密码算法可重构实现分析报告

# 修订历史

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订时间 | 修改者 | 描述 |
| V0.1 | 2015-03-31 | 杨锦江 | 初始版本 |
| V0.2 | 2015-04-30 | 闵婧 | 添增架构celator， Cryptoraptor，RCPA |
| V0.3 | 2015-05-04 | 闵婧 | 增添CoARX，COBRA并修改格式 |
| V0.4 | 2015-05-08 | 李小泉 | 增添ProDFA |
| V0.5 | 2015-05-12 | 李小泉&闵婧 | 修订格式 |

# 目录

[1. 修订历史 2](#_Toc419211899)

[2. 目录 3](#_Toc419211900)

[3. 实现密码算法的可重构架构汇总 6](#_Toc419211901)

[4. 可重构架构描述 7](#_Toc419211902)

[**4.1 celator** 7](#_Toc419211905)

[4.1.1 架构概述 7](#_Toc419211906)

[4.1.2 整体架构 7](#_Toc419211907)

[4.1.3 PE微结构 9](#_Toc419211908)

[4.1.4 互联结构 10](#_Toc419211909)

[4.1.5 存储系统 10](#_Toc419211910)

[4.1.6 配置机制 10](#_Toc419211911)

[4.1.7 工具链 11](#_Toc419211912)

[4.1.8 实现性能 11](#_Toc419211913)

[4.1.9 参考文献 11](#_Toc419211914)

[**4.2 COARX** 12](#_Toc419211915)

[4.2.1 架构概述 12](#_Toc419211916)

[4.2.2 整体架构 12](#_Toc419211917)

[4.2.3 PE微结构 13](#_Toc419211918)

[4.2.4 互联结构 13](#_Toc419211919)

[4.2.5 存储系统 13](#_Toc419211920)

[4.2.6 配置机制 14](#_Toc419211921)

[4.2.7 工具链 15](#_Toc419211922)

[4.2.8 实现性能 15](#_Toc419211923)

[4.2.9 参考文献 15](#_Toc419211924)

[**4.3 COBRA** 16](#_Toc419211925)

[4.3.1 架构概述 16](#_Toc419211926)

[4.3.2 整体架构 16](#_Toc419211927)

[4.3.3 PE微结构 17](#_Toc419211928)

[4.3.4 互联结构 18](#_Toc419211929)

[4.3.5 存储系统 18](#_Toc419211930)

[4.3.6 配置机制 18](#_Toc419211931)

[4.3.7 工具链 18](#_Toc419211932)

[4.3.8 实现性能 19](#_Toc419211933)

[4.3.9 参考文献 19](#_Toc419211934)

[**4.4** **Cryptoraptor** 20](#_Toc419211935)

[4.4.1 架构概述 20](#_Toc419211936)

[4.4.2 整体架构 20](#_Toc419211937)

[4.4.3 PE微结构 21](#_Toc419211938)

[4.4.4 互联结构 24](#_Toc419211939)

[4.4.5 存储系统 24](#_Toc419211940)

[4.4.6 配置机制 24](#_Toc419211941)

[4.4.7 工具链 26](#_Toc419211942)

[4.4.8 实现性能 27](#_Toc419211943)

[4.4.9 参考文献 29](#_Toc419211944)

[**4.5** **ProDFA** 30](#_Toc419211945)

[4.5.1 架构概述 30](#_Toc419211946)

[4.5.2 整体架构 30](#_Toc419211947)

[4.5.3 PE微结构 31](#_Toc419211948)

[4.5.4 互联结构 32](#_Toc419211949)

[4.5.5 存储系统 32](#_Toc419211950)

[4.5.6 配置机制 32](#_Toc419211951)

[4.5.7 工具链 33](#_Toc419211952)

[4.5.8 实现性能 33](#_Toc419211953)

[4.5.9 参考文献 34](#_Toc419211954)

[**4.6** **RCPA** 35](#_Toc419211955)

[4.6.1 架构概述 35](#_Toc419211956)

[4.6.2 整体架构 35](#_Toc419211957)

[4.6.3 PE微结构 36](#_Toc419211958)

[4.6.4 互联结构 37](#_Toc419211959)

[4.6.5 存储系统 39](#_Toc419211960)

[4.6.6 配置机制 39](#_Toc419211961)

[4.6.7 工具链 40](#_Toc419211962)

[4.6.8 实现性能 40](#_Toc419211963)

[4.6.9 参考文献 40](#_Toc419211964)

[5. 算法在可重构架构上的性能汇总 42](#_Toc419211965)

[5.1 AES 42](#_Toc419211966)

[5.2 DES 42](#_Toc419211967)

[5.3 SHA 42](#_Toc419211968)

# 实现密码算法的可重构架构汇总

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 架构名称 | 支持算法 | 工具链支持 | 标志文献 | 提出时间 | 研究机构 |
| celator | AES/DES/SHA | 无 | RCFPGA论文 | 2008 | 埃克斯马赛大学 |
| COARX | 以加法、移位、抑或作为基本算子的算法，主要是hash算法。 | 无 | DAC会议 | 2013 | 德国的RWTH Aachen University |
| COBRA | 各种对称密钥算法 | 有：包括一个汇编器和一个时序分析器 | Parallel and Distributed Processing Symposium | 2003 | 德国的University of Massachusetts Lowell |
| cryptoraptor | 多种算法：AES Blowfish, Camellia, CAST-128, DES, GOST, Kasumi, RC5, SEED, and Twofish | 自动配置工具 | ICCAD论文 | 2014 | 德克萨斯州大学奥斯汀分校 |
| ProDFA | 块密码算法 | 无 | ICPADS | 2012 |  |
| RCPA | 多种算法 |  | ASIC会议 | 2007 | 解放军信息工程大学 |

# 可重构架构描述



**4.1 celator**

### 4.1.1 架构概述

Celator架构是由法国埃克斯马赛大学研究提出的，研究本架构可以参考论文[1][2]，其中[1]发表于Reconfigurable Computing and FPGAs会议，[2]则是celator对AES算法实现的一个特别分析，发表于2008年的ICECS会议上。

Celator的PE阵列总共16个计算单元，每个单元处理8-bit的数据，每一个PE可完成各种逻辑操作，便于实现矩阵操作。但是吞吐率不高。

论文中对celator进行的是RTL代码仿真综合，进行0.18nm的工艺验证。

主要优势：和GPP相比能够重构，有着更大的灵活性和更小的面积。

### 4.1.2 整体架构

celator和它的外围架构图示如下：

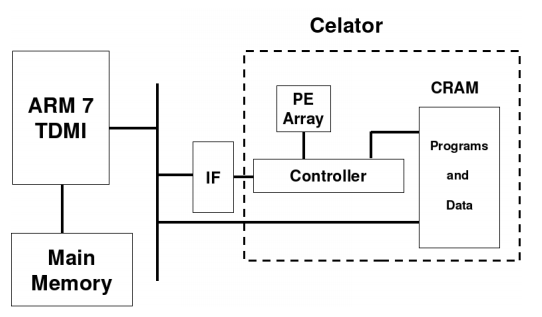


图 4.1.1 celator的外围结构图

ARM7TDMI处理器控制celator需要处理的数据，并且控制所执行的算法，IF接口可以保证处理器和celator之间工作在不同的频率。

Celator包括PE阵列、controller、控制存储器CRAM三个部分。

* PE阵列：4\*4的PE阵列，每一个PE可独立操作，处理8-bit的数据，可以进行byte和bit的操作。由controller控制。

PE的阵列结构图示如下：

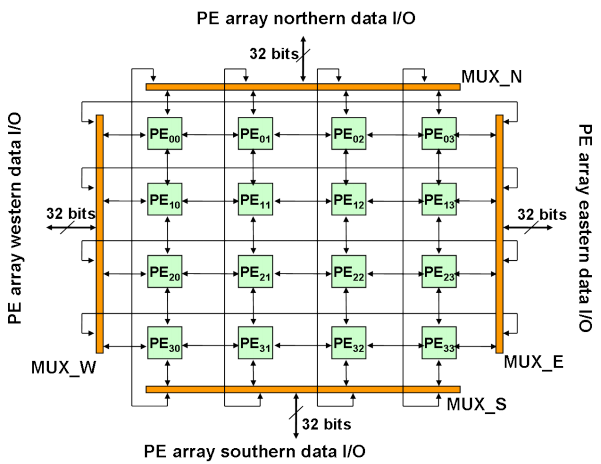


图 4.1.2 PE阵列结构

1. 四个32-bit的输入/输出端口，东西南北各一个
2. 四个多路选择器，控制输入/输出在东和南北之间切换，同时也将PE阵列连为环状，使得PE阵列间可以操作循环移位

* Controller：用于产生PE阵列的控制信号、CPU的指令，以及控制PE阵列的输入输出数据。controller包括一个FSM，5个寄存器。

controller的结构图示如下：

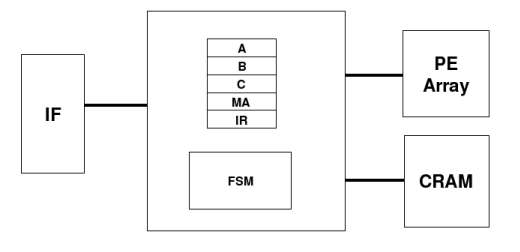


图 4.1.3 controller结构及外围结构图

FSM指令的四种类型：

* RAM访问
* PE操作：PE阵列被分成4行
* PE操作：PE阵列被分成4列
* FSM计算

Controller里面的5个寄存器分别是：

1. A：32-bit，用于查找表操作，存储其配置信息
2. B：32-bit
3. C：12-btt，计数寄存器
4. MA：12-bit存储器地址寄存器
5. IR：29-bit，指令寄存器

* CRAM：可以存储32-bit的数据和指令字，数据包括要处理数据所需的数据（AES的SBox）和加密或是解密的输入输出数据，还有包括controller的指令集。

### 4.1.3 PE微结构

每一个PE包括：

1. 一个算数逻辑单元ALU
2. 两个8-bit寄存器A,B
3. 四个8-bit的数据输入，东南西北每一个方向一个
4. 一个8-bit的数据输出，和最邻近的输入数据的PE相连
5. 两个多路选择器，控制东南西北的输入和输出
6. 两个多路选择器，选择数据从A/B的输入是从计算结果输入还是其他寄存器输入

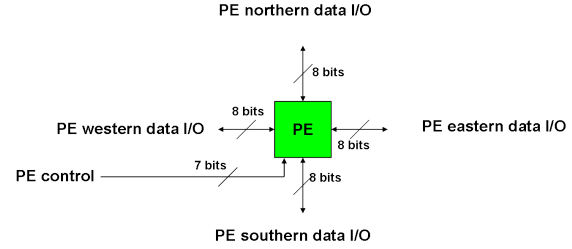


图 4.1.4 PE微结构

每个PE的多路选择器也是由controller控制，PE的ALU的功能可以进行8-bit操作，主要完成的功能如下：

1. xtime（AES需要的操作，包括一个乘一个十六进制书0x02，一个byte的左移，一个和0x1B的异或bit操作[1]）
2. xor
3. and
4. not
5. bypass
6. adder modulo 256
7. 在相邻的两个PE之间进行bit右/左移

### 4.1.4 互联结构

四个多路选择器将整个PE阵列连接起来成为环状，每一个PE和相邻的四个PE都可以进行双向的数据传输。

### 4.1.5 存储系统

CRAM是拥有双向端口RAM，cpu和controller都可以进行读写访问。数据位宽32-bit，地址为10-bit，总大小4KB。存储的数据包括要处理数据所需的数据（AES的SBox）和加密或是解密的输入输出数据，还有包括controller的指令集。

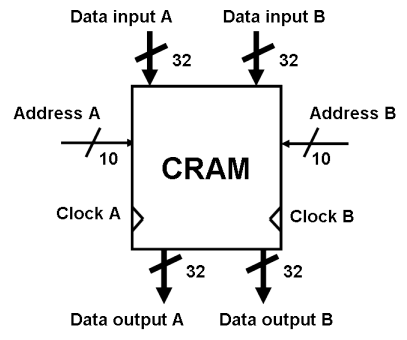


图 4.1.5 CRAM结构

### 4.1.6 配置机制

Celator的配置流程为：

1. CPU处理数据传送给FSM，并且控制所执行的算法
2. FSM从CRAM里加载指令，产生控制信息
3. 控制信息被FSM传送给PE（包括PE的操作信息和输入输出的数据信息）,CRAM（控制CRAM和PE之间的数据传输）,IF。

PE从controller里面加载控制信号，由图4.1.4知道控制信息是7bits，控制信息的功能包括：

1. 选择PE数据的输入来源
2. 控制寄存器的读写
3. 从寄存器数据和非寄存器中的数据中选择PE中A、B的输入
4. 选择ALU的操作
5. 选择PE的输出方向

### 4.1.7 工具链

无

### 4.1.8 实现性能

支持算法：多种算法

实现算法：AES-128-CBC/DES-EBC/SHA-256

实现结果如下：

1. AES-128-CBC，190MHz下，需要514cycles,吞吐率47Mbps，AES-128-EBC需要524cycles,吞吐率为46Mbps
2. DES-EBC算法，需要476cycles，190MHz下吞吐率为26Mbps
3. SHA-256算法，需要2720cycles，在190MHz下吞吐率为36Mbps

### 4.1.9 参考文献

1. Fronte, Daniele, Annie Perez, and Eric Payrat. "Celator: A multi-algorithm cryptographic co-processor." Reconfigurable Computing and FPGAs, 2008. ReConFig'08. International Conference on. IEEE, 2008.
2. Fronte, Daniele, Annie Perez, and Eric Payrat. "The AES in a systolic fashion: Implementation and results of Celator processor." Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on. IEEE, 2008.

**4.2 COARX**

### 4.2.1 架构概述

CoARX（Coprocessor for ARX-based Cryptographic Algorithms）架构由德国的RWTH Aachen University提出，相关学术论文见2013年的一篇论文，单篇论文最高引用4次，论文发表在2013年的Design Automation Conference（DAC）上。

架构主要针对以A（addition）、R（Rotation）、X（eXclusive-or）三种操作作为基本操作的密码算法。

架构用LISA (Languagefor Instruction Set Architectures)语言描述，然后被综合成Verilog RTL，芯片总规模是9.5万门，主频达到700MHz。

架构研究的相关贡献有：

1. 对一个ARX协处理器设计进行了详细的探索；
2. 映射了5个常见的基于ARX操作的密码算法，并针对性地进行了保护和优化；
3. 和其它可编程实现进行了详细的对比。

### 4.2.2 整体架构

CoARX的整体架构如下图所示：

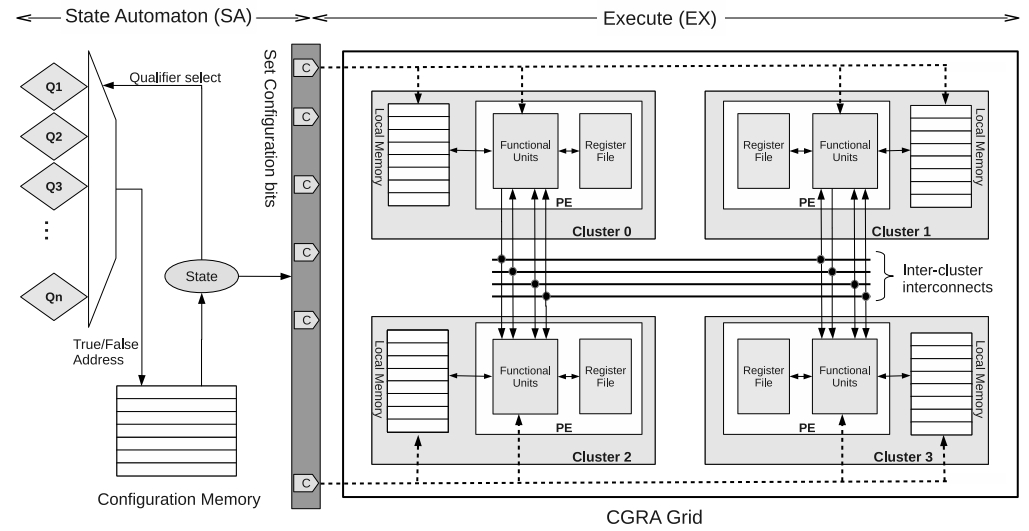


图 4.2.1 整体架构

整体架构分成两部分：State Automaton和Execute。State Automaton负责从配置存储中获取指令，Execute负责执行指令。

State Automaton模块中主要包含配置存储器和Qualifier Control。Execute模块则包含4个Cluster，以及Cluster之间的互连。

### 4.2.3 PE微结构

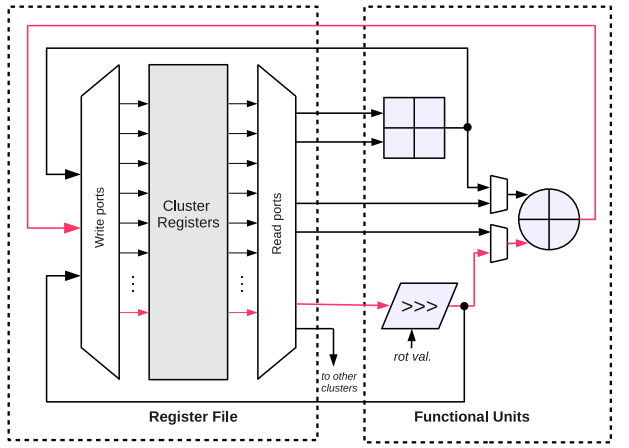


图4.2.2 PE微结构

PE微结构如上图所示可以分为两部分：寄存器堆和功能单元。寄存器堆由6个64位的寄存器组成，功能单元则包含三个基本的运算单元，分别完成A（addition）、R（Rotation）、X（eXclusive-or）三种操作，并且运算单元通过互连还可以完成的额外的两个操作AX,RX。

### 4.2.4 互联结构

由于只有四个Cluster，所以互连变得很简单，架构中实现了每一个Cluster之间三个寄存器位宽（每个寄存器64bit）的数据互联，互联结构如图4.2.3所示。

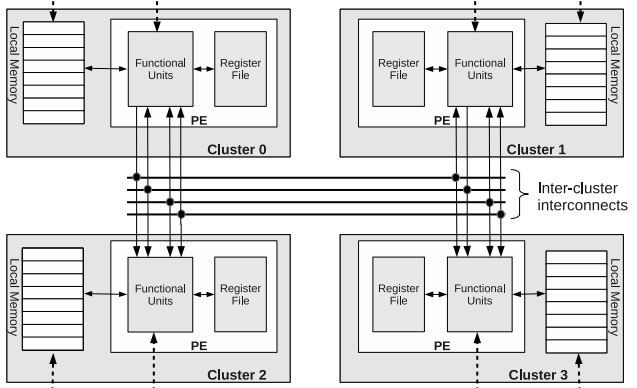


图4.2.3 Cluster间的互联

### 4.2.5 存储系统

整个架构中有两类存储器，一类是每一个Cluster内部的本地存储，用来存储明文、密文、hash消息、初始向量和密钥。本地内存是一个两层结构，第一层是两个64位的寄存器用来保存读写缓存的数据，第二层是一个双口SRAM，大小是64X1024bit。

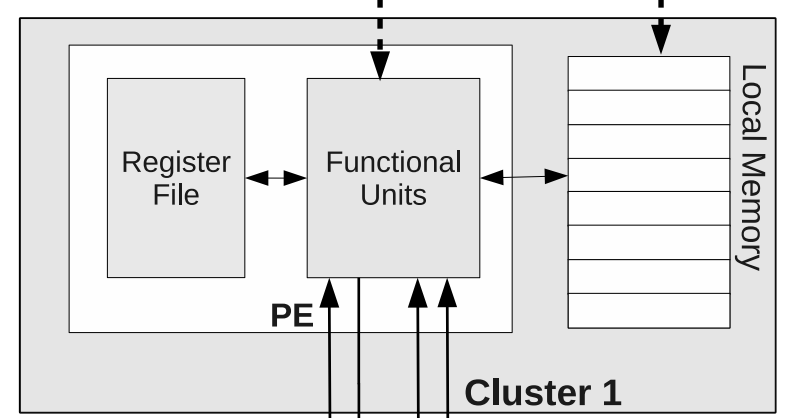


图4.2.4 Cluster结构中的本地存储

另一类存储器是State Automaton中的配置存储，存储算法对应的配置指令。配置大小是269X46bit。

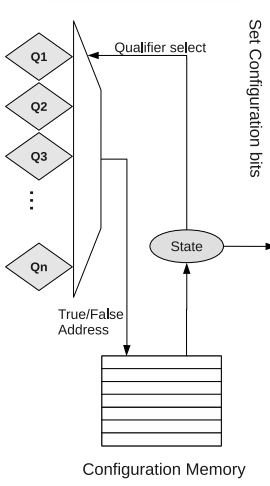


图4.2.5 State Automaton中的配置存储

### 4.2.6 配置机制

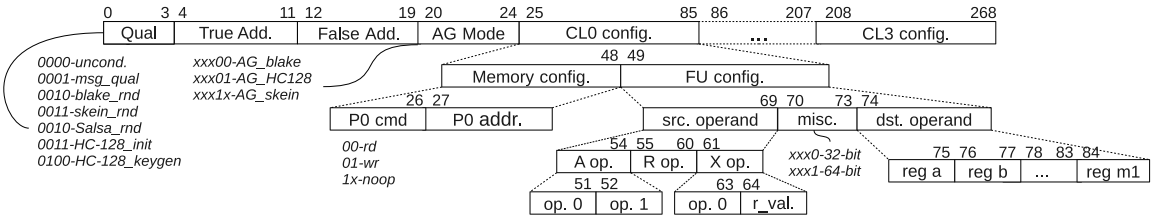


图4.2.6 配置指令字

架构中的配置采用定长（269bit）指令，指令被分成不同的bit段。具体的控制划分如图4.2.6所示，其中：

1. 0-4是Qualifier Control段，用来完成真、假条件地址的获取；
2. 4-11、12-19分别是真、假条件地址；
3. 20-24对应的是特殊的算法模式，因为有些算法中会包含一些特殊的操作，这些操作不属于ARX的范畴，因此架构中针对这些算法定制了功能模块，当运行这些算法时会通过进入特定的算法模式来启动这些定制模块；
4. 25-268对应4个Cluster的配置，每一个Cluster需要61bits的配置信息。

### 4.2.7 工具链

目前采用的是手动书写配置指令的方式，但是由于指令定长且类型固定，因此想要对其进行自动化生成是非常简单的。

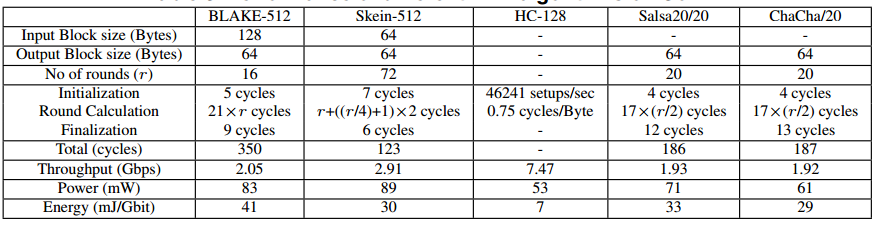
### 4.2.8 实现性能

支持算法：设计目标是支持各种以加法、移位、抑或作为基本算子的算法，主要是hash算法。

实现算法：BLAKE-512、Skein-512、HC-128、Salsa20/20、ChaCha/20

在90nm的ASIC实现的算法性能

表4.2.1 算法在CoARX上的实现性能



论文中还将对应的实现与ASIC、通用多核架构进行了对比。

表4.2.2 与ASIC对比（对齐到90nm）

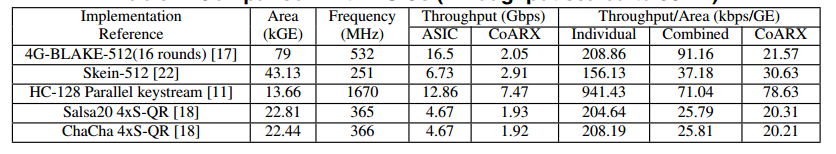
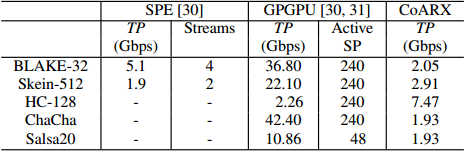


表4.2.3 与通用多核架构对比



### 4.2.9 参考文献

[1] Shahzad, Khawar, et al. "CoARX: a coprocessor for ARX-based cryptographic algorithms." Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE. IEEE, 2013.

**4.3 COBRA**

### 4.3.1 架构概述

COBRA（Cryptographic (Optimized for Block Ciphers) Reconfigurable Architecture）架构由德国的University of Massachusetts Lowell提出，相关学术论文见2003年和2005年的两篇论文，单篇论文最高引用63次，论文发表在03、05年的Parallel and Distributed Processing Symposium上。

架构主要针对分组密码设计，对比较常用的41个分组算法的算子进行了分析，得到了下图所示的算子统计。

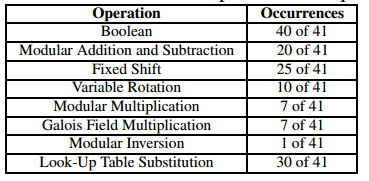


图4.3.1算子统计

在最终PE设计时包含了除Modular Inversion的所有算子，因此架构可以支持大部分的分组算法，并且同样可以实现未来提出的含有这些算子的分组算法。

架构用VHDL语言描述，综合后芯片总规模是6691514门，用SRAM代替触发器存储后面积可以缩减到250万门。在对AES进行映射时能达到1451.25Mbps的吞吐性能。

### 4.3.2 整体架构

COBRA的整体架构如图4.3.2所示，它是一个4X4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、3列的RCE是不带乘法器的，而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的单元，实现数据的按字节换位。在阵列的两边各有8个eRAM，左边的8个供左边的两列使用，右边的8个则被分配给右边的两列。在第3行输出的地方有4个32位的寄存器，实现每轮数据输出的反馈。 整个架构通过微控制码控制，因此还有一个iRAM来存储微控制码。

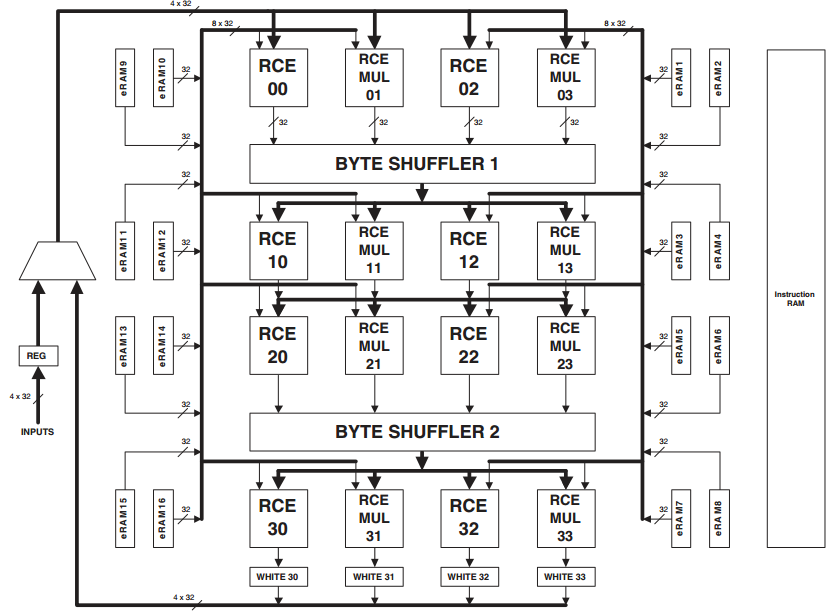


图 4.3.2 整体架构

### 4.3.3 PE微结构

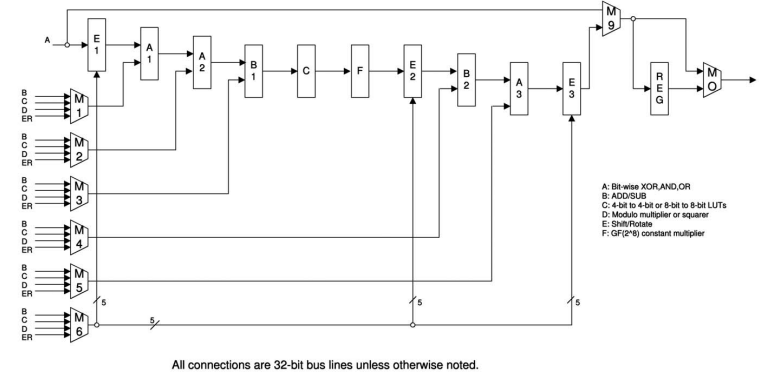


图4.3.3 RCE微结构

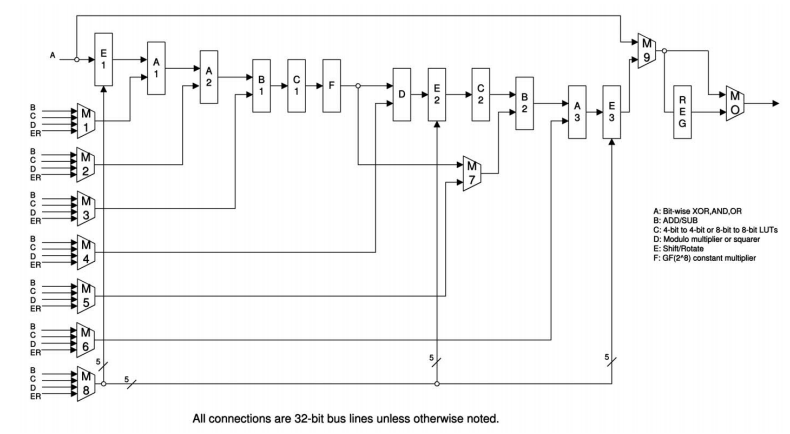


图4.3.4 带乘法的RCE微结构

如图4.3.3和4.3.4所示，PE微结构根据支持运算的不同分为两种，一种是普通的RCE，一种是带乘法的RCE。图中的A-F的含义如下：

1. A:按位与、或、抑或
2. B: 2^8、2^16、2^32的模加、模减
3. C:查找表（8-8，或者8个4-4）
4. D:2^16、2^32的模乘和2^32的模乘方
5. E:左右移位，左右循环移位，移动的位数可以配置
6. F: 2^8的有限域乘法
7. M:多路选择器
8. REG:寄存器

PE中的连线都是固定的，每一条连线都是32位，其中数据A作为第一个操作数固定，第二操作数则由M(1:6,8)的输入决定，结构中的每一个子运算单元都被精心排序来实现更多的运算组合。每一个算子单元都可以通过控制码配置成enable或者disable，查找表可以被配置成一个8-8，或者8个4-4。当需要对RCE进行流水时则通过配置REG寄存器来实现。RCE使用一条配置指令控制。

### 4.3.4 互联结构

架构的互联方式如图4.3.2；互联采用的是固定互联和两个可以实现字置换的功能单元。128位的输入数据被分成4组，分别分配给4列，作为每一列中所有的RCE的第一个输入。第二个输入则可以来自上一行的输出、来自字置换单元的输出或者来自eRAM。这些输入经过RCE内部的MUX得到RCE内部各个阶段所需要的第二个操作数。

### 4.3.5 存储系统

架构中的存储系统主要包括16个eRAM和一个iRAM。其中eRAM用来存储中间值、轮密钥等；一半eRAM被分配给1、2列，其余的一半则被分配给3、4列。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。iRAM则被用来存储微控制码，系统采用的超长指令字(VLIW)来实现控制的，控制码为80位；iRAM的大小是12bit X 80bit，因此可以存储多达4096条控制码。

### 4.3.6 配置机制

采用80位的超长指令字控制指令，80位中主要包含如下配置位：operation code、slice address、element address、LUT address、configuration data fields，其中operation code表示操作类型，slice address表示配置对象，element address表示RCE内部的操作对象，LUT address表示RCE内部的查找表和有限域运算，configuration data fields存储RCE内部单元的配置信息。

当需要对一个算法进行配置时，首先会编写配置的汇编代码，这些代码就是对架构中的模块以及模块内部的可配置单元进行编程，当汇编代码编写完成后通过汇编器将代码转换成80位的控制码的形式。

### 4.3.7 工具链

软件工具包括一个汇编器和一个时序分析器。汇编器的工作是把COBRA 汇编语言编译成COBRA架构的配置控制码。目前只支持从汇编到控制码的流程，未来希望支持从C等高级语言到配置控制码的编译。时序分析器则用来获取特定映射的延迟信息。时序分析器会对不同配置的RCE、shuffler等的延迟进行分析，当配置更新时需要根据指令的特点生成一个mapping structures，时序分析器最后会得到多个mapping structures，这些mapping structures包含对应的row、byte shuffler和whitening register的延迟，最后对比每一个mapping structures中的最大延迟，确定最终延迟。

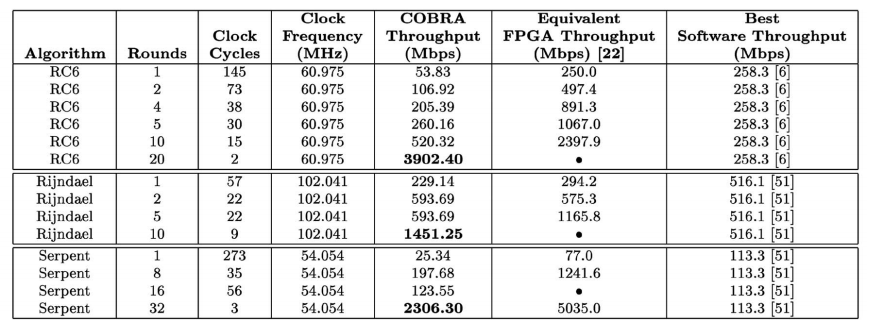
### 4.3.8 实现性能

支持算法：设计目标是支持各种对称密钥算法

实现算法：RC6、Rijndael、Serpent

在0.35um下实现，对应的算法性能如4.3.1表所示，论文还将实现性能与当时的FPGA实现、软件实现进行了对比。

表4.3.1 算法映射性能与对比



### 4.3.9 参考文献

[1] Elbirt A J et al. “Instruction-Level Distributed Processing for Symmetric-Key Cryptography.” Parallel and Distributed Processing Symposium. 2003. Apr. 22, 2003. pp. 78-87.

[2] Elbirt, Adam J., and Christof Paar. "An instruction-level distributed processor for symmetric-key cryptography." Parallel and Distributed Systems, IEEE Transactions on 16.5 (2005): 468-480.

* 1. **Cryptoraptor**

### 4.4.1 架构概述

Cryptoraptor是由德克萨斯州大学奥斯汀分校于2014年研究提出，研究论文可参考[1,2]，其中[1]发表于2014年的ICCAD会议，提供了详细的和ASIC、FPGA、CPU/GPU的实现对比结果。

Cryptoraptor和现有的ASIC电路相比虽然没有ASIC的吞吐率更高，但是它有着更大的灵活性，可以实现更多的算法，并且面积更小，有着更小的功耗，在每单位面积的吞吐率上和ASIC电路的高水平取得了差不多的结果。

对于cryptoraptor性能的介绍，研究提供了两种验证方式，一种是45nm工艺的ASIC验证，一种是采用40nm的FPGA Virtex6进行验证。所实现的算法包括AES, Blowfish, Camellia, CAST-128, DES, GOST, Kasumi, RC5, SEED, and Twofish。

Cryptoraptor的优势：

1. 主要针对于对称密码算法和哈希算法，设计的一款快速灵活高性能的处理器。
2. 可以支持现有的和可以预见的对称密钥算法标准，拥有最广泛的密码算法实现的覆盖率，唯一的一个针对于未来加密标准的专门的加密算法处理器
3. 拥有高性能、高灵活性、面积小的优势

### 4.4.2 整体架构

整体架构结构图如下图所示：

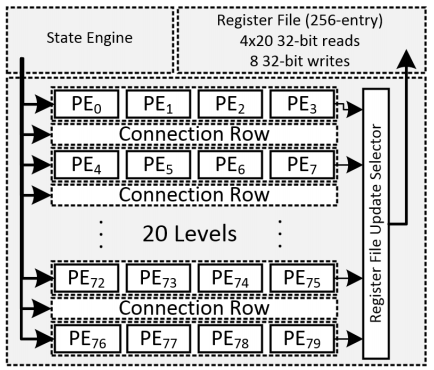


图4.4.1整体架构

包括一个执行模块ET，状态机控制模块SE，和一个256位的寄存器堆。

1. 状态机SE，可以进行初始设置，并且在算法不改变的时候保持常值不变。包括一个状态计数器，一个小的控制存储模块。
2. ET：完成主要功能的部分，由许多相似的stage组成，每一个stage包括一系列的PE，这些PE通过connection row（CR）连接到下一个stage。每个stage里包括一行PE和CR，PE每四个并行，并且可以独立配置。因为87.2%的密码算法只需要4个或是更少的PE就可以完成。在研究的148种算法里面只有19中需要8个或是16个PE的方式实现，但是8个或是16个PE的方式则会降低资源的使用率，并且将PE间的互连变得复杂，成本提高。
3. Cryptoraptor包括20个stage，一个256位的寄存器堆进行配置。充分利用控制存储器的话可以实现四个独立的20级stage片的工作。

### 4.4.3 PE微结构

PE，一个PE包括五个可以做通路也可以独立配置的FU单元，一个算数单元AU，一个逻辑操作单元LOU，一个查找表TLU，一个移位变换寄存器SRU，还有一个置换扩展单元PEU。每个PE旁边放置一个小的存储器模块，存储每个PE的控制信息，实现PE的重构。这些小的存储器可以存储4个由状态机控制的序列信息，可以在不同的cycle进行不同的配置。

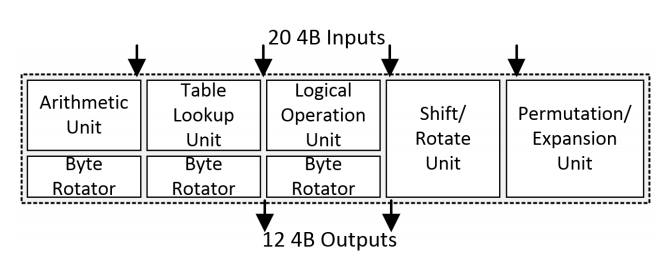
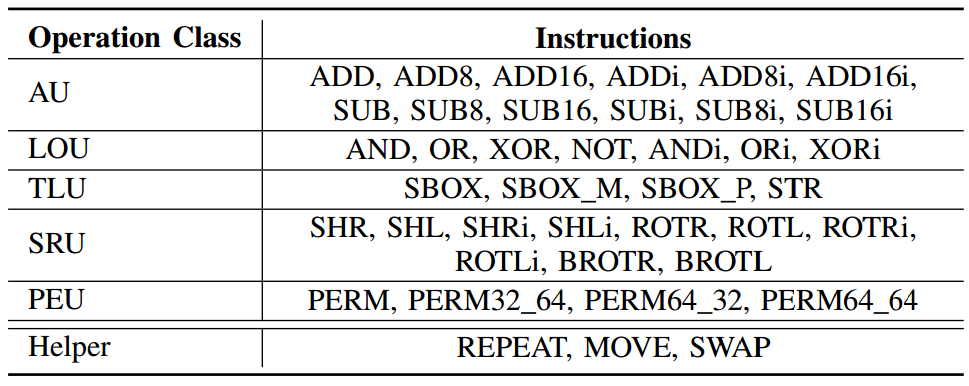


图 4.4.2 PE结构

因为PE的五个部分可以并行执行，所以每一行的PE接收20x4B输入信息和4x7-bit的控制信息，产生12x4B的输出信息。因为有四分之一以上的算法的置换操作是byte的，所以在AU、TLU、LOU都置入了byte的置换操作，可以减少算法运行的cycle。

PE的操作指令：

表 4.4.1 PE的操作指令



1. TLU

查找表的结构图如下所示：

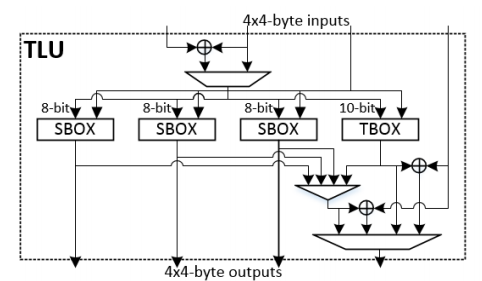


图4.4.3 TLU的结构

包括三个SBOX一个TBOX，SBOX大小是256\*4B，TBOX大小是1024\*4B。加入了XOR操作，可以使得查表的工作在一个周期内完成。

查找表可以进行三种工作模式：

* 一个查找表最大10bits的地址，反馈4B的输出
* 4个查找表每一个查找表4B输入作为一个输入地址，返回4个4B的输出
* 一个查找表输入4B的输入地址信息，返回4B的输出信息

1. AU

算数单元的结构图如下图所示：

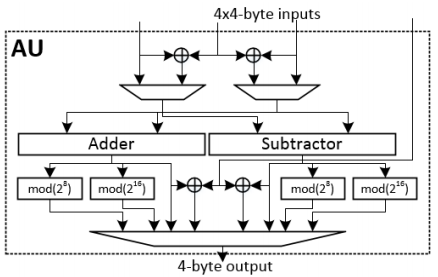


图4.4.4 AU的结构

支持8个16-bit的数据操作。

1. LOU

逻辑操作单元的结构图如下图所示：

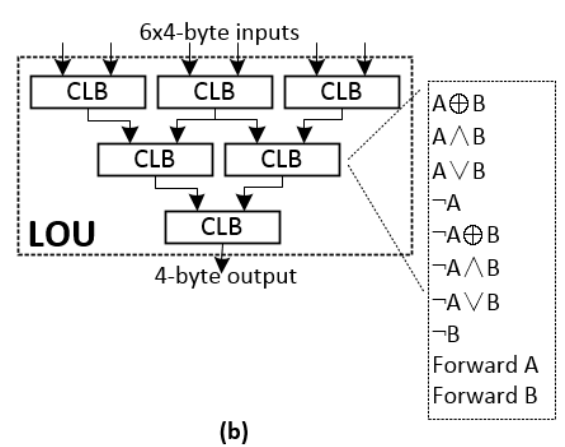


图 4.4.5 LOU的结构

包括6个独立的配置/旁路逻辑单元（CLB），每个CLB能够进行4种逻辑操作，与、或、非、异或。输入信息是6个4B数据，输出一个4B数据。LOU可以完成大多哈希算法的计算方程，最高输入为6的方程式可以在一个cycle里面完成。总共实现三级操作。

1. PEU比特操作，可以合并、操作、扩展两个32bits的数产生一个最高到64bits的输出或是2个32bits的输出。
2. SRU:32bits的双向移位循环移位操作，在SRU之前和之后可以进行灵活的逻辑操作（与、或、非、异或）

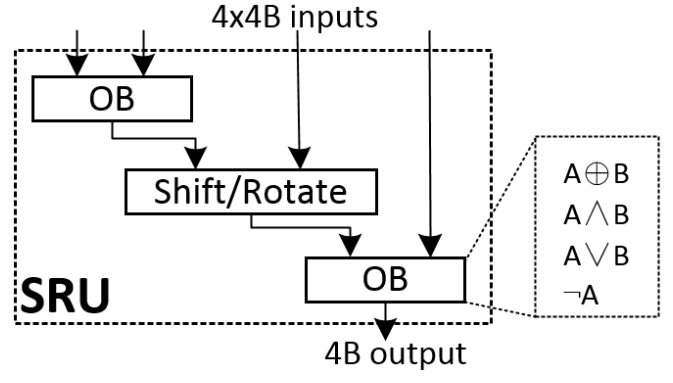


图 4.4.6 SRU的结构

每个shift/rotate接收两个4B的输入数据，一个作为操作数，一个是移位的数目。因为大多数的移位操作之前或结束之后都会有异或操作，所以在shift/rotate的之前之后都加入了异或操作功能。

### 4.4.4 互联结构

CR通过8个并行的register file的写端口，控制各个stage的8个输出到寄存器堆，CR在FU之间的连线已经确定，连接着这一stage各个PE通向下一个stage各个PE的通路，CR旁边放置一个存储器模块提供CR需要的配置信息。每一行的PE都可以连接至后面一行的任一个PE。

### 4.4.5 存储系统

1. 256位的存储器，32位操作，8个并行的32位写端口，存储立即数，可以根据不同的算法配置不同级数的流水，每一行有4个，总共20级的32位读。
2. PE、CR内部的存储器，每一个PE、CR放置一个小的配置、控制信息存储器（没有说明大小）
3. 状态机SE中，一个控制信息存储的存储器。（没有说明大小）

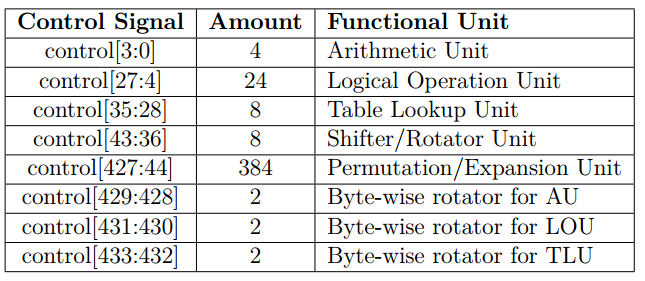
### 4.4.6 配置机制

根据指令集配置，指令也可以在32位的立即数上操作，不支持乘法器运算结构。采用一个工具链展开轮循环，产生数据流图，优化操作顺序。分配操作给可用的FU，也可以进行手工的配置。

* PE的控制

整个PE的控制信息如下：

表 4.4.2 PE的控制信息



每一个PE采用434-bit的控制信息配置。每个PE加载信息是4B形式，根据不同的加密算法最多会花费56个周期。

控制PE下载控制信息的接口控制为7-bit，控制信息组织形式为：

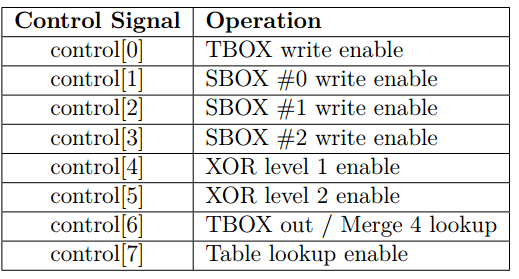
PEcontrol = {WriteEnable || State || WordAddress}

2-bit的state 4-bit的wordaddress，加载控制信息的时候，writeenable需要打开，在其他阶段保持只读模式。

1. TLU的控制

查找表控制信息为8bits。

表 4.4.3 TLU的控制信息



1. LOU的控制

控制信息总共24bits，，每一个CLB有4-bit的控制信息。形式为：

LOUcontrol = {CLB5||CLB4||CLB3||CLB2||CLB1||CLB0}

1. PEU的控制

每一个PEU包括64个控制输出bit的BS，BS由6-bit的控制信息控制。整个PEU的控制信息383bits，形式为

PEUcontrol = {BS63||BS62|| . . . ||BS2||BS1||BS0}

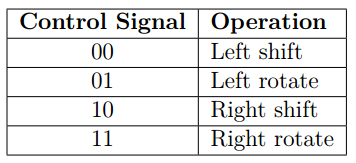
1. SRU的控制

由8-bit的控制信息控制，分为两个部分，一部分是移位变换的控制，一部分是逻辑操作的控制，形式为：

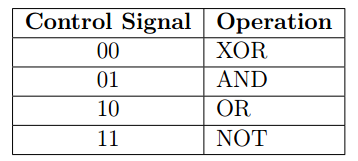
LOUcontrol = {enOp2||ctrlOp2||ctrlSHF T ||enOp1||ctrlOp1}

移位变换的控制如下：

表 4.4.4 移位控制信息配置



Opration block的控制如下：

表 4.4.5 逻辑操作控制信息配置

* CR的控制

CR控制PE的连接，每一个PE的连接由120bits的控制信号配置，6bits控制输出信息，每一个CR总共(8x6)+(4x120) bits的控制信息，组织形式为

control = {RegOutSLCT || P EconnectorCT RL}

RegOutSLCT = {OU T slct7 || OU T slct6 || . . . || OU T slct1 || OU T slct0}

每一个CR里面的控制信息由状态机里的控制存储器支配，在最开始的时候加载，根据不同的算法最多需要17周期。每一个CR的控制信息以32bits的形式输入，每一个CR的控制信息加载的控制是7-bit，其组织成如下形式：

CONN control = {W riteEnable || State || W ordAddress}

2-bit的state 4-bit的wordaddress，加载控制信息的时候，writeenable需要打开，在其他阶段保持只读模式。

### 4.4.7 工具链

一个可以自动配置的工具，相对于手动配置优化了PE的资源利用率。

### 4.4.8 实现性能

支持算法：AES, Blowfish, Camellia, CAST-128, DES, GOST, Kasumi, RC5, SEED, and Twofish

实现算法：AES Blowfish, Camellia, CAST-128, DES, GOST, Kasumi, RC5, SEED, and Twofish

AES算法的实现：

AES算法实现结果如下：

1. ASIC实现，45nm工艺，针对于AES-128加密算法，频率可以达到1GHz, CBC模式吞吐率达到 6.4Gbps, CTR模式吞吐率达到 128Gbps,面积6.32mm2。
2. FPGA实现，Virtex6，40nm工艺，针对AES加密算法，频率可以达到203.8MHZ, CBC模式吞吐率达到 1.3Gbps, CTR模式吞吐率达到26.09Gbps。

结果对比如下：

AES算法ASIC实现在Cryptoraptor架构和其他架构上的对比结果：

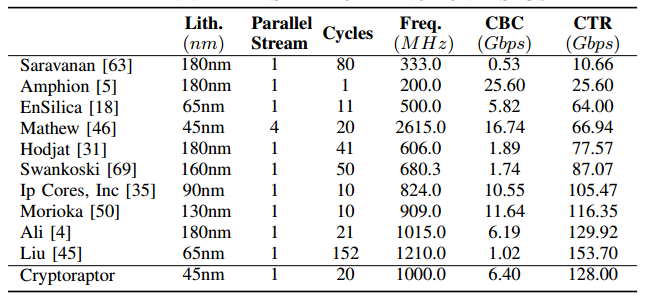


图 4.4.7 cryptoraptor的ASIC验证及性能对比

由上图分析，所实现的结果与ASIC电路结果相比，cryptoraptor并不是性能最好的，morioka在CBC和CTR模式，吞吐率分别达到了11.64Gbps和116.35Gbps，crptoraptor的CBC模式的吞吐率差不多只有它的一半大小，原因是因为流水线的深度会影响CBC模式下的吞吐率，但是CTR模式下的吞吐率cryptoraptor也是没有Morioka的性能好，但是，Morioka是只针对于AES加密算法的，它通过优化AES的外循环和数据通路达到高性能，是AES假面算法的准用处理结构，cryptoraptor在支持多种算法的基础上达到了这样的性能。

Liu的AES处理架构借助于152级深度流水、高主频、多核处理在CTR模式达到了很高的性能，153.7Gbps，但是在CBC模式的性能并不好，cryptoraptor在支持多种密码算法的基础上，可以在单位面积上的吞吐率达到了相差不多的结果，与Liu的结果相比，分别为20.25Gbps/mm2 和23.18Gbps/mm2。

Amphion因为它频率低在CBC模式取得了最优的性能，但是他的CTR模式下的性能却不好，Ip Cores则是在CBC和CTR模式之间做了一个很好的平衡，但是，cryptorator依旧凭借着可重构性，支持多种算法的优势是多种架构中的最优。

AES算法在FPGA上实现Cryptorator实现性能和其他架构的对比结果

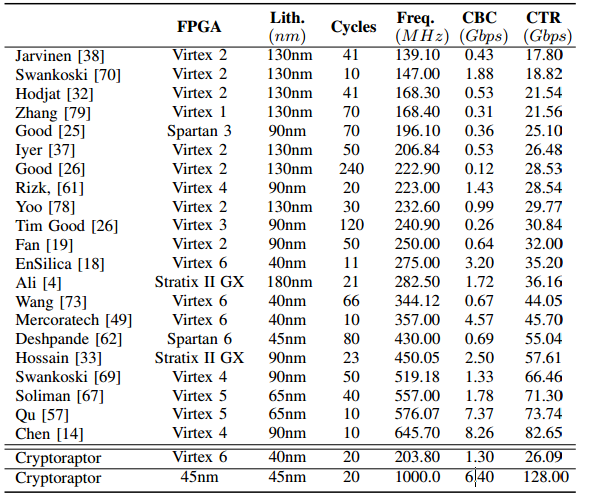


图 4.4.8 cryptoraptor的FPGA验证及性能对比

Chen的处理中，针对AES算法采用外循环的方法，在Xilinx Virtex-4 FPGA上频率达到了645.7MHz，也因此在AES-CTR的吞吐率高达82.65Gbps，但是cryptoraptor的作者根据Chen所提供的源代码，采用ISE Design Suite 14.6综合仿真，结果只得到284.43MHz，36.41Gbps的结果。同样的，Qu也声称在Virtex-5上达到了576.07MHz的频率，Soliman声称达到了557MHz的频率，作者感觉是不可信的。另一方面，即使是EnSilica和 Mercoratech声称的商业AES处理器，也只达到了275MHz和357MHz的时钟频率。

但是图4.4.8确实反映了流水深度和AES-CTR之间的关系，以及与CBC模式之间的制约。因为cryptorator计划进行ASIC实现，所以在FPGA上达到203.8MHz，AES-CBC模式下1.3Gbps，AES-CTR模式下达到27.31Gbps的性能是不错的。

AES实现在cryptoraptor和GPPs上的对比结果：

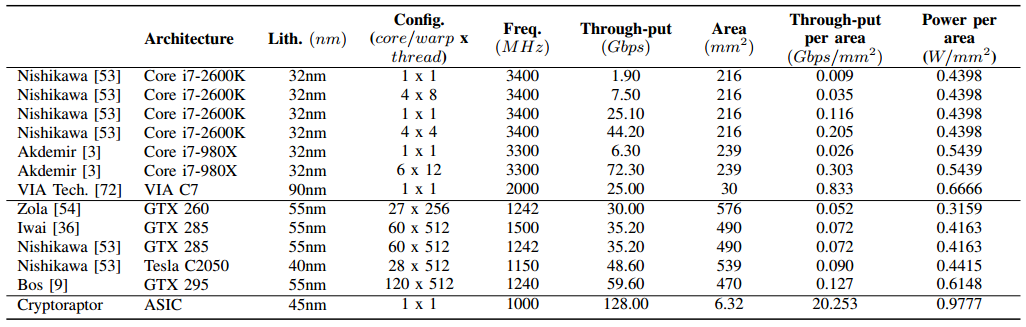


图 4.4.9 cryptoraptor与GPP的性能对比

CPU和GPU依靠着高时钟频率、多线程也达到了相当不错的性能，最高达72.3Gbps/59.6Gbps，但是他们占用的面积是相当大的，cryptoraptor在单位面积上的吞吐率上有着不可比拟的优势。

Cryptoraptor所实现的算法以及其性能：

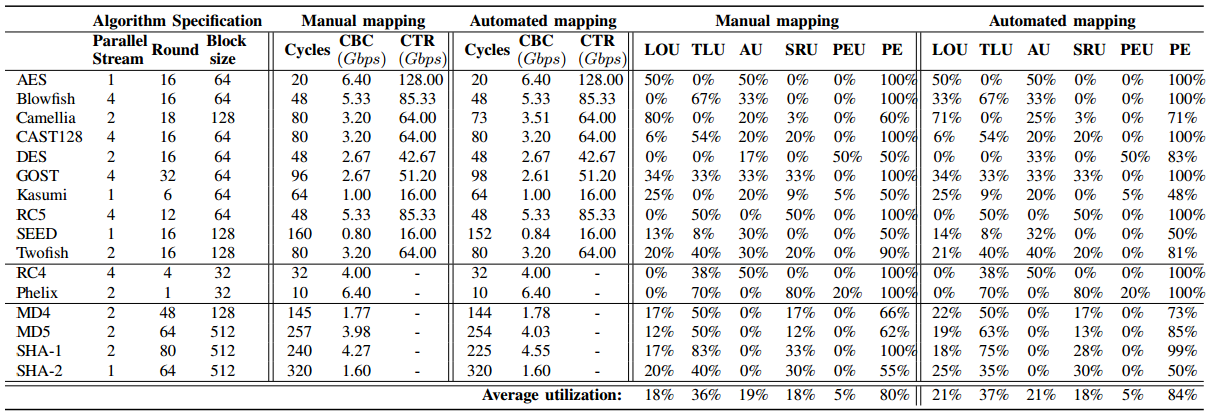


图 4.4.10 Cryptoraptor所实现的算法及其性能

### 4.4.9 参考文献

1. Chiou D. Cryptoraptor: high throughput reconfigurable cryptographic processor[C]//Proceedings of the 2014 IEEE/ACM International Conference on Computer Aided Design. IEEE Press, 2014: 154-161.
2. Chiou D. Cryptoraptor: High Throughput Reconfigurable Cryptographic Pro cessor for Sy mmetric Key Encryption and Cryptographic Hash Functions [D]. The University of Texas at Austin 2014.
   1. **ProDFA**

### 4.5.1 架构概述

ProDFA（programmable dataflow computing architecture）架构由国防科技大学提出，相关学术论文见2012年的一篇论文，单篇论文最高引用1次，论文发表在2012年的Parallel and Distributed Systems (ICPADS)上。

架构的整体思想是采用单周期的配置时间、更长的执行时间来提高整体性能。

架构用Verilog语言描述，采用SMIC的0.13um工艺库。对于含有四个RPU的架构，总面积是0.72mm（不包括SRAM的）。芯片主频是400MHz, AES算法的并行模式下能达到4.2Gbps的吞吐率。

架构研究的相关贡献有：

1. 提出一个单周期的配置时间、长执行时间的架构
2. 介绍了架构的结构和执行方式，介绍了配置和优化；
3. 对架构进行了映射测试，实现了多个分组密码，并对结果进行了分析。

### 4.5.2 整体架构

ProDFA的整体架构如下图所示：

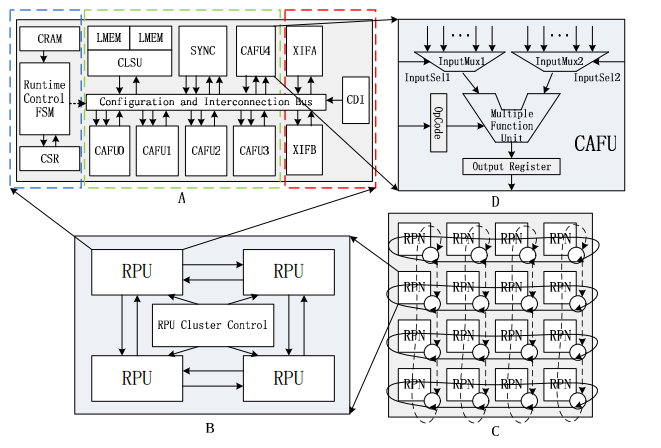


图 4.5.1 整体架构

A子图是架构处理单元微结构RPU(reconfigurable processing unit)，B子图是可重构处理节点RPN(reconfigurable processing node)，子图C是由多个RPN组成的可重构处理阵列，子图D是一个可配置定制功能单元CAFU(configurable application-specific function unit)。通过这个分层结构，应用有更大的需求时，整个架构可以被扩展成一个更大的规模。

### 4.5.3 PE微结构

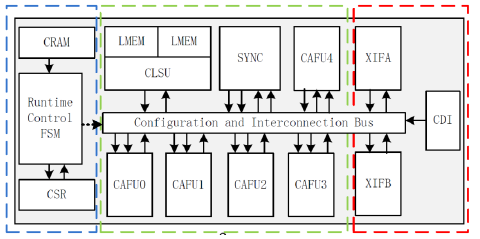


图4.5.2 PE微结构

PE微结构如上图所示可划分为三个部分，配置控制部分、功能运算部分、外部接口部分。如图4.5.2 所示。

配置控制部分主要负责结构运行环境和配置信息的管理。包含配置存储器（CRAM）、运行控制有限状态机（Runtime Control FSM）、控制状态寄存器（CSR）。CRAM所存配置信息；FSM相当于一个指令译码器，派发配置信息，并且保持RPU中各个部件的运行上下文；CSR则保存RPU的运行状态，这个状态可被外部的处理器读取。

功能运算部分包括可配置的专用运算单元CAFU0~CAFU4，完成所有的运算。存储单元LMEM，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。

外部接口部分包含两个外部数据接口（XIFA 和 XIFB）和一个外部输入接口（CDI），可以通过互联接口扩展ProDFA的结构。

在时下密码算法时，架构为密码算法设计了6个运算单元，这些运算单元是根据密码算法的需求设计。

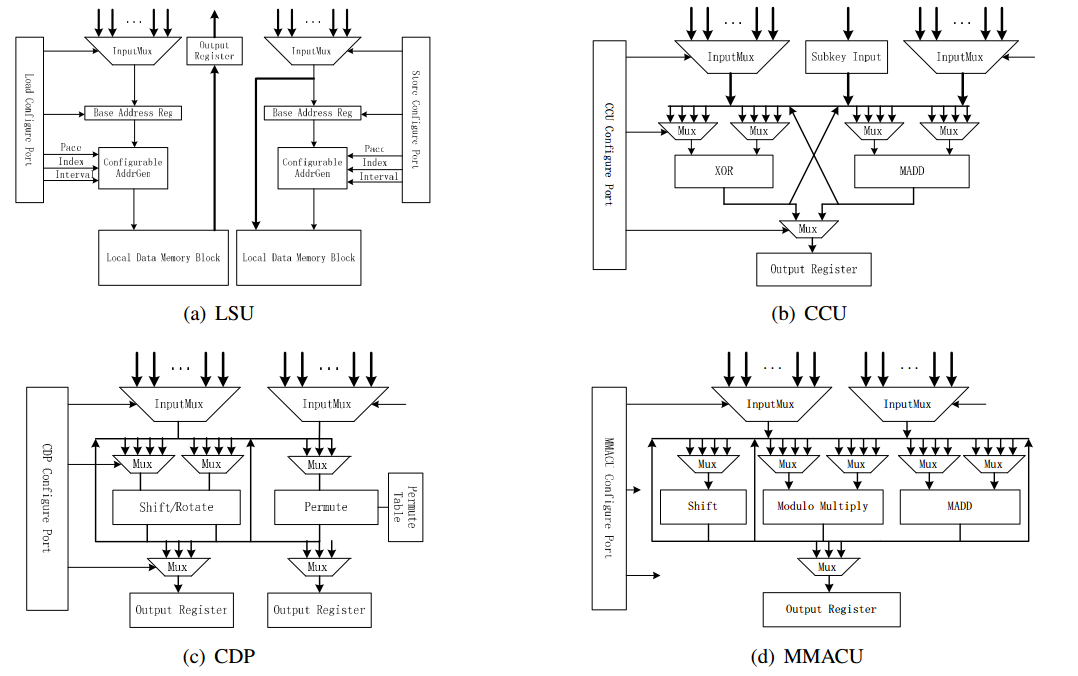


图4.5.3 密码算法定制的运算单元

LSU：load/store单元，完成数据从LMEM到运算单元的存取。

CCU：可配置运算单元，完成抑或和模加运算

CSU：完成查找表SBOX

CDP：移位和置换

MMACU：模乘单元

SYN：数据同步单元；在配置流水线时两边的数据可能是不同步的，因此快的需要等待慢的数据到达，因此需要对快的数据进行数据锁存等待。

### 4.5.4 互联结构

如图4.5.2，在RPU内部，各个执行单元之间的互连通过可配置的CIB完成。

### 4.5.5 存储系统

如图4.5.2，存储分为两部分，配置存储CRAM，大小为16K。数据存储单元LMEM，存储运算单元的所需的数据。

### 4.5.6 配置机制

架构通过为每一个单元保存执行上下文，支持零延迟的局部配置切换。通过一个同步单元可将整个RPU配置成一个流水线执行模式。

执行上下文切换：

指令操作的时间序列、寄存器文件和程序计数器成为指令流体系结构的抽象计算模型。对于可编程数据流体系结构，其计算过程与各个功能单元构成的特定数据流结构紧密相关,其抽象计算模型由特定数据流结构的时间序列构成。计算过程中，任意时刻 ProDFA 可重构处理单元内部的数据流结构称之为该时刻可重构处理单元内部的执行上下文。

在 ProDFA中,执行上下文是指由所有功能单元的功能配置和互联配置构成的运行状态信息，通常为一个规模较小的数据流图。而在阵列结构的粗粒度可重构结构中，执行上下文则是一个规模很大的数据流图结构，因此带来上下文切换时需要的配置信息量过大的问题；但在ProDFA中，上下文的内容要少很多,这使得执行上下文的切换可以通过硬件得到直接支持，用于实现执行上下文的零延迟切换。

在可重构处理单元 RPU 的控制单元内部设置有执行上下文缓存,当对每条配置信息进行译码时，可根据配置中的标志位信息，将配置信息放入执行上下文缓存或直接流出至配置总线。被放入执行上下文缓存的配置信息只有在进行上下文切换时才会流出至配置总线并生效,而直接流出至配置总线的配置信息则可视为立即切换执行上下文的配置。通过上下文切换命令或者设置自动切换标志位等两种方式，均可以实现上下文的单周期切换。实现执行上下文的快速切换是提高可重构结构计算效率和计算性能的重要一环。

单周期的执行上下文切换在如图 4.5.4所示的两种不同情况下均能发挥作用。第一种情况如图 4.5.4 中所示的执行上下文A转换至执行上下文B，上下文A和B存在较大的差别,亦即当RPU内部的数据流结构需要较大改变时的情况，快速执行上下文切换可以在一个时钟周期的开销下实现运算结构的改变。这种数据流结构较大改变的执行上下文切换能够执行的前提是当前数据流结构中各个功能单元的当前相关距离为1，即当前周期各个功能单元之间的数据依赖在下一个时钟周期即可满足。否则需要等待功能单元构成的数据通路流水线继续执行，直到功能单元之间的最大相关距离为1。

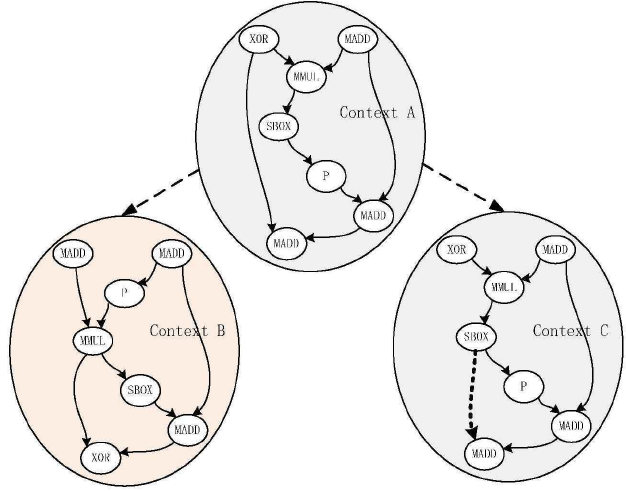


图4.5.4 执行上下文切换的两种情况

第二种快速切换的情况如图4.5.4中所示的执行上下文A切换至执行上下文C，上下文A和C之间的差异较小，亦即当 RPU 内部的计算结构仅需作微小调整的情况，执行上下文的切换可以实现重构开销的完全隐藏。考虑到具体应用中可能出现的RPU内部计算结构在两个微小差异的结构之间频繁切换，RPU内部的运行控制部件的上下文缓存设置上下文自动切换标志位，设置该标志为之后，控制单元则会在当前执行上下文和下一执行上下文之间进行自动的切换。如此，则可在重构开销不可见的情况下实现数据流计算结构的频繁改变。

差异较小的上下文快速切换另一个重要用处在于实现动态部分重构，即在数据通路流水线计算过程中，当数据流结构需要改变时，往往流水线计算过程需要消除数据相关而等待流水线流空，当流水线较长时，等待的周期会较多。此时可通过上下文的微小差异逐渐改变数据流结构，即实现两个执行上下文的流水线流空过程和充满过程的部分重叠。在执行上下文切换过程中，功能单元可不暂停计算过程，实现一部分功能单元进行计算而另一部分功能单元进行重构的部分动态可重构计算过程(Dynamical and Partial Reconfigurable)。

### 4.5.7 工具链

无

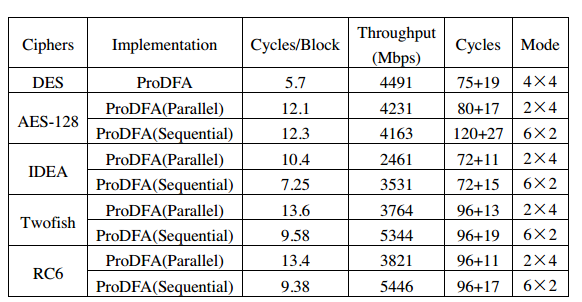
### 4.5.8 实现性能

支持算法：块密码算法

实现算法：DES、AES、IDEA、Twofish、RC6

架构在0.13um下实现。对五种块密码算法进行了实现分析。

表4.5.1 算法映射性能



### 4.5.9 参考文献

[1] Yan, Ming, et al. "ProDFA: Accelerating Domain Applications with a Coarse-Grained Runtime Reconfigurable Architecture." ICPADS. 2012.

[2]严明.面向领域应用的异构多核SoC系统结构设计与优化[D].国防科学技术大学,2011.DOI:10.7666/d.d159846.

[3] 周光华.粗粒度可重构体系结构建模方法研究与实现[D].国防科学技术大学,2010.DOI:10.7666/d.d138539.

* 1. **RCPA**

### 4.6.1 架构概述

RCPA（reconfigurable cipher processing architecture）是由解放军信息工程大学研究提出，最初发表的论文[1]发表于2007年ASIC会议。研究论文参考[1~4]。

RCPA的功能阵列支持各种逻辑操作，比特变换，S盒置换，模运算，适合分组密码算法，并且可以静态和动态配置，和GPP相比吞吐率性能是其10~20倍，和其他的可重构系统相比有着1.1~1.5倍的性能。

该架构写成RTL代码形式，并且完成了两种验证方式FPGA验证以及0.18um工艺的ASIC验证。

RCPA的优势：

流水深度可以设置，对于实现不同的算法有着更大的灵活性。

### 4.6.2 整体架构

RCPA的整体结构图示如下：

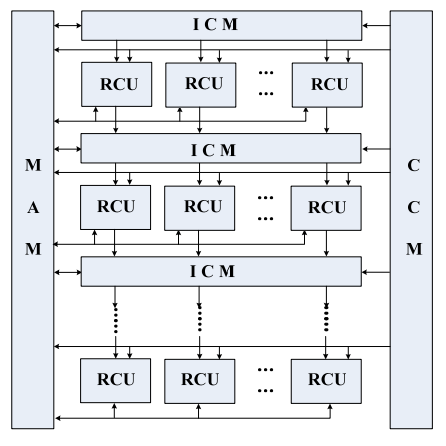


图 4.6.1 RCPA的整体架构图

RCPA包括可重构的计算单元RCU，互连模块ICM，存储器访问模块MAM，配置控制模块CCM。

1. CCM控制RCPA的所有配置和控制信息，包括内部模块的配置和内部模块之间的数据交换和输入输出端口。
2. ICM完成RCU之间的数据交换以及MAM和RCU之间的互连。
3. MAM存储加密算法中的密钥、常数以及临时值。
4. RCU是计算处理模块。可以并行连接，串行连接和串并行混合连接。

RCPA可以进行数据块内的并行计算和数据块间的流水，可以有可变化的位宽和可变化的流水深度。其组织形式如下图4.6.2所示。

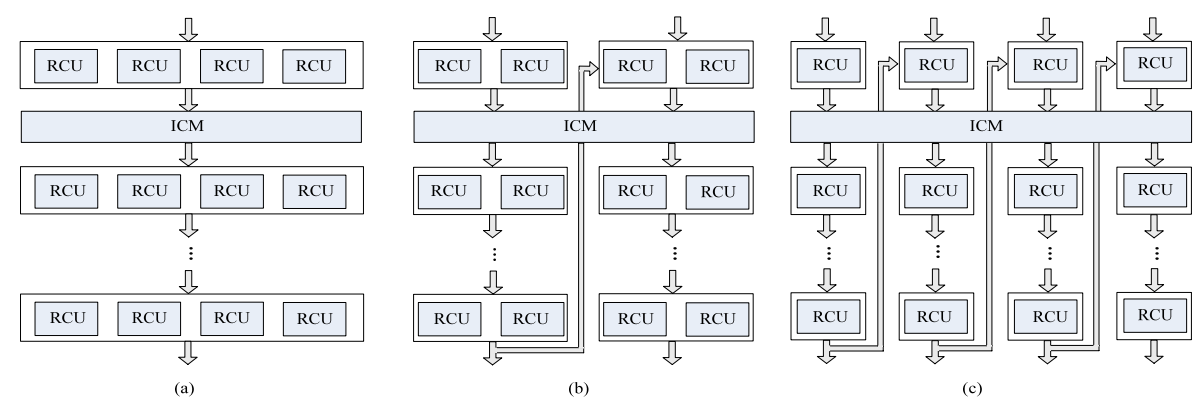


图 4.6.2 RCU的串并连接方式

其可设置的参数有：

N：数据块的长度

K：每一轮的操作数

r：算法的轮数

R：每一行RCU的个数

Cr：RCU的行数

Ci：ICM的个数

据此来实现RCPA执行算法时候流水深度的选择，从而达到性能的最优化。

### 4.6.3 PE微结构

RCU是计算处理模块，32-bit位宽，可以并行计算也可以串行连接，每一个RCU需要完成的操作包括：

1. 逻辑操作
2. 移位
3. 比特置换
4. S盒置换
5. 模加模减
6. 模乘
7. GF域的乘法

并行连接设计简单易于实现，并且有更好的延迟性能，串行连接需要结合密码算法的处理特点，对算法处理顺序进行分析，找出最佳顺序，每次RCU可执行多种运算，有着更高的执行效率，但是使得内部电路延迟和路径延迟达到了最大，降低了整体大的最高运算时钟频率。在平衡路径延时和实现多种操作时候RCPA选择串并联混合连接方式，如图4.6.3所示，通过MUX来选择需要执行的操作。

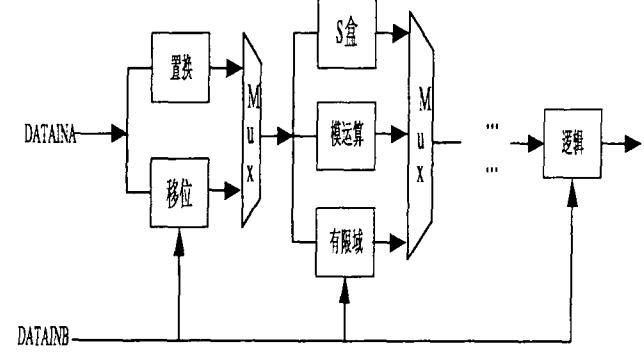


图 4.6.3 RCU串并混合连接方式

### 4.6.4 互联结构

ICM用来实现各级RCU之间的数据传输以及存储模块与RCU之间的互连。采用benes、omega-flip以及LPS型交换结构。

1. Benes交换网络

网络可以通过开关状态的改变实现 N× N 的任意交换。因此可以利用它来构造能完成任意比特置换的可重构比特置换单元。应用于比特置换操作时，这种网络的特点就表现为可以产生各个开关的控制信息，实时改变各级节点开关的状态(直通或交叉)，实现 N！种置换中要求的任何一种。其结构图如下图4.6.4所示。参考文献见[13]。

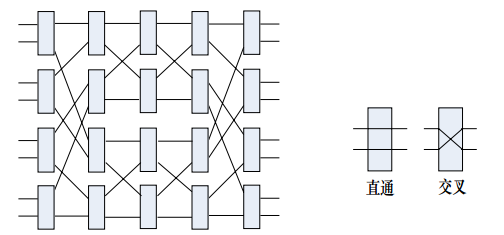


图4.6.4 8\*8benes交换网络

1. LPS交换网络

LPS网是一种链间函数一直为左混洗变换(LPS Left Perfect Shuffle)的可重排无阻塞交换网。参考文献见[6]。

左混洗变换的数学定义为:输入一组元素Ik(k=0,1,…,N-1),N=2^m,其输出为

Ok’(k,=0,l,…,N-l),k和k’满足

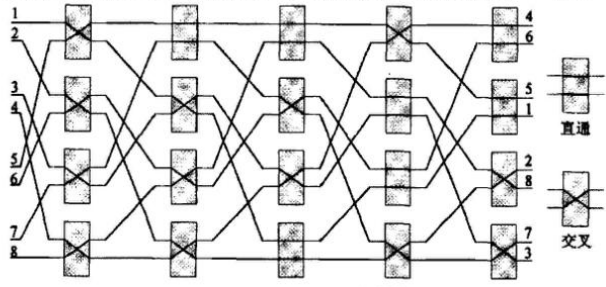


图 4.6.5 LPS交换网络结构

1. Omega交换网络

Omega交换网络是一种阻塞性的网络，不能够保证在一个源点到一个终点的路径上的开关不会发生并用。改进后的flip交换网络，可以达到无阻塞性。两种交换网络的结构图如下图示，参考文献见[5,7~12]。

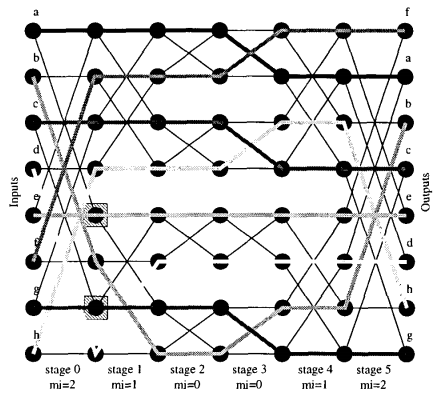


图 4.6.6 omega交换网络结构

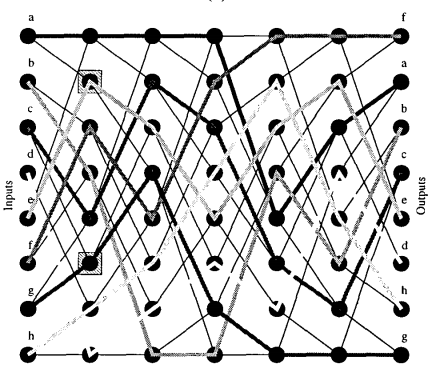


图 4.6.7 omega-filp交换网络结构

### 4.6.5 存储系统

MAM负责存储加解密算法中的密钥、常数以及运算中的临时数据。

### 4.6.6 配置机制

配置指令由上层编译产生，存储在指令存储单元，由指令译码单元译码注入到相应的模块中（ICM/MAM/RCU）。RCU和ICM的输出端插入了可配置的寄存器。CCM先根据算法的需要向RCU/ICM以及MAM注入静态配置信息，写入相应的配置寄存器，在算法执行过程中，需要频繁改变的运算功能部件RCU和ICM，在配置信息不大的情况下使用CCM来动态配置。

CCM的结构图大致如下：

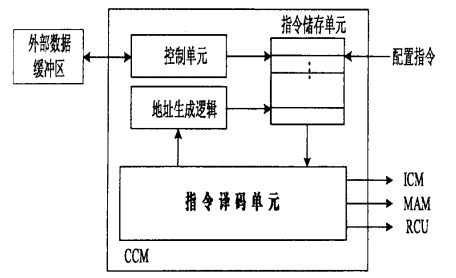


图 2.4.8 CCM框图

配置控制指令采用超长指令字VLIW/EPIC结构。

1. 控制单元发出读使能有效同时地址生成逻辑产生读地址，配置指令从指令存储单元读出。
2. 指令译码单元对指令进行译码产生ICM以及MAM所需配置信息，并注入到相应模块中。在执行算法之前，CCM需要对部分可重构密码处理单元RCU或互连模块ICM进行静态重构，算法执行时，CCM根据算法的要求对RCU和ICM注入部分配置信息进行动态重构。

### 4.6.7 工具链

无

### 4.6.8 实现性能

支持算法：分组密码算法

实现算法：AES

实现结果如下：

1. 平台：Altera’s Stratix II EP2S180F1020I4

性能：最大频率达到47.74 MHz，占用运算资源32863个ALUT，占用458752bits存储资源，32个DSP slices

1. 平台：0.18um CMOS

性能：最大频率达到180MHz，面积14899092um2，相当于1489.9Kgates，关键路径延迟5.55ns。

### 4.6.9 参考文献

1. Dai, Zibin, et al. "The research and design of reconfigurable cipher processing architecture targeted at block cipher." ASIC, 2007. ASICON'07. 7th International Conference on. IEEE, 2007.
2. 杨晓辉. 面向分组密码处理的可重构设计技术研究[D].解放军信息工程大学,2007.
3. 杨晓辉,戴紫彬,张永福. 可重构分组密码处理结构模型研究与设计[J]. 计算机研究与发展,2009,06:962-967.
4. 刘全果,戴紫彬. 可重构分组密码处理结构的研究与设计[J]. 电子技术,2008,05:40-43.
5. Yang, Xiao, and Ruby B. Lee. "Fast subword permutation instructions using omega and flip network stages." Computer Design, 2000. Proceedings. 2000 International Conference on. IEEE, 2000.
6. 向楠,戴紫彬,武清芳. 基于LPS交换网络的快速比特置换指令系统设计[J]. 电子技术应用,2007,02:138-141.
7. 张联. 无阻塞多播通信在Omega网上的实现[D].中国科学技术大学,2006.
8. 黄海漩,徐平,李孟珏,周育成,杨拓,张旭琳. 多级级连omega光交换网络路由算法研究[A]. 中国光学学会.中国光学学会2011年学术大会摘要集[C].中国光学学会:,2011:2.
9. 张以皓,沈越泓,蒋嵘. 基于正规序列的Omega网络路由构造方法[J]. 计算机科学,2014,01:192-195+201.
10. 魏尊策,陈实,孙济洲. 一种无阻塞的Omega网络实现方案[J]. 计算机工程,2003,03:52-53+133.
11. 艾军,曹明翠,李再光. 光互连Omega网络与榕树网络拓扑等价的多样性[J]. 光学学报,1994,07:724-728.
12. 罗风光,徐军,曹明翠,李洪谱,艾军. 光计算中全排列无阻塞双Omega光互连网络的光学实现方法[J]. 中国激光,1994,03:220-224.
13. 向楠,戴紫彬,徐劲松. 一种基于BENES网络的可重构比特置换系统设计[J]. 计算机工程,2007,22:178-180.

# 算法在可重构架构上的性能汇总

5.1 AES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 架构 | 工艺（nm） | 频率（MHz） | 性能（Mbps） | 面积（mm2） | 功耗（mW） |
| celator |  | 190 | CBC:47  EBC:46 |  |  |
| CoARX |  |  |  |  |  |
| COBRA | 350 | 102 | 1451.25 |  |  |
| cryptoraptor | 45 | 1000 | CBC:6400  CTR:128000 | 6.32 | 0.977\*6.32 |
| Virtex 6:40 | 203.8 | CBC:1300  CTR:26090 |  |  |
| ProDFA | 130 | 400 | 4231 | 0.72 |  |
| RCPA | 180 | 180 |  | 14.9/1489.9Kgates |  |
| Altera’s Stratix II EP2S180F1020I4 | 47.74 |  | 32863LUT |  |

5.2 DES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 架构 | 工艺（nm） | 频率（MHz） | 性能（Mbps） | 面积（mm2） | 功耗（mW） |
| celator |  | 190 | 26 |  |  |
| CoARX | 90 |  |  |  |  |
| COBRA |  |  |  |  |  |
| cryptoraptor | 45 | 416.7 | CBC:2670  CTR:42670 |  |  |
| ProDFA | 130 | 400 | 4491 | 0.72 |  |
| RCPA |  |  |  |  |  |

5.3 SHA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 架构 | 工艺（nm） | 频率（MHz） | 性能（Mbps） | 面积（mm2） | 功耗（mW） |
| celator |  | 190 | 36 |  |  |
| CoARX |  |  |  |  |  |
| COBRA |  |  |  |  |  |
| cryptoraptor | 45 | 625 | 1600 |  |  |
| ProDFA |  |  |  |  |  |
| RCPA |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 架构名称 | 支持算法 | 工具链支持 | 标志文献 | 提出时间 | 架构形式 | 研究机构 |
| COBRA | 各种对称密钥算法 | 有：包括一个汇编器和一个时序分析器 | Parallel and Distributed Processing Symposium | 2003 | 轮迭代结构 | 德国的University of Massachusetts Lowell |
| celator | AES/DES/SHA | 无 | RCFPGA论文 | 2008 | 轮迭代结构 | 埃克斯马赛大学 |
| RCPA | 多种算法 |  | ASIC会议 | 2007 | 轮展开阵列结构 | 解放军信息工程大学 |
| ProDFA | 块密码算法 | 无 | ICPADS | 2012 | 轮展开阵列结构 |  |
| cryptoraptor | 多种算法：AES Blowfish, Camellia, CAST-128, DES, GOST, Kasumi, RC5, SEED, and Twofish | 自动配置工具 | ICCAD论文 | 2014 | 轮展开阵列结构 | 德克萨斯州大学奥斯汀分校 |