(r_k^{(j)}|y_k^{(j)})=\sum_{j=0}^{n_0-1}r_k^{(j)}y_k^{(j)} \tag{4.12}$$

表 4-5：不同基数不同码率下所需加法器数目

基数\码率	1/2	1/3	1/4
基二	2	3	4
基四	4	6	8

4. 结尾方式

卷积编码器一般有两种结尾方式：零结尾和咬尾。零结尾通过向编码器输入 m 个“0”，使编码器最终回到全“0”状态，此种结尾方式会带来信息冗余，尤其在数据长度比较小的情况下，对码率影响比较显著。另一种结尾方式是将数据尾端 m 段数据存储在移位寄存器中，作为编码器的初始状态。编码结束后，编码器的结束状态与初始状态相同。咬尾的结尾方式可以提高传输效率，但由于编码器初始状态和结束状态未知，一定程度上提高了译码器复杂度[120-122]。

本文采用滑窗流水机制，假设待译码数据分为 N_{sw} 个滑窗，滑窗长度为 L。在 $t=(N_{sw}-2)L$ 时刻，译码器开始对第 $N_{sw}-2$ 滑窗进行回溯。此时，可以同时对第 $N_{sw}-1$ 滑窗的起始回溯状态进行获取，这样在 $t=(N_{sw}-1)L$ 时刻就可以开始对第 $N_{sw}-1$ 滑窗进行回溯。基于此，可以在待译码数据后面添加一个滑窗的冗余数据，用于获取第 $N_{sw}-1$ 滑窗的起始回溯状态，如图 4-11 所示。

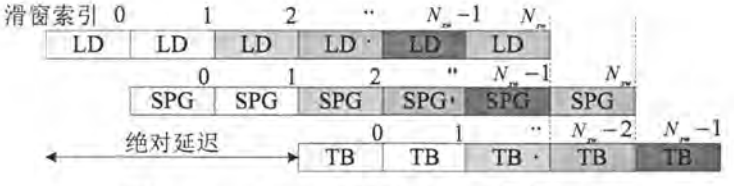

图 4-11：Viterbi 译码器对结尾方式处理

冗余滑窗添加法对零结尾和咬尾均适用。若是零结尾方式，冗余滑窗数据均为 0；若是咬尾方式，将待译码数据的头部添加在尾部，作为冗余滑窗数据。冗余滑窗添加完成后，后续译码过程不再区分零结尾和咬尾两种结尾方式，此处理虽增加了

数据冗余，但时间上没有增加开销，且在一定程度上降低了算法复杂度。

4.3.3 浮点算法仿真

仿真平台由 C 语言搭建，如图 4-12 所示。仿真条件：在 AWGN 信道下，采用 QPSK 调制，码块长度为 1080 比特。仿真了基于 LTE 协议的咬尾卷积编码和译码性能。仿真结果如图 4-13 所示。仿真表明：基于滑窗流水的后向回溯法比传统后向回溯法，BLER 性能提升约 1.8dB。这是由于冗余滑窗添加，对译码器初始状态获取更加准确。

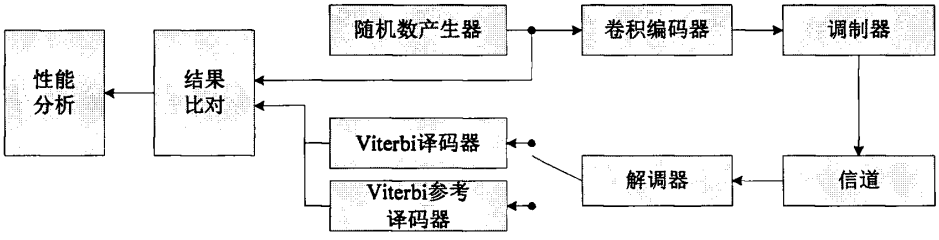

图 4-12：Viterbi 译码器算法仿真平台

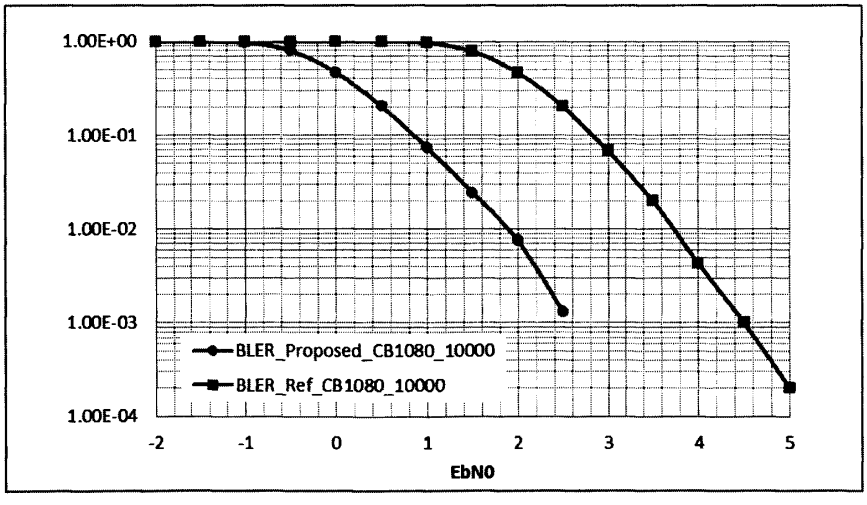

图 4-13：Viterbi 译码器 BLER 性能曲线

4.3.4 定点化系统仿真

译码器的输入端可以是硬比特或者软信息。硬比特通常是比特{0,1}，软信息一般是用 3~8 比特位宽表示的符号。一般来说，软判决比硬判决会带来 3dB 性能增益，

所以在实际应用中一般采用软判决译码。

目前的数字信号处理器无法直接处理模拟信号，所以需要把连续的数据量化成离散的、在一定范围内的数值数据，然后再进行处理。这一过程就是量化过程。量化之后的数据送入处理器进行运算，目前的处理器设计绝大部分都是支持定点运算。浮点运算由于运算复杂、价格昂贵等原因还没有得到大范围普及，所以需要对采集到的浮点数做定点化设计。

定点化设计是把采集到的浮点数限定在有限位宽内，并用固定精度的定点数值来表示浮点数。定点化主要聚焦两点：位宽和精度。一般的数字信号处理器，定点数位宽通常为 8/16/32 位，定点化主要确定小数点在数值中的位置，即可表示不同大小和不同精度的小数。数的定标一般采用 Q 表示法。表 4-6 列出了一个 16 位数的 16 种 Q 表示法及其能表示的十进制数数值范围。

表 4-6：Q 表示法

Q 表示	近似精度	十进制数表示范围
Q0	1	$-32768 \le X \le 32767$
Q1	0.5	$-16384 \le X \le 16383.5$
Q2	0.25	$-8192 \le X \le 8191.75$
Q3	0.125	$-4096 \le X \le 4095.875$
Q4	0.0625	$-2048 \le X \le 2047.9375$
Q5	0.0313	$-1024 \le X \le 1023.96875$
Q6	0.0156	$-512 \le X \le 511.9804375$
Q7	0.0078	$-256 \le X \le 255.9921875$
Q8	0.0039	$-128 \le X \le 127.9960938$
Q9	0.002	$-64 \le X \le 63.9980469$
Q10	0.001	$-32 \le X \le 31.9990234$
Q11	0.0005	$-16 \le X \le 15.9995117$
Q12	0.0002	$-8 \le X \le 7.9997559$

Q13	0.0001	$-4 \le X \le 3.9998779$
Q14	0.00006	$-2 \le X \le 1.999939$
Q15	0.00003	$-1 \le X \le 0.9999695$

从上表可以看出：同样一个数，若定标不同，它所表示的数也不同。例如十六进制数 2000H，若定标为 Q0，则表示 8192；若定标为 Q15，则表示 0.25。不同的 Q 表示的数值范围和精度都不同，Q 越大，表示的精度越高，但表示的数据范围越小。所以，对定点化设计而言，数值范围与精度是一对矛盾：一个变量若想提高数值范围，则必须要以牺牲精度为代价；反之，一个变量精度越高，其表示的数值范围越小。

浮点数与定点数之间的转换关系如公式(4.13)所示。

$$\begin{aligned} \lambda_q &= (\text{int})\lambda * 2^{Q} \\ \lambda &= (float)\lambda_q * 2^{-Q} \end{aligned} \tag{4.13}$$

基于 ASIC 实现形式的 viterbi 译码器，其定点化后的数据位宽一般没有 8/16/32 的限制。位宽越大，精度提升空间也越大，但需要的存储空间也随之增加。所以 viterbi 译码器定点化设计的准则是在满足精度要求的情况下尽可能选取位宽较小的定标。

本文中定点化仿真的评价标准是平均误差（均方根误差与定点傅里叶变换结果的均方根的比值）和信号量化噪声比（Signal-to-Quantization Noise Ratio，SQNR）。一般情况下，良好的定点化至少要保证 SQNR 不小于 30。平均误差和信号量化噪声比如公式(4.14)和(4.15)所示。

$$\frac{rms(error)}{rms(result)} = \sqrt{\frac{\frac{1}{N}\sum_{k=0}^{N-1}\left|X_{float}(k) - X_{fix}(k)\right|^2}{\frac{1}{N}\sum_{k=0}^{N-1}\left|X_{fix}(k)\right|^2}} \tag{4.14}$$

$$SQNR = 10\log_{10}\frac{\frac{1}{N}\sum_{k=0}^{N-1}\left|X_{float}(k)\right|^2}{\frac{1}{N}\sum_{k=0}^{N-1}\left|X_{float}(k) - X_{fix}(k)\right|^2} \tag{4.15}$$

定点化设计流程，首先统计送入译码器的输入码元数值范围，称为粗定标。根据输入码元数值范围，初步确定定点数据位宽和精度。然后将初步定标的输入码元

送入定点译码器，与浮点译码器的输出结果做比较，根据平均误差和信号量化噪声比对数据位宽和精度做微调，称为细定标。最后，整体比较浮点译码器和定点译码器的 BER/BLER 性能，确定最终标值。定点化设计流程如图 4-14 所示。

通过大量的定点化仿真和迭代设计，最终确定 viterbi 译码器输入码元标值为（8,5），即总位宽为 8 比特，小数位宽为 5 比特。Viterbi 译码器内部信号分支度量标值为（11,5），状态度量标值为（12,5）。在此定标下，定点译码器与浮点译码器的 BLER 性能曲线如图 4-15 所示，其仿真条件与上节浮点算法仿真相同。从图中可以看出，定点化设计几乎没有带来性能损失。

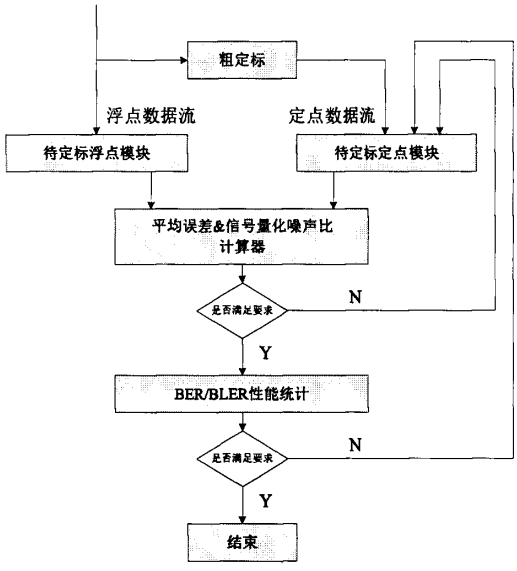

图 4-14：定点化设计流程

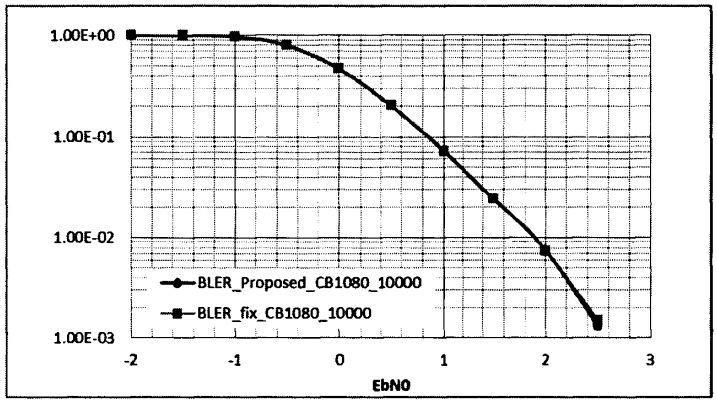

图 4-15：定点译码器 BLER 性能曲线

4.4 Viterbi 译码器实现

本文在算法仿真和定标确定的基础上进行 RTL 设计。首先根据流水划分，完成各个子功能模块设计；然后通过控制模块 CTR 将各个子功能模块联结起来。本文中 viterbi 译码器分为三个流水级：加载数据 LD、网格图前向递归计算产生幸存路径 SPG、后向回溯 TB。LD 模块加载数据到译码器内部缓存区。SPG 模块包含十六组蝶形处理单元，用于状态度量更新运算。每个蝶形处理单元又包含一个分支度量计算部件和四个加比选（Add-Compare-Select，ACS）运算部件。加比选运算部件路径延迟比较大，是本设计中重点优化部件。TB 模块用于后向回溯，每个时钟周期产生两比特译码结果。OUTPUT 用于将译码结果输出。译码器功能框图如图 4-16 所示。

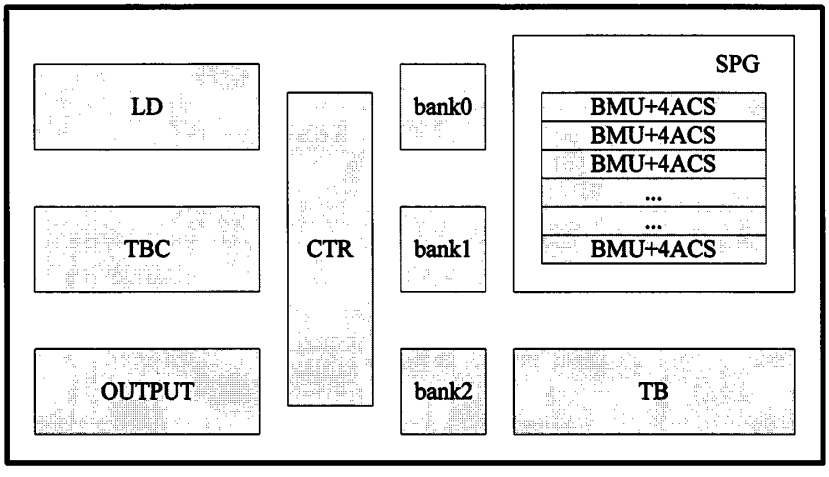

图 4-16：Viterbi 译码器功能框图

4.4.1 Viterbi 译码器整体架构

Viterbi 译码器电路结构如图 4-17 所示。译码器内部包含的缓存单元有：四路校验信息输入缓存、网格分支存储单元、路径度量存储单元、起始回溯状态存储单元和幸存比特存储单元。其译码状态机如图 4-18 所示，状态说明如下：

S0：网格分支计算。

S1：加载数据到内部缓存区。

S2：网格图前向递归计算，产生所有状态的幸存路径。

S3：后向回溯译码。

S4：将译码结果输出到外部缓存区。

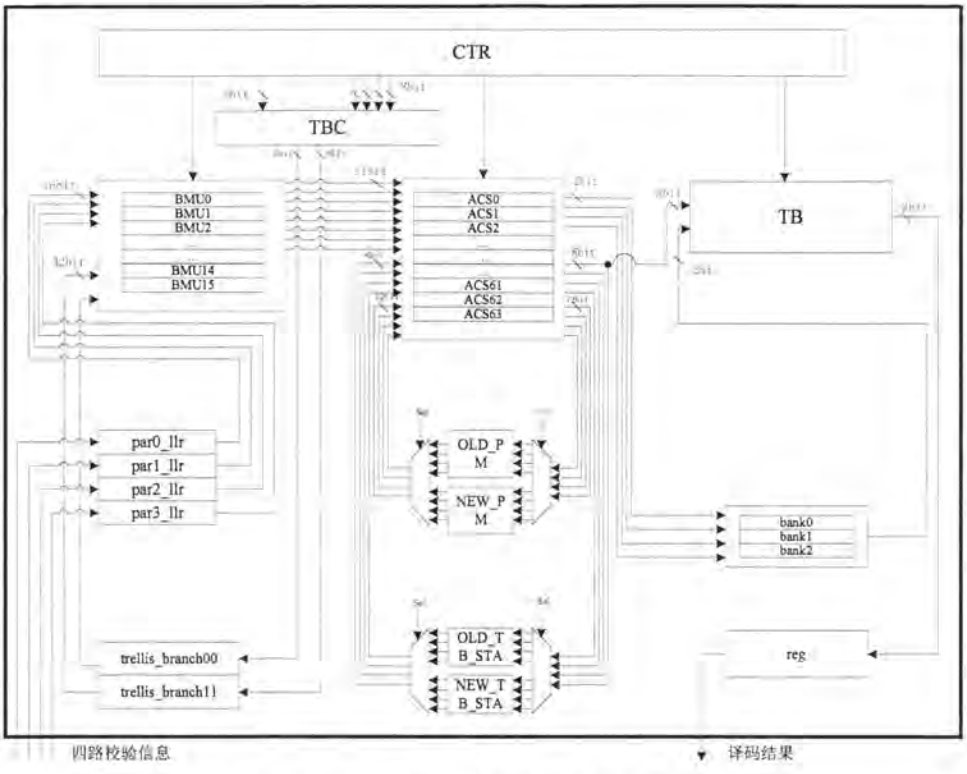

图 4-17：Viterbi 译码器电路结构图

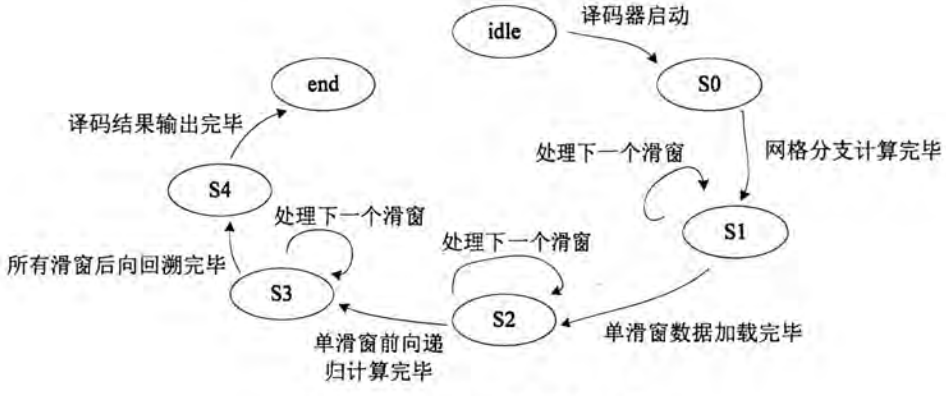

图 4-18：Viterbi 译码器状态机

译码器启动后，首先进行网格分支计算。网格分支计算结束后，加载一个滑窗的数据到内部缓存。数据加载结束后，如果当前滑窗索引小于滑窗数目，继续加载下一滑窗数据，同时译码器进入前向递归状态，产生幸存路径。前向递归得到前一滑窗起始回溯状态，当前滑窗前向递归结束后，前一滑窗进入后向回溯译码状态。后向回溯译码结束后，将译码结果写回输出缓存。

下面重点分析译码器中设计复杂度最高的 SPG 模块。

以（3,1,7）卷积码为例进行说明，前向状态递归如图 4-19 所示。基四算法下，每四组状态之间的转移关系可以组成一个闭环，对四组状态的处理单元称为 PE（Processing Engine）单元，PE 状态转移关系可以用公式(4.16)来表示。

$$\begin{bmatrix} S_{4k+0} \\ S_{4k+1} \\ S_{4k+2} \\ S_{4k+3} \end{bmatrix} \rightarrow \begin{bmatrix} S_{k+0*K} \\ S_{k+2*K} \\ S_{k+1*K} \\ S_{k+3*K} \end{bmatrix} \tag{4.16}$$

其中，K 表示 PE 个数，$K=n/4$，n 为译码器总状态数目，k 为当前 PE 索引。PE 计算逻辑可以用公式(4.17)来表示，其中分支度量计算如公式(4.18)所示，n_0 表示码率倒数。

$$\begin{aligned}
V_{t+1,k+0*K} &= \min\{V_{t,4k+0}+M(r_t|y_{t,(4k+0\rightarrow k+0*K)}),V_{t,4k+1}+M(r_t|y_{t,(4k+1\rightarrow k+0*K)}),\\
&\quad V_{t,4k+2}+M(r_t|y_{t,(4k+2\rightarrow k+0*K)}),V_{t,4k+3}+M(r_t|y_{t,(4k+2\rightarrow k+0*K)})\}\\
V_{t+1,k+1*K} &= \min\{V_{t,4k+0}+M(r_t|y_{t,(4k+0\rightarrow k+1*K)}),V_{t,4k+1}+M(r_t|y_{t,(4k+1\rightarrow k+1*K)}),\\
&\quad V_{t,4k+2}+M(r_t|y_{t,(4k+2\rightarrow k+1*K)}),V_{t,4k+3}+M(r_t|y_{t,(4k+2\rightarrow k+1*K)})\}\\
V_{t+1,k+2*K} &= \min\{V_{t,4k+0}+M(r_t|y_{t,(4k+0\rightarrow k+2*K)}),V_{t,4k+1}+M(r_t|y_{t,(4k+1\rightarrow k+2*K)}),\\
&\quad V_{t,4k+2}+M(r_t|y_{t,(4k+2\rightarrow k+2*K)}),V_{t,4k+3}+M(r_t|y_{t,(4k+2\rightarrow k+2*K)})\}\\
V_{t+1,k+3*K} &= \min\{V_{t,4k+0}+M(r_t|y_{t,(4k+0\rightarrow k+3*K)}),V_{t,4k+1}+M(r_t|y_{t,(4k+1\rightarrow k+3*K)}),\\
&\quad V_{t,4k+2}+M(r_t|y_{t,(4k+2\rightarrow k+3*K)}),V_{t,4k+3}+M(r_t|y_{t,(4k+2\rightarrow k+3*K)})\}
\end{aligned} \tag{4.17}$$

$$M(r_t|y_t)=\sum_{i=0}^{n_0-1} r_{t,i}\times y_{t,i} \tag{4.18}$$

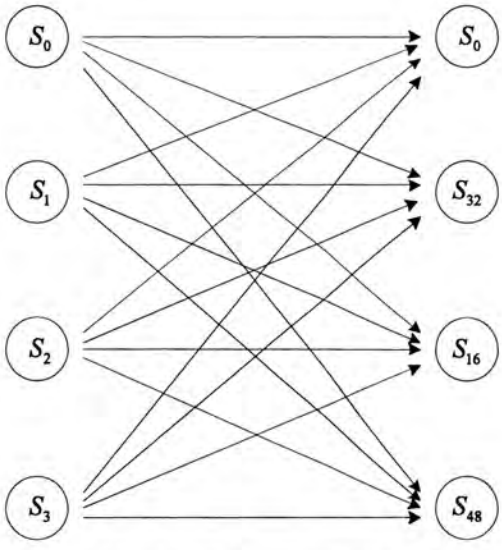

图 4-19：（3,1,7）卷积码状态转移图

本文通过分析分支度量数据特点，得到表 4-7 所示规律：十六组分支度量计算可以简化为八组。根据简化分支度量计算和 PE 状态转移关系，可以得到 PE 模块电路结构，如图 4-20 所示。该模块对状态度量值和起始回溯状态进行更新，并得到部分路径幸存比特。

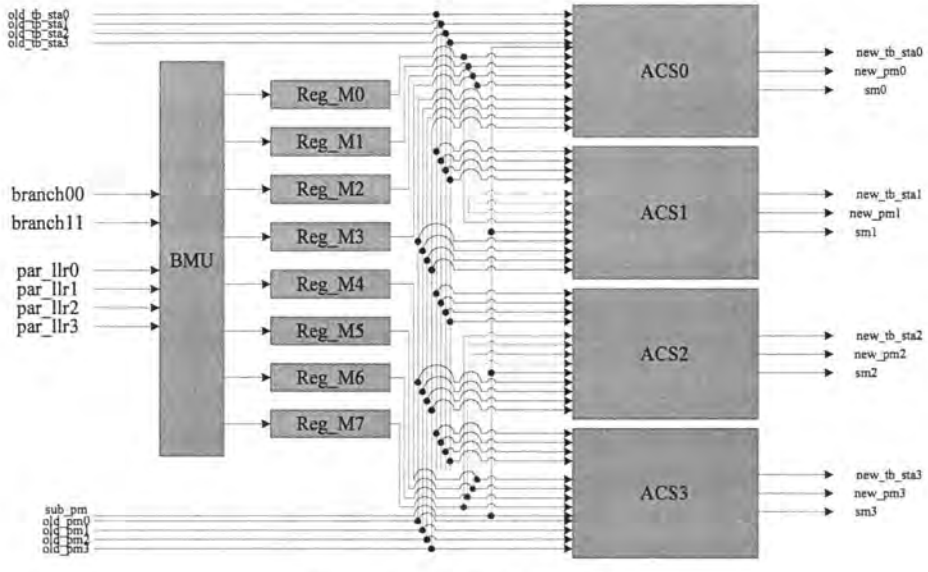

图 4-20：PE 模块电路结构

表 4-7：分支度量计算规律

转移分支	分支度量
$S_{t,4k+0} \rightarrow S_{t+1,k+0*K}$	M_0
$S_{t,4k+1} \rightarrow S_{t+1,k+0*K}$	M_1
$S_{t,4k+2} \rightarrow S_{t+1,k+0*K}$	M_2
$S_{t,4k+3} \rightarrow S_{t+1,k+0*K}$	M_3
$S_{t,4k+0} \rightarrow S_{t+1,k+2*K}$	$-M_1$
$S_{t,4k+1} \rightarrow S_{t+1,k+2*K}$	$-M_0$
$S_{t,4k+2} \rightarrow S_{t+1,k+2*K}$	$-M_3$
$S_{t,4k+3} \rightarrow S_{t+1,k+2*K}$	$-M_2$
$S_{t,4k+0} \rightarrow S_{t+1,k+1*K}$	$-M_5$
$S_{t,4k+1} \rightarrow S_{t+1,k+1*K}$	$-M_4$
$S_{t,4k+2} \rightarrow S_{t+1,k+1*K}$	$-M_7$
$S_{t,4k+3} \rightarrow S_{t+1,k+1*K}$	$-M_6$
$S_{t,4k+0} \rightarrow S_{t+1,k+3*K}$	M_4
$S_{t,4k+1} \rightarrow S_{t+1,k+3*K}$	M_5
$S_{t,4k+2} \rightarrow S_{t+1,k+3*K}$	M_6
$S_{t,4k+3} \rightarrow S_{t+1,k+3*K}$	M_7

4.4.2 基四算法下加比选结构优化

PE 中加比选部件是 viterbi 译码器中时序约束最紧张的。传统基二算法下加比选结构如图 4-21 所示。

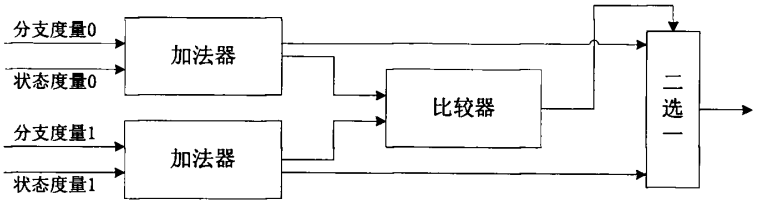

图 4-21：基二算法下加比选结构

本文采用基四算法，其加比选结构如图 4-22 所示。面积上比基二算法多了两个加法器和两个比较器，时间上比基二算法多了一级比较运算。网格图递归计算，状态度量值不断累加，一定时间后会超出数据表示范围，此时译码器失效。为了避免数据溢出，加比选部件还承担了规约运算。最直接的规约运算是每时刻数据累加后，从状态度量值中选取最大的一个，每个状态度量值分别减去这个数。这种方法延迟比较大，在每级加比选运算后，需要增加若干级比较和减法运算，时间消耗与加比选运算相当，这种方法是不划算的。还有一种模规约算法，需要增加两级异或门操作，其结构如图 4-23 所示。此规约算法不能与加比选操作同时进行，会进一步增加关键路径延迟。

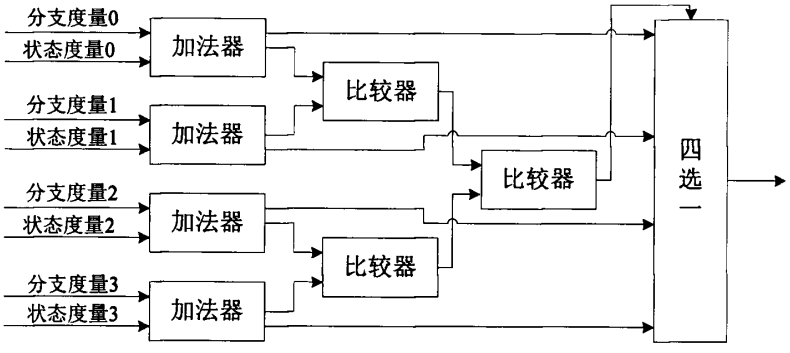

图 4-22：基四算法下加比选结构

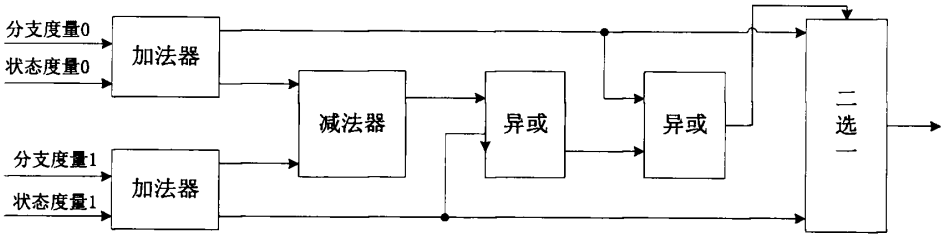

图 4-23：模规约结构图

本文加比选部件主要在两个方面进行优化：缩短四选一逻辑、规约运算。具体实现思路如下：

1. 将分支度量与前一时刻状态度量值累加后的结果记为sum_0、sum_1、sum_2、sum_3。
2. sum_0、sum_1、sum_2、sum_3两两进行比较，比较的结果以0、1表示，比较逻辑为$sum_i > sum_j ?0:1$。
3. 比较的同时，sum_0、sum_1、sum_2、sum_3分别减去前一时刻的某一状态度量值，例如减去sum_0，得到sum_0'、sum_1'、sum_2'、sum_3'，即得到规约后的状态度量值。
4. 根据比较结果，从sum_0'、sum_1'、sum_2'、sum_3'选出最小的一个。四选一逻辑如表 4-8 所示。

加比选部件如图 4-24 所示，包括四个加法器、四个减法器、六个比较器和一个选择逻辑。本文对基四算法下加比选部件进行优化，使四选一逻辑在一级比较中完成，同时把规约运算隐藏在加比选计算中，降低关键路径延迟，进而提升系统整体性能。

表 4-8：四选一逻辑

sum_0,sum_1	sum_0,sum_2	sum_0,sum_3	sum_1,sum_2	sum_1,sum_3	sum_2,sum_3	比较结果
1	1	1	0	0	0	sum_0
1	1	1	0	0	1	sum_0
1	1	1	1	0	0	sum_0
1	1	1	1	1	0	sum_0
1	1	1	0	1	1	sum_0
1	1	1	1	1	1	sum_0
0	0	0	1	1	0	sum_1
0	0	0	1	1	1	sum_1
0	1	0	1	1	0	sum_1
0	1	1	1	1	0	sum_1
0	0	1	1	1	1	sum_1
0	1	1	1	1	1	sum_1
0	0	0	0	0	1	sum_2
0	0	0	0	1	1	sum_2
1	0	0	0	0	1	sum_2
1	0	1	0	0	1	sum_2
0	0	1	0	1	1	sum_2
1	0	1	0	1	1	sum_2
0	0	0	0	0	0	sum_3
0	0	0	1	0	0	sum_3
1	0	0	0	0	0	sum_3
1	1	0	0	0	0	sum_3
0	1	0	1	0	0	sum_3
1	1	0	1	0	0	sum_3

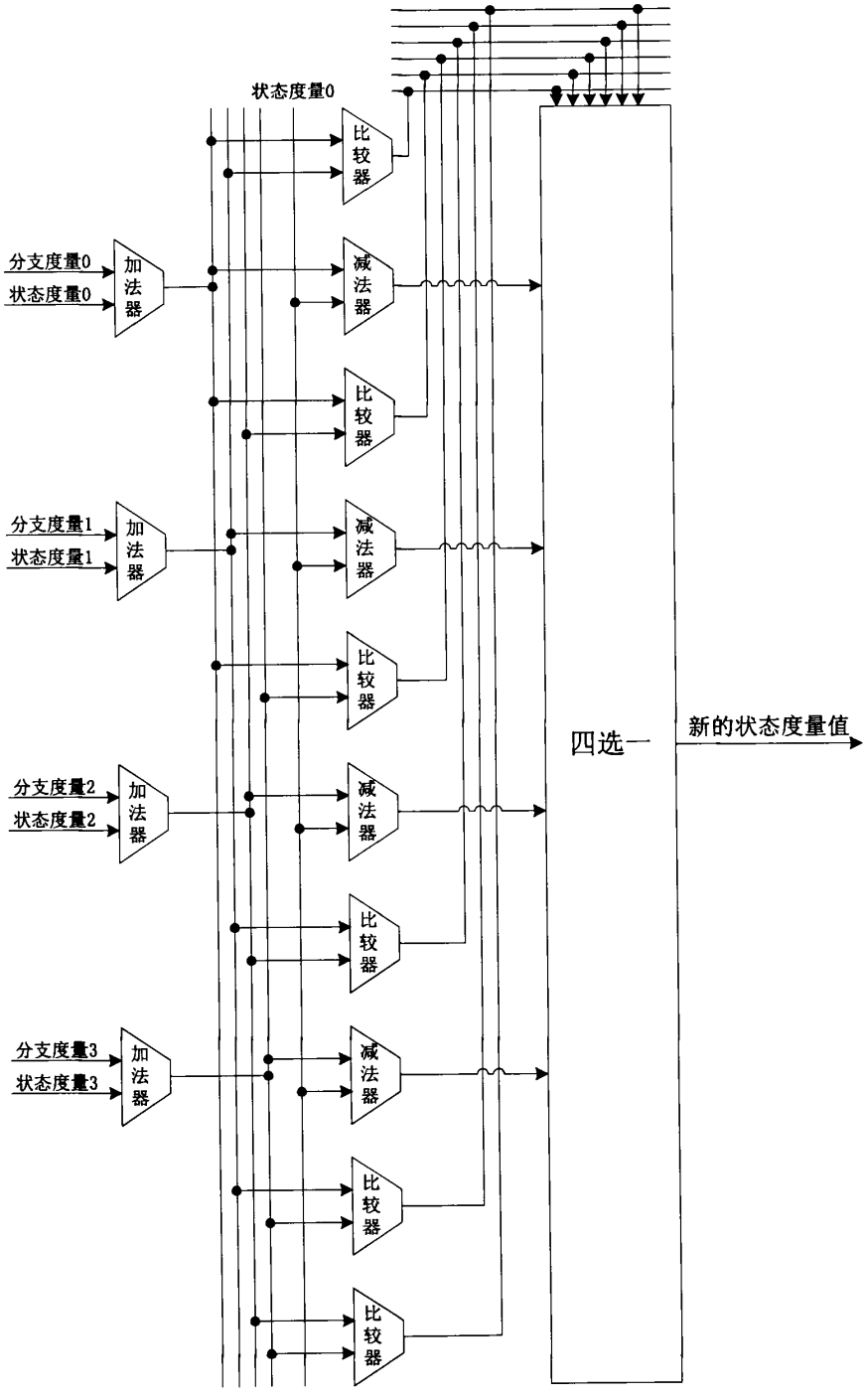

图 4-24：基四算法下加比选部件电路结构

4.4.3 Viterbi 译码器性能评估

1. 数据吞吐

假设码块长度为 K，主频为 f，译码器流水之前的绝对延迟为 $96+n$ 个周期，其中 n 为译码器状态数目。译码器进入完全流水后，每个周期产生两个比特。所有数据处理完毕需要 $K/2$ 周期。所以，译码器处理一个码块的时间如公式(4.19)所示，数据吞吐如公式(4.20)所示。

$$t=(96+n+\frac{K}{2})\times\frac{1}{f} \tag{4.19}$$

$$Throughtput=\frac{2Kf}{2n+192+K} \tag{4.20}$$

译码器数据吞吐与约束长度有关，表 4-9 统计了不同约束长度下 viterbi 译码器吞吐量计算公式，以及当码块长度为 6144，主频为 600MHz 时，译码器峰值吞吐。

表 4-9：不同约束长度下 viterbi 译码器吞吐量计算

约束长度	吞吐量计算公式	吞吐量（K=6144）
5	$\frac{2Kf}{224+K}$	1.15Gbps
6	$\frac{2Kf}{256+K}$	1.15Gbps
7	$\frac{2Kf}{320+K}$	1.14Gbps
8	$\frac{Kf}{288+K}$	573.13Mbps
9	$\frac{Kf}{544+2K}$	287.28Mbps

通过波形仿真，译码器实际吞吐与理论估计基本吻合。以码块长度 6144，约束长度 6 为例，图 4-25 和图 4-26 分别记录了译码器起始工作时刻和结束工作时刻，译码器工作周期如公式(4.21)所示：

$$t=\frac{t_{end}-t_{start}}{2000}cycle=\frac{6830000-418000}{2000}=3206cycle \tag{4.21}$$

当主频为 600MHz 时，数据吞吐为：

$$\frac{6144}{3206*\frac{1}{f}}=1.15Gbps \tag{4.22}$$

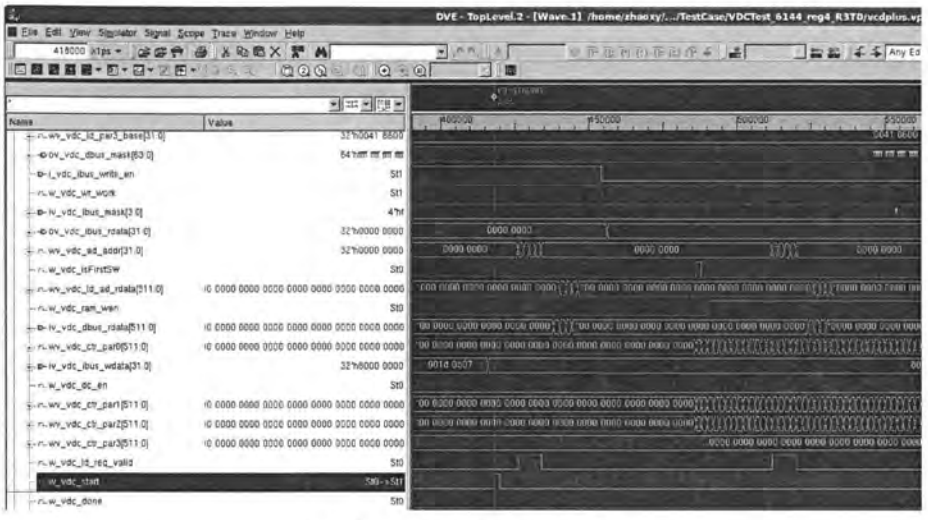

图 4-25：译码器起始工作时刻

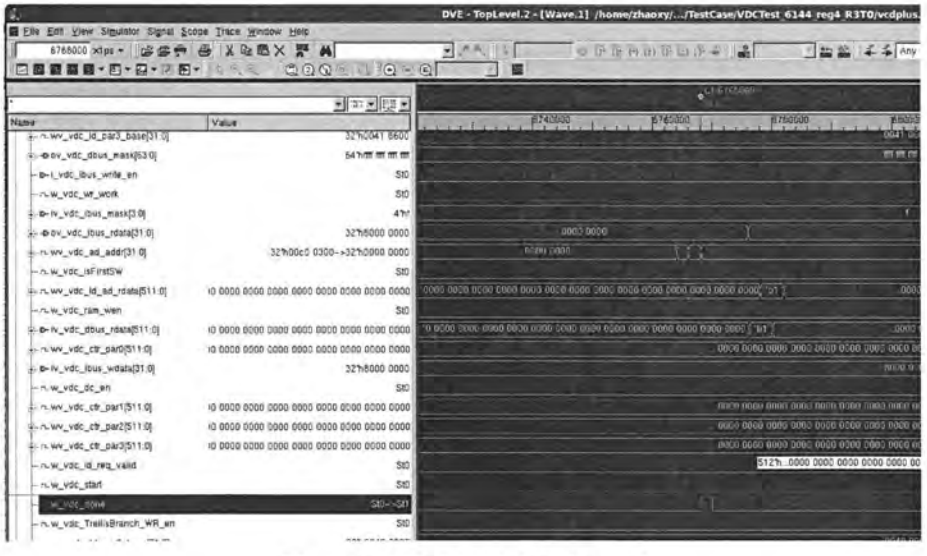

图 4-26：译码器结束工作时刻

主流商用 viterbi 译码器 VCP2，数据处理能力为 9.5Mbps@40bit,333MHz，本文译码器数据处理能力为 32.173Mbps@40bit,333MHz，性能提升约 3.3 倍。

2. 综合结果

本文采用 Synopsys 标准单元库，利用 Synopsys Design Compiler 进行综合， 综合结果如表 4-10 所示。从表中可以看出，本文中 viterbi 译码器支持的标准更为丰富， 例如对 CDMA2000 中 1/4 码率支持。文献[102] 不支持 1/4 码率译码，且高吞吐是通过提高主频来获得的。在时钟主频为 3.8GHz 时，峰值吞吐为 1.9Gbps。本文通过提高译码基数，采用基四算法，得到了较高的数据吞吐。从面积和功耗角度来看，其性能也优于其它可配置型 viterbi 译码器。

表 4-10：Viterbi 译码器 DC 综合结果

类别	本论文	[102]	[73]	[103]
码率	1/2,1/3,1/4	1/2,1/3	1/2,1/3	1/2,1/3
约束长度	5~9	5~9	3,5,7	5~9
工艺	28nm	90nm	28nm	28nm
主频	600MHz	3.8GHz	--	270.5MHz
峰值吞吐	1.15Gbps	1.9Gbps	--	541Mbps
面积	0.211mm^2	1.9mm^2	--	--
功耗	46mW	358mW	78mW	--

4.5 小结

本章采用基于滑窗流水的后向回溯基四算法，实现了支持多标准的高性能可配置 viterbi 译码器。通过冗余滑窗添加，在不增加译码延迟基础上，BLER 性能提升约 1.8dB。该译码器支持码率 1/2、1/3、1/4，约束长度 5~9，生成多项式可任意配置，支持零结尾和咬尾。与主流商用 viterbi 译码器相比，全球领先的半导体公司-TI 的 VCP2 数据处理能力为 9.5Mbps@40bit,333MHz，本文中译码器数据处理能力为 32.173Mbps@40bit,333MHz，性能提升约 3.3 倍。

本章对基于滑窗流水的后向回溯基四算法进行了浮点仿真和定点化设计，对译码器进行了 RTL 实现，重点对基四算法下加比选结构进行优化。在 28nm 标准单元库下，主频可达到 600MHz，面积为 0.211mm^2，功耗为 46mW。该译码器在数据处

理速度、面积与功耗方面的综合性能要优于其它可配置型 viterbi 译码器，适用于高容量网络的译码需求。

第5章 基于连续消除列表的 polar 译码器

5.1 引言

SCL 算法对于 polar 码是一种有效译码算法，具有完美的纠错性能。传统 SCL 算法在对信息比特进行译码时，会伴随路径分裂，搜索路径数目由 L 扩大为 $2L$。为了避免搜索路径数目指数增加，会对搜索路径进行删减。路径分裂后，对所有候选路径根据度量值大小进行排序，选择具有最好度量值的 L 条路径，将其对数似然比 LLR、路径度量值、部分和向量、部分译码比特等数据进行复制。路径扩展和删减消耗大量计算资源，译码延迟比较大。

目前已知文献中对路径分裂比较有效的研究是快速简化 SCL 算法，该算法针对 Rate-1 节点，特定情况下，得到前 L-1 译码比特后，节点中的后续比特可直接译出，无需路径扩展[137]。文献[138]提出一种低复杂度路径删减策略，最大度量值乘以某个因子作为删减阈值，小于该阈值的路径将被删除。文献[139]提出了一种基于后验概率的删减策略来提高低 SNR 区间 BLER 性能。文献[140]了提出一种降低路径分裂数目的 SCL 译码算法，该算法经过若干译码阶段后，仅会留下一条候选路径，因此译码性能不是很理想。

本文提出了一种基于对数似然比的路径扩展优化方法和一种基于置信区间的新型删减策略来降低译码复杂度。仿真表明：基于路径扩展优化方法的平均分裂路径数目与传统 SCL 算法相比降低了 49%；在性能损失可忽略情况下，新型删减策略可以降低搜索路径数目：中 SNR 区间可以降低 60%，高 SNR 区间降低 80%。

5.2 SCL 算法概述

5.2.1 Polar 码

Polar 码来源于信道极化现象，如图 5-1 所示。信道极化包括两个阶段：信道合并和信道分离[141]。对于 B-DMC 信道，随着信道分裂数目 N 变大，较小索引的信道对称容量趋近于 0，较大索引的信道对称容量趋近于 1。合理利用极化现象可以使数据传输更可靠。Polar 码生成矩阵基于克罗内克积，克罗内克积 $F^{\otimes n}$ 定义为 $F^{\otimes n}=F\otimes F^{\otimes(n-1)}$（$n\geq 1$），且 F 初始值为

$$F=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}。$$

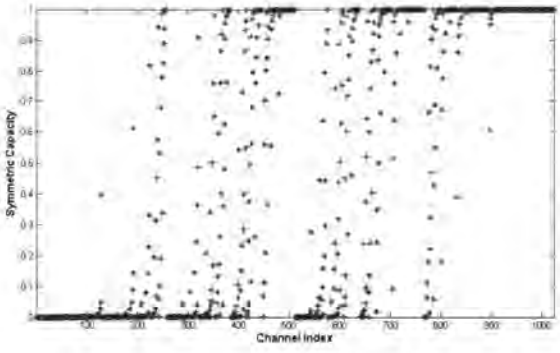

图 5-1：信道极化现象

Polar 编码定义为

$$x_1^N = u_1^N G_N = u_1^N B_N F^{\otimes n} \tag{5.1}$$

其中，u_1^N 表示数据源序列（$u_1,u_2,\cdots,u_N$），x_1^N 表示编码后码字序列（$x_1,x_2,\cdots,x_N$），B_N 表示逆序矩阵。数据源序列包含信息比特和冻结比特，其中，信息比特索引用集合 A 表示，冻结比特索引用集合 A^C 表示。Polar 编码也可以写作

$$x_1^N = u_A G_N(A) \oplus u_{A^c} G_N(A^c) \tag{5.2}$$

其中，$G_N(A)$表示生成矩阵 G_N 的子矩阵，由矩阵中索引为 A 的行组成。

5.2.2 SCL 译码

SCL 算法由 SC 演进而来，下面首先介绍 SC 算法。

信道分裂后，第 i 个子信道传输概率定义为

$$W_N^{(i)}(y_1^N, u_1^{i-1} | u_i) \triangleq \sum_{u_{i+1}^N \in \chi^{N-i}} \frac{1}{2^{N-1}} W_N(y_1^N | u_1^N) \tag{5.3}$$

其中，

$$W_N(y_1^N | u_1^N) = \prod_{i=1}^{N} W(y_i | x_i) \tag{5.4}$$

判决比特通过下式得到：

$$\hat{u}_i = \begin{cases} 0, & L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \geq 0 \\ 1, & otherwise \end{cases} \tag{5.5}$$

其中，对数似然比 LLR 定义为

$$L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \ln \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | 1)} \tag{5.6}$$

LLR 递归计算如公式(5.7)和(5.8)所示，

$$\begin{aligned} & L_N^{(2i-1)}(y_1^N, \hat{u}_1^{2i-2}) \\ & = L_{N/2}^{i}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) \boxplus L_{N/2}^{i}(y_{N/2+1}^{N}, \hat{u}_{1,e}^{2i-2}) \end{aligned} \tag{5.7}$$

$$\begin{aligned} & L_N^{(2i)}(y_1^N, \hat{u}_1^{2i-1}) \\ & = L_{N/2}^{i}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})^{\hat{u}_{2i-1}} + L_{N/2}^{i}(y_{N/2+1}^{N}, \hat{u}_{1,e}^{2i-2}) \end{aligned} \tag{5.8}$$

其中，“$\boxplus$”运算定义为 $a \boxplus b \triangleq \log((1+e^{a+b})/(e^a+e^b))$。SC 算法递归特性使得长度为 N 的 LLR 计算可以缩减为两个长度为 $N/2$ 的 LLR 计算。通常，奇索引的 LLR 计算用 f 函数表示，偶索引用 g 函数表示。

SC 译码过程可以看作是对深度为 n 的满二叉树 T_n 遍历，其中，叶子节点数目等于码块长度 $N=2^n$，如图 5-2 所示。白色圆圈表示 Rate-0 节点，黑色圆圈表示 Rate-1 节点，灰色圆圈表示 Rate-R 节点。对于某个节点 v，其子树深度、LLR 信息、部分和向量分别用 d_v、α_v 和 β_v 表示。每个节点 v 可以看作一个分量译码器，根节点接收信道 LLR，产生部分和向量 β_v，对 β_v 重新编码可以得到译码序列。每个节点的左分支遍历相当于 f 操作，右分支遍历相当于 g 操作，部分和更新如公式(5.9)所示。SC 译码是对满二叉树进行深度优先搜索，完整遍历需要 $2N-2$ 周期（假设每级 f、g 操作能在一个周期内完成）。

$$\begin{cases} \beta_v[2i] = \beta_{vl}[i] \oplus \beta_{vr}[i] \\ \beta_v[2i+1] = \beta_{vr}[i] \end{cases} (i = 0 : 2^{n-d_v-1} - 1) \tag{5.9}$$

SCL 算法在 SC 基础上进行改进，用于提高 polar 码译码性能。在每个译码阶段，路径分裂为并行的“0”“1” 两个线程。为了避免路径数目指数增加，当路径数目达到 $2L$ 后，一半的路径将被删除，只保留 L 条候选路径。每条路径均进行 SC 译码过程，并计算相应的路径度量值。SCL 译码复杂度为 $O(LN\log N)$。根据文献[131]，在每个译码阶段 i，每条路径 l 的度量值定义为

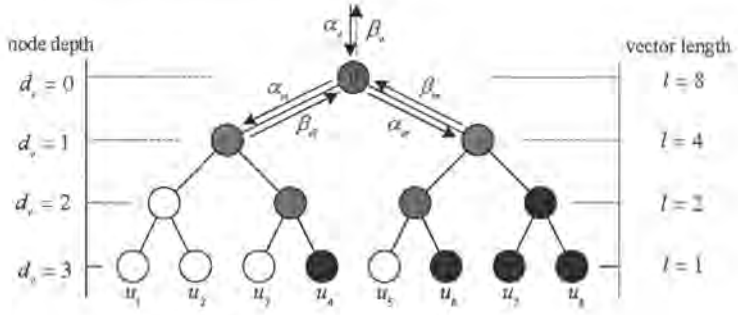

图 5-2：SC 满二叉树遍历

$$PM_l^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1+e^{-(1-2\hat{u}_j[l])\cdot L_N^{(j)}[l]}\right) \tag{5.10}$$

根据近似

$$\ln(1+e^x) \approx \begin{cases} 0, & x<0 \\ x, & x \geq 0 \end{cases} \tag{5.11}$$

路径度量值可以简化为

$$PM_l^{(i)} = \begin{cases} PM_l^{(i-1)} + \left|L_N^{(i)}[l]\right|, & 1-2\hat{u}_i \neq sign(L_N^{(i)}[l]) \\ PM_l^{(i-1)}, & otherwise \end{cases} \tag{5.12}$$

在译码的最后阶段，从 L 条路径中选择具有最好路径度量值的序列作为译码输出。其中，CRC 辅助的 SCL（CRC-aided SCL，CA-SCL）算法由文献[142]提出。在译码的最后阶段，通过 CRC 校验的候选路径作为译码输出。

SCL 译码过程如下：

（1） 初始化路径 $L(0)=\varnothing$，路径度量值设置为 $PM(\varnothing)=0$。

（2） LLR 递归计算，得到每条路径判决信息 $L_N^{(i)}(y_1^N, \hat{u}_1^{i-1})$。

（3） 路径分裂并计算分裂后每条路径的度量值。如果分裂后的路径数目大于预设搜索路径数目 L，对路径度量值进行排序，保留具有较小度量值的 L 条路径。然后进入步骤（2），进行下一比特译码。

（4） 重复步骤（2）和（3），直到到达译码阶段 N，选择具有最小度量值的路径作为译码输出。

5.3 路径扩展优化

本节从理论方面论证当对数似然比 $L_N^{(i)}(y_1^N,\hat{u}_1^{i-1})$ 满足一定条件时，可以避免路径扩展，降低路径选择复杂度；并对路径扩展优化方法在 Rate-1 节点中的应用进行分析说明。

5.3.1 理论和证明

引理 5.1：对于搜索路径数目为 L 的 SCL 译码器，对信息比特进行译码时，分裂路径最大数目处于区间[L,2L]，其译码结果与传统 SCL 算法相同。

证明：采用数学归纳法进行证明。根据路径度量值定义，对于信息比特译码，分裂路径度量值要么保持不变，要么加上一个增量。与冻结比特不同，信息比特基于似然比进行估计，概率较大的路径度量值保持不变，概率较小的路径度量值增加一个惩罚因子。

假设第 i 译码阶段，L 条幸存路径升序排列的度量值表示为 $PM^{(i)}=\left\{PM_0^{(i)},PM_1^{(i)},...,PM_{L-1}^{(i)}\right\}$。当 l=0，且 $PM_0^{(i-1)}+\left|L_N^{(i)}[0]\right|>PM_{L-1}^{(i-1)}$ 时，现有已知升序排列的 L+1 条路径度量值如下所示：

$$\begin{aligned}PM_0^{(i)}&=PM_0^{(i-1)}\\PM_1^{(i)}&=PM_1^{(i-1)}\\&\cdots\\PM_{L-1}^{(i)}&=PM_{L-1}^{(i-1)}\\PM_L^{(i)}&=PM_0^{(i-1)}+\left|L_N^{(i)}[0]\right|\end{aligned}$$

由于候选路径数目最大为 L，度量值为 $PM_0^{(i-1)}+\left|L_N^{(i)}[0]\right|$ 的路径将被删除。在这种场景下，路径 l=0 的扩展是没有意义的。

当 l=k-1，$PM_l^{(i-1)}+\left|L_N^{(i)}[l]\right|>PM_{L-1}^{(i-1)}$ 时（$2\le k\le L$），考虑两种极端情况：

（1） 路径索引满足 $l=\{0,1,...,k-2\}$ 的路径均没有扩展：

当 $PM_{k-1}^{(i-1)}+\left|L_N^{(i)}[k-1]\right|>PM_{L-1}^{(i-1)}$ 时，现有已知升序排列的 L+1 条路径度量值如下所示：

$$PM_0^{(i)} = PM_0^{(i-1)}$$
$$PM_1^{(i)} = PM_1^{(i-1)}$$
$$\cdots$$
$$PM_{L-1}^{(i)} = PM_{L-1}^{(i-1)}$$
$$PM_L^{(i)} = PM_{k-1}^{(i-1)} + \left|L_N^{(i)}[k-1]\right|$$

由于候选路径数目最大为 L，度量值为 $PM_{k-1}^{(i-1)} + \left|L_N^{(i)}[k-1]\right|$ 的路径将被删除。在这种场景下，路径 l=k-1 的分裂是没有意义的。

（2） 路径索引满足 $l = \{0,1,...,k-2\}$ 的路径均有扩展：

当 $PM_{k-1}^{(i-1)} + \left|L_N^{(i)}[k-1]\right| > PM_{L-1}^{(i-1)}$ 时，现有已知 L+k 条路径度量值如下所示：

$$PM_0^{(i)} = PM_0^{(i-1)}$$
$$PM_1^{(i)} = PM_0^{(i-1)} + \left|L_N^{(i)}[0]\right|$$
$$PM_2^{(i)} = PM_1^{(i-1)}$$
$$\cdots$$
$$PM_m^{(i)} = PM_k^{(i-1)}$$
$$\cdots$$
$$PM_n^{(i)} = PM_{L-2}^{(i-1)}$$
$$PM_p^{(i)} = PM_{L-1}^{(i-1)}$$
$$PM_q^{(i)} = PM_{k-1}^{(i-1)} + \left|L_N^{(i)}[k-1]\right|$$

其中，最小和最大度量值分别为 $PM_0^{(i)}$ 和 $PM_q^{(i)}$，度量值 $PM_1^{(i)} \sim PM_n^{(i)}$ 是乱序的。由于候选路径数目最大为 L，度量值为 $PM_{k-1}^{(i-1)} + \left|L_N^{(i)}[k-1]\right|$ 的路径将被删除。在这种场景下，路径 l=k-1 的分裂是没有意义的。

总之，当路径 l 的对数似然比满足条件 $\left|L_N^{(i)}[l]\right| > PM_{L-1}^{(i-1)} - PM_l^{(i-1)}$ 时，该路径可以省略路径扩展，度量值保持不变，译码比特可直接得出。

证毕。

路径扩展后，从候选路径中选择 L 条路径继续进行 SC 运算。传统 SCL 算法中，分裂路径数目为 $2L$，经过路径扩展优化后，分裂路径数目小于 $2L$。新算法译码流程如图 5-3 所示。

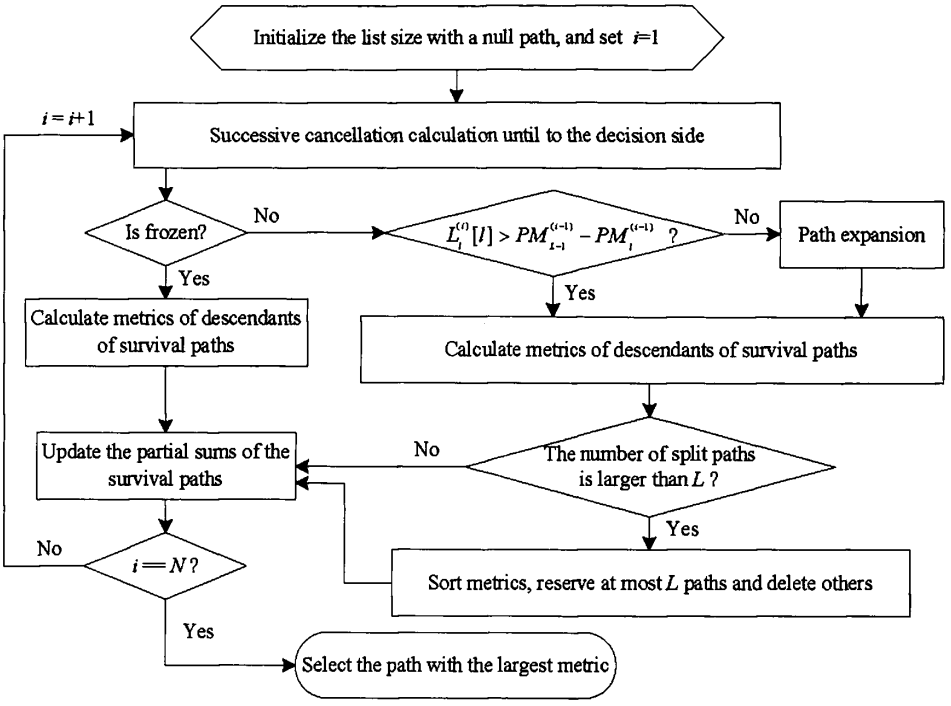

图 5-3：优化 SCL 算法译码流程

5.3.2 Rate-1 节点

传统 SCL 算法中，信息比特串行译码，没有任何并行度。由于 Rate-1 节点所有比特均为信息比特，路径扩展优化方法在该节点中的应用最为典型。

Rate-1 节点度量值计算如公式(5.12)所示。在第 i 译码阶段，结合路径扩展优化方法，如果

$$\begin{aligned}\sum_{l=0}^{L-1}\left|L_N^{(i)}[l]\right| &> \sum_{l=0}^{L-1}\left(PM_{L-1}^{(i-1)} - PM_l^{(i-1)}\right) \\ &> L\cdot PM_{L-1}^{(i-1)} - \sum_{l=0}^{L-1} PM_l^{(i-1)}\end{aligned} \tag{5.13}$$

所有路径可以省略扩展，直接得到译码结果。假设度量值向量和 LLR 软信息满足如下条件

$$PM_l^{(i)} \le PM_{l+1}^{(i)}, \quad 0 \le l < L-1 \tag{5.14}$$

$$\left|L_N^{(i)}[l]\right| \le \left|L_N^{(i+1)}[l]\right|, \quad 0 \le i < n_v\text{-}1 \tag{5.15}$$

对第 i 比特译码时，当公式(5.16)成立时，

$$\sum_{l=0}^{L-1}\left|L_N^{(i)}[l]\right| > L\cdot PM_{L-1}^{(i-1)} - \sum_{l=0}^{L-1} PM_l^{(i-1)} \tag{5.16}$$

由于 $\left|L_N^{(i)}[l]\right| \le \left|L_N^{(i+1)}[l]\right|$，则

$$\sum_{l=0}^{L-1}\left|L_N^{(i+1)}[l]\right| > L\cdot PM_{L-1}^{(i-1)} - \sum_{l=0}^{L-1} PM_l^{(i-1)} \tag{5.17}$$

在这种情况下，索引为$\{i+1, \ldots, n_v-1\}$的比特可以直接被译出。

接下来考虑随着译码阶段 i 的增长，候选路径度量值的变化情况。本文定义路径度量值方差来观测度量值离散情况，度量值方差定义如公式(5.18)所示。

$$\mathrm{var} = \frac{\sum_{k=0}^{L-1}(PM_k^{(i)} - \frac{1}{L}\sum_{l=0}^{L-1} PM_l^{(i)})^2}{L-1} \tag{5.18}$$

根据实验仿真，路径度量值方差随着译码阶段 i 的增长，呈现波动下降的趋势，如图 5-4 所示。在译码的最后阶段，度量值方差保持不变，意味着最后若干比特可以省略路径扩展直接译出，且不会带来任何性能损失。

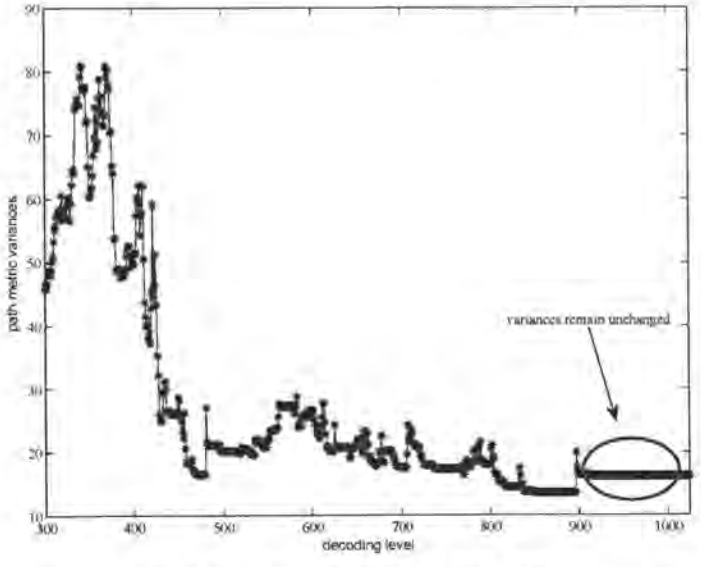

图 5-4：路径度量值方差变化趋势

与固定周期的 min(L-1,n_v)不同[137]，本文提出的优化方法可以自适应调整路径

扩展条件。最好情况下，长度为 n_v 的 Rate-1 节点所需周期数为 1；最差情况下，所需周期数为 n_v，与 SSCL 算法相同[136]。本文统计了一个码块译码中最差情况出现频率，如图 5-5 所示。不同列表长度下最差情况出现频率小于 1%，表明该优化方法可极大降低译码延迟。

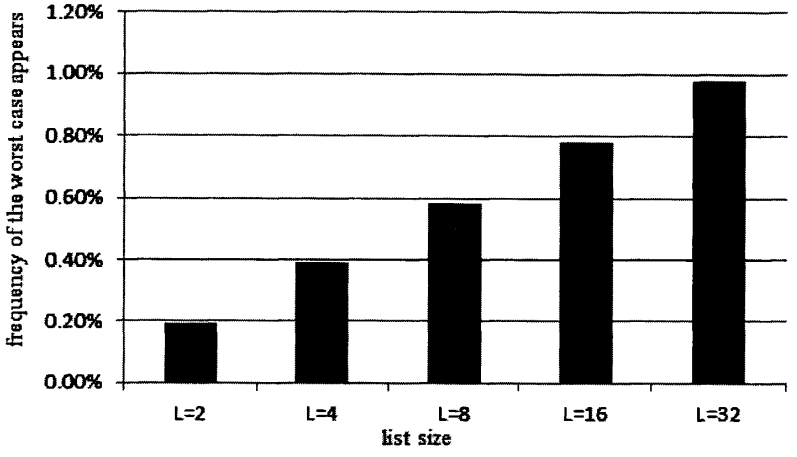

图 5-5：不同列表下一个码块译码中最差情况出现频率（N=1024，R=1/2）

5.4 新型删减策略

删减技术可以降低译码复杂度，一定程度上也会带来性能损失。实际上，我们需要在复杂度和性能方面找到折中。错误概率是评价译码性能的关键指标，在研究删减技术前，首先对错误概率进行分析。

5.4.1 错误概率分析

1. SC 估计错误概率

对第 i 比特进行译码时，估计错误事件可以描述为

$$\begin{aligned}\varepsilon_i &\triangleq \{(u_1^N, y_1^N) \in X^N \times Y^N : u_i \neq L_N^{(i)}(y_1^N, u_1^{i-1})\} \\ &= \{(u_1^N, y_1^N) \in X^N \times Y^N : W_N^{(i)}(y_1^N, u_1^{i-1} | u_i) \\ &\leq W_N^{(i)}(y_1^N, u_1^{i-1} | u_i \oplus 1)\}\end{aligned} \tag{5.19}$$

在文献[125]中，基于巴氏参数的估计错误概率 $P(\varepsilon_i)$ 上限定义如下

$$\begin{aligned} P(\varepsilon_i) &\le \sum_{u_1^N, y_1^N} \frac{1}{2^N} W_N(y_1^N | u_1^N) \sqrt{\frac{W_N^{(i)}(y_1^N, u_1^{i-1} | u_i \oplus 1)}{W_N^{(i)}(y_1^N, u_1^{i-1} | u_i)}} \\ &= Z(W_N^{(i)}) \end{aligned} \tag{5.20}$$

其中，

$$\begin{aligned} & Z(W_N^{(i)}) \\ &= \sum_{y_1^N \in Y^N} \sum_{u_1^{i-1} \in X^{i-1}} \sqrt{W_N^{(i)}(y_1^N, u_1^{i-1} | 0) W_N^{(i)}(y_1^N, u_1^{i-1} | 1)} \end{aligned} \tag{5.21}$$

对于 B-DMC 信道，$Z(W_N^{(i)})$ 很难有效计算出来。本文给出另一种估计错误概率计算方法。

为了论述方便，假设传输码字为全 0 序列，经过 AWGN 信道进行传输。由于 $L_N^{(i)}$ 分布对于 u_i=0 和 u_i=1 是对称的，所以采用全 0 码字进行分析不失一般性[140]。在这种场景下，$L_1^{(i)}(y_i) \sim \mathcal{N}(\frac{2}{\sigma^2}, \frac{4}{\sigma^2})$，其中，$\sigma^2$ 是信道噪声方差。

为了简化描述，本文重新定义公式(5.7)和(5.8)为 $f(a,b) = a \boxplus b$ 和 $g(a,b) = a + b$。根据高斯近似准则[143]，如果一个校验节点的输入概率密度函数（probability density function，pdf）满足对称条件，则输出的 pdf 也近似满足对称条件。对称条件可以描述为 $f(x) = f(-x)e^x$，接收到的信道 LLR 概率密度函数表示为

$$f(x) = \frac{\sigma}{\sqrt{8\pi}} \exp(-\frac{\sigma^2(x - \frac{2}{\sigma^2})^2}{8}) \tag{5.22}$$

由于

$$\begin{aligned} f(-x)e^x &= \frac{\sigma}{\sqrt{8\pi}} \exp(-\frac{\sigma^2(-x - \frac{2}{\sigma^2})^2}{8}) \exp(x) \\ &= \frac{\sigma}{\sqrt{8\pi}} \exp(-\frac{\sigma^2(x - \frac{2}{\sigma^2})^2}{8}) \\ &= f(x) \end{aligned} \tag{5.23}$$

所以，接收到的信道 LLR 满足对称条件。根据高斯近似准则，可以认为 $f(a,b)$ 和 $g(a,b)$ 也近似服从高斯分布。因此，$f(a,b)$ 和 $g(a,b)$ 的中间结果 $L_N^{(i)}$ 也可以认为近似服从高斯分布。

对数似然比递归计算如公式(5.7)和(5.8)所示，其数学期望 $E[L_N^{(i)}]$ 也可以通过递

归计算得到。根据高斯近似准则，我们有

$$E[L_N^{(2i-1)}] = \phi^{-1}(1-(1-\phi(E[L_{N/2}^{(i)}]))^2) \tag{5.24}$$

$$E[L_N^{(2i)}] = 2E[L_{N/2}^{(i)}] \tag{5.25}$$

其中，

$$\phi(x) = \begin{cases} 1-\dfrac{1}{\sqrt{4\pi x}}\displaystyle\int_{-\infty}^{\infty}\tanh\frac{u}{2}e^{-\frac{(u-x)^2}{4x}}dx, & x>0 \\ 1, & x=0 \end{cases} \tag{5.26}$$

对数似然比初始期望值为 $E[L_1^{(1)}] = 2/\sigma^2$。

公式(5.26)可以近似为[144]

$$\phi(x) = \begin{cases} \exp(-0.4527x^{0.86}+0.0218), & 0<x<10 \\ \sqrt{\dfrac{\pi}{x}}\exp(-\dfrac{x}{4})(1-\dfrac{10}{7x}), & x\geq 10 \end{cases} \tag{5.27}$$

第 i 比特估计错误概率可以通过 Q 函数计算得到

$$P(\varepsilon_i) = Q(\sqrt{E[L_N^{(i)}]/2}) \tag{5.28}$$

其中，$Q(x) = \dfrac{1}{\sqrt{2\pi}}\displaystyle\int_x^{+\infty} e^{-\frac{t^2}{2}}dt$。由于 $E[L_N^{(i)}]$ 依赖噪声方差，所以估计错误概率也与 SNR 相关。图 5-6 描述了不同 SNR 下，估计错误概率分布情况(码块长度为 32)。

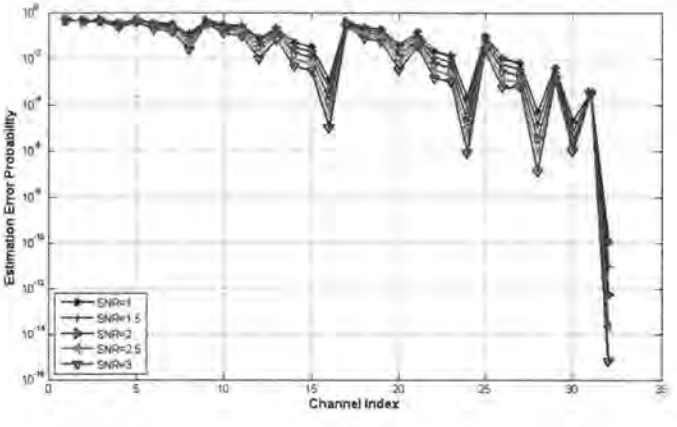

图 5-6：不同 SNR 下估计错误概率分布

2. SCL 路径幸存概率

本节内容为了直观分析路径幸存概率，路径度量值惩罚因子设置为负数，即似然值大的子节点度量值直接继承于父节点，似然值小的子节点度量值减去一个正数。用$\Theta \subseteq \{1,2,...,L_i\}$表示第 i 译码阶段路径索引集合，L_i表示搜索路径数目，$j \in \Theta$。根据度量值定义，可以得到

$$PM_j^{(i)} = -\sum_{\varepsilon_i} \left| L_N^{(i)} \right| \tag{5.29}$$

在没有噪声干扰情况下，估计错误事件ε_i将不会发生，因此正确路径度量值始终为零。

考虑冻结比特译码：当判决正确时，冻结比特能够提高后续软信息判决可靠度，反之，冻结比特适当纠正后续软信息错误。假设冻结比特固定为 0，每个译码阶段度量值增量用Δ_i表示，公式(5.29)可以写为：

$$PM_j^{(i)} = \sum_{t=1}^{i} \Delta_t \tag{5.30}$$

表 5-1 对每个译码阶段度量值增量进行分析。从表中可以看出，具有最大度量值的路径成为正确路径的概率最大。进行第 i 比特译码时，路径正确概率可以表示为

$$\begin{aligned} P_{correct}(i) &= P\{(u_1^N, y_1^N) \in X^N \times Y^N : u_1^i = \hat{u}_1^i\} \\ &= (1-P(\varepsilon_1))(1-P(\varepsilon_2))...(1-P(\varepsilon_i)) \end{aligned} \tag{5.31}$$

表 5-1：路径度量值增量

比特类型	似然比	$\Delta_{most\ likely}$	$\Delta_{contrary}$
信息比特	$L_N^{(i)} \geq 0$	0	$-L_N^{(i)}$
	$L_N^{(i)} < 0$	0	$L_N^{(i)}$
冻结比特	$L_N^{(i)} \geq 0$	0	$-\infty$
	$L_N^{(i)} < 0$	$L_N^{(i)}$	$-\infty$

每条搜索路径成为正确路径的概率称为幸存概率，可以用如下公式来表示：

$$P_j^{(i)}=\begin{cases} P_{correct}(i), & L_j^{(i)}=0 \\ \dfrac{PM_j^{(i)}-\min PM_j^{(i)}}{\beta_i}(1-P_{correct}(i)), & 0<L_j^{(i)}<1 \end{cases} \tag{5.32}$$

其中，

$$\beta_i=\sum(PM_j^{(i)}-\min PM_j^{(i)}) \quad (j\in\{\Theta\backslash\Gamma\}) \tag{5.33}$$

Γ 表示具有最大度量值的路径索引。幸存概率模型基于以下发现得出：当 SNR 增大时，最大度量值路径的幸存概率变大；相应地，最小度量值路径的幸存概率变得越来越小。幸存概率符合归一化定律，即所有路径幸存概率之和为 1。

度量值最大路径和其他候选路径的相对距离用 $\alpha_j^{(i)}$ 表示，其定义如公式(5.34)所示，置信水平为 $P_j^{(i)}$ 情况下置信区间为 $[0,\alpha_j^{(i)}]$。当置信区间上限小于预设阈值时，候选路径幸存概率将会减小。

$$\alpha_j^{(i)}=\frac{\max\limits_{j\in\Theta}PM_j^{(i)}-PM_j^{(i)}}{D_i} \tag{5.34}$$

其中，D_i 表示第 i 比特译码时最大最小度量值差值。

3. 误块率分析

第 i 比特译码时，正确路径被删除的概率为

$$P_e(i)=\sum_{P_j^{(i)}<\eta,j\in\Theta}P_j^{(i)} \tag{5.35}$$

其中，η 表示由置信水平阈值得到的幸存概率阈值，比幸存概率阈值小的路径将被删除。对于长度 $N=2^n$ 的 polar 码，其信息比特长度为 $K=NR$（R 表示码率）。当对冻结比特进行译码时，判决比特总是正确（因为冻结比特已知）。由此，译码最后阶段误块率可以表示为

$$P_e=P_e(1)+(1-P_e(1))P_e(2)+\ldots+(1-P_e(1))(1-P_e(2))\ldots(1-P_e(K-1))P_e(K) \tag{5.36}$$

通常，误差容忍概率设置为 P_{tol}。结合公式(5.35)和(5.36)，幸存概率阈值上限可以表示为

$$\eta<\frac{P_{tol}}{K(L-1)} \tag{5.37}$$

其中，L 表示预设最大搜索路径数目。

5.4.2 删减策略

本文提出的删减策略基于路径度量值及其相对距离。结合幸存概率分析，满足公式(5.38)的候选路径将被删除。

$$\alpha_j^{(i)} > \tau \tag{5.38}$$

其中，τ 表示删减阈值。在置信水平为 $1-P_{tol}$ 下，置信区间为 $[0,\tau]$。当进一步缩小置信区间，更多候选路径将被删除。因此，误差概率随置信区间的减小而增大。处于置信区间以外的候选路径，其幸存概率比较小，删减带来的性能损失可以忽略。

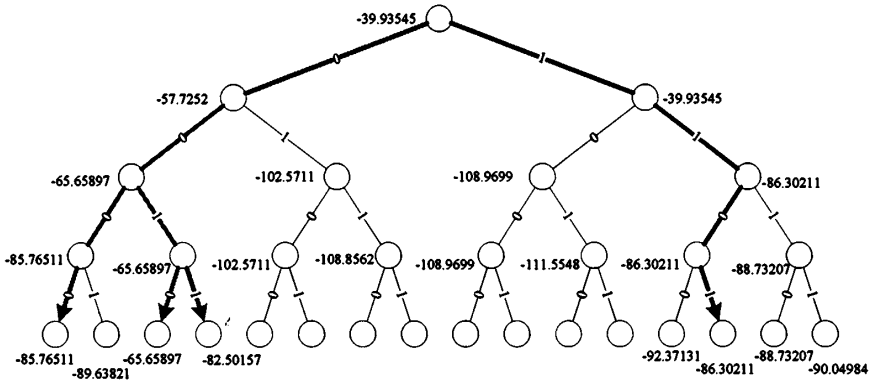

图 5-7：SCL 译码树搜索示例（L=4）

由于噪声等干扰，距离幸存概率最大路径较远的候选路径，也有一定概率成为正确路径。SCL 译码是一个不断向最可能路径收敛的递归过程。在译码的最初阶段，由于估计错误概率相对较大，更多的候选路径需要被保留；在译码的最后阶段，候选路径之间的距离变小。所以，路径度量值最大距离需要增加一个修正项，避免最远路径总是被删除。

引理 5.2：信息比特译码时，候选路径度量值方差随信道索引的增加而减小，候选路径度量值向最大度量值收敛。

证明：图 5-7 描述了一个简单的 SCL 译码树搜索过程，其中，搜索路径数目 L=4。当搜索路径数目达到最大值后，具有较小度量值的扩展路径将被删除。假设某个译码阶段降序排列的度量值表示为 $[a,b,c,d,e,f,g,h]$，度量值范围为 $[h,a]$。经过删减，度量值范围变为 $[d,a]$。接下来的路径扩展，度量值会加上一个增量，路径删减近似在 $[d+\Delta_j^{(i)},a]$ 或 $[d,a+\Delta_j^{(i)}]$ 中进行。与父节点度量值范围 $[h,a]$ 相比，子节点度量值分布更加集中。当极化信道索引 i 变大时，估计错误概率 $P(\varepsilon_i)$ 降低。所以，对数似

然比可靠度提高，度量值增量为 0 的概率变大。当噪声干扰很小时，最大似然路径鲁棒性很强，度量值方差随信道索引的变大而减小。反之，当噪声干扰很大时，度量值增量不稳定，度量值方差波动下降。

冻结比特和信息比特度量值方差如图 5-8 所示。从右边图片可以看出信息比特度量值方差呈下降趋势。在译码的最后几个阶段，度量值方差保持稳定。这是信道极化的结果：信道索引较小时信道容量趋近于 0，较大时趋近于 1。左边图片描述了冻结比特度量值方差随信道索引变大而上升的趋势。这是由于冻结比特译码没有路径分裂，更新后的度量值要么等于前一时刻度量值，要么加上一个增量，最大似然路径与幸存概率最小路径之间的差值越来越大，尤其当 SNR 比较大时，差距比较明显。

证毕。

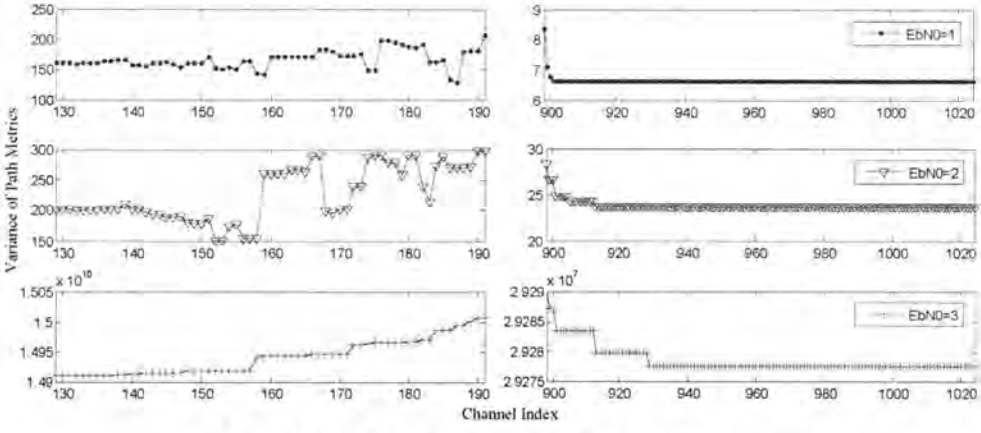

图 5-8：冻结比特和信息比特路径度量值方差变化趋势

第 i 比特译码时，假设具有最小度量值的路径是正确路径（此事件发生概率很小）。译码时，子节点的判决比特与原发送比特不同，这种情况下，该路径的度量值减去 $\left|L_N^{(i)}\right|$。为了修正以上情形引起的估计误差，第 i 阶段路径度量值最大距离增加一个修正项，修正后的 D_i 如公式(5.39)所示。

$$D_i = \max_{j\in\Theta} PM_j^{(i)} - \min_{j\in\Theta} PM_j^{(i)} + mean\left(\left|L_{Nj}^{(i)}\right|\right) \tag{5.39}$$

5.4.3 置信区间

当搜索路径幸存概率 $P_j^{(i)}$ 小于误差概率阈值 η（$0<\alpha_j^{(i)}<1$），该路径将被删除。

因此，我们有

$$\frac{PM_j^{(i)} - \min PM_j^{(i)}}{\beta_i}(1 - P_{correct}(i)) < \eta \tag{5.40}$$

合并公式(5.31)(5.34)(5.37)和(5.40)，相对距离下限表示为

$$\alpha_j^{(i)} > \frac{\max_{j\in\Theta} PM_j^{(i)} - \min_{j\in\Theta} PM_j^{(i)}}{D_i} - \frac{P_{tol}\beta_i}{K(L-1)(1-\prod_{j=1}^{i}(1-P(\varepsilon_j)))D_i} \tag{5.41}$$

在条件$\{P(\varepsilon_i) \neq 0, j \in \Theta\}$下，置信区间上限（删减阈值）为

$$\tau = 1 - \frac{mean(|L_{Nj}^{(i)}|)}{D_i} - \frac{P_{tol}\beta_i}{K(L-1)(1-\prod_{j=1}^{i}(1-P(\varepsilon_j)))D_i} \tag{5.42}$$

公式计算中存在估值，阈值计算不是很精确。本文采用公式与统计分析相结合的方法得到删减阈值。图 5-9 给出了最终幸存路径的起始路径统计概率，所列路径按度量值降序排列。仿真结果与理论分析一致：当 SNR 变大时，估计错误概率降低，相应地，度量值接近最大值的路径，其幸存概率接近 $P_{correct}(i)$，其它路径幸存概率接近于 0。根据实际应用场景调整删减阈值，可以进一步提高译码性能。

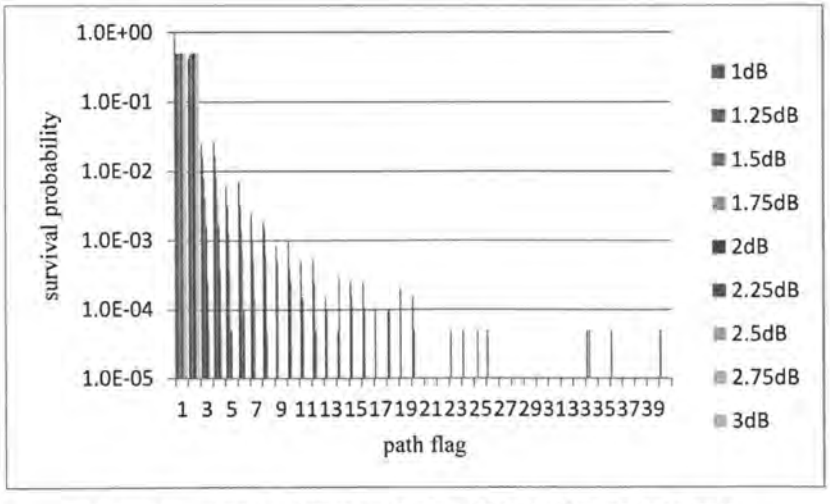

图 5-9：不同信道环境下幸存路径的起始路径统计概率

幸存路径中 99%起始路径来源于具有最大度量值和次大度量值的路径，具有较

小度量值的路径成为幸存路径的概率很小。以 SNR=2dB 为例，起始路径在不同置信区间下的幸存概率如图 5-10 所示。根据统计仿真，L 条候选路径中，六条路径有机会成为幸存路径。具有较小度量值的路径主要分布在区间[0.3,0.6]，其幸存概率在 10^{-4} 到 10^{-5} 之间；具有最大度量值的路径主要分布在区间[0,0.1]，其幸存概率最大。根据误差容忍概率，幸存概率低于 10^{-5} 的候选路径可以被删除。实际上，我们主要关心置信区间上限，统计表明置信区间设置为[0,0.6]，其置信度可以达到 99.999%。删减阈值通过概率统计初步确定，然后根据 BLER 性能进行校正，最终得出图 5-11 所示不同信道状态下的置信区间上限分布。

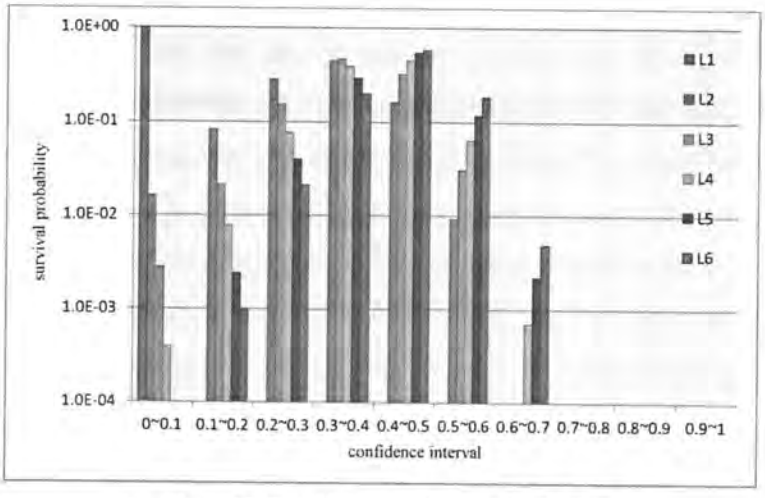

图 5-10：不同置信区间下起始路径幸存概率

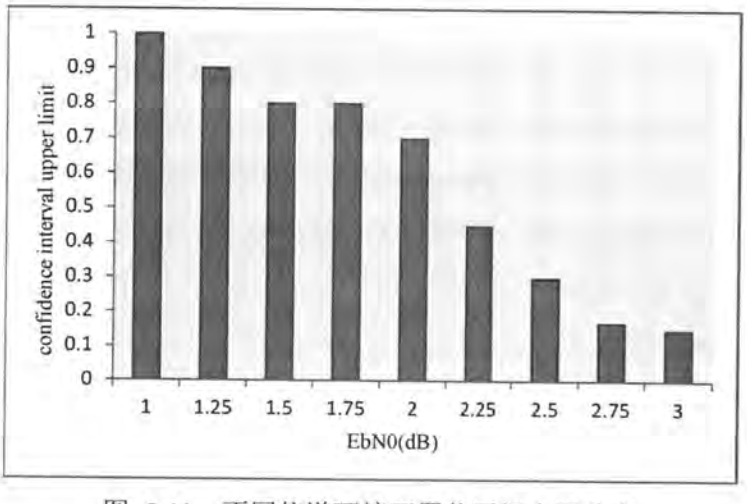

图 5-11：不同信道环境下置信区间上限分布

5.5 仿真结果和性能分析

本节对路径扩展优化方法和新型删减策略进行性能仿真和复杂度对比。仿真基于 AWGN 信道，采用 BPSK 调制，码块长度 N=1024，码率为 1/2。

5.5.1 路径扩展优化仿真结果

不同搜索路径数目下优化 CA-SCL 和传统 CA-SCL 算法的 BLER 性能如图 5-12 所示，其中，CRC 比特数目为 24。由仿真结果可以看出优化后的 CA-SCL 算法与参考性能线基本重合，路径扩展优化没有引起任何性能损失。

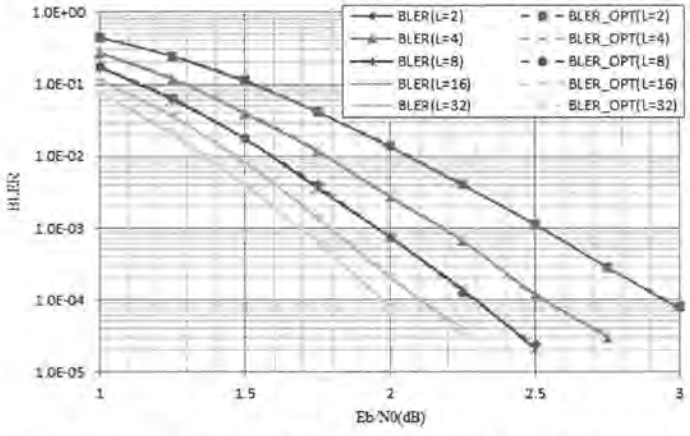

图 5-12：路径扩展优化前后 CA-SCL 算法性能对比

路径扩展优化方法通过避免冗余路径分裂，可以降低路径选择复杂度。现有路径选择解决方案有删减双调排序器（Pruned Bitonic Sorters，PBS）、简化冒泡排序器（Simplified Bubble Sorters，SBS）[145]、奇偶排序器（Odd-Even Sorters，OES）[146]、两步排序器[147]。两步排序器在 $L \leq 32$ 情况下需要的运算部件比较少，但该排序器是基于硬件时空分解，对硬件实现方案依赖性比较强。本文采用 OES 实现变长路径数目排序。OES 将待排序序列分为奇偶两部分分别进行排序，然后将两排序结果进行合并。OES 架构如图 5-13 所示，其中，两端带圆圈的竖线表示一个比较选择单元（Compare-And-Select Unit，CASU）。

本文中满足条件 $\left|L_N^{(l)}[l]\right| > PM_{L-1}^{(i-1)} - PM_l^{(i-1)}$ 的路径不会出现在排序列表中，与该路径相关的 CASU 可以被删除。图 5-13 中，假设 m_{12}、m_{14} 和 m_{16} 不在排序列表中，则灰色虚线所示的 CASU 可以被删除。与标准 OES 相比，减少了 6 个 CASU。本文

统计了不同列表和不同信道环境下平均分裂路径数目，如图 5-14 所示。其中，一个码块中平均分裂路径数目计算公式如(5.43)所示，N 为码块长度，R 为码率，K 为仿真码块数目。

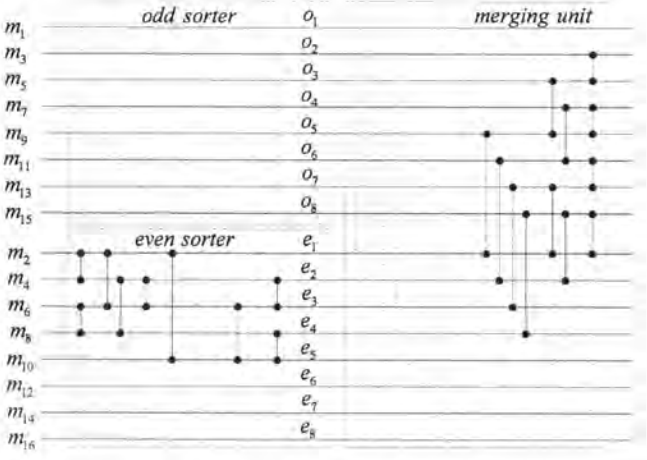

图 5-13：OES 排序器架构

$$L_{aver-num} = \frac{1}{N \cdot RK} \sum_{i=0}^{K-1} \sum_{j=0}^{NR-1} L_{split-num} \tag{5.43}$$

仿真结果表明分裂路径数目与信道环境关系不大。传统 SCL 算法中，L 条路径分裂数目近似为 $2L$，优化后分裂路径数目约为传统算法的一半。分裂路径数目的减少可以有效降低 CASU 数目。

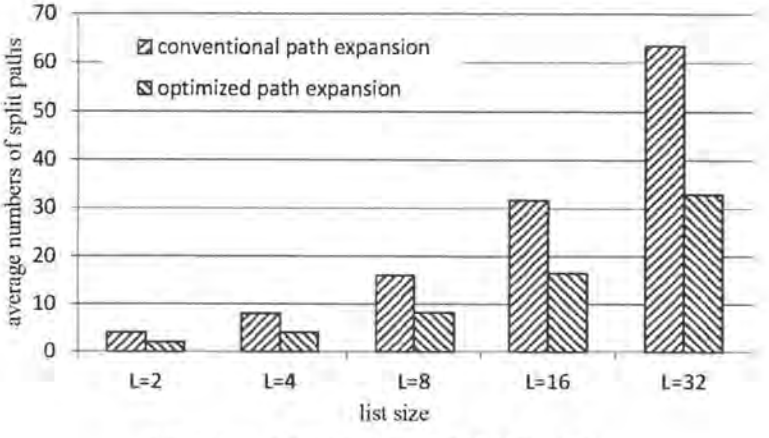

图 5-14：不同 L 下平均路径分裂数目

5.5.2 新型删减策略仿真结果

SC、SCL 和 CA-SCL 算法下 BLER 性能如图 5-15 所示，其中，搜索路径数目 L=32。仿真表明：SCL 算法采用删减技术后，其 BLER 曲线与未删减算法重合，删减带来的性能损失可以忽略。CA-SCL 算法下 BLER 性能有一定损失，这是由于干扰存在，具有最大度量值的候选路径在一定概率上不是正确路径所致。

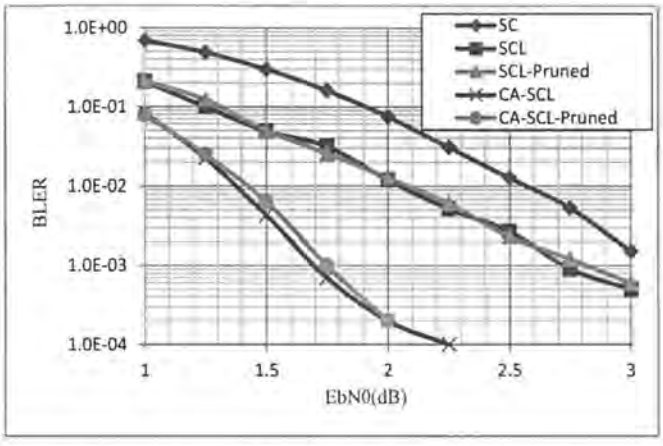

图 5-15：路径删减前后 BLER 性能对比

采用删减技术后，平均搜索路径数目有所降低。SCL 算法和删减 SCL 算法下平均搜索路径数目如图 5-16 所示。删减 SCL 算法中平均搜索路径数目在中 SNR 区下降了 60%，在高 SNR 区下降了 80%。较少的搜索路径意味着较少的 f、g 运算和较少的数据读写操作，可以有效降低译码复杂度。

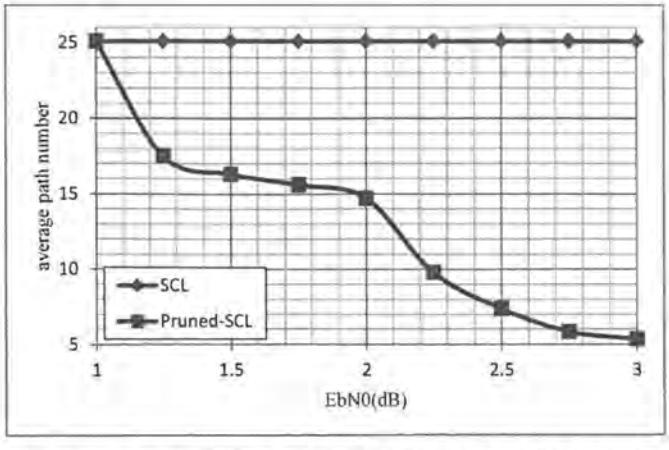

图 5-16：SCL 和删减 SCL 算法下平均搜索路径数目

5.6 小结

SCL 算法被看作 polar 码最有前途的译码算法之一，由于存在路径扩展和删减，导致译码延迟比较大，尤其在搜索路径数目比较大的情况下更加明显。本章从理论上证明了当对数似然比满足$\left|L_N^{(i)}[l]\right| > PM_{L-1}^{(i-1)} - PM_l^{(i-1)}$时，该条路径可以避免路径扩展操作，且没有任何性能损失。实验结果也与理论分析相一致。此外，本章对路径删减进行了深入研究，分析了删减带来的误差概率，提出了基于置信区间的删减方法，并对该方法进行了性能仿真。最后对优化方法进行了复杂度分析。路径扩展优化后，平均路径分裂数目比传统 SCL 算法低 49%；路径删减后，平均搜索路径数目在中 SNR 区下降了 60%，在高 SNR 区下降了 80%，有效降低了译码复杂度。

第6章 总结与展望

6.1 总结

针对通用通信处理器对不适于矢量处理的信道译码算法，本文面向通用通信基带处理器，对协处理器进行研究与设计。首先提出了一种新型二维可配置协处理器架构，将加速引擎以分簇的形式挂接在总线上，降低了总线互连网络功耗和带宽占用比。接下来对协处理器中运算量较大的信道译码加速引擎：turbo 译码器、viterbi 译码器和 polar 译码器进行了研究和设计。针对 turbo 加速引擎退出迭代机制，本文提出了一种二阶差分辅助的 CRC 校验停止准则，极大改善了 turbo 译码器迭代次数。针对支持多标准的高性能 viterbi 译码器，本文进行了算法设计和实现，其较高的吞吐能力适用于高容量网络的译码需求。最后，对基于 SCL 的 polar 译码器路径扩展和路径删减进行了分析和优化，在性能损失可接受范围内，有效降低了译码时延和实现复杂度。本文主要工作和创新点包括以下几个方面：

1. 针对主流通信专用协处理器架构存在互连网络功耗大、协处理器调度频繁等问题，提出了一种面向通用通信基带处理器的二维可配置协处理器架构。通过将加速引擎分簇，并以特定工作模式重新编程加速引擎内部联结关系，使协处理器在灵活性和可靠性方面达到平衡。第一维配置为工作模式和协处理器公共参数配置，由主处理器发起，协处理器实时响应；第二维配置为加速引擎私有参数配置，由主处理器离线完成。通过功耗评估模型，总线互连网络功耗仅为主流通信处理器架构的 1/3；对于无线通信标准数据帧处理，总线带宽占用比由 6.88%降到 2.05%。新型协处理器架构的提出为通信处理器低功耗、低复杂度设计提供了有益探索。
2. 针对在信道传输环境比较差情况下，turbo 译码器迭代多次而性能增益不明显的问题，提出了一种基于二阶差分的 CRC 校验停止准则。该准则通过对传递信息进行二阶差分值计算，可以提前感知信道情况并及时退出迭代。仿真实验表明：与常规 CRC 校验停止准则相比，该方法在信道恶劣情况下，turbo 译码器平均迭代次数下降约 20%。
3. 针对当前多标准 viterbi 译码器数据吞吐不高的问题，设计并实现了一种支持多标准的高性能可配置 viterbi 译码器。该译码器支持的编码约束长度为

5~9，码率为 1/2,1/3,1/4，支持零结尾和咬尾。译码器峰值吞吐为 1.15Gbps@6144bit,600MHz。主流商用 viterbi 译码器 VCP2，数据处理能力为 9.5Mbps@40bit,333MHz，本文中译码器数据处理能力为 32.173Mbps@40bit,333MHz，性能提升约 3.3 倍，可满足日益增长的数据量处理需求。

4. 针对 SCL 算法中路径分裂和路径删减运算复杂度比较高、译码延迟比较大的问题，提出了一种路径扩展优化方法和新型路径删减策略。该优化方法可以避免冗余路径扩展，且不会带来任何性能损失；基于置信区间的路径删减策略，可以有效降低 SCL 译码复杂度。仿真实验表明：路径扩展优化方法可以降低路径分裂数目，最高可达 49%；在性能损失可忽略范围内，新型删减策略可以降低搜索路径数目，在中 SNR 区间降低 60%，高 SNR 区间降低 80%。SCL 优化算法为 polar 译码器低延迟实现提供了有益探索。

6.2 展望

本文对面向通用通信基带处理器的专用协处理器架构和信道译码加速引擎进行了研究，实现了基于 LTE 协议的发送端和接收端协处理器设计。下一代通信系统对硬件灵活度、高可靠性和低功耗提出了更高要求。5G 标准呼之欲出，各国都在加快 5G 商用步伐，谁能尽早推出高性能、高可靠通信处理器，就能在未来通信领域占得先机。未来的研究工作会围绕 5G 关键技术，对通信处理器进一步优化，具体包括：

1. 根据 5G 典型应用场景，扩展协处理器应用范围。5G 用户体验速率为“Gbps”量级，相比较 4G 150Mbps 数据吞吐，对处理器实时性提出了更高要求。5G 标准中信道编码采用了低密度奇偶校验码（Low Density Parity Check，LDPC）和 polar 码，如果通用处理器无法满足实时性要求，需考虑采用协处理器来实现信道编译码。
2. 对协处理器功耗进一步优化，实现更高能效比。与 4G 相比，5G 不仅对传输速率要求更高，还将解决人与人之外的人与物、物与物之间的沟通，即万物互联，对低时延、高可靠、低功耗提出了要求。协处理器面向终端设备，功耗问题是不可忽视的。协处理器设计中，存储器功耗占比比较大，针对这个问题，后期可以考虑对算法进行改进，降低存储器面积；或者优化存储器地址、数据解析逻辑，进一步降低功耗。
3. Polar 码在 2016 年 3GPP 会议上已被确定选为 5G 控制信道编码方案，本文

对 polar 译码算法进行了研究和优化。SCL 算法本质上是多条路径同时进行 SC 运算，传统 SC 算法具有天然串行特性，很难实现并行化。接下来考虑对 SC 算法进行优化，打破数据依赖；在此基础上对 SCL 算法进一步优化。

参考文献

[1] 佚名. 国家集成电路产业发展推进纲要[J]. 中国集成电路, 2014, 23(7):14-16.

[2] Wang D, Du X, Liu Z, et al. MaPU: A novel mathematical computing architecture[C]. IEEE International Symposium on High PERFORMANCE Computer Architecture. IEEE, 2016:457-468.

[3] Han B, Gopalakrishnan V, Ji L, et al. Network function virtualization: Challenges and opportunities for innovations[J]. IEEE Communications Magazine, 2015, 53(2):90-97.

[4] Kuon I, Rose J. Measuring the Gap Between FPGAs and ASICs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2007, 26(2):203-215.

[5] Mehri H, Alizadeh B. An analytical dynamic and leakage power model for FPGAs[C]// Iranian Conference on Electrical Engineering. IEEE, 2014:300-305.

[6] 魏晓云, 陈杰, 曾云. DSP 技术的最新发展及其应用现状[J]. 半导体技术, 2003, 28(9):18-21.

[7] Eyre J, Bier J. The evolution of DSP processors[J]. IEEE Signal Processing Magazine, 2000, 17(2):43-51.

[8] Edfors O, Sandell M, Jan-Jaap V D B, et al. OFDM channel estimation by singular value decomposition[J]. IEEE Transactions on Communications, 1998, 46(7):931-939.

[9] Spencer Q H, Swindlehurst A L, Haardt M. Zero-forcing methods for downlink spatial multiplexing in multiuser MIMO channels[J]. IEEE Transactions on Signal Processing, 2004, 52(2):461-471.

[10]Berrou C, Glavieux A, Thitimajshima P. Near Shannon limit error correcting coding and decoding: Turbo-codes[M]. Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. 1993:1064-1070 vol.2.

[11]Viterbi A. Error bounds for convolutional codes and an asymptotically optimum

decoding algorithm[J]. IEEE Transactions on Information Theory, 1967, 13(2):260-269.

[12]3GPP TS 36.212: "Evolved Universal Terrestrial Radio Access(E-UTRA); Multiplexing and channel coding".

[13]3GPP TS 36.213: "Evolved Universal Terrestrial Radio Access(E-UTRA); Physical layer procedures".

[14]Baumhof C. A New VLSI Vector Arithmetic Coprocessor for the PC[C]// Computer Arithmetic, 1995. Proceedings of the, Symposium on. DBLP, 1995:210-215.

[15]Dang P P. An efficient VLSI architecture for H.264 subpixel interpolation coprocessor[C]// Consumer Electronics, 2006. ICCE '06. 2006 Digest of Technical Papers. International Conference on. IEEE, 2006:87-88.

[16]Beldianu S F, Ziavras S G. ASIC Design of Shared Vector Accelerators for Multicore Processors[J]. 2014:182-189.

[17]Ezer G. Xtensa with User Defined DSP Coprocessor Microarchitectures[C]// IEEE International Conference on Computer Design: Vlsi in Computers & Processors. IEEE Computer Society, 2000:335.

[18]Liang M, Chen J. A DSP-coprocessor architecture for image/video applications [coding/decoding][C]// International Conference on Solid-State and Integrated Circuits Technology, 2004. Proceedings. IEEE Xplore, 2004:1601-1604 vol.3.

[19]Gao L, Guo L, Lu C. Acceleration of MELP Algorithm Using DSP Coprocessor with Extended Registers[C]// Digital System Design, Architectures, Methods and Tools, 2009. Dsd '09. Euromicro Conference on. IEEE Xplore, 2009:659-666.

[20]Shekhar H, Mahto C B, Vasudevan N. FPGA Implementation of Tunable FFT For SDR Receiver[J]. International Journal of Computer Science & Network Security, 2009(5).

[21]Taj M I, Hammami O, Akil M. SDR waveform components implementation on single FPGA multiprocessor platform[C]// IEEE International Conference on Electronics, Circuits, and Systems. IEEE, 2010:790-793.

[22]Palmer J. The Intel®8087 numeric data processor[C]. American Federation of Information Processing Societies: 1980 National Computer Conference, 19-22 May

1980, Anaheim, California, USA. DBLP, 1980:887-893.

[23]TMS320C6670 Multicore Fixed and Floating-Point System-on-Chip data Manual. Texas Instruments Incorporated, 2012.

[24]KeyStone Architecture Fast Fourier Transform Coprocessor (FFTC) User Guide. Texas Instruments Incorporated, 2011.

[25]KeyStone Architecture Turbo Decoder Coprocessor (TCP3d) User Guide. Texas Instruments Incorporated, 2010.

[26]KeyStone Architecture Viterbi Coprocessor (VCP2) User Guide. Texas Instruments Incorporated, 2011.

[27]KeyStone Architecture Turbo Encoder Coprocessor (TCP3e) User Guide. Texas Instruments Incorporated, 2010.

[28]KeyStone Architecture Bit Rate Coprocessor (BCP) User Guide. Texas Instruments Incorporated, 2011.

[29]MSC8157 Reference Manual, Freescale Semiconductor, Inc.,2012.

[30]MAPLE-B3 Baseband Accelerators, Freescale Semiconductor, Inc.,2015.

[31]CEVA-XC4000 product brief. CEVA Incorporated, 2012.

[32]CEVA-XC4500 product brief. CEVA Incorporated, 2014.

[33]Ten Brink S. Convergence behavior of iteratively decoded parallel concatenated codes[J]. IEEE Transactions on Communications, 2001, 49(10):1727-1737.

[34]Ten Brink S. Convergence behavior of iteratively decoded parallel concatenated codes[J]. IEEE Transactions on Communications, 2001, 49(10):1727-1737.

[35]Hagenauer J, Offer E, Papke L. Iterative decoding of binary block and convolutional codes[J]. IEEE Transactions on Information Theory, 1996, 42(2):429-445.

[36]Shao R Y, Lin S, Fossorier M P C. Two simple stopping criteria for turbo decoding[J]. Communications IEEE Transactions on, 1999, 47(8):1117-1120.

[37]Wu Y, Woerner B D, Ebel W J. A simple stopping criterion for turbo decoding[J]. IEEE Communications Letters, 2000, 4(8):258-260.

[38]Ngatched T M N, Takawira F. Simple stopping criterion for turbo decoding[J]. Electronics Letters, 2001, 37(22):1350-1351.

[39]Yu N Y, Min G K, Yong S K, et al. Efficient stopping criterion for iterative decoding of turbo codes[J]. Electronics Letters, 2003, 39(1):73-75.

[40]Hou J, Lee M H, Ju Y P. Adaptive SNR turbo decoding algorithm and stop criterion[C]// International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE Xplore, 2003:893-895.

[41]Li F M, Wu A Y. A new stopping criterion for efficient early termination in turbo decoder designs[C]// International Symposium on Intelligent Signal Processing and Communication Systems. IEEE Xplore, 2006:585-588.

[42]Wu Z, Peng M, Wang W. A new parity-check stopping criterion for turbo decoding[J]. IEEE Communications Letters, 2008, 12(4):304-306.

[43]Gazi O. New early termination methos for turbo decoders[C]// Signal Processing and Communications Applications Conference. 2014:1215-1218.

[44]Shibutani A, Suda H, Adachi F. Complexity reduction of turbo decoding[C]// Vehicular Technology Conference, 1999. Vtc 1999 - Fall. IEEE VTS. IEEE, 1999:1570-1574 vol.3.

[45]Bai C, Jiang J, Zhang P. Hardware implementation of Log-MAP turbo decoder for W-CDMA Node B with CRC-aided early stopping[C]// Vehicular Technology Conference, 2002. Vtc Spring 2002. IEEE. IEEE Xplore, 2002:1016-1019 vol.2.

[46]Zhai F, Fair I J. Techniques for early stopping and error detection in turbo decoding[J]. IEEE Transactions on Communications, 2003, 51(10):1617-1623.

[47]Condo C, Martina M, Piccinini G, et al. Variable Parallelism Cyclic Redundancy Check Circuit for 3GPP-LTE/LTE-Advanced[J]. IEEE Signal Processing Letters, 2014, 21(11):1380-1384.

[48]Kim H, Choi I, Byun W, et al. Distributed CRC Architecture for High-Radix Parallel Turbo Decoding in LTE-Advanced Systems[J]. Circuits & Systems II Express Briefs IEEE Transactions on, 2015, 62(9):1-1.

[49]Kim H, Lee Y, Kim J H. Low-complexity CRC-aided early stopping unit for parallel turbo decoder[J]. Electronics Letters, 2015, 51(21):1660-1662.

[50]Omura J. On the Viterbi decoding algorithm[J]. IEEE Transactions on Information Theory, 1969, IT-15(1):177-179.

[51]Ping S, Yan Y, Feng C. An effective simplifying scheme for Viterbi decoder[J]. IEEE Transactions on Communications, 1991, 39(1):1-3.

[52]Berrou C, Adde P, Angui E, et al. A low complexity soft-output Viterbi decoder architecture[C]// Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on. IEEE Xplore, 1993:737-740 vol.2.

[53]Wang X A, Wicker S B. An artificial neural net Viterbi decoder[J]. IEEE Transactions on Communications, 1996, 44(2):165-171.

[54]Casseau E, Luthi E. Architecture of a high-rate VLSI Viterbi decoder[C]// IEEE International Conference on Electronics, Circuits, and Systems. IEEE Xplore, 1996:21-24 vol.1.

[55]Liu, Xun, Papaefthymiou, et al. Design of a high-throughput low-power IS95 Viterbi decoder[J]. 2003:263-268.

[56]Li Q, You Y, Wang J, et al. VLSI implementation of a high-speed and low-power punctured Viterbi decoder[C]// TENCON '02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. IEEE, 2002:1205-1208 vol.2.

[57]Engin N, Van Berkel K. Viterbi decoding on a coprocessor architecture with vector parallelism[C]// Signal Processing Systems, 2003. SIPS 2003. IEEE Workshop on. IEEE Xplore, 2003:334-339.

[58]Tessier R, Swaminathan S, Ramaswamy R, et al. A reconfigurable, power-efficient adaptive Viterbi decoder[J]. Very Large Scale Integration Systems IEEE Transactions on, 2005, 13(4):484-488.

[59]Sun F, Zhang T. Low-Power State-Parallel Relaxed Adaptive Viterbi Decoder[J]. Circuits & Systems I Regular Papers IEEE Transactions on, 2007, 54(5):1060-1068.

[60]Obeid A M, Garcia A, Petrov M, et al. A multi-path high speed Viterbi decoder[C]// IEEE International Conference on Electronics, Circuits and Systems. IEEE, 2003:1160-1163 Vol.3.

[61]Feygin G, Gulak P G. Architectural tradeoffs for survivor sequence memory management in Viterbi decoders[J]. IEEE Transactions on Communications, 1993,

41(3):425-429.

[62]Cheng C, Parhi K K. Hardware Efficient Low-Latency Architecture for High Throughput Rate Viterbi Decoders[J]. Circuits & Systems II Express Briefs IEEE Transactions on, 2008, 55(12):1254-1258.

[63]Santhi M, Lakshminarayanan G, Sundaram R, et al. Synchronous pipelined two-stage radix-4 200Mbps MB-OFDM UWB Viterbi decoder on FPGA[C]// Soc Design Conference. IEEE, 2009:468-471.

[64]Habib I, Paker Ö, Sawitzki S. Design Space Exploration of Hard-Decision Viterbi Decoding: Algorithm and VLSI Implementation[J]. IEEE Transactions on Very Large Scale Integration Systems, 2010, 18(5):794-807.

[65]Chakraborty D, Raha P, Bhattacharya A, et al. Speed optimization of a FPGA based modified viterbi decoder[C]// International Conference on Computer Communication and Informatics. IEEE, 2013:1-6.

[66]Putra R V W, Adiono T. A configurable and low complexity hard-decision viterbi decoder in VLSI architecture[C]// International Conference on Information and Communication Technology. IEEE, 2014:182-186.

[67]Sugur N V, Siddamal S V, Vemala S S. Design and Implementation of High Throughput and Area Efficient Hard Decision Viterbi Decoder in 65nm Technology[C]// International Conference on Vlsi Design and 2014, International Conference on Embedded Systems. IEEE, 2014:353-358.

[68]Li C, Pan D, Feng Y, et al. Carrier phase estimation scheme for faster-than-Nyquist optical coherent communication systems[J]. 中国光学快报:英文版, 2016, 14(10):100601.

[69]Uenohara H, Aikawa Y. Proposal and analytical investigation of optical comparator for optical and electrical converted Viterbi-decoding scheme[J]. 2016, 52(16).

[70]Wang H, Tan X, Chen F, et al. A Viterbi decoder for UHF RFID digital baseband[C]// IEEE, International Conference on Asic. IEEE, 2015:1-4.

[71]Veshala M, Padmaja T, Ghanta K. FPGA based design and implementation of modified Viterbi decoder for a Wi-Fi receiver[C]// Information & Communication Technologies. IEEE, 2013:525-529.

[72]Lina Alfaridah Z H. Performance Evaluation of K-best Viterbi Decoder for IoT Applications[J]. 2015.

[73]Mostafa K, Hussein A, Youness H, et al. High performance reconfigurable Viterbi Decoder design for multi-standard receiver[C]// National Radio Science Conference. 2016:249-256.

[74]Cakir C, Ho R, Lexau J, et al. Modeling and Design of High-Radix On-Chip Crossbar Switches[C]. International Symposium on Networks-On-Chip. ACM, 2015:20.

[75]Passas G, Katevenis M, Pnevmatikatos D. Crossbar NoCs Are Scalable Beyond 100 Nodes[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2012, 31(4):573-585.

[76]Liu D, Svensson C. Power consumption estimation in CMOS VLSI chips[J]. IEEE Journal of Solid-State Circuits, 1994, 29(6):663-670.

[77]Singh H, Lee M H, Lu G, et al. MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications[J]. Computers IEEE Transactions on, 2000, 49(5):465-481.

[78]Miyamori T, Olukotun K. A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications[C]. FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on. IEEE, 1998:2-11.

[79]Jung C Y, Sunwoo M H, Oh S K. Design of reconfigurable coprocessor for communication systems[C]. Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on. IEEE, 2004:142-147.

[80]Wang X, Zhang S, Ni W, et al. Design of a hybrid reconfigurable coprocessor[J]. Security & Communication Networks, 2013, 6(12):1490–1495.

[81]Tang L, Ambrose J A, Parameswaran S. Reconfigurable pipelined coprocessor for multi-mode communication transmission[C]. Design Automation Conference. ACM, 2013:134.

[82]Harju L, Nurmi J. A Synchronization Coprocessor Architecture for WCDMA/OFDM Mobile Terminal Implementations[C]// International Symposium on System-On-Chip, 2005. Proceedings. IEEE, 2005:141-145.

[83]Xydis S, Pekmestzi K, Soudris D, et al. High-Level Synthesis Methodologies for Delay-Area Optimized Coarse-Grained Reconfigurable Coprocessor Architectures[C]// IEEE Symposium on Vlsi. IEEE Computer Society, 2010:486-487.

[84]Marculescu R, Ogras U Y, Peh L S, et al. Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2009, 28(1):3-21.

[85]黄岗. MPSoC 互连网络功耗模型及其应用[D]. 西安电子科技大学, 2011.

[86]Sha L, Abdelzaher T, Årzén K E, et al. Real Time Scheduling Theory: A Historical Perspective[J]. Real-Time Systems, 2004, 28(2):101-155.

[87]Jerraya A, Tenhunen H, Wolf W. Multiprocessor Systems-on-Chips[C]// IEEE Computer Society Symposium on Emerging Vlsi Technologies and Architectures. IEEE Computer Society, 2006:4.

[88]Kogel T, Meyr H. Heterogeneous MP-SoC: the solution to energy-efficient signal processing[C]. Design Automation Conference, 2004. Proceedings. IEEE, 2004:686-691.

[89]Mandwale A J, Mulani A O. Different Approaches For Implementation of Viterbi decoder on reconfigurable platform[C]. International Conference on Pervasive Computing. IEEE, 2015:1-4.

[90]Hsu J M, Wang C L. A parallel decoding scheme for turbo codes[C]// IEEE International Symposium on Circuits and Systems. IEEE Xplore, 1998:445-448 vol.4.

[91]Raouafi F, Dingninou A, Berrou C. Saving memory in turbo-decoders using the max-log-MAP algorithm[C]// Iee Colloquium Turbo Codes in Digital Broadcasting-could It Double Capacity. IEEE Xplore, 1999:14/1-14/4.

[92]Pearl J. Probabilistic reasoning in intelligent systems[J]. Computer Science Artificial Intelligence, 1988, 70(14):521–538.

[93]Kschischang F R, Frey B J, Loeliger H A. Factor graphs and the sum-product algorithm[J]. IEEE Transactions on Information Theory, 2001, 47(2):498-519.

[94]Pradhan A K, Nandy S K. A reconfigurable viterbi decoder for SDR and mobile

communications[C]// International Conference on High PERFORMANCE Computing and Applications. IEEE, 2014:1-6.

[95] Tran T H, Kato H, Takamaeda-Yamazaki S, et al. Performance evaluation of 802.11ah Viterbi decoder for IoT applications[C]// International Conference on Advanced Technologies for Communications. IEEE, 2015:320-325.

[96] Zafra S O, Pang X, Jacobsen G, et al. Phase noise tolerance study in coherent optical circular QAM transmissions with Viterbi-Viterbi carrier phase estimation[J]. Optics Express, 2014, 22(25):30579-30585.

[97] Annamalai P, Bapat J, Das D. Coverage Enhancement for MTC Devices using Reduced Search Viterbi Decoder Across RATs[J]. 2016, 20(9):1-1.

[98] Peng H, Liu R, Hou Y, et al. A Gb/s Parallel Block-based Viterbi Decoder for Convolutional Codes on GPU[J]. 2016.

[99] Bickerstaff M, Garrett D, Prokop T, et al. A unified tur-bo/viterbi channel decoder for 3GPP mobile wireless in 0.18 /spl mu/m CMOS[C]// Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International. IEEE, 2002:124-451 vol.1.

[100] Abdallah R A, Leey S J, Goel M, et al. Low-power pre-decoding based viterbi decoder for tail-biting convolutional codes[J]. 2009:185-190.

[101] He J, Liu H, Wang Z, et al. High-Speed Low-Power Viterbi Decoder Design for TCM Decoders[J]. IEEE Transactions on Very Large Scale Integration Systems, 2012, 20(20):755-759.

[102] Anders M A, Mathew S K, Hsu S K, et al. A 1.9 Gb/s 358 mW 16–256 State Reconfigurable Viterbi Accelerator in 90 nm CMOS[J]. IEEE Journal of Solid-State Circuits, 2008, 43(1):214-222.

[103] 夏飞, 聂晶, 李荣春,等. 现场可编程门阵列参数化多标准高吞吐率基 4 Viterbi 译码器[J]. 国防科技大学学报, 2016(1):86-92.

[104] Forney G D J. The viterbi algorithm[J]. Proceedings of the IEEE, 1993, 61(5):268-278.

[105] Forney G D. Review of random tree codes[J]. NASA Ames Res. Cen., Moffett Field, Calif., Contract NAS2-3637, NASA CR 73176, Final Rep., Dec. 1967,

appendix A.

[106] Han J S, Kim T J, Lee C. High performance Viterbi decoder using modified register exchange methods[C]// International Symposium on Circuits and Systems. IEEE Xplore, 2004:III-553-6 Vol.3.

[107] El-Dib D A, Elmasry M I. Modified register-exchange Viterbi decoder for low-power wireless communications[J]. Circuits & Systems I Regular Papers IEEE Transactions on, 2004, 51(2):371-378.

[108] Shieh M D, Wang T P, Yang D W. Low-power register-exchange survivor memory architectures for Viterbi decoders[J]. 2009, 3(2):83-90.

[109] Muhammad B A, Zanna M A, Mohammed D A, et al. Low complexity FPGA implementation of Register Exchange Based Viterbi decoder[C]// IEEE International Conference on Emerging & Sustainable Technologies for Power & ICT in A Developing Society. IEEE, 2013:21-25.

[110] Lee J H, Park W H, Moon J H, et al. Efficient DSP architecture for Viterbi decoding with small trace back latency[C]// Circuits and Systems, 2004. Proceedings. The 2004 IEEE Asia-Pacific Conference on. IEEE, 2006:129-132 vol.1.

[111] Truong T K, Shih M T, Reed I S, et al. A VLSI design for a trace-back Viterbi decoder[J]. IEEE Transactions on Communications, 1992, 40(3):616-624.

[112] Wicker S B. Error control systems for digital communication and storage[M]. Prentice-Hall, Inc. 1994.

[113] Pribylov V P, Plyasunov A /. A convolutional code decoder design using Viterbi algorithm with register exchange history unit[C]. IEEE International Siberian Conference on Control and Communications. IEEE Xplore, 2005:13-18.

[114] The Xilinx LogiCORE: Viterbi Decoder v3.0. Product Specification DS247 (v1.0), March 28, 2003.

[115] Cypher R, Shung C B. Generalized trace back techniques for survivor memory management in the Viterbi algorithm[J]. Journal of Signal Processing Systems, 1993, 5(1):85-94.

[116] Manzoor R, Rafique A, Bajwa K B. VLSI Implementation of an Efficient Pre-Trace Back Approach for Viterbi Algorithm[J]. 2007:27-30.

[117] Gang Y, Erdogan A T, Arslan T. An Efficient Pre-Traceback Architecture for the Viterbi Decoder Targeting Wireless Communication Applications[J]. Circuits & Systems I Regular Papers IEEE Transactions on, 2006, 53(9):1918-1927.

[118] Kamuf M, ÖWall V, Anderson J B. Survivor Path Processing in Viterbi Decoders Using Register Exchange and Traceforward[J]. Circuits & Systems II Express Briefs IEEE Transactions on, 2007, 54(6):537-541.

[119] Rhee M Y. Error-correcting coding theory[M]. McGraw-Hill, 1989.

[120] Cox R V, Sundberg C E W. An efficient adaptive circular Viterbi algorithm for decoding generalized tailbiting convolutional codes[J]. Vehicular Technology IEEE Transactions on, 1994, 43(1):57-68.

[121] Shao R Y, Lin S, Fossorier M P C. Two decoding algorithms for tailbiting codes[J]. IEEE Transactions on Communications, 2003, 51(10):1658-1665.

[122] Chen T T, Tsai S H. Reduced-complexity wrap-around Viterbi algorithm for decoding tail-biting convolutional codes[C]// Wireless Conference, 2008. Ew 2008. European. IEEE, 2008:1-6.

[123] Black P J, Meng T H. A 140-Mb/s, 32-state, radix-4 Viterbi decoder[J]. 1993, 27(12):1877-1885.

[124] Bruels N, Sicheneder E, Loew M, et al. A 2.8 Gb/s, 32-state, radix-4 Viterbi decoder add-compare-select unit[C]// VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on. IEEE, 2004:170-173.

[125] Arikan E. Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels[J]. IEEE Transactions on Information Theory, 2009, 55(7):3051-3073.

[126] E. Arikan, Polar codes: A pipelined implementation, in Proc. 4th Int. Symp. on Broad. Commun. ISBC 2010, pp. 11-14, July 2010.

[127] Tal I, Vardy A. List decoding of polar codes[C]. IEEE International Symposium on Information Theory Proceedings. IEEE, 2012:1-5.

[128] Li B, Shen H, Tse D. An Adaptive Successive Cancellation List Decoder for Polar Codes with Cyclic Redundancy Check[J]. IEEE Communications Letters, 2012, 16(12):2044-2047.

[129] Balatsoukas-Stimming A, Raymond A J, Gross W J, et al. Hardware Architecture for List Successive Cancellation Decoding of Polar Codes[J]. Circuits and Systems II Express Briefs IEEE Transactions on, 2014,61(8):609-613.

[130] Yuan B, Parhi K K. Low-Latency Successive-Cancellation List Decoders for Polar Codes With Multibit Decision[J]. Very Large Scale Integration Systems IEEE Transactions on, 2014, 23(10):2268-2280.

[131] Alexios Balatsoukas-Stimming, Mani Bastani Parizi, Andreas Burg. LLR-Based Successive Cancellation List Decoding of Polar Codes[J]. IEEE Transactions on Signal Processing, 2015, 63(19):5165-5179.

[132] Lin J, Xiong C, Yan Z. A High Throughput List Decoder Architecture for Polar Codes[J]. IEEE Transactions on Very Large Scale Integration Systems, 2016, 24(6):2378-2391.

[133] Alamdar-Yazdi A, Kschischang F R. A Simplified Successive-Cancellation Decoder for Polar Codes[J]. IEEE Communications Letters, 2011, 15(12):1378-1380.

[134] Sarkis G, Gross W J. Increasing the Throughput of Polar Decoders[J]. IEEE Communications Letters, 2013, 17(4):725-728.

[135] Sarkis G, Giard P, Vardy A, et al. Fast Polar Decoders: Algorithm and Implementation[J]. IEEE Journal on Selected Areas in Communications, 2014, 32(5):946-957.

[136] Hashemi S A, Condo C, Gross W J. A Fast Polar Code List Decoder Architecture Based on Sphere Decoding[J]. IEEE Transactions on Circuits & Systems I Regular Papers, 2016, 63(12):2368-2380.

[137] Hashemi S A, Condo C, Gross W J. Fast Simplified Successive-Cancellation List Decoding of Polar Codes[C]// Wireless Communications and NETWORKING Conference Workshops. IEEE, 2017:957-960.

[138] Chen K, Niu K, Lin J. A Reduced-Complexity Successive Cancellation List Decoding of Polar Codes[C]. Vehicular Technology Conference. IEEE, 2014:1-5.

[139] Chen K, Li B, Shen H, et al. Reduce the Complexity of List Decoding of Polar Codes by Tree-Pruning[J]. IEEE Communications Letters, 2015, 20(2):204-207.

[140] Zhang Z, Zhang L, Wang X, et al. A Split-Reduced Successive Cancellation List

Decoder for Polar Codes[J]. IEEE Journal on Selected Areas in Communications, 2015, 34(2):292-302.

[141] Arikan E. Channel combining and splitting for cutoff rate improvement[J]. IEEE Transactions on Information Theory, 2006, 52(2):628-639.

[142] Niu K, Chen K. CRC-Aided Decoding of Polar Codes[J]. IEEE Communications Letters, 2012, 16(10):1668-1671.

[143] Chung S Y, Richardson T J, Urbanke R L. Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation[J]. IEEE Transactions on Information Theory, 2002, 47(2):657-670.

[144] Wu D, Li Y, Sun Y. Construction and Block Error Rate Analysis of Polar Codes Over AWGN Channel Based on Gaussian Approximation[J]. IEEE Communications Letters, 2014, 18(7):1099-1102.

[145] Balatsoukas-Stimming A, Parizi M B, Burg A. On metric sorting for successive cancellation list decoding of polar codes[C]// IEEE International Symposium on Circuits and Systems. IEEE, 2015:1993-1996.

[146] Kong B Y, Yoo H, Park I C. Efficient Sorting Architecture for Successive-Cancellation-List Decoding of Polar Codes[J]. IEEE Transactions on Circuits & Systems II Express Briefs, 2016, 63(7):673-677.

[147] Bioglio V, Gabry F, Godard L, et al. Two-Step Metric Sorting for Parallel Successive Cancellation List Decoding of Polar Codes[J]. IEEE Communications Letters, 2017, 21(3):456-459.

致谢

时光荏苒，转眼间来到自动化所已经五个多年头了。五年间，稚嫩、懵懂悄悄褪去，成熟、自信爬上脸庞。科研对于我，不光是对某一领域有了认识和理解；更多地，通过解决问题，精益求精，对生活有了更深的感悟。

古人云“君子博学而日参省乎己”。思考，使我们明确对生活的预期，驱使我们寻求解决问题的方法。博学，使我们更加了解生活的常态，执着而不执念，处变不惊。生活，以她独特的方式，给我们抛出一道道考题。通过考验的人，也会收获甜美的奖品。

博士论文完成过程中，发生了一些波折。现在回首，这些波折更像是一个人走向睿智的历练。感谢老师和同学们的无私帮助，感谢父母对我的鼓励和支持。

首先感谢我的导师王东琳研究员。王老师是处理器设计领域的专家，有着丰富的工程经验和敏锐的洞察力，对学术要求严谨，是我们学习的楷模。生活之中，王老师更像一个大家长，以他亲身经历，教导我们好好学习，好好生活。

感谢通信事业部的王晓琴老师和郭晨老师。王晓琴老师在专用集成电路设计和无线通信技术领域有多年积累，她引导我认识了 LTE、认识了 5G。感谢郭晨老师在通信算法领域给予了我很多启发，并对学位论文的撰写提出了宝贵意见。

感谢集成中心党支部书记马小军老师。马老师是我的硕士导师，在生活中给予了我很多帮助。在我遇到困难时，主动打电话开导我，并动员其他人帮助我。感谢马老师。

感谢集成中心的张志伟老师、谢少林老师、杜学亮老师、刘子君老师、蒿杰老师、韩华老师，感谢自动化所研究生部的老师们。他们都在学习和生活中给予我无私的帮助，是我科研路上的强大后盾和支援团，深深感谢他们。

五年学习生活，离不开日夜相伴的同窗。我们一起加班到很晚，我们一起去郊游，我们一起去聚餐。科研的日子，有了他们的陪伴，更加五彩斑斓。同窗数载凝聚无数美好瞬间，将永远铭刻在我的记忆中。感谢他们。

求学路上，父亲的那句话一直在我耳边回响：只要你们愿意读书，借钱也会供你们上学。父母辛苦工作，供我和弟弟上学。如今，弟弟已从北京交通大学硕士毕

业，在北京找到了很好的工作。父母很欣慰，依然在鼓励我好好学习。看到父母头上出现的斑驳白发，我很心疼，唯有更加勤奋努力，才能回报他们。

最后，衷心感谢各位评审专家在百忙之中对论文进行评审和指导，您的宝贵意见将是我以后努力的方向！

2017 年 11 月

个人简历

基本情况

姓　　名：赵旭莹　　性别：女

出生日期：1988 年 9 月 7 日　　籍贯：河北定州

教育经历

2012.09 – 2017.12：中国科学院自动化研究所　计算机应用技术　硕博连读

2008.09 – 2012.07：　河北大学　电气工程及其自动化　学士学位

攻读博士学位期间主要科研项目

1. 2013.06-2015.10：在中国科学院自动化研究所国家专用集成电路设计工程技术研究中心进行中国科学院战略性先导科技专项“代数处理器芯片（课题编号 XDA06011000）”项目研究，主要负责 turbo 译码器设计。
2. 2015.10-2016.03：在中国科学院自动化研究所国家专用集成电路设计工程技术研究中心进行国家科技支撑计划“代数处理器在公众移动通信领域应用研究（课题编号 2014BAH32B00）”项目研究，主要负责 LTE 物理层比特级算法设计和实现。
3. 2016.04-至今：在中国科学院自动化研究所国家专用集成电路设计工程技术研究中心进行军用电子元器件型谱系列科研项目“自主创新极光微处理器及技术平台（课题编号 Y4W1011GB1）”项目研究，主要负责协处理器架构设计。

获奖情况

1. 荣获中国科学院大学“优秀学生干部”称号
2. 荣获中国科学院大学“三好学生”称号
3. 荣获中国科学院大学“梦想有我 青春先行”演讲比赛三等奖
4. 多次获得中国科学院大学学业奖学金

攻读博士学位期间研究成果

会议论文

[1] **Zhao X**, Wang X, Wang D. Second Order Difference Aided CRC Check Stopping Criterion for Turbo Decoding[C]// International Conference on Mechatronics and Intelligent Robotics. Springer, Cham, 2017:30-35. (**EI**)

[2] **Xuying Zhao**, Huan Li, Xiaoqin Wang. "A High Performance Multi-standard Viterbi Decoder," IEEE 7th International Conference on Electronics Information and Emergency Communication(ICEIEC), ShenZhen, 2017, pp. 1-4. (**EI**)

[3] Huan Li, **Xuying Zhao**, Chen Guo, et al. "A High-Parallelism Detection Algorithm for Massive MIMO Systems," IEEE 7th International Conference on Electronics Information and Emergency (ICEIEC), ShenZhen, 2017, pp. 83-86. (**EI**)

[4] Huan Li, **Xuying Zhao**, Chen Guo, XiaoQin Wang. "A Low-Complexity Detection Method Based on Iteration for Massive MIMO Systems," IEEE International Conference on Communication Software and Networks (ICCSN), GuangZhou, 2017, pp. 487-491. (**EI**)

期刊论文

[1] **赵旭莹**, 李桓, 王晓琴, 王东琳. "基于滑窗流水的高性能可配置 viterbi 译码器," 微电子学与计算机[J], 2017. （**中文核心**，已录用）

[2] **赵旭莹**, 李桓, 王晓琴, 王东琳. "通信专用新型二维可配置协处理器架构研究," 哈尔滨工程大学学报[J]. (**EI**，under review)

[3] **Xuying Zhao**, Dongjun Ma, Huan Li, Chen Guo, Donglin Wang. "Variable-Latency List Decoding of Polar Codes Based on Optimized Path Expansion," IEEE Communications Letters[J]. (**SCI**，under review)

发明专利

[1] **赵旭莹**，王晓琴，吴军宁，田燕. 一种基四算法下的加比选计算方法和装置. 公开号：CN105162474A，公开日期：2015.12.16.

[2] **赵旭莹**，王晓琴，林啸. 一种多粒度并行解速率匹配方法和装置. 公开号：CN105187162A，公开日期：2015.12.23.

[3] 王晓琴，**赵旭莹**，吴军宁. 一种 Turbo 迭代译码方法和译码装置. 公开号：CN104579369A，公开日期：2015.04.29.

[4] 吴军宁，**赵旭莹**，吴义如，董佳佳，王晓琴. 一种 Turbo 译码器的位宽非对称仿存接口，公开号：CN104702294A，公开日期：2015.06.10.

[5] 王戈，郭晨，**赵旭莹**，郭璟，李桓，王晓琴. PUSCH 信道检测 ACK/NACK 状态的方法及装置. 公开号：CN105847199A，公开日期：2016.08.10.

[6] 王晓琴，张森，**赵旭莹**，吴军宁，郭晓龙，林啸，郭璟，王伟康. 一种 LTE 系统速率匹配的并行实现方法和装置. 公开号：103929271B，公开日期：2017.04.19.

[7] 吴军宁，王晓琴，郭晓龙，**赵旭莹**，林啸，张森，郭璟，王伟康. 一种 LTE 系统中伪随机序列并行生成方法. 公开号：103873181B，公开日期：2017.01.18.

[8] 吴军宁，王晓琴，张森，**赵旭莹**，林啸，郭晓龙，郭璟，王伟康. 一种用于 LTE 系统 Turbo 码内交织的并行实现方法及装置. 公开号：103888224B，公开日期：2017.05.10.

[9] 郭晓龙，王晓琴，吴军宁，郭璟，王伟康，林啸，**赵旭莹**，张森. 一种 LTE 系统资源映射的向量化实现方法和装置. 公开号：103957090B，公开日期：2017.05.24.

[10] 郭晓龙，王晓琴，王伟康，吴军宁，林啸，郭璟，张森，**赵旭莹**. 一种面向可编程代数处理器的矩阵乘法计算装置及方法. 公开号：103902507B，公开日期：2017.05.10.

与项目相关的技术文档

[1] **赵旭莹**. 无线通信接收端协处理器整体架构设计文档, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2016.

[2] **赵旭莹**. 无线通信接收端协处理器用户手册, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2016.

[3] **赵旭莹**. 高性能可配置 viterbi 译码器设计文档, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2015.

[4] **赵旭莹**. 高性能可配置 viterbi 译码器覆盖率分析报告, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2015.

[5] **赵旭莹**. 解速率匹配&HARQ 合并协处理器设计文档, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2016.

[6] **赵旭莹**. 解速率匹配&HARQ 合并协处理器覆盖率分析报告, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2016.

[7] **赵旭莹**. 无线通信接收端协处理器测试报告, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2016.

[8] 王晓琴, 吴军宁, **赵旭莹**. MaPU 专用 turbo 译码指令实现说明, 中国科学院自动化研究所国家专用集成电路设计技术研究中心, 2014.

附录-中英文术语对照表

（按英文简写首字母排序，数字排在最前面）

英文简写	英文全称	中文释义
3GPP	the 3rd Generation Partnership Project	第三代合作伙伴计划
ACS	Add-Compare-Select	加比选
ASIC	Application Specific Integrated Circuit	专用集成电路
AWGN	Additive White Gaussian Noise	加性高斯白噪声
BER	Bit Error Rate	误码率
BLER	Block Error Rate	误块率
BP	Belief Propagation	置信传递
BPSK	Binary Phase Shift Keying	二进制相移键控
B-DMC	Binary-input Discrete Memoryless Channel	二进制输入离散无记忆信道
BSC	Binary Symmetric Channel	二进制对称信道
CCE	Convolutional Code Encoding	卷积码编码
CDMA	Code Division Multiple Access	码分多址
CIM	Channel Interleaver Module	信道交织模块
CMOS	Complementary Metal Oxide Semiconductor	互补金属氧化物半导体
CPU	Central Processing Unit	中央处理器
CRC	Cyclic Redundancy Check	循环冗余校验
CSCP	Communication Specific CoProcessor	通信专用协处理器
DC	Design Compiler	逻辑综合工具
DeRM	DeRate Matching	解速率匹配
DFT	Discrete Fourier Transform	离散傅里叶变换
DSP	Digital Signal Processor	数字信号处理器
DUT	Design Under Test	待测模块
DVB	Digital Video Broadcasting	数字视频广播
DVD	Digital Video Disc	数字视频光盘

FDD	Frequency Division Duplexing	频分双工
FFT	Fast Fourier Transformation	快速傅里叶变换
FPGA	Field-Programmable Gate Array	现场可编程门阵列
GPP	General Purpose Processor	通用处理器
GPU	Graphics Processing Unit	图形处理器
GSM	Global System for Mobile Communication	全球移动通信系统
HARQ	Hybrid Automatic Repeat reQuest	混合自动重传请求
ICW	Initial Configuration Word	初始配置字
IP	Intellectual Property	知识产权
ITRS	International Technology Roadmap for Semiconductors	国际半导体技术发展线路图
LDPC	Low Density Parity Check	低密度奇偶校验码
LLR	Log-Likelihood Ratio	对数似然比
LTE	Long Term Evolution	长期演进
MAC	Media Access Control Layer	媒体接入控制层
MAP	Maximum A Posteriori	最大后验概率
MIMO	Multiple Input Multiple Output	多输入多输出
MPU	Microcode Processing Unit	微码处理单元
OFDM	Orthogonal Frequency Division Multiplexing	正交频分复用
PBCH	Physical Broadcast Channel	物理广播信道
PDCCH	Physical Downlink Control Channel	物理下行控制信道
PDSCH	Physical Downlink Shared Channel	物理下行共享信道
PE	Processing Engine	处理引擎
PMCH	Physical Multicast Channel	物理多媒体信道
PUSCH	Physical Uplink Shared Channel	物理上行共享信道
QAM	Quadrature Amplitude Modulation	正交振幅调制
QPSK	Quadrature Phase Shift Keying	正交相移键控
RE	Register Exchange	寄存器交换
RISC	Reduced Instruction Set Computing	精简指令集计算机
RM	Rate Matching	速率匹配
RSC	Recursive Systematic Convolutional	递归系统卷积码

RTL	Register Transfer Level	寄存器传输级
RxCP	Receiver CoProcessor	接收端协处理器
SC	Successive Cancellation	连续消除
SCL	Successive Cancellation List	连续消除列表
SIMD	Single Instruction Multiple Data	单指令多数据
SMC	Scrambling Modulation Coprocessor	加扰调制协处理器
SOC	System On Chip	片上系统
SOVA	Soft Output Viiterbi Algorithm	软输出 viterbi 译码算法
SPG	Survivor Path Generation	幸存路径生成
SQNR	Signal-to-Quantization Noise Ratio	信号量化噪声比
SRAM	Static Random Access Memory	静态随机存取存储器
TB	Trace Back	回溯
TBC	Trellis Branch Compute	网格分支计算
TDC	Turbo Decoder Coprocessor	Turbo 译码协处理器
TDD	Time Division Duplexing	时分双工
TEC	Turbo Encoding Coprocessor	Turbo 编码协处理器
TI	Texas Instruments	德州仪器
TTI	Transmission Time Interval	传输时间间隔
TxCP	Transmission CoProcessor	发送端协处理器
UCP	Universal Communication Processor	通用通信处理器
VDC	Viterbi Decoder Coprocessor	viterbi 译码协处理器
VLSI	Very Large Scale Integration	超大规模集成电路
WiMAX	Worldwide Interoperability for Microwave Access	全球微波互联接入
WLAN	Wireless Local Area Networks	无线局域网络