

学校代码: 10286  
分类号: TP302  
密 级: 公开  
U D C: 62  
学 号: 089207



东南大学  
SOUTHEAST UNIVERSITY  
博士学位论文

分组密码可重构处理器的混合寄存器  
文件架构研究

研究生姓名: 葛伟

导师姓名: 时龙兴

申请学位类别 工学博士 学位授予单位 东南大学

一级学科名称 电子科学与技术 论文答辩日期 2015年11月12日

二级学科名称 微电子与固体电子学 学位授予日期 20 年 月 日

答辩委员会主席 陈军宁 评 阅 人 盲 审

2015年11月13日



東南大學  
博士学位论文

分组密码可重构处理器的混合寄存器文件  
架构研究

专业名称: 微电子学与固体电子学

研究生姓名: 葛伟

导师姓名: 时龙兴



# Hybrid Register File Architecture Research for Reconfigurable Block Cipher Processor

A Dissertation Submitted to  
Southeast University  
For the Academic Degree of Doctor of Engineering

BY  
GE Wei

Supervised by  
Professor SHI Longxing

School of Electronic Science and Engineering  
Southeast University

November 2015



## 东南大学学位论文独创性声明

本人声明所呈交的学位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，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。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的说明并表示了谢意。

研究生签名: \_\_\_\_\_ 日期: \_\_\_\_\_

## 东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名: \_\_\_\_\_ 导师签名: \_\_\_\_\_ 日期: \_\_\_\_\_



## 摘要

针对特殊领域中密码算法动态重构的需求，粗粒度可重构处理器与专用密码电路 ASIC 和定制指令集处理器 ASIP 相比，在性能和灵活性等关键指标方面具有更好的表现，已经成为相关领域的重要实现手段。其中，寄存器文件的带宽限制、访存延时和映射方法成为影响分组密码算法可重构实现的设计瓶颈，本文重点研究了高面积效率（性能面积比）的分组密码可重构处理器中的寄存器文件。

本文从分组密码的算法特征入手，统计分析了影响可重构处理器的算法流程和数据特征，为其架构优化提供理论支撑；抽象密码算法循环核心、架构资源和映射方法，建立面向分组密码应用的可重构处理器性能解析模型，优化设计局部寄存器文件和全局寄存器文件；基于混合寄存器文件的研究结果，实现面向分组密码算法的可重构处理器。具体工作和创新点如下：

(1) 针对全局寄存器资源限制导致的分组密码可重构处理器性能瓶颈，通过分析分组密码算法的轮函数次数、密码处理器的重构行数和重构过程的配置暂停时间等特征参数，提出架构和算法特征依赖的全局寄存器性能模型，并用于全局寄存器文件的设计参数优化。实验结果表明，在相同的算法集合和架构特征约束下，采用本文性能模型优化的全局寄存器文件，与同类研究相比算法性能平均提高 17.24%。

(2) 针对由于存储容量大和互联复杂度高而导致全局寄存器文件的面积资源开销巨大的问题，通过分析分组密码算法在轮函数之间相互独立，轮函数写后读的数据依赖特点，基于减少寄存器互联复杂度的策略，提出分组全互联的分布式全局寄存器文件结构，减少面积资源开销。实验结果表明，相同数据访问并发度的条件下，与同类研究相比寄存器文件面积资源开销减少 41.92%。

(3) 针对局部寄存器在适配分组密码算法中 S-box 内容可变、结构多样的复杂需求，通过分析目标算法集合确定 S-box 的表数量、尺寸和输入输出位宽等约束，提炼满足目标算法集合 S-box 需求的最小局部寄存器容量和最少访存并发度，提出面向多种分组密码算法的多端口统一结构的跨域寄存器文件结构，有效减少局部寄存器文件面积资源开销。实验结果表明，在获得算法最大并行度的基础上，本文提出的局部寄存器文件结构的面积开销相比同类研究减少 26.14%。

以上混合寄存器文件的研究成果应用于课题组研发的一款分组密码可重构处理器中，目前已经完成流片前的物理设计。选取 14 种主流分组密码算法作为实验测试集合，对比分析不同寄存器文件，本文提出的混合寄存器文件架构面积效率平均提高 117.21%；与其他面向密码应用的可重构处理器采用的寄存器文件架构对比，本文混合寄存器文件架构的面积效率平均提高 66.56%。芯片物理设计的结果表明，本文实现的分组密码可重构处理器与其他同类实现方案相比，支持更多的分组密码算法的同时面积效率提高 10.62%~40.48 %。

**关键词：**粗颗粒度可重构，分组密码算法，寄存器文件，分组密码可重构处理器



## Abstract

Compared with specific circuit ASIC and custom instruction set processor ASIP, coarse-grained reconfigurable processor performs better in the key indicators of performance and flexibility for the need of cryptographic algorithms dynamic reconfiguration in specific domain. It has become an important research direction in related fields. Especially, the bandwidth limitations, memory access delays and mapping methods of the register file are the bottlenecks when block cipher algorithm implemented by reconfigurable design method. In order to meet the demand for performance and area of block cipher algorithm, the present study focused on the high area efficiency (performance area ratio) of block cipher reconfigurable processor register file.

From the block cipher algorithm characteristics, this study statistically analyzed the impact of algorithms and data flow characteristics, which provided theoretical support for architecture optimization of block cipher reconfigurable processor. A performance analytical model of the block cipher reconfigurable processor was established by abstracting cipher circulation core, architecture resources and mapping methods. And this model optimized the design of global register file and local register file. Based on the research of the hybrid register file, a reconfigurable cryptographic processor for block cipher algorithm was implemented. The specific research and innovations are as follows:

Firstly, for the performance bottlenecks of block cipher reconfigurable processor caused by the global register resource constraints, a global register performance model depended on the architecture and algorithm characteristics was proposed by analyzing the number of round function of block cipher algorithm, reconfigurable rows and the configuration dwell time in reconstruction process. This model could be used to optimize the design parameters of the global register file. The experimental results showed that the algorithm performance of the global register file optimized by the performance model increased by 17.24% on average than similar work under the same constraints of algorithms set and architectures feature.

Secondly, to resolve the problem of area resource overhead of global register file caused by the large storage capacity and connectivity range, a grouping whole interconnected distributed global register file structure which was based on the reduction register interconnection-wide policy was proposed by analyzing features that the block cipher function doesn't interact with each other on the wheel data and the data dependent on the read-after-write inner wheel data. The experimental results showed that compared with similar studies, the area resource overhead of the register file reduced 41.92% with the same parallel data access capability.

Thirdly, to meet the complex needs of content variability and structure diversity of S-box in block cipher, a cross-domain register file with diversified block ciphers and multi-ported and unified structures was proposed, which effectively reduced the area resource overhead of local register file. This was accomplished through two steps: ascertained the constraints of input and output data width, number and size of S-box in the manner of analyzing the algorithm, and then extracted the minimum capacity of the

---

local register file and minimal access concurrency which could satisfy the target cryptographic algorithm set. The experimental results showed that compared with the similar studies, the area overhead of local register file structure based on the maximum degree of algorithm parallelism reduced 26.14%.

The above findings from hybrid register file have been applied to a block cipher reconfigurable processor studied by our group. They have been taped out already. Compared with other register files, the area efficiency of hybrid register file in this study improved 117.21% on average by selecting the 14 mainstream block cipher algorithm as the experimental test collection. Compared with other register file architectures of reconfigurable cryptographic processor, area efficiency of the hybrid register file architecture increased 66.56% on average. The results of physical design showed that area efficiency of block cipher algorithm of reconfigurable cryptographic processor improved 10.62% ~ 40.48% than other cryptographic processors.

**Keywords:** coarse-grained reconfigurable , block cipher algorithm , register file , block cipher reconfigurable processor

# 目录

摘要.....	I
Abstract .....	III
目录.....	V
图.....	IX
表.....	XI
第一章 绪论.....	1
1.1 研究背景与意义.....	1
1.2 研究现状.....	2
1.2.1 可重构寄存器文件概述.....	2
1.2.2 国内外研究现状.....	5
1.3 论文研究内容和创新点.....	8
1.3.1 论文的主要工作.....	8
1.3.2 论文的创新点.....	9
1.4 论文组织结构.....	10
第二章 面向分组密码算法的可重构处理器设计.....	13
2.1 前言.....	13
2.2 分组密码算法特征.....	13
2.2.1 分组密码算法.....	13
2.2.2 分组密码特征.....	18
2.3 面向分组密码算法的可重构处理器设计模型.....	22
2.3.1 可重构密码处理器行为级模型.....	23
2.3.2 可重构密码处理器寄存器传输级模型.....	25
2.4 本章小结.....	30
第三章 分组全互联的分布式全局寄存器文件研究.....	33
3.1 前言.....	33
3.2 研究现状分析.....	33
3.3 提出全局寄存器文件的性能模型.....	34
3.3.1 可重构计算软件流水的原语.....	34
3.3.2 可重构计算软件流水的性能评估.....	35
3.3.3 基于可重构计算软件流水的全局寄存器文件性能模型.....	37
3.4 基于可重构阵列结构与映射场景的全局寄存器文件性能评估.....	42
3.4.1 面向有限重构阵列规模与可变全局寄存器文件深度的性能评估 .....	42
3.4.2 面向有限重构阵列规模与可变全局寄存器文件宽度的性能评估 .....	43
3.4.3 面向可变重构配置效率与可变全局寄存器文件深度的性能评估 .....	45

---

3.5 一种基于面积效率优化的分布式全局寄存器文件结构的提出 .....	47
3.5.1 性能约束下全局寄存器文件的面积分析 .....	47
3.5.2 分布式全局寄存器文件结构的优化设计 .....	49
3.6 实验与分析.....	52
3.7 本章小结.....	54
第四章 多端口统一结构的跨域寄存器文件研究 .....	55
4.1 前言 .....	55
4.2 研究现状分析.....	55
4.3 面向多种分组密码算法的 S-box 约束研究.....	56
4.3.1 S-box 的算法特征分析 .....	56
4.3.2 S-box 对局部寄存器文件的设计约束.....	59
4.4 一种多端口统一结构的跨域寄存器文件的提出 .....	60
4.4.1 寄存器文件的面积表达式 .....	60
4.4.2 寄存器文件的面积参数分析 .....	62
4.4.3 S-box 特征约束下的架构参数探索.....	64
4.4.4 多端口统一结构的跨域寄存器结构设计 .....	66
4.5 实验与分析.....	69
4.6 本章小结.....	70
第五章 基于混合寄存器文件的可重构密码处理器实现与分析 .....	71
5.1 前言 .....	71
5.2 可重构密码处理器硬件结构设计 .....	71
5.2.1 可重构计算单元结构设计 .....	71
5.2.2 可重构寄存器文件设计 .....	74
5.2.3 可重构阵列互联单元 .....	75
5.2.4 可重构密码阵列结构设计 .....	76
5.2.5 可重构密码处理器结构设计 .....	78
5.3 可重构密码处理器验证及物理设计 .....	79
5.3.1 可重构密码处理器的 FPGA 验证.....	79
5.3.2 可重构密码处理器的物理设计 .....	81
5.4 可重构混合寄存器文件架构的对比与分析 .....	82
5.4.1 混合寄存器文件的性能对比 .....	82
5.4.2 混合寄存器文件的面积对比 .....	83
5.4.3 混合寄存器文件的面积效率对比 .....	84
5.5 可重构密码处理器的对比分析.....	85
5.5.1 多种算法在可重构密码处理器的实验结果对比 .....	85
5.5.2 与多种计算平台的实验结果对比 .....	86
5.6 本章小结 .....	87

目录

---

第六章 总结与展望.....	89
6.1 总结.....	89
6.2 展望.....	90
致谢.....	91
参考文献.....	93
博士阶段获得的研究成果.....	99



**图**

图 1-1 密码算法各种实现方案比较.....	2
图 1-2 全局寄存器文件示意图.....	3
图 1-3 局部寄存器文件的典型互联结构示意图.....	4
图 1-4 私有寄存器文件示意图.....	4
图 1-5 CRYPTORAPTOR 可重构密码处理器结构框图 .....	5
图 1-6 COBRA 可重构密码处理器结构框图 .....	6
图 1-7 PRODFA 可重构密码处理器结构框图 .....	7
图 1-8 MORPHOSYS 可重构处理器结构框图及私有寄存器文件结构 .....	7
图 1-9 ADRES 可重构处理器结构框图.....	8
图 1-10 论文结构示意图.....	10
图 2-1 不同算法执行轮数的数量分布.....	19
图 2-2 不同密钥长度的算法个数分布.....	20
图 2-3 不同分组位宽的算法分布统计.....	21
图 2-4 可重构处理器 SYSTEMC 模型结构框图.....	23
图 2-5 可重构计算阵列的结构示意图.....	24
图 2-6 基于指针的重构单元模型.....	24
图 2-7 重构单元的互联建模方法.....	25
图 2-8 可重构密码处理器 RTL 模型结构框图 .....	26
图 2-9 可重构计算引擎结构框图.....	26
图 2-10 ALU 结构示意图 .....	27
图 2-11 查找表互联方式.....	28
图 2-12 可重构数据控制器结构框图.....	29
图 2-13 可重构配置控制器结构框图.....	30
图 2-14 配置控制器工作流程图.....	30
图 3-1 循环迭代的映射实例.....	35
图 3-2 不同计算循环的流水示意图.....	36
图 3-3 循环迭代在可重构阵列上的软件流水过程.....	37
图 3-4 全局寄存器无限的计算示意图.....	39
图 3-5 重构计算资源有限，全局寄存器无限的性能分析.....	39
图 3-6 重构计算资源有限，全局寄存器有限的性能分析.....	40
图 3-7 分组密码算法执行轮数的区间分布情况.....	42
图 3-8 32 轮密码算法在不同阵列大小的性能比较.....	43
图 3-9 密码算法分组位宽的区间分布情况.....	44
图 3-10 128 比特分组位宽的密码算法在不同阵列大小的性能比较.....	45
图 3-11 32 轮密码算法在不同配置信息切换约束下的性能比较 .....	45
图 3-12 性能和面积随全局寄存器的变化趋势 .....	47
图 3-13 全局寄存器数量增加的性能变化率.....	48
图 3-14 不同阵列大小条件下处理器面积随全局寄存器数量变化趋势 .....	48
图 3-15 分组密码性能和面积效率随全局寄存器变化趋势 .....	50
图 3-16 寄存器体和译码逻辑面积随寄存器数量和读写端口数变化趋势 .....	51
图 3-17 全局寄存器架构示意图.....	51
图 3-18 分布式全局寄存器文件设计细节.....	52
图 3-19 不同设计方案的全局寄存器面积分布.....	53

---

图 4-1 分组密码算法的 S-BOX 大小分布 .....	57
图 4-2 分组密码算法中 S-BOX 数量分布图 .....	57
图 4-3 分组密码算法中每轮计算对 S-BOX 的并发访存次数.....	58
图 4-4 分组密码算法的输入位宽统计.....	58
图 4-5 分组密码算法的输出位宽统计.....	59
图 4-6 理想 S-BOX 的结构示意图 .....	61
图 4-7 端口数为 1 时，寄存器体和译码逻辑面积与寄存器容量的关系.....	62
图 4-8 寄存器容量固定时译码逻辑面积与端口数变化的趋势 .....	63
图 4-9 寄存器容量固定时寄存器体面积与端口数变化的趋势 .....	63
图 4-10 寄存器体和译码逻辑面积随寄存器数量和端口数变化趋势 .....	65
图 4-11 模式 1 数据输入输出示意图.....	66
图 4-12 模式 2 数据输入输出示意图.....	66
图 4-13 模式 3 数据输入输出示意图.....	67
图 4-14 可重构跨域寄存器文件示意图.....	67
图 4-15 跨域寄存器文件簇结构示意.....	68
图 4-16 输入控制逻辑结构示意图.....	68
图 4-17 输出控制逻辑结构示意图.....	69
图 5-1 AES 算法的加解密过程示意图 .....	72
图 5-2 AES 算法的加解密轮函数示意图 .....	73
图 5-3 AES 算法算子级操作映射过程 .....	73
图 5-4 全局寄存器文件架构设计 .....	74
图 5-5 跨域寄存器文件架构设计 .....	75
图 5-6 互联单元结构图 .....	75
图 5-7 BENES128 网络的内部结构 .....	76
图 5-8 计算引擎中可重构阵列行单元结构 .....	77
图 5-9 可重构计算引擎结构 .....	78
图 5-10 可重构处理器系统 SoC 架构示意图 .....	79
图 5-11 可重构密码处理器系统的 FPGA 开发原型 .....	80
图 5-12 可重构密码处理器版图 .....	81
图 5-13 本文寄存器文件架构的算法性能增益 .....	83
图 5-14 不同模块在可重构密码处理器中面积百分比占用情况 .....	84
图 5-15 混合寄存器文件面积效率增益 .....	85

## 表

---

表 1-1 寄存器连接数量和寄存器数目录示例.....	3
表 2-1 分组密码算法列表.....	14
表 3-1 影响计算性能的约束条件.....	37
表 3-2 不同架构约束条件的分析场景.....	38
表 3-3 分组密码算法特征.....	49
表 3-4 全局寄存器设计参数.....	51
表 3-5 不同全局寄存器设计方案的设计参数对比.....	53
表 3-6 不同全局寄存器设计方案的性能和面积效率对比.....	54
表 4-1 部分算法的 S-BOX 特征参数 .....	60
表 4-2 分组密码算法特征.....	64
表 4-3 面积对比 CRYPTORAPTOR 架构 S-BOX 设计实现 .....	69
表 4-4 不同架构实现 AES 算法的 S-BOX 面积对比 .....	70
表 5-1 密码算法的操作序列叠加.....	72
表 5-2 算子叠加操作对不同分组密码算法的操作周期影响.....	74
表 5-3 可重构密码处理单元内部模块的资源利用情况.....	80
表 5-4 可重构密码处理器的面积和资源开销.....	82
表 5-5 不同寄存器文件方案的性能对比.....	83
表 5-6 不同寄存器文件方案的面积对比.....	84
表 5-7 不同寄存器文件方案的性能面积对比.....	84
表 5-8 不同的处理平台的分组密码算法性能面积对比.....	86
表 5-9 不同计算平台处理器的分组密码算法的性能面积对比.....	87



# 第一章 绪论

## 1.1 研究背景与意义

随着国民经济和国防建设的不断发展，信息安全成为社会经济发展的基础条件和国家安全稳定的重要保障。传统行业中计算机和通信技术不断渗透，党政军机要部门则越来越依赖信息技术提高效率，使得保密通信成为保障关键领域信息安全的重要手段。密码技术作为保证信息的机密性、安全性和可用性等安全要求最有效和可行的手段之一，通过采用数据加密、消息认证和数字签名等方式，能够防止篡改、抵赖和伪造等不安全通信或数据存储行为。分组密码算法作为现代密码处理系统中使用最多的私用算法，广泛应用于密码安全系统和通信安全<sup>[1, 2]</sup>。在网络通信中，多种安全协议用于保证信息的安全，如 Ipsec、TLS/SSL、SSH、OpenPGP 和 S/MIME 等，这些协议中包含了多种密码算法，例如在 IPsec 协议中可以选择 AES<sup>[3]</sup>、3DES<sup>[4]</sup>、IDEA<sup>[5]</sup>等算法作为数据加密算法。

密码算法动态可变是提高保密通信安全等级的重要手段。作为实现密码技术的基础，密码算法的安全性体现在一定数学理论支撑下的计算困难度。密码算法的使用特性决定其具有较高的灵活性，一方面密码算法种类繁多，目前国际上公开的各类密码算法多达数百种，各个国家为保证特殊领域安全还研制了大量非公开算法。另一方面，随着通信安全协议的升级，密码算法不断更新，即使同一个安全协议中也可以选择不同的算法加密。特别在特殊安全领域中，不定期更新非公开密码算法是保障通讯安全的重要手段。

作为保护通信安全的有效方法，密码算法的安全性一方面体现在数学理论支持下的一定长度密钥参与下的明文加密计算，另一方面体现在密码处理硬件实现本身的安全性。不同安全等级的密码算法具有不同的考虑，针对商用算法，密码处理器通过多种电路级安全措施抵御旁路攻击等攻击手段<sup>[6-11]</sup>；而针对非公开密码算法，不仅要抵御旁路攻击，还要考虑在处理器制造过程中不泄露密码算法的信息。当前，通过在制造过程中添加部分攻击电路到密码芯片<sup>[12-14]</sup>，或者通过修改制造工艺过程来改变芯片晶体管功能<sup>[15]</sup>，都将泄露密码及算法信息，严重威胁到保密通信的安全。尤其在目前国内集成电路制造能力依然落后的形势下，高性能密码处理器境外流片的风险尤其巨大。

基于粗粒度可重构的密码处理器不仅能够满足密码算法动态可变的需求，而且能够有效防止制造过程中密码算法信息泄露的安全隐患。作为一种将软件的灵活性和硬件的高效性结合在一起的计算结构，可重构密码处理器在性能和灵活性等关键指标之间具有更好的平衡。大量基础处理单元通过一定互联方式形成新的计算结构<sup>[16]</sup>，通过软件改变单元功能和互联方式，具有足够的灵活性以满足不同密码算法和安全标准不断发展的需求。同时，通过配置信息更新可以改变硅后芯片的功能，防范在设计和制造过程中可能的算法信息泄露风险。

可重构密码处理器可以实现高吞吐率加解密计算。可重构计算<sup>[16-20]</sup>被公认为是高性能计算中替代特殊定制 ASIC 设计极具吸引力的选择<sup>[21, 22]</sup>，它在软件编程级别灵活性的基础上获得媲美传统 ASIC 设计实现的高性能。高效率和高吞吐量加解密处理是密码系统的关键，如图 1-1 所示，除了通用处理器 GPP 以外，高性能密码处理主要可以分为三类：专用密码电路 ASIC<sup>[23-26]</sup>、定制指令集处理器 ASIP<sup>[27-29]</sup>和可重构密码处理器。专用密码电路 ASIC 的研究非常深入，能够获得特定算法在单

位面积条件下最高的加解密性能。专用密码电路 ASIC 对每种密码算法都要设计独立的硬件模块，较低灵活性和高昂的一次性工程（Non-Recurring Engineering, NRE）费用使其难以满足应用环境中灵活实现密码算法的需求。定制指令集处理器 ASIP 针对密码算法特点扩展指令集和处理单元，具有很强灵活性，但是受制于有限计算资源和存储带宽使得执行效率难以进一步提高。以 FPGA 为基础实现的细粒度可重构密码处理器具有极高灵活性，但是由于其编程单元粒度（多为 1-4 比特）与密码算法的处理字长（一般为 8-32 比特）不一致，导致其在面积方面效率低下。相对于细粒度 FPGA，粗粒度可重构的计算逻辑特别适合并行计算，在消耗较少资源的条件下就能获得很高的计算性能。可重构计算阵列既可以采用多个重构单元共享配置信息的方式实现 SIMD<sup>[30]</sup>计算方式<sup>[31]</sup>，也可以实现典型的 MIMD<sup>[30]</sup>计算方式<sup>[32]</sup>。同时，粗粒度可重构运行中更新配置信息量小，可以实现运行时动态重构（Run-Time Dynamic Reconfigurable）。因此，近年来陆续提出了 Cryptoraptor<sup>[33]</sup>、COBRA<sup>[34]</sup>、Celator<sup>[35]</sup>和 proDFA<sup>[36]</sup>等针对密码算法的粗粒度可重构架构。

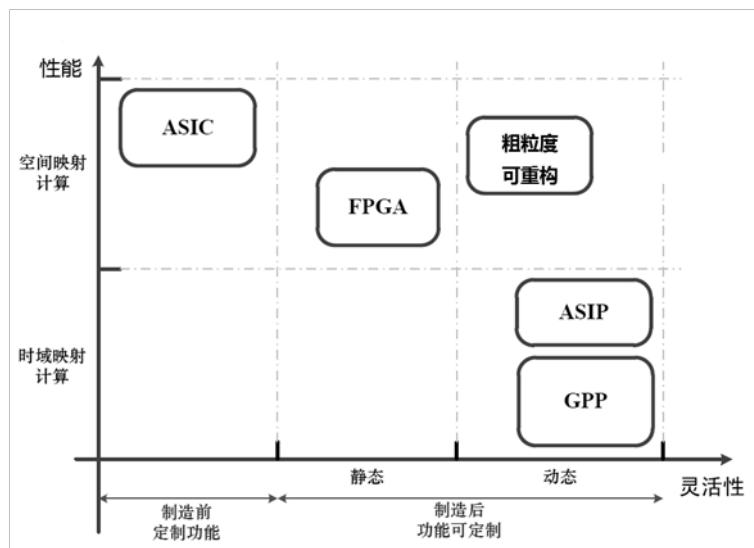


图 1-1 密码算法各种实现方案比较

综上所述，可重构密码处理器是适应密码算法动态可变、高效率和高吞吐量需求，而且满足制造过程安全的一种密码实现方式，可以作为安全系统的核心部件应用于保密通信、保障信息安全等重要场合，可以预期可重构密码处理器将成为未来某些关键领域密码算法极其重要的实现手段。

## 1.2 研究现状

### 1.2.1 可重构寄存器文件概述

寄存器文件是关系到可重构密码处理的重要研究内容<sup>[37-39]</sup>。一方面，分组密码算法具有分组数据相互独立和计算数据写后读（RAW, read-after-write）依赖的特点<sup>[40, 41]</sup>，对数据的访存延时和数据带宽提出了很高要求；另一方面，可重构处理器计算阵列规模巨大，通常包含几十甚至上百个重构计算单元，大量计算资源为高吞吐率密码处理提供了并行加速的可能，但当阵列计算的数据等待时间远远大于阵列本身的计算时间时，计算性能的加速收益将被存储限制消耗殆尽<sup>[19]</sup>。因此，存储带宽的限制、存取访问的延时、存储映射的困难都使寄存器文件的架构设计和硬件实现成为可重构密码处理器的限制瓶颈，是可重构密码处理的重要研究内容。

寄存器文件按照互联范围可以分为私有寄存器、局部寄存器和全局寄存器。可重构处理器将临时数据存储在寄存器文件中，减少甚至避免了计算阵列存取外部存储器的延时，从而大幅提升了可重构处理的性能<sup>[42, 43]</sup>。由于寄存器的互联开销与互联范围呈指数关系，互联范围是寄存器文件最重要的表征参数，因此如表 1- 1 所示，按照寄存器文件不同的互联范围可以分为：全局寄存器、局部寄存器和私有寄存器，不同寄存器的资源开销和数量也不同。

表 1- 1 寄存器连接数量和寄存器数目示例

名称	互联范围	资源开销	数量
私有寄存器	计算单元	非常多	少量
局部寄存器	局部阵列	一般	一般
全局寄存器	全阵列	很少	很多

### (1) 全局寄存器文件

全局寄存器文件（Global Register File, GRF）用于可重构计算阵列所有单元的数据传输，常用于阵列动态配置过程中的中间变量存储，以及阵列流水线展开过程中的数据缓存。

全局寄存器文件作为可重构阵列处理单元之间的共享寄存器，实现类似于传统处理器一级缓存的作用，而且拥有可重构阵列中最大的扇出系数。全局寄存器拥有最高的访存灵活性，但是巨大的扇出数目导致物理实现过程中面临时序收敛困难和互联面积开销巨大的问题。因此，全局寄存器的架构参数设计和面积资源开销优化都是可重构研究的关键问题。

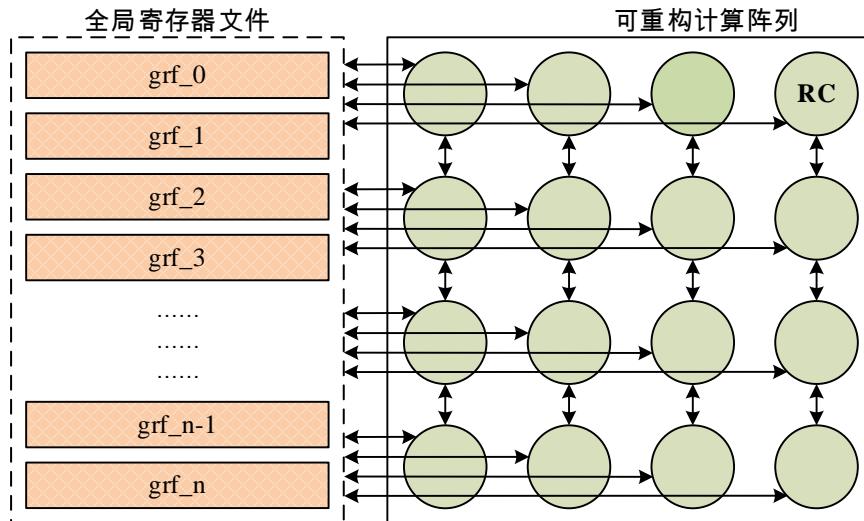


图 1- 2 全局寄存器文件示意图

图 1- 2 给出全局寄存器的示意图，可重构阵列中所有的计算单元都通过直连线与全局寄存器文件互联。增加全局寄存器文件的并发端口数，有助于提高计算阵列的数据访问带宽，其端口数应满足重构阵列需求，端口位宽与重构单元的端口位宽一致。

### (2) 局部寄存器文件

局部寄存器（Local Register File, LRF）用于局部计算阵列单元之间的数据交换，常用于单次配置内多个计算单元之间的数据共享和交换，以及部分静态数据存储。

局部寄存器文件的数量和互联方式是可重构架构空间探索的一个关键内容。在满足应用所需的

存储空间和访存并发度的条件约束下，寄存器体的数据量和互联方式相互制约，成为影响可重构处理器面积效率的关键因素。

局部寄存器文件可以为实现多种互联方式提供灵活的存取机制，如图 1-3 所示分别为行互联，隔行互联，块互联以及分散互联的设计结构。其中，行互联寄存器可以满足每一行数据之间的传输，但是不能进行行与行以及列与列之间的数据传输；隔行互联寄存器可以实现多行数据之间的传输，但是寻址复杂和互联距离长成为实现的主要困难；块互联寄存器的扇出结构类似于全局寄存器文件可以为局部可重构单元提供了灵活的数据共享方法，但是块互联寄存器之间的数据交换依赖外部数据存储，其数据传输范围的局限性成为设计的主要限制；分散互联寄存器文件可以灵活实现阵列内多个离散可重构单元的数据共享和传输，但是复杂的寻址方式成为设计的主要瓶颈。

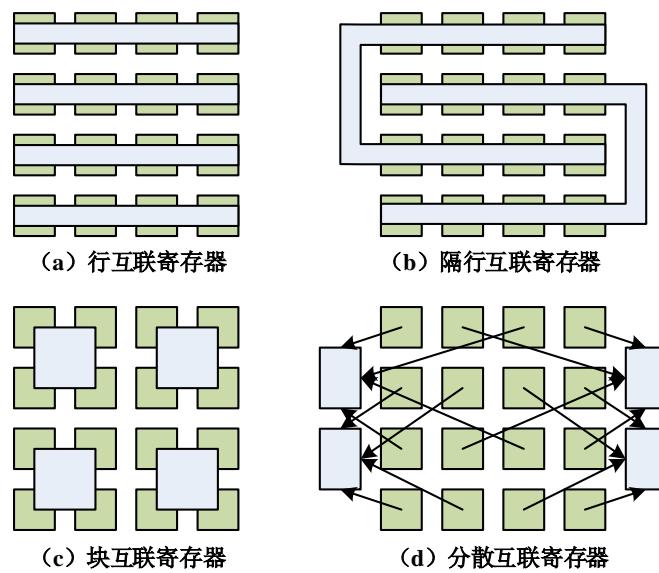


图 1-3 局部寄存器文件的典型互联结构示意图

### (3) 私有寄存器文件

私有寄存器文件（Private Register File, PRF）作为仅供计算单元自己使用的内部寄存器，常用于存储算法常量、计算结果和临时变量。私有寄存器文件可以实现单周期的高速读写操作，对于形如 $i = i + 1$ 的计算可以在单周期内完成。

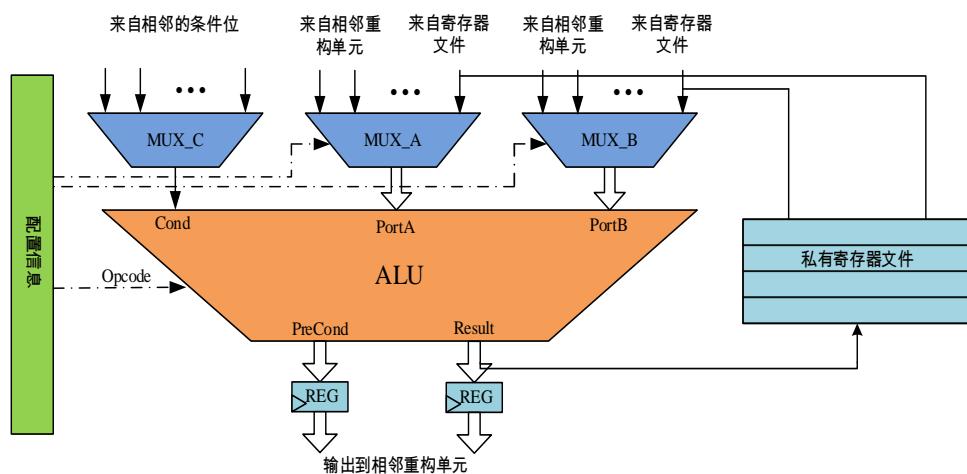


图 1-4 私有寄存器文件示意图

图 1-4 给出私有寄存器文件的示意图。可重构计算单元通过配置信息中源操作域和目的操作域

指定的标号进行读写，或者通过计算中生成的变量作为寻址标号。与传统通用处理器拥有多组多个通用寄存器不同，每个可重构单元仅有少量几个寄存器用于存储生命期较长而空间位置固定的数据变量。由于私有寄存器文件的数量与可重构计算单元的数量相同，因而私有寄存器文件的设计对于资源开销尤其敏感，增加私有寄存器容量将导致面积开销的成倍增加。

### 1.2.2 国内外研究现状

作为影响可重构密码处理器性能和面积的关键，寄存器文件成为各类可重构密码处理器的核心研究内容。近年来，随着在某些特殊领域的研究和应用，出现了 Cryptoraptor<sup>[33]</sup>、COBRA<sup>[34]</sup>、Celator<sup>[35]</sup> 和 proDFA<sup>[36]</sup> 等多款可重构密码处理器。下文着重阐述了寄存器文件的功能、结构、大小和特点，综述如下：

Cryptoraptor<sup>[33]</sup> 可重构密码处理器采用一个集中式的全局寄存器文件，并为每个可重构计算单元配置用于实现 S-box 的局部寄存器文件。全局寄存器文件采用 1 个包含 256 个 32bit 寄存器的寄存器体 ( $8 \times 32\text{bit}$ )，拥有 80 个 32bit 位宽的读端口，8 个 32bit 位宽的写端口。整个全局寄存器文件的面积为  $1.78\text{mm}^2$ ，占整个密码处理器面积的 28.16%。Cryptoraptor 处理器为每个可重构计算单元设计局部寄存器文件 TLU (Table Lookup Unit)。为支持更广泛的密码算法，TLU 中提供了三个  $256 \times 32\text{bit}$  和一个  $1024 \times 32\text{bit}$  表（分别命名为 SBOX 和 TBOX）。采用这样的设计方法，可以将较大的查找表分割成多个 SBOXes / TBOXes，并支持多级查找操作。局部寄存器文件的面积为  $0.88\text{mm}^2$ ，占整个密码处理器面积的 13.91%。

Cryptoraptor 可重构密码处理器提供了满足算法最大并发访存度和最大算法存储需求的寄存器文件，可以兼容多种算法的设计实现，是目前实现算法最多的高性能密码处理器。Cryptoraptor 在目前可重构密码处理器中拥有最高的性能和面积效率，但是冗余的存储空间和低效的架构设计占用大量的面积资源开销，严重制约了处理器的面积效率比。

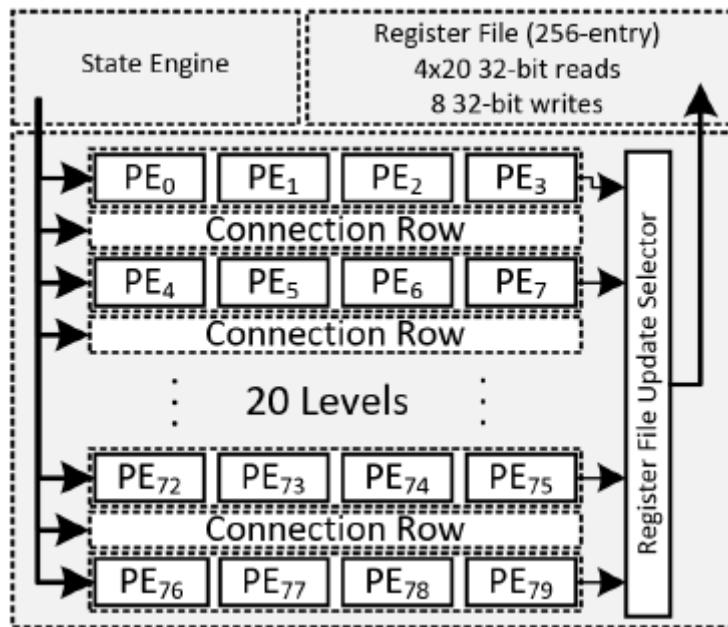


图 1-5 Cryptoraptor 可重构密码处理器结构框图

COBRA<sup>[34]</sup> (Cryptographic (Optimized for Block Ciphers) Reconfigurable Architecture) 密码处理器架构在常用的 41 个分组密码算法进行分析的基础上进行设计空间的优化探索，其重构计算单元包含

除 Modular Inversion 之外的所有操作算子，因而可以满足现有大部分分组密码算法及未来包括这些操作算子的新密码算法。

COBRA 的整体架构如图 1-6 所示，它是一个  $4 \times 4$  的 RCE 阵列，RCE 作为阵列的基本运算单元，支持 32 位的基本算子运算，四列的 RCE 总共可以组成 128 位的数据通路。在阵列中有两类微结构：

- (1) 1、3 列的 RCE 不带乘法器；
- (2) 2、4 列的 RCE 支持乘法算子。

在 1、2 行和 3、4 行之间各有一个字节置换单元，实现数据的按字节换位。架构的寄存器文件主要包括 16 个 eRAM，用于存储中间值、轮密钥等。阵列两边各有 8 个 eRAM 分别供左右两列使用，RCE 与 eRAM 没有对应关系，每一个 RCE 都可以访问所分配的 8 个 eRAM。在第 3 行输出的地方有 4 个 32 位的寄存器，实现每轮数据的输出反馈。局部寄存器 eRAM 的实现需要 1210640 个门单元，占整个可重构密码处理器逻辑资源的 18.09%。

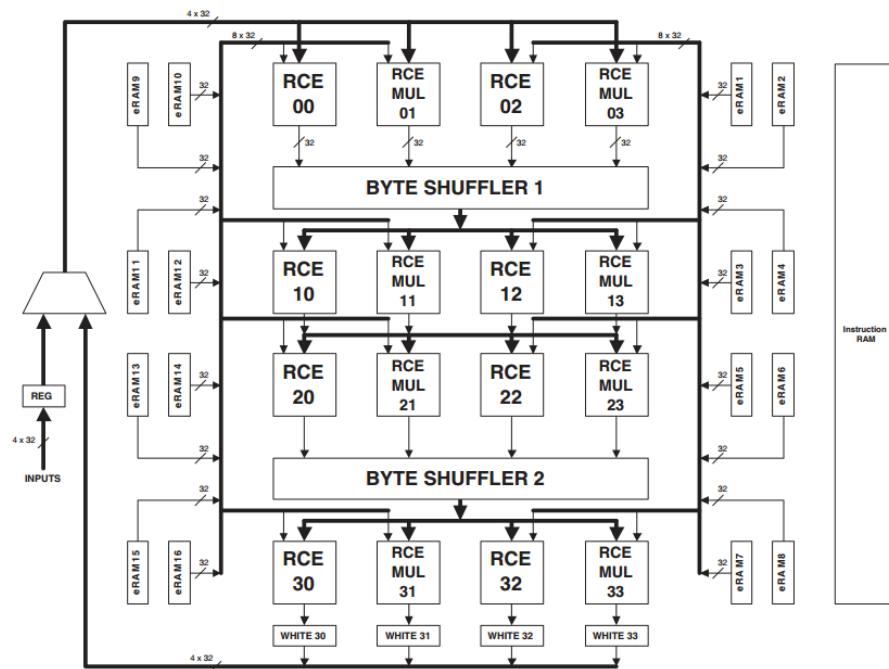


图 1-6 COBRA 可重构密码处理器结构框图

ProDFA<sup>[36, 44]</sup> (programmable dataflow computing architecture) 是一个单周期配置、长执行周期的可重构密码架构，如图 1-7 的子图 C 所示。整个架构采用分层结构，通过将多个子图 B 所示的可重构处理节点 RPN(reconfigurable processing node) 组合从而扩展成一个更大规模的可重构计算阵列。RPN 由 4 个子图 A 所示的可重构处理单元微结构 RPU(reconfigurable processing unit) 和 RPU 簇控制逻辑组成。

RPU 可划分为三个部分：

- (1) 配置控制部分；
- (2) 功能运算部分；
- (3) 外部接口部分。

功能运算部分包括多个可配置定制功能单元 CAFU(configurable application-specific function unit)。存储单元 LMEM 用于保存运算结果。互连总线 CIB 用来传输配置和运算单元之间的互连。同步单元 SYNC 完成数据同步。CAFU 设计为可配置替换单元 CSU(configurable substitute unit)，作为

RPU 的私有寄存器文件使用。在 SIMC 120nm 标准单元库工艺下，每个 CSU 的面积为 39316um<sup>2</sup>，占总面积的 21.89%。

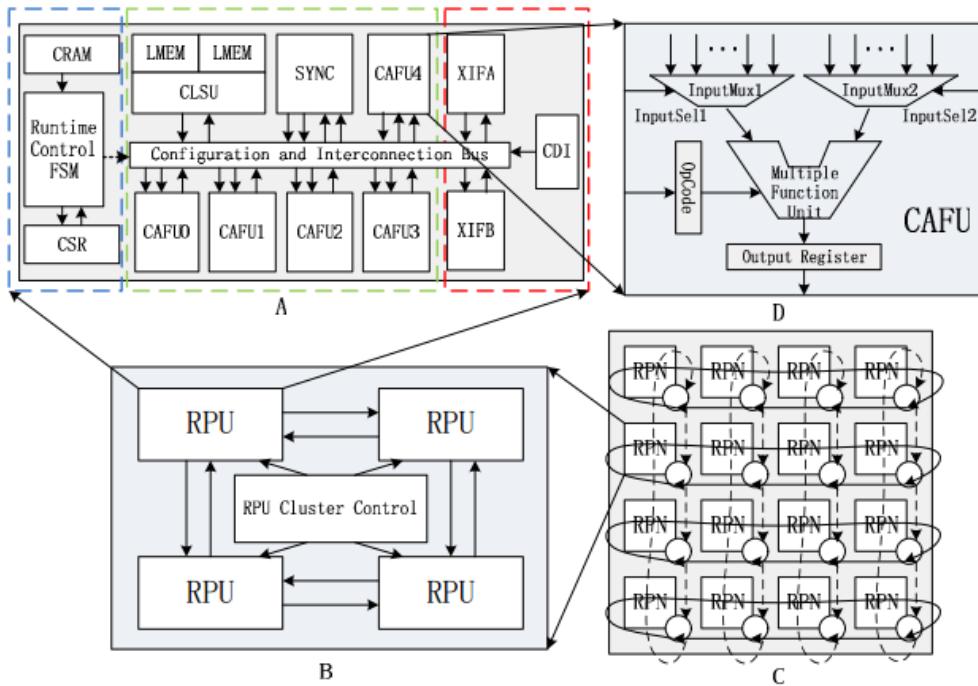


图 1-7 ProDFA 可重构密码处理器结构框图

MorphoSys<sup>[31, 45]</sup>采用复杂的可变宽度的块数据计算和传输，利用帧缓存（Frame Buffer）区实现数据和计算的重叠，如图 1-8 所示。MorphoSys 的每一个可重构单元 RC 包含 4 个 16bit 的私有寄存器，用于缓存计算的输出结果。此外，MorphoSys 利用内嵌的 TinyRISC 处理器扩展 DMA 指令完成数据传输，采用猝发访问（Burst）的方式完成外存数据到帧缓存区之间的数据传输。同时，利用双帧缓存（Double Frame Buffer）实现计算阵列与处理器主存储器的同时读写操作，另一个帧缓冲实现计算阵列数据的输入输出存储，一个帧缓冲实现和外部的数据交换，通过 Ping-Pang 方式降低数据交换对性能的影响，同时每个 PE 内部的私有寄存器用于存储中间变量。通过将 IDEA 分组密码算法映射到 MorphoSys 架构最高实现并行度为 4 的算子操作，整个算法获得 0.22Block/Cycle 的性能。

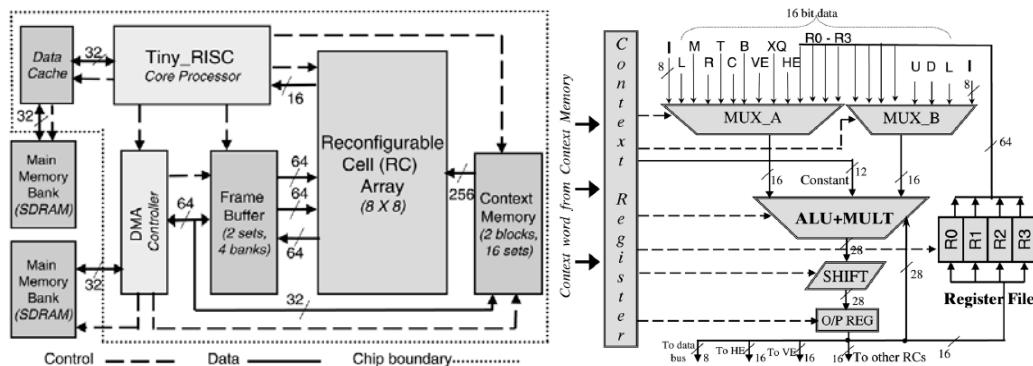


图 1-8 MorphoSys 可重构处理器结构框图及私有寄存器文件结构

ADRES<sup>[32, 46, 47]</sup>采用类似于 MorphoSys 的可重构功能单元（Function Unit, FU）组成二维计算阵列 CGRA（Coarse-Grained Reconfigurable Array）处理应用中计算密集的操作，利用 VLIW（Very Long Instruction Word）处理器已有的数据访存通路实现数据存取。ADRES 架构中采用参数化的紧耦合全局寄存器文件（Global Register File）作为外部数据的缓存区，同时满足 VLIW 模式和 CGRA 模式相

互切换时的数据交换需求,如图 1-9 所示。此外,ADRES 还设计了局部寄存器用于重构功能单元(FU)之间的数据交换。然而, VLIW 模式下的数据访存通路在数据密集型应用,如软件无线电( SDR) 应用中,带宽限制仍然成为计算瓶颈。

为了满足并行计算应用的数据需求,减少访存冲突和硬件开销,采用访存队列重排序的机制<sup>[48]</sup>,并且研究了数据在 Multibank 片上存储器的映射方式。ADRES<sup>[49]</sup>中通过设置每个功能单元(Function Unite) 的私有寄存器(Private Register File) 资源和连接对角线功能单元的 4 个局部寄存器,可以探索不同互联、容量的寄存器文件设计参数下重构配置信息和循环迭代的优化空间。Kwok, Z.<sup>[50]</sup>探索了 ADRES 架构中私有寄存器和全局寄存器文件对可重构处理系统面积和性能的影响,以及架构约束下私有寄存器的互联方式。Andres, G.<sup>[51]</sup>特别研究了定制的 LUT 指令加速局部寄存器文件访存速率对 AES 算法实现的性能影响。

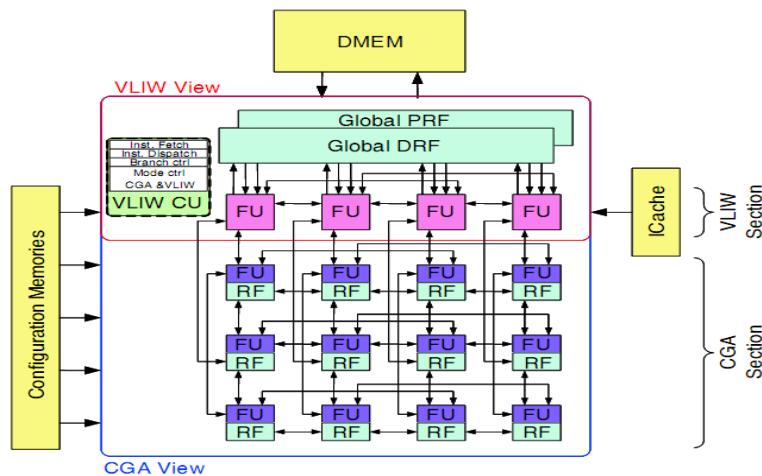


图 1-9 ADRES 可重构处理器结构框图

通过以上研究分析可知,可重构寄存器文件研究主要集中在提高数据吞吐率、减少寄存器文件面积两个方面。现有研究针对限制可重构密码处理器性能的寄存器文件的容量和并发度、访存接口设计等架构设计进行优化,有效解决了存储带宽的限制和存取访问的延时问题。

然而,由于对分组密码算法特征分析不足,现有的可重构寄存器文件架构存在一定的缺陷,难以面向多种密码算法实现高的面积效率,存在灵活性不足和面积效率低下的问题。分组密码算法的计算操作类型多样,使用频度分布不均,导致架构灵活性不足,仅能实现某些特点的密码算法。计算操作的位宽在不同算法中以及同一算法中的不同步骤不固定,导致架构中使用大量的硬件资源为代价,存在大量的计算单元和存储资源的冗余,面积效率低下。

因而,可以从以下两个方面对可重构寄存器文件进行优化:①全局寄存器文件:通过优化寄存器容量减少计算暂停等待的时间,同时通过寄存器架构的优化设计减少寄存器面积资源的开销。②局部寄存器文件:通过架构空间探索,在满足目标算法最小限制的条件设计面积效率最优的局部寄存器文件。通过寄存器文件架构多样化、层次化、异构化的探索研究,解决重构逻辑中不同层次的数据流瓶颈,从而优化可重构计算阵列的性能面积比。

## 1.3 论文研究内容和创新点

### 1.3.1 论文的主要工作

通过以上研究现状的分析可知，寄存器文件与可重构密码处理器面积效率密切相关：寄存器文件的访存并发度不足导致密码处理停顿，寄存器文件容量不足导致处理性能下降，同时寄存器文件的面积开销巨大。为了解决寄存器文件对密码算法并行计算性能的制约，课题着重研究了可重构架构中不同层次的寄存器文件，提高可重构密码处理器的面积效率。

本文的主要工作内容如下：

- (1) 分组密码算法特征分析研究。分析整理现有文献和应用中的分组密码算法，研究其影响性能和面积的算法特征，为架构优化提供支撑。
- (2) 全局寄存器的动态重构计算性能模型研究及架构优化设计。通过对密码算法并行计算的循环核心、可重构架构资源以及对应的映射过程抽象，获得可重构密码处理的性能解析模型。利用解析模型，分析限制密码处理流水性能的影响因素，探索算法特征和阵列设计约束下的性能和面积最优解，优化可重构全局寄存器文件设计参数；利用密码处理轮函数之间的独立性，采用分组全互联的分布式全局寄存器结构减少面积开销。
- (3) 针对分组密码算法 S-box 的局部寄存器的优化设计。分析分组密码中 S-box 算法，提炼影响架构设计的 S-box 特征参数。综合考虑不同密码算法中 S-box 差异化需求，探索最优 S-box 设计。研究寄存器文件的面积资源的解析表达式，优化设计寄存器文件，实现高面积效率的局部寄存器文件。
- (4) 包含混合寄存器文件（分组全互联的分布式全局寄存器文件+多端口统一结构的跨域寄存器文件）的可重构密码处理器设计实现。
  - (a) 可重构密码处理器的仿真模型和 RTL 设计。针对可重构密码处理器的架构空间探索需求，设计可以参数化实现的设计模板，包括 SystemC 构建的周期精确行为级模型以及 Verilog HDL 实现的寄存器传输级 RTL 模型。
  - (b) 可重构密码处理器的 FPGA 原型实现。通过将可重构密码处理器在 FPGA 平台上实现，有效验证了其功能正确性，并获得分组密码算法的处理性能。
  - (c) 可重构密码处理器实现。基于本文混合寄存器文件的优化设计，采用 TSMC 40nm 1P8M LP 低功耗工艺库实现面向分组密码的可重构密码处理器。

### 1.3.2 论文的创新点

本文研究了可重构混合寄存器文件架构，建立其性能和面积解析模型，针对可重构密码处理器高面积效率的需求，侧重优化了寄存器文件的性能和面积。本文研究工作的创新点主要体现在以下三个方面：

- (1) 提出架构和算法特征依赖的全局寄存器性能模型，并优化全局寄存器设计结构

针对全局寄存器资源限制而造成可重构密码处理器的性能瓶颈，通过分析分组密码算法的轮函数次数，密码处理器的重构行数和重构过程的配置暂停时间等特征参数，提出架构和算法特征依赖的全局寄存器性能模型，并用于全局寄存器文件的设计参数优化。实验结果表明，在相同的算法集合和架构特征约束下，采用本文性能模型优化的全局寄存器文件，与 Cryptoraptor 设计采用的全局寄存器架构相比，算法性能平均提高 17.24%；与 ADRES 设计采用的重排序全局寄存器文件架构相比，算法性能平均提高 230.67%。

### (2) 提出分组全互联的分布式全局寄存器文件结构

针对由于存储容量和互联范围大而导致全局寄存器文件的面积资源开销巨大的问题，通过分析分组密码算法在轮函数之间相互独立，轮函数写后读的数据依赖特点，基于减少寄存器互联范围的策略，提出分组全互联的分布式全局寄存器文件结构，减少面积资源开销。实验结果表明，在相同数据访问并发度的条件下，比 Cryptoraptor 设计实现的寄存器文件面积资源开销减少 41.92%。

### (3) 提出面向多种分组密码算法的多端口统一的跨域寄存器结构

针对局部寄存器在适配分组密码算法中 S-box 内容可变、结构多样的复杂需求，通过分析目标算法集合确定 S-box 的表数量、尺寸和输入输出位宽等约束，提炼满足分组密码算法 S-box 的最小局部寄存器容量和最少访存并发度，提出面向多种分组密码算法的多端口统一的跨域寄存器文件结构，有效减少局部寄存器文件面积开销。实验结果表明，在满足算法最大并行度的基础上，本文提出的局部寄存器文件结构与文献[33]相比，面积开销减少 26.14%。

通过以上对混合寄存器文件的优化设计，针对制约可重构密码处理器计算性能和面积的关键问题，最大限度的提高可重构密码处理器的面积效率。

## 1.4 论文组织结构

本论文结构如图 1-10 所示，本章介绍课题研究背景和意义，研究现状以及论文的主要工作和创新点，其他章节的内容如下：

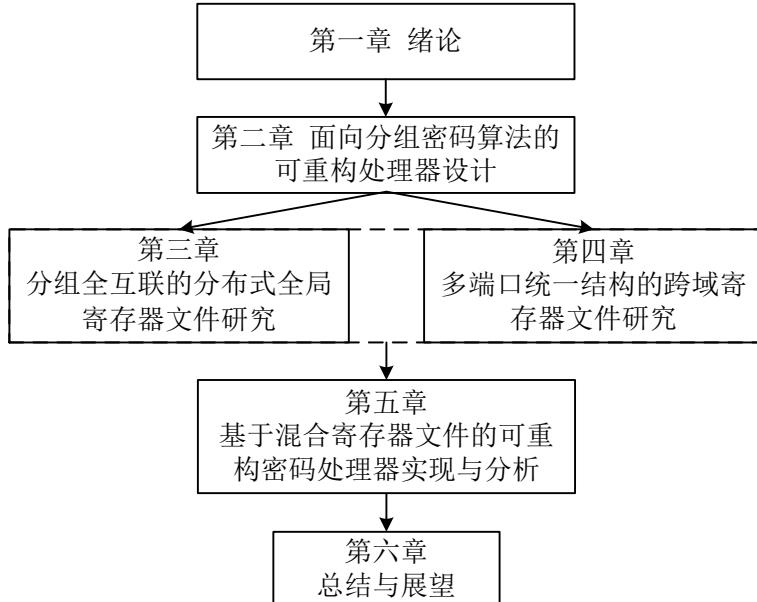


图 1-10 论文结构示意图

第二章分析了现有文献中涉及的分组密码算法，介绍本文采用的可重构密码设计平台，为可重构寄存器文件优化提供设计基础。

第三章首先说明了密码算法在可重构处理器的映射实现，分析全局寄存器文件对性能的影响。然后讨论架构约束条件下性能和面积的变化趋势，获得最优性能面积比的设计参数。通过对硬件微结构的优化，设计实现全局寄存器文件。最后，通过对比实验给出性能和面积的优化效果。

第四章首先介绍分组密码算法中 S-box 的研究现状，特别分析算法中 S-box 特征对设计的约束条件。然后，以此为基础讨论算法特征约束下寄存器文件面积开销，探索面积效率最优的局部寄存

器文件设计参数，满足不同算法 S-box 的差异化需求。最后，通过对比实验给出架构的面积优化效果。

第五章基于第三、四章研究得到的设计参数实现 RTL 设计，在现有分组密码算法中选取广泛应用的主流算法作为实验内容，对比不同寄存器文件对可重构密码处理器面积效率的影响，最后对比分析了本文提出的混合寄存器文件优化方法。

第六章总结归纳了论文研究工作，提出了论文未来工作方向及改进意见。论文结尾列出了相关参考文献和资料。



## 第二章 面向分组密码算法的可重构处理器设计

### 2.1 前言

为了能够实现灵活、高性能面积比的可重构密码架构，本章首先对现有参考文献中相关的分组密码算法进行分析，针对密码算法的规格、结构和特征进行统计研究。虽然每个密码算法具有不同的结构和差异化的特征，但是本文研究致力于探索通用模式及共同特征，同时满足不同算法的多样性功能。然后，本文介绍面向分组密码算法的可重构处理器设计框架，采用不同精度层次的模型实现参数化的架构空间探索。可以针对分组密码算法的结构和特征，能够比较多种可重构密码处理器的设计方法，总结和提炼出算法对架构潜在的设计要求和规范。

### 2.2 分组密码算法特征

分组密码，又称为分块密码，是一种对称密钥算法，将明文或密文分成多组等长模块，通过对称密钥对每组数据分别进行加密解密。分组密码算法的结构主要分为 Feistel 网络结构和 SP 网络结构。Feistel 网络结构的数据块分为两个部分，其中一半的操作依赖于另一半；SP 网络结构的数据块作为一个整体进行操作。两种网络结构都具有数据位宽整齐、执行速度快和操作执行密集的特点。分组密码作为重要的加密协议组成部分在密码协议中起到了至关重要的作用，广泛应用于网络通信和电子商务的各个领域。

目前的密码研究主要集中在一个小密码集合和少数的操作类型<sup>[33, 35, 36, 52]</sup>，综合分析现有的加密算法的功能和硬件结构，并没有对实现高性能的可重构分组密码架构提供足够详细和全面的算法依赖的硬件结构分析。另外，文献[53, 54]对一组密码算法的软件实现和计算负载进行分析，根据算子的适用频率和操作类型给出分类集合，由于仅限定在一个小的算法集合中，因而不足以设计一个高度可配置的密码处理器。Elbirt<sup>[34]</sup>通过分析 41 个对称分组密码，提供了基础的硬件单元和详细的硬件结构。然而，算法分析仅限于对明文长度为 64bit 和 128bit 的分组密码算法，因而，类似于其他研究，并没有给出功能单元、计算模式和连接结构的依赖关系。现有的研究尚不能提供满足可重构分组密码算法所需的足够的详细分析。

本文分析主要集中在密码算法结构，目的是弥合硬件设计人员和密码算法开发人员之间的差距，帮助架构设计者灵活实现高性能专用密码处理器。不同于已有研究关注算法和结构优化，本文研究提供了更为广泛的研究空间探索，使得研究和开发人员能够根据不同的目标算法集合，配置合适的开发环境和硬件结构。

#### 2.2.1 分组密码算法

本文通过整理现有参考文献[1, 55-58]研究的分组密码算法，同时面向 Ipsec、TLS/ SSL、WTLS、SSH、S / MIME 和 OpenPGP 等安全协议和 OpenSSL、GNU 等加密库，统计分析总计 105 种分组密码算法的规格、结构和特征，如表 2-1 所示。

表 2-1 分组密码算法列表

编号	算法	表个数	输入位数	输出位数	每轮并发数	算法执行轮数	密钥长度	分组长度
1	AES	1	8	8	16	10	128	128
						12	196	
						14	256	
2	DES	8	6	4	1	16	56	64
3	Blowfish	4	8	32	1	16	32~	64
							448	
4	Camellia	4	8	8	1	18	128	128
						24	196	
							256	
5	CAST-128	4	8	32	1	12	40~	64
						16	128	
6	GOST	8	4	4	1	32	256	64
7	KASUMI	1	7	7	1	8	128	64
							196	
8	SEED	2	8	8	4	16	128	128
9	Twofish	4	8	8	2	16	128	128
							192	
							256	
10	Serpent	8	4	4	32	32	128	128
							192	
							256	
11	ICE	1	10	8	2	16	64	64
12	MARS	1	9	32	16	32	128	128
							192	
							256	
13	3-Way	1	3	3	1	11	96	96
14	Akelarre	0	0	0	0	4	128	128
15	Anubis	1	8	8	16	12	128	128
						13	160	
							14	
						15	192	
							224	
						16	256	
							288	
16	BaseKing	-	-	-	-	11	192	192
17	ARIA	2	8	8	16	12	128	128
						14	192	
							256	
							256	
18	CAST-256	4	8	32	1	48	128	128
							160	
							192	
							224	
							256	
19	CIKS-1	-	-	-	-	8	256	64
20	CIPHERUNICORN-E	4	8	8	1	16	128	64
21	CIPHERUNICORN-A	4	8	8	1	16	128	128
						16	192	

						16	256	
22	CLEFIA	4	4	4	8	36	128	128
						44	192	
						52	256	
23	CMEA	-	-	-	-	3	64	16~ 64
24	COCONUT98	1	8	24	1	8	256	64
25	Crab	-	-	-	-	-	80	8192
26	CRYPTON	2	8	8	1	12	128	128
							196	
							256	
27	C2	1	8	32	1	10	56	64
28	CS-Cipher	1	8	8	1	8	128	64
29	DEAL	8	6	4	1	6	128	128
							192	
							256	
30	DESX	8	6	4	1	16	56	64
31	DFC	1	6	32	1	8	128	128
							192	
							256	
32	E2	1	8	8	1	12	128	128
							192	
							256	
33	FEAL	-	-	-	-	4	64	64
34	FEALNX	-	-	-	-	8	128	64
						32~		
35	FEA-M	-	-	-	-	1	4094.2	2096
36	FOX	1	8	8	16	16	0~	64
							256	
37	Frog	-	-	-	-	8	128	128
							192	
							256	
38	Grand Cru	1	8	8	16	10	128	128
39	Hasty Pudding cipher	-	-	-	-	-	可变	可变
40	Hierocrypt-3	1	8	8	4	6.5	128	128
							7.5	
							8.5	
41	Hierocrypt-L1	1	8	8	4	6.5	128	64
42	IDEA	-	-	-	-	9	128	64
43	KeeLoq	-	-	-	-	528	64	32
44	KHAZAD	8	8	8	1	8	128	64
45	Intel Cascade Cipher	1	8	8	16	10	128	128
							128	
46	KLEIN							
47	KN-Cipher	-	-	-	-	6	198	64
48	Ladder-DES	8	6	4	1	4	224	128
49	LED	1	8	8	8	32	64	64
						48	128	
50	LOKI97	1	13	8	1	16	128	128
							192	

		1	11	8	1		256	
51	Lucifer	2	4	4	8	16	48	48
							64	32
							128	128
52	M6	-	-	-	-	10	40~	64
							64	
53	M8	-	-	-	-	10	-	64
54	MacGuffin	8	6	2	1	32	128	64
55	Madryga	-	-	-	-	8	~	24
56	MAGENTA	1	8	8	1		6	128
							8	196
							8	256
57	Mercy	1	8	8	1	6	128	4096
58	Mesh	-	-	-	-		8.5	128
							10.5	192
							12.5	256
59	MMB	-	-	-	-	6	128	128
60	MULTI2	-	-	-	-	~	64	64
61	MultiSwap	-	-	-	-	2	374	64
62	New Data Seal	2	4	4	1	16	2048	128
63	NewDES	8	6	4	1	17	120	64
64	Nimbus	-	-	-	-	5	128	64
65	Noekeon	1	4	4	32	16	128	128
66	NUSH	-	-	-	-		9	128
							17	192
							33	256
67	NXT	1	8	8	16	16	~256	64
								128
68	PRESENT	1	4	4	16	31	80	64
							128	
69	PRINCE	1	4	4	16	12	128	64
70	Q	1	8	8	16		8	128
							9	192
		1	4	4	32		256	
71	RC2	-	-	-	-	18	8~	64
							1024	
72	RC5	-	-	-	-		1~	0
							12	32
							255	64
73	RC6	-	-	-	-	20	128	128
							196	
							256	
74	REDOC III	-	-	-	-	~	~20480	80
75	SAFER K/SK	2	8	8	1	8	128	64
							64	
76	SAFER+/++	1	8	8	8	8	128	64
		1	9	8	8			
77	SC2000	1	4	4	32	6.5	128	128
		1	6	6	4	7.5	192	
		1	5	5	8		256	

78	SEED	2	8	8	1	16	128	128
79	SHACAL		-	-	-	-	128	160
80	SHACAL-2		-	-	-	-	512	256
81	Shark	8	8	8	1	6	128	64
82	Skipjack	-	-	-	-	32	80	64
83	SMS4	1	8	8	4	32	128	128
84	Spectr-H64		-	-	-	12	256	64
85	Square	1	8	8	16	8	128	128
86	SXAL	1	8	8	8	8	64	64
87	TEA	-	-	-	-	1~	128	64
						64	128	64
88	Threefish	-	-	-	-	72	256	256
						72	512	512
						80	1024	1024
89	UES	8	6	4	1	48	128	128
							192	
							256	
90	Xenon	-	-	-	-	16	128	128
							192	
							256	
91	Xmx	-	-	-	-	2~	~	~
92	XTEA	-	-	-	-	1~	128	64
						64	128	64
93	XXTEA	-	-	-	-	58~	128	64~
94	Zodiac	2	8	8	1	16	128	128
							192	
							256	
95	Speed	-	-	-	-	64~	64~	64
						48~	64~	128
						48~	64~	256
96	BassOmatic	8	8	8	1	1	8~	2048
						8	2048	
97	BATON	-	-	-	-	-	320	96
								128
98	Chiasmus	-	-	-	-	12	160	64
99	Iraqi	1	8	8	16	5	160	256
100	Khufu and Khafre	1	8	32	1	16	512	64
		4		8		16~		
101	MISTY1	1	7	7	1	4~	128	64
		1	9	9		8		
102	Red Pike	-	-	-	-	~	64	64
103	Simon	-	-	-	-	32	64	32
						36	72	48
						36	96	
						42	96	64
						44	128	
						52	96	96
						54	144	
						68	128	128
						69	192	

						72	256	
104	Speck	-	-	-	-	22	64	32
						22	72	48
						23	96	
						26	96	64
						27	128	
						28	96	96
						29	144	
						32	128	128
						33	192	
						34	256	
105	Treyfer	1	8	8	1	32	64	64

通过对以上 105 种分组密码的分析可见：

- (1) 密码设计基于相似的设计理论。主要分为 Feistel 网结构和 SP 网络结构。基于 Feistel 网络结构或扩展 Feistel 网结构设计的分组密码，如 DES、FEAL、Lucifer、LOKI、GOST、DFC、E2、Twofish、CAST、Blowfish、MARS 及 RC6 等；基于 SP 网络结构的分组密码，如 AES、CRYPTON、SAFER、RIJNDEAL 及 SERPENT 等。
- (2) 加密和解密过程的算法结构类似。分组密码的加解密算法结构非常规整，针对相同的算法其加解密过程仅有少数操作的不同，以及常数和 S-box（替换盒）的差异。
- (3) 加密和解密的流程相似，都分为初始变换、中间变换和末尾变换。初始变换和末尾变换仅执行一次，其计算过程与中间变换有较大的差异。中间变换包含多轮的密码计算，每轮计算的结构基本相同，每轮计算的输出数据直接进入下一轮。
- (4) 分组数据的分组位宽之间具有计算并行潜力。受限传统处理器机器位宽的限制，需要将一组分组数据分拆成为几个相同或不同的更小粒度的计算数据。对于位宽高达 (64bit~128bit) 的分组数据，可以研究开发分组数据的位宽并行性。
- (5) 加解密的计算过程中存在强烈的数据 RAW (写后读) 相关性。不仅在每轮变换的计算操作中存在紧密的 RAW 数据相关性，而且各轮变换之间也存在明显的 RAW 关系。RAW 的数据依赖关系使得不能对算法进行横向展开而必须串行顺序执行。
- (6) 加解密过程具有单向、开环的计算关系。分组密码算法的条件判断分支少，计算操作之间没有数据依赖的反馈关系，可以方便的进行计算流水线的分割。

以上总结了分组密码算法的基本特征，然后由于每个分组密码算法都具有不同的结构和特征，因而需要通过分析提炼算法的共同特征以及差异化需求，用以指导可重构处理器的架构优化。

## 2.2.2 分组密码特征

本文针对分组密码算法的差异化特征，主要从以下几个方面对算法进行分析：算法轮数、密钥长度、分组位宽、操作类别。具体的结果和详细的讨论将在下面的章节中展开。

### 2.2.2.1 算法轮数

分组密码算法的计算轮数直接影响算法顺序执行过程的流水线展开。除了考虑算法在每轮计算中存在的操作差异，计算轮数将制约算法在特定硬件架构中的展升级数。对于硬件资源受限的密码

处理器，完整的算法实现需要对硬件采用时分复用的方法，每个时间片中完成部分算法轮数。因而，对于理想分组密码算法实现能够获得的最高性能 1BPC (Block per Cycle)，算法划分的时间片数量越多，硬件能够实现的性能越低。对于算法在特定硬件架构中的时间片数量为 N，计算性能为  $1/N$  BPC。

分组密码算法的执行轮数可以分为两类：固定执行轮数和可变执行轮数。固定执行轮数的密码算法如 DES 算法包含 16 轮，AES 算法针对不同的密钥长度分为 10、12 和 14 轮。可变执行轮数的密码算法如 RC5，可以实现从 1~255 轮变化，主要采用 12 轮进行计算；TEA 算法可以实现 1~64 轮变化。

从表 2-1 中可知，可变算法轮数占据分组密码算法总量的 16%。其中，可变算法的执行轮数包含有限范围，如 RC5<sup>[59]</sup> 算法的轮数 1~255 轮。同时包含无限范围，如 Speed 算法在分组长度为 64bit 时，算法执行轮数大于 64 轮；在分组长度为 128bit 或 256bit 时，算法执行轮数大于 48 轮。

固定轮数算法作为分组密码算法中主要的算法类型，占据算法总量的 84%。根据不同的密钥长度，同一算法也会体现出不同的算法执行轮数，如 AES<sup>[3]</sup>、Camellia<sup>[60]</sup>、Twofish<sup>[61]</sup>、Serpent<sup>[62]</sup> 等。图 2-1 给出了固定轮数算法中不同轮数的分布情况。从图中可以看出，高达 96.59% 的分组密码算法执行轮数集中在小于 64 轮的范围内，仅有 3 个算法 (KeeLoq、Threefish<sup>[63]</sup> 和 Simon) 在 7 个条件下算法执行轮数大于 64 轮。进一步可知，59.09% 的分组密码算法执行轮数主要集中在 8、16 和 32 轮，分别占据所有执行轮数的 23.86%、23.86% 和 11.36%。掌握分组密码执行轮数的分布趋势将指导可重构处理器硬件优化设计。

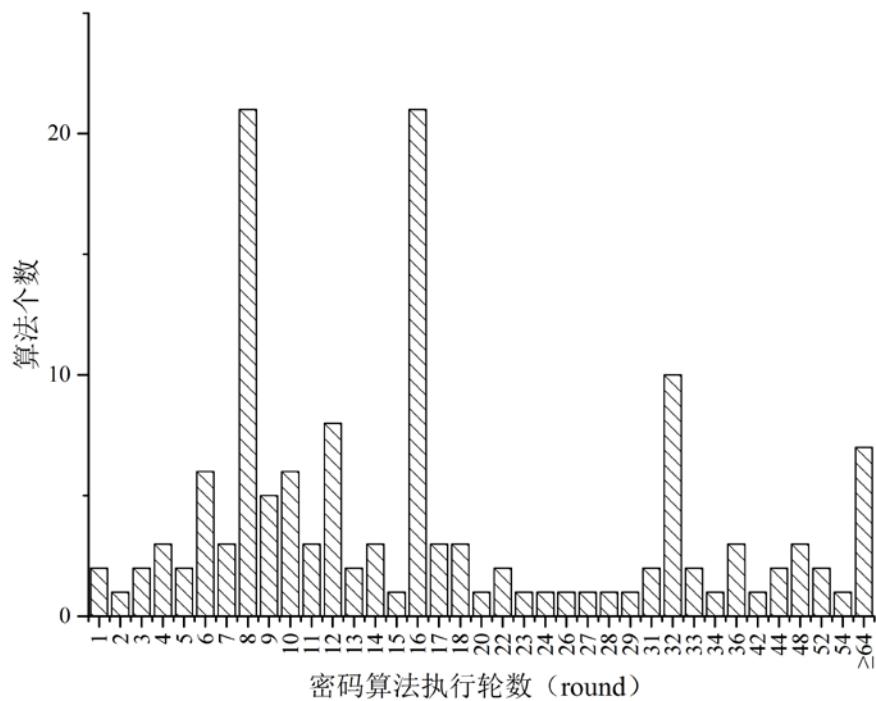


图 2-1 不同算法执行轮数的数量分布

### 2.2.2.2 密钥长度

密码算法的密钥长度影响到密码算法在数学解析过程的加密强度。对于相同的算法，密钥长度越长，加解密所需的计算过程和计算资源都会增加。密码算法通过扩展密钥获得子密钥参与加解密的处理过程。子密钥的扩展过程可以是简单的复制，也可能要进行大量的计算 (如 Blowfish, LOKI97)。

虽然部分算法的密钥扩展过程计算复杂，但是由于仅需进行频度很低的单次或多次计算，其计算量相对于海量明文数据可以忽略。

与算法的执行轮数类似，密钥长度可包含两类：固定密钥长度和可变密钥长度。从表 2-1 中可知，固定密钥长度和可变密钥长度的算法分别占分组密码总数的 11% 和 89%。固定密钥长度的密码算法其密钥长度为一个或多个离散点，如 ICE<sup>[64]</sup>和 CMEA<sup>[65]</sup>算法密钥长度为固定 64bit，Simon 和 Speck 算法的密钥长度则从 64bit 开始，根据不同的分组块长度和算法执行轮数的变化，最长达到 256bit。可变密钥长度的密码算法其密钥长度为封闭区间或半开放区间内连续变化值，如 Blowfish 算法的密钥长度可在 32~448bit 之间选择，FOX<sup>[66]</sup>算法的密钥长度可在 40~64bit 之间变化。Speed 算法的密钥长度可以选择大于 64bit 的任意值，REDOC III 算法的密钥长度则可以在小于 20480bit 的范围内任意选择。

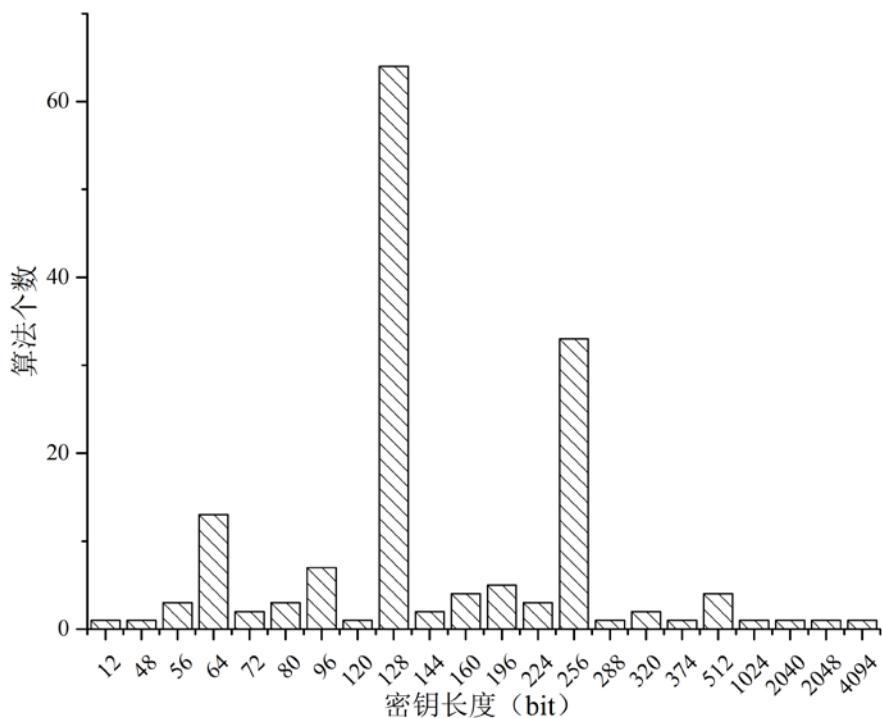


图 2-2 不同密钥长度的算法个数分布

图 2-2 给出了分组密码算法中密钥长度的分布情况。固定密钥长度的密码算法中，密钥长度的分布范围极大（12 轮到 4094 轮）。密钥长度大于 1024bit 的分组密码算法仅占很小的比例约 2.60%，超过 92.21% 的分组算法的密钥长度小于等于 256bit。进一步可知，71.43% 的分组密码算法的密钥长度主要集中在 128 和 256bit，分别占据所有执行轮数的 41.56% 和 21.43%。这是由于 64bit 密钥长度已经无法满足密码算法抗攻击的安全强度。在折中考虑算法的加解密性能和硬件实现复杂度的基础上，目前主流分组密码算法的密钥长度为 128bit，并进一步的增加到 256bit。

### 2.2.2.3 分组位宽

分组位宽是分组密码算法单次处理明文数据的最小单位，其位宽长度决定了密码算法单次能够并行处理的最大数据量。分组位宽是影响密码处理器硬件设计的关键因素之一。除了少数算法支持可变的分组位宽，如 CMEA 算法支持分组位宽以 Byte 为单位变化，范围在 16bit~64bit；XXTEA<sup>[67]</sup>

支持最少为 64bit 的任意分组位宽。其他大部分分组密码支持有限的分组位宽，如图 2-3 所示给出分组密码算法的分组位宽分布情况，超过 80.00% 的分组密码算法的分组位宽集中在 64bit 和 128bit，分别占据所有算法分组位宽的 40.80% 和 39.20%。

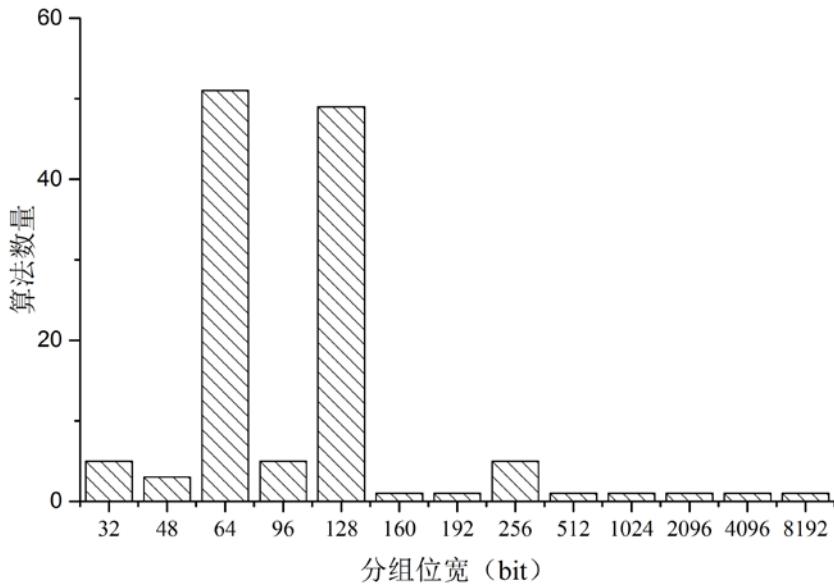


图 2-3 不同分组位宽的算法分布统计

#### 2.2.2.4 操作类别

通过对分组密码算法的研究分析表明，算法的基本操作按照使用过程的频率，可以划分为 5 大类操作：(i) 算术运算，(ii) 逻辑，(iii) 查表替换，(iv) 移位/旋转，以及(v) 置换/扩展。部分需要特殊处理的算法操作由于并没有被广泛使用，因而并未纳入操作类中。下面对每一类基本操作的特征进行描述：

##### (i) 算术操作

算术操作包括不同标量长度的加/减法。分组算法中采用定点无符号整数作为操作数，因此不需要硬件支持复杂的浮点运算操作，也不需要考虑有符号整数操作的符号位扩展和饱和处理等操作。所有的分组密码算法中都没有除法操作，大量算术操作的位宽不超过 32bit，而乘法操作也是采用 32bit 模乘操作。

##### (ii) 逻辑操作

逻辑操作包括基本的比特位操作：异或，与，或和非。逻辑操作是分组密码算法的开始轮数中最常用的操作，超过 95% 的算法在整个轮计算中使用一个或多个逻辑操作，但是仍然有一些算法不需要逻辑运算，如 KLEIN<sup>[68]</sup>、MultiSwap、PRESENT、SWIFFT、RC4 和 Turing。分组密码算法涉及的逻辑操作的数据位宽大多是字节（Byte）或字节的整数倍。

##### (iii) 查表替换

查表操作，也称为 S-box 查找，是一个简单数组索引的替换操作。查找表操作作为分组密码算法中最常用的操作之一，为加解密过程提供非线性计算操作。虽然有文献提供了线性计算来替代查找表操作以减少寄存器的使用，但本文仍然认为他们是查找表操作。这种结构并不限定在有明确定

义的查找表操作的算法中，一些类似于矩阵乘法的操作功能也可以通过预先计算好结果的方式采用查找表实现。查表替换操作作为影响可重构处理器性能和面积的关键操作将在后面详细说明。

#### (iv) 移位/旋转

移位和旋转操作是密码算法中第二常用的操作类型，包括可变位宽的左右两个方向移位和旋转。移位和选择操作能够以可逆的方式改变数据比特位的顺序。移位和旋转操作能够以多达 32bit 的数据进行处理。

#### (v) 置换/扩展

置换和扩展操作能够对多达 64bit 的数据进行处理。由于置换和扩展需要大量的控制信号，因而密码算法中并没有广泛的使用。然而，算法的部分操作仍然依赖于这种操作。对于部分置换操作，尤其是字节方式，也可以表示为查找表操作。

本文对分组密码算法的以下几个方面进行分析：算法轮数、密钥长度、分组位宽、以及操作类别。总结以上影响密码算法在可重构处理器上实现的关键因素，可以考虑从以下方面对可重构密码处理器进行优化设计：

- (1) 利用分组密码类似的设计理论和加解密过程相似的算法结构，设计可以同时满足加解密过程中不同变换的可重构轮计算单元。
- (2) 分组密码算法中仅涉及有限的操作类型，可以采用多个小粒度的基本操作构成重构操作序列。
- (3) 利用分组密码规整的数据位宽，开发分组密码横向的并行处理能力，在实现上完全或部分展开可以获得分组计算最大的并行度。
- (4) 虽然加解密数据存在强烈的“写后读”数据相关性，但是利用单项、开环的计算特点，可以设计具有纵向深度的重构流水线计算架构。

其中，分组密码算法复杂、多轮函数的迭代特点，使得可重构处理器不能提供足够的计算资源，需要全局寄存器缓存加解密中间数据以及密钥数据等计算变量，如何在满足可重构阵列的数据带宽、访存时延的基础上，优化全局寄存器文件的设计参数和结构成为分组密码算法可重构实现的关键。同时，对于分组密码操作类别中查找替换的深入剖析，算法多样的表大小、寻址方式、并发操作和互联结构等特点，使得局部寄存器文件在实现查找替换操作时需要占用大量的计算和面积资源才能完成以上替换操作，严重影响可重构处理器的面积效率。因而，本文将基于分组密码算法的特点，针对影响可重构密码处理器性能和面积的寄存器文件进行优化设计。

## 2.3 面向分组密码算法的可重构处理器设计模型

为了满足可重构密码处理器架构设计空间探索的需求，本文实现的可重构处理器设计框架包括不同精度层次的两个模型：利用 SystemC 构建的周期精确高层行为级模型以及利用 Verilog HDL 实现的寄存器传输级 RTL 模型。SystemC 模型仅对可重构处理器的核心模块进行建模，保持重构处理器完整的接口信号，用于对算法的功能级仿真。通过将访存延时抽象为响应参数，将寄存器容量和互联抽象为预定义架构参数，SystemC 模型可以在功能级探索架构空间的最优设计参数。RTL 模型

是对 SystemC 模型的进一步详细设计，不仅可以验证可重构处理器设计功能的时序，而且可以利用物理综合工具获得功耗和面积的信息。

### 2.3.1 可重构密码处理器行为级模型

为了实现架构设计探索及评估，SystemC 模型能够准确描述硬件特征并且易于扩展。SystemC 通过扩展 C++语言的类结构，引入多进程并发、定时事件和硬件数据类型等重要概念对硬件进行建模描述。通过模块和进程的概念，SystemC 将复杂的系统模型细化成一系列进程的实现。根据不同的需求，SystemC 可以有效的利用三种进程：SC\_METHOD、SC\_THREAD 和 SC\_CTHREAD 来仿真目标系统的行为。

可重构处理器 SystemC 模型采用并行模拟方式，根据系统组件分为可重构阵列流水、数据流水模拟、配置流水模拟、访存模拟等。本文实现的可重构处理器 SystemC 模型首先抽象出各个模块的共性，基于这些共性为每个模块定义各种基类以及模板类，然后再根据各个模块的实际特性派生出具体的子类，通过在模型中调用和例化不同的子类实现模型架构扩展性的低成本实现。为了提高建模效率以及模型的仿真性能，采用基于函数指针的方式来进行可重构处理器的建模。

可重构处理器 SystemC 模型的总体结构如图 2-4 所示。模型包括了 6 个顶层模块：

- (1) 可重构计算阵列 (Reconfigurable Computing Array, RCA);
- (2) 数据存储器 (Data Memory, DM);
- (3) 数据流控制器 (Data Flow Controller, DFC);
- (4) 配置存储器 (Context Memory, CM);
- (5) 配置流控制器 (Context Flow Controller, CFC);
- (6) 系统接口 (SystemC Connect Interface, SCI)。

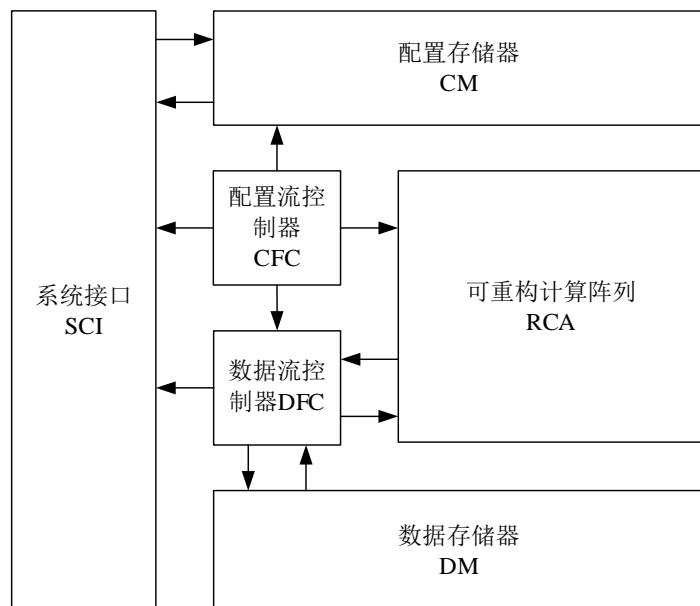


图 2-4 可重构处理器 SystemC 模型结构框图

可重构计算阵列 RCA 是可重构处理器的核心计算部件，负责动态实现并行计算的循环核心。重构阵列计算阵列 RCA 的结构如图 2-5 所示，包含重构计算单元 (Reconfigurable Cell, RC)、互联拓扑路由和阵列内的局部寄存器文件 (Local Register File, LRF) 资源。

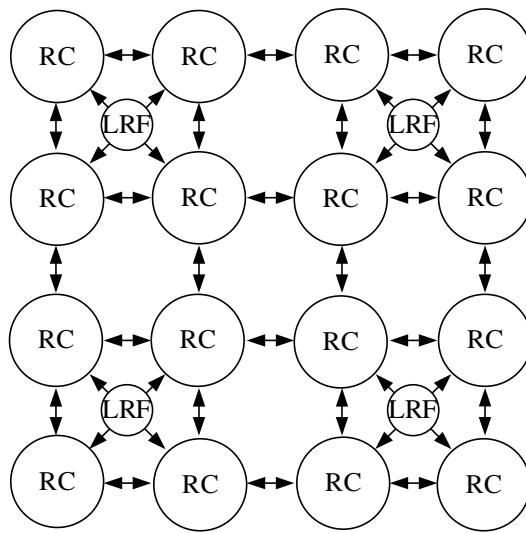


图 2-5 可重构计算阵列的结构示意图

作为组成可重构计算单元 RC 的主要功能主体, 功能单元 (Function Unit) 负责实现基本的逻辑和代数计算, 例如加、减、乘、逻辑与、逻辑或、逻辑非等操作。重构计算单元 RC 包含数据输入输出端口、预测位输入输出端口以及配置信息输入端口, 根据应用需求的不同可以选择多种端口设计参数。RC 在计算中的具体操作行为根据配置存储器 (Context Memory) 变化在循环迭代中动态切换计算配置。在参数化的架构实现过程中, 通过对 RC 的执行操作进行宏定义设计, 可以根据设计需求进行灵活的功能定制。

以可重构计算阵列 RCA 的建模过程为例进行说明。首先, 利用 SC\_THREAD 构建可重构基础模块 sc\_module, 通过在 sc\_module 调用函数指针指向不同的函数实现 RC 功能的改变。可重构基础模块 sc\_module 由一个 sc\_mutex 管道和一个功能进程来模拟可重构处理行为。管道 sc\_mutex 用来保护共享资源, 避免多个进程同时读写共享资源, 导致系统行为的不确定。如图 2-6 所示, 可重构计算单元 RC 模型是一个派生于 sc\_module 的模块, 模型中包括保存配置信息的寄存器 (Configuration Register), 以及实现可重构计算功能的 sc\_thread 型进程。模型中没有实现具体的算术和逻辑运算功能, 与计算相关的函数在功能库 (function library) 中定义。RC 模型进程根据配置寄存器中的值, 将函数指针指向不同的功能函数从而实现计算功能的动态改变。

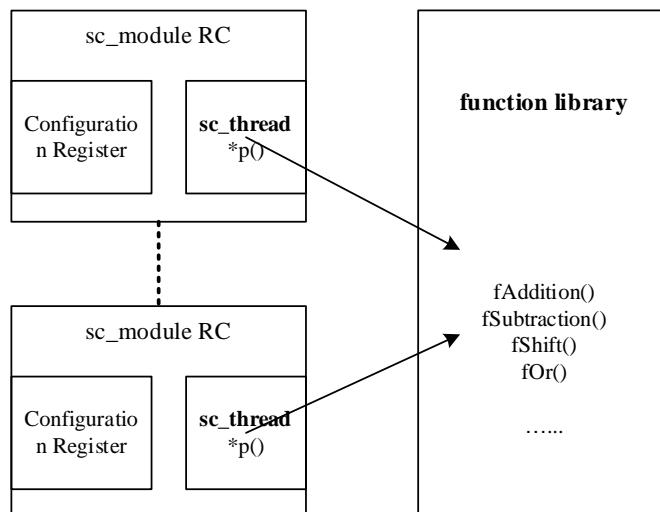


图 2-6 基于指针的重构单元模型

可重构计算阵列 RCA 最基本的局部互连是可重构计算单元 RC 的数据通路选择器 MUX。为了使 RC 的功能单元以及互连结构的选择可以相互独立，将 RC 的运算功能部分（ALU）与数据选择部分（MUX）分为两个子模块，分别对应不同的函数指针。每个 RC 拥有各自的 MUX 子模块，用于实现与相邻 RC 的互连，同时选择当前 RC 的输入数据来源。如图 2-7 所示，仿真开始前进行端口进程的初始化，MUX 模块与邻近的所有允许互连的 RC 都相连接；仿真开始后，根据配置寄存器的配置信息从可选的互连端口中选择对应的互连方式。

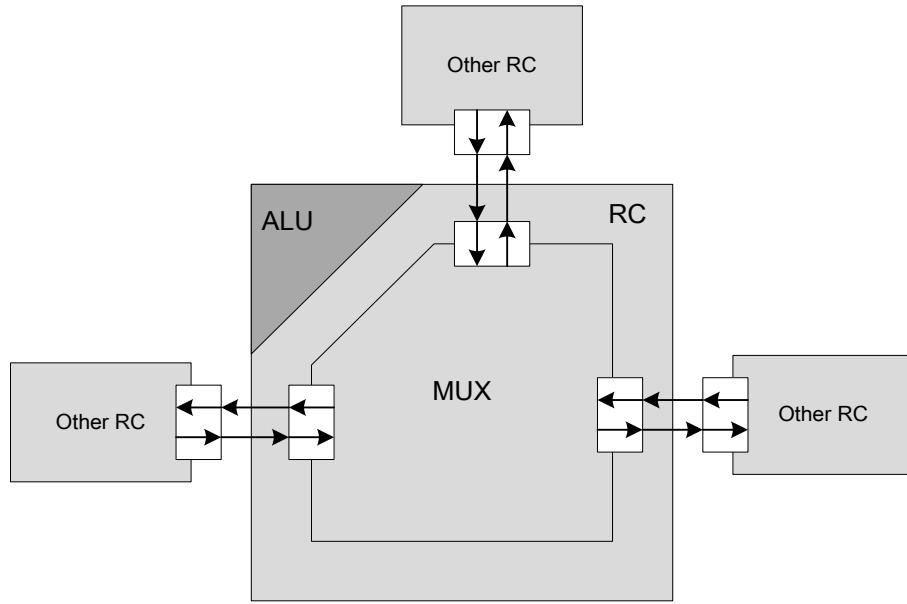


图 2-7 重构单元的互联建模方法

通过 SystemC 构建的功能级仿真模型，可以对可重构处理器的核心——可重构阵列的架构参数进行详尽的设计空间探索。能够在硬件实现之前，比较不同算法在可重构阵列不同设计参数下的性能，最终确定性能和面积约束下最优的设计参数。

### 2.3.2 可重构密码处理器寄存器传输级模型

寄存器传输级（RTL）模型是基于 SystemC 模型获得的设计参数进行的进一步详细设计，在寄存器传输级的实现过程中不仅要考虑硬件实现的电路细节，而且要综合考虑设计对性能、面积和功耗的影响。

本文基于如图 2-8 所示面向分组密码算法的可重构架构模型。模型包含三个核心部件：可重构计算引擎、数据流控制单元和配置流控制单元。可重构计算引擎是密码处理器实现算法的计算核心，其中计算阵列、全局寄存器文件和局部寄存器文件对计算性能具有关键影响。配置流控制单元负责可重构阵列重构配置信息的加载、更新和调度等操作，实现配置文件在计算阵列上的动态重构。数据流控制单元负责实现重构阵列的数据传输以及阵列间的数据交换行为，阵列的数据请求转化为系统总线接口访存协议。注意到行为级和寄存器传输级模型在结构上的差异，配置流控制单元包含 SystemC 模型中配置流控制器和配置存储器以及部分系统接口的功能；数据流控制器单元包含 SystemC 模型中数据流控制器、数据存储器和部分系统接口的功能。

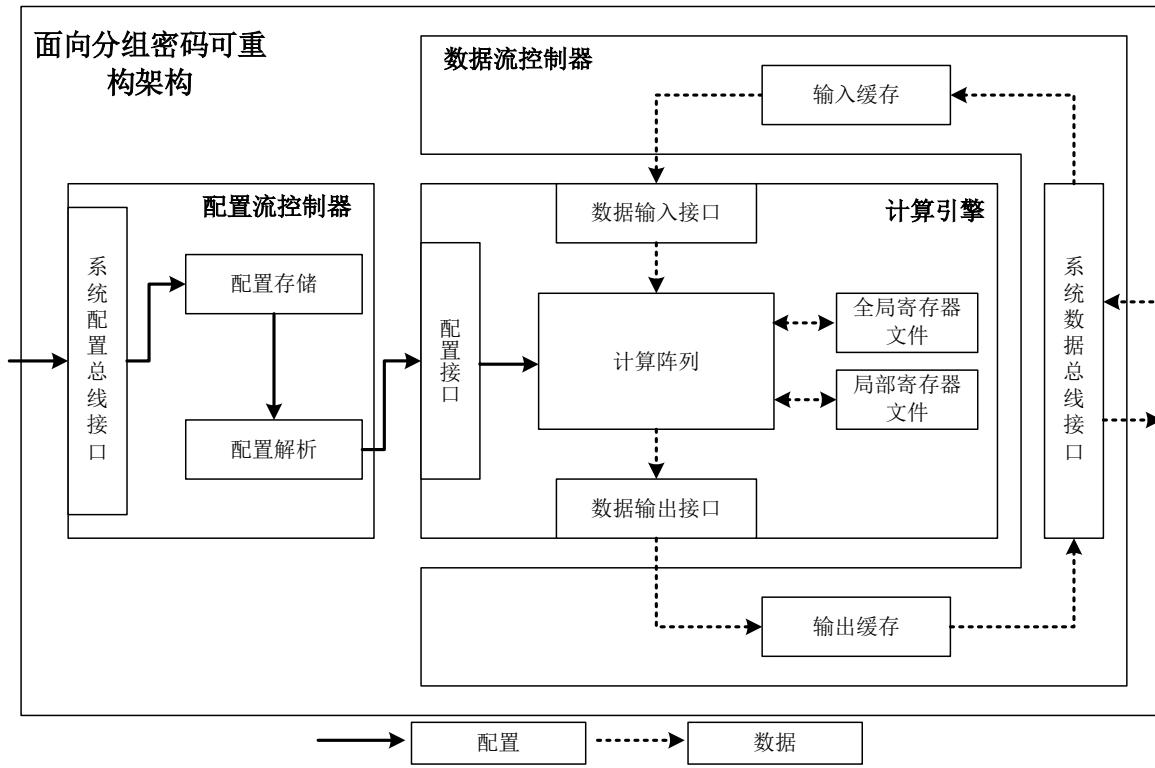


图 2-8 可重构密码处理器 RTL 模型结构框图

整个可重构密码处理器的工作流程如下：系统上电之后，配置信息由系统配置总线接口加载到片上配置存储器中，利用配置解析单元处理配置存储器中相应的配置信息，输出到可重构计算阵列的配置接口，并根据算法差异在计算过程中实现动态配置。计算数据通过系统数据总线接口从外部存储器中加载到输入缓存（INFIFO）中，并在完成整个密码算法的所有计算流水之后，将计算结果写入到输出缓存（OUTFIFO）中。可重构计算阵列在配置信息和计算数据流的驱动下实现密码处理，阵列计算的中间结果利用全局寄存器进行缓存。

### 2.3.2.1 可重构计算引擎

计算引擎为可重构处理器中主要运算模块，其功能模块包括结构框图 2-9 所示的可重构计算阵列、全局寄存器文件 GRF（Global Register File）和局部寄存器文件 LRF（Local Register File）。其他配置接口，以及数据输入输出接口仅实现简单的数据连接和组合等简单的组合逻辑操作。

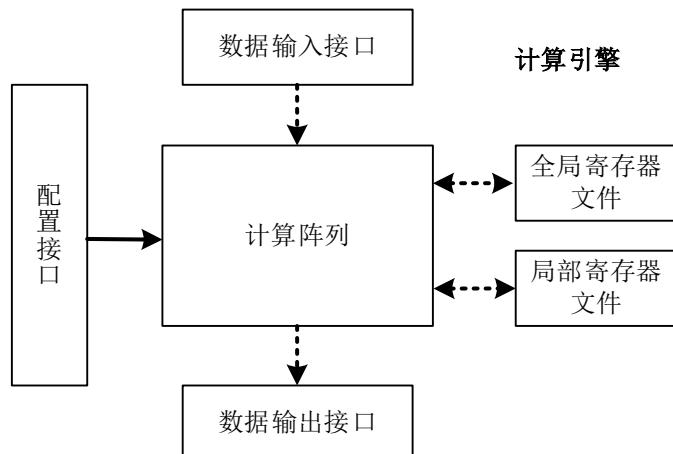


图 2-9 可重构计算引擎结构框图

### (1) 可重构计算阵列

可重构计算阵列的基本单元是行，整个重构阵列利用多行重构单元级联组成，每一行的输出和下一行的输入模块直接相连。每一行都可以从 FIFO 或者全局寄存器文件读入数据，并且都可以将数据读出到 FIFO 或者全局寄存器文件中。除首行外，每行均可选择由上一行输出作为输入；除尾行外，每行输出均可作为下一行输入。可重构阵列的行单元主要包括：ALU、互联单元、数据端口和控制单元。

#### (A) ALU

可重构阵列行单元包含多个算术逻辑单元 ALU (Arithmetic and Logic Unit)，每个 ALU 包含多个输入端口和输出端口。如图 2- 10 所示是 ALU 的结构示意图。ALU 接受 Config\_in 的配置，实现计算，输出数据 Out\_0。Out\_1 的结果可以是 In\_3 的旁路输出或者和 Out\_0 相同，由配置内容决定。与此同时，ALU 可以通过 Cin 和 Cout 的级联，多个 ALU 级联可以实现高位加法器。

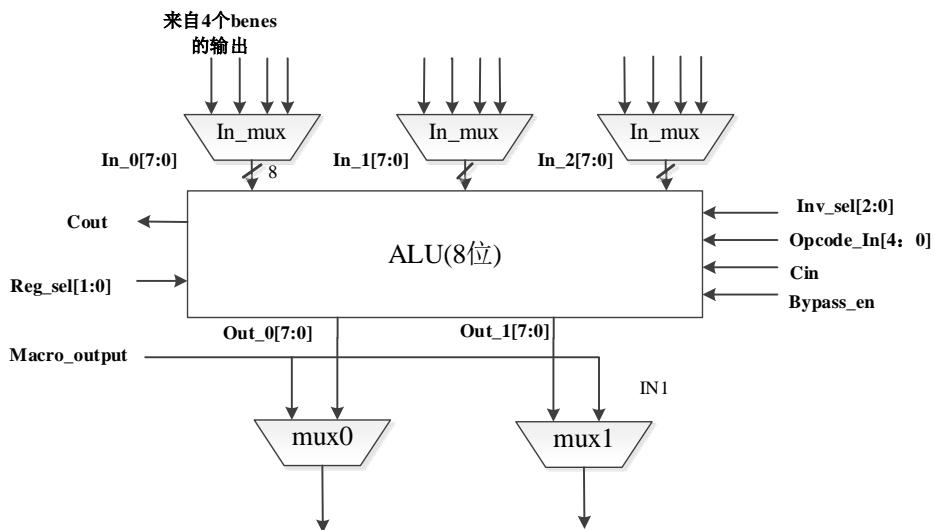


图 2- 10 ALU 结构示意图

#### (B) 互联单元

互联拓扑结构是影响可重构处理器灵活度、资源利用率以及性能的关键因素之一。除了根据不同的运算操作改变可重构单元的计算行为，可重构阵列可以通过选择不同的互连通路实现算子级别的重构。可重构计算阵列采用的二维互联拓扑结构实现计算数据和控制信号的传输，可以将其大致分为两类：①动态互联，通过路由连接处理单元或者路由部件的数据通道，根据数据的路由信息，动态仲裁数据的路由目的地；②静态互联，连接和通讯关系不会依赖传输数据信息而变化，物理连接完成之后通过开关矩阵实现数据通路的选择和开关控制。本文设计的传输级模型采用静态互联的拓扑结构，每个 RC 利用硬连线或寄存器文件与周围临近的 RC 实现互联。可重构阵列行单元中的互联单元主要由两个部分组成：行输入多路选择器和 Benes 网络。

#### (C) 数据端口

通过输入模块（以下称为 Load）和输出模块（以下称为 Store）将数据在可重构阵列行单元各行

之间传递。其中 Load 模块负责把数据从输出缓存 (INFIIFO)、上一行或者全局寄存器文件中读进来，传递给 Benes 网络处理。Store 模块负责把数据从计算阵列行结构的数据输出到输出缓存(OUTFIFO)、下一行或者全局寄存器文件中去。同时，Load 和 Store 模块接受配置，根据配置信息的内容发出读请求和写请求，从而获得相应的数据。

#### (D) 控制单元

可重构阵列行单元的控制单元，主要负责两部分的工作：配置控制信息的解析和握手信号的产生。一要根据获得的配置信息解析，得到计算阵列的时序等相关信息；二要根据解析得到的时序信息，产生相应的握手信号与外部的配置控制器进行交互。只有在计算阵列与配置控制器握手成功后，才能接受配置，接受完配置之后，再通过配置控制器给出开始信号，整个计算阵列开始运行。

#### (2) 全局寄存器文件

全局寄存器文件分别有多个独立的读端口与写端口，每个端口都可以对寄存器堆中的所有数据进行读写操作。寄存器文件的写端口都支持以字为单位的 MASK (屏蔽) 操作，即每个写端口都有多位 MASK 信号控制屏蔽使能。MASK 信号定义高电平有效，低电平保持不变。本架构允许多个端口对同一个地址进行读操作，但是禁止同时对一个地址进行写操作。

#### (3) 局部寄存器文件

局部寄存器文件用于实现密码算法的查找表 LUT (Look Up Table) 结构，其在可重构阵列中的互联结构如图 2- 11。可重构阵列行单元的输出结果可以通过多路选择器到 LUT 中进行查表操作，查找到的数据返回到相应的下一层的输出。(下图只以层间直连的形式说明 LUT 互联结构)

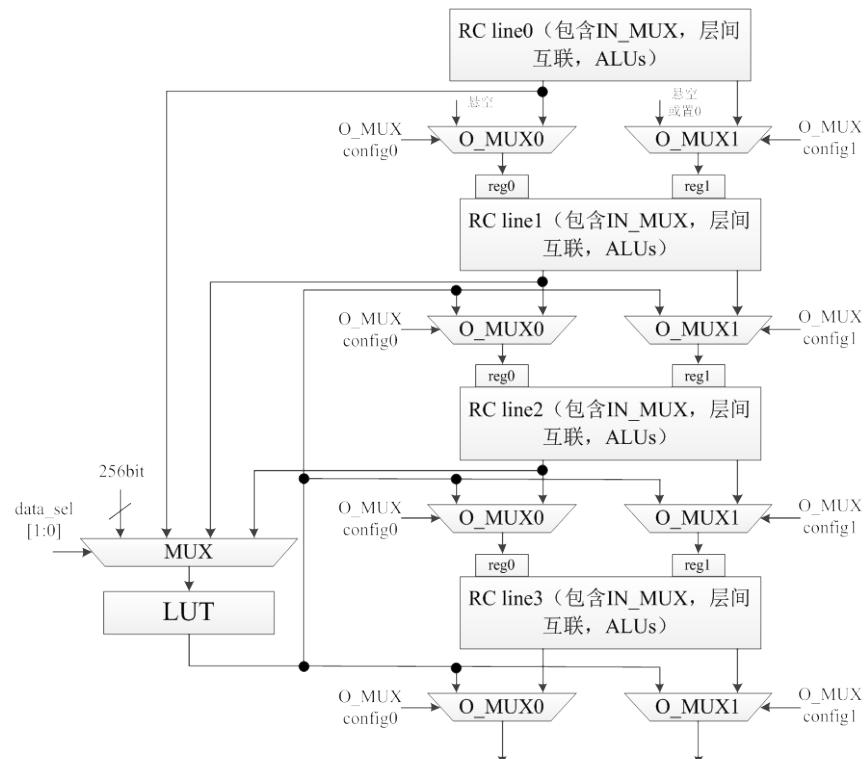


图 2- 11 查找表互联方式

由图 2-11 可见，可重构阵列行单元的计算操作和对 LUT 的查表操作不可以同时进行。LUT 的输入来源通过 data\_sel 实现配置。每 4 行可重构阵列行单元中，可以选择第 0 行、第 1 行、第 2 行的输出作为 LUT 的查表输入，并将输出作为第 1 行、第 2 行、第 3 行的输出。与此同时第 3 行的输出也可以作为 LUT 的查表输入，只是输出只能作为下一个 4 行可重构阵列行单元的第 0 行的输出。

### 2.3.2.2 数据流控制单元

数据流控制单元介于可重构计算阵列和数据存储器之间，根据不同的存储介质的访存特性和接口协议的差异，实现外部存储器、片上存储器。本架构的数据接口采用两个 FIFO 与外部数据进行交互，分别是负责明文输入的输入缓存（INFIFO）和密文输出的输出缓存（OUTFIFO），两者的参数设计宽度均为 128bit，深度为 32，总计大小为 512Byte 的异步 FIFO。整个系统开始工作除了要接收配置控制器的开始信号，还要输入 INFIFO 处于非空状态，输出 OUTFIFO 处于非满状态，加密算法的明文通过输入 INFIFO 进入可重构阵列，经过计算完成之后，加密算法的密文被写入输出 OUTFIFO 中，再被外部读走，完成整个加密过程，与此同时，加密算法的一些初始化数据也从 INFIFO 输入，包括初始密钥和一些常数等。

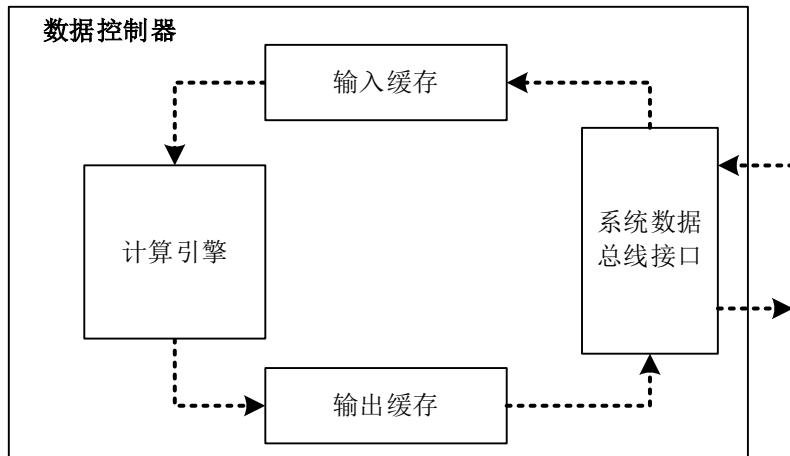


图 2-12 可重构数据控制器结构框图

### 2.3.2.3 配置流控制单元

如图 2-13 所示是配置包存储器、配置组存储器与配置索引解析电路的结构图，由三者共同组成配置控制器，用于完成配置信息的读取，并且发送给计算阵列，同时负责数据流图的配置信息切换，完成整个配置过程。配置包存储器和配置组存储器首先由外部初始化存储内容，初始化完成之后，由外部发送一个启动信号，配置索引解析电路开始从配置包存储器中读取配置包，按照配置包索引的解析结果，从配置组存储器中读取配置组，按照配置包的索引格式组成一张完整的数据流图，发送给可重构计算阵列。

与传统可重构架构中的配置存储器相比，本文会把传统的一份片上配置存储器分割为两个部分：配置组存储器（Configuration Memory, CM）和配置包存储器（Configuration Packet Memory, CPM），两者共同存储算法的完整配置信息。CM 存储分组密码算法的配置信息，包括 PE 的配置、Benes 网络的配置、LUT 的配置、以及数据流控制的配置。配置组存储器中的各类存储器是连续存储的。每类配置的首地址通过配置包（Configuration Packet）中的信息提供给配置索引解析电路。CPM 存储

分组密码算法的配置索引信息。在系统上电之后，配置信息初始化完成之后，配置包被写入配置包存储器中。通过多套配置信息的动态切换，使得可重构计算阵列能够完成远远超过自身阵列大小的并行计算。

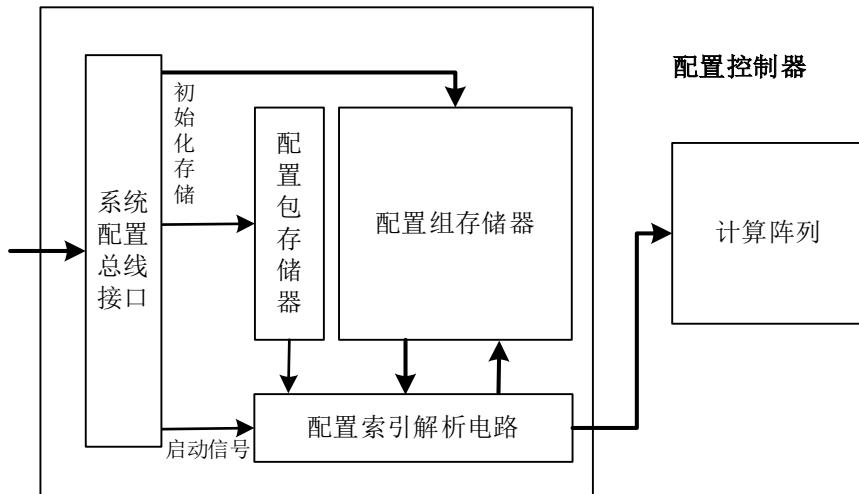


图 2-13 可重构配置控制器结构框图

配置控制器的工作流程如图 2-14 所示，首先系统重置完成之后，完成配置控制器中的配置信息初始化；其次在计算阵列处于可配置状态时，配置解析模块解析配置信息；最后配置控制器发送配置信息给计算阵列从而实现计算引擎的配置，在计算过程中，同样会负责配置信息的切换。

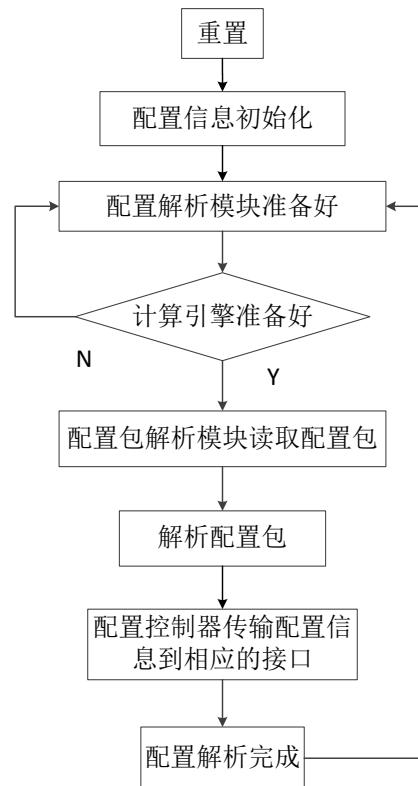


图 2-14 配置控制器工作流程图

## 2.4 本章小结

本章首先收集目前主流分组密码算法，详细分析影响可重构处理器的性能和面积的算法特征，整

理限制可重构处理器设计的关键约束条件，指导可重构处理器的架构实现。然后，介绍面向分组密码算法基于 SystemC 语言实现的功能级仿真模型。最后，详细介绍作为研究实验平台的寄存器传输级模型，说明可重构密码处理器硬件架构中设计的主要核心部件：可重构计算引擎、数据流控制单元和配置流控制单元。



## 第三章 分组全互联的分布式全局寄存器文件研究

### 3.1 前言

可重构密码处理器为了实现高性能和高灵活性的并行计算，提供算法潜在的变化需求，需满足分组密码算法的数据特点：分组数据相互独立和计算数据写后读（RAW）的依赖关系。分组密码算法有不同的执行轮函数量，并且轮函数之间的子算法也不同，导致分组数据在可重构计算阵列上执行周期的不同。同时，受到可重构计算资源的限制，密码算法在有限重构计算阵列上实现时需要进行算法的时空切割。当不同执行周期的密码算法在重构阵列上实现时，不仅要处理多个计算步骤切换而产生的大量临时数据，而且要满足不同算法利用重构阵列实现任意流水级的需求。

全局寄存器文件（GRF）用于可重构阵列内部各模块之间数据交换，起到类似于一级缓存的作用。利用全局寄存器文件可以满足密码算法多轮计算配置切换的高速数据缓存，同时可以实现不同密码算法的任意流水级展开。全局寄存器文件的容量大小对应密码算法在可重构架构限制中切割后的最大执行长度；全局寄存器文件的并发访存能力对应能够提供重构计算阵列的最大数据通量。当全局寄存器的存储容量或并发访存能力受限时，将影响可重构计算阵列的流水线效率，严重时将导致流水线暂停。因而，如何在满足不同密码算法数据需求的条件下，提供高性能面积比的全局寄存器文件设计成为可重构密码处理器研究的关键难点之一。

本章针对全局寄存器文件的设计方法和架构实现进行研究，首先分析全局寄存器文件对可重构密码处理器计算性能的影响，给出具体架构约束下全局寄存器文件性能模型。然后根据目标架构硬件约束和算法集合特征给出全局寄存器文件的设计参数。最后利用算法数据特点对寄存器文件架构进行优化设计，并通过对比实验证明设计的优化结果。

### 3.2 研究现状分析

传统的可重构密码处理器采用集中式、多端口访存的设计来实现循环流水和配置切换，在提供极高的数据带宽和访问效率的同时，也导致整个全局寄存器文件的面积开销巨大。Cryptoraptor<sup>[33]</sup>可重构密码处理器采用一个集中式的全局寄存器文件，每行执行单元通过互联行连接到下一级，每一级都可以直接将结果存入全局寄存器文件。因而，可以利用重构阵列的任意部分实现不同算法的任意流水级。全局寄存器文件采用 1 个包含 256 个 32bit 寄存器的寄存器体 ( $8 \times 32\text{bit}$ )，拥有 80 个 32bit 位宽的读端口，8 个 32bit 位宽的写端口。整个全局寄存器文件的面积为  $1.78\text{mm}^2$ ，占整个密码处理器面积的 28.16%。Cryptoraptor 可重构密码处理器为算法提供了最大并发访存能力，但是也占用了处理器大量的面积资源开销。

MorphoSys<sup>[31, 45]</sup>架构利用双帧缓存（Double Frame Buffer）实现数据传输和计算处理的重叠，通过 Ping-Pang 的数据交换方式降低由于数据传输等待而造成的计算性能影响，满足可重构计算阵列与内嵌处理器的的同时操作。MorphoSys 利用内嵌的 TinyRISC 处理器扩展 DMA 指令完成数据传输，采用猝发访问的方式完成外存数据到帧缓存区之间的数据传输。虽然双帧缓存结构可以有效增加全局寄存器的容量，但是在同一时间仅能为重构计算阵列或内嵌处理器提供一个读端口和一个写端口，

有限的数据带宽严重制约了算法在重构阵列上的流水实现。

ADRES 架构中采用参数化的紧耦合全局寄存器文件 (Global Register File) 作为外部数据的缓存区，同时满足 VLIW 模式和 CGRA 模式相互切换时的数据交换需求。为了满足并行计算的数据需求，减少访存冲突和硬件开销，采用访存队列重排序的机制<sup>[69]</sup>，并且研究了数据在 Multibank 片上存储器的映射方式<sup>[48]</sup>。Kwok, Z.<sup>[50]</sup>探索了 ADRES 架构中私有寄存器和全局寄存器文件对可重构处理系统面积和性能的影响。由于 ADRES 架构中全局寄存器文件的端口受限于紧耦合的 VLIW 的端口限制，即使采用访存队列重排序的方法来缓解计算阵列的访存带宽限制，仍然严重制约了可重构计算阵列的计算性能。

通过以上研究分析可知，可重构全局寄存器文件的研究主要集中在提高阵列的计算性能，同时减少寄存器文件占用的面积资源。如何满足分组密码算法差异化的算法特征，匹配可重构计算阵列提供的计算性能是全局寄存器文件研究的关键。本文将基于密码算法在可重构架构约束下的性能分析结果，给出全局寄存器文件的设计方法和架构实现。

### 3.3 提出全局寄存器文件的性能模型

分组密码算法以轮函数为基本单位进行循环迭代，可重构阵列的执行效率决定了轮函数的性能。受制于可重构阵列的规模，难以满足多种分组密码算法中的各轮函数在可重构阵列上全展开的需求，需要通过全局存储器文件对轮函数中间结果数据进行缓存。由于迭代中计算所需数据的空间排布和寻址方式差异，使得计算核心在阵列循环计算过程和存储空间上体现出不同数据流特征，对全局寄存器文件架构的设计和实现提出挑战。本节面向分组密码算法在可重构阵列映射过程中对中间结果数据的访存需求，基于可重构计算软件流水对性能的影响，对全局寄存器文件进行性能建模。

#### 3.3.1 可重构计算软件流水的原语

为了便于表述本课题对可重构计算抽象所涉及的问题，本节对涉及到的原语进行定义和说明<sup>[70]</sup>。

**循环核心：**并行计算的循环核心满足有向无环图的条件<sup>[71]</sup>，即可以表示为一个单向非周期的数据依赖图 (Data Dependence Graph, DDG)。DDG 表示为  $V$  和  $E$  的集合  $K$ ,  $K = \{V, E\}$ ，其中参数  $V$  和  $E$  定义如下：

$V$  -- DDG 中所有操作节点的集合，对于任意的  $v \in V$  表示重构计算单元支持的操作算子。

$E$ --对于 DDG 中任意有数据依赖的两个节点  $u$  和  $v$ ，用  $e = (u, v) \in E$  表示两个节点的有向数据依赖。

**可重构阵列：**对于一个大小为  $M \times N$  的可重构阵列 (Coarse-grained Reconfigurable Array, CGRA)，其硬件资源图可以表示为  $C = (P, L)$ 。其中， $P$  为重构阵列中所有计算单元  $p_{ij}$  的集合，其中下标  $i, j$  表示  $p_{ij}$  在重构阵列中的位置， $1 \leq i \leq M, 1 \leq j \leq N$ ；对于阵列中的两个计算单元  $p, q \in P$ ， $l = (p, q) \in L$  表示计算单元  $q$  可以使用计算单元  $p$  的计算结果。

**应用映射：**并行计算循环核心在可重构阵列上映射的过程用函数  $\phi: K \rightarrow C$  表示，其中包括两个函数， $\phi_v: V \rightarrow P$  和  $\phi_E: E \rightarrow 2^L$ 。其中， $\phi_v$  是重构计算单元的映射函数，需要将每个计算操作  $v$  映射到阵列的计算单元  $p \in P$ ，映射结果可能会出现某些计算单元不被使用的情况。 $\phi_E$  是一个多值函数，将计

算节点的数据依赖  $e \in E$  映射到一组阵列连接关系的集合  $l \in L$  中。映射过程中，DDG 图所有的计算节点都需要同时满足函数  $\phi_v(v)$  和  $\phi_v(v)$ 。

因此，可重构计算的实现过程可以描述为：对于一个给定的 DDG 图  $K = \{V, E\}$  和 CGRA 阵列  $C = (P, L)$ ，找出映射函数  $\phi(K)$  在 CGRA 阵列的资源约束下 DDG 图的最优映射结果。如图 3-1 给出算法循环迭代在重构阵列的映射过程实例。

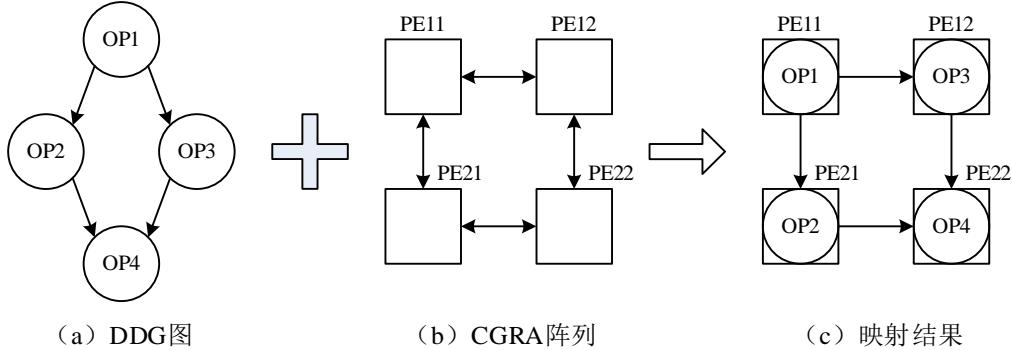


图 3-1 循环迭代的映射实例

### 3.3.2 可重构计算软件流水的性能评估

对于分组密码算法在可重构处理器上的映射实现，可以采用软件流水的计算方式提高执行并行度，缩短运行时间。整个可重构计算流水过程中，在密码算法循环体的一个迭代执行完毕之后才能启动下一个迭代，相邻两个循环体的启动间隔被称为迭代间隔<sup>[72]</sup> (Initiation Interval,  $II$ )。软件流水线包括三部分：载入（Prelogue）、核心（Steady）和排空（Epilogue）。载入代表循环启动到循环稳定所需的启动周期，而排空则代表循环从稳定状态到循环结束的退出周期。在载入和排空阶段计算流水线并没有以最大并行度运行，但是当循环次数足够多时，大部分执行时间将消耗在核心上。此时，核心重复一次需要  $II$  个周期。在软件流水计算过程中，循环体被划分为若干段，各段使用的硬件资源没有冲突。与顺序执行的计算过程相比，软件流水计算过程中存在于相邻迭代的不同段的操作可以同时执行，从而大大减少了循环执行的时间。

图 3-2 给出软件流水的原理示意。子图（a）所示的循环体被划分为时间上相等的 3 段 A、B、C，这 3 段需要的硬件资源不存在冲突。子图（b）给出循环顺序执行的方式，在循环 1 结束之前循环 2 不能启动。3 个循环依次执行，直到所有的循环迭代完成。子图（c）给出软件流水示意。每隔一个迭代间隔时间  $II$ ，启动一个迭代。经过 2 个  $II$  后，有 3 个连续的迭代在同时执行，且依次位于 C 段、B 段、A 段中，此后每隔  $II$  时间各迭代的执行前进一段，当一次迭代开始退出流水线时，同时又有一个新的迭代进入，这样每个  $II$  时间段中执行的操作保持不变，刚好包含原循环体中的所有操作，这一重复出现的稳态即构成核心部分。

分组密码算法在可重构架构上实现的关键性能指标可以通过 Gbps, BPC 和 SD 来评价可重构计算的执行效率和调度效果。性能指标的表达式如下：

$$Gbps = BPC \times BlockSize \times Frequency \quad (3.1)$$

$$BPC = Block/Cycle \quad (3.2)$$

$$SD = BPC/ArraySize \quad (3.3)$$

Gbps，每秒执行十亿比特数 (Giga Bit per Second) 体现了单位时间内处理分组数据量的能力，

Gbps 越大说明处理器的加解密能力越强。其中，Frequency 是可重构处理器的工作主频，与设计结构和工艺参数等实现手段有关。BlockSize 是明文数据的分组位宽，不同的密码算法根据应用场景和设计参数差异存在不同的 BlockSize 大小。

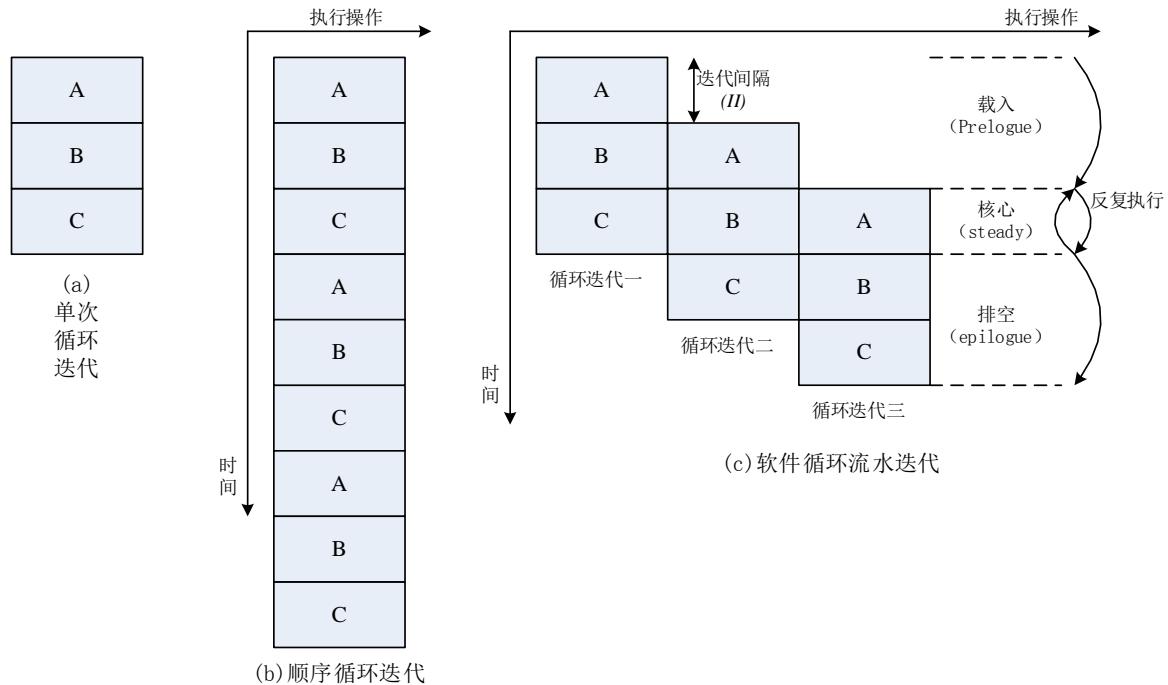


图 3-2 不同计算循环的流水示意图

BPC，单周期分组数密度（Block per Cycle）是表征并行计算能力的性能指标，BPC 由分组数（Block）和执行周期（Cycle）决定。其中，分组数（Block）是算法实现过程中处理分组明文数据的总和；执行周期（Cycle）是处理器处理分组明文所需的周期数。在优化 BPC 的过程中，由于具体应用的分组数目（Block）是确定的，因而优化的关键问题在于减少应用的执行周期（Cycle）。

SD，阵列调度密度（Schedule Density）表征单个计算周期中阵列平均活动的计算单元个数，是比较不同阵列大小的可重构计算并行度的归一化参数。对于固定的重构阵列大小，SD 越大表明应用调度后的计算并行度越高；相同的应用映射在不同大小阵列映射后，SD 越大表明重构阵列的利用率越高。

执行周期（Cycle）是分组密码算法在可重构处理器上并行展开计算实现所需的周期数。密码算法在可重构处理器上的计算效率与执行周期密切相关，执行周期越短说明软件流水的效率越高。式（3.4）为软件流水效率的计算公式，SC 表示循环体被划分的段数（Stage Counts，简称 SC），由循环体本身的特征决定。启动间隔 II 是衡量循环流水性能的重要指标。启动间隔 II 决定了软件流水的效率，II 越小，表明循环执行得越快，软件流水的效率越高。N 为待处理的分组明文数量。 $T_{stall}$  为可重构阵列外部配置流和数据流的准备时间，与系统实现的配置加载方式和数据存储方式有关。

$$Cycle = SC \times II + (N - 1) \times II + T_{stall} \quad (3.4)$$

循环迭代 II 的下限称为最小迭代间隔（Minimum Initiation Interval，minII），受到两个因素的影响：*RecMII* 和 *ResMII*。如式（3.5）所示，minII 取两者最大值。*ResMII* 代表由于阵列中计算资源和互联结构限制而造成的启动间隔下限；*RecMII* 代表由于迭代计算过程中数据依赖限制而造成的启动间隔限制，包括数据在阵列计算过程中的依赖和数据在存取过程中的依赖。

$$minII = max(ResMII, RecMII) \quad (3.5)$$

下面以图 3-3 所示为例,说明密码算法在可重构阵列的流水化运行过程。图中显示了  $SC=3, II=1, N=3$  这种配置条件下, 可重构阵列的软件流水过程。可重构阵列配置映射的结果为  $OP1 \rightarrow PE12$ ,  $OP2 \rightarrow PE11$ ,  $OP3 \rightarrow PE21$ ,  $OP4 \rightarrow PE22$ 。 $T=1$  时,  $PE12$  执行循环第一次迭代的操作  $OP1$ , 并将结果传递给  $PE11$  和  $PE21$ ; 接着在  $T=2$  时刻,  $PE11$ 、 $PE21$  分别执行第一次迭代的操作  $OP2$ 、 $OP3$ , 而  $PE12$  执行第二次迭代的  $OP1$ ,  $PE11$ 、 $PE21$  将计算结果传递给  $PE22$ ;  $T=3$  时,  $PE22$  执行第一次迭代的操作  $OP4$ ,  $PE11$ 、 $PE22$  执行第二次迭代的  $OP2$ 、 $OP3$ , 此时  $PE12$  开始执行第三次迭代的  $OP1$ 。直到执行完所有迭代, 阵列的运行结束。在流水线开始时的载入阶段和结尾时的排空阶段, 流水线没有完全“填满”, 可重构阵列上被映射的处理单元没有都开始运行。可重构阵列通过同时执行循环不同迭代的操作能够获得更高的并行度, 当循环次数足够大可以忽略载入和排空部分时, 本例可以获得的近似等于 4 的 BPC, 而如果顺序执行的话, BPC 只有  $4/3 \approx 1.33$ 。

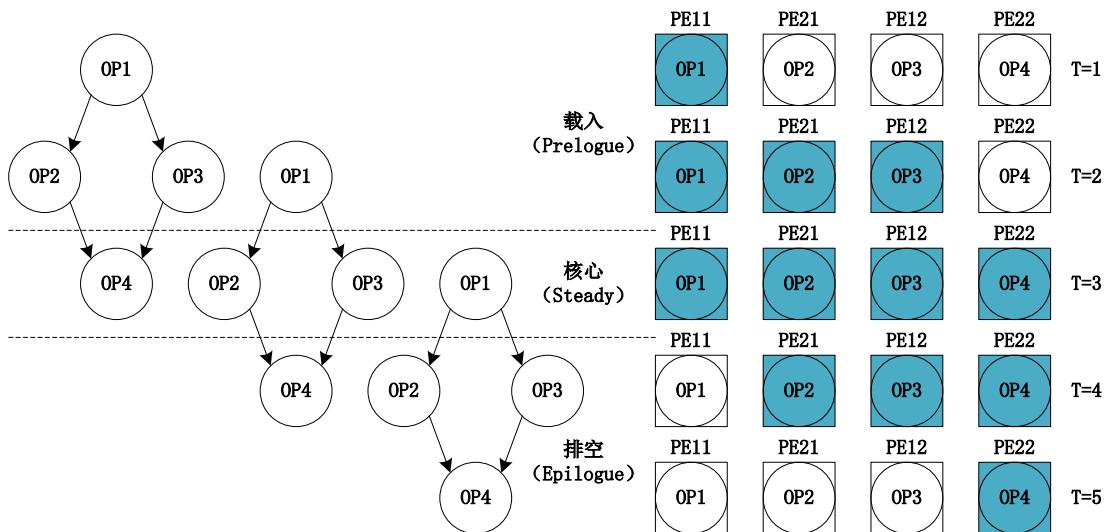


图 3-3 循环迭代在可重构阵列上的软件流水过程

### 3.3.3 基于可重构计算软件流水的全局寄存器文件性能模型

分组密码算法在可重构密码处理器上实现时, 性能受到算法特征和硬件架构的约束条件影响。为了能够全面分析影响计算性能的关键因素, 需要同时考虑分组密码的算法特征和硬件架构的多种限制条件。对影响计算性能的约束条件进行抽象如表 3-1 所示, 影响分组密码性能的关键因素不仅与算法本身的特征有关, 而且与硬件架构的设计参数以及应用中配置调度方法有关。其中, 按照算法的轮函数将分组密码算法分段, 假设轮函数的长度即为算法在阵列上划分的段数  $SC$ 。同时, 可重构阵列的行数为  $RCL$ , 假设每行重构单元均能够在一个迭代间隔 ( $II$ ) 内完成一轮函数的计算处理。全局寄存器文件能够容纳最多的临时数据的行数定义为全局寄存器文件轮数 (GRC)。将配置切换和数据初始化时间定义为切换气泡数 ( $T_{stall}$ )。

表 3-1 影响计算性能的约束条件

参数	标示	说明
轮函数	$SC$	分组密码算法的轮函数
阵列轮数	$RCL$	重构阵列单元的行数
全局寄存器文件轮数	$GRC$	全局寄存器文件的行数
切换气泡数	$T_{stall}$	配置切换和数据初始化时间

本文根据约束条件的不同，将性能情况的讨论按照由简到繁的过程划分为如表 3- 2 所示的三种场景：

表 3- 2 不同架构约束条件的分析场景

标号	重构计算资源说明	全局寄存器资源说明
1	重构计算资源无限	无
2	重构计算资源有限	全局寄存器文件无限
3	重构计算资源有限	全局寄存器文件有限

情况 1 对应于重构计算资源远远大于计算需求的情况。此时，可重构阵列能够容纳所有并行展开的密码计算，在整个计算过程中仅需加载一次配置信息，而且不需使用全局寄存器文件。

情况 2 对应于有限的重构计算资源无法满足密码计算展开的需求，但是拥有足够的全局寄存器文件满足所有的数据缓存需求的情况。此时，需将密码算法拆分为多套配置信息，在整个计算过程中根据配置信息的数量完成多次加载操作。在两次配置切换之间，所有的临时数据都缓存在全局寄存器文件中。

情况 3 对应于实际应用中有限的重构计算资源和有限的全局寄存器文件的情况。密码算法拆分为多套配置信息，仅采用有限的全局寄存器文件用于数据缓存。每次仅能处理全局寄存器文件容量大小的数据量，需要反复切换配置信息以完成完整的算法处理过程。在当前分组数据全部处理完成之后，才能对新的数据进行处理。

### 3.3.3.1 重构计算资源无限的性能模型

当密码算法可以在可重构处理器的计算阵列上完全展开时，数据从阵列外部进入，经过重构阵列处理之后，直接输出阵列外部。此时，整个密码算法在可重构处理器上并行展开处理所需的周期如式（3. 6）所示：

$$Cycle = SC \times II + (N - 1) \times II + T_{stall} \quad (3. 6)$$

执行周期（Cycle）的大小与循环体的段数（SC），迭代启动间隔 II 和阵列准备时间  $T_{stall}$  三个因素密切相关。当待处理的分组密码的明文数量 N 足够大时，执行周期近似等于 N 与 II 的乘积。

### 3.3.3.2 重构计算资源有限，全局寄存器无限的性能模型

当重构阵列的大小不能满足密码算法展开之后的流水线时，需要通过切换多套配置信息来完成整个计算。同时，利用全局寄存器文件缓存流水计算中的临时数据，直至所有计算完成之后将数据输出到阵列外部。

当全局寄存器文件的容量无限大时，可重构计算阵列大小是限制算法实现的主要制约因素。如图 3- 4 所示，当算法的操作节点数量为 7，可重构阵列的计算单元数量为 4 时，整个算法映射在重构阵列需要两套配置信息。其中，第一套配置信息包含 4 个计算节点，第二套配置信息包含 3 个计算节点。数据经过第一套配置信息后生成的 3 个数据将全部缓存到全局寄存器中，在配置信息切换完成之后，从全局寄存器中读取的数据在第二套配置信息中进一步处理。

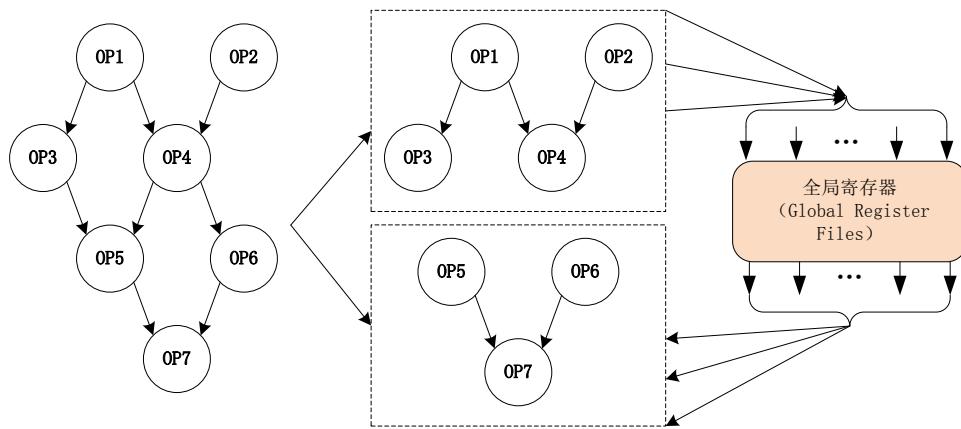


图 3-4 全局寄存器无限的计算示意图

为了能够直观的量化算法实现的过程，假设输入数据量为 6，每次配置切换周期  $T_{\text{stall}}$  仅需一个周期。如图 3-5 给出全局寄存器文件容量无限大时，算法在可重构计算阵列实现的示意图。在  $T=1$  时刻，数据进入重构阵列计算  $OP1$  和  $OP2$ 。在  $T=2$  时刻执行  $OP3$  和  $OP4$  操作，计算的结构输出到全局寄存器中。同时，新的数据继续在 PE11 和 PE12 单元上执行  $OP1$  和  $OP2$  操作。在  $T=7$  时刻，不再有新的数据进入计算阵列，第一套配置的计算进入排空阶段，计算结果继续写入全局寄存器。在  $T=8$  时刻，对重构阵列的配置信息进行更新。在  $T=9$  时刻，第二套配置开始进入载入阶段，从全局寄存器中加载访存数据执行  $OP5$  和  $OP6$  操作。在  $T=10$  时刻，第二套配置进入稳定状态， $OP7$  操作的结果输出到阵列外部。在  $T=15$  时刻，完成第二套配置的所有计算。

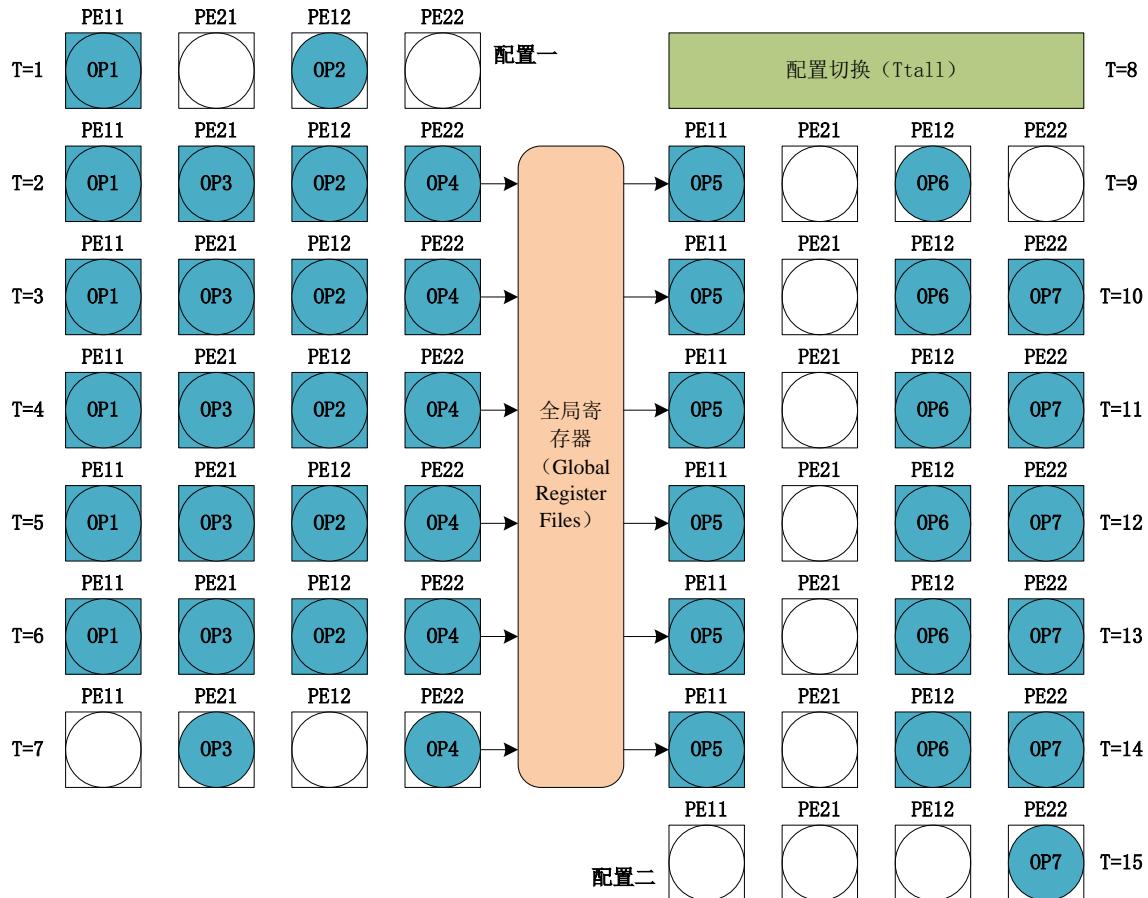


图 3-5 重构计算资源有限，全局寄存器无限的性能分析

考虑重构阵列的行数对性能的影响，此时整个密码算法在可重构处理器上实现所需的计算周期

的表达式如 (3.7) 所示:

$$\begin{aligned} Cycle_{infinite} = & \left\{ \frac{SC}{RCC} \times [RCC \times II + (N - 1) \times II + T_{stall}] \right\} \\ & + [mod(SC, RCC) \times II + (N - 1) \times II + T_{stall}] \end{aligned} \quad (3.7)$$

全局寄存器无限大时执行周期 ( $Cycle_{infinite}$ ) 的表达式分为两个部分。前半部分是算法的轮函数完全利用阵列资源的性能表达式，其中  $SC/RCC$  是完整利用阵列资源的配置信息套数；后半部分是算法的轮函数使用部分阵列资源的性能表达式。从式中可以看出，当  $SC$  小于  $RCC$  时，式 (3.7) 退化为式 (3.8)。

$$Cycle_{infinite} = [SC \times II + (N - 1) \times II + T_{stall}], \text{when } (SC \leq SCC) \quad (3.8)$$

### 3.3.3.3 重构计算资源有限，全局寄存器有限的性能模型

当重构阵列的计算资源不能满足密码算法展开的硬件需求，同时全局寄存器文件的容量大小仅能满足有限临时数据的缓存时，需要进行更多的配置信息切换，进一步加剧了对可重构处理器的性能制约。

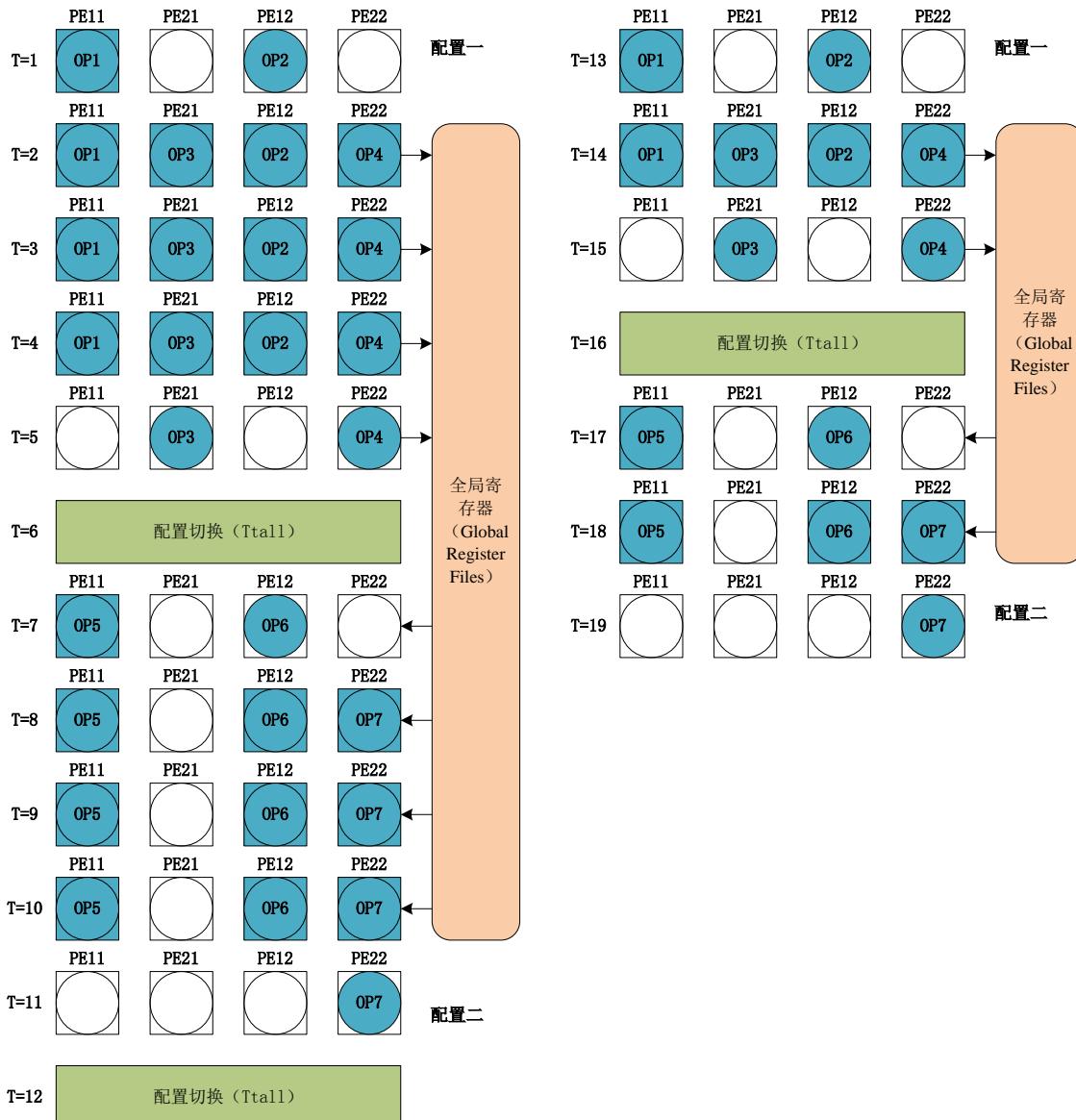


图 3-6 重构计算资源有限，全局寄存器有限的性能分析

图 3-6 给出了全局寄存器有限时，算法在可重构阵列实现的示意图。假设输入数据量  $N$  为 6，循环迭代  $II=1$ ，配置信息的套数为 2，配置切换的时间  $T_{stall}$  为 1，全局寄存器文件的大小为 4。在  $T=1$  时刻，数据从阵列外部直接进入重构阵列计算 OP1 和 OP2，第一套配置开始载入。在  $T=2$  时刻第一套配置进入稳定阶段，新的数据在 PE11 和 PE12 单元上执行 OP1 和 OP2 操作。同时，PE21 和 PE22 单元执行 OP3 和 OP4 操作，并将计算结果输出到全局寄存器中。由于全局寄存器中仅能容纳 4 组明文计算所生成的临时数据，因而在  $T=5$  时刻，第一套配置进入排空阶段。在  $T=6$  时刻进行配置切换之后， $T=7$  时刻第二套配置进入载入阶段，从全局寄存器中加载访存数据执行 OP5 和 OP6 操作。在  $T=10$  时刻，阵列将全局寄存器中最后一个数据读出，并在  $T=11$  时刻进入排空阶段，所有 OP7 操作的结果输出到阵列外部，完成第二套配置的所有计算。在  $T=12$  时刻再次进行配置切换，重新加载第一套配置。从  $T=13$  时刻开始，对剩余明文数据进行计算。由于仅剩余 2 组明文，因而在  $T=15$  时刻即可完成对第一套配置的计算，全部临时数据都寄存在全局寄存器文件中。在  $T=16$  时刻，切换到第二套配置。从  $T=17$  时刻，将全局寄存器中的临时数据取出进行处理，并在  $T=19$  时刻完成所有计算。

通过对图 3-5 和图 3-6 的对比可知，由于有限全局寄存器文件的限制，可重构密码处理器在计算过程中分多次对输入数据进行处理。这样不仅增加了配置切换的周期开销，而且每次配置切换后循环迭代的加载和排空阶段都会增加周期开销，使得整个计算周期从 15 增加到 19，周期开销增加 4 个周期。

全面考虑重构阵列的行数以及全局寄存器文件的大小对性能的影响，整个密码算法在可重构处理器上实现所需的计算周期的表达式如 (3.9) 所示，

$$\begin{aligned} Cycle_{finite} = & \frac{N}{GRC} \\ & \times \left\{ \left\{ \frac{SC}{RCC} \times [RCC \times II + (\min(N, GRC) - 1) \times II + T_{stall}] \right\} \right. \\ & \left. + [mod(SC, RCC) \times II + (\min(N, GRC) - 1) \times II + T_{stall}] \right\} \\ & + \left\{ \frac{SC}{RCC} \times [RCC \times II + (mod(N, GRC) - 1) \times II + T_{stall}] \right\} \\ & + [mod(SC, RCC) \times II + (mod(N, GRC) - 1) \times II + T_{stall}] \end{aligned} \quad (3.9)$$

全局寄存器有限大时执行周期 ( $Cycle_{infinite}$ ) 的表达式分为四个部分。前两个部分组成分组明文完全填满全局寄存器文件的性能表达式，其中  $N/GRC$  是分组明文数量除以全局寄存器行数的整数部分；后两个部分是分组明文数量小于全局寄存器文件大小的性能表达式。前后两个部分又分别组成利用有限阵列计算资源实现完整算法轮函数的功能。

从公式 (3.9) 可以看出，当全局寄存器的数量 (GRC) 无限大的时候， $\min(N, GRC) \approx N$ ， $N/GRC \approx 0$ ，同时， $mod(N, GRC) \approx N$ 。此时，整个性能公式退化为全局寄存器文件无限大情况下的式 (3.10)。

$$\begin{aligned} Cycle_{finite} = & \left\{ \frac{SC}{RCC} \times [RCC \times II + (N - 1) \times II + T_{stall}] \right\} \\ & + [mod(SC, RCC) \times II + (N - 1) \times II + T_{stall}], \text{when } (GRC \gg N) \end{aligned} \quad (3.10)$$

通过以上全局寄存器文件性能模型对可重构处理器的性能影响表达式可知，可重构处理器的性能受到多个关键因素的影响：算法特征、架构参数和调度方法。因此，本文在研究全局寄存器文件的设计方法时，同样需考虑到不同分组算法和架构参数的约束，才能获得最佳性能面积比的全局寄存器文件设计。

### 3.4 基于可重构阵列结构与映射场景的全局寄存器文件性能评估

在可重构计算软件流水中，全局寄存器文件对计算性能的影响与可重构阵列结构特征与映射场景密切相关。本节从可重构阵列的规模约束、配置信息切换效率以及全局寄存器文件的深度和宽度等因素出发，评估全局寄存器文件对可重构处理器的性能影响。

#### 3.4.1 面向有限重构阵列规模与可变全局寄存器文件深度的性能评估

通过全局寄存器文件的性能模型分析可知，不同轮数的算法采用不同的阵列大小实现时，密码算法的计算性能存在很大的差异。在第 2 章 2 节的分析中，分组密码算法的执行轮数主要集中在小于 64 轮的范围内。

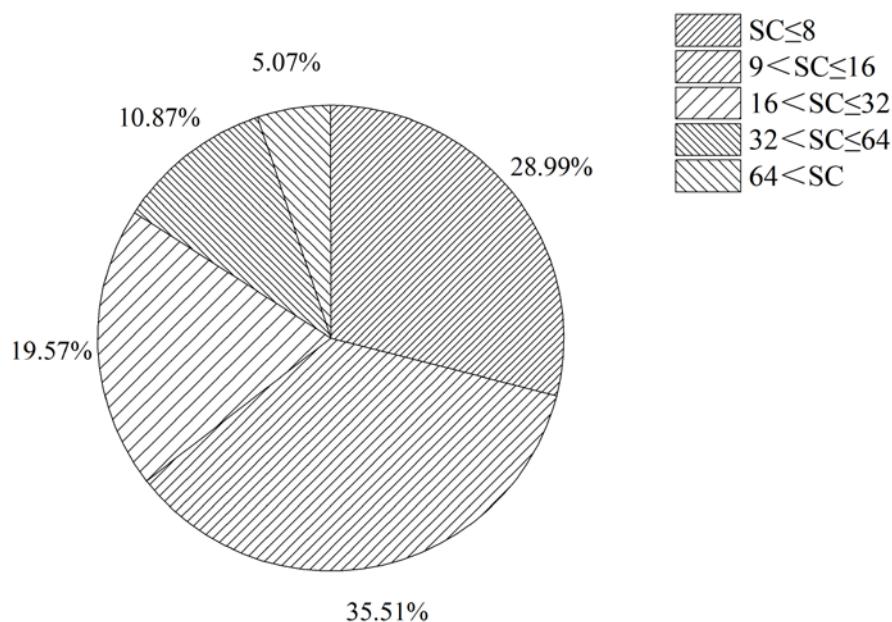


图 3-7 分组密码算法执行轮数的区间分布情况

图 3-7 给出分组密码算法中，执行轮数在不同区间的分布情况。从图中可知，以 8、16、32 和 64 为分隔点，算法执行轮数被切割成 5 个区域。算法执行轮数小于等于 32 轮的算法占总数的 84.06%，算法执行轮数小于等于 16 轮的算法占总数的 64.49%。特别注意到，算法执行轮数分别为 8、16 和 32 轮的分组密码分别占 23.86%、23.86% 和 11.36%。因此，本文在研究全局寄存器文件的深度对可重构处理器性能的影响时，着重考察算法轮数小于 32 轮的情况。

考察不同重构阵列行数的条件下，全局寄存器文件深度对分组密码算法计算性能的差异化影响。针对阵列行数分别为 4、8、12、16、24 和 32 的情况，假设每轮计算仅需 1 个周期，迭代间隔  $H$  为 1 个周期，配置切换  $T_{stall}$  为 1 个周期。同时，为了排除全局寄存器文件其他设计参数对性能的影响，假设全局寄存器文件的数据位宽、访问并发度和访存时延均能满足算法实现的最大需求。

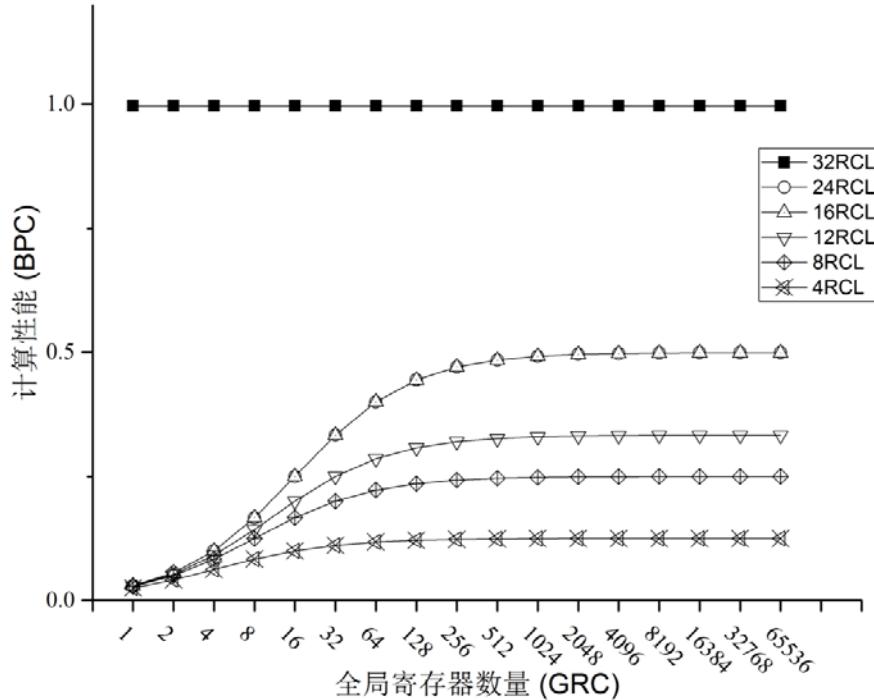


图 3-8 32 轮密码算法在不同阵列大小的性能比较

从图 3-8 中可以看出，仅在阵列大小与密码算法的轮数相同（32）时，密码算法的轮数拥有最高的计算性能（1BPC），计算数据可以不经过全局寄存器就输出阵列。当阵列行数（4、8、12、16、24）小于密码算法的轮数（32）时，算法计算性能随着全局寄存器的容量增加而不断提高，最终趋于稳定值。

当密码算法的轮数（32）大于阵列行数（4、8、12、16、24）时，算法计算性能随着全局寄存器的容量增加而不断提高，最终趋于稳定值。同时，密码算法的最高计算性能与配置信息的套数成反比，最高获得  $((1/SC/RCL) BPC)$  的计算性能。进一步分析，阵列大小在 24 轮时获得的性能与阵列大小 16 轮是一致的。这是由于 16 和 24 轮阵列都需要两套配置来完成全部的密码计算流水，在每次配置切换的代价都是一个周期的条件下，性能仅与配置的套数有关。

通过分析以上不同重构阵列行数的条件下，全局寄存器文件深度对分组密码算法计算性能的影响可知：

- (1) 对于密码算法轮数小于或者等于阵列支持轮数的条件，整个计算不需要使用全局寄存器缓存数据，密码算法拥有最高的理论计算性能（1BPC）。
- (2) 当密码算法的轮数大于阵列能够支持的算法轮数时，密码算法的计算性能受限于重构阵列资源约束，最高理论计算性能为  $((1/SC/RCL) BPC)$ 。
- (3) 当密码算法的轮数大于阵列能够支持的算法轮数时，全局寄存器的深度严重影响可重构密码处理器的性能。全局寄存器文件的深度越深，能够缓存的临时数据越多，越能够有效减少配置切换的频率，减少循环加载和排空的性能开销。

### 3.4.2 面向有限重构阵列规模与可变全局寄存器文件宽度的性能评估

通过分析不同明文位宽的密码算法在全局寄存器文件的性能模型，可知寄存器文件的数据位宽对密码算法的计算性能存在很大的影响。分析密码算法分组位宽的分布情况可知，分组密码算法的

的分组位宽主要集中在小于 128 位的范围内。

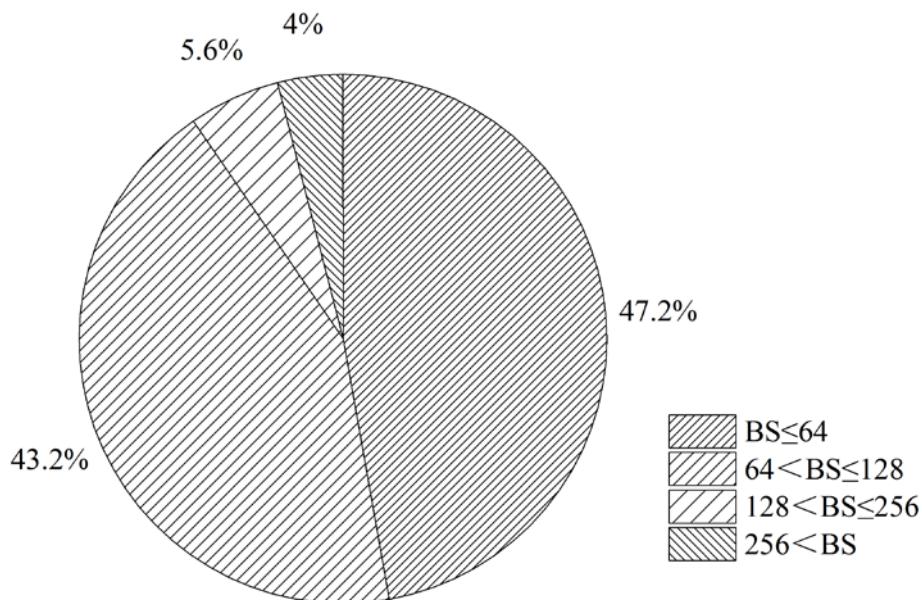


图 3-9 密码算法分组位宽的区间分布情况

图 3-9 给出分组密码算法中分组位宽在不同区间的分布情况。从图中可知，以 64、128 和 256 为分隔点，密码算法的分组位宽 BS（BlockSize）被切割成 4 个区域。分组位宽小于等于 64 比特的算法占总数的 47.20%，分组位宽大于 64 比特而小于等于 128 比特的算法占总数的 43.20%，其他算法的分组位宽仅占总数的 9.6%。特别注意到，超过 80.00% 的分组密码算法的分组位宽集中在 64bit 和 128 bit，分别占据所有算法分组位宽的百分比为 40.80% 和 39.20%。因而，本文在研究全局寄存器文件的宽度对性能的影响时，着重考察分组位宽小于等于 128 比特的情况。

考察不同重构阵列行数的条件下，全局寄存器文件宽度对分组密码算法计算性能的差异化影响。针对分组密码算法轮数为 32 轮，同时重构阵列行数分别为 4、8、12、16、24 和 32 的情况。假设每轮计算仅需 1 个周期，配置切换  $T_{stall}$  为 1 个周期，同时全局寄存器文件的数据深度、访问并发度和访存时延均能满足算法实现的最大需求。全局寄存器文件的数据位宽作为重构阵列计算过程中的资源依赖，直接影响到迭代间隔  $II$  的周期数。

从图 3-10 中可以看出，对于不同的重构阵列行数，分组密码算法计算性能随着全局寄存器的位宽增加而不断提高，最终趋于稳定值。密码算法在不同阵列大小实现时最高计算性能与阵列所需的配置信息套数成反比，最高获得  $((1/SC/RCL) BPC)$  的计算性能。随全局寄存器位宽的增加不断提高，计算性能呈指数变化增加。特别当全局寄存器位宽大于等于分组密码的分组位宽时，性能不再变化，达到最大值。

通过分析以上不同重构阵列行数的条件下，全局寄存器文件宽度对分组密码算法计算性能的影响可知：

- (1) 当全局寄存器位宽小于密码算法的分组位宽时，全局寄存器的位宽严重影响可重构密码处理器的性能。全局寄存器文件的位宽越大，单次能够加载的数据量越多，越能够有效减少对全局寄存器的访存次数，减少由于资源限制而导致的循环迭代的等待开销。
- (2) 当全局寄存器位宽大于或等于密码算法的分组位宽时，密码算法的计算性能达到最高理

论计算性能为 ((1/SC/RCL) BPC)。

- (3) 当全局寄存器位宽远远大于密码算法的分组位宽时, 密码算法的计算性能不随全局寄存器位宽的增加而变化, 保持在最高理论计算性能 ((1/SC/RCL) BPC)。

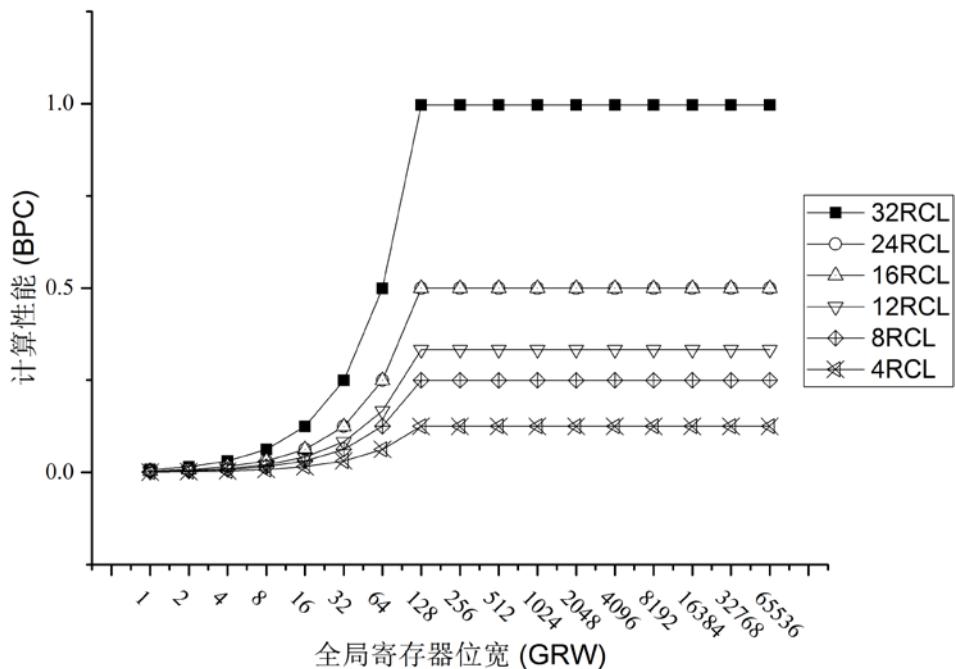


图 3-10 128 比特分组位宽的密码算法在不同阵列大小的性能比较

### 3.4.3 面向可变重构配置效率与可变全局寄存器文件深度的性能评估

在分析可重构阵列计算行资源对密码处理器性能的影响之后, 下面进一步分析配置切换开销对性能的影响。此处分析在配置切换开销约束下, 密码算法的轮数 (32) 大于可重构阵列行数 (8、12、16、24) 时, 算法计算性能随着全局寄存器深度增加的变换趋势。

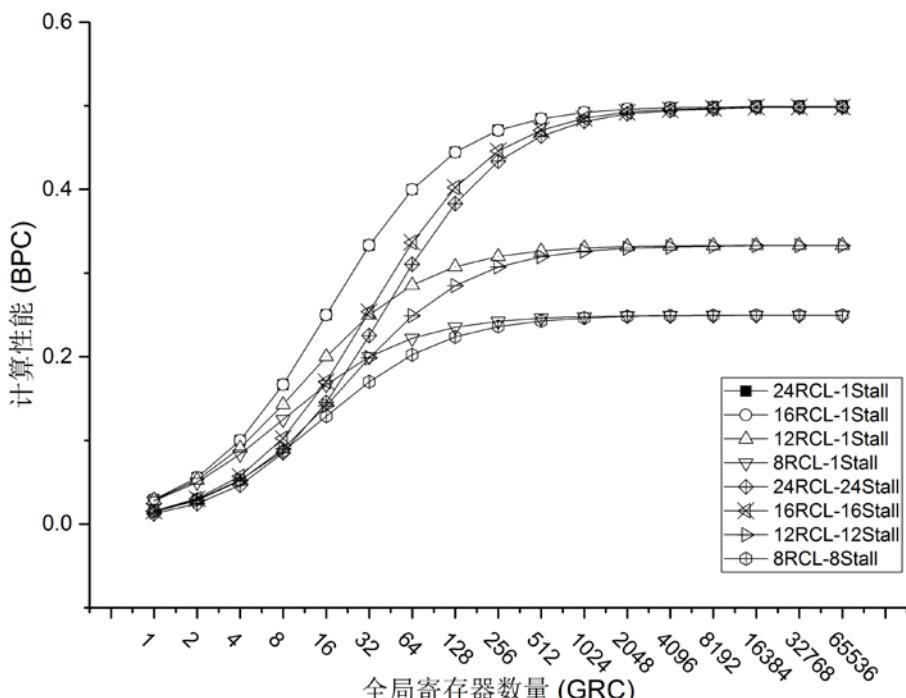


图 3-11 32 轮密码算法在不同配置信息切换约束下的性能比较

从图 3-11 可知，对于配置信息切换周期为 1 的情况，阵列大小在 24 轮和 16 轮时获得的性能是一致的，性能仅与配置的套数相关。当配置信息的切换周期与阵列大小的行数相同时，虽然随着寄存器文件容量的增加最终性能一致。但是，当全局寄存器容量较少时，16 轮阵列大小的计算性能略高于 24 轮阵列大小。这是由于同样两套配置完成全部的密码计算流水级，24 轮的阵列较 10 轮阵列每次配置更新过程需要更多的配置信息切换周期。对于阵列大小分别为 12 轮和 8 轮的情况，配置切换周期直接影响相同全局寄存器深度下的可重构处理器性能。

通过以上分析可知：

- (1) 当配置信息的切换周期增加时，不会影响密码算法达到最高理论计算性能 ((1/SC/RCL) BPC)。
- (2) 当配置信息的切换周期增加时，需要增加全局寄存器文件深度才能达到最高理论计算性能 ((1/SC/RCL) BPC)。
- (3) 当配置信息的切换周期增加时，将影响相同全局寄存器文件深度下密码算法的实现性能。配置切换周期越大，可重构处理器的计算性能越低。

通过以上对全局寄存器文件设计参数的影响分析可知，作为限制可重构密码处理器性能瓶颈的关键因数，全局寄存器文件的设计受限于目标算法集合和阵列架构的特征限制。为了能够获得满足密码算法性能需求的全局寄存器文件结构，需要同时考虑架构和算法特征的综合影响。如 3.3 节全局寄存器文件的性能模型描述，分组密码算法在不同重构阵列行数的密码处理器实现时受到如下影响。

- (1) 当密码算法轮数小于等于可重构阵列能够提高的计算资源时，整个计算过程不使用全局寄存器缓存数据，密码算法拥有最高的理论计算性能 (1BPC)。
- (2) 当密码算法的轮数大于重构阵列能够支持的算法轮数，寄存器文件能够提供所需的所有资源时，最高理论计算性能可以达到 ((1/SC/RCL) BPC)。
- (3) 当密码算法的轮数大于重构阵列能够支持的算法轮数时，寄存器文件的深度和位宽成为制约可重构密码处理器的关键因素。同时，随着深度和位宽增加而带来的资源优势，有效减少计算过程中的配置切换频率和寄存器访存次数，最终达到可重构处理器的最高理论计算性能。
- (4) 配置信息切换的开销差异，不会影响密码算法达到最高理论计算性能 ((1/SC/RCL) BPC)。但是影响相同全局寄存器文件深度下的算法性能，只有更大的全局寄存器文件深度才能达到最高理论计算性能 ((1/SC/RCL) BPC)。

为了能够满足目标算法集合的性能需求，全局寄存器文件的结构需要根据重构阵列的架构特征进行优化。首先，根据算法轮数 (SC) 和可重构阵列行数 (RCL) 确认最高理论计算性能是否能够满足目标算法的性能需求。然后，根据目标算法集合的分组位宽 (BS) 确认最优的全局寄存器文件位宽。最后，选择能够实现目标算法集合性能需求的最小全局寄存器文件深度。

## 3.5 一种基于面积效率优化的分布式全局寄存器文件结构的提出

### 3.5.1 性能约束下全局寄存器文件的面积分析

全局寄存器的数量越多，可重构密码处理器的性能越高。但是，随着数量的增加，全局寄存器所需的面积资源也呈指数级增加。如图所示 32 轮密码算法在阵列大小为 16 的条件下，密码算法在可重构阵列实现性能和面积随全局寄存器文件变化趋势。横坐标是全局寄存器文件的个数 GRC ( $2^N$  变化)，主纵坐标是计算性能（线性变化），次纵坐标是全局寄存器文件在 TSMC40 工艺条件下的面积 ( $2^N$  变化)。从图 3-12 中可知，当全局寄存器文件的容量为 4096 行时，可重构处理器的性能接近达到理论最大性能 0.5BPC。同时，可以看到全局寄存器文件的面积随数量的增加而线性变化。特别注意到，面积变化曲线在全局寄存器文件数量 1 到 2 的过程中出现一个变化速率下降的折点。这是由于 GRC 的数量为 1 时，整个全局寄存器文件中不需要译码逻辑；而当 GRC 的数量大于或者等于 2 时，全局寄存器文件的面积包含了寄存器体和译码逻辑两个部分的面积。

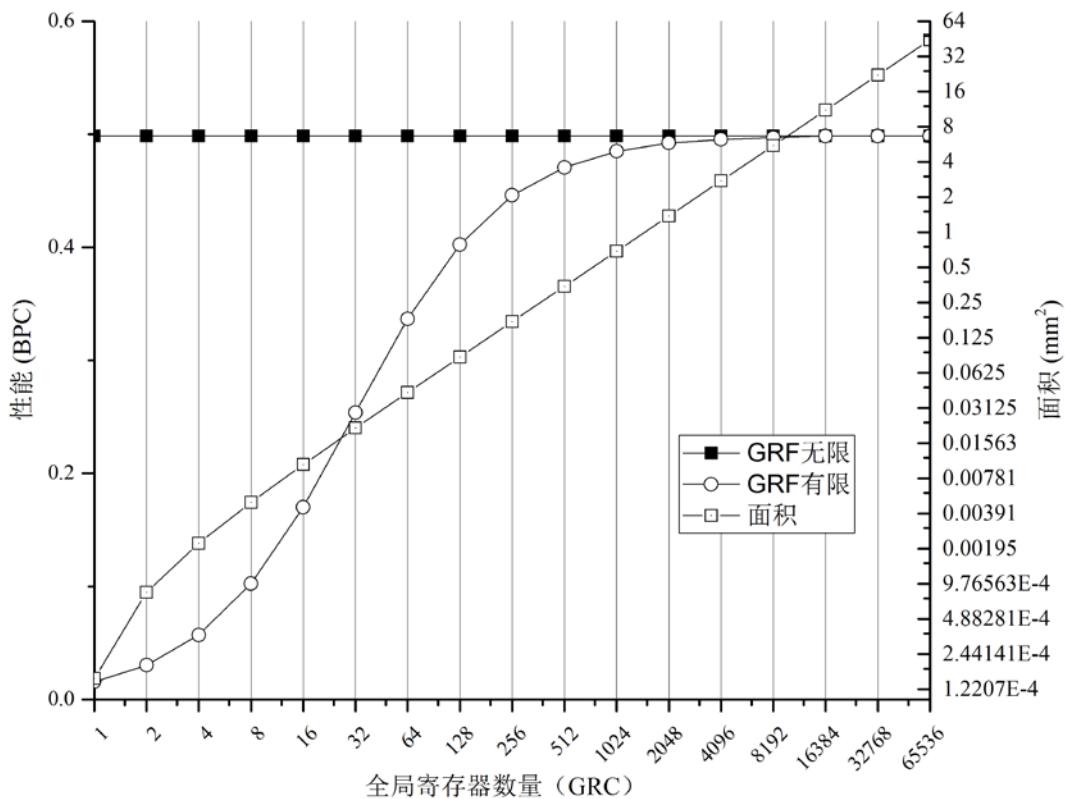


图 3-12 性能和面积随全局寄存器的变化趋势

图 3-13 给出随着全局寄存器数量增加时，可重构处理器的性能变化率。从图中可知，对于可重构阵列行数 RCL=16，算法轮函数 SC=32 的情况，计算性能随数量增加呈指数变化，在 GRC 的数量从 16 变化到 32 的过程中，拥有最高的性能变化率。此时，密码处理的计算性能为 0.25BPC。之后随着 GRC 数量增加，可重构密码处理器的性能持续增加，但是性能变化率不断减少，直至变化率减低为 0。此时，可重构密码处理器获得最高的理论计算性能 0.5BPC。

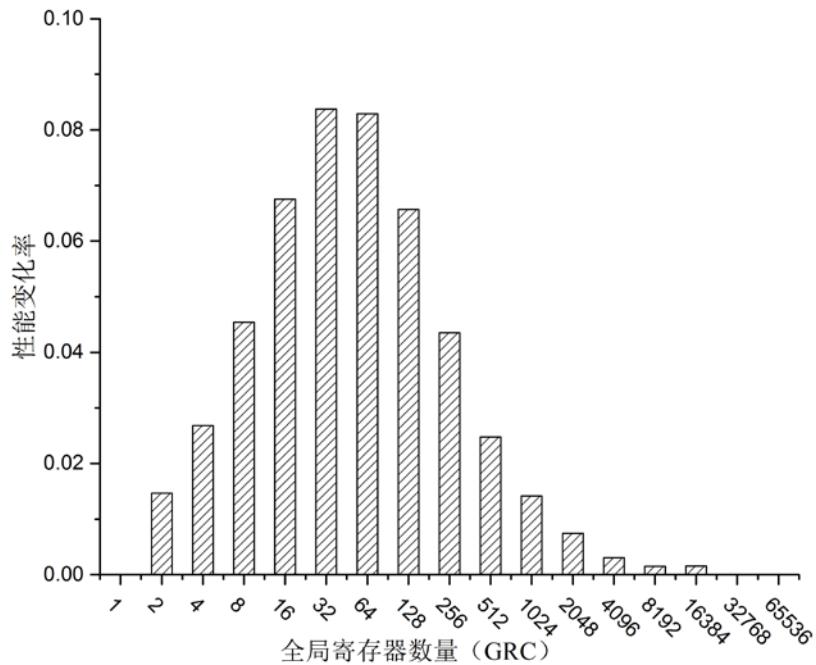


图 3-13 全局寄存器数量增加的性能变化率

通过以上对全局寄存器文件面积和性能的影响分析可知，通过增加全局寄存器文件可以有效的增加处理器的计算性能。但是，随着全局寄存器文件容量的增加，计算性能将达到饱和，容量的增加仅能带来处理器面积的增大。因此，考虑全局寄存器文件的优化设计时，不仅要考虑全局寄存器文件数量变化对计算性能的影响，还要考虑可重构阵列轮数变化对性能和面积的影响。

本文参考文献[33]给出的 45nm 工艺下每一全行（计算行和互联行）重构阵列的平均面积，然后考虑可重构阵列轮数对整个处理器性能面积比的影响，实验结果如下图所示。从图 3-14 中可知，当全局寄存器文件的容量较小时，RCA 的面积对整体面积的影响更大，当全局寄存器文件的数量大于 256 时，全局寄存器文件的面积开销在整体面积中占主导地位。

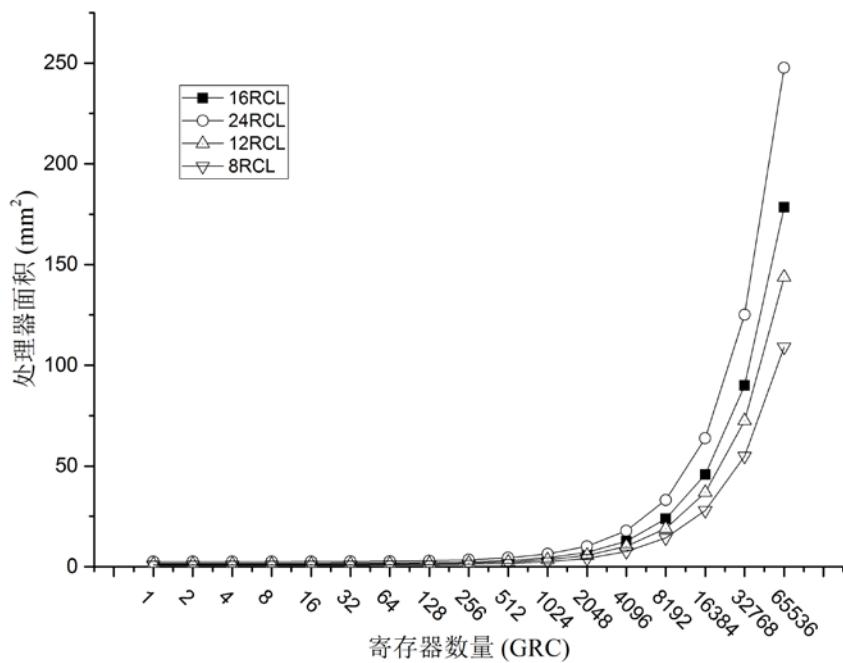


图 3-14 不同阵列大小条件下处理器面积随全局寄存器数量变化趋势

通过对比不同阵列大小的面积曲线可知，随着全局寄存器的数量增加所需的面积资源也呈指数级增加。可重构密码处理器的性能不断提高，但是性能变化率不断减少，即寄存器面积增加带来的性能收益越来越小，容量的增加仅能带来处理器面积的增大。

### 3.5.2 分布式全局寄存器文件结构的优化设计

以上分析全局寄存器文件对可重构密码处理器的性能和面积的影响时，假设不同算法的轮函数计算均能在在一个周期内计算完成。然而，实际的密码算法通常需要多个周期才能完成一轮函数的计算，而且不同算法轮函数的周期也不相同。为了满足数据在算法轮函数之间的依赖关系，密码算法在可重构阵列上进行多套配置的划分时，需要在算法轮函数之间进行切割。

本文采用与参考文献[33]相同的可重构阵列和密码算法集合，可重构阵列可以容纳最大位宽128bit，最多40行的计算操作。本文实验采用的密码算法特征如表3-3所示，其中算法轮数是目标算法集合实现的轮函数迭代次数。以AES算法为例，当密钥长度为128bit时，完整实现输入明文的10轮函数迭代计算需要20个周期。周期数是密码算法在可重构阵列上完成计算所需的操作数。密码算法的分组位宽小于128bit时，可以利用分组密码在计数器模式（CTR）分组数据无关的特性进行并发计算。

表 3-3 分组密码算法特征

算法	算法轮数 (Round)	分组位宽 (BlockSize)	周期 (Cycle)
AES	10	128	20
Blowfish	16	64	48
Camellia	18	128	80
CAST-128	16	64	80
DES	16	64	48
GOST	32	64	96
KASUMI	6	64	64
RC5	12	64	48
SEED	16	128	160
Twofish	16	128	80

同时参考文献[33]给出的重构阵列每一全行(Full Column)在45nm工艺下的平均面积 $0.2115\text{mm}^2$ ，给出计算并行度为一时的性能和性能面积图表。如图3-15性能面积和性能随全局寄存器文件的变化趋势所示，计算阵列行数RCL=40，横坐标是全局寄存器文件的个数GRC ( $2^N$ 变化)，主坐标轴是分组算法的性能变化，副坐标轴是算法的性能面积变化。

从图中可知，除了AES算法的计算性能始终保持在1BPC之外，其他算法的计算性能随着GRC的增加而变大，最终趋于饱和。通过对比不同算法周期在阵列实现的性能曲线可知，除了AES算法的计算性能始终保持在1BPC之外，其他算法的计算性能均小于1BPC。这是因为除了AES算法能够在阵列上全展开仅需一套配置信息之外，其他算法需要多套配置切换才能完成。

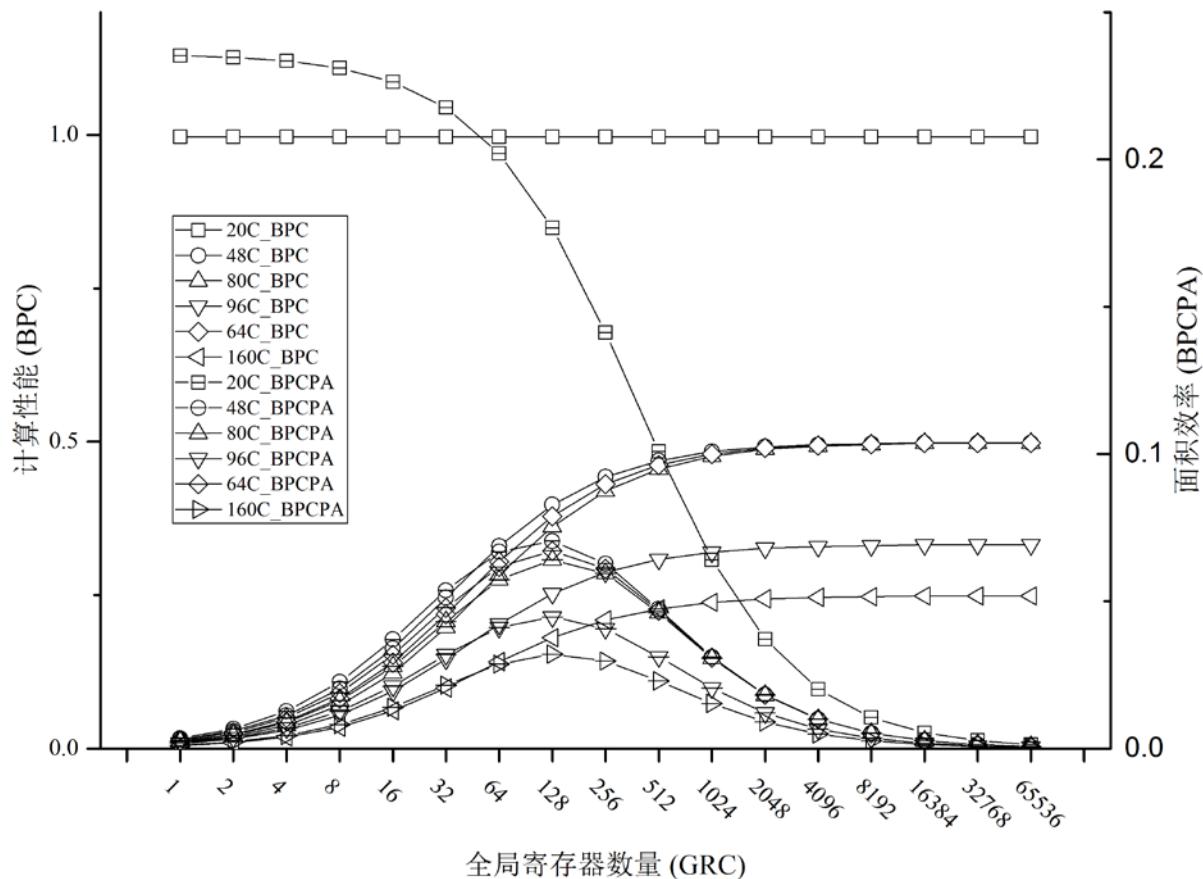


图 3-15 分组密码性能和面积效率随全局寄存器变化趋势

算法的性能面积比随着全局寄存器文件的数量的增加而逐渐增大，在达到最大值之后随着全局寄存器文件数量的进一步增加而逐渐减小。除了AES算法随着全局寄存器文件的增加而逐渐减少之外，其他算法都存在性能面积的极大值点。通过对不同算法周期在阵列实现的在性能面积比曲线可知，当全局寄存器的GRC选择128时，可以获得最高的性能面积比。此时，AES算法的面积效率为0.18BPCPA，其他算法的面积效率为0.03~0.07 BPCPA。

在确定全局寄存器GRC选择128时，容量为128个128bit ( $7 \times 128\text{bit}$ ) 共16Kbit之后，进一步考虑全局寄存器文件的架构设计对面积的影响。基于密码算法轮函数之间的数据相互独立，下一轮函数不依赖上一轮计算的输出结果，因而全局寄存器文件在可以采用多个独立的寄存器体来取代单个寄存器体的物理结构设计。

全局寄存器文件的架构约束条件主要包括以下几项：

- 寄存器总容量 (RegisterSize)
- 并发度 (Parallel)
- 寄存器行数 (RegisterFileColumns)
- 寄存器每行并发读取数 (RegisterParallelReadPerRound)
- 寄存器每行并发写入数 (RegisterParallelWritePerRound)

在满足以上约束条件的情况下，按照寄存器数量 (RegisterNumber) 递减给出如图3-16所示寄存器文件中寄存器体和译码逻辑面积随寄存器数量和读写端口数的变化趋势。

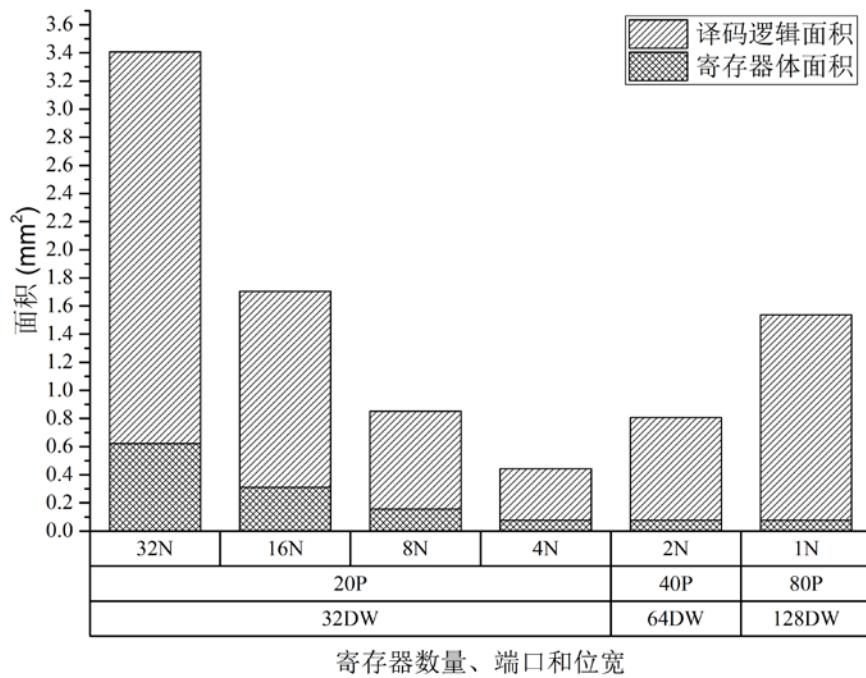


图 3-16 寄存器体和译码逻辑面积随寄存器数量和读写端口数变化趋势

从图3-16中可以看出，在满足约束条件的最小面积情况下，寄存器文件的译码逻辑面积成为影响的关键因素。当寄存器个数越少，译码逻辑所需的寻址控制逻辑也就越大，因而可以看到当增加寄存器体个数时，可以有效的减少译码逻辑的面积。继续增加寄存器体的数量时，为了满足平均每轮寄存器的位宽需求，可以减少数据位宽，这样的设计并不会影响面积的变化。因而，本文中最优解集合中选取设计参数如表3-4下：

表 3-4 全局寄存器设计参数

名称	数量
数据地址位宽	7
数据位宽	32
寄存器体数量	4
读端口数量	20
写端口数量	2

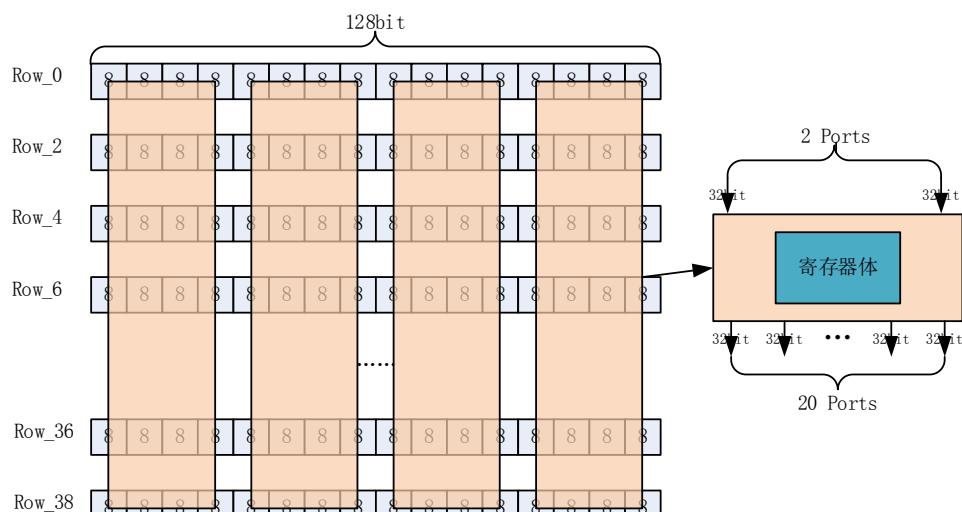


图 3-17 全局寄存器架构示意图

如图3-17所示，每个分组的全局寄存器文件为重构阵列设计了多个独立的读端口，同时提供两个独立的写通道，多个写端口可以通过选择仲裁对全局寄存器文件进行写操作。40行可重构阵列按照每2行设计10个读端口设计，每组寄存器体包含20个读端口，2个写端口。

图3-18给出了分布式全局寄存器文件架构的设计细节，整个全局寄存器文件包含4个 $7 \times 32\text{bit}$ 的寄存器体。每组寄存器的读端口分别连接每行中32bit的重构计算单元，写端口分别连接到第20行和第40行。利用多个全局寄存器端口的并发优势，可以满足不同算法流水级数，不同配置套数的密码算法对寄存器的存取需求。

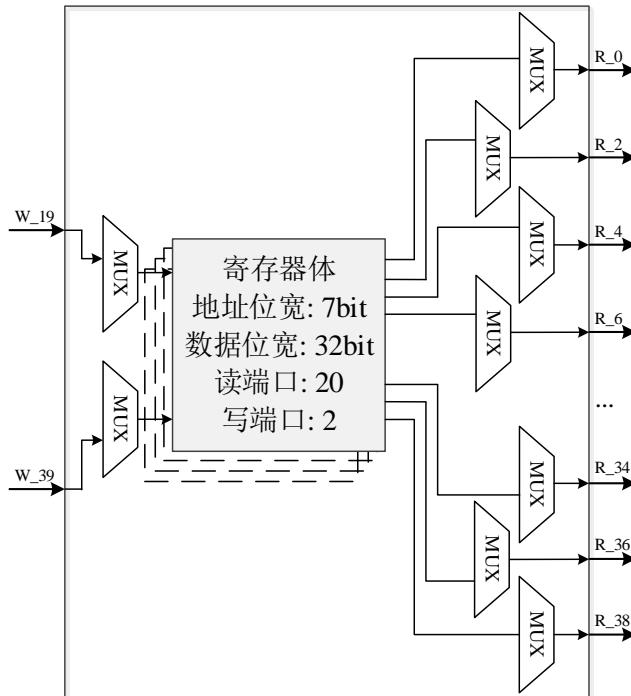


图 3-18 分布式全局寄存器文件设计细节

### 3.6 实验与分析

为了能够针对全局寄存器文件的优化设计进行有效的实验分析，本文采用与可重构密码处理器 Cryptoraptor 相同的架构基础。Cryptoraptor 是目前实现算法最多，计算性能最高的可重构密码处理器。本章实验在可重构阵列规模、算子操作行为、算法执行周期等方面均保持一致，仅改变全局寄存器文件的实现方案。

如表3-5所示，Cryptoraptor 对比设计方案中，寄存器文件容量为8Kbit，仅含有1个寄存器体有80个读端口，2个写端口，每个寄存器体中有256个32bit寄存器（ $8 \times 32\text{bit}$ ）。ADRES 对比设计方案中，采用减少访存队列访存冲突和硬件开销的全局寄存器文件重排序机制。寄存器文件的设计参数参考文献[69]，采用4个单端口寄存器，每个寄存器体为32bit。数据采用Interleave方式存储在多块寄存器中，重构单元通过访存仲裁实现对数据多端口竞争的重排序访问。

本文采用基于可重构阵列结构和映射场景的全局寄存器文件性能评估获得的设计参数，以及分组全互联的分布式寄存器文件结构。全局寄存器文件容量为16Kbit，采用4个寄存器体，每个寄存器体中有128个32bit寄存器（ $7 \times 32\text{bit}$ ），每个寄存器体包含20个读端口，2个写端口。特别注意到，由于 ADRES 架构采用设计模板参数化定义全局寄存器文件容量<sup>[48]</sup>，为了突出 ADRES 架构的全局寄存器文

件重排序机制和端口设计参数差异对性能和面积的影响，此处采用与本文相同的寄存器文件容量16Kbit。

本文将以上所有对比全局寄存器文件架构在TSMC 40nm CMOS工艺下实现，并利用综合工具DC（Design Compiler）获得面积信息，最后将密码算法在可重构阵列上映射获得计算性能。表3-5给出不同设计方案中全局寄存器文件的设计参数。

表 3-5 不同全局寄存器设计方案的设计参数对比

对比架构	寄存器体个数	寄存器体地址位宽 (bit)	寄存器容量 (Kbit)	寄存器体读端口数	寄存器体写端口数
Cryptoraptor	1	8	8	20	2
ADRES	4	7	16	1	1
本文设计	4	7	16	20	2

图3-19给出不同设计方案的全局寄存器文件面积比较结果。从图中可以看出，本文提出的全局寄存器文件架构相比Cryptoraptor设计容量增加了1倍，但是面积资源却减少了41.92%。这是由于本文提出的全局寄存器文件，虽然寄存器体面积增加1倍，但是译码逻辑相比Cryptoraptor减少50.20%。与ADRES设计方法相比，在相同的容量限制下，面积资源增加131.04%。进一步分析可知，本文提出的全局寄存器文件的译码逻辑面积开销增加233.33%。这是由于ADRES架构采用仲裁访问的单端口寄存器设计，所需的译码逻辑面积开销更小，但是，这也大大限制了密码数据的访问并发度。

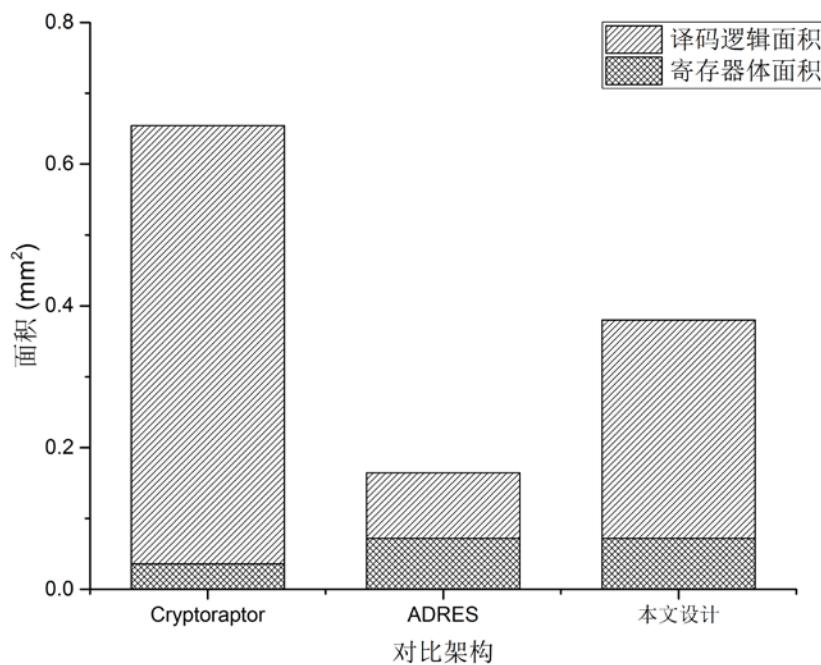


图 3-19 不同设计方案的全局寄存器面积分布

表 3-6 给出了不同全局寄存器文件架构下的算法性能对比和面积效率对比。本文实验采用包括AES, Blowfish 等主流分组算法在内的共 10 种算法。在算法性能 (BPC) 一栏中采用单位周期内处理的明文数量 (Block/Cycle) 作为比较参数，面积效率 (BPCPA) 是单位面积内的算法性能，单位是 (Block/Cycle/mm<sup>2</sup>)。

与 Cryptoraptor 设计相比，采用更大容量的分布式全局寄存器文件设计参数之后，算法平均性能

从 0.35BPC 提高到 0.41BPC，平均提高 17.24%。其中对于算法 AES 由于仅有一套配置，其性能仅受到少量加载和排空阶段的影响，可以接近最高理论性能。对于算法计算周期大于阵列轮数的密码算法，由于寄存器容量的增加而减少了配置切换的次数使得算法性能得到提高。从面积效率对比可以看到，结合优化的结构设计参数，本文实现的全局寄存器在相同的可重构阵列和应用算法条件下可以获得更高的面积效率。与 Cryptorapter 设计相比，全局寄存器文件的平均面积效率从 0.53BPCPA 提高到 1.07BPCPA，平均提高 101.86%。

与 ADRES 设计相比，本文设计的全局寄存器文件利用寄存器的多端口设计能够提供更高的数据访问并发度，减少由于寄存器文件位宽和密码算法分组位宽不匹配而导致的访存次数增加。针对不同分组位宽的密码算法，计算性能从 0.03~0.24BPC 提高到 0.18~1.00BPC。算法算法平均性能从 0.12BPC 提高到 0.41BPC，平均提高 230.67%。对比面积效率可知，ADRES 设计受益于面积资源的优势，对于 64bit 分组位宽密码算法具有相近的面积效率，但是对于 128bit 分组位宽密码算法其面积效率大大降低。与 ADRES 设计相比，全局寄存器文件的平均面积效率从 0.75BPCPA 提高到 1.07BPCPA，平均提高 43.12%。

表 3-6 不同全局寄存器设计方案的性能和面积效率对比

算法	算法参数		算法性能(Block/Cycle)			面积效率((Block/Cycles)/mm <sup>2</sup> )		
	算法轮数	分组位宽	Cryptorapter	ADRES	本文设计	Cryptorapter	ADRES	本文设计
AES	10	128	1.00	0.24	1.00	1.52	1.44	2.62
Blowfish	16	64	0.33	0.16	0.40	0.50	0.96	1.05
Camellia	18	128	0.28	0.07	0.36	0.43	0.45	0.95
CAST128	16	64	0.28	0.13	0.36	0.43	0.76	0.95
DES	16	64	0.33	0.16	0.40	0.50	0.96	1.05
GOST	32	64	0.20	0.08	0.25	0.31	0.50	0.66
KASUMI	6	64	0.30	0.14	0.38	0.47	0.85	1.00
RC5	12	64	0.33	0.16	0.40	0.50	0.96	1.05
SEED	16	128	0.14	0.03	0.18	0.22	0.19	0.48
Twofish	16	128	0.28	0.07	0.36	0.43	0.45	0.95
Average	NA	NA	0.35	0.12	0.41	0.53	0.75	1.07

### 3.7 本章小结

通过分析研究现状中全局寄存器对密码算法性能和面积的影响，本章对架构约束下全局寄存器文件的设计参数进行空间探索，给出全局寄存器文件对性能影响的表达式，分析不同可重构阵列行数和配置信息切换条件下，全局寄存器文件深度和宽度变化对性能的影响。给出不同密码应用需求和目标算法集合条件下，全局寄存器文件的最优设计参数。采用分组全互联的分布式全局寄存器文件设计，本文实现目标密码算法集合的性能较 Cryptorapter 设计方法平均提高 17.24%，较 ADRES 设计方法平均提高 230.67%。密码算法集合在有限阵列容量限制下，本文实现的面积效率较 Cryptorapter 设计方法平均提高 24.70%，较 ADRES 设计方法平均提高 43.12%。

## 第四章 多端口统一结构的跨域寄存器文件研究

### 4.1 前言

密码学中置换盒 S-box，即“substitution box”，是对称密钥算法执行置换计算的一个基本组件。S-box 的功能是实现数据的非线性置换，用于模糊密钥和密文之间的关系——香农的混淆理论。通常 S-box 接受特定数量的输入比特 m，并将其转换为特定数量的输出比特 n，其中 n 不一定等于 m。本文对现有分组密码算法分析表明，超过 56.19% 的算法都包含 S-box 替换操作。作为密码算法中唯一的非线性模块，S-box 的设计是高效实现密码算法的关键。

分组密码算法在实现替换操作过程中采用了多种 S-box 结构。不同密码算法的 S-box 个数从 1 到 8 个不等，而且其输入位数从 3bit 到 13bit 不等，输出位数从 2bit 到 32bit 不等。具体分析密码算法在每一轮的算法特征可知，AES 算法在每轮计算中查 16 个一样的表；DES、SM4、Blowfish、Camelila、CAST128、GOST、KASUMI 需要查 4 到 8 个不同的表；SEED 算法每轮查 2 次，每次查找 4 个不同的表；RC4 每轮查找的表则在计算过程中不断变化。通过以上分析，在算法不同的查找表个数、查找并发度等多样性需求条件下，兼容所有算法 S-box 特征是架构设计必须满足的基础。

为了能够满足不同算法差异化的需求，S-box 设计单元需要满足目标算法集合对并发访存数量和存储容量的最小需求，可重构 S-box 架构成为密码处理器的设计难点。为每个可重构单元设计对应的 S-box 单元可以满足各种计算的需求，但是这样的设计会导致 S-box 单元占用的面积资源大大增加；减少寄存器文件的端口数和数据位宽可以有效的减少面积开销，但是由于资源复用需要等待多个周期来完成替换操作，严重制约可重构密码算法的并行流水线设计。已有研究表明<sup>[33, 73]</sup>，实现替换操作的寄存文件面积占参考文献中处理器面积的 20.50% 到 24.57%。同时，寄存器的端口数量和数据位宽设计成为限制可重构处理器并发存取的关键。因此，通过对替换操作的寄存器文件的研究，对系统面积和性能优化都存在重大意义。

### 4.2 研究现状分析

面向密码算法的可重构处理器追求高的性能面积比，而不同算法中 S-box 的差异使得局部寄存器文件成为架构实现的重点和难点。S-box 的实现有两种方法，一是利用复合域运算<sup>[74, 75]</sup>的原理，用逻辑电路实现 S-box，例如文献[76, 77]分别在  $GF(2^4)^2$  和  $GF((2^2)^2)^2$  上实现了 AES 的 S-box；另一种方法是采用查找表（Look Up Table，LUT）的方式，将密码算法的中 S-box 表的内容存储到相应的存储器中，算法执行时对该存储器进行查表得到 S-box 操作的结果，如 Cryptoraptor 处理器<sup>[33]</sup>，文献[73]等。采用逻辑电路实现 S-box 可以获得 ASIC 定制实现方法的优势：很高的运算性能和极少的资源消耗。但是，由于密码算法 S-box 计算的差异性，需要实现不同逻辑电路来满足差异化的需求。而且，不是所有的密码算法都可以用复合域运算实现，如 DES、Blowfish 等。LUT 的实现方法具有很好的灵活性，可以满足密码算法中不同类型 S-box 的计算需求，相比复合域设计方法更适合可重构的灵活实现。然后，为了满足不同密码算法中不同形式 S-box 的高性能实现，LUT 需要大的存储容量和复杂的设计逻辑，在消耗大量的面积资源的同时也成为影响系统性能的关键路径。因此，为

了满足高性能的可重构密码处理需求，需要对实现 S-box 的寄存器 LUT 进行优化设计。

S-box 是影响密码算法高性能实现的关键。作为架构实现过程中的关键路径，Cryptoraptor 架构在寄存器 LUT 电路之前和之后加入异或（XOR）逻辑，优化算法执行的流水线过程。ProDFA<sup>[36]</sup>架构利用多个 SRAM Block 实现可配置的替换单元（CSU）用于实现 S-box 算法，SRAM 块将输入数据作为地址进行寻址操作，利用存储器读操作在一个周期内完成 S-box 替换操作。文献[78]分别采用两个 8 入 8 出的寄存器 LUT 实现编码和解码算法中 S-box，支持最大 128bit 位宽输出 S-box 算法。文献[78]中设计了结构简单的 LUT，采用两个固化 ROM 分别实现加/解密。由于加解密分别采用不同的 LUT 单元，而且 LUT 的输入/输出的端口数和位宽固定，使得面积资源开销增加的同时限制了架构适用算法的灵活性。

S-box 面积开销是影响可重构密码处理器面积资源的关键。Cryptoraptor 处理器为每个可重构计算单元设计局部寄存器文件 TLU (Table Lookup Unit)。为了支持广泛的密码算法，TLU 中提供了三个  $256 \times 32$ bit 和一个  $1024 \times 32$ bit 表（分别命名为 SBOX 和 TBOX）。TLU 提供三种方式来执行查表操作：(i) 一种查找表具有高达 10 位地址，返回一个 32bit 输出，(ii) 4 个查找表采用 32bit 输入的每个 8bit 作为一个地址返回四个 32bit 输出，以及 (iii) 每个查找表采用 32bit 输入的每个 8bit 作为输入，输出对应的每个 8bit 组成 32bit 输出。采用这样的设计方法，可以将较大的查找表分割成多个 SBOXes / TBOXes，并支持多级查找操作。所有局部寄存器文件的面积为  $0.88\text{mm}^2$ ，占整个密码处理器面积的 13.91%。文献[73]采用存储共享的设计方法实现 AES、DES 和 Serpent 三种算法，降低了实现 S-box 的寄存器 LUT 的面积开销。但是，由于缺乏对 S-box 操作并发访问的考虑，其设计仅能支持 AES 算法一轮函数展开，限制了密码算法重构阵列高性能的实现。

通过以上研究分析可知，可重构密码处理器在采用寄存器 LUT 的方式实现 S-box 时，主要研究如何在提高 S-box 替换操作的性能的同时减少寄存器文件占用的面积资源。为满足分组密码中 S-box 的差异化需求，提供目标算法集合所需的 S-box 表容量存储空间，需在此基础上探索面积效率最优的寄存器文件架构成为研究的关键。本文将基于分组密码算法中 S-box 的特征分析结果，考虑限制寄存器文件面积资源的关键参数，给出可重构密码处理器中实现 S-box 的最优寄存器文件的设计方法和架构实现。

## 4.3 面向多种分组密码算法的 S-box 约束研究

### 4.3.1 S-box 的算法特征分析

查表替换操作是分组密码算法中最常用的操作之一，本文第二章对分组密码算法集合的分析表明：超过 56.19% 的分组密码算法中包含一个或多个 S-box（替换盒）。S-box 的设计实现是影响密码系统的性能和面积的关键因素。S-box 作为算法实现中非线性运算单元，根据密码算法采用的替换方式的不同，在采用硬件实现时可以采用不同的实现方法。对于替换过程可以采用线性计算来替代的算法，定制的组合逻辑功能序列可以获得最佳的性能和面积效果；对于非线性的替换过程，则必须通过数据到地址的转换以及寄存器体，实现数据对应地址空间的索引完成替换操作，具有极高的算法通用性。本文详细分析分组密码算法替换盒 S-box 的各个方面，包括 S-box 的大小、输入数据宽度、寻址方案、不同数量的表、每个算法对表并行的需求等几个方面进行分析。

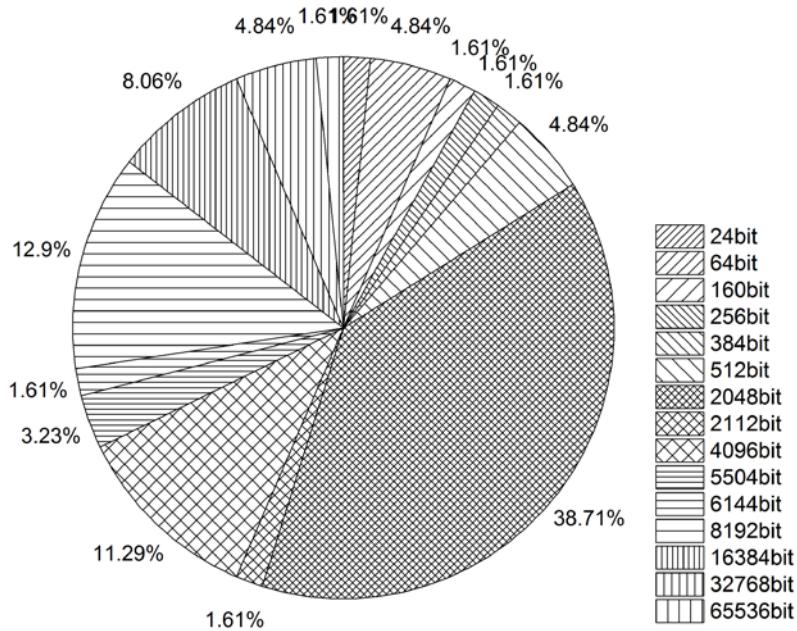


图 4-1 分组密码算法的 S-box 大小分布

如图 4-1 所示，密码算法中的替换盒 S-box 的大小从 24bit 到 65536bit 变化，不同算法所需的 S-box 大小差异巨大，最小 24bit 和最大 65536bit 的算法仅分别占据算法总量的 1.61%。大小为 2048bit、8192bit 和 4096bit 的 S-box 作为算法最常用的表大小，分别占据 38.71%、12.90% 和 11.29%。因此，基于算法研究分析，在实现 LUT 结构时至少应该满足 2048bit 大小，而当 LUT 的容量达到 8192bit 时则可以满足超过 86.48% 分组密码算法对 S-box 大小的需求。

理想的S-box的设计满足覆盖范围内的所有密码算法的单个S-box的容量和算法中并行访问的要求，任何不满足要求的S-box设计会导致算法性能的严重下降。从图4-2中可知，至少超过53.03%的分组算法采用了多个不同的S-box。其中，更有多达19.70%的算法使用了8个S-box。多个S-box的算法设计不仅需要LUT结构能够容纳所有的替换查找数据，而且要满足算法在流水过程中对多个S-box同时访问的需求。

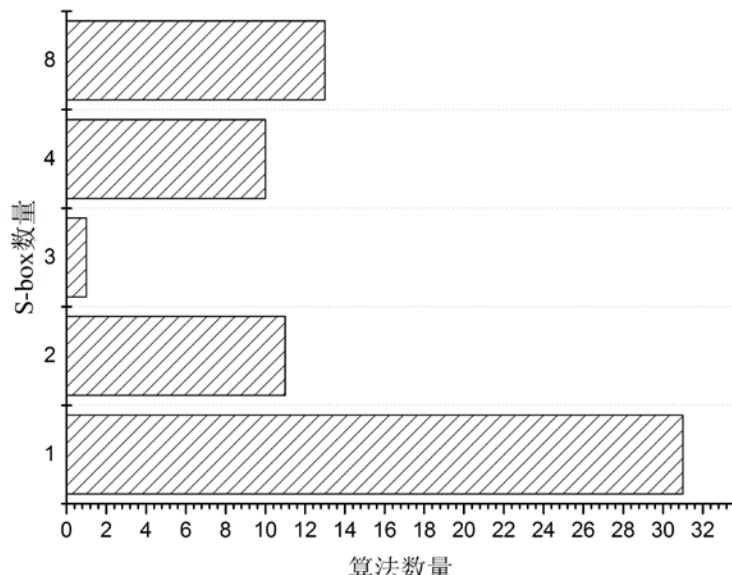


图 4-2 分组密码算法中 S-box 数量分布图

进一步分析 S-box 的并发访问次数。图 4-3 统计分组算法在每一轮中对 S-box 的并发访存的次数，54.17% 的分组密码算法在每轮计算中仅访问一次 S-box，但是仍然有 45.87% 的算法需要在每轮计算中多次访问 S-box。特别注意到，算法每轮函数中的多次访问可能是针对不同的 S-box，这进一步增加了 LUT 结构设计的复杂性。

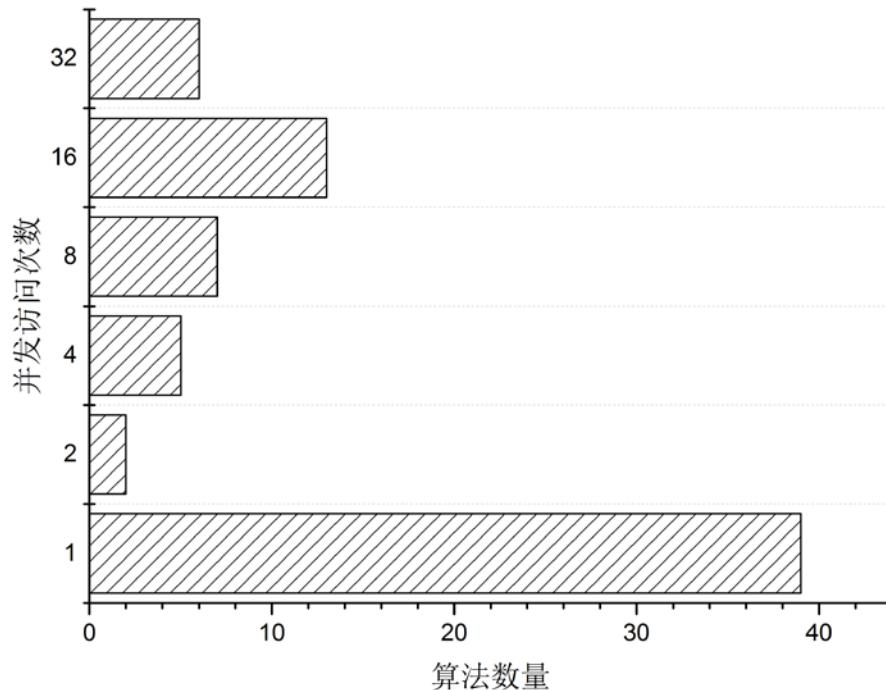


图 4-3 分组密码算法中每轮计算对 S-box 的并发访存次数

输入位宽和输出位宽是查找表结构的另一个重要因素。过大的输入输出位宽会导致资源的严重浪费，过小的输入输出位宽难以满足密码算法的高性能需求，因此，表的大小和表的输入输出位宽是S-box的设计关键。

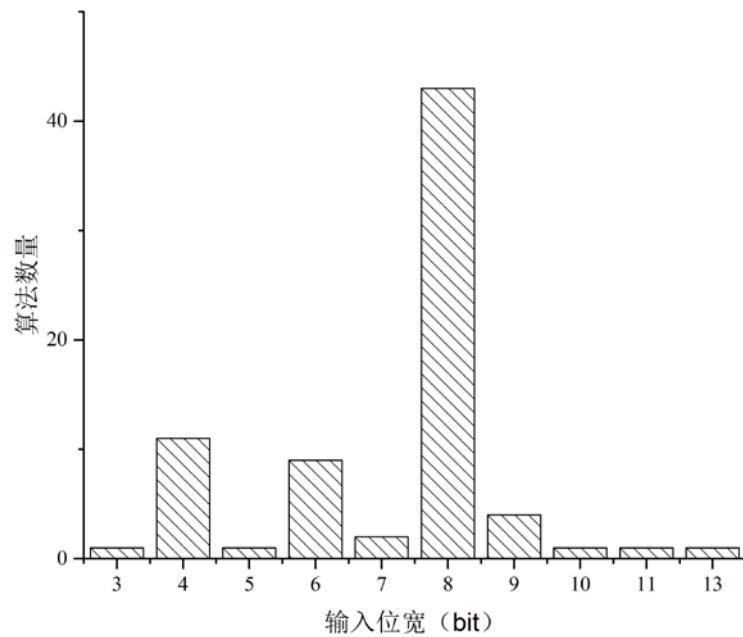


图 4-4 分组密码算法的输入位宽统计

图4-4给出了分组密码算法的输入位宽的统计情况，从图中可知分组密码算法的输入位宽分布在3bit到13bit的范围内。高达58.11%的算法采用8bit的输入位宽，仅有9.46%的算法采用S-box的输入位宽超过8bit。在实现S-box的查找表结构时，不必要的大输入位宽会造成严重的资源浪费；而输入位宽的不足造成更多的查找执行周期，则会导致计算性能的损失。

图4-5给出分组密码算法输出位宽的统计结果，从图中可知输出位宽分布在2bit到32bit范围内。其中，输出位宽8bit、4bit和32bit作为分别占据所有密码总量的55.41%、22.97%和9.46%。对于少数算法所具有的异常的输出位宽，在S-box实现过程中可以将表内容分别放置到多个并行的替换表中，采用输出数据拼接的方式实现多种数据宽度的输出。

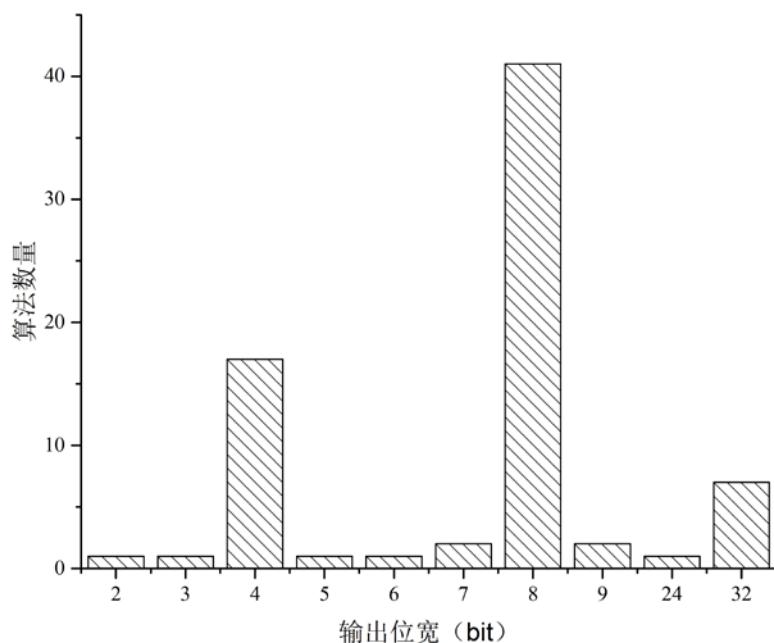


图 4-5 分组密码算法的输出位宽统计

通过以上算法分析可知，分组密码的 S-box 可以分为 3 类：

- (1) 算法中所有轮数均采用相同的 S-box，如 AES，3-Way 等。
- (2) 算法同一轮计算中采用多个 S-box，而不同轮计算采用相同的 S-box，如 DES、NSSU 等。
- (3) 算法中同一轮计算采用相同的 S-box，而不同轮计算采用的 S-box 不同，如 SERPENT 等。

最常见的 S-box 结构为  $8 \times 8$  (8 入 8 出) 或  $8 \times 32$  (8 入 32 出)。同时，S-box 还需要支持并发访存的需求，对 S-box 的访存端口的数目要求较高。当设计不能满足需求时，可能会节省一些面积资源，但也会大大降低密码处理器的性能。

### 4.3.2 S-box 对局部寄存器文件的设计约束

通过以上对分组密码算法S-box的特征分析可知，S-box的特征受到密码算法中表数量，每轮并发访问数，输入位宽和输出位宽的影响。表4-1给出部分算法的S-box特征参数表示。

其中，N (Number) 代表表个数，IW (InputWidth) 代表输入位数，OW (OutputWidth) 代表输出位数，R 代表每个算法每轮的并发数。以算法AES和DES为例，AES算法每轮查询16个相同的  $8 \times 8$ bit的表，DES算法每轮查询8个不同的  $6 \times 4$ bit的表。因而，对于AES算法，每轮仅需2048bit容量的

数据，但是相同数据能够支持并行度为16的同时访问；对于DES算法，每轮需要8个256bit容量共计2048bit总容量的数据，每个数据仅需支持并发度为1的访问。

表 4-1 部分算法的 S-box 特征参数

算法	轮数 (L)	表个数 (N)	输入位数 (IW)	输出位数 (OW)	每轮每表并发数 (R)
AES	10	16	8	8	1
Blowfish	16	1	8	32	4
Camellia	18	1	8	8	4
CAST128	16	1	8	32	8
DES	16	1	6	4	8
GOST	32	1	4	4	8
KASUMI	6	1	9	9	1
			7	7	1
SEED	12	4	8	8	2
Twofish	16	2	8	8	4
Serpent	32	32	4	4	8

分组密码算法在其高性能实现时会按照算法轮数在硬件上完全展开，此时S-box需要同时满足多轮算法同时计算时对表容量和并发度的需求。对于目标算法集合，密码算法展开的轮数L各不相同，算法特征可以用以下参数表示：

- 表容量TS (Table Size)
- 并发访问量P (Parallel)
- 每轮并发输入位宽PIWPR (ParallelInputWidthPerRound)
- 每轮并发输出位宽POWPR (ParallelOutputWidthPerRound)

为了满足算法实现S-box替换功能所需的查找表容量，算法每轮所需表容量TS的表达式为：

$$TS = N \times OW \times 2^{IW} \quad (4.1)$$

由于算法展开之后，算法并行展开之后的查找表容量不变。但是，S-box的并发访问数量P为：

$$P = L \times R \times N \quad (4.2)$$

其中，L为密码算法并行实现展开的流水线轮数。

算法展开之后，每轮查找表所需的并发数据输入位宽PIWPR为：

$$PIWPR = N \times IW \times R \quad (4.3)$$

同时，不仅要满足并发数据位宽的需求，而且要满足并发访问量PTPR为：

$$POWPR = N \times OW \times R \quad (4.4)$$

以上是密码算法在展开之后，S-box 函数的特征参数。下面将就 S-box 特征参数下，可重构架构采用寄存器 LUT 实现 S-box 进行理论分析和设计空间探索。

## 4.4 一种多端口统一结构的跨域寄存器文件的提出

### 4.4.1 寄存器文件的面积表达式

为了能够满足不同密码算法的高性能替换操作实现，寄存器LUT的设计难点在于满足不同S-box替换操作的功能设计、减少寄存器文件容量、以及优化寄存器文件的关键路径。因此，如何通过算

法分析, 寻找最优的寄存器LUT端口互联设计, 实现减少寄存器LUT的数量以及减少单个LUT的面积, 成为替换操作寄存器设计的核心问题。如何利用有限的寄存器LUT数量, 满足所有目标密码算法S-box的需求, 也是实现面积效率(性能面积比)的关键。

理想的寄存器 LUT 结构示意图如下:

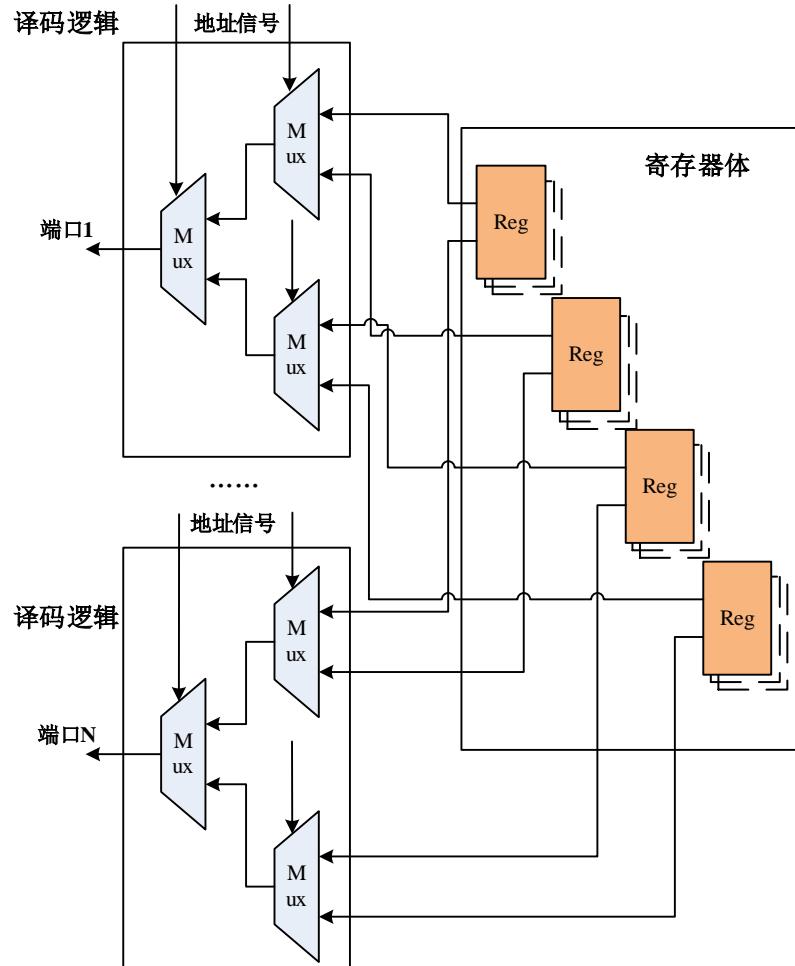


图 4-6 理想 S-box 的结构示意图

寄存器 LUT 的结构中包括寄存器文件、端口译码逻辑, 整个 S-box 的面积 TA (Total Area) 可以表达为:

$$TA = RA + MA \quad (4.5)$$

其中, RA (Register Area) 是所有寄存器块面积, MA (Mux Area) 是所有端口译码逻辑的面积。表达式中RN (Register Number) 是寄存器块个数, PN (Port Number) 是并发端口数。

$$RA = SRA \times RN \quad (4.6)$$

$$MA = SMA \times PN \times RN \quad (4.7)$$

单个寄存器块的面积SRA的表达式为:

$$SRA = RAU \times RDW \times 2^{RAW} \quad (4.8)$$

单个端口译码逻辑的面积SMA的表达式为:

$$SMA = MAU \times RDW \times (2^{RAW} - 1) \quad (4.9)$$

当 $2^{RAW}$ 远远大于1时, SMA可以简化为:

$$SMA = MAU \times RDW \times 2^{RAW} \quad (4.10)$$

其中, RAU(Register Area Unit)是单位寄存器在目标工艺库下的面积参数, MAU(Mux Area Unit)是单位多路选择器在目标工艺库下的面积参数。将SRA和SMA的表达式带入(4.6)和(4.7)式可知:

$$RA = RAU \times RDW \times 2^{RAW} \times RN \quad (4.11)$$

$$MA = MAU \times RDW \times 2^{RAW} \times PN \times RN \quad (4.12)$$

于是, 整个寄存器LUT的面积TA可以表示为:

$$\begin{aligned} TA &= RAU \times RDW \times 2^{RAW} \times RN + MAU \times RDW \times 2^{RAW} \times PN \times RN \\ &= (RAU + MAU \times PN) \times RDW \times 2^{RAW} \times RN \end{aligned} \quad (4.13)$$

上式寄存器LUT面积公式中, RAU和MAU是常量, TA与变量PN, RDW, RN成正比, 与变量RAW成指数变化关系。进一步考虑:

- 当RA为定值时:  $TA = RA \times (1 + MAU/RAU)$  (4.14)

即当RA为定值时, S-box的面积与PN成正比关系。

- 当MA为定值时:  $TA = MA \times (1 + RAU/MAU)$  (4.15)

即当MA为定值时, 寄存器LUT的面积与PN成反比关系。

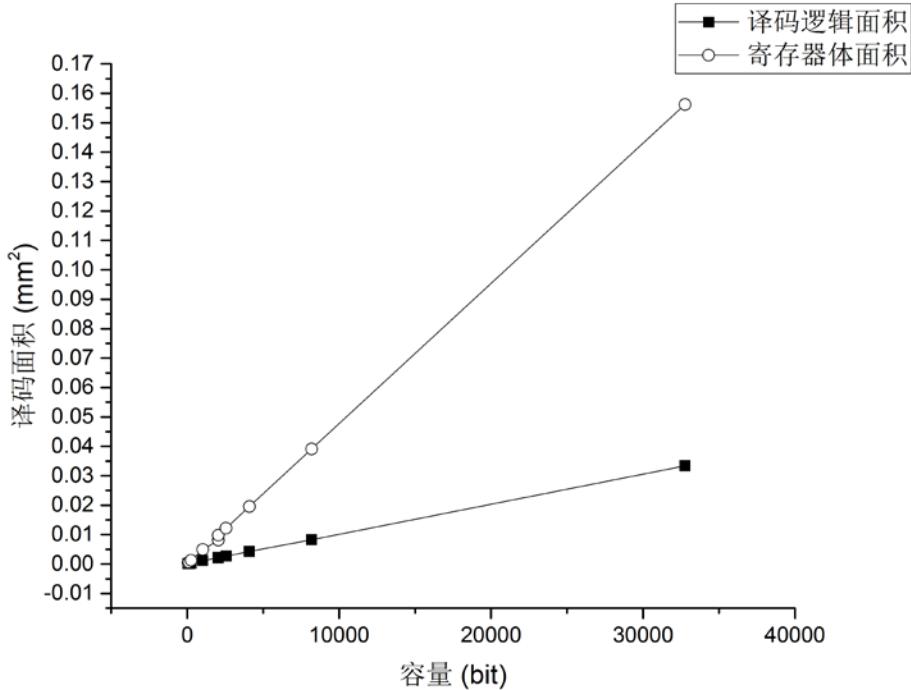


图 4-7 端口数为 1 时, 寄存器体和译码逻辑面积与寄存器容量的关系

#### 4.4.2 寄存器文件的面积参数分析

针对不同的物理实现工艺条件, 可以确定不同的MUX和RUX的参数值。以TSMC公司40nm IP8M工艺库作为Design Compiler综合条件, 分两种情况进行讨论:

1、端口数固定, 寄存器和译码逻辑面积与容量的变化关系。

如图 4-7 所示, 当寄存器的读端口为 1 时, 端口译码逻辑与寄存器的容量成线性关系, 即随着寄存器容量的增大, 端口译码逻辑线性增大, 系数为 1.02。当寄存器端口为 1 时寄存器面积与寄存器的容量同样成正比关系, 系数为 4.76。在单端口的情况下, 寄存器的面积只和寄存器的容量有关, 且随着寄存器容量的增大, 寄存器文件的面积增加较快, 端口译码逻辑的面积增长相对较慢。

根据以上分析, 将面积变量代入公式, 可知  $MUX=1.02$ ,  $RUX=4.74$ 。整个寄存器 LUT 的面积公式解析式可以表示为:

$$\begin{aligned} TA &= RAU \times RDW \times 2^{RAW} \times RN + MAU \times RDW \times 2^{RAW} \times PN \times RN \\ &= (4.74 + 1.02 \times PN) \times RDW \times 2^{RAW} \times RN \end{aligned} \quad (4.16)$$

## 2、寄存器容量固定, 译码逻辑面积与端口数的变化关系

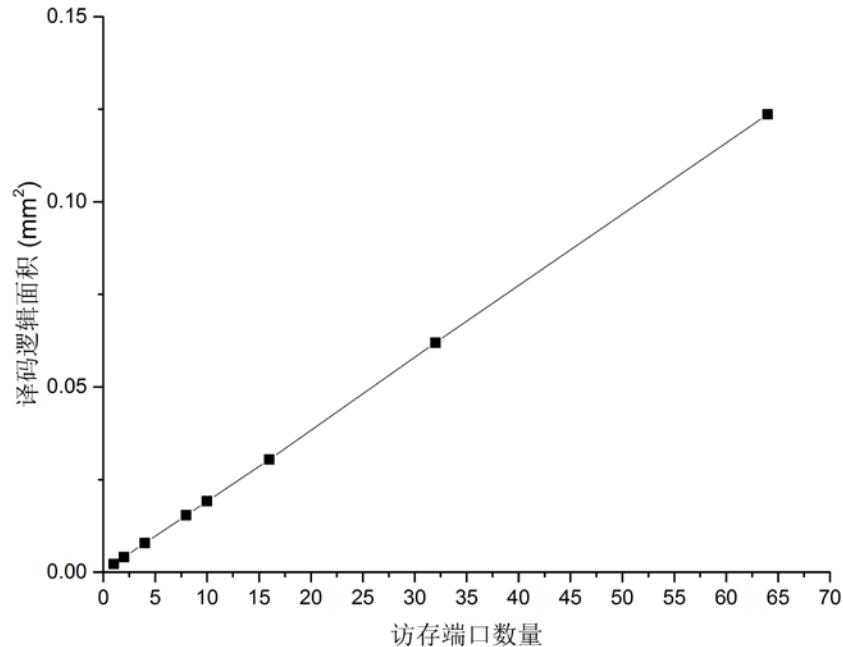


图 4-8 寄存器容量固定时译码逻辑面积与端口数变化的趋势

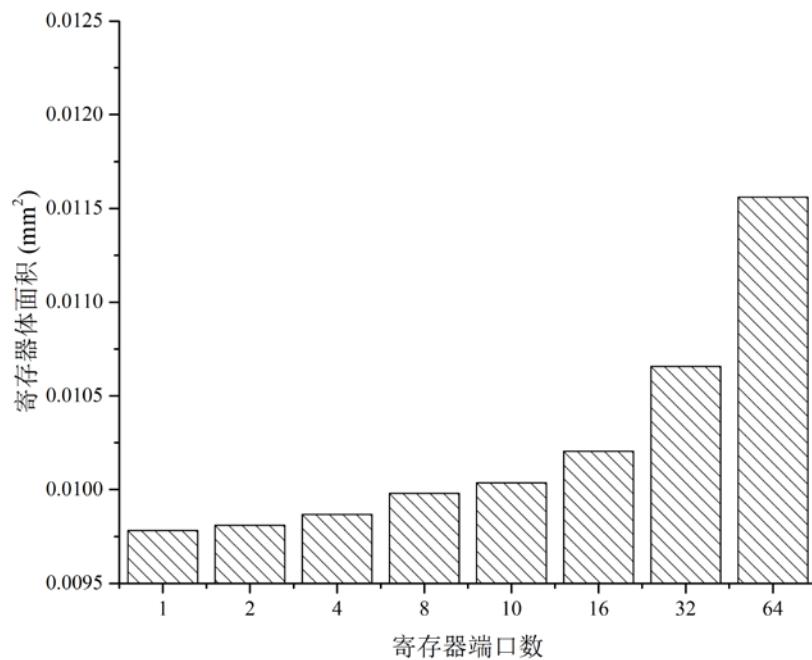


图 4-9 寄存器容量固定时寄存器体面积与端口数变化的趋势

如图 4-8 所示, 横坐标为端口数, 纵坐标为相应端口数时的端口译码逻辑面积与端口数为 1 时候的端口译码逻辑比值。当寄存器容量固定, 寄存器文件的端口数从 1 逐渐变化到 32 时, 端口译码逻辑面积随着端口数的变大而增大, 其变化率约为 0.8796 倍。

从综合结果来看，当端口数增多时寄存器文件的面积也会稍大，如图 4-9 所示。这是由于多端口输出导致对寄存器文件的驱动能力需求变大，单个寄存器文件的输出 Buffer 增加。鉴于其变化幅度不大于 0.29%，此处忽略不计。

#### 4.4.3 S-box 特征约束下的架构参数探索

本文将分析包括 AES, Blowfish 等主流分组算法在流水线完全展开的条件下，S-box 的硬件架构设计需要满足下表所列的约束条件。而针对未来标准更新或密码算法集合的变化，只需代入新的算法特征信息，采用同样的方法即可以确定寄存器文件的架构，因此本文提出的方法具有通用性。

表 4-2 分组密码算法特征

算法	轮数 (L)	表容量 (TS)	并发访问数 量 (P)	每轮并发输入位宽 (PIWPR)	每轮并发输出位宽 (POWPR)
AES	10	2048	160	128	128
Blowfish	16	2048	128	32	128
Camellia	18	8192	96	32	32
CAST128	16	32768	64	64	256
DES	16	2048	128	48	32
GOST	32	512	256	32	32
KASUMI	6	9216	18	9/7	9/7
SEED	12	4096	96	64	64
Twofish	16	4096	128	64	64
Serpent	32	2048	8192	1024	1024

此时，可重构密码处理器架构设计参数满足如下的约束条件：

(1) 寄存器总容量RS (RegisterSize) 的约束。

$$RS = RDW \times 2^{RAW} \times RN \geq TS \quad (4.17)$$

寄存器总容量需要大于或者等于 S-box 算法所需的总查找表容量，才能保证 S-box 算法所需的查找表数据能够完整的存放在寄存器中。

(2) 寄存器总并发访问数RP (RegisterParallel) 的约束。

$$RP = RN \times PN \geq P \quad (4.18)$$

寄存器的个数和每个寄存器文件的端口数的乘积需要满足算法并行展开之后的最大访问数量。

(3) 每轮寄存器的并发输入位宽RIWPR (RegisterInputWidthPerRound) 的约束。

$$RIWPR = RAW \times RN \times PN/L \geq PIWPR \quad (4.19)$$

寄存器的并发访问输入数据位宽需要满足每轮计算对查找表寻址需求的最大数据位宽，保证寄存器的输入数据位宽满足算法需求。

(4) 每轮寄存器的并发输出位宽ROWPR (RegisterOutputWidthPerRound) 的约束

$$ROWPR = RDW \times RN \times PN/L \geq POWPR \quad (4.20)$$

寄存器的并发访问输出数据位宽需要满足每轮计算对查找表最大并发输出的需求，保证寄存器输出数据位宽和寄存器端口数量的满足算法需求。下面给出寄存器文件空间探索的伪码：

```

For i=1 to RN
  For j= 1 to RDW
    For k = 1 to RAW
      For m=1 to PN
        If (i*j*2^k > Ts) and (i*m>P)
          Set i,j,k,m to arrayA[p],p++
        End If
      End For
    End For
  End For
End For

For p=0 to Pmax
  Area = TA=RA+MA (iCalculate area of LUT accoding i, j, k, m in arrayA)
  Set area to Result[q]
End For

Result = min(Result[q])

```

在上述约束条件下，可以在寄存器地址宽度、寄存器数据宽度、寄存器块个数和寄存器端口数组成的4维设计空间进行探索。寄存器文件架构的空间探索首先满足容量和并发访问的约束，然后探索不同寄存器体数量和端口数对寄存器文件面积的影响。

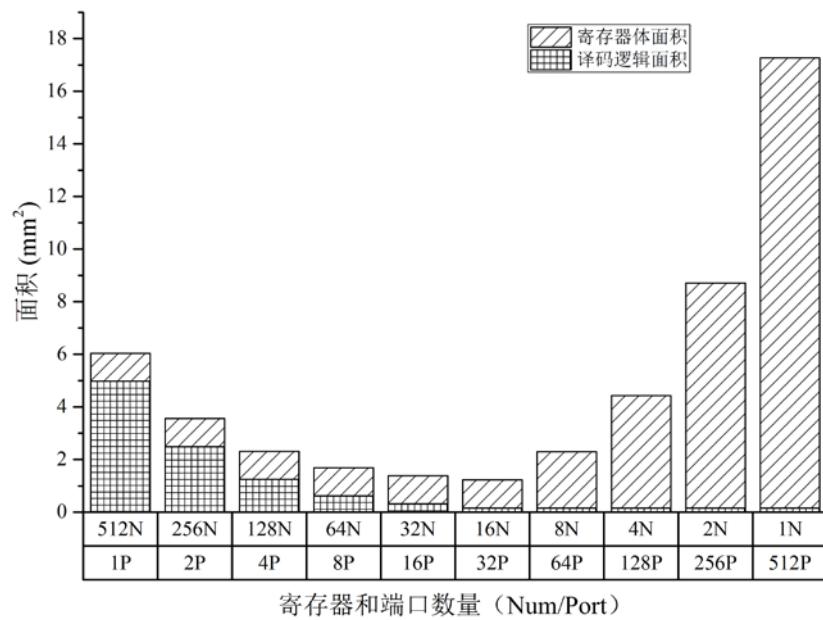


图 4-10 寄存器体和译码逻辑面积随寄存器数量和端口数变化趋势

图4-10给出了设计参数空间探索过程中不同寄存器端口数和数量变化过程中，寄存器体面积和译码逻辑面积的变化趋势。从图中可以看到，随着寄存器体数量的减少，寄存器体面积逐渐达到最小值不再变化，最小值在16N/32P的条件下获得；同时，随着端口数的增加，当端口数大于32P之后，译码逻辑的面积占寄存器文件面积的百分比不断增大。在算法流水线完全展开，同时满足约束条件的情况下，16个寄存器体，32个访存端口的设计可以获得最小的寄存器文件面积。寄存器体端口数为1，数量为512时，面积是最优设计参数的4.93倍。特别注意到，当寄存器体端口数为512，数量为1时，面积是最优设计参数的14.10倍。此时，用于地址译码的逻辑远远大于存储体的面积。

#### 4.4.4 多端口统一结构的跨域寄存器结构设计

在目标算法集合的约束条件下，为了满足算法输入数据位宽和输出数据位宽的差异化需求，需要对寄存器文件进行面积优化的物理实现，得到最优的寄存器LUT架构设计参数。寄存器数据宽度和寄存器地址宽度在适应不同算法的输出数据位宽和输入数据位宽。对于寄存器数据宽度大于输出数据位宽时，可以仅使用寄存器的低数据位；而当寄存器输出位宽小于输出数据位宽时，可以在不增加逻辑资源的条件下直接硬线连接完成数据的拼接。同理，当寄存器地址宽度大于输入数据位宽时，可以仅使用寄存器的低地址空间。但是，当寄存器地址宽度小于输入数据宽度时，需要增加额外的译码选择逻辑对数据进行选择。

为了满足目标算法集合差异化的数据访存方式，本文提出多端口统一结构的跨域寄存器文件（Cross-domain Register File, CDRF）结构满足不同算法的动态重构需求。输入数据在跨域寄存器内根据配置信息进行动态重组，根据S-box的输入和输出位宽的差异，可以将其分为3种模式，利用2比特的配置信息可以满足目标算法的所有S-box替换算法的功能需求。

模式1、8入8出：输入8bit数据，输出8bit数据。此配置情况同时满足所有小于8入8出的S-box算法需求，如6入4出，4入4出。

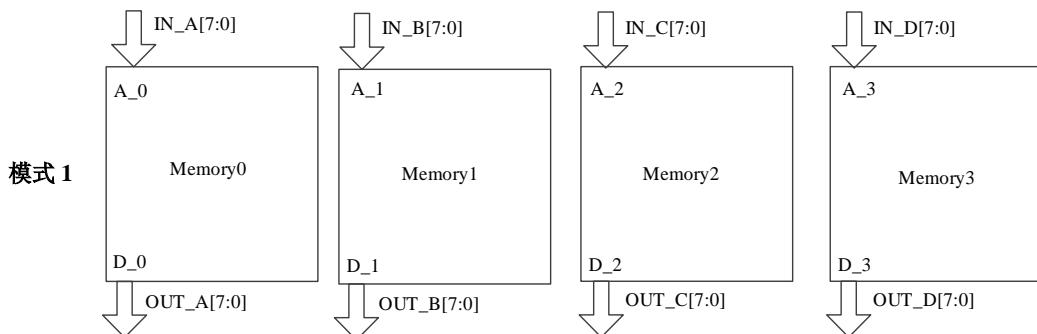


图 4-11 模式 1 数据输入输出示意图

模式2、9入16出：输入9bit数据，输出9bit数据。此配置情况主要用于满足9入9出等S-box算法需求。

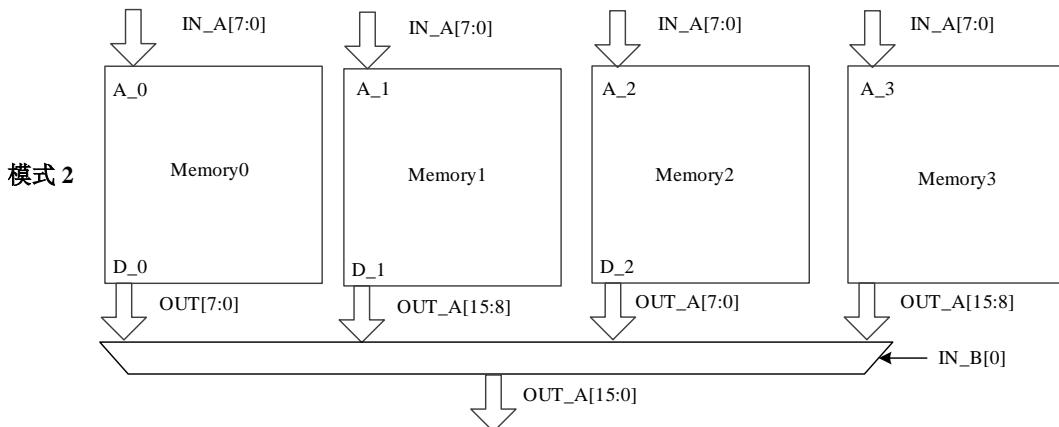


图 4-12 模式 2 数据输入输出示意图

模式3、8入32出：输出8bit数据，输出32bit数据。此配置条件下，通过将单个8bit输入数据同时作为4个寄存器体的寻址地址，在输出端将4个8bit数据拼接组合成为一个32bit数据。

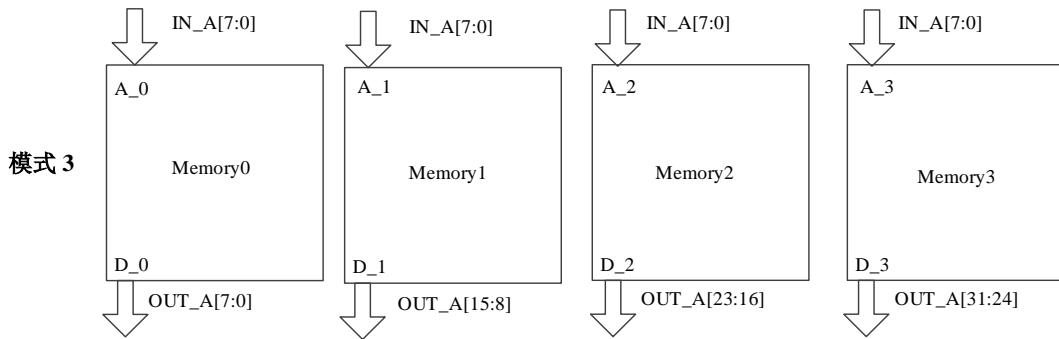


图 4-13 模式 3 数据输入输出示意图

当目标算法集合按照算法执行标准展开所有轮数之后，最大需要支持128bit数据位宽，最多需要支持32轮展开，此时S-box算法的跨域寄存器文件架构如下所示。如图4-14所示，理想的可重构单元行能够实现分组密码算法的单个轮函数，因而仅需32行可重构单元组成整个重构计算阵列。整个跨域寄存器文件包含4个寄存器文件簇，每个寄存器文件簇包含32个读取端口和一个写入端口。

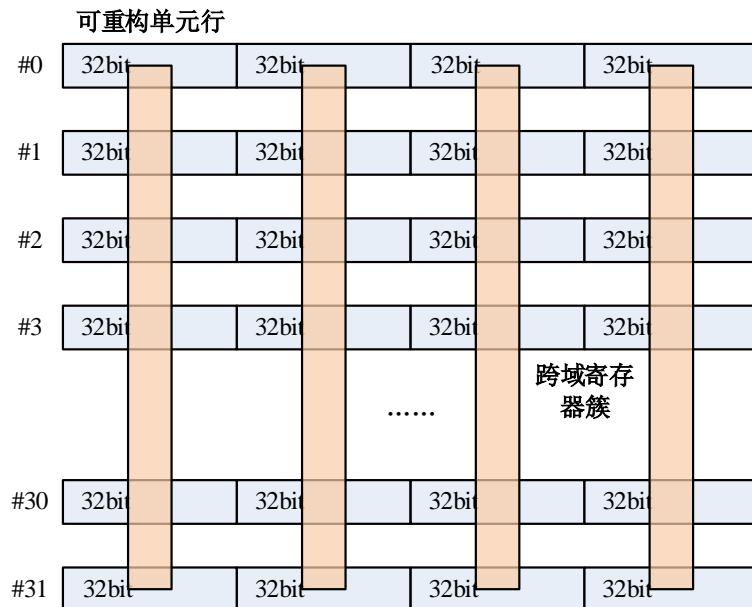


图 4-14 可重构跨域寄存器文件示意图

图4-15给出了跨域寄存器簇的结构设计细节。每行可重构计算单元均分别有4个输入和输出端口与跨域寄存器簇相连，同时2bit的重构配置信息用于适配不同的密码算法。为了满足不同算法对不同输入、输出位宽和容量的需求，每个寄存器文件簇包含4个8入8出的寄存器体，同时分别包含32个输入控制单元和输出控制单元。每行重构单元的输入信号经过输入控制单元的处理后分别作为寻址地址输入寄存器体，寄存器体的寻址结果经过输出控制单元的处理后返回重构单元行。以第#0行处理单元为例，寻址数据IN\_A#0、IN\_B#0、IN\_C#0和IN\_D#0输入到#0输入控制单元，在配置信息的控制下输入控制单元输出数据A1\_0、A2\_0、A3\_0和A4\_0分别连接到4个寄存器体的A\_0接口，寄存器体的输出数据D1\_0、D2\_0、D3\_0和D4\_0经过输出控制单元的处理后返回替换操作的结果OUT\_A#0、OUT\_B#0、OUT\_C#0和OUT\_D#0。

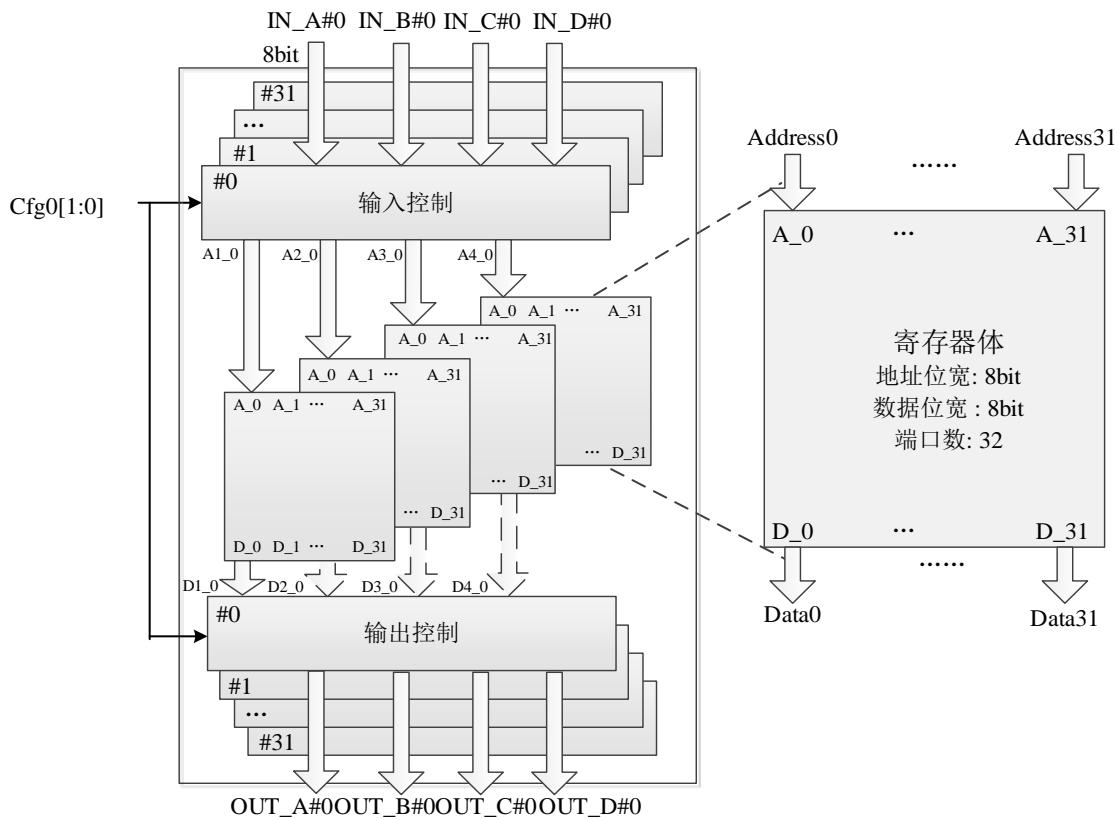


图 4-15 跨域寄存器文件簇结构示意

图4-16给出了跨域寄存器文件簇中的输入控制单元的设计实现结构图。从图中可知，来自重构单元行的输入信号IN\_A, IN\_B, IN\_C和IN\_D根据配置信息进行处理并选择输出。在模式1中，4个8bit的输入信号组成一个32bit的信号；在模式2/3中，8bit的输入信号IN\_A复制4份组成32bit的信号。

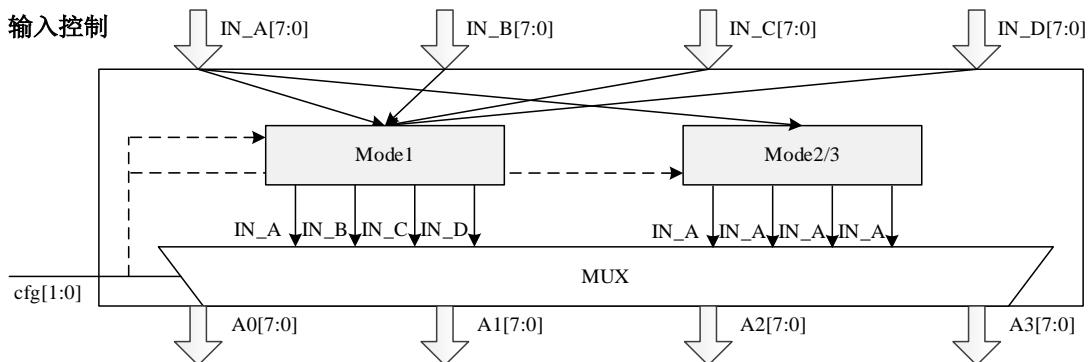


图 4-16 输入控制逻辑结构示意图

图4-17给出了跨域寄存器文件中的输出控制单元的设计实现结构图。从图中可知，来自寄存体的输出数据D0、D1、D2和D3根据配置信息进行处理器并选择输出。在模式1/3中，4个8bit的数据顺序组成一个32bit的数据。在模式2中，D0、D1和D2、D3分别组成16bit的数据，同时根据输入信号IN\_B的最低位进行输出选择。当IN\_B[0]等于1时，输出D2和D3的组合；当IN\_B[0]等于0时，输出D0和D1的组合。选择的结果D0'和D1'再进行复制组成一个32bit的数据。

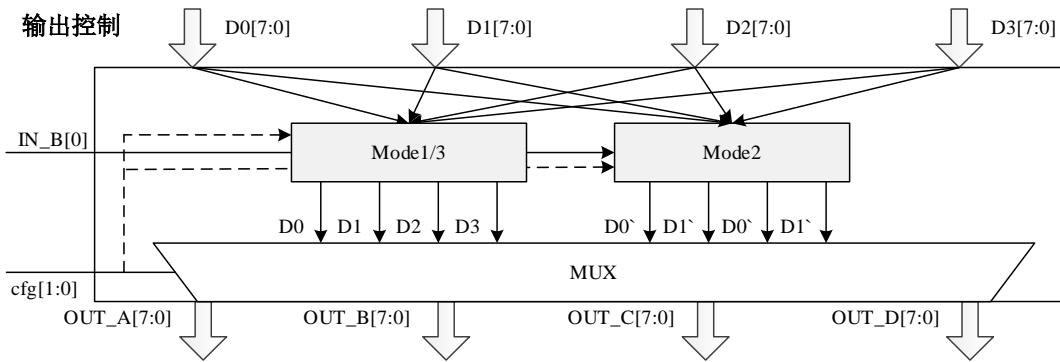


图 4-17 输出控制逻辑结构示意图

## 4.5 实验与分析

本文提出的S-box多端口统一结构的跨域寄存器文件（Cross-domain Register File, CDRF）结构可以支持多种分组密码算法的高性能实现。为了能够与其他架构的S-box设计方法对比分析，本文在TSMC 40nm CMOS工艺下复现类似的工作。表4-3给出了本文提出的跨域寄存器文件（CDRF）与文献[33]中的TLU的对比结果，其中TLU以寄存器LUT的方式实现算法中的S-box操作。为了对比两种方法的面积开销，本文将一个跨域寄存器文件的变种用于文献[33]的可重构密码处理器中以替代其原本的TLU。在比较两者面积时，本文选取同样的目标算法集合，并保持目标算法集合的性能统一。TLU中包含3个输入位宽为8bit的SBOX和1个输入位宽为10bit的TBOX，在同一时刻4个表中仅有1个表会被使用，因而设计利用率极低；同时这些表采用32bit位宽，导致冗余的面积开销增加。

表 4-3 面积对比 Cryptoraptor 架构 S-box 设计实现

目标算法	算法定义		性能 (BPC)	LUT的描述		LUT的面积( $\text{mm}^2$ )	
	Round	Block Size		TLUs	CDRF	TLUs	CDRF
AES	10	128	1.00	寄存器体：单端口 数量：80 容量： 1024×32bit 数量：240 容量： 256×8bit	寄存器体：16端口 数量：16 容量： 256×8bit	0.88	0.65
Blowfish	16	64	1.33				
Camellia	18	128	0.50				
CAST-128	16	64	1.00				
KASUMI	8	64	0.25				
DES	16	64	0.67				
GOST	32	64	0.80				
Seed	12	128	0.13				
Twofish	16	128	0.50				

如前文所述，精心设计的跨域寄存器文件参数才能保证其面积最小。按照表4-3中算法及其性能的约束，4个跨域寄存器簇可以代替TLU，其中每个跨域寄存器簇中包含4个存储器体，每个存储器体的输入位宽为8比特，输出位宽为8比特，端口数为16个。如表4-3所示，本文提出的跨域寄存器文件的面积开销较TLU面积减少26.14%。CDRF结构较TLU设计具有面积优势的两个原因是：

- (1) CDRF结构的存储器容量使用率高于TLU设计。
- (2) CDRF结构通过增加单个存储器的端口而不是增加存储器的数量来满足并行访问。

表 4-4 给出了本文设计与另一个基于寄存器 LUT 实现 S-box 方案的对比。从表中可以看出，文

献[73]设计中面积是最小的。但是按照其描述，一组 205bit 的寄存器，每个寄存器 16 个端口的设计仅仅能完成一轮 AES 的 S-box，导致 AES 的性能低下（0.1BPC）。TLU 和 CDRF 的面积开销相对文献[73]较大，因为要保证众多的分组密码算法同时具有高的性能。文献[73]的另一个缺点是，其只针对 AES、DES 和 Serpent 三种算法进行设计优化。因此，文献[73]的方法不适合密码算法的可重构并行实现。

表 4-4 不同架构实现 AES 算法的 S-box 面积对比

	目标算法	AES的性能 (BPC)	LUT的面积 (mm <sup>2</sup> )
TLUs <sup>[33]</sup>	AES, Blowfish, Camellia, CAST-128, KASUMI, DES, GOST, Seed, Twofish	1	0.88
文献[73]	AES, DES, Serpent	0.1	0.05*
CDRF	AES, Blowfish, Camellia, CAST-128, KASUMI, DES, GOST, Seed, Twofish, Serpent	1	0.65

## 4.6 本章小结

本章通过对分组密码算法中 S-box 的特征分析，分析影响 S-box 的设计约束，在总结寄存器文件的面积表达式的基础上，对算法约束下寄存器文件的设计参数进行空间探索，给出目标算法集合条件下 S-box 实现的最优解。本文根据设计探索的结果，采用跨域寄存器文件的设计方法，可以有效减少密码处理器的面积开销高达 26.14%，同时为密码处理器的关键路径设计留有足够的余量。

## 第五章 基于混合寄存器文件的可重构密码处理器实现与分析

### 5.1 前言

本章综合采用本文对寄存器文件研究中所采用的各种优化设计方法和策略，选取主流分组密码算法作为实验测试集合，通过分析对比不同的寄存器文件设计方案，量化说明本文研究的寄存器文件对可重构密码处理器性能面积的优化作用；对比不同密码架构实现方案，体现本文设计实现的密码处理器在高性能和灵活性的优势。

本章首先说明课题组实现的可重构密码处理器硬件结构，描述处理器的设计参数。然后说明可重构密码处理器的 FPGA 验证原型和处理器版图的物理实现。最后在对比实验和结果分析部分，① 对比分组密码算法在不同寄存器文件设计参数下的性能面积差异；② 通过与 ASIC、FPGA 实现和其他可重构架构的性能对比，分析设计方案对架构设计中性能和面积的影响。

### 5.2 可重构密码处理器硬件结构设计

本节针对 AES、DES、Blowfish、SM4 等国际、国内标准分组密码算法或标准参选分组密码算法的特征，基于本文研究方法内容设计了可重构密码阵列结构，包括计算单元结构、寄存器文件结构、互联单元结构等主要模块。在此可重构密码阵列结构的基础上，进一步设计了可重构密码处理器结构，以进行算法验证和硬件实现。

#### 5.2.1 可重构计算单元结构设计

根据本文 2.2.2 节分组密码特征的分析结果，可重构计算单元应针对算术操作、逻辑操作、查表替换、移位/旋转、置换/扩展等密码算子级操作完成结构设计。然而，由于每轮密码算法包含多个算子级操作，如果按照实现单个算子级操作设计计算单元结构，将使得所需的计算单元个数急剧上升，从而导致计算单元硬件利用率下降，密码算法配置信息规模上升，最终影响可重构密码阵列的面积效率。

通过分析目标典型分组密码算法的算子级操作执行序列和频度，综合算子级操作所需计算资源和计算延时的影响，将多种操作进行叠加，采用同一计算单元在单周期内实现，如表 5-1 所示。

本文将 15 种原始操作序列进行操作的叠加，以压缩算法的执行操作。15 类原始操作序列的叠加优化可以分为以下 4 种情况：

- (1) 替换操作。不同分组密码算法的替换操作前后存在异或，因而本文设计的重构计算单元也为替换操作的前后分别叠加了两个异或操作。在可重构密码处理器的计算过程中，采用配置信息进行操作算子的选择。
- (2) 计算操作。计算操作包括加、减、模加、模减、乘法和模乘操作，通过在计算操作之后叠加异或操作并不会对计算延时造成影响。
- (3) 逻辑和移位操作。由于分组密码算法中与/或操作之后往往包含移位操作，因而设计可重构计算单元将两类算子进行级联叠加。

(4) 异或操作。由于算法设计中数据在不同传播路径上经过的操作不同, 因而可能会出现三个数据进行异或操作的行为, 将多个异或操作进行叠加仅需增加三输入的叠加操作算子即可。

表 5-1 密码算法的操作序列叠加

编号	原始操作	操作序列叠加
1	替换操作	异或操作+查表替换+异或操作
2	加法操作	加法操作+异或操作
3	减法操作	减法操作+异或操作
4	模加法操作	模加法操作+异或操作
5	模减法操作	模减法操作+异或操作
6	乘法操作	乘法操作+异或操作
7	模乘操作	模乘操作+异或操作
8	左移操作	与操作+左移操作
9	右移操作	或操作+右移操作
10	或操作	或操作+左移操作
11	与操作	与操作+右移操作
12	非操作	非操作
13	异或操作	异或操作+异或操作
14	交换操作	交换操作
15	扩展交换操作	扩展交换操作

下面以 AES 算法为例, 分析说明采用基于算法操作序列叠加的计算单元结构设计, 以及对密码算法性能提升的影响。AES 算法的加解密过程如图 5-1 所示, 解密过程与加密过程对称。加密过程中, 128bit 明文首先做密钥加操作, 然后进行 10 轮加密迭代, 最终输出密文。

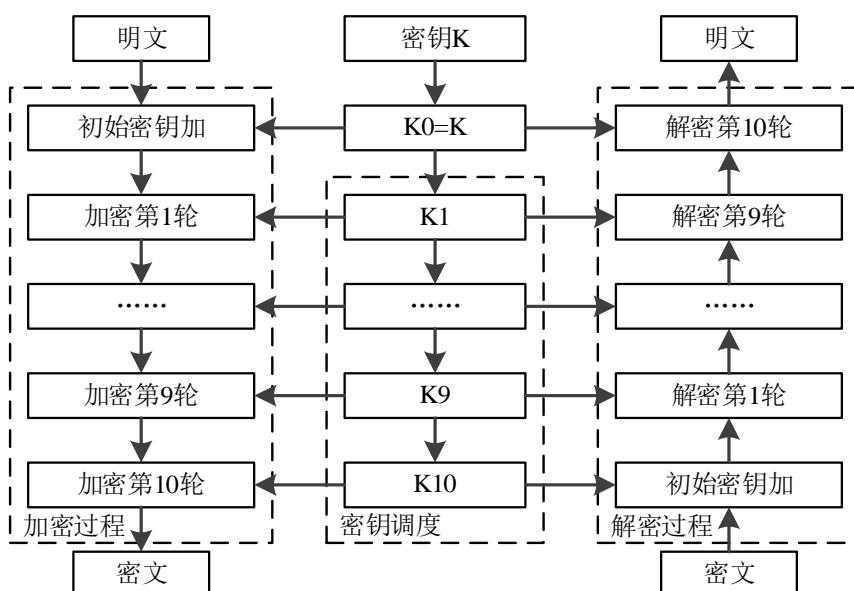


图 5-1 AES 算法的加解密过程示意图

其中, 加解密过程中的第 10 轮与前面的 9 轮操作上存在细小差异, 如图 5-2 所示。加密过程中

的前 9 轮函数分别包含字节替换、行移位、列混合和密钥加的操作，而加密操作的第 10 轮函数则仅包含字节替换、行移位和密钥加。解密过程的轮函数与加密过程类似。

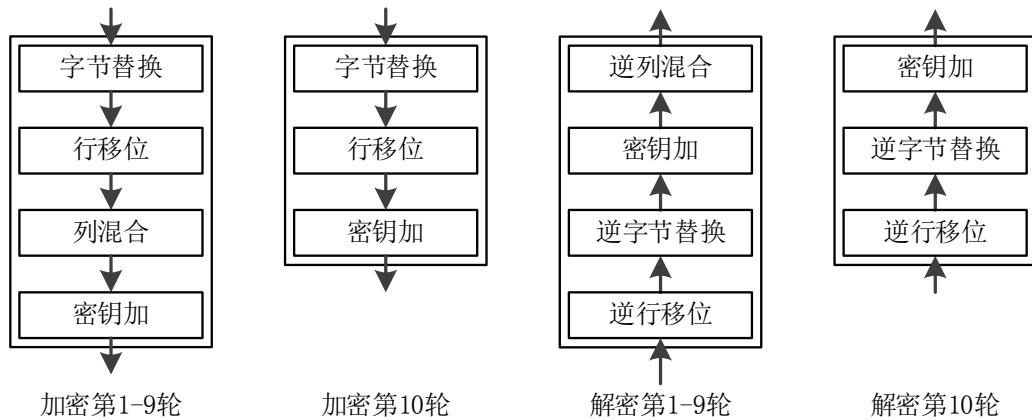


图 5-2 AES 算法的加解密轮函数示意图

针对 AES 算法轮函数算子级操作，采用原始操作序列和操作序列叠加方式的计算单元结构，其映射过程对比如图 5-3 所示。密码算法在完全展开之后，数据之间具有 RAW 的特性，需要按照算法设计逐轮完成迭代计算。可重构计算单元用一个或多个操作组成操作序列来实现整个算法。本文利用操作序列叠加方法，可以获得操作叠加之后的新序列。从图中可知，对于加密算法的第 1-9 轮，平均每轮减少 2 个计算周期；对于加密算法的第 10 轮，平均减少 1 个计算周期。特别注意到，在进行操作叠加的过程中，虽然将上一轮函数的异或操作与下一轮函数的查找操作进行叠加，打破了原有算法的轮函数划分，但是并没有影响数值计算的正确性和数据计算的依赖关系，不会对计算结果造成错误影响。

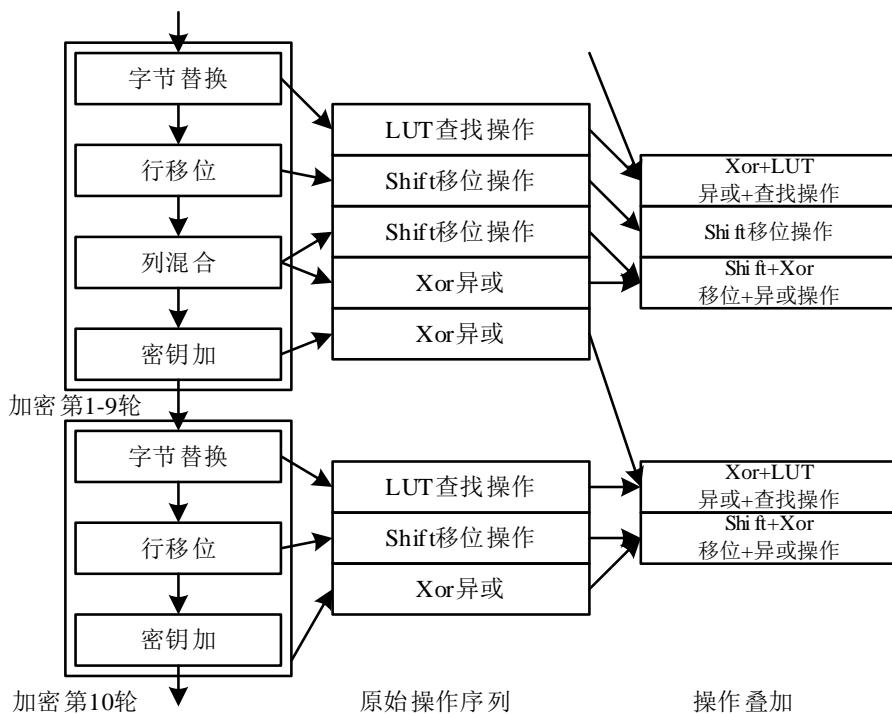


图 5-3 AES 算法算子级操作映射过程

进一步采用上述算子级操作序列叠加方案进行可重构计算单元结构设计，对典型分组密码算法

集进行映射验证，算法操作序列叠加之后所需周期数如表 5-2 所示。由该表可见，密码算法所需执行性能平均提高 59.95%，最高提升 224.00%。特别注意到，RC6 算法中没有使用异或操作，无法收受益于操作序列叠加的效果，因而执行性能没有得到提高。

表 5-2 算子叠加操作对不同分组密码算法的操作周期影响

算法	轮数	分组位宽	原始操作序列周期	操作序列叠加周期
AES	10	128	39	29
Blowfish	16	64	97	49
Camellia	18	128	120	80
CAST128	16	64	112	90
DES	16	64	162	50
GOST	32	64	128	64
KASUMI	6	64	138	72
RC5	12	64	72	48
SEED	16	128	288	160
Twofish	16	128	146	130
RC6	20	128	142	142
IDEA	9	64	54	54
SM4	32	128	256	160
Serpent	32	128	253	221

## 5.2.2 可重构寄存器文件设计

根据第三章全局寄存器文件的研究成果，本文采用分组全互联的分布式全局寄存器文件的实现方式。全局寄存器文件为重构阵列行设计了多个独立的读端口，同时每个提供两个独立的写通道，多个写端口可以通过选择仲裁对全局寄存器文件进行写操作。40 行可重构阵列按照每 2 行设计 1 个读端口设计，每个寄存器体包含 20 个读端口，2 个写端口。图 5-4 给出了寄存器文件架构的设计细节，整个全局寄存器文件包含 4 个  $7 \times 32\text{bit}$  的寄存器体。

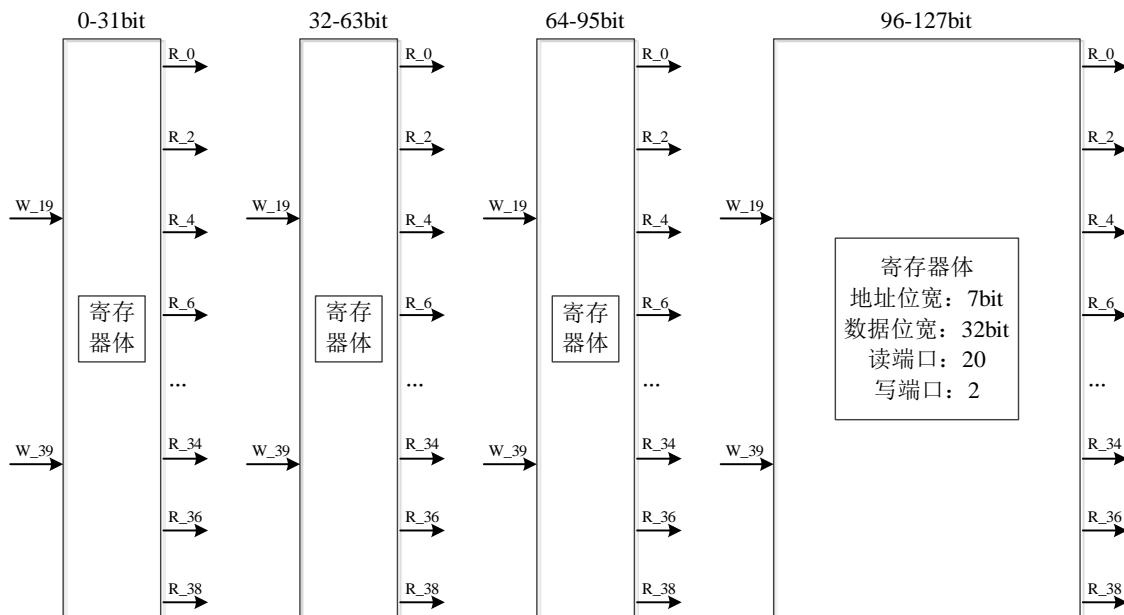


图 5-4 全局寄存器文件架构设计

根据第四章对局部寄存器文件的研究成果，本文针对目标算法集合 S-box 特点，采用的多端口统一结构的跨域寄存器文件设计。整个跨域寄存器文件包含 4 个寄存器文件簇。如图 5-5 所示，每个寄存器文件簇包含 32 个读取端口和一个写入端口。每行可重构计算单元均分别有 4 个输入和输出端口与跨域寄存器簇相连，同时 2bit 的重构配置信息用于适配不同的密码算法。为了满足不同算法对不同输入、输出位宽和容量的需求，每个寄存器文件簇包含 4 个 8 入 8 出的寄存器体，同时分别包含 32 个输入控制单元和输出控制单元。每行重构单元的输入信号经过输入控制单元的处理后分别作为寻址地址输入寄存器体，寄存器体的寻址结果经过输出控制单元的处理后返回重构单元行。

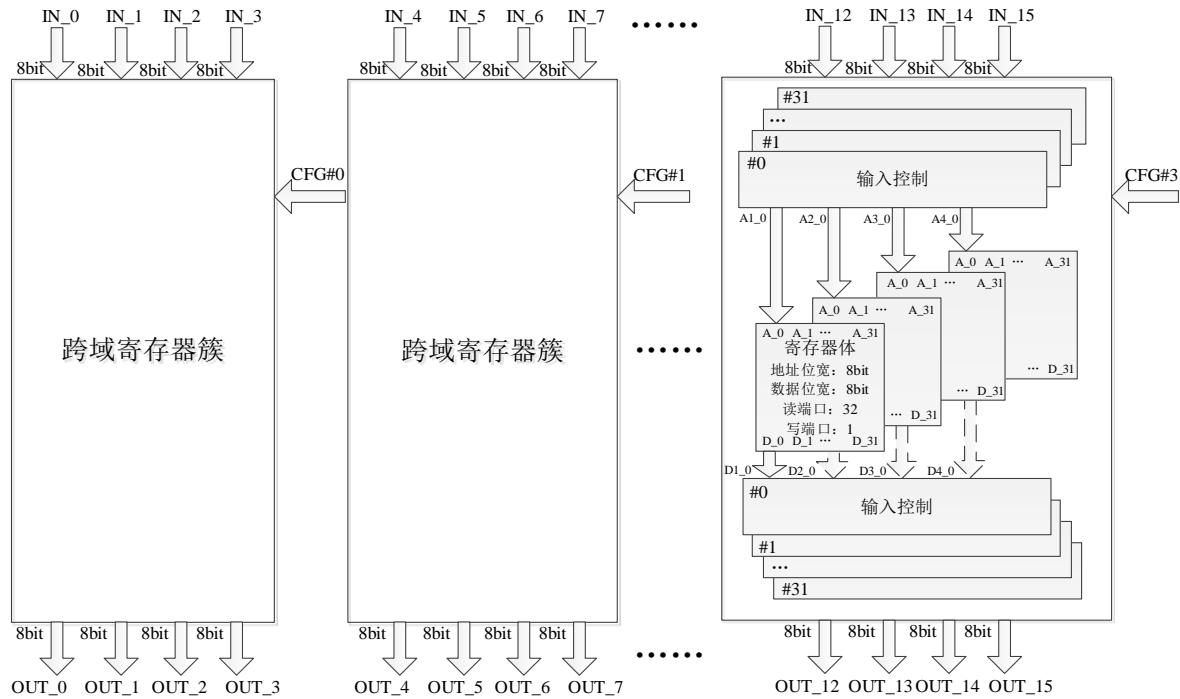


图 5-5 跨域寄存器文件架构设计

### 5.2.3 可重构阵列互联单元

本文采用静态互联的拓扑结构，每个 RC 利用硬连线或寄存器文件与周围临近的 RC 实现互联。可重构阵列行单元中的互联单元主要由两个部分组成：行输入多路选择器和 Benes 网络。如图 5-6 所示是所用互联单元的结构图。

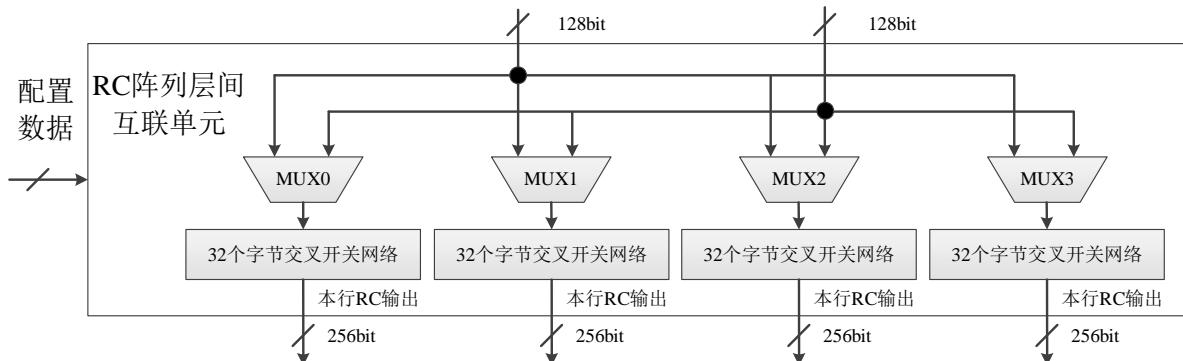


图 5-6 互联单元结构图

#### (1) 行输入多路选择器

各行输入两个 128 比特数据：IN0、IN1；输出两个 128 比特数据：OUT0、OUT1。IN0 和 IN1

分别连接到四个 128 位二选一的多路选择器 (multiplexer): Smux0、Smux1、Smux2 和 Smux3。

## (2) Benes 网络

在行输入 MUX 的输出和 ALU 的输入之间用 Benes 网络实现任意无重复置换模式，每个行输入多路选择器对应一个 Benes 网络，三个组合的 128 位操作数可以通过输入多路选择器选取来自任何一个 Benes 网络的输出。Benes 网络有两种类型：Benes16 和 Benes128。Benes16 实现一个 128 比特数据中的连续 16 个字节的置换，Benes128 实现一个 128 比特数据中的 128 个比特的置换。图 5-7 给出了 Benes128 的结构。在一列中的四个 Benes 的类型从左至右依次为：Benes16、Benes128、Benes128 和 Benes16。

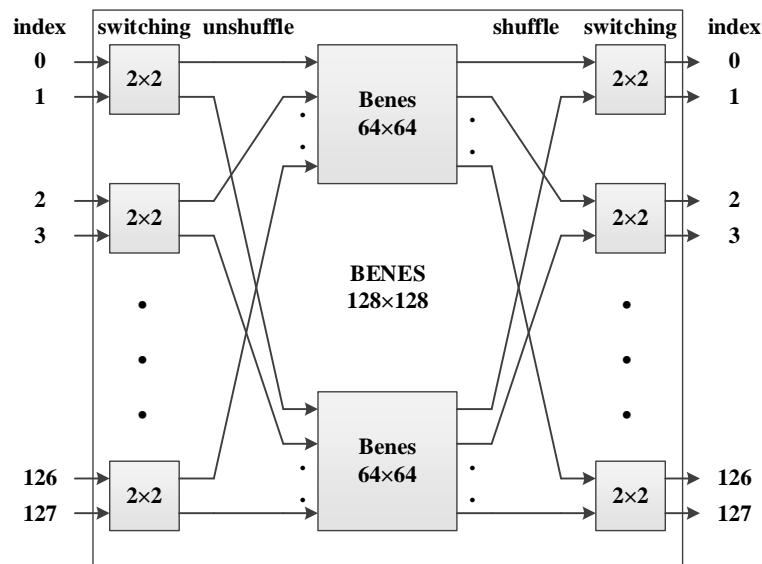


图 5-7 Benes128 网络的内部结构

通过行输入多路选择器和 Benes 网络的组合，各个操作数可以来自两个行输入 IN0 和 IN1 中的任何一个，并且可以在输入 ALU 时完成字节级和比特级的置换。

$N \times N$  的 Benes 需要  $N \log_2 N - N/2$  个  $2 \times 2$  开关。因此，Benes16 需要 56 位配置码，Benes128 需要 832 位配置码。

### 5.2.4 可重构密码阵列结构设计

可重构计算阵列引擎的基本单元是行，每行由 16 个 8 位算数逻辑单元 ALU(Arithmetic and Logic Unit) 组成。每一行的输出和下一行的输入模块直接相连。每一行都可以从 FIFO 或者全局寄存器文件读入数据，并且都可以将数据读出到 FIFO 或者全局寄存器文件中。除首行外，每行均可选择由上一行输出作为输入；除尾行外，每行输出均可作为下一行输入。

可重构阵列为 40 行，行结构如下图 5-8 所示，每一行阵列有两个输入：输入 0 与输入 1。通过 MUX 选择，分别输入 4 个 Benes 网络（2 个 128bit 置换，2 个 16 字节置换）。Benes 网络的置换结果输出至对应 ALU 的 IN0-IN2 端口。ALU 的输入端口通过 4 位多路选择器选择一个 Benes 网络的置换结果作为输入，并接收外界控制信号对数据进行运算，运算结果通过 OUT\_0、OUT\_1 端口输出。在与 LUT 查表信息进行选择后，成为本行的输出 0、输出 1。

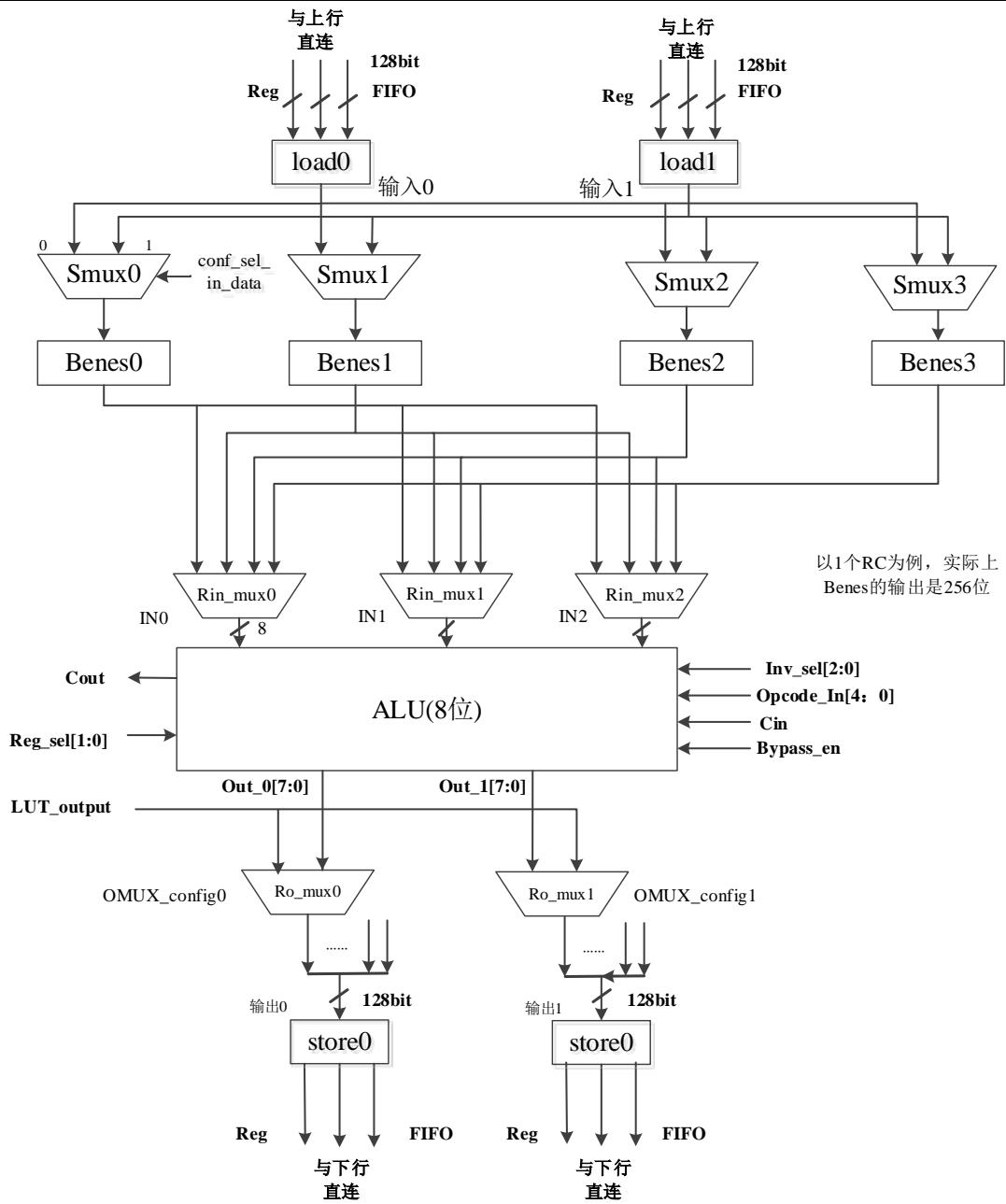


图 5-8 计算引擎中可重构阵列行单元结构

可重构阵列行单元包含 16 个 ALU，每个 ALU 有 3 个输入端口，2 个输出端口，且都是 8 位数据位宽的。ALU 接受 Config\_in 的配置，实现计算，输出数据 Out\_0。Out\_1 的结果可以是 In\_3 的旁路输出或者和 Out\_0 相同，由配置内容决定。与此同时，ALU 可以通过 Cin 和 Cout 的级联，4 个 ALU 级联可以实现 32 位加法器。

可重构计算引擎中包括可重构计算阵列、全局寄存器文件和局部寄存器文件等，如图 5-9 所示。可重构计算阵列由 40 行 128bit 的重构行组成，通过 128bit 位宽的数据通道与数据输入和输出接口相连。同时，包含本文研究的分组全互联分布式全局寄存器文件和多端口统一结构的跨域寄存器文件。

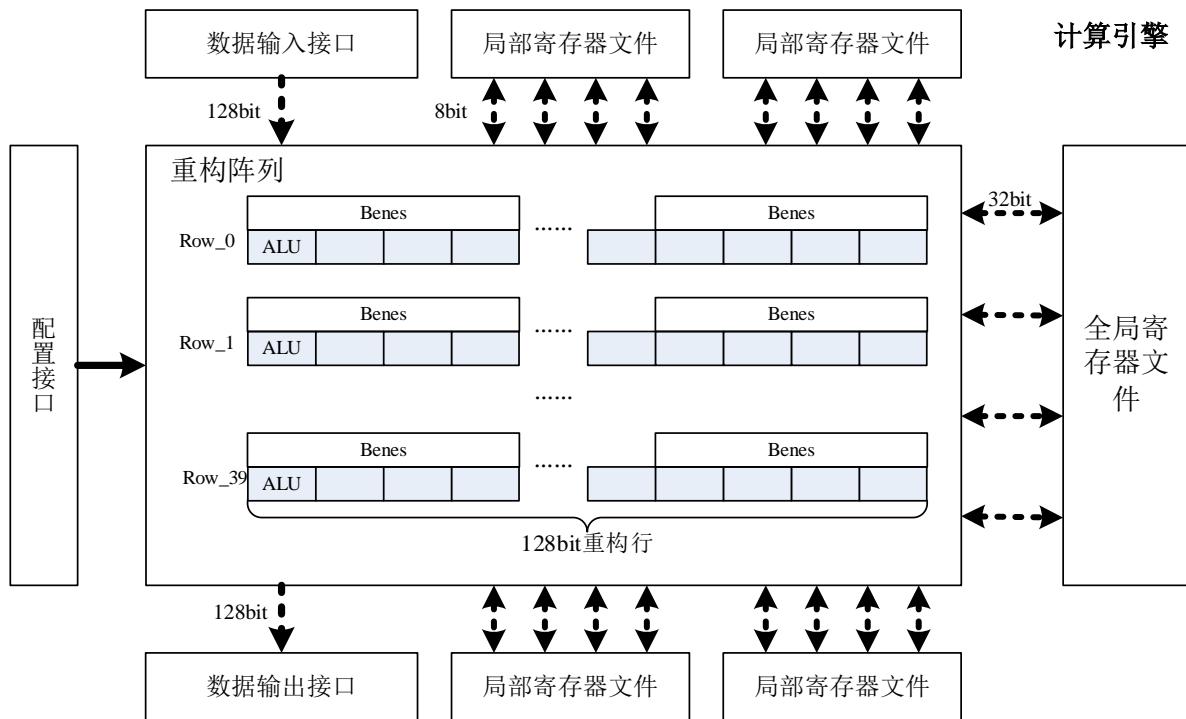


图 5-9 可重构计算引擎结构

### 5.2.5 可重构密码处理器结构设计

课题组基于混合寄存器文件的架构研究，设计开发面向分组密码算法的可重构密码处理器，包含针对处理器性能研究分析的全局寄存器文件参数和结构设计，以及针对 S-box 的可重构跨域寄存器文件的参数和结构设计。整个可重构密码处理器的设计原型如图 5-10 所示，PLL 单独为可重构处理单元（Reconfigurable Processing Unit, RPU）提供 650MHz 时钟以满足高性能的计算需求，处理器的其他模块采用 150MHz 时钟。

可重构处理单元（RPU）中包括可重构计算阵列（Reconfigurable Cell Array, RCA），数据流控制单元（Date Flow Control Unit, DFCU）和配置流管理单元（Context Flow Management Unit, CFMU）。其中，RCA 包含 640 个重构计算单元 RCs，DFCU 负责数据和信号的异步处理。此外，可重构密码处理器中还包括以下系统模块：

- 用作系统主控制器的 RISC 内核 ARM7TDMI。
- 用作系统外存访问接口的 EMI。
- 用作系统数据搬运的数据传输控制器 DMAC。
- 用作系统片上存储器 SPM\_0 和 SPM\_1，大小均为 64KB。每个 SPM 均有两个数据端口，其中 32bit 的数据端口与主处理器总线互联，128bit 的数据端口与可重构密码处理器连接。
- 用作系统中断控制器的 IntCtrl（中断控制器）。
- 用作系统状态控制器的 SysCtrl（系统控制器）。用于控制启动方式，地址映射空间等架构参数。
- 用作系统互连总线的 AMBA2AHB Matrix，外设总线桥接接口 APB Bridge（外设总线）等模块。
- 用作系统数据传输的串行通信接口 UART\_0 和 UART\_1。

- 用作系统串行数据传输的接口控制器 SDIO\_0 和 SDIO\_1。可以用于存储程序启动加载程序和存储非易失数据，或外接输入/输出设备。
- 用作系统计时器的 Timer（计数器）。
- 用作生成系统时钟的内部锁相环 PLL。

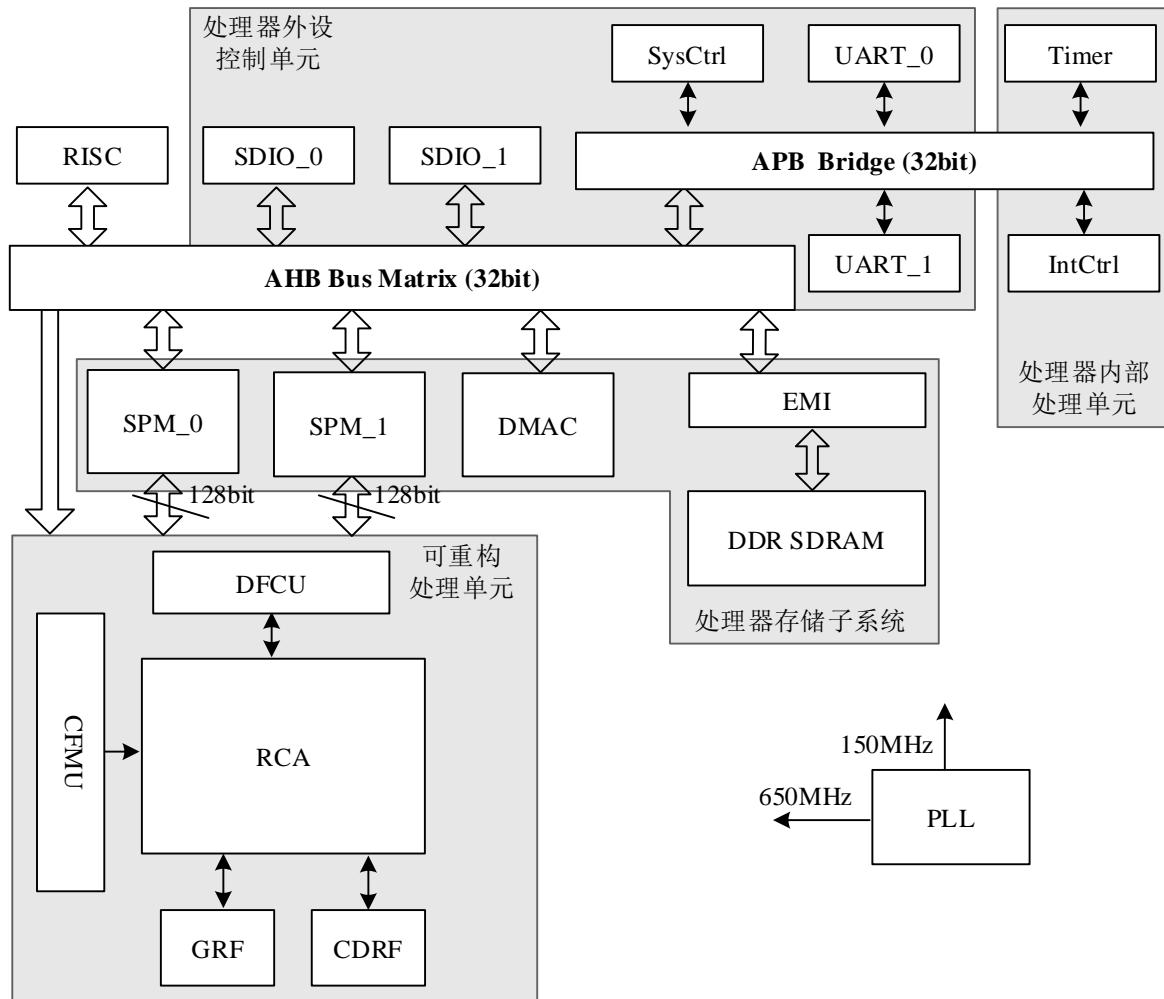


图 5-10 可重构处理器系统 SoC 架构示意图

## 5.3 可重构密码处理器验证及物理设计

### 5.3.1 可重构密码处理器的 FPGA 验证

可重构密码处理器的设计原型利用 FPGA 开发平台进行功能和性能的验证。课题组采用 S2C 公司的 Dual Virtex-7 TAI LM Hardware 高级开发平台，包含两颗 Xilinx Virtex-7 2000T FPGA 芯片颗粒。其中，Xilinx Virtex-7 FPGA XC7V2000T 芯片颗粒包含高达 305300 Slices、1954560 查找表单元、2443200 寄存器资源、46512Kbit 的存储资源、2160 个 DSP48E1 单元和最多 576 对差分 I/Os 资源。Dual Virtex-7 TAI LM Hardware 开发平台可以实现高达 40M ASIC 逻辑门的容量，包含高达 90Mbits 的 FPGA 内部存储资源和高达 4320 的内部 DSP48E1 单元。同时开发板搭载 DDR2 和 DDR3 扩展槽，而且可以通过板级扩展接口连接开发子板。

可重构密码处理系统的 FPGA 设计原型如图 5-11 所示。整个可重构密码处理器利用 Xilinx

Virtex-7 2000T FPGA 芯片中提供的资源进行实现，采用 I/Os 资源实现外部资源的扩展，包括 JTAG 接口、UART 接口、SD 存储卡、SDRAM 存储器等。从图中可以看到，整个可重构密码处理系统的 FPGA 原型除了利用开发板自身的外设，同时采扩展板进行外部接口扩展和外设连接。SysCtrl 利用板载拨码开关实现。SDRAM 存储器利用 SO-DIMM 扩展槽位的 I/Os 资源实现单独的扩展。UART\_0 和 JTAG 采用位于图片右方的扩展板实现。



图 5-11 可重构密码处理器系统的 FPGA 开发原型

本文着重考虑可重构密码处理单元的资源利用情况。可重构密码处理单元主要使用查找表单元和寄存器资源，分别占用 FPGA 芯片资源的 61.50% 和 7.35%，同时占用 FPGA 芯片约 0.23% 的存储资源。

表 5-3 中给出可重构密码处理器单元内部模块的资源自用情况。从表中可以看出，配置存储器和配置包头存储器主要采用存储资源实现，分别采用 2 块和 1 块单个容量为 36Kbit 的 RAM 块实现。可重构计算阵列占用 40.60% 的查找表资源和 78.06% 的寄存器资源，是整个可重构密码处理单元中资源使用量最高的模块。跨域寄存器文件和全局寄存器文件分别占用 39.19% 和 17.92% 的查找表资源，以及 11.13% 和 9.13% 的寄存器资源。

表 5-3 可重构密码处理单元内部模块的资源利用情况

模块名称	查找表单元 (LUTs)	寄存器资源 (REGISTER)	存储资源 (RAMs)
配置存储器	0	129	2
配置包头存储器	51	33	1
配置解析解析电路	17067	2808	0
可重构计算阵列	305016	140096	0
跨域寄存器文件	294440	19968	0
全局寄存器文件	134678	16384	0
其他	93	48	0
总计	751345	179466	3

### 5.3.2 可重构密码处理器的物理设计

整个可重构密码处理器在寄存器传输级 (Register Transform Level, RTL) 利用 TSMC 40nm 1P8M LP 低功耗工艺库, 采用 Verilog HDL 描述语言实现, 利用 Synopsys Verilog Complier Simulator (VCS) 工具进行功能和时序仿真。硬件实现基于 TSMC 40LP 1P8M CMOS 工艺库实现, 利用 Synopsys Design Compiler (DC) 工具进行综合生成网表, 并获得面积、时序等参数。通过 IC Compiler (ICC) 工具实现布局布线 (Place and Route) 工作后获得可重构密码处理器的 GDS II 版图文件。

可重构密码处理器的版图如图 5-12 所示, 处理器按照 650MHz 时序约束进行综合设计和静态时序调整, 按照典型室温环境下工作电压 1.2V 条件进行性能和面积评估。整个处理器芯片尺寸 (包含管脚) 为  $3003\mu\text{m} \times 3003\mu\text{m}$ , 面积为  $9.02\text{ mm}^2$ 。除掉管脚之后处理器资源的尺寸为  $2683\mu\text{m} \times 2683\mu\text{m}$ , 面积为  $7.20\text{ mm}^2$ , 其中可重构处理单元面积为  $3.70\text{ mm}^2$ 。

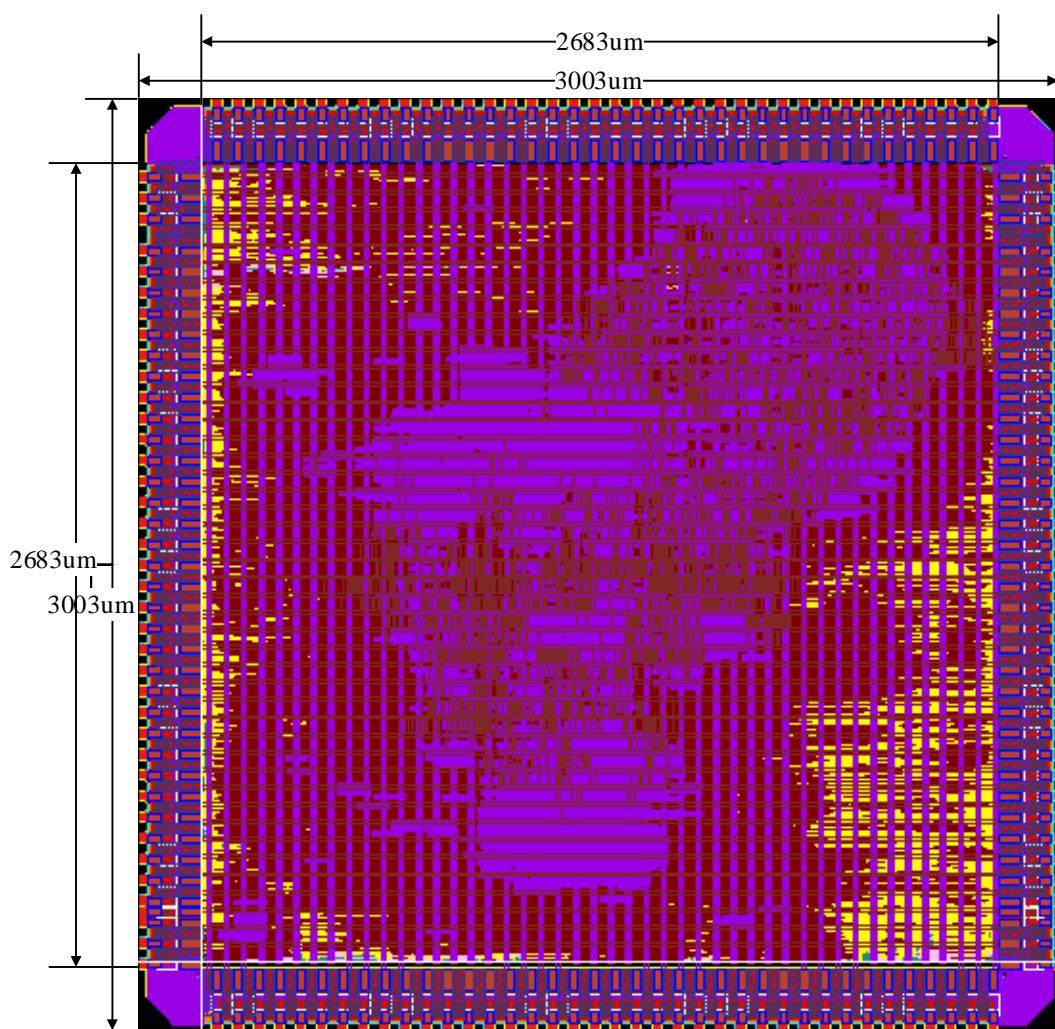


图 5-12 可重构密码处理器版图

表 5-4 给出了可重构密码处理器中重构处理单元主要模块的面积和面积百分比情况。从表中可以看出可重构计算阵列占据整个可重构面积的 43.02%, 配置流控制器包含的 CM、CPM 和配置解析电路占据的面积为 15.19%。混合寄存器文件中包括的全局寄存器文件和跨域寄存器文件占据的面积分别为 10.28% 和 31.09%。其中, 模块之间的部分胶合逻辑约占整个面积的 0.42%。

表 5-4 可重构密码处理器的面积和资源开销

模块名称	模块面积/ $\mu\text{m}^2$	面积百分比
配置存储器	300041.83	8.12%
配置包头存储器	128020.43	3.46%
配置解析解析电路	133568.78	3.61%
可重构计算阵列	1590460.65	43.02%
跨域寄存器文件	1149365.72	31.09%
全局寄存器文件	380100.28	10.28%
其他	15527.76	0.42%
总计	3697085.45	100.00%

## 5.4 可重构混合寄存器文件架构的对比与分析

混合寄存器文件架构的对比实验中，重点比较不同寄存器文件设计方案对可重构密码处理器性能和面积的影响。基于课题组实现的可重构密码处理器原型，分别实现 Cryptorapter 架构的寄存器文件系统和本文提出的混合寄存器文件系统。Cryptorapter 可重构密码处理器仅完成 10 种分组密码算法实验，本文特别考虑国产 SM4 密码算法，基于 14 种目标密码算法集合在可重构密码处理器原型上的实现，采用相同的算法轮数和分组数据位宽。为了突出混合寄存器文件对性能和面积的影响，密码算法在阵列上展开的周期数以及重构计算单元支持的算子都保持一致。

本文采用的寄存器文件的设计参数如下，全局寄存器文件采用 4 个  $128 \times 32\text{bit}$  容量，20 个读端口，2 个写端口的架构参数；用于实现 S-box 的跨域寄存器文件采用 16 个  $8 \times 8\text{bit}$  容量，32 个读端口，1 个写端口的架构参数。

在 Cryptorapter 设计方案中，全局寄存器文件采用 1 个  $256 \times 32\text{bit}$  容量，80 个读端口，1 个写端口的架构参数；在 S-box 的实现中，由于没有 Memory Compiler，文献[33]采用 CACTI 6.5 存储模型<sup>[79]</sup>估计出 SRAM 的面积。本文采用 Artisan by ARM 公司 40nm 低功耗 Memory 设计开发工具，由于 Memory Compiler 无法产生容量小于  $2048 \times 32\text{bit}$  的存储器，故采用 Register Compiler 生成单端口高密度的寄存器文件。在 40nm LL 条件下，单个 Bit Cell 的面积为  $0.299\mu\text{m}^2$ 。设计约束条件为 1GHz 的主频，SS 工艺（PMOS Slow, NMOS Slow），电压 0.99V，温度 125°度。

### 5.4.1 混合寄存器文件的性能对比

表 5-5 中给出在不同寄存器文件架构条件下，课题组实现可重构密码处理器的性能比较，对于 Cryptorapter 架构中提出的全局寄存器文件和 S-box 设计，可重构密码处理器平均 BPC 从 0.48 提高到 0.58，获得平均 28.62% 的 BPC 性能提高。

每个算法的性能提高率可以从图 5-13 中看到，其中 AES 算法由于可重构阵列能够提高足够的计算资源不需要配置切换，所以寄存器文件的优化不能体现出效果。对于其他算法均能获得 13.52% 到 100.00% 的性能提升，特别注意到对于算法 SEED, KASUMI 和 DES 算法分别取得 58.88%、100% 和 57.77%，这是由于本文设计的 S-box 能够提供更高的算法并发度。

表 5-5 不同寄存器文件方案的性能对比

算法	轮数	分组位宽	执行周期	算法并行度	BPC	
					Cryptorapter 架构	本文架构
AES <sup>[3]</sup>	10	128	29	1	1.00	1.00
Blowfish <sup>[80]</sup>	16	64	49	4	1.01	1.15
Camellia <sup>[60]</sup>	18	128	80	2	0.36	0.42
CAST-128 <sup>[81]</sup>	16	64	90	4	0.60	0.68
DES <sup>[56]</sup>	16	64	50	2	0.50	0.79
GOST <sup>[82]</sup>	32	64	64	4	0.76	0.86
KASUMI <sup>[83]</sup>	6	64	72	1	0.19	0.37
RC5 <sup>[59]</sup>	12	64	48	4	1.02	1.15
SEED <sup>[84]</sup>	16	128	176	1	0.08	0.13
Twofish <sup>[61]</sup>	16	128	130	2	0.21	0.24
RC6 <sup>[85]</sup>	20	128	142	1	0.13	0.16
IDEA <sup>[5]</sup>	9	64	54	2	0.65	0.79
SM4 <sup>[86]</sup>	32	128	160	1	0.14	0.18
Serpent <sup>[62]</sup>	32	128	221	1	0.10	0.12

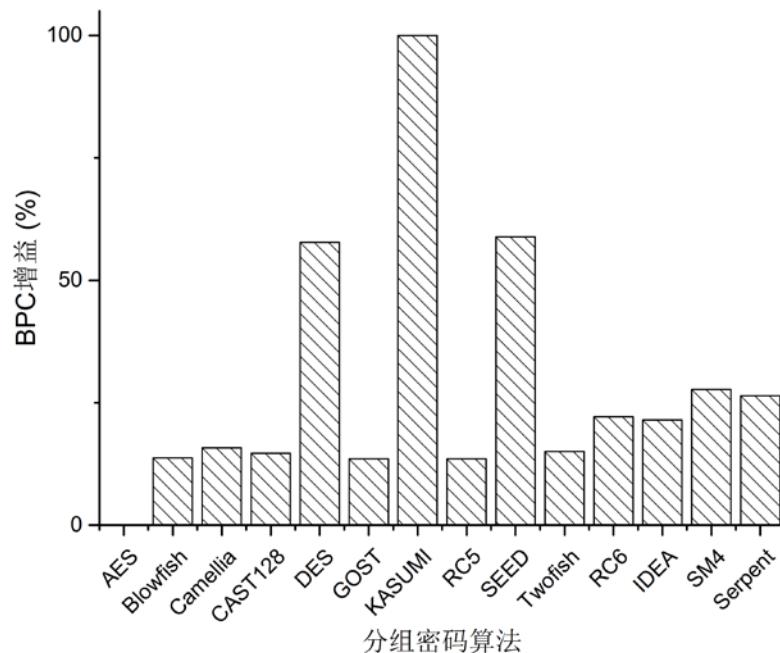


图 5-13 本文寄存器文件架构的算法性能增益

#### 5.4.2 混合寄存器文件的面积对比

表 5-6 给出本文采用的寄存器文件架构面积与传统设计方法的比较。从图中可以看到，本文采用的设计方法可以有效的减少面积资源开销。

图 5-14 中给出对比方案中不同模块在整个设计中的面积百分比占用情况，设计中除去混合寄存器文件架构所占用的面积，包括配置存储器、配置控制器、可重构计算阵列、数据流控制器在内的其他模块面积为  $2.17\text{mm}^2$ 。采用传统的设计方法密码处理器的总面积达到  $4.75\text{mm}^2$ ，采用本文的设计方法密码处理器的总面积达到  $3.70\text{mm}^2$ ，寄存器文件架构的面积较传统方法减少 22.18%。

表 5-6 不同寄存器文件方案的面积对比

	全局寄存器文件		局部寄存器文件	
	Cryptorapter 架构	本文架构	Cryptorapter 架构	本文架构
面积 ( $\text{mm}^2$ )	0.66	0.38	1.92	1.15

从图 5-14 中可以看到，采用本文设计的方法可以有效减少全局寄存器和跨域寄存器的面积开销。全局寄存器文件从 13.95% 减少到 10.28%，跨域寄存器文件从 40.42% 减少到 31.09%。

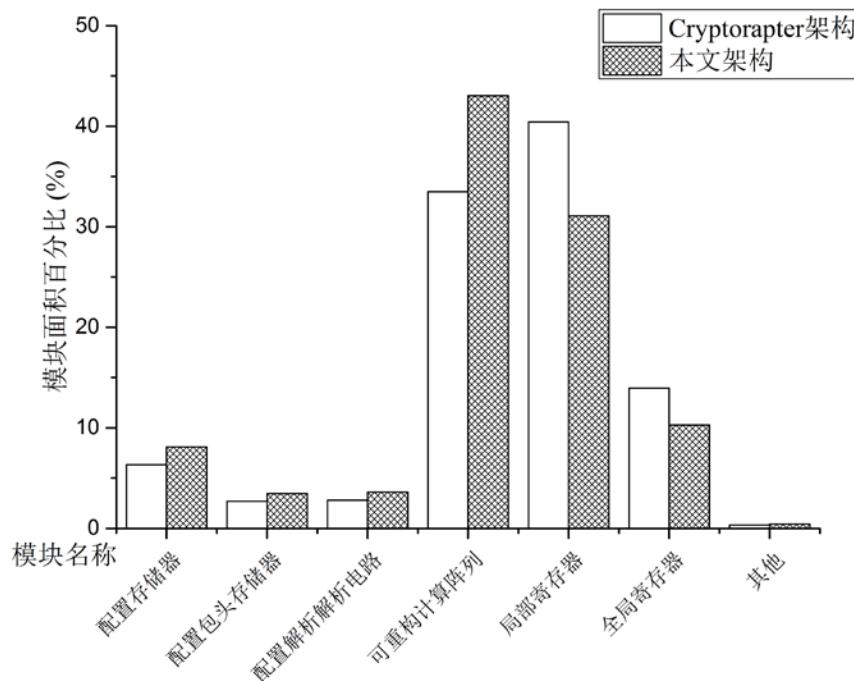


图 5-14 不同模块在可重构密码处理器中面积百分比占用情况

#### 5.4.3 混合寄存器文件的面积效率对比

表 5-7 不同寄存器文件方案的性能面积对比

算法	轮数	分组位宽	性能 (BPC)		寄存器面积 ( $\text{mm}^2$ )		面积效率 (BPCPA)	
			Cryptorapter 架构	本文架构	Cryptorapter 架构	本文架构	Cryptorapter 架构	本文架构
AES	10	128	1.00	1.00			0.39	0.65
Blowfish	16	64	1.01	1.15			0.39	0.75
Camellia	18	128	0.36	0.42			0.14	0.28
CAST128	16	64	0.60	0.68			0.23	0.45
DES	16	64	0.50	0.79			0.20	0.52
GOST	32	64	0.76	0.86			0.29	0.57
KASUMI	6	64	0.19	0.37			0.07	0.24
RC5	12	64	1.02	1.15	2.58 1.53		0.39	0.75
SEED	16	128	0.08	0.13			0.03	0.08
Twofish	16	128	0.21	0.24			0.08	0.16
RC6	20	128	0.13	0.16			0.05	0.10
IDEA	9	64	0.65	0.79			0.25	0.51
SM4	32	128	0.14	0.18			0.05	0.12
Serpent	32	128	0.10	0.12			0.04	0.08

表 5-7 给出了单位寄存器文件的面积效率，其中传统的寄存器文件中包括集中式的多端口寄存器文件和 Register Compiler 工具生成的高密度寄存器文件。从表中可知，相比传统设计方法实现平均 0.19BPCPA 的面积效率，本文提出的混合寄存器文件架构设计方法可以达到 0.39BPCPA 的面积效率。从图 5-15 中可知，面积效率平均获得 117.21% 的提高。其中，AES 算法由于计算性能没有变化，收益于面积减少使得面积效率提升 68.88%。KASUMI 算法同时收益于计算性能和寄存器面积的减少，获得最大面积效率提升 237.77%。

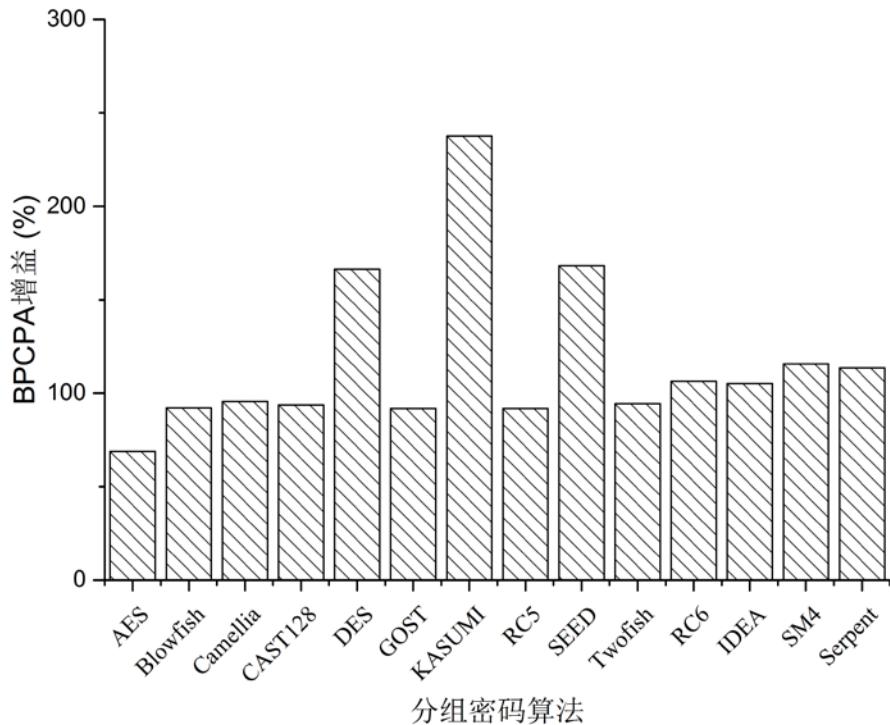


图 5-15 混合寄存器文件面积效率增益

## 5.5 可重构密码处理器的对比分析

### 5.5.1 多种算法在可重构密码处理器的实验结果对比

不同的可重构密码处理器架构在处理分组密码算法时，在算法的执行周期、计算并行度上都会有差异，特别是不同可重构处理器在架构以及工艺实现上的差异更会导致面积存在较大的差异。对于不同密码算法，可重构密码处理器的面积效率越高意味着在有限的面积下算法的性能越高。本文对比了 10 种主流分组密码算法在不同的处理平台运行给出性能、面积以及面积效率的差异。

表 5-8 给出实验中算法采用的轮数和分组位宽，同时两种架构均实现相同的计算并行度。从表中可以看出，算法在不同架构上展开后有不同的执行周期，这是由于不同的算子叠加策略导致的差异。虽然架构的 40 行阵列提供更多的计算操作，但是本文并没有获得较对比架构更大的计算性能。这是由于对比架构给出了阵列能够提供的理论最大计算性能，并没有考虑到多套配置切换过程中气泡间隔、加载和排空时间的影响，以及由于全局寄存器容量限制而导致的配置切换开销增加。对比算法 DES、SEED 和 KASUMI 的计算性能提高得益于跨域寄存器并发端口数增加。算法 GOST 的性能收益则来源于执行周期减少而带来的循环迭代间隔减少。

表 5-8 不同的处理平台的分组密码算法性能面积对比

算法	轮数	分组位宽	算法并行度	执行周期		性能 (BPC)		处理器面积 (mm <sup>2</sup> )		面积效率 (BPCPA)	
				Cryptorapte r	本文设计	Cryptorapte r	本文设计	Cryptorapte r	本文设计	Cryptorapte r	本文设计
AES <sup>[3]</sup>	10	128	1	20	29	1.00	1.00	6.32	3.70	0.16	0.27
Blowfish <sup>[80]</sup>	16	64	4	48	49	1.33	1.15			0.21	0.31
Camellia <sup>[60]</sup>	18	128	2	80	80	0.50	0.42			0.08	0.11
CAST-128 <sup>[81]</sup>	16	64	4	80	90	1.00	0.68			0.16	0.18
DES <sup>[56]</sup>	16	64	2	48	50	0.67	0.79			0.11	0.22
GOST <sup>[82]</sup>	32	64	4	96	64	0.67	0.86			0.11	0.23
KASUMI <sup>[83]</sup>	6	64	1	64	72	0.25	0.37			0.04	0.10
RC5 <sup>[59]</sup>	12	64	4	48	48	1.33	1.15			0.21	0.31
SEED <sup>[84]</sup>	16	128	1	160	176	0.13	0.13			0.02	0.03
Twofish <sup>[61]</sup>	16	128	2	80	130	0.50	0.24			0.08	0.07

受益于可重构密码处理器架构面积的优势，架构的面积效率 BPCPA 较对比架构的平均 0.12 提高到 0.18，面积效率平均提高 66.56%。其中，KASUMI 算法获得最高 154.37% 的提高，这是由于算法的计算性能提高同时面积减少的结果。同时注意到 Twofish 算法中，面积效率降低 16.97%，这是由于算法的执行周期太多导致计算性能仅有对比架构的一半。

### 5.5.2 与多种计算平台的实验结果对比

对于分组密码算法在多种计算平台的实现而言，密码处理器的面积效率越高意味着单位面积下算法的性能越高。本文对比了不同计算平台处理器在采用不同工艺实现 AES 和 DES 算法时的面积、性能，并且通过面积效率对比不同平台实现结果的优劣。如表 5-9 所示，这些平台包括 CGRA、ASIC、FPGA 和 CUDA 的实现方式。本文采用 40nm 工艺实现可重构处理器架构，最高主频达到 650MHz，整芯片面积为 9.02 mm<sup>2</sup>，其中核心的可重构处理单元面积为 3.70mm<sup>2</sup>。此处，特别注意到不同计算平台的硬件资源开销统计采用不同单位，文献中 CGRA 计算平台按照面积进行统计，ASIC 计算平台按照等效门数进行统计，FPGA 计算平台则给出布局布线之后占用的 Slice 资源数。

通过对比参考文献中不同计算平台实现 DES 和 AES 算法的性能可知：

对比 CGRA 计算平台：DES 和 AES 算法的吞吐率除了小于 Cryptoraptor 架构，相比其他可重构密码处理器的性能均有大幅度提高。本文利用混合寄存器文件设计带来的面积优势，使得整个可重构密码处理器的面积仅为 Cryptoraptor 的 58.54%，最终获得 DES 和 AES 算法的面积效率的提高。相比现有的可重构密码处理器，DES 算法的性能面积比至少提高 40.68%，AES 算法的性能面积比至少提高 10.62%。

对比 FPGA 计算平台：本文实现的 DES 和 AES 算法的性能均优于比较对象，在折换成面积效率之后，实验结果仍然具有一定的优势。

对比 ASIC 计算平台：文献[76]中 AES 算法的实现数据折算成 45nm 工艺下的面积效率约为 500Gbps/mm<sup>2</sup>，面积效率高于本文实现结果。由于 ASIC 实现针对 AES 算法做了定制优化，导致芯片实现面积很小，而本文设计的可重构处理器面向所有分组密码算法，灵活性远高于 ASIC 的实现。

方式，而且配置信息未加载到片上的时候能够满足某些特殊场合的白片需求。

综上所述，本文的实现结果面积效率方面优于同类型可重构平台，本文实现结果和 FPGA 结果具有一定优势，和 ASIC 相比较具有可比性，但是面积方面仍需要进一步优化。

表 5-9 不同计算平台处理器的分组密码算法的性能面积对比

平台	架构	算法	工艺 /nm	主频 /MHz	面积	吞吐率/Gbps	面积效率 (BPCPA)
CGRA	本文	DES	40	650	3.70*	30.13	9.50*
		AES				82.88	22.40*
	Cryptoraptor <sup>[33]</sup>	DES	45	1000	6.32*	42.70	6.75*
		AES				128	20.25*
	ProDFA <sup>[36]</sup>	DES	130	400	0.72*	4.38	6.09*
		AES				4.20	5.83*
	RCPA <sup>[87]</sup>	DES	180	180	14.9*	0.46	0.03*
		AES				0.79	0.05*
	博士论文 <sup>[88]</sup>	DES	180	250	1.44*	1.50	1.04*
		AES				3.60	2.50*
	CGRA <sup>[89]</sup>	AES	65	400	51.36	51.2	0.99
	CGRA <sup>[90]</sup>	AES	55	300	5.01**	0.44	0.088**
		DES				0.35	0.07**
	Celator <sup>[35]</sup>	DES	130	190	0.1*	0.03	0.25*
	CORBA <sup>[34]</sup>	AES	350	120	6.7**	1.45	0.22**
	DRP <sup>[91]</sup>		/	30.1	/	0.84	/
	DREAM <sup>[92]</sup>		180	200	/	0.546	/
	AESTHETIC <sup>[93]</sup>	AES	250	66	6.29*	0.84	0.13*
	CGRA <sup>[94]</sup>		/	149	/	5.71	/
	CGRA <sup>[95]</sup>		180	200	15.02*	0.79	0.05*
ASIC	ASIC <sup>[76]</sup>	AES	45	2100	0.15*	53.00	353.33
	ASIC <sup>[96]</sup>		180	100	21.93**	1.28	0.058**
	ASIC <sup>[97]</sup>		130	197.29	23.18**	2.52	0.11**
FPGA	XC4000 <sup>[98]</sup>	DES	180	225	240***	14.40	0.06***
	Virtex6 <sup>[99]</sup>	AES	40	319	9097***	40.90	0.004***
	Virtex5 <sup>[100]</sup>		65	92	1015***	0.95	0.0009***
	Virtex6 <sup>[101]</sup>		40	533	2751***	58.00	0.02***
	Virtex6 <sup>[97]</sup>		40	388.09	7582***	4.97	0.000655***
CUDA	CUDA <sup>[102]</sup>	AES	/	/	/	48.60	/

\*： 面积单位 mm<sup>2</sup>； 面积效率(Gbps/mm<sup>2</sup>)

\*\*： 门数单位 Mgate； 面积效率 (Gbps/Mgate)

\*\*\*： 门数单位 Slice； 面积效率 (Gbps/Slice)

## 5.6 本章小结

本章综合采用本文对可重构密码处理器的寄存器文件研究成果，将全局寄存器文件和跨域寄存器文件的混合层次化设计融入课题组设计实现的可重构处理器中。利用基于寄存器传输级 RTL 实现

的可重构 SoC 设计，通过 FPGA 原型验证设计的功能和性能，同时通过在 TSMC 40nm 工艺下物理实现获得密码处理器的频率和面积结果。本文利用主流分组密码算法作为实验测试集合，针对寄存器文件创新点对性能和面积的影响，对比分析不同的寄存器文件设计方案。同时，对比不同密码处理器实现方案的性能面积优缺点。

## 第六章 总结与展望

### 6.1 总结

信息安全问题作为信息系统设计的核心问题之一，是支持计算机技术和通信技术快速发展的基石。为了满足信息安全中高速处理差异化加密数据的需求，如何高效灵活的实现密码算法成为信息安全领域的重要研究方向。可重构密码处理器在提供大规模二维阵列计算资源满足密码计算需求的同时，通过配置信息动态重构的计算方式满足了密码算法差异化的实现需求。为了满足可重构密码处理器对计算数据带宽和访存效率的需求，探索高面积效率的密码处理器架构设计，本文选取混合寄存器文件中的全局寄存器和局部寄存器作为研究对象。

本文从分组密码算法特征分析入手，讨论分析影响可重构密码处理器计算性能的算法特点和数据特征；从计算性能分析出发，抽象密码算法在可重构密码处理器的计算过程，建立可重构密码处理器的性能解析模型；通过全面考虑密码算法特征和可重构计算架构约束，课题组基于本文研究的分组全互联分布式全局寄存器文件和多端口统一结构的跨域寄存器文件，实现面向分组密码算法的可重构密码处理器。

针对全局寄存器资源限制而造成可重构密码处理器的性能瓶颈，通过分析分组密码算法的轮函数次数，密码处理器的重构行数和重构过程的配置暂停时间等特征参数，提出架构和算法特征依赖的全局寄存器性能模型，并用于全局寄存器文件的设计参数优化。实验结果表明，在相同的算法集合和架构特征约束下，采用本文性能模型优化的全局寄存器文件，与 Cryptoraptor 设计采用的全局寄存器架构相比算法性能平均提高 17.24%；与 ADRES 设计采用的重排序全局寄存器文件架构相比，算法性能平均提高 230.67%。

针对由于存储容量和互联范围大而导致全局寄存器文件的面积资源开销巨大的问题，通过分析分组密码算法在轮函数之间相互独立，轮函数写后读的数据依赖特点，基于减少寄存器互联范围的策略，提出分组全互联的分布式全局寄存器文件结构，减少面积资源开销。实验结果表明，相同数据访问并发度的条件下，比 Cryptoraptor 设计实现的寄存器文件面积资源开销减少 41.92%。

针对局部寄存器在适配分组密码算法中 S-box 内容可变、结构多样的复杂需求，通过分析目标算法集合确定 S-box 的表数量、尺寸和输入输出位宽等约束，提炼满足分组密码算法 S-box 的最小局部寄存器容量和最少访存并发度，提出面向多种分组密码算法的多端口统一的跨域寄存器文件结构，有效减少局部寄存器文件面积开销。实验结果表明，在满足算法最大并行度的基础上，本文提出的局部寄存器文件结构与文献[33]相比，面积开销减少 26.14%。

为了验证本文研究的混合寄存器文件架构性能、面积和灵活性等参数指标，上述研究内容应用于课题组研发的一款可重构密码处理器中，目前已完成流片前的物理设计。整个可重构密码处理器采用 TSMC 40nm 1P8M LP 低功耗工艺库，最高工作主频可达 650MHz，芯片面积为  $9.02 \text{ mm}^2$ 。选取主流分组密码算法作为实验测试集合，能够实现最高 82.88Gbps 吞吐率，面积效率达到  $22.40 \text{ Gbps/mm}^2$  的性能指标。通过与其他密码处理器实现方案对比发现，可知基于本文所述混合寄存器文件设计和调度策略的可重构密码处理器，不仅具有高性能和灵活性的优势，而且能够获得优

---

异的性能面积比，分组密码算法 AES 的面积效率提高 10.62%，DES 的提高 40.48 %。

## 6.2 展望

本文针对可重构密码处理器的混合寄存器文件架构关键技术进行研究，特别对影响可重构处理器性能和面积的全局寄存器文件和局部寄存器文件进行了深入的探讨和研究，取得了一定的研究成果。但是，仍然还有许多研究工作需要进一步探索和完善：

- (1) 可重构计算阵列作为主要计算模块，是影响可重构密码处理器的性能和面积开销的关键问题。可重构计算单元一方面要满足并行高性能的计算需求，另一方面要在高性能面积比的约束下实现灵活的微结构设计。为了进一步提高可重构密码处理器的计算性能和面积效率，可以针对密码算法中不同运算操作的使用频度和依赖关系，优化计算单元的关键路径；针对不同密码算法的计算位宽差异，设计面积优化的混合粒度的重构计算单元。
- (2) 高效便捷的可重构编译工具开发是拓展可重构计算应用领域的亟待解决的关键问题。可重构计算架构特殊的阵列结构和计算流水使得传统的编译方法已经不能胜任应用需求，自动化、高性能、高稳定性的可重构计算编译技术研究成为迫切需求。编译工具可以利用抽象的高级编程语言直接获得适用于可重构执行的配置流，自动实现源码分析、软硬件划分、计算调度和配置信息生成等工作。此外，通过编译工具和架构设计的交互设计，既能指导可重构架构硬件系统的结构设计，又可以为编译流程提供高精度的架构约束描述。

## 致谢

光影荏苒，从东南大学百年校庆的时候踏进浦口校区开始本科生活，在经历了东南大学 112 周年纪念之后终于在四牌楼结束博士研究生生活。值此论文即将完成之际，谨向人生路上一起经历过这段难忘博士研究生生活的亲人、老师、朋友致以衷心的感谢！

首先，要感谢我的博士生导师时龙兴教授，是他在博士研究生的科研过程中不断教导我，帮我确定研究方向，引领我在集成电路研究领域进行探索。他严肃的科学态度、严谨的治学精神和渊博的专业知识，不仅让我由衷敬佩，而且也让我受益匪浅。在他的谆谆教导之下，使我对研究方向有了更深刻的理解，对研究内容都有了全新的认识。

其次，特别感谢东南大学国家专用集成电路系统工程技术研究中心的吴建辉老师、杨军老师、曹鹏老师、齐志老师，感谢他们在整个博士课题研究过程中给予的精心的指导和帮助。在整个课题的研究过程中，从问题提炼、现状分析、难点确定、方案设计和实验验证的每个步骤都进行全面而细致的讨论和指导，不仅指出研究中存在的不足和缺点，而且帮助深挖创新点的深度和意义，在此谨向各位老师致以诚挚的谢意。同时，感谢可重构密码处理器课题组成员杨锦江、胡建兵、李小泉、闵靖、申艾麟、李兆奇和尹玲在项目进行中的通力配合和相互帮助，以及清华大学的朱敏博士、郭岩松博士的大力支持，没有大家共同的努力，就不可能见证课题组成功的流片。

再次，特别感谢在博士研究生期间一起奋战在工程项目的各位老师、同学和同事，感谢杨军老师、刘新宁老师、凌明老师、戚隆宁老师、黄少珉老师，以及逝去的张哲老师给予的指导，你们的关心和帮助是我成长过程的幸运；感谢同事时建龙、黄丹丹、邵金梓、刘伟明和刘忻，大家在一起共事非常愉快；感谢这么多项目中陪伴一起通宵熬夜的师兄、师姐、同学、学弟和学妹，抱歉一直说着和你们一起毕业却还是落下了。

此外，特别感谢王镇、谢震、孙大鹰、武建平、刘波、肖建、张阳、梅晨、杨锦江、高谷刚等各位博士生同学，感谢大家在科研和生活中给予的真切关心，相信时光流逝、铅华洗尽会铭记那些一起度过的岁月。

最后感谢我的父母、妻子、姐姐和家人，是你们不离不弃的支持，默默地鼓励我的生活：快乐时分享每一刻的喜悦，挫折时一起面对每一个困难！感谢爸爸和妈妈在物质和精神上给予无私的深沉支持，很难想象在艰难养育我和姐姐的同时包容我的任性和放肆，养育之恩无以为报，你们健康快乐是我最大的心愿！感谢妻子一起陪伴走过的这段岁月，一起经历生活中的喜悦、感动、争吵和失落！感谢姐姐无声的支持和鼓励，手足情深无论相隔千里！



## 参考文献

- [1] Menezes A J, Van Oorschot P C, Vanstone S A, *Handbook of Applied Cryptography*[M], CRC press, 1996
- [2] Ravi S, Raghunathan A, Kocher P, et al. Security in Embedded Systems: Design Challenges[J].*ACM Transactions on Embedded Computing Systems (TECS)*. 2004, 3 (3): 461-491
- [3] Daemen J, Rijmen V. Aes Proposal: Rijndael[J].1998
- [4] Matsui M, The First Experimental Cryptanalysis of the Data Encryption Standard, in *Advances in Cryptology — Crypto '94*Springer Berlin Heidelberg, 1994, 1-11
- [5] Lai X, Massey J, A Proposal for a New Block Encryption Standard, in *Advances in Cryptology — Eurocrypt '90*Springer Berlin Heidelberg, 1991, 389-404
- [6] Weiwei S, Xin C, Bo L, et al. Evaluation of Correlation Power Analysis Resistance and Its Application on Asymmetric Mask Protected Data Encryption Standard Hardware[J].*Instrumentation and Measurement, IEEE Transactions on*. 2013, 62 (10): 2716-2724
- [7] Po-Chun L, Chang H-C, Chen-Yi L. A True Random-Based Differential Power Analysis Countermeasure Circuit for an Aes Engine[J].*Circuits and Systems II: Express Briefs, IEEE Transactions on*. 2012, 59 (2): 103-107
- [8] Jun W, Yiyu S, Minsu C. Measurement and Evaluation of Power Analysis Attacks on Asynchronous S-Box[J].*Instrumentation and Measurement, IEEE Transactions on*. 2012, 61 (10): 2765-2775
- [9] Paul R, Chakrabarti A, Ghosh R. Fault Detection for Rc4 Algorithm and Its Implementation on Fpga Platform[J].arXiv preprint arXiv:1401.2732. 2014,
- [10] Igarashi H, Youhua S, Yanagisawa M, et al. Concurrent Faulty Clock Detection for Crypto Circuits against Clock Glitch Based Dfa[C]. *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*.2013. 1432-1435
- [11] Xiaofei G, Karri R. Recomputing with Permuted Operands: A Concurrent Error Detection Approach[J].*Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2013, 32 (10): 1595-1608
- [12] King S T, Tucek J, Cozzie A, et al. Designing and Implementing Malicious Hardware[J].*LEET*. 2008, 8: 1-8
- [13] Lin L, Kasper M, Güneysu T, et al., Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering, in *Cryptographic Hardware and Embedded Systems - Ches 2009*Springer Berlin Heidelberg, 2009, 382-395
- [14] Jin Y, Makris Y. Hardware Trojans in Wireless Cryptographic Integrated Circuits[J].*Design & Test, IEEE*. 2013, PP (99): 1-1
- [15] Becker G, Regazzoni F, Paar C, et al., Stealthy Dopant-Level Hardware Trojans, in *Cryptographic Hardware and Embedded Systems - Ches 2013*Springer Berlin Heidelberg, 2013, 197-214
- [16] Estrin G. Organization of Computer Systems: The Fixed Plus Variable Structure Computer[C]. Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference.1960. 33-40
- [17] Compton K, Hauck S. Reconfigurable Computing: A Survey of Systems and Software[J].*ACM Comput. Surv.* 2002, 34 (2): 171-210
- [18] Hartenstein R, A Decade of Reconfigurable Computing: A Visionary Retrospective, in *Proceedings of the conference on Design, automation and test in Europe (Munich, Germany: IEEE Press, 2001)*, pp. 642-649
- [19] Todman T J, Constantinides G A, Wilton S J, et al. Reconfigurable Computing: Architectures and Design Methods[J].*IEE Proceedings-Computers and Digital Techniques*. 2005, 152 (2): 193-207

- [20] DeHon A, Reconfigurable Architectures for General-Purpose Computing, (Massachusetts Institute of Technology, 1996)
- [21] Compton K, Hauck S. Reconfigurable Computing: A Survey of Systems and Software[J].ACM Computing Surveys (csUR). 2002, 34 (2): 171-210
- [22] Amano H. A Survey on Dynamically Reconfigurable Processors[J].IEICE transactions on communications. 2006, 89 (12): 3179-3187
- [23] Ali L, Aris I, Hossain F S, et al. Design of an Ultra High Speed Aes Processor for Next Generation It Security[J].Computers & Electrical Engineering. 2011, 37 (6): 1160-1170
- [24] Dong C, Guochu S, Yihong H, et al. Efficient Architecture and Implementations of Aes[C]. Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on.2010. V6-295-V296-298
- [25] Hossain F S, Ali M L, Al Abedin Syed M A. A Very Low Power and High Throughput Aes Processor[C]. Computer and Information Technology (ICCIT), 2011 14th International Conference on.2011. 339-343
- [26] Bin L, Baas B M. Parallel Aes Encryption Engines for Many-Core Processor Arrays[J].Computers, IEEE Transactions on. 2013, 62 (3): 536-547
- [27] Akdemir K, Dixon M, Feghali W, et al. Breakthrough Aes Performance with Intel® Aes New Instructions[J].White paper, June. 2010,
- [28] Brown J D, Woodward S, Bass B M, et al. Ibm Power Edge of Network Processor: A Wire-Speed System on a Chip[J].IEEE Micro. 2011, (2): 76-85
- [29] Bertoni G M, Breveglieri L, Roberto F, et al. Speeding up Aes by Extending a 32 Bit Processor Instruction Set[C]. Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference on.2006. 275-282
- [30] Flynn M. Very High-Speed Computing Systems[J].Proceedings of the IEEE. 1966, 54 (12): 1901-1909
- [31] Singh H, Ming-Hau L, Guangming L, et al. Morphosys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications[J].Computers, IEEE Transactions on. 2000, 49 (5): 465-481
- [32] Mei B, Vernalde S, Verkest D, et al., Adres: An Architecture with Tightly Coupled Vliw Processor and Coarse-Grained Reconfigurable Matrix, in Field-Programmable Logic and ApplicationsSpringer Berlin / Heidelberg, 2003, 61-70
- [33] Sayilar G, Chiou D. Cryptoraptor: High Throughput Reconfigurable Cryptographic Processor[C]. Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on.2014. 155-161
- [34] Elbirt A J, Paar C. An Instruction-Level Distributed Processor for Symmetric-Key Cryptography[J].Parallel and Distributed Systems, IEEE Transactions on. 2005, 16 (5): 468-480
- [35] Fronte D, Perez A, Payrat E. Celator: A Multi-Algorithm Cryptographic Co-Processor[C]. Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on.2008. 438-443
- [36] Ming Y, Ziyu Y, Lei L, et al. Prodfa: Accelerating Domain Applications with a Coarse-Grained Runtime Reconfigurable Architecture[C]. Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on.2012. 834-839
- [37] Saluja D, Register File Organization for Coarse-Grained Reconfigurable Architectures: Compiler-Microarchitecture Perspective:[Master].Arizona State University. 2014
- [38] Hamzeh M, Shrivastava A, Vrudhula S, Regimap: Register-Aware Application Mapping on Coarse-Grained Reconfigurable Architectures (Cgras), in Proceedings of the 50th Annual Design Automation Conference (Austin, Texas: ACM, 2013), pp. 1-10
- [39] De Sutter B, Coene P, Aa T V, et al. Placement-and-Routing-Based Register Allocation for Coarse-Grained Reconfigurable Arrays[J].Acm Sigplan Notices. 2008, 43 (7): 151-160

- [40] Fontaine C, Galand F. A Survey of Homomorphic Encryption for Nonspecialists[J].EURASIP J. Inf. Secur. 2007, 2007: 1-15
- [41] Thomas E, A Survey of Lightweight-Cryptography Implementations, 12, 2007, pp. 522-533
- [42] Van Essen B, Panda R, Wood A, et al. Managing Short-Lived and Long-Lived Values in Coarse-Grained Reconfigurable Arrays[C]. Field Programmable Logic and Applications (FPL), 2010 International Conference on.2010. 380-387
- [43] Kim Y, Lee J, Shrivastava A, et al., Memory-Aware Application Mapping on Coarse-Grained Reconfigurable Arrays, in High Performance Embedded Architectures and CompilersSpringer, 2010, 171-185
- [44] 严明, 面向领域应用的异构多核 soc 系统结构设计与优化:[博士].2011
- [45] Parizi H, Niktash A, Bagherzadeh N, et al., Morphosys: A Coarse Grain Reconfigurable Architecture for Multimedia Applications, in Euro-Par 2002 Parallel ProcessingSpringer, 2002, 844-848
- [46] Mei B, De Sutter B, Vander Aa T, et al. Implementation of a Coarse-Grained Reconfigurable Media Processor for Avc Decoder[J].Journal of Signal Processing Systems. 2008, 51 (3): 225-243
- [47] Mei B, Veredas F-J, Masschelein B. Mapping an H. 264/Avc Decoder onto the Adres Reconfigurable Architecture[C]. Field Programmable Logic and Applications, 2005. International Conference on.2005. 622-625
- [48] De Sutter B, Allam O, Raghavan P, et al. An Efficient Memory Organization for High-Ilp Inner Modem Baseband Sdr Processors[J].Journal of Signal Processing Systems. 2010, 61 (2): 157-179
- [49] Mei B, Berekovic M, Mignolet J, Adres & Dresc: Architecture and Compiler for Coarse-Grain Reconfigurable Processors, in Fine-and Coarse-Grain Reconfigurable ComputingSpringer, 2008, 255-297
- [50] Kwok Z, Wilton S J. Register File Architecture Optimization in a Coarse-Grained Reconfigurable Architecture[C]. Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on.2005. 35-44
- [51] Garcia A, Berekovic M, Aa T V. Mapping of the Aes Cryptographic Algorithm on a Coarse-Grain Reconfigurable Array Processor[C]. Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on.2008. 245-250
- [52] Theodoropoulos D, Siskos A, Pnevmatikatos D, Ccproc: A Custom Vliw Cryptography Co-Processor for Symmetric-Key Ciphers, in Reconfigurable Computing: Architectures, Tools and ApplicationsSpringer Berlin Heidelberg, 2009, 318-323
- [53] Burke J, McDonald J, Austin T. Architectural Support for Fast Symmetric-Key Cryptography[J].SIGOPS Oper. Syst. Rev. 2000, 34 (5): 178-189
- [54] Chang J K-T, Liu C, Liu S, et al., Workload Characterization of Cryptography Algorithms for Hardware Acceleration, in Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering (Karlsruhe, Germany: ACM, 2011), pp. 381-390
- [55] Sorkin A. Lucifer, a Cryptographic Algorithm[J].Cryptologia. 1984, 8 (1): 22-42
- [56] Biham E, Shamir A, Differential Cryptanalysis of the Data Encryption Standard[M], Springer Science & Business Media, 2012
- [57] Karl H, Willig A, Protocols and Architectures for Wireless Sensor Networks[M], John Wiley & Sons, 2007
- [58] Borghoff J, Canteaut A, Güneysu T, et al., Prince – a Low-Latency Block Cipher for Pervasive Computing Applications, in Advances in Cryptology – Asiacrypt 2012Springer Berlin Heidelberg, 2012, 208-225
- [59] Rivest R, The Rc5 Encryption Algorithm, in Fast Software EncryptionSpringer Berlin Heidelberg, 1995, 86-96

- [60] Aoki K, Ichikawa T, Kanda M, et al., Specification of Camellia-a 128-Bit Block Cipher, (2000)
- [61] Schneier B, Kelsey J, Whiting D, et al. Twofish: A 128-Bit Block Cipher[J].NIST AES Proposal. 1998, 15
- [62] Anderson R, Biham E, Knudsen L. Serpent: A Proposal for the Advanced Encryption Standard[J].NIST AES Proposal. 1998, 174
- [63] Ferguson N, Lucks S, Schneier B, et al. The Skein Hash Function Family[J].Submission to NIST (round 3). 2010, 7 (7.5): 3
- [64] Kwan M, The Design of the Ice Encryption Algorithm, in Fast Software EncryptionSpringer Berlin Heidelberg, 1997, 69-82
- [65] Reeds Iii J A, Cryptosystem for Cellular Telephony, in U.S. Patent 5,159,634[P]. (1992)
- [66] Junod P, Vaudenay S, Fox : A New Family of Block Ciphers, in Selected Areas in CryptographySpringer Berlin Heidelberg, 2005, 114-129
- [67] Yarkov E. Cryptanalysis of Xxtea[J].IACR Cryptology ePrint Archive. 2010, 2010: 254
- [68] Gong Z, Nikova S, Law Y W, Klein: A New Family of Lightweight Block Ciphers[M], Springer, 2012
- [69] Bougard B, De Sutter B, Verkest D, et al. A Coarse-Grained Array Accelerator for Software-Defined Radio Baseband Processing[J].Micro, IEEE. 2008, 28 (4): 41-50
- [70] Yoon J W, Shrivastava A, Park S, et al. Spkm : A Novel Graph Drawing Based Algorithm for Application Mapping onto Coarse-Grained Reconfigurable Architectures[J].2008 Asia and South Pacific Design Automation Conference, Vols 1 and 2. 2008: 752-758
- [71] Mei B. A Coarse-Grained Reconfigurable Architecture Template and Its Compilation Techniques[J].status: published. 2005,
- [72] Rau B R, Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops, in Proceedings of the 27th annual international symposium on Microarchitecture (San Jose, California, USA: ACM, 1994), pp. 63-74
- [73] Shan W, Zhang X, Fu X, et al. Vlsi Design of a Reconfigurable S-Box Based on Memory Sharing Method[J].IEICE Electron. Express. 2014, 11 (1): 20130872-20130872
- [74] Hodjat A, Verbauwhede I. Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/S Aes Processors[J].Computers, IEEE Transactions on. 2006, 55 (4): 366-372
- [75] Jarvinen K U, Tommiska M T, Skytta J O, A Fully Pipelined Memoryless 17.8 Gbps Aes-128 Encryptor, in Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays (Monterey, California, USA: ACM, 2003), pp. 207-215
- [76] Mathew S K, Sheikh F, Kounavis M, et al. 53 Gbps Native  $Gf(2^4)^2$  Composite-Field Aes-Encrypt/Decrypt Accelerator for Content-Protection in 45 Nm High-Performance Microprocessors[J].Solid-State Circuits, IEEE Journal of. 2011, 46 (4): 767-776
- [77] Ahmad N, Hasan R, Jubadi W M. Design of Aes S-Box Using Combinational Logic Optimization[C]. Industrial Electronics & Applications (ISIEA), 2010 IEEE Symposium on. 2010. 696-699
- [78] Megalingam R K, Joseph I P, Gautham P, et al. Reconfigurable Cryptographic Processor for Multiple Crypto-Algorithms[C]. Students' Technology Symposium (TechSym), 2011 IEEE. 2011. 204-210
- [79] Muralimanohar N, Balasubramonian R, Jouppi N, Optimizing Nuca Organizations and Wiring Alternatives for Large Caches with Cacti 6.0, in Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (IEEE Computer Society, 2007), pp. 3-14
- [80] Schneier B, Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), in Fast Software Encryption, Cambridge Security Workshop (Springer-Verlag, 1994), pp. 191-204
- [81] Adams C. The Cast-128 Encryption Algorithm[J].1997,
- [82] Pieprzyk J, Tombak L, Soviet Encryption Algorithm[M], University of Wollongong. Department of Computing Science, 1994

- [83] Matsui M, New Block Encryption Algorithm Misty, in Fast Software EncryptionSpringer Berlin Heidelberg, 1997, 54-68
- [84] Lee J, Park J, Lee S, et al. The Seed Encryption Algorithm[J].SEED. 2005,
- [85] Rivest R L, Robshaw M, Sidney R, et al. The Rc6tm Block Cipher[C]. First Advanced Encryption Standard (AES) Conference.1998.
- [86] Diffie W, Ledin G. Sms4 Encryption Algorithm for Wireless Networks[J].IACR Cryptology ePrint Archive. 2008, 2008: 329
- [87] Zi-bin D, Xiao-hui Y, Qiao R, et al. The Research and Design of Reconfigurable Cipher Processing Architecture Targeted at Block Cipher[C]. ASIC, 2007. ASICON '07. 7th International Conference on.2007. 814-817
- [88] 赵学秘, 可编程密码处理器关键技术研究与实现:[博士].国防科学技术大学. 2006
- [89] Bo W, Leibo L. A Flexible and Energy-Efficient Reconfigurable Architecture for Symmetric Cipher Processing[C]. Circuits and Systems (ISCAS), 2015 IEEE International Symposium on.2015. 1182-1185
- [90] Yang J, Cao P, Hu J, et al. A Hierarchical Context Organization of a Coarse-Grained Reconfigurable Architecture for Block Ciphers[C]. 2015 International Conference on Automation, Mechanical Control and Computational Engineering.2015.
- [91] Suzuki M, Hasegawa Y, Tuan V M, et al. A Cost-Effective Context Memory Structure for Dynamically Reconfigurable Processors[C]. Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International.2006. 8 pp.
- [92] Mucci C, Vanzolini L, Campi F, et al., Interactive Presentation: Implementation of Aes/Rijndael on a Dynamically Reconfigurable Architecture, in Proceedings of the conference on Design, automation and test in Europe (Nice, France: EDA Consortium, 2007), pp. 355-360
- [93] Mao-Yin W, Chih-Pin S, Chia-Lung H, et al. Single- and Multi-Core Configurable Aes Architectures for Flexible Security[J].Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. 2010, 18 (4): 541-552
- [94] Mancillas-Lopez C, Chakraborty D, Rodriguez Henriquez F. Reconfigurable Hardware Implementations of Tweakable Enciphering Schemes[J].Computers, IEEE Transactions on. 2010, 59 (11): 1547-1561
- [95] Zibin D, Wei L, Xiaohui Y, et al. The Research and Implementation of Reconfigurable Processor Architecture for Block Cipher Processing[C]. Embedded Software and Systems, 2008. ICESS '08. International Conference on.2008. 587-594
- [96] Chen-Hsing W, Chieh-Lin C, Cheng-Wen W. An Efficient Multimode Multiplier Supporting Aes and Fundamental Operations of Public-Key Cryptosystems[J].Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. 2010, 18 (4): 553-563
- [97] Bouhraoua A. Design Feasibility Study for a 500 Gbits/S Advanced Encryption Standard Cipher/Decipher Engine[J].Computers & Digital Techniques, IET. 2010, 4 (4): 334-348
- [98] 曲英杰. 可重构密码协处理器的概念及其设计原理[J].计算机工程与应用. 2003, 39 (12): 7-9,19
- [99] Yi W, Yajun H. Fpga-Based 40.9-Gbits/S Masked Aes with Area Optimization for Storage Area Network[J].Circuits and Systems II: Express Briefs, IEEE Transactions on. 2013, 60 (1): 36-40
- [100] Li H, Ding J, Pan Y. Cell Array Reconfigurable Architecture for High-Efficiency Aes System[J].Microelectronics Reliability. 2012, 52 (11): 2829-2836
- [101] Anwar H, Daneshthalab M, Ebrahimi M, et al. Fpga Implementation of Aes-Based Crypto Processor[C]. Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on.2013. 369-372
- [102] Nishikawa N, Iwai K, Kurokawa T. High-Performance Symmetric Block Ciphers on Cuda[C].



## 博士阶段获得的研究成果

### ● 发表的学术论文

- [1] Wei GE, Zhi QI, Lu MA. A Data Prefetch and Reuse Strategy for Coarse-Grained Reconfigurable Architectures[J]. IEICE TRANSACTIONS on Information and Systems, 2013, 96(3): 616-623.
- [2] Ge W, Wen W, Qi Z. A novel loop adaptive hardware design for Coarse-Grained Reconfigurable array[C]//Intelligent Control and Information Processing (ICICIP), 2013 Fourth International Conference on. IEEE, 2013: 518-522.
- [3] Ge W, Zhao M, Wu C, et al. The Design and Implementation of DDR PHY Static Low-Power Optimization Strategies[M]//Communication Systems and Information Technology. Springer Berlin Heidelberg, 2011: 1-6.
- [4] 葛伟,齐志,温闻,等.多种中值滤波算法在可重构架构上的映射实现[J].东南大学学报(自然科学版),2014,44(4):697-702.[doi:10.3969/j.issn.1001-0505.2014.04.003]
- [5] Ma L, Ge W, Qi Z. A Graph-Based Spatial Mapping Algorithm for a Coarse Grained Reconfigurable Architecture Template[M]//Informatics in Control, Automation and Robotics. Springer Berlin Heidelberg, 2012: 669-678.
- [6] Liu X, Ge W, Du Y. Exploration of Spatial Pipeline Computation for Heuristic Access Coarse-Grained Reconfigurable Cells[M]//Mechanical Engineering and Technology. Springer Berlin Heidelberg, 2012: 163-170.
- [7] Xie Z, Liu X, Shan W, et al. ESL Based SoC System Bandwidth Estimation Method[M]//Communication Systems and Information Technology. Springer Berlin Heidelberg, 2011: 15-21.

### ● 授权或受理的发明专利

#### 实用新型专利:

- [1] 葛伟,曹鹏,马俊,等.一种粗粒度动态可重构数据规整控制单元结构. 专利号: ZL 2014 2 0060846.X 公开号: CN103761075 A
- [2] 曹鹏,葛伟,徐凯,等.一种粗粒度可重构层次化的阵列寄存器文件结构. 专利号: ZL 2014 2 0060189.9 公开号: CN103761072 A

#### 发明专利:

- [1] 葛伟,曹鹏,马俊,等.一种粗粒度动态可重构数据规整控制单元结构. 申请号: CN201410046567
- [2] 曹鹏,葛伟,徐凯,等.一种粗粒度可重构层次化的阵列寄存器文件结构. 申请号: CN201410046664
- [3] 刘波,齐志,葛伟,等.支持数据预取与重用的可重构系统. 申请号: 2012105844708

● 参与研究的项目

项目代号	项目名称	项目类型	本人主要工作
NA.	粗粒度可重构阵列及开发工具	委托开发	可重构密码处理器和混合寄存器文件研究
2009ZX01031-001-004	个人移动信息终端SoC芯片研发与应用	国家科技重大专项(核高基重大专项)	在SoC芯片设计实现中，负责架构定义、关键IP的前端设计和验证工作，以及与后端的交互工作
2009AA011701	嵌入式可重构移动媒体处理核心技术	国家高技术研究发展计划（863计划）	可重构媒体处理计算架构建模工作，自动映射流程研究