学　　号：	10520061010
密　　级：	公开
中图分类号：	TP393
学科分类号：	510.503

解放军信息工程大学

博士学位论文

高速串模式匹配算法研究

作者姓名：	姜鲲鹏
指导教师姓名：	兰巨龙教授
学科门类：	工　学
学科专业：	通信与信息系统
研究方向：	宽带信息网络
论文提交日期：	2012 年4 月15 日
论文答辩日期：	2012 年6 月15 日

解放军信息工程大学

信息工程学院

二〇一二年 四月

A Dissertation Submitted to
PLA Information Engineering University
for the Degree of Doctor of Engineering

Research on High-Throughput string patterns matching algorithm

Candidate: **Jiang Kunpeng**

Supervisor: **Prof. Lan Julong**

Apr. 2012

原创性声明

本人声明所提交的学位论文是本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和撰写过的研究成果，也不包含为获得信息工程大学或其他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谢意。

学位论文题目：高速串模式匹配算法研究

学位论文作者签名：_____ 日期：　　年　　月　　日

作者指导教师签名：_____ 日期：　　年　　月　　日

学位论文版权使用授权书

本人完全了解信息工程大学有关保留、使用学位论文的规定。本人授权信息工程大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：高速串模式匹配算法研究

学位论文作者签名：_____ 日期：　　年　　月　　日

作者指导教师签名：_____ 日期：　　年　　月　　日

目　录

表目录

图目录

摘 要

模式匹配不仅是计算理论的基础，而且在计算机和网络处理中，有着广泛地应用。随着信息爆炸及网络带宽的迅速增加，无论是信息查询的需要还是网络安全的需求，线速地处理网络数据成为了必然要求。本文对此进行了比较详细的研究与实践，主要贡献和创新点包括：

1. 针对研究 NFA(非确定型有穷自动机)算法时无数学证明的问题。提出了六个定理，一方面，使用这些定理，形式化地证明了本文提出的算法正确性、算法与 NFA 等价性；另一方面，迄今为止，在公开发表的文献中对于 NFA 的证明都是说明性的，没有适合的数学公式证明。本文提出了数学定义、定理，为 NFA 的证明初步建立了数学基础；最后，由于数学公式的建立，为以后研究 NFA、NFA 推导打下了基础。
2. 提出了一种新的固定模式串匹配算法。通过设计新的解码矩阵对正文的解码和模式比较结果的提取这两种操作，一方面减少了正则表达式实现时，大量模式符和输入正文字符间的比较操作对资源的大量占用，另一方面加快了大量模式符和输入正文字符之间比较操作的运算速度。解决了模式匹配时大量比较运算资源占用高和运行速度慢的难题。
3. 提出了“向量与”算法。这个算法把正则表达式中连接运算变为简单的逻辑与操作。正则表达式中最常用的操作是连接运算。每个正则表达式中都使用了很多连接操作，因此加快连接操作的运算速度对提高正则表达式的运算速度有着至关重要的意义。本文提出的算法将正则表达式中的各种运算全部转化为最简单的逻辑运算，从而提高了算法的整体运算速度。
4. 本文提出的算法不仅支持正则表达式的常用运算，而且很好地支持补运算。补运算一直是正则表达式实现算法中的一个难题。公开文献中要么很少提及补运算的实现，要么所有提及的算法都是以大量资源占用的方式来实现补运算。本文把补运算转化为常用的连接运算，解决了正则表达式中补运算实现的难题。
5. 提出了扩展解码矩阵方案。利用这个方案解决了正则表达式中单个模式符之间的并、交、非运算。

通过在 FPGA（现场可编程门阵列）上验证本文提出的算法，结果表明：本文提出的算法充分地利用了 FPGA 的特性，提高了正则表达式的吞吐率。迄今为止，公开文献中给出的最大吞吐率都在 40Gbps 之下，有资料可查的商用最大吞吐率不超过 20Gbps。在本文最后一章给出了本文提出的算法能达到的最大吞吐率。通过实验表明，在现有 FPGA 中本文提出的算法的吞吐率可以达到 512Gbps。这个吞吐率已经远远超过现代网络中最大带宽。另外，由于本文提出的算法仅使用 FPGA 逻辑资源，不使用其它资源，因此也不会受 FPGA 与外部设备之间连线及传输的影响。同时也增加了系统的可扩展性，可通过使用更多资源

的 FPGA 和增加 FPGA 数量的方式增加整个系统中实现的模式符数量。

关键词：模式匹配，高速，固定串匹配，正则表达式，非确定型有穷自动机，现场可编程门阵列

ABSTRACT

Pattern matching not only is the basis of computing theory, but also has been widely applied in computer system and network processing. With the information explosion and the rapid increase in network bandwidth, the requirement of both information query and network security are bound of wire-speed processing of network data. Detailed researches are done about pattern matching in this thesis. The main works and innovations are as the following:

1. A series of mathematical formula was established to solve the problem of mathematical proof for the NFA algorithm studying. On the one hand, the mathematical formula proved the correctness of our algorithm, and the equivalence between the proposed algorithm and NFA. On the other hand, so far, the proofs of NFA in published literature were illustrative, and had not established the appropriate mathematical formula. This paper presented mathematical definitions, theorems which establish the preliminary basis for mathematical proof of the NFA. Finally, due to the establishment of mathematical formula, the article laid the foundation for the establishment and derivation of the new NFA.

2. The paper proposed a novel static pattern matching algorithm. The algorithm designed two operations which are decoding matrix decoding the text and the extraction of pattern comparison results. On the one hand, the operations reduced the resources occupied by the regular expression to achieve the comparison operation between a large number of pattern symbols and text symbols. On the other hand, the operations accelerated the computational speed of the comparison between a large number of pattern symbols and text symbols. The operations resolved the problem of high resource consumption and running slow of a large number of comparison operations.

3. The paper proposed the "vector-and" algorithm. The algorithm transformed the concatenation of regular expressions into a simple bitwise and operation. The most commonly operation of regular expressions is the concatenation. Each regular expression includes a lot of concatenation operations, and therefore to accelerate the computing speed of the concatenation operation plays an important role to improve the regular expression operation speed. The proposed algorithm expression transformed all of the operations of regular expression into bitwise operations, thus greatly improved the overall computing speed.

4. The proposed algorithm supports not only the common operations of regular expression, but also the complementation operation. Complementation operation has been the puzzle of kinds of regular expression algorithms. The article transformed the complementation into the common concatenation by the appropriate structure, therefore, resolved the puzzle of the

complementation operation in regular expression implementation.

5. The paper proposed extended decoding matrix. The scheme completely resolved the union, intersection, and complementation operations of single symbol of regular expression.

Established a set of mathematical formulas, and proposed novel algorithm, and verify the novel algorithm on the FPGA, the article showed that: The proposed algorithm made full use of FPGA features, and greatly increased the regular expression throughput. So far, the maximum throughput in the open literature is less than 40Gbps, and the maximum throughput of commercial implementation does not exceed 20Gbps. The final chapter in this article showed the maximum throughput of the proposed algorithm. The experiment showed that: the throughput of the proposed algorithm can be 512Gbps on the existing FPGA. The throughput has been far more than the maximum bandwidth of modern networks. In addition, the proposed algorithm used only the FPGA logic resources, do not use other resources, and therefore was not affected by the connection and the transmission between the FPGA and the external devices, also increased the scalability of the system, and can increase the number of the pattern symbols in the entire system by using more resources FPGA and increasing the FPGA number.

Key words: Pattern matching, High throughput, Static pattern matching, Regular expression, Nondeterministic Finite Automata, Field Programmable Gate Array

第一章 引 言

1.1 研究意义

一方面，无论是私人的商业、贸易、服务、运输和制造业，还是公共的医疗、政府，甚至是国家经济、防务都大量使用计算机系统，都需要传输大量数据。对网络的依赖性越来越强，有些行业甚至无法离开网络而存在。这些需求导致当前网络带宽飞速发展，链路带宽从100Mbps、1000Mbps迅速发展到10Gbps，40Gbps的国际链路已经开始应用，实验性100Gbps的链路已经开始出现。另一方面数字信息系统的安全性对现代社会、现代经济以及国家安全有着越来越重要的意义。信息最大的价值在于其安全性，网络安全性日益成为了社会关注的关键性问题。网络运行环境变得越来越恶劣，网络蠕虫、病毒、垃圾邮件、DOS攻击、恶意访问等层出不穷。这些系统中发生的信息或网络安全事件经常导致巨大的经济灾难。这些年的分析显示网络安全对经济的影响越来越重大，详情见表1。据评估，2007年之前，世界范围内数字攻击就已经产生了十亿美元以上的损失[1,2,3]，2009年时报道[4]称世界范围内的这个数字评估为八百六十亿美元。安全软件公司McAfee的一个报告显示2010年仅在工作时间内由于病毒攻击导致经济损失就已经达到每天6.3百万美元。2011年十月止已发现的攻击数量超过了2亿次。为了保证网络安全，出现了各种网络安全设备。最常用的基于包头处理的防火墙，已经无法满足现代网络安全需求。基于包内容的网络入侵检测系统（IDS）和入侵防护系统（IPS）得到大量应用。传统的IDS与IPS解决方式为把可能包含恶意的网络数据包转发到高性能服务器，而后由服务器进行深度包识别（DPI）来识别包中的数据，根据识别结果对输入的包进行相应的处理。在1000Mbps以下的网络中，这种工作方式能起到很好地作用。但随着链路带宽的进一步增加，服务器直接处理网络流量的工作方式已经无法满足链路处理的时延要求。还有很多系统的核心任务就是高速模式匹配。如：电子邮件检测系统（ClamAV[5]）、应用层过滤[6]等。还有一些应用，如：基于内容的路由[7]、出版/评阅网络[8,9]和语义网络[10]等，也需要线速包内容检测的支持。

由于网络中恶意行为的泛滥，这些系统中规则的复杂度日益提高，例如在2004年SNORT2.2时，规则集中静态模式串的数量为1631个，正则表达式模式串的数量只有157个，到2007年SNORT2.6时，规则集中静态模式串的数量为2927个，正则表达式模式串的数量有1687个，到最近SNORT2.9时，规则集中静态模式串已经超过了50000个，同时正则表达式的数量也超过了20000个。

另外，Gilder[11]发现网络带宽与计算能力之间巨大缺口。一方面，众所周知的摩尔定律[38]：计算能力每八个月翻一翻，另一方面，网络带宽每六个月翻一翻。

图1显示出与上面提到的一些相关事实。上世纪九十年代至今，广域网带宽每年大约增加四倍，接入带宽每年大约增加两倍，而单芯片的处理器计算能力每年增长速度不超过1.6倍。在过去的二十年里，广域网带宽从几百Kbps增长到了几十Gbps。接入带宽在早些年与广域网同步增长，2000年后，两者的差距才越拉越大。主要原因是接入网一直使用有线

和无线的电信号进行数据传输，而广域网从开始的电信号转为现代的光信号进行数据传输。因此接入带宽以每年1.5倍左右速度增加，广域网带宽每年以4倍左右速度增加。特别是近几年，单核中用做高主频方式提高计算能力的方式由于受到光速的影响而不能进一步提高系统主频，转而向多核发展，以期望能进一步增加计算能力。同时，由于受到编译发展的影响，多核应用也无法完全发挥多核的计算能力，导致应用无法完全利用芯片中的计算能力，因此计算能力的提高速度与网络带宽速度之间差距进一步扩大。2000年以后无论是广域网与接入网之间带宽差距，还是接入网带宽与计算能力之间差距都越来越大。

由此可知，带宽增加速度是计算能力增加速度的三倍。根据以上的说法，网络处理任务需要更多的计算能力，网络处理引擎所需的计算能力高于现有计算机系统所能提供的计算能力。由硬件来直接进行深度包解析成为了必然的选择。

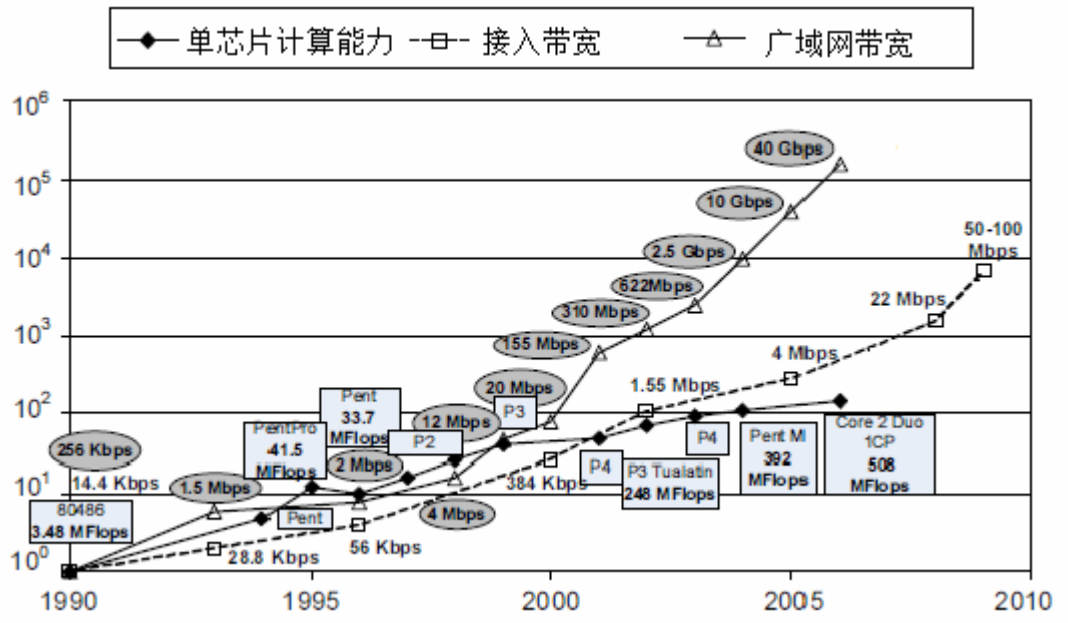

图 1[11] 单芯片计算能力、接入网带宽和广域网带宽发展图

无论是基于包内容的网络入侵检测系统（IDS）和入侵防护系统（IPS），还是电子邮件检测系统（ClamAV[5]）、应用层过滤[6]、基于内容的路由[7]、出版/评阅网络[8,9]和语义网络[10]等应用，都需要线速包内容检测的支持。这些系统与应用的基础就是模式串匹配。现阶段深度包识别面临的主要难题是在满足链路带宽条件下，需要处理大量模式匹配。例如：在 SNORT　IDS 规则集[12,13]、Bro[14]等系统中，每一个模式代表一种恶意流量。所有到达的数据包都要与系统中的所有规则进行匹配，每一个成功的匹配都会触发一种预定义的行为对到达的数据包进行相应地处理。

由以上的讨论可知，信息安全问题对经济与社会有着巨大影响，同时随着网络带宽的迅速增加，信息安全问题越来越严重。解决信息安全问题需要大量计算能力，仅靠计算机芯片计算能力的提高已经不能满足信息安全问题需要。因此，为了满足现在与将来信息安全的需要，需要开发出更高的计算能力。由硬件来直接进行深度包解析成为了必然的选择。

正如上面所说，现代网络防护系统的核心算法是模式匹配，怎样在 40Gbps 以上的链

路上支持模式匹配成为了关键问题。

表 1[1,2,3,4] 世界范围内网络攻击对经济产生的影响(十亿美元)

年份	Eleanor McKenzie	Computer Economics	Mi2g
2010	6.3/天		
2009	86/年		
2007	13.3/年		
2006		13.3/年	NA
2005		14.2/年	NA
2004		17.5/年	NA
2003		13.0/年	185-226/年
2002		11.1/年	110-130/年
2001		13.2/年	33-40/年
2000		17.1/年	25-30/年
1999		13.0/年	19-23/年
1998		6.1/年	3.8-4.7/年
1997		3.3/年	1.7-2.9/年

在高速网络中实现正则表达式匹配主要需求为：一、要满足链路的线速处理。随着网络的发展，现代网络带宽早已超过 1000Mbps，达到了 40Gbps，也就要求所设计的模式匹配速度要达到 40Gbps 以上才能满足链路需求，同时必须保证最差的时延也不能大于链路的要求；二、要满足并行处理的模式数量要求。现代网络中面临着越来越多的主动、被动攻击以及用户的各种需求，无论是网络中常用的 IDS 系统（SNORT、Bro、Linux 7 层过滤），还是具体的用户需求都需要成千上万条以上的模式串，才能达到系统的要求。三、要求得出所有匹配结果。实际运行中对于待匹配的成千上万条模式，不但要求输出是否有匹配，还要求输出所有匹配成功的模式。这就要求“同时”输出所有与正文相匹配的模式串。甚至更进一步要求输出在一个正文中对某个模式匹配的数量。

1.2 研究内容

模式匹配不但是计算理论的基础，而且研究模式匹配，特别是研究正则表达式可以为此后的文法描述及图灵机的研究打下坚实的基础。迄今为止，公开文献中对正则表达式匹配算法的构造以及证明都使用的是构造性地说明方法，还没有建立适合的数学公式来证明，缺乏数学地证明，导致对其后的下推自动机、上下文无关文法及图灵机的证明也是构造性的。因此需要建立一套数学定义和定理来完成正确性地证明。本文通过定义一组定义与定理，建立了一套数学公式，一方面，在理论上证明了我们算法的正确性、算法与 NFA 等价性；另一方面，本文提出了数学定义、定理建立了 NFA 的数学证明的初步基础。另外，由于数学公式的建立，对新 NFA 的建立、NFA 推导打下了理论基础。

首先，现阶段，正则表达式应用在与计算机相关的各种领域内，大的方面：搜索引擎

的基础是正则表达式；数据库的实现中也大量运用正则表达式；网络安全中网络入侵与检测系统中正则表达式的使用量也呈现出迅速增加态势。这些应用对算法的吞吐率要求很高。迄今为止，公开文献中所有给出的模式匹配最大吞吐率都在 40Gbps 之下，有资料可查的商用最大吞吐率不超过 20Gbps。已经不能满足现阶段网络防护与应用。因此本文首要研究能够满足 40Gbps 以上链路的模式匹配算法。对于固定模式串，不仅需要大量比较操作，而且需要在正文串中找出与模式串相匹配的位置。对于正则表达式的模式串，不仅需要完成固定模式串的任务，还要处理正则表达式模式串中引入的补、并、交和 Kleene 星号算子。这都需要大量计算任务。在很短的时间内完成这些计算任务对硬件算法是一个巨大的挑战。本文提出的固定模式匹配算法和正则表达式匹配算法充分地利用了 FPGA 的特性，大大提高了匹配的吞吐率。

其次在网络中不但对算法的吞吐率要求很高，另外对处理时延也有要求。因为 IPv4 网络中最小包长为 40 字节、IPv6 网络中最小包长为 60 字节，所以在 40Gbps 链路中，包与包的最小间隔分别为 8ns 和 12ns。同时现代 FPGA 支持的内部时钟最大频率为 500MHz，因此从接收到数据开始，4 到 6 个时钟周期必须完成模式匹配任务。本文提出的固定模式匹配算法和正则表达式匹配算法针对这项要求，实现了在一个时钟周期完成算法匹配。

最后，由于现在不论是公开的 SNORT、BRO、Linux7 层过滤，还是用户的实际需求，都需要单芯片内实现大规模的模式串才能满足要求，同时还要求在 4 到 6 个时钟周期内完成所有模式串匹配任务，并生成相应的结果。这就要求设计的算法，不仅要能满足时延要求，而且还要求每个模式符的资源占用率要尽可能的低。

综上所述，本文设计的算法要求满足吞吐率在 40Gps 以上，最大时延不超过 4 个时钟周期，单芯片内至少要实现上千个模式符。针对这些现实需要，本文通过研究固定模式串和正则表达式的特点，结合前人研究成果，一方面优化需要大量计算的正文字符与模式符的比较运算，把比较运算优化为更简单、快速的字符解码；另一方面优化连接、并、交、补和 Kleene 星号运算，把这些运算优化为最简单的逻辑与、或和非运算；最后使用 NFA 技术并行处理多个正文字符满足算法时延要求。

1.3 主要贡献

针对在高速网络中实现正则表达式匹配主要需求：一、要满足链路的线速处理；二、要满足并行处理的模式数量要求。本文在前人研究的基础上，研究不使用外部资源，仅使用 FPGA 逻辑资源完成正则表达式匹配的算法。取得了以下成果：

1. 针对研究 NFA(非确定型有穷自动机)算法时无数学证明的问题。提出了六个定理，一方面，使用这些定理，形式化地证明了本文提出的算法正确性、算法与 NFA 等价性；另一方面，迄今为止，在公开发表的文献中对于 NFA 的证明都是说明性的，没有适合的数学公式证明。本文提出了数学定义、定理，为 NFA 的证明初步建立了数学基础；最后，由于数学公式的建立，为以后研究 NFA、NFA 推导打下了基础。

2. 提出了一种新的固定模式串匹配算法。通过设计新的解码矩阵对正文的解码和模式比较结果的提取这两种操作，一方面减少了正则表达式实现时，大量模式符和输入正文字符间的比较操作对资源的大量占用，另一方面加快了大量模式符和输入正文字符之间比较操作的运算速度。解决了模式匹配时大量比较运算资源占用高和运行速度慢的难题。
3. 提出了“向量与”算法。这个算法把正则表达式中连接运算变为简单的逻辑与操作。正则表达式中最常用的操作是连接运算。每个正则表达式中都使用了很多连接操作，因此加快连接操作的运算速度对提高正则表达式的运算速度有着至关重要的意义。本文提出的算法将正则表达式中的各种运算全部转化为最简单的逻辑运算，从而提高了算法的整体运算速度。
4. 本文提出的算法不仅支持正则表达式的常用运算，而且很好地支持补运算。补运算一直是正则表达式实现算法中的一个难题。公开文献中要么很少提及补运算的实现，要么所有提及的算法都是以大量资源占用的方式来实现补运算。本文把补运算转化为常用的连接运算，解决了正则表达式中补运算实现的难题。
5. 提出了扩展解码矩阵方案。利用这个方案解决了正则表达式中单个模式符之间的并、交、非运算。

在建立了一套数学公式和完成新的算法设计后，在 FPGA 上验证新的算法表明：新算法充分地利用了 FPGA 的特性，大大提高了正则表达式的吞吐率。迄今为止，公开文献中给出的最大吞吐率都在 40Gbps 之下，有资料可查的商用最大吞吐率不超过 20Gbps。在本文最后一章给出了本文提出的算法能达到的最大吞吐率。通过实验，在现有 FPGA 中验证了本文提出的算法的吞吐率可以达到 512Gbps。这个吞吐率已经远远超过现代网络中最大带宽。另外，由于本文提出的算法仅使用 FPGA 逻辑资源，不使用其它资源，因此也不会受 FPGA 与外部设备之间连线及传输的影响。同时也增加了系统的可扩展性，可通过使用更多资源的 FPGA 和增加 FPGA 数量的方式增加整个系统中实现的模式符数量。

1.4 论文整体框架

第二章简单介绍了几个经典模式匹配算法，这些算法是研究模式匹配的基础。

第三章给出了本文算法研究的基础约定、和定义，提出和证明了定理，建立了本文研究的数学基础，给出了三种基本模块。随后提出一种使用三种模块构建、基于 FPGA 的新型固定模式串匹配算法，并用上述数学定理证明了算法的正确性。

第四章在第三章研究的基础上，提出一种使用上述三种模块构建的新型正则表达式匹配算法，并用上述数学定理证明了“向量与”算法对于任意一个正则表达式都可以构造一个实例接受该正则表达式，并用上述数学定理证明了算法的正确性。其次证明了“向量与”算法与非确定自动机是等价的，因此也就证明了“向量与”算法和正则表达式是等价的。

第五章给出了三种针对“向量与”算法的改进算法。第一种算法，针对模式符与正文符的比较运算进行了改进，节约了大量资源，降低了模式串中每个模式符的资源占用率；

第二种算法，针对模式串中最常用的连接运算进行了改进，同样节约了资源，降低了模式串中每个模式符的资源占用率；最后一种算法，针对正则表达式中补运算进行了改进，给出了完善后的支持正则表达式补运算实现方案，使特殊的补运算变换为常用的连接运算。解决了正则表达式中补运算实现的难题。

第六章首先对“向量与”算法进行了性能分析，给出了时钟频率和吞吐率的理论值，随后通过实验的方式给出了在现有 FPGA 中实现算法时的实验时钟频率和吞吐率。在一定程度上验证了第一节中给出的时钟频率和吞吐率的理论值。

第二章 经典模式匹配算法简介

本章对几种经典匹配算法进行简单介绍。这些算法是研究模式匹配的基础，为以后的研究提供参考与借鉴。多年来，众多的人都是从研究这些算法开始，仍然有许多人对这些算法进行改进。其中 KMP 算法是固定模式匹配的经典算法，其思想仍然在现代算法中时有体现，特别是由它引入的模式串预处理思想，现在几乎所有模式匹配算法都在使用。Aho-Corasick[15]算法中共同子串的思想也是大量被后续算法所使用。最后 Clark 的字符解码思想大大减少了 FPGA 中资源占用。

2.1 KMP 算法简介

KMP算法[16]是由Knuth－Morris－Pratt共同开发出来的，是固定模式匹配经典算法，也是研究模式匹配算法的基础。这个算法把固定模式匹配的时间复杂度缩小到O（m+n）,而空间复杂度也只有O(m),n是正文的长度，m是模式串的长度。在此算法在发明之前并不是没有如此高效的算法，但是原算法比较复杂。KMP算法优雅高效，但是实现却不难理解且代码长度很短，是优秀算法设计的典范。下面根据[16]对算法思想做非正式地简单介绍。

通过举例说明，把模式串与正文串放在一起，并把模式串按正确的方法沿着正文串从左到右滑动，我们就能很容易理解这种方案的思想。考虑下面这个例子，模式串是“abcabcacab”，正文串是“babcbabcabcaabcabcabcacabc”。

首先让模式串的左端与正文串左端对齐，而后开始匹配过程：

```
a b c a b c a c a b
b a b c b a b c a b c a a b c a b c a b c a c a b c
↑
```

图 2 KMP 算法开始匹配

箭头指示当前要比较的正文字符。因为它指向b，与模式串中a不匹配。沿着正文串把模式串向右移动一个位置。

```
  a b c a b c a c a b
b a b c b a b c a b c a a b c a b c a b c a c a b c
  ↑
```

图 3 KMP 算法匹配第 2 个字符

现在一次匹配出现，于是不再移动模式串，继续匹配下面几个字符，直到出现一个失配。

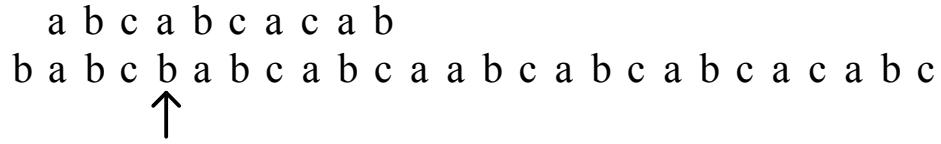

图 4 KMP 算法匹配第 5 个字符

在这个位置上前三个字符模式字符与正文字符匹配，但第四个失配，于是知道正文最近的四个字符是“abcx”，且x ≠a。因为根据模式串的位置，有足够的信息可以重新构建出前面扫过的正文字符，所以不需要记忆那些扫过的字符。在这个例子中，不论x是什么，只要它不是a，模式串就可以立即向右移动四个字符位置，因为前三个位置也不可能是模式串与正文串发生匹配的开始位置。

移动后，又开始了部分匹配，这时失配会发生在模式串中的第八个字符。

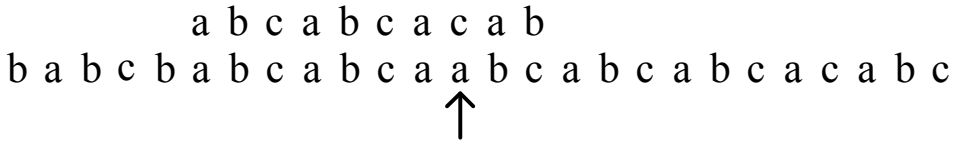

图 5 KMP 算法匹配第 13 个字符

现在，最近的八个正文字符是“abcabcax”，且x ≠c。因此模式串需要向右移动三个位置。

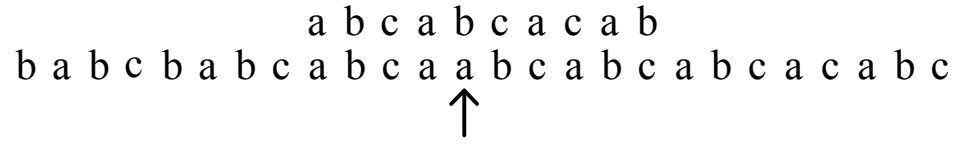

图 6 KMP 算法匹配第 13 个字符且滑动模式串

在新的位置上匹配模式字符。同样会失配，于是模式串需要再次向右移动四个位置。移动后，又开始了部分匹配，这时另一次失配同样会发生在模式串中的第八个字符。

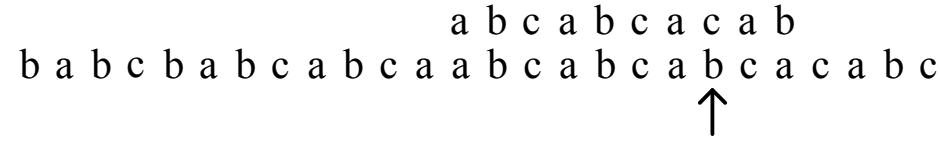

图 7 KMP 算法匹配第 20 个字符

由于同样的原因，模式串需要再一次向右移动三个位置。但这一次匹配出现了，并且最终发现了完整的模式匹配。

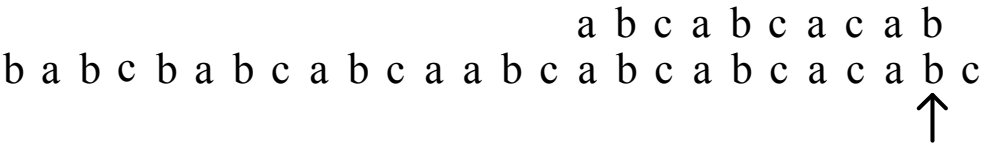

图 8 KMP 算法匹配完成

图2到图8描述了模式处理是有效的。这里需要附加一个表2，来告诉我们在检测到模式串的第j个字符失配时，应该滑动多少个模式字符。令next[j]为当发第j个字符发生失配时，下一个比较的模式串中模式字符的位置。这样，模式串相对于正文串应该滑动j-next[j]个字符。下面是例子中next[j]的列表：

表 2　KMP 算法 Next 示例

j	1	2	3	4	5	6	7	8	9	10
pattern[j]	a	b	c	a	b	c	a	c	a	b
next[j]	0	1	1	0	1	1	0	5	0	1

需要注意的是：next[j]=0表示把模式串中所有字符滑动到当前位置的右侧。

有关next[j]的计算是：首先处理一种简单的情况，即在定义next[j]时不考虑pattern[i]≠pattern[j]。令f[j]等于在pattern[1]…pattern[i-1]=pattern[j-i+1]…pattern[j-1]条件下的小于j的最大的i。因为这种条件下，i的最小值是1，因此总是有当j>1时，f[j]≥1。约定f[1]=0。对于例子而言，f函数取值为表3所示：

表 3　KMP 算法 f 函数示例

j	1	2	3	4	5	6	7	8	9	10
pattern[j]	a	b	c	a	b	c	a	c	a	b
f[j]	0	1	1	1	2	3	4	5	1	2

f[j]的计算有以下特点：如果pattern[j] =pattern[f[j]]，则f[j+1]=f[j] +1；但如果pattern[j] ≠pattern[f[j]]，这种情况下，计算f[j+1]所用的算法本质上与前面的模式匹配算法是一样的。注意到前面计算模式匹配要求的是相对于k求出最大的i使得pattern[1]…pattern[i-1] =text[k-i+1]… text[k-1]，这里要求是相对于j求出最大的i使得pattern[1]…pattern[i-1] =text[j-i+1]… text[j-1]。因此，可以使用前面的技术来解决本问题。由于我们有初始条件：约定f[1]=0，可以假定f[1]到f[j]已经计算完成，下面的程序可以计算出f[j+1]:

```
t=f[j];
while t>0 and pattern[j] ≠pattern[t]
    do t=f[t]
f[j+1] =t+1;
```

通过以前的演示可知算法的正确性。可以设想两个一样的模式串，其中一个在另一个上滑动。例如，对于上述例子中的模式串，已经完成f[8]=5的计算，现在考虑如何计算f[9]。

如图9所示

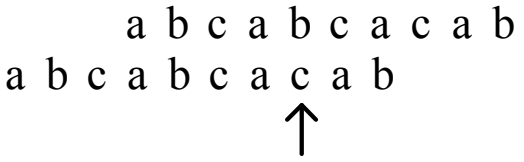

图 9 KMP 算法模式串计算 f[9]

因为pattern[8] ≠b，必须滑动上面的模式串，由最近在下面的模式串中的检测是abcax且x≠b，另外已经计算出的f函数说明向右滑动四个字符，得到图10：

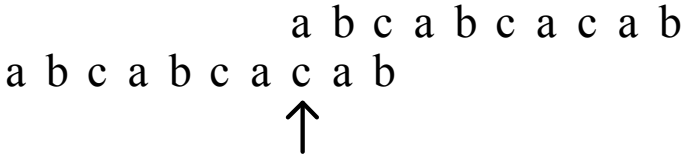

图 10 KMP 算法模式串计算 f[9]结果

这里还是不匹配，下一次滑动的结果使得t=0，于是f[9]=1。

通过以上的f函数计算过程，再一次对固定串模式匹配有一理解。然而，以上的f函数取值在j=4时且与正文失配时即模式字符是“a”，正文字符不是“a”。滑动后与正文相比的模式符还是“a”，肯定还是失配的，这次比较是不必要的，因此f还需要优化。进而可以得到next[j]的计算方法为：对于j>0，

$$next[j]=\begin{cases} f[j], & if\ \ pattern[j]\neq pattern[f[j]]; \\ next[f[j]], & if\ \ pattern[j]=pattern[f[j]]. \end{cases} \tag{1}$$

合并上面的两个计算过程，可以得到如图11所示的算法完成的next[j]的计算：

```
j := 1; t := 0; next[1] := 0;
while j < m do
    begin comment t = f[j];
        while t > 0 and pattern[j] ≠ pattern[t]
            do t := next[t];
        t := t + 1; j := j + 1;
        if pattern[j] = pattern[t]
        then next[j] := next[t]
        else next[j] := t;

    end.
```

图 11 KMP 算法 next 函数计算方法

图11算法中，首先j从1开始，最外层循环每循环一次j都增加1、直到m，内层循环中对t的计算，由于t是非负的，而且每一次t=next[t]都会减少t的值。因此next[t]的计算复杂度是 $O(m)$ 。有关正文串的处理与模式串的处理类似，第次处理一个正文字符，每一次t=next[t]都会将模式串向右移动。因此正文串的计算复杂度是 $O(n)$ 。

KMP算法实质上是首先把模式串当做正文串，对模式串进行处理，找出模式串内子串包含关系，进而生成next函数。再根据生成的结果处理正文串。现在看来KMP算法最大的贡献是利用字符串本身的特点，提出计算复杂度为$O(m+n)$的固定模式串匹配算法，另外一个贡献是提出了模式串预处理的思想，把固定模式串匹配分为了两步，第一步首先预处理模式串，第二步是在根据预处理模式串的结果，再处理正文串。最终达到正文串无回溯的目的。预处理模式串的思想在几乎所有模式匹配算法中都在使用。正文无回溯也是每一种模式匹配算法的基本要求。

2.2 Aho-Corasick 简介

Aho-Corasick[15]可以认为是 KMP 算法在多串查找的扩展。是一种字符串多模式匹配算法。该算法在 1975 年产生于贝尔实验室，是著名的多模匹配算法之一。最新版本的 Snort 系统中也使用了该算法。先把所有要查找的串放在一起，根据包含关系建成一棵字典树。然后，根据输入的正文串从字典树的根开始在树里进行查找。这样每次查找的复杂度为 O（m），m 为最长的子串。总共要查 n 次，所以复杂度为 O(nm)。这样做的话，只能说是穷举算法在多串下的直接扩展。所以还要在字典树里添加一些转移，使得源串在匹配过程中不出现回退。假设当前节点为 i，模式串匹配到正文串中字符 a，如果节点 i 不存在 a 字母对应的转移。这时候应该跳转到一个节点 j，从字典树根节点到这个节点的字母组成的字符串，应该是从根节点组成的字符串的后缀，如果有多个这样的节点，则跳转到最短的一个。分析一下这个自动机在匹配时的复杂度。在匹配过程中，由于源串不回退，所以遍历模式串的复杂度为 O（n）。但是在匹配失败的时候，会出现跳转，有可能需要跳转多次才可以匹配成功。但需要注意的是，每次跳转会使得在字典树上深度至少减 1，而深度最多与最长模式串的长度相等，所以跳转的最大次数不会超过最长模式串的长度，所以总的复杂度是 O(n)。

要注意一个地方，后缀关系是可以传递的，a 是 b 的后缀，b 是 c 的后缀，则 a 是 c 的后缀。所以在上面提到的字典树中，由于包含关系，最终会把一个串的所有后缀，从长到短串成了一个链表。

难的地方就是自动机的构造。通用的方法是将预处理分析模式串分成建字典树与增加跳转两个过程，方便建造过程。树根代表正文字符与模式串没有发生任何部分匹配的状态。树中每一个分支都代表了一条可能匹配路径，因此对于像 Snort 这样的规则，每个结点就可能有 256 个分支。假设现在已经建好字典树。要为一个节点增加跳转，最简单的方法是，把根结点到它一路上的字符串组成的字符串，求后缀，再到 trie 树里查找，因为是求最长的后缀，所以从长到短，逐一去查找，直至找到为止，找到的节点就是要跳转的节点。这种方法的复杂度很高。观察一下，假设在字典树里，i 节点是 j 节点的父结点，j 的跳转结点的父结点表示的串，也是 i 的后缀。反过来说，根据一个节点的父结点的跳转可以快速地找到这个结点的跳转。假设当前节点的父结点的跳转结点为 k,下一个要匹配的字符为 c，看 k 是否有 c 的跳转，如果没有，则 k 继续跳转。会否出现没有找到合法的跳转，而 k 又

不能再跳转的情况呢？初始的时候，把所有节点的跳转都指向根结点，可以避免这种情况。

在实现的时候，因为跳转总是从深度大的结点跳到深度小的结点，所以可以通过在字典树上作广度遍历。现在来分析一下构造的复杂度。字典树中结点的总数不会超过待查找的子串的长度和。但是每个结点的查找会进行多次跳转。分析一个子串它对应的一系列节点，每次跳转会使得深度至少减 1，而这些节点的深点至多为子串的长度，所以一个子串对应的所有结点的构造深度之和，不会超过字串的长度，所以总的构造复杂度是跟字符长度和成正比的。

2.3 字符解码

匹配算法中要使用大量正文字符与模式串的模式符相比的运算。2003 年时，Clark 首次提出了字符解码的概念[17]之前，各种模式匹配算法都将输入的正文字符送个各个比较单元中，与各模式符直接与正文字符进行比较操作，可以称为分布比较方案。导致大量的逻辑占用和连线。这些方案要求把输入的正文字符用 8*n 根线连接到所有模式符比较单元中，而一个模式集中一般都要包含了成千上万的模式符，因此产生了大量的连线，如图 12 所示

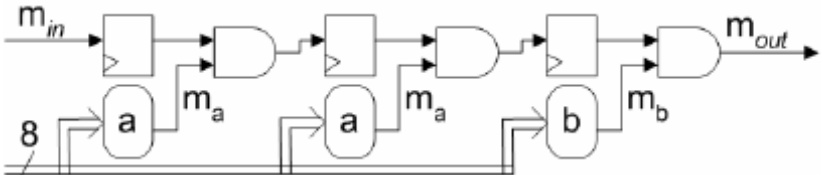

图 12 分布式比较方案中的“aab”比较电路

同时还有大量的不必要的比较操作，如图 12 中就存在两个“a”的比较，其中一个是不必要的。虽然一个模式集中包含了成千上万的模式符，但模式符所属的字符集一般都用 8 位来表示。同时每一个模式符比较单元没必要知道每一个输入的正文字符是什么，也就是说，在每个模式符比较单元中都进行完全 8 位的比较操作是不必要的；每一个模式符比较单元仅需要知道该单元所用的模式符与输入的正文字符是否匹配就可以了。基于这样的思想，Clark 和 Schimmel 首先提出了复用 8 到 256 的 ASCII 解码器对正文字符的解码，而后把 1 位的解码输出，代替原来将 8 位的正文字符，送到相应的模式符比较单元，如图 13 所示。这种字符解码方案可以在他们实现的 NFA 中共享字符比较结果，减少了分布比较方案中每个模式符比较单元中所必需的比较器，因此大大减少了对硬件资源的需求，提高了单个器件中的最大模式符总量，同时也减少了大量连线。

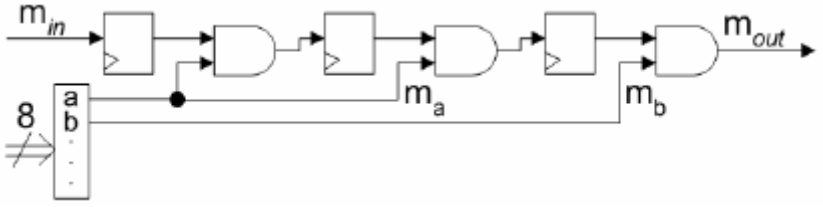

图 13 字符解码方案中的“aab”比较电路

2004 年时，Clark 又将这种思想做了进一步发展，提出了多字符解码 NFA。这种技术用于每个时钟周期处理多个输入字符。可以在不提高时钟频率的条件下，提高吞吐率。为了每时钟周期处理 N 个字符解码，就需要 N 个字符解码器并行地解码 N 个不同的输入正文字符。由于每次会输入正文移动 N 个字符，模式匹配会开始于这 N 个字符的任意位置，所以必须在 N 个可能的位置上搜索所有的模式串。进而需要 N 个并行的状态机跟踪这些所有的位置。

图 14 显示了使用一个 N 个字符解码 NFA 模块的多个模式的系统结构图。输入的每个字符首先被解码，而后字符匹配信号分配到各个模式匹配器的状态机中。其中一个 c_i 代表第 i 个输入字符对字符 c 的匹配信号。模式匹配器代表目标模式。当检测到在任意位置匹配时，每个模式匹配器会输出匹配信号。可能会需要通过与功能集合相关的几个模式匹配器的输出生成综合模式匹配结果。在处理输入流时，一个 k 位的匹配向量记录所有匹配结果。在处理完最后一组数据后，匹配结果编码为 32 位数据发送到分析和判决引擎。

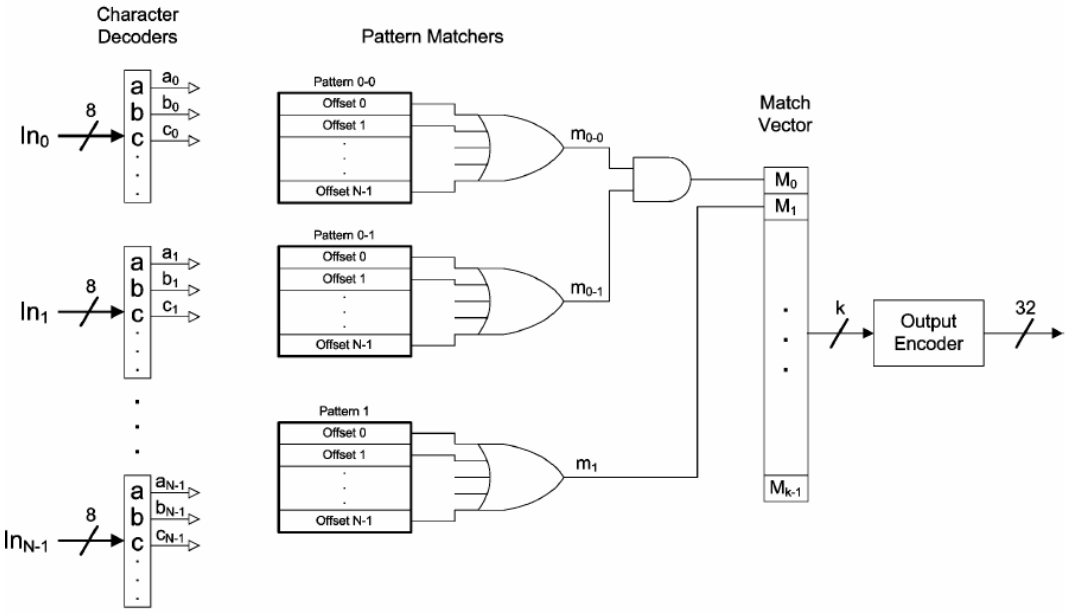

图 14 多字符解码方案总体结构图

图 15 显示图 14 中的一个使用多字符解码 NFA 技术、每时钟周期处理四个输入字符的模式匹配器电路。该电路由四个并行 NFA 组成，每个能检测四个可能位置中的一个。最后一个或门是为了把四个可能位置的检测结果合成生成整个模式匹配器的匹配输出。

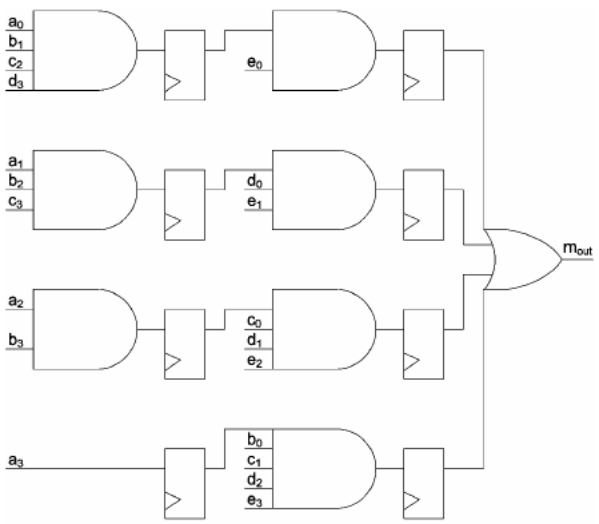

图 15 多字符解码方案模式串示例图

通过以上对字符解码技术的描述，说明字符解码技术的提出是模式匹配的理论与 FPGA 应用相结合的一项关键技术，大减少了对 FPGA 资源的需求。不足之处在于对于多输入字符的匹配使用了四个不同的但功能类似的状态机来完成模式匹配，加大了资源的需求。

后面本文提出的算法利用并发展了字符解码技术，对状态机的实现提出了新的算法利用一个状态机完成多输入字符的匹配。

第三章 一种高速固定模式串匹配算法

固定模式串是指模式串为由字母表Σ上的字符所组成的有穷序列[18]。正如前面所言，在 40Gbps 链路中，从接收到数据开始，4 到 6 个时钟周期必须完成固定模式串匹配任务。这期间的计算任务有：进行大量比较操作，从正文串中找出与模式串相匹配的位置，在规定的时间内完成这些计算任务对硬件算法是一个巨大的挑战。

3.1 固定模式串匹配算法概述

3.1.1 固定模式串匹配的提出

文本信息可以说是迄今为止最主要的一种信息交换手段，比如说这篇论文就是文本信息的一个特例。而作为文本处理中的一个重要领域——固定模式串匹配，也常被称为字符串匹配，简称为串匹配，指的是从文本中找出给定字符串（称为模式）的一个或所有出现的位置。这是我们经常用到的一个功能，如在一个文本中找我们感兴趣的一个单词或一个句子、在一个文件系统中找到某个特定的文件、或者在我们常用的谷歌、百度上找包含我们感兴趣的某个词汇的相关文章、网页，都会用到固定模式串匹配技术。这个技术也是伴随着计算机技术的发展不停地发展。

3.1.2 固定模式串匹配研究现状

上一章中提到的 KMP 算法[16]及 Aho-Crosick（简称 AC）[15]等算法都是固定模式串算法。在这两个算法基础上，出现了一些改进算法和新的算法。如：KR 算法[19]采用哈希方法，可以很容易在大部分情况下避免二次比较，通过合理的假设，这种算法是线性时间复杂度的。它最先由 Harrison 提出，而后由 Karp 和 Rabin 全面分析。Shift-OR 算法[20]，为了最大限度的发挥出位运算的能力，使用位操作，软件中一条指令就可完成运算，来加快运算速度，由此也带来了这个算法最大的局限：模式长度不能大于机器字长。Shift-OR 算法是很高效的匹配算法，同时它可以很容易扩展到模糊匹配上。Hwang[21]介绍了一种基于 Shift-OR 算法的 FPGA 实现的方法，但由于这种方法是通过将多个字符作为一个字符的方法来提高匹配速度，从而在一个时钟周期内可以处理多个正文符的输入，加速了算法的运行速度。Boyer-Moore 字符串搜索算法简称 BM 算法[22]，由 Bob Boyer 和 J Strother Moore 设计于 1977 年，是一种非常高效的字符串搜索算法。这个算法不需要逐一比较被搜索的字符串中的字符，会跳过字符串的某些部分。通常搜索关键字越长，算法速度越快。对于非周期性的模式串而言，这种算法的比较次数上界是 3n，但对于周期性模式，最坏情况下会达到 n 的二次方。为了避免了 BM 算法的二次方问题，出现了一些算法：Apostolico and Giancarlo 算法、Turbo BM 算法和 Reverse Colussi 算法。Aldwairi 等人[23]介绍了一种基于 AC 的自动机存储应用的可重构字符匹配加速器。Tan 等人[24]针对 Aho-Crosick 算法中存在的大量转移边，提出了 bit-split 自动机把大的 AC 状态机分解为小的状态机用于减少存储

转移边所需的存储资源。Jung 等人[25]在 FPGA 上实现了上述的 bit-split，Lunteren[26]提出了 B-FSM 的方法，首先把模式集转变为一种优化的转移边的内部结构，而后再转变为 HASH 表由硬件直接使用，达到运算速度提高的目的。以上这些算法从本质上讲都是使用 DFA（确定型有穷自动机）的方法来处理固定模式串匹配。由 DFA 的特性，每次只能接受一个正文字符，而后根据输入的正文字符和状态机的当前状态确定进入状态机的下一个状态。由于这种特性的影响，导致不论哪种算法，吞吐率都不能很高，无法满足现代网络带宽下内容处理的需求。

为了进一步加速匹配速度，除了以上这些传统的，主要由软件实现的算法外，近些年出现了利用硬件 CAM（内容寻址存储器）、TCAM（三态内容寻址存储器）和 FPGA 配合完成的固定模式串匹配算法。

同时因为相对于长度为 m 的模式串而言，DFA 的状态机中状态的个数是 $O(2^m)$级的，以上算法，在预处理后生成的状态机的状态数很大，不得不需要片外存储才能存储下完整的状态机，所以如何压缩状态机对存储的需求，也是 DFA 算法研究的一个热点。Bremler-Barr, A. [27]在研究 Aho-Crosick 算法的基础上，通过对前缀进行编码，提出了 CompactDFA 算法，减少了对存储资源的需求，同时使用 TCAM 加速用于 IP-查找的速度，达到了 10Gbps 的查寻速度。Piyachon 等人[28]提出使用标签转移表和基于 CAM 的查找表大大减少了存储占用。Gokhale 等人[29]利用 CAM 来实现并行查找字符串。Sourdis 等人[30]介绍了一种基于 CAM 的预解码技术减少资源的使用量。另外，多个芯片的使用同样带来功耗的上升，因此有人同样在研究提高电源效率的方法。Wen 等人[31]研究了正则表达式匹配算法的电源效率问题。他们提出了一种算法通过利用时钟的上升沿和下降沿来处理匹配算法，达到降低时钟频率的目的，从而改善了电源消耗。

3.2 算法基础

为了叙述方便，本文其他部分使用以下定义、模块和定理。

3.2.1 相关定义

为了叙述方便，本文其他部分不做特殊说明情况下，使用以下定义：

1、 字母表Σ：是一个有穷符号的集合。由于本文主要研究高速网络中的匹配算法，因此考虑到网络特性，不失一般性，我们假定字符表中所有字符可用 8bits（位）来表示，即$|\Sigma|$=256。

2、 字母表Σ上的的所有字符串（包括空串在内）的集合记作Σ^*[18]。

3、 Kleene 星号[18]：记作L^*。是连接 0 个或多个字符串得到的所有字符串的集合。

4、 模式符：正则表达式中位于字符表中的单个字符或单个字符间的并、交和补运算的结果。

5、 算子：正则表达式中的表示关系的运算符号，包括。（连接），⋂（交），⋃（并），*（Kleene 星号）和$^-$（补）五种运算。

6、 字符串长度： 一个字符串 w 的长度是它作为字符序列的长度，记为$|w|$。例如空串$|\varepsilon|=0$，$|length|=6$。

7、 对于每一个字符串 w，和每一个自然数 i，字符串 w^i 定义如下：

$$\begin{aligned} &w^0=\varepsilon, \\ &w^{i+1}=w^i\circ w，对每一个 i\geq 0。 \\ &例如，\ w^1=w，do^2=dodo。 \end{aligned} \tag{2}$$

8、 正文串 T_n：从线路输入上，待匹配的字符表中的符号的有穷序列，记为 $T_n=t_0t_1\cdots t_{n-1}\in\Sigma^*$。

9、 每时钟周期处理正文 T_l：一般情况下正文串很长，如网络条件下，正文可能是一个数据包甚至是几个数据包。在 FPGA 中处理时，一个时钟周期内只能处理有限数量的字符，记为 $T_l=t_0t_1\cdots t_{l-1}\in\Sigma^*$。

10、 模式串 P_m：用于匹配正文的正则表达式。

11、 模式串 P_m 长度：模式串中的模式符个数，记为$|P_m|=m$。例如：$|P_4=abad|=4$，$|P_4=a(ba)^*d|=4$。

12、 考虑到匹配可能会需要多个时钟周期的正文，因此定义 t_{-j}。t_{-j} 为前$\left\lceil \frac{j}{l} \right\rceil$时钟的第$\left\lceil \frac{j}{l} \right\rceil \cdot l-j$个正文字符。

13、 使能向量

$$\begin{aligned} &E_i^j=\begin{cases} 1\ if\ i=0\ \ or\ \ t_{j-i}t_{j-i+1}\cdots t_{j-1}=p_0p_1\cdots p_{i-1} \\ 0\ if\ i\neq 0\ and\ \ t_{j-i}t_{j-i+1}\cdots t_{j-1}\neq p_0p_1\cdots p_{i-1} \end{cases} \\ &for 0\leq i<m, 0\leq j<l. \end{aligned} \tag{3}$$

14、 结果向量

$$\begin{aligned} &R_i^j=\begin{cases} 1\ if\ t_{j-i}t_{j-i+1}\cdots t_j=p_0p_1\cdots p_i \\ 0\ if\ t_{j-i}t_{j-i+1}\cdots t_j\neq p_0p_1\cdots p_i \end{cases} \\ &for 0\leq i<m, 0\leq j<l. \end{aligned} \tag{4}$$

15、 结果使能向量 RE_i 与之对应的是结果使能函数函数：完成将结果向量 R_i 映射为向量 RE_i，映射函数为

$$\begin{aligned} &RE_i^j=\begin{cases} R_i^{j-1}\ if\ 0<j<l \\ R_i^{l-1}\ if\ j=0 \end{cases} \\ &所有RE_i^k产生于同一时钟周期，RE_i^0比RE_i^k晚一时钟周期， \\ &for 0\leq i<m, 0\leq j<l. \end{aligned} \tag{5}$$

为了完成 RE_i^0 比所有 RE_i^k 晚一个时钟周期的目的，在 FPGA 中实现时，令 R_i^{l-1} 通过一个 D 触发器来产生 RE_i^0。

16、 匹配路径（Matching Path）：与模式串相匹配的正文字符序列。例如：正文为 cababad，模式串为 a(ba)*d，则匹配路径为 ababad。

17、 匹配路径长度：匹配路径中的字符个数。例如上例中的匹配路径为 ababad，则匹配路径长度为：$|MP|=6$。

18、 模式比较结果C_i：本文研究的中模式匹配算法，因此需要生成模式串中每个字符与每时钟周期处理正文 T_l 的比较结果。生成方式为根据模式串中的 p_i 从解码矩阵中选取其中的一列生成C_i。选取方式为$C_i=[A_{j,k}]$，其中$0\le i<m, 0\le j<l, k=p_i$，$[A_{j,k}]$ 是解码矩阵 A 中的第 p_i 列。

定理 1：如果$C_i=[A_{j,k}]$，其中$0\le i<m, 0\le j<l, k=p_i$，A 是解码矩阵有下式成立

$$C_i^j=\begin{cases}1\ if\ t_j=p_i\\0\ if\ t_j\neq p_i\end{cases}\quad for\ \ 0\le i<m,\ 0\le j<l \tag{6}$$

证明：$\forall i,j, C_i=[A_{j,k}]$，其中$0\le i<m, 0\le j<l, k=p_i\Rightarrow$

$$\left.\begin{cases}C_i^j=A_{j,k}\\k=p_i\\A_{j,k}=\begin{cases}1 if\ t_j=k\\0 if\ t_j\neq k\end{cases}\end{cases}\right\}\Rightarrow C_i^j=\begin{cases}1 if\ t_j=p_i\\0 if\ t_j\neq p_i\end{cases}\quad for\ 0\le i<m, 0\le j<l \tag{7}$$

证毕。

根据定理 1 知，通过以上方案生成C_i，完成了模式串的第 i 个字符与一个时钟周期内输入的所有正文 T_l 的比较操作，C_i 中的每个元素代表了比较结果。

3.2.2 基础模块

下面介绍我们算法中使用的三个基础模块：

1、解码矩阵（Decoder Matrix A）模块：因为我们假定每个字符的编码长度为 8 位，所以对于每时钟周期处理正文T_l的编码长度为l*8位可以解码为l*256位的矩阵A。每一行为 256 位，则解码矩阵的每个元素为：

$$A_{i,j}=\begin{cases}1\ if\ t_i=j\\0\ if\ t_i\neq j\end{cases}\quad for\ \ 0\le i<l,\ 0\le j<256 \tag{8}$$

2、字符比较模块：这个模块完成将两个输入的列向量映射为一个列向量。其中一个列向量为模式比较结果C_i，另外一列为使能向量E_i，输出的列向量为结果向量

R_i，映射函数为$R_i = C_i \,\&\, E_i$，其中&为位与运算。

定理 2：如果使能向量E_i与模式比较结果C_i进行位与运算，则得到的结果为结果函数R_i，即$R_i^j = C_i^j \,\&\, E_i^j$。

证明：$\forall i, j,$其中$0 \le i < m, 0 \le j < l$

a) 假设：$E_i^j = 0 \Rightarrow$

$$\begin{cases} \overset{\text{使能向量}}{\Rightarrow} t_{j-i}t_{j-i+1}\cdots t_{j-1} \neq p_0 p_1 \cdots p_{i-1} \\ \Rightarrow E_i^j \,\&\, C_i^j = 0 \end{cases}$$

$$\Rightarrow \begin{cases} t_{j-i}t_{j-i+1}\cdots t_{j-1}t_j \neq p_0 p_1 \cdots p_{i-1}p_i \overset{\text{结果向量}}{\Rightarrow} R_i^j = 0 \\ E_i^j \,\&\, C_i^j = 0 \end{cases} \tag{9}$$

$$\Rightarrow R_i^j = E_i^j \,\&\, C_i^j.$$

b) 假设：$E_i^j = 1 \; and \; C_i^j = 0 \Rightarrow$

$$\begin{cases} E_i^j = 1 \overset{\text{使能向量}}{\Rightarrow} \begin{cases} E_i^j \,\&\, C_i^j = C_i^j \\ t_{j-i}t_{j-i+1}\cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \end{cases} \\ C_i^j = 0 \overset{\text{定理1}}{\Rightarrow} t_j \neq p_i \end{cases}$$

$$\Rightarrow \begin{cases} E_i^j \,\&\, C_i^j = C_i^j = 0 \\ t_{j-i}t_{j-i+1}\cdots t_j \neq p_0 p_1 \cdots p_i \end{cases} \tag{10}$$

$$\overset{\text{结果向量}}{\Rightarrow} \begin{cases} E_i^j \,\&\, C_i^j = 0 \\ R_i^j = 0 \end{cases} \Rightarrow R_i^j = E_i^j \,\&\, C_i^j$$

c) 假设：$E_i^j = 1 \; and \; C_i^j = 1 \Rightarrow$

$$\begin{cases} E_i^j = 1 \overset{\text{使能向量}}{\Rightarrow} \begin{cases} E_i^j \,\&\, C_i^j = C_i^j \\ t_{j-i}t_{j-i+1}\cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \end{cases} \\ C_i^j = 1 \overset{\text{定理1}}{\Rightarrow} t_j = p_i \end{cases}$$

$$\Rightarrow \begin{cases} E_i^j \,\&\, C_i^j = C_i^j = 1 \\ t_{j-i}t_{j-i+1}\cdots t_j = p_0 p_1 \cdots p_i \end{cases} \tag{11}$$

$$\overset{\text{结果向量}}{\Rightarrow} \begin{cases} E_i^j \,\&\, C_i^j = 1 \\ R_i^j = 1 \end{cases} \Rightarrow R_i^j = E_i^j \,\&\, C_i^j$$

综上所述，有$R_i^j = E_i^j \,\&\, C_i^j$成立。

3、结果转换函数：首先使用定义 15 中的函数。在随后的证明（定理 3 的证明）中得到另外的结果，该结果在定理 3 的证明完成后给出。

定理 3: 结果转换函数的结果向量与下一级的使能向量相等，即$E_i = RE_{i-1}, \; for \; 0 < i < m.$

证明：

a) 假设$0 < i < m, 0 < j < l$

$$\left.\begin{aligned} RE_{i-1}^{j} = R_{i-1}^{j-1} &=^{Definition2} \begin{cases} 1\ if\ t_{j-i}t_{j-i+1}\cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \\ 0\ if\ t_{j-i}t_{j-i+1}\cdots t_{j-1} \neq p_0 p_1 \cdots p_{i-1} \end{cases} \\ E_i^j &= \begin{cases} 1\ if\ \ t_{j-i}t_{j-i+1}\cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \\ 0\ if\ t_{j-i}t_{j-i+1}\cdots t_{j-1} \neq p_0 p_1 \cdots p_{i-1} \end{cases} \end{aligned}\right\} \tag{12}$$

$$\Rightarrow E_i^j = RE_{i-1}^j$$

b) 假设 $0 < i < m,\ j = 0$

由于结果转换函数为了完成 RE_i^0 比所有 RE_i^k 晚一个时钟周期的目的，在 FPGA 中实现时，令 R_i^{l-1} 通过一个 D 触发器来产生 RE_i^0，所以每次使用 RE_i^0 时，$RE_i^0 = R_i^{-1}$。

$$E_i^j = \begin{cases} 1\ if\ i = 0\ \ or\ \ t_{j-i}t_{j-i+1}\cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \\ 0\ if\ i \neq 0\ and\ \ t_{j-i}t_{j-i+1}\cdots t_{j-1} \neq p_0 p_1 \cdots p_{i-1} \end{cases}$$

$$\Rightarrow E_i^0 = \begin{cases} 1\ if\ \ t_{-i}t_{-i+1}\cdots t_{-1} = p_0 p_1 \cdots p_{i-1} \\ 0\ if\ t_{-i}t_{-i+1}\cdots t_{j-1} \neq p_0 p_1 \cdots p_{i-1} \end{cases}$$

$$\left.\begin{aligned} RE_{i-1}^{j} = R_{i-1}^{-1} &=^{Definition2} \begin{cases} 1\ if\ t_{-i}t_{-i+1}\cdots t_{-1} = p_0 p_1 \cdots p_{i-1} \\ 0\ if\ t_{-i}t_{-i+1}\cdots t_{-1} \neq p_0 p_1 \cdots p_{i-1} \end{cases} \\ E_i^0 &= \begin{cases} 1\ if\ \ t_{j-i}t_{-i+1}\cdots t_{-1} = p_0 p_1 \cdots p_{i-1} \\ 0\ if\ t_{j-i}t_{j-i+1}\cdots t_{j-1} \neq p_0 p_1 \cdots p_{i-1} \end{cases} \end{aligned}\right\} \tag{13}$$

$$\Rightarrow E_i^j = RE_{i-1}^j$$

综上所述，$E_i = RE_{i-1},\ for\ 0 < i < m.$

通过以上的证明，可以知道 FPGA 中实现时，结果转换函数使用定义 15 中的函数来构建。但在本文证明过程中应使用以下定义：

$$\begin{aligned} RE_i^j &= R_i^{j-1} \\ for 0 \le i < m, &0 \le j < l. \end{aligned} \tag{14}$$

3.2.3 解码矩阵研究

本文的研究过程中发现，现代网络、数据库和搜索引擎等领域对正则表达式匹配的速率要求越来越高，需要发展 Clark 的字符解码方案。通过引入分而治之的思想，把模式符与正文字符比较这个操作分解为对正文字符解码和根据模式符取比较结果这两个过程，提出了解码矩阵方案。这种方案的思想是，首先生成解码矩阵。把并行输入的 l 个正文字符解码，按模式符集的需要生成 l 行 $|\Sigma|$ 列的矩阵（在本文 5.1 节后，会进一步扩展这个矩阵的列数）。每一位的取值如 3.2.2 中解码矩阵（Decoder Matrix A）模块所示，因此每一列的列号就是模式符的编码值，进而每一列的值就是并行输入的 l 个正文字符与列号相对应的模式符比较结果。解码过程中只需要知道模式符集中模式符的个数，不考虑取比较结果的具体操作；其次取比较结果。每个模式符比较单元根据自己的模式符，从解码矩阵中取出与该模式符的编码值相等列号的列。通过从解码矩阵中取得相应的列，完成了模式符与并

行输入的 l 个正文字符的 l 次比较。取比较结果操作不需要知道并行输入的 l 个正文字符具体的值，只关注自己的模式符对应解码矩阵的列号，而这个对应关系在设计之初就已经确定，在运行过程中是不会发生变化的。

通过解码矩阵的引入，将模式匹配中，必需的且大量的正文字符与模式符的比较操作分成两步简单的操作，第一步解码，等效与完成并行输入的正文字符与所有模式符的比较操作，形成比较后的结果矩阵；第二步取结果，每个模式符从上步的结果矩阵中取得与正文字符比较的结果。优势在于将原来分散在各处的大量比较操作集中在一处完成，既减少了总的计算量，相应减少了逻辑占用量，又减少了大量连线资源。

3.3 一种高速固定模式串匹配算法——“向量与”算法

在前人对固定模式串算法研究的基础上，为了适应现代高速网络的发展，满足在高网络带宽的条件下，高吞吐率固定模式串匹配的要求。在对固定模式串匹配及 FPGA 芯片资源深入研究的基础上，本文提出“向量与”算法用基本逻辑电路的与门实现固定模式串中的连接运算，达到提高运算速度的目的。经在当前 FPGA 上实验验证，本文提出的算法可以以 512Gbps 的吞吐率完成固定模式串的匹配[32]。

下面分两部分对该算法进行说明与证明，第一部分为算法的构造说明；第二部分为利用第二章的定义和定理，完成本算法的理论证明。

3.3.1 算法构造

本算法完全使用第三章第二节中的各种定义及三个基本模块进行构造，其中分为以下几步完成构造：

1、首先构造公用的解码矩阵，完成每时钟周期处理正文T_l的解码任务。方便后续 C_i 的提取。

2、针对模式串中第一个字符（ p_0 ）匹配的构造方法：构建字符比较模块，令 $E_0 = \left[\overbrace{1,1,\cdots 1}^{l}\right]^T$ ；根据 p_0 从解码矩阵中取得模式比较结果C_0，即 $C_0 = \left[A_{j,k}\right]$, 其中 $0 \le j < l, k = p_0$；使用字符比较模块将 E_0 和 C_0 映射为 R_0；最后使用结果转换函数将 R_0 映射为 RE_0。

3、针对模式串中除第一个字符与最后一个字符外的所有字符（ p_i 其中$0 < i < m-1$ ）匹配的构造方法：构建字符比较模块，令 $E_i = RE_{i-1}$；同样根据 p_i 从解码矩阵中取得模式比较结果 C_i，即 $C_i = \left[A_{j,k}\right]$, 其中 $0 < \mathrm{i} < \mathrm{m}\text{-}1, 0 \le j < l, k = p_i$；使用字符比较模块将 E_i 和 C_i 映射为 R_i；最后使用结果转换函数将 R_i 映射为 RE_i。

4、针对模式串中最后一个字符匹配（ p_{m-1} ）的构造方法：构建字符比较模块，令

$E_{m-1}=RE_{m-2}$，同样根据 p_{m-1} 从解码矩阵中取得模式比较结果 C_{m-1}；使用字符比较模块将 E_{m-1} 和 C_{m-1} 映射为 R_{m-1}；

5、最后一级生成的 R_{m-1} 就是固定模式串与正文串 T_n 的匹配结果。

$$R_{m-1}^{j}=\begin{cases}1\ if\ t_{j-m+1}t_{j-m+2}\cdots t_j=p_0p_1\cdots p_{m-1}\ and\ t_j\in T_l\\0\ if\ t_{j-m+1}t_{j-m+2}\cdots t_j\neq p_0p_1\cdots p_{m-1}\ and\ t_j\in T_l\end{cases}\quad for\ 0\le j<l. \tag{15}$$

3.3.2 算法证明

通过算法构造过程可以看出：我们可以通过使用固定模式串的长度 m，使用数学归纳法来证明算法可以完成任意长度的固定模式串匹配。

1、基本步骤。设固定模式串的长度 $|P|$ 为1，即，$|P_m|$=1，则 $P_m=p_0$。根据每时钟周期处理正文 T_l，同时时钟周期内可生成解码矩阵模块A。根据定理1，有

$$C_0^j=\begin{cases}1\ if\ t_j=p_0\\0\ if\ t_j\neq p_0\end{cases}\quad for\ \ 0\le j<l \tag{16}$$

另外根据构造和定理 2 知

$$\left.\begin{aligned}&E_0=\left[\overbrace{1,1,\cdots1}^{l}\right]^T\Rightarrow E_0^j=1\\&R_0^j=E_0^j\ \&\ C_0^j\end{aligned}\right\}\Rightarrow R_0^j=E_0^j\ \&\ C_0^j=C_0^j=\begin{cases}1\ if\ t_j=p_0\\0\ if\ t_j\neq p_0\end{cases}\quad for\ \ 0\le j<l \tag{17}$$

于是，算法的结果（R_0）的第个元素确切表示固定模式串与正文串 T_j 的匹配结果。如果 R_0=1，则 $t_j\in T_l$,且 $t_j=p_0$；否则，$t_j\in T_l$,且 $t_j\neq p_0$。

2、归纳假设。设m≥1，“向量与”算法能够完成所有长度小于等于m的固定模式串匹配。即：

$$R_{m-1}^{j}=\begin{cases}1\ if\ t_{j-m+1}t_{j-m+2}\cdots t_j=p_0p_1\cdots p_{m-1}\ and\ t_j\in T_l\\0\ if\ t_{j-m+1}t_{j-m+2}\cdots t_j\neq p_0p_1\cdots p_{m-1}\ and\ t_j\in T_l\end{cases}\quad for\ 0\le j<l. \tag{18}$$

3、归纳步骤。设固定模式串的长度 $|P|$ 为m+1，则 $P_m=p_0p_1\cdots p_m$。根据归纳假设，R_{m-1} 中记录了对于 $p_0p_1\cdots p_{m-1}$ 的匹配结果，

$$R_{m-1}^{j}=\begin{cases}1\ if\ t_{j-m+1}t_{j-m+2}\cdots t_j=p_0p_1\cdots p_{m-1}\ and\ t_j\in T_l\\0\ if\ t_{j-m+1}t_{j-m+2}\cdots t_j\neq p_0p_1\cdots p_{m-1}\ and\ t_j\in T_l\end{cases}\quad for\ 0\le j<l. \tag{19}$$

根据构造和结果使能函数知，

$$E_m^j = RE_{m-1}^j = R_{m-1}^{j-1} =$$
$$\begin{cases} 1\ if\ t_{j-m}t_{j-m+1}\cdots t_{j-1} = p_0 p_1 \cdots p_{m-1} \\ 0\ if\ t_{j-m}t_{j-m+1}\cdots t_{j-1} \neq p_0 p_1 \cdots p_{m-1} \end{cases} \quad (20)$$
$$for\ 0 \le j < l.$$

根据解码矩阵 A 和定理 1，固定模式串的最后一个字符 p_m 与当前时钟周期处理正文 T_l 的匹配结果表示为

$$C_m^j = \begin{cases} 1\ if\ t_j = p_m \\ 0\ if\ t_j \neq p_m \end{cases} \quad (21)$$
$$for\ \ 0 \le j < l$$

由等式（20）、（21）和定理 2，有

$$R_m^j = E_m^j \,\&\, C_m^j =$$
$$\begin{cases} 1\ if\ t_{j-m}t_{j-m+1}\cdots t_j = p_0 p_1 \cdots p_m \\ 0\ if\ t_{j-m}t_{j-m+1}\cdots t_j \neq p_0 p_1 \cdots p_m \end{cases} \quad (22)$$
$$for\ 0 \le j < l.$$

于是，算法结果 R_m 就是固定模式串与正文串 T_n 的匹配结果

综上所述，“向量与”算法能够完成对任意长度的固定模式串与正文串 T_n 的匹配，匹配结果存放于与每个固定模式串相对应的最后一级 R_{m-1} 中。

3.4 小结

Shift-Or 改进算法[21] 是 2009 年 Hwang 发表的仅使用 FPGA 逻辑实现的固定模式串匹配算法。本节提出的“向量与”算法同样仅使用 FPGA 逻辑实现的固定模式串匹配算法。同样都是在 Altera 公司[33]的 Stratix II 系列 FPGA 上实现。正则表达式模式串也同样取自 SNORT 的规则集。因此具有可比性。

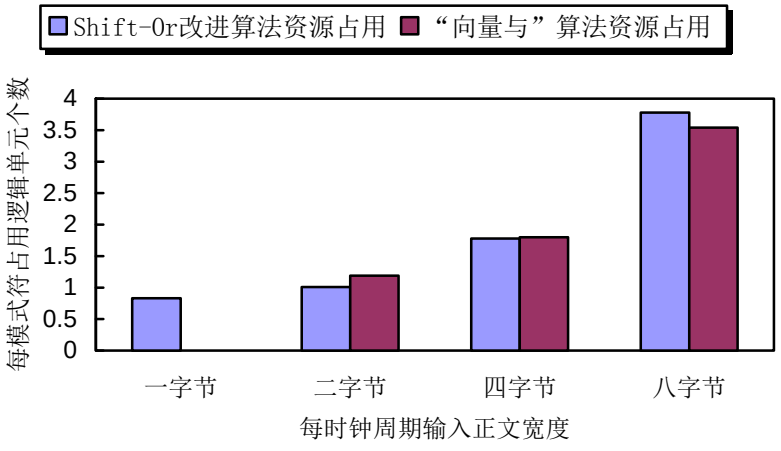

图 16 Shift-Or 改进算法和“向量与”算法资源占用比较

图 16 列出了两种算法中每模式符占用 FPGA 基本逻辑单元的比较结果。由于本节提出

的算法不能实现每时钟周期输入一个正文字符，因此第一项中无本节算法的数据。另外，Hwang 在论文中仅提供了每时钟周期输入一、二、四和八个正文宽度的实现结果，所以只能进行每时钟周期输入二、四和八个正文字符的对比结果。

图 16 是“向量与”算法和 Shift-Or 改进算法[21]吞吐率和资源占用比较，通过比较可以看出：

1、资源占用方面，在输入正文字符宽度较小情况下本文提出的算法资源占用高，但随着每时钟周期输入正文字符宽度的增加，“向量与”算法的资源占用增加量较小，增加幅度小于Shift-Or改进算法。

2、吞吐率方面，由于Shift-Or改进算法的特性，通过将多个字符作为一个字符的方法来提高匹配速度，所以Shift-Or改进算法每时钟周期输入正文宽度不能很大，也就是吞吐率最大也就是该文中提出的32Gbps。与之相比“向量与”算法模式串与正文串无关，且每个正文字符间也相互独立，所以每时钟周期输入正文宽度只受器件特性和资源限制，可以增加的比较大，“向量与”算法的吞吐率可以达到512Gbps。

第四章 基于 NFA 的“向量与”算法

近年来由于网络攻击日益频繁，正则表达式模式串的数量增幅远大于固定模式串数量。例如在 2004 年 SNORT2.2 时，规则集中静态模式串的数量为 1631 个，正则表达式模式串的数量只有 157 个，到 2007 年 SNORT2.6 时，规则集中静态模式串的数量为 2927 个，正则表达式模式串的数量有 1687 个，到最近 SNORT2.9 时，规则集中静态模式串已经超过了 5000 个，同时正则表达式的数量也超过了 2000 个。正则表达式匹配不仅面临与固定模式串匹配相同的很短的时间内完成这些计算任务要求，而且还面临正则表达式特有的算子所带来的复杂性难题。与一个固定模式串相匹配的正文子串只有一种，但与一个正则模式串相匹配的正文子串却有可数无限种。

上一章介绍了一种固定模式串匹配算法，完成了该算法的构造与正确性证明。固定模式串匹配是正则表达式匹配的基础，由正则表达式的定义可以看出，正则表达式是在固定模式串的基础上通过引入并、交、补和 Kleene 星号运算形成。因此如果使用与固定模式串匹配相似的逻辑来实现正则表达式匹配，只需实现并、交、补和 Kleene 星号算子即可。

4.1 正则表达式匹配算法概述

4.1.1 正则表达式历史与应用

正则表达式也译为正规表示法、常规表示法。在计算机科学中，指用来匹配一系列符合这个句法规则的字符串的单个串。最早的描述是上世纪四十年代 McCulloch[34]在一篇用数学方式描述神经网络的新方法中创新性地把神经系统中的神经无描述成简单的自动控制单元，这些自动控制单元就是自动机的原型。Kleene[35]在 McCulloch 的基础上，进一步研究提出了正则集合的数学符号描述该模型，正式引入了正则表达式的概念。Thompson[36]把这一成果应用于搜索算法，提出了著名的 Thompson 算法，并把这一算法应用于编辑器 QED，然后是 Unix 系统中的编辑器 ed，最终在现有所有 Unix、Linux 系统中都是常用工具的 grep 中使用。

现阶段，正则表达式出现在计算机的相关的各种领域内，大的方面：搜索引擎的基础是正则表达式；数据库的实现中也大量应用正则表达式；网络安全中网络入侵与检测系统中正则表达式的使用量也呈现出迅速增加态势，如常用的 SNORT 中 2003 年时只有 65 个正则表达式，2004 年就已经达到 104 个，2006 年时就上升到 1504 个，到最近，这个数量更是达到了 21254，不到十年中，正则表达式数量已经增加了 300 多倍，年平均增加了 30 多倍；另外病毒扫描与检测中也同样越来越多在使用都正则表达式。小的方面 Unix、Linux 下的编辑器 ed、qed、vi 等、工具 grep、awk、sed、还有众多的工具都广泛支持正则表达式，近些年 Windows 下的开发工具包中也从开始的支持发展到广泛使用。其它，各种开发语言(C#、Java、VB、C++)、脚本语言（Javascript、PHP、Perl、Tcl)都支持正则表达式的应用。其中 perl 语言中使用的正则表达式已经发展为 PCRE（Perl Compatible Regula

Expression），pcre 成为了许多应用的库，如典型的网络入侵与检测系统 SNORT 就是以 PCRE 为基础，各种规则都用 PCRE 的语法来表示。表 4 中列出了 PCRE 中常用语法：

表 4　PCRE 常用语法

元字符	描述
a	单个字符。所有 ASCII 字符
.	匹配任何单个字符。匹配除“\n”之外的任何单个字符。要匹配包括“\n”在内的任何字符，请使用像“(.|\n)”的模式。例如正则表达式 r.t 匹配这些字符串：rat、rut、r t，但是不匹配 root。
$	匹配行结束符。例如正则表达式 doctor$ 能够匹配字符串"He's a doctor"的末尾，但是不能匹配字符串"They are doctors."
^	匹配一行的开始。例如正则表达式^We defined 能够匹配字符串"We defined three basic modules."的开始，但是不能匹配"We introduced some definitions and theorems"
\	引用符，用来将这里列出的这些元字符当作普通的字符来进行匹配。例如正则表达式\$被用来匹配美元符号，而不是行尾，类似的，正则表达式\.用来匹配点字符，而不是任何字符的通配符。
[abc]	字符集。匹配括号中的任何一个字符。例如正则表达式 r[aou]t 匹配 rat、rot 和 rut，但是不匹配 ret。
[a-f]	范围字符集。例如正则表达式 [0-9]可以匹配任何数字字符；还可以制定多个区间，例如正则表达式[A-Za-z]可以匹配任何大小写字母。
[^abc]	非字符集。匹配除方括号内字符的任一字符.要想匹配除了指定区间之外的字符——也就是所谓的补集——在左边的括号和第一个字符之间使用^字符，例如正则表达式[^269A-Z] 将匹配除了 2、6、9 和所有大写字母之外的任何字符。
\< \>	匹配词（word）的开始（\<）和结束（\>）。例如正则表达式\<the\>能够匹配字符串"correctness of the algorithm"中的"the"，但是不能匹配字符串"otherwise"中的"the"。
\(\)	将 \(和 \) 之间的表达式定义为“组”（group），并且将匹配这个表达式的字符保存到一个临时区域（一个正则表达式中最多可以保存 9 个），它们可以用 \1 到\9 的符号来引用。主要在替换中使用。
RegExp*	Kleene 星号。匹配 0 或多个正好在它之前的那个字符。例如正则表达式 .* 意味着能够匹配任意数量的任何字符。
RegExp +	匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。
?	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”中的“do”。
RegExp1RegExp2	连接。连接正则表达式 1 和正则表达式 2
RegExp1 | RegExp2	并。匹配正则表达式 1 或正则表达式 2。例如，“z|foo”能匹配“z”或“foo”。“(z|f)ood”则匹配“zoo”或“foo”。

元字符	描述
\000	匹配 8 进制 000 表示的 ASCII 字符
\d	匹配 0-9 的数字
\D	匹配除 0-9 的数字外的字符
\w	匹配包括下划线、字母和数字的字符的任一字符。
\W	匹配除下划线、字母和数字的字符的任一字符。
\s	匹配空白字符。包括空格、制表符、换页符等。等价于[\f\n\r\t\v]。
\S	匹配除空白外的字符
\n	匹配回车符（LF，NL）
\N	匹配除回车符外的字符
\r	匹配换行符
\t	匹配 tab 符
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。
\B	匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。
\cx	匹配由 x 指明的控制字符。例如，\cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的“c”字符。
\f	匹配一个换页符。等价于\x0c 和\cL。
\v	匹配一个垂直制表符。等价于\x0b 和\cK。

4.1.2 正则表达式理论基础

正则表达式、确定型有穷自动机（DFA）、非确定型有穷自动机（NFA）的理论研究开始于上世纪五十年代中期，完成于六、七十年代。基本理论主要有：

1、正则语言的提出：正则表达式、确定型有穷自动机、非确定型有穷自动机三者的形式化定义。

2、三者等价性证明：最终的结论是三者在功能上是在所能接受或生成的语言类上是等价的。三者可以相互表现，相互转化。同时，给出了三者相互转化时的构造方法。

3、著名的“泵原理”。使用“泵原理”可以确定给定的语言是不是正则语言。确定了正则语言的界限。

4、状态转移。虽然理论上确定型有穷自动机、非确定型有穷自动机是等价的，但就一对等价的确定型有穷自动机和非确定型有穷自动机而言，确定型有穷自动机的状态数不小于非确定型有穷自动机的状态数，最差情况确定型有穷自动机的状态数是确定型有穷自动机的状态数指数型的。正文串一个字符一个字符输入时，因为确定型有穷自动机在任何时刻只有一个活跃状态，所以状态转移是每次有且只有一个状态转移；与此相应地，因为非确定型有穷自动机可以有多个活跃状态，

所以一次可能需要多个状态的转移，最大可能的数量为非确定型有穷自动机的状态数。

4.1.3 相关工作

Mealy[37] 和 Moore[38]在早期自动机进行了论述。对 Rabin 和 Scott[39]提出了 NFA 的概念，并给出了对任一 NFA 存在一台等价 DFA 的证明，从而证明了 DFA 与 NFA 是等价的。Kleene 给出了有穷自动机接受正则语言的方法，McNaughton 和 Yamada[40]给出了从自动机到正则语言和正则语言到自动机的转换方法。至此在理论上完成了正则表达式、NFA 和 DFA 三者等价的理论证明。方便了后人的研究，在一个方向的成果可以方便地转换到其它领域。

从上世纪 79 年代开始，人们开始大量研究用硬件来实现正则表达式。Mukhopadhyay[41]给出了用于连接、并和 Kleene 星号的基本块结构的一种硬件实现。Floyd[42]和 Ullman[60]讨论了在可编程逻辑（PLA）上的 NFA 的实现方法，并给出对于具有 N 个状态的 NFA 的电路复杂度至多为$O(N)$。Foster[43]在描述了一些 NFA 实现时经常用到的电路后，给出避免锁存电路对实现的影响的方法。Sidhu 和 Prasanna[44]设计了实现在 FPGA 上的基于 NFA 的正则表达式实现，实现了 Mukhopadhyay[41]提出的各种结构。Yang[45]等人在上述结构进行了一些修改后，能够支持一次输入多个字符，加快了处理速度。Hutchings 等人[46]将所有 SNORT[12,13]的固定模式全部组成一个正则表达式完成匹配，大大减少了对 FPGA 的资源需求。Clark 和 Schimmel[17, 47]首先提出了复用 8 到 256 的 ASCII 解码器对正文字符的解码，可以在他们实现的 NFA 中共享字符比较结果，因此大大减少了对硬件资源的需求。Lin 等人[48]提出了多个正则表达式可以共享公共子串，通过这种方式以节约硬件资源。Moscola 等人[49,50]组合 Sidhu[50]和 Clark[47]的方法来完成正则表达式匹配。Yamagaki[51]提出了一种每时多个钟周期处理正文字符的 NFA 方案。Hwang[21]介绍了一种基于 Shift-OR 算法的 FPGA 实现方法，同时由于所基于 Shift-OR 算法是一种固定串匹配算法，因此这种方法是通过将多个字符作为一个字符的方法来提高匹配速度，而在文章中没有讨论正则表达式中 Kleene 星号运算的实现方案。Wang[52]等人提出了一种基于 NFA 的 FPGA 实现结构，用于改善 Kleene 星号运算，同时利用 FPGA 中的 Block RAM 存储针对 Kleene 星号运算结构，提高了效率。

尽管 FPGA 非常适合 NFA 的实现，但由于模式匹配开始于固定串的研究，进而发展为用 DFA 来实现固定串，并且 DFA 的运算复杂度是 O(1)，很适合基于处理机的软件来实现，因此开始对正则表达式研究时，人们首先接触到的是大量的 DFA 实现算法，因此还是有很多人在 FPGA 上研究 DFA 的应用。DFA 来实现正则表达式时面临的最主要问题是 DFA 相对于长度为 m 模式串其状态复杂度是 $O(2^m)$，由此带来的问题是状态数量很大，只能使用外部存储器进行存储。导致运算时需要大量访问片外存储器，因此研究 DFA 算法实现时，大量的研究集中在如何压缩状态数量。Baker 等[53]描述了一种在 FPGA 中使用微控制器的 DFA 实现，优势在于规则集放在存储器中，通过更新存储器中的内容就可以完成规则集的

改变。针对 DFA 规则集过大，许多人都进行了各种努力，Kumar 等人[54]提出了针对状态转移的 D^2FA 存储压缩技术。Brodie 等人[55]介绍了通过字符和字符序列编码，用于减少转移表中列的数量，对状态重排来减少每次转移对存储访问的次数。于强等人[56]提出了 WD^2FA 可以将状态转移数目压缩为 D^2FA 的 95%左右。

4.2 “向量与”算法

通过以上的讨论知道：正则表达式模式匹配都是通过 DFA 或 NFA 来实现。假定模式串长度为 m，正文串长度为 n 的条件下，每输入一个正文字符，DFA 的计算复杂度是 O(1)，而 NFA 的计算复杂度是 O(m)。一方面，由于 FPGA 的使用开始于上世纪九十年代左右，真正大量应用是在本世纪；另一方面，处理机中核的数量有限，不支持大规模并行计算，因此开始于上世纪六、七十年代正则表达式研究中，大量的研究人员能使用的工具只有软件，于是在研究正则表达式匹配算法时首选的方案是 DFA。由于上述原因，在 DFA 的实现上有大量研究成果。NFA 上研究的人员相对要少得多。随着网络带宽的迅速增加和实时应用的需求，模式匹配的速度要求越来越高，同时模式串数量也越来越多。自动机的另外一个约束——状态机的存储需求变得越来越重要。这里有个 DFA 的状态复杂度是 $O(2^m)$，NFA 的状态复杂度 O(m)的例子：

设字母表 $\Sigma = \{a_1, a_2, \cdots a_n\}$，其中 n≥2。语言 $L = \{w: \text{有一个符号} a_i \in \Sigma \text{不出现在} w \text{中}\}$，设计一台接受这种语言的 NFA 是很方便的。假定 n=4，可得到接受语言 L 的非确定型有穷自动机的状态图为图 17：

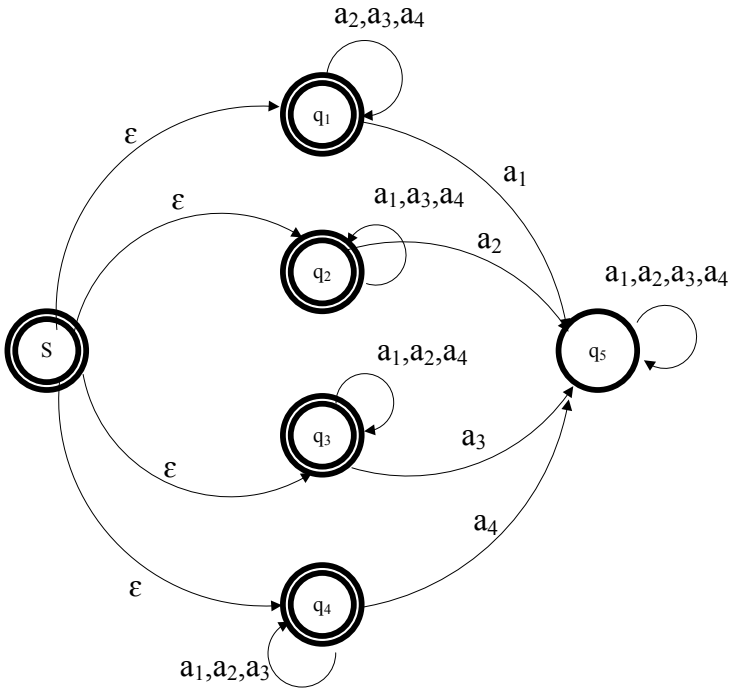

图 17 接受语言 L 的 NFA

同时也可得到接受语言 L 的确定型有穷自动机的状态图为图 18：

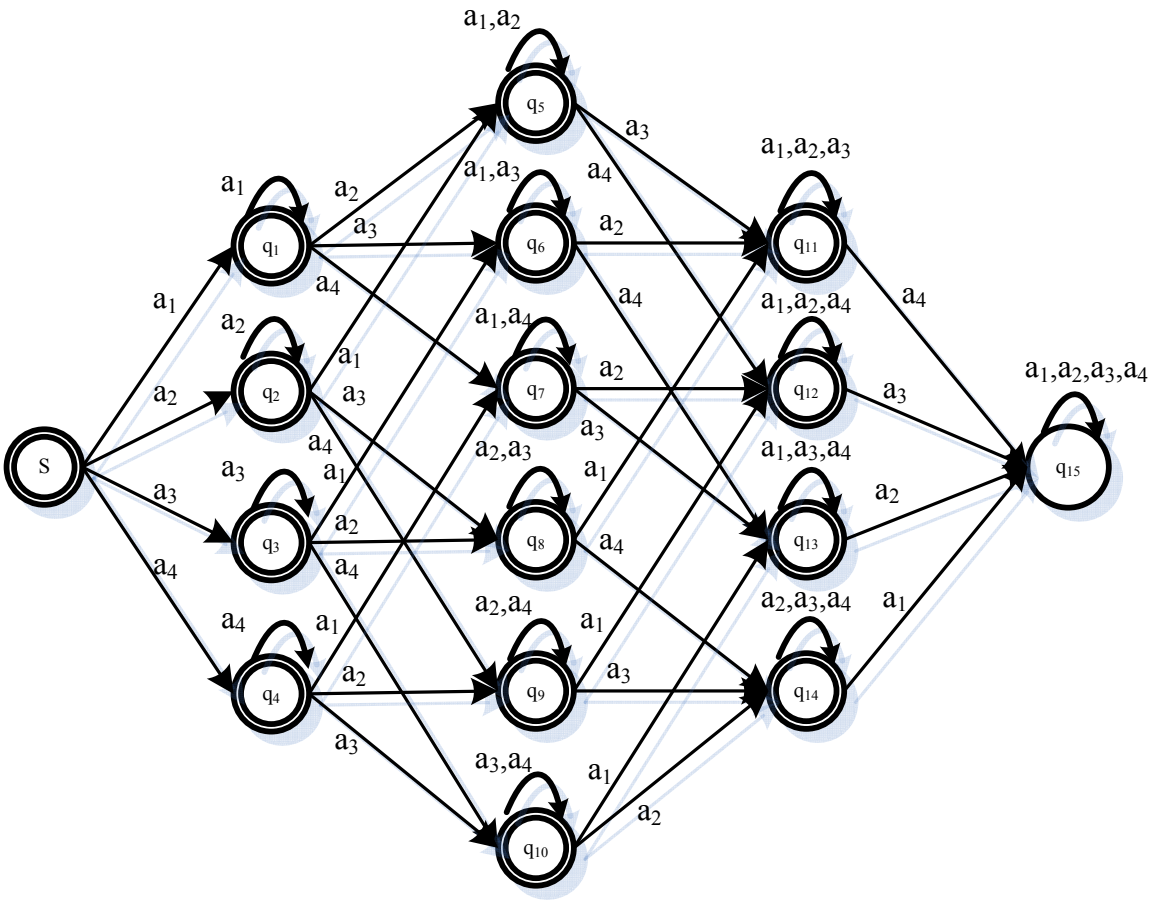

图 18 接受语言 L 的 DFA

由图 17 和图 18 可知，对于上述正则表达式，NFA 只需要 6 个状态和 13 种状态转移就可完成，DFA 需要 16 个状态和 47 种状态转移才能完成。

实际应用中，不仅要存储所有状态，而且要存储所有状态间的转移，最终导致 DFA 的存储需求很大，只能使用片外专用存储器来存储所有状态；同时状态机的运行基本操作是根据状态机的当前状态和输入的正文字符确定下一个状态，由此，如果使用片外存储，则需要很高的片间吞吐率。

相对于 DFA 的状态复杂度 $O(2^m)$，NFA 的状态复杂度 O(m)；DFA 的计算复杂度 O(1)，而 NFA 的计算复杂度 O(m)，NFA 在状态复杂度和计算复杂度上平衡性强于 DFA。同时 FPGA 的最大特点就是片内可以自由构造出大量运算单元，由此通过合适的构造可以使输入正文后，NFA 完成所有运算。再加上 NFA 的状态复杂度不高，可以在芯片内完成所有状态的存储，NFA 就成为一个重要的正则表达式匹配算法的方向。在 FPGA 上使用 NFA 要解决的最重要难题是：每时钟周期处理多个正文字符，且在同一时钟周期内完成所有的状态转移。

在前人对正则表达式算法研究的基础上，为了适应现代高速网络了发展，满足在高网络带宽的条件下，高吞吐率正则表达式匹配的要求。在对正则表达式及 FPGA 芯片资源深入研究的基础上，本文提出的“向量与”算法用基本逻辑电路的与门、或门和非门实现正

则表达式中的连接、交、并、Kleene 星号和补运算，从而达到提高运算速度的目的。经在当前 FPGA 上实验验证，本文提出的“向量与”算法可以以 512Gbps 的吞吐率完成正则表达式的匹配[57]。

4.2.1 “向量与”算法接受所有的正则表达式

正则语言类是某些有穷的语言在并、连接及 Kleene 星号运算下的闭包，因此我们应该证明“向量与”算法接受的语言类也有同样的闭包性质。

该闭包的性质为：有穷自动机接受的语言类在下述运算下是封闭的。

A. 补；

B. 连接;

C. 并；

D. Kleene 星号；

E. 交。

下面我们对上一章的“向量与”算法进行适当的扩展，分别对上述五种情况进行构造和证明。

证明：给定自动机 Mi，要说明如何构造接受所需语言的自动机 M 。

A. 补。设 $M1$ 是一台本算法已实现的非确定型有穷自动机。那么补语言 $L(M)=\overline{L(M1)}=\Sigma^{*}-L(M1)$ 被非确定型有穷自动机 $M=(K,\Sigma,\delta,s,K-F)$ 接受。也就是说如果 $M1$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，则 M 的模式串为 $P_m=\overline{p_0p_1\cdots p_{m1-1}}$。

构造方法：根据 $P_{m1}=p_0p_1\cdots p_{m1-1}$ 按上一章的方法构造 $M1$；而后把 $M1$ 的输出 R_{m1-1} 按位取反，得到 M 的结果 $\overline{R_{m1-1}}$。

证明：由 3.2.2 算法证明结果知，$M1$ 的输出 R_{m1-1} 为

$$R_{m-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j=p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{23}$$

把 $M1$ 的输出 R_{m1-1} 按位取反，得到 M 的结果 $\overline{R_{m1-1}}$

$$\overline{R_{m1-1}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j\neq p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j=p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{24}$$

恰好为 M 的结果。

B. 连接。令 $M1$ 和 $M2$ 是两台本算法已实现的非确定型有穷自动机，要构造一台非确定型有穷自动机 M 使得 $L(M)=L(M1)\circ L(M2)$

根据 D 的证明知，补的特性为从模式串的第一个字符的使能向量 $E_0=\left[\overbrace{1,1,\cdots 1}^{l}\right]^T$ 开始，直到算法结果输出 $\overline{R_{m1-1}}$ 为止，必须保证补运算的完整性。因此在构造连接运算时，需要分别对以下四种情况进行讨论。

a） $L(M1)$ 和 $L(M2)$ 都不是补。

令 $P_{m1}=p_0p_1\cdots p_{m1-1}$ ， $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$ ， 则 $L(M)=L(M1)\circ L(M2)$ ，$P_m=p_0p_1\cdots p_{m1-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$。

构造方法：按上一章的方法分别构造 $M1$ 和 $M2$，而后用 $M1$ 的 RE_{m1-1} 代替 $M2$ 的 E_0，即，$E_0^{'}=RE_{m1-1}$。这样 $M2$ 的结果 R_{m2-1} 就是 M 的结果。

证明：由 3.2.2 算法证明结果知，$M1$ 的输出 R_{m1-1} 为

$$R_{m1-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j=p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{25}$$

由结果使能函数，知

$$RE_{m1-1}^{j}=R_{m1-1}^{j-1}=\begin{cases}1\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}=p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{26}$$

由构造中 $E_0^{'}=RE_{m1-1}$，知

$$E_0^{'j}=\begin{cases}1\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}=p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{27}$$

与 3.2.2 算法类似的证明，有

$$R_{m2-1}^{j}=$$

$$\begin{cases}1\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}t_{j-m2+1}t_{j-m2+2}\cdots t_j=p_0p_1\cdots p_{m-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}t_{j-m2+1}t_{j-m2+2}\cdots t_j\neq p_0p_1\cdots p_{m-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{28}$$

恰好为 M 的结果。

b） $L(M1)$ 和 $L(M2)$ 都是补

令 $M3$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$ ， $M4$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M1)=\overline{L(M3)}$，$L(M2)=\overline{L(M4)}$，则，$L(M)=L(M1)\circ L(M2)=\overline{L(M3)}\circ\overline{L(M4)}=\overline{L(M3)\circ L(M4)}$， M 的模式串为 $P_m=\overline{p_0p_1\cdots p_{m1-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}}$。

构造方法：按上一章的方法分别构造 $M3$ 和 $M4$，而后按 4.2-B-a）的方法构造

$L(M3)\circ L(M4)$，最后将$L(M3)\circ L(M4)$的结果按位反，就得到$L(M)$的结果。

证明：由 34.2-B-a）算法证明结果知

$$R_{m2-1}^{j}=$$

$$\begin{cases}1\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0p_1\cdots p_{m-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\ 0\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0p_1\cdots p_{m-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad(29)$$

$$for\ 0\le j<l.$$

把R_{m2-1}^{j}按位反，得到

$$\overline{R_{m2-1}^{j}}=$$

$$\begin{cases}1\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0p_1\cdots p_{m-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\ 0\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0p_1\cdots p_{m-1}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad(30)$$

$$for\ 0\le j<l.$$

恰好为M的结果。

c） $L(M1)$是补，$L(M2)$不是补。

令$M3$的模式串为$P_{m1}=p_0p_1\cdots p_{m1-1}$，$M2$的模式串为$P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M1)=\overline{L(M3)}$，则，$L(M)=L(M1)\circ L(M2)\ =\overline{L(M3)}\circ L(M2)$，$M$的模式串为$P_m=\overline{p_0p_1\cdots p_{m1-1}}p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$。

构造方法：按上一章的方法分别构造$M3$和$M2$，而后按 4.2-D 的方法构造$\overline{L(M3)}$，最后用$M1$的RE_{m1-1}代替$M2$的E_0，即，$E_0^{'}=RE_{m1-1}$。这样$M2$的R_{m2-1}结果就是M的结果。

证明：由 4.2-D 算法证明结果知

$$\overline{R_{m1-1}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\\ 0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\end{cases}\quad(31)$$

$$for\ 0\le j<l.$$

由结果使能函数，知

$$RE_{m1-1}^{j}=R_{m1-1}^{j-1}=\begin{cases}1\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}\neq p_0p_1\cdots p_{m1-1}\\ 0\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}=p_0p_1\cdots p_{m1-1}\end{cases}\quad(32)$$

$$for\ 0\le j<l.$$

由构造中$E_0^{'}=RE_{m1-1}$，知

$$E_0^{'j}=\begin{cases}1\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}\neq p_0p_1\cdots p_{m1-1}\\ 0\ if\ t_{j-m1}t_{j-m1+1}\cdots t_{j-1}=p_0p_1\cdots p_{m1-1}\end{cases}\quad(33)$$

$$for\ 0\le j<l.$$

与 3.2.2 算法类似的证明，有

$$R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m1-m2+1}t_{j-m1-m2+1}\cdots t_{j-m2}\neq p_0p_1\cdots p_{m1-1}\\ \quad and\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j-m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\ 0\ else\end{cases}\quad (34)$$
$$for\ 0\le j<l.$$

恰好为M 的结果。

d） $L(M1)$ 不是补，$L(M2)$ 是补

令 $M1$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$M3$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M2)=\overline{L(M3)}$，则，$L(M)=L(M1)\circ L(M2)\ =L(M1)\circ\overline{L(M3)}$，$M$ 的模式串为 $P_m=p_0p_1\cdots p_{m1-1}\overline{p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}}$。

构造方法：第一步，按上一章的方法分别构造$M3$和$M1$；第二步按 4.2-D 的方法构造$\overline{L(M3)}$生成$\overline{R_{m2-1}}$，第三步把$M1$的结果R_{m1-1}使用 *m2* 个结果转换函数生成$R_{m1-1}^{'}$，最后把$\overline{R_{m2-1}}$和$R_{m1-1}^{'}$位与生成M 的结果R_{m-1}。

证明：由 4.2-D 算法证明结果知

$$\overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\ 0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad (35)$$
$$for\ 0\le j<l.$$

由 3.2.2 算法证明结果知

$$R_{m1-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\\ 0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad (36)$$
$$for\ 0\le j<l.$$

把$M1$的结果R_{m1-1}使用 *m2* 个结果转换函数生成

$$R_{m1-1}^{'j}=\begin{cases}1\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}=p_0p_1\cdots p_{m1-1}\\ 0\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad (37)$$
$$for\ 0\le j<l.$$

最后，最后把$\overline{R_{m2-1}^{'}}$和$R_{m1-1}^{'}$位与生成

$$R_{m-1}^{j}=R_{m1-1}^{'j}\ \&\ \overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m1-m2+1}t_{j-m1-m2+2}\cdots t_{j-m2}=p_0p_1\cdots p_{m1-1}\\ \quad and\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\ 0\ else\end{cases}\quad (38)$$
$$for\ 0\le j<l.$$

恰好为M 的结果。

C. 并。令$M1$和$M2$是两台本算法已实现的非确定型有穷自动机，要构造一台非确定型有穷自动机M 使得$L(M)=L(M1)\bigcup L(M2)$。

根据 D 的证明知，补的特性为从模式串的第一个字符的使能向量 $E_0 = \left[\overbrace{1,1,\cdots 1}^{l}\right]^T$ 开始，直到算法结果输出 $\overline{R_{m1-1}}$ 为止，必须保证补运算的完整性。因此在构造并运算时，需要分别对以下四种情况进行讨论。

a） $L(M1)$ 和 $L(M2)$ 都不是补。

令 $P_{m1} = p_0 p_1 \cdots p_{m1-1}$ ， $P_{m2} = p_0^{'} p_1^{'} \cdots p_{m2-1}^{'}$ ， 则 $L(M) = L(M1) \bigcup L(M2)$ ，$P_m = (p_0 p_1 \cdots p_{m1-1}) \bigcup (p_0^{'} p_1^{'} \cdots p_{m2-1}^{'})$。

构造方法：按上一章的方法分别构造 $M1$ 和 $M2$，而后把 $M1$ 的结果 R_{m1-1} 和 $M2$ 的的结果 R_{m2-1} 按位或，就是 M 的结果 R_{m-1}。

证明：由 3.2.2 算法证明结果知， $M1$ 和 $M2$ 的输出分别为

$$R_{m1-1}^{j} = \begin{cases} 1 \; if \; t_{j-m1+1} t_{j-m1+2} \cdots t_j = p_0 p_1 \cdots p_{m1-1} \\ 0 \; if \; t_{j-m1+1} t_{j-m1+2} \cdots t_j \neq p_0 p_1 \cdots p_{m1-1} \end{cases} \quad (39)$$
$$for \; 0 \le j < l.$$

$$R_{m2-1}^{j} = \begin{cases} 1 \; if \; t_{j-m2+1} t_{j-m2+2} \cdots t_j = p_0^{'} p_1^{'} \cdots p_{m2-1}^{'} \\ 0 \; if \; t_{j-m2+1} t_{j-m2+2} \cdots t_j \neq p_0^{'} p_1^{'} \cdots p_{m2-1}^{'} \end{cases} \quad (40)$$
$$for \; 0 \le j < l.$$

把 $M1$ 的结果 R_{m1-1} 和 $M2$ 的的结果 R_{m2-1} 按位或

$$R_{m-1}^{j} = R_{m1-1}^{j} \mid R_{m2-1}^{j} = \begin{cases} 1 \; if \; t_{j-m1+1} t_{j-m1+2} \cdots t_j = p_0 p_1 \cdots p_{m1-1} \\ \quad or \; t_{j-m2+1} t_{j-m2+2} \cdots t_j = p_0^{'} p_1^{'} \cdots p_{m2-1}^{'} \\ 0 \; else \end{cases} \quad (41)$$
$$for \; 0 \le j < l.$$

恰好为 M 的结果。

b） $L(M1)$ 和 $L(M2)$ 都是补。

令 $M3$ 的模式串为 $P_{m1} = p_0 p_1 \cdots p_{m1-1}$， $M4$ 的模式串为 $P_{m2} = p_0^{'} p_1^{'} \cdots p_{m2-1}^{'}$，$L(M1) = \overline{L(M3)}$ ， $L(M2) = \overline{L(M4)}$ ， 则 ， $L(M) = L(M1) \bigcup L(M2)$ $= \overline{L(M3)} \bigcup \overline{L(M4)}$， M 的模式串为 $P_m = \overline{p_0 p_1 \cdots p_{m1-1}} \bigcup \overline{p_0^{'} p_1^{'} \cdots p_{m2-1}^{'}}$。

构造方法：首先，按上一章的方法分别构造 $M3$ 和 $M4$，其次，按 4.2-D 的方法构造 $\overline{L(M3)}$ 和 $\overline{L(M4)}$，最后把 $\overline{L(M3)}$ 的结果 $\overline{R_{m1-1}}$ 和 $\overline{L(M4)}$ 的的结果 $\overline{R_{m2-1}}$ 按位或，就是 M 的结果 R_{m-1}。

证明：由 4.2-D 算法证明结果，知

$$\overline{R_{m1-1}^{j}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{42}$$

$$\overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{43}$$

$\overline{L(M3)}$ 的结果 $\overline{R_{m1-1}}$ 和 $\overline{L(M4)}$ 的的结果 $\overline{R_{m2-1}}$ 按位或

$$R_{m-1}^{j}=\overline{R_{m1-1}^{j}}\,|\,\overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\\\quad or\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{44}$$

恰好为 M 的结果。

c） $L(M1)$ 是补，$L(M2)$ 不是补。

令 $M3$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$M2$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M1)=\overline{L(M3)}$，则，$L(M)=L(M1)\bigcup L(M2)\ =\overline{L(M3)}\bigcup L(M2)$，$M$ 的模式串为 $P_m=\overline{p_0p_1\cdots p_{m1-1}}\bigcup(p_0^{'}p_1^{'}\cdots p_{m2-1}^{'})$。

构造方法：首先，按上一章的方法分别构造 $M3$ 和 $M2$，其次，按 4.2-D 的方法构造 $\overline{L(M3)}$，最后把 $\overline{L(M3)}$ 的结果 $\overline{R_{m1-1}}$ 和 $L(M2)$ 的结果 R_{m2-1} 按位或，就是 M 的结果 R_{m-1}。

证明：由 4.2-D 算法证明结果，知 $\overline{L(M3)}$ 的结果

$$\overline{R_{m1-1}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{45}$$

由 3.2.2 算法证明结果知，$L(M2)$ 的结果

$$R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{46}$$

$\overline{L(M3)}$ 的结果 $\overline{R_{m1-1}}$ 和 $\overline{L(M4)}$ 的的结果 $\overline{R_{m2-1}}$ 按位或

$$R_{m-1}^{j}=\overline{R_{m1-1}^{j}}\,|\,R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\\\quad or\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{47}$$

恰好为M 的结果。

d) $L(M1)$不是补，$L(M2)$是补。

令 $M1$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$M3$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M2)=\overline{L(M3)}$，则，$L(M)=L(M1)\bigcup L(M2)\ =L(M1)\bigcup\overline{L(M3)}$，$M$ 的模式串为 $P_m=(p_0p_1\cdots p_{m1-1})\bigcup\overline{p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}}$。

构造方法：首先，按上一章的方法分别构造$M3$和$M1$，其次，按 4.2-D 的方法构造$\overline{L(M3)}$，最后把$\overline{L(M3)}$的结果$\overline{R_{m2-1}}$和$L(M1)$的结果R_{m1-1}按位或，就是M 的结果R_{m-1}。

证明：由 4.2-D 算法证明结果，知$\overline{L(M3)}$的结果

$$\overline{R_{m2-1}^{j}}=\begin{cases}1\,if\ t_{j-m2+1}t_{j-m2+2}\cdots t_j\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\,if\ t_{j-m2+1}t_{j-m2+2}\cdots t_j=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad(48)$$
$$for\ 0\le j<l.$$

由 3.2.2 算法证明结果，知$L(M1)$的结果

$$R_{m1-1}^{j}=\begin{cases}1\,if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j=p_0p_1\cdots p_{m1-1}\\0\,if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad(49)$$
$$for\ 0\le j<l.$$

把$\overline{L(M3)}$的结果$\overline{R_{m2-1}}$和$L(M1)$的结果R_{m1-1}按位或，得到

$$R_{m-1}^{j}=R_{m1-1}^{j}\mid\overline{R_{m2-1}^{j}}=\begin{cases}1\,if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j=p_0p_1\cdots p_{m1-1}\\\quad or\ t_{j-m2+1}t_{j-m2+2}\cdots t_j\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\,else\end{cases}\quad(50)$$
$$for\ 0\le j<l.$$

恰好为M 的结果。

D. Kleene 星号。为了论述方便，本文仅证明形如 $P_m=p_0p_1\cdots p_{m1-1}(p_0^{'}p_1^{'}\cdots p_{m2-1}^{'})^{*}p_0^{''}p_1^{''}\cdots p_{m3-1}^{''}$的包含 Kleene 星号的模式串。

令$M1$的模式串为$P_{m1}=p_0p_1\cdots p_{m1-1}$，$M4$的模式串为$P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$M3$的模式串为 $p_0^{''}p_1^{''}\cdots p_{m3-1}^{''}$，$L(M2)=L(M4)^{*}$，则，$L(M)=L(M1)\circ L(M2)\circ L(M3)$ $=L(M1)\circ L(M4)^{*}\circ L(M3)$，$M$ 的模式串为 $P_m=p_0p_1\cdots p_{m1-1}(p_0^{'}p_1^{'}\cdots p_{m2-1}^{'})^{*}p_0^{''}p_1^{''}\cdots p_{m3-1}^{''}$。[58]

构造方法：第一步，按上一章的方法分别构造$M1$，$M3$和$M4$；第二步，用$M1$的RE_{m1-1}和$M4$的RE_{m2-1}按位或后的结果$E_0^{'}$，代替$M4$的E_0；第三步，同样用$E_0^{'}$代替$M3$的E_0；最后，$M3$的结果R_{m3-1}，就是M 的结果R_{m-1}

证明：由 3.2.2 算法证明结果，知$L(M1)$的结果

$$R_{m1-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{51}$$

$L(M4)$ 的结果

$$R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{52}$$

$$E_0^{'j}=RE_{m1-1}^{j}\,|\,RE_{m2-1}^{j}=R_{m1-1}^{j-1}\,|\,R_{m2-1}^{j-1}=$$

$$\begin{cases}1\ if\ \exists i\in N(\text{自然数集合}),t_{j-m1-i*m2}t_{j-m1-i*m2+1}\cdots t_{j-i*m2-1}(t_{j-m2}t_{j-m2+1}\cdots t_{j-1})^{i}\\\qquad =p_0p_1\cdots p_{m1-1}(p_0^{'}p_1^{'}\cdots p_{m2-1}^{'})^{i}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{53}$$

与 3.2.2 算法类似的证明，有

$$R_{m-1}^{j}=\begin{cases}1\ if\ \exists i\in N,t_{j-m1-i*m2-m3+1}t_{j-m1-i*m2-m3+2}\cdots t_{j-i*m2-m3}(t_{j-m3-m2+1}t_{j-m2-m3+2}\cdots t_{j-m3})^{i}\\\qquad t_{j-m3+1}t_{j-m3+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}(p_0^{'}p_1^{'}\cdots p_{m2-1}^{'})^{i}\,p_0^{''}p_1^{''}\cdots p_{m3-1}^{''}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{54}$$

恰好为 M 的结果。

E. 交。令 $M1$ 和 $M2$ 是两台本算法已实现的非确定型有穷自动机，要构造一台非确定型有穷自动机 M 使得 $L(M)=L(M1)\bigcap L(M2)$。

根据 D 的证明知，补的特性为从模式串的第一个字符的使能向量 $E_0=\left[\overbrace{1,1,\cdots 1}^{l}\right]^{T}$ 开始，直到算法结果输出 $\overline{R_{m1-1}}$ 为止，必须保证补运算的完整性。因此在构造并运算时，需要分别对以下四种情况进行讨论。

a) $L(M1)$ 和 $L(M2)$ 都不是补。

令 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，则 $L(M)=L(M1)\bigcap L(M2)$，$P_m=(p_0p_1\cdots p_{m1-1})\bigcap(p_0^{'}p_1^{'}\cdots p_{m2-1}^{'})$。

构造方法：首先，按上一章的方法分别构造 $M3$ 和 $M4$，其次，按 4.2-D 的方法构造 $\overline{L(M3)}$ 和 $\overline{L(M4)}$，最后把 $\overline{L(M3)}$ 的结果 $\overline{R_{m1-1}}$ 和 $\overline{L(M4)}$ 的的结果 $\overline{R_{m2-1}}$ 按位与，就是 M 的结果 R_{m-1}。

证明：由 3.2.2 算法证明结果知，$M1$ 和 $M2$ 的输出分别为

$$R_{m1-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{55}$$

$$R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{56}$$

把$M1$的结果R_{m1-1}和$M2$的的结果R_{m2-1}按位与

$$R_{m-1}^{j}=R_{m1-1}^{j}\ \&\ R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\\\quad and\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{57}$$

恰好为M 的结果。

b) $L(M1)$和$L(M2)$都是补

令 $M3$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$M4$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M1)=\overline{L(M3)}$，$L(M2)=\overline{L(M4)}$，则，$L(M)=L(M1)\bigcap L(M2)\ =\overline{L(M3)}\bigcap\overline{L(M4)}$，$M$ 的模式串为$P_m=\overline{p_0p_1\cdots p_{m1-1}}\bigcap\overline{p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}}$

构造方法：首先，按上一章的方法分别构造$M3$和$M4$，其次，按 4.2-D 的方法构造$\overline{L(M3)}$和$\overline{L(M4)}$，最后把$\overline{L(M3)}$的结果$\overline{R_{m1-1}}$和$\overline{L(M4)}$的的结果$\overline{R_{m2-1}}$按位与，就是M 的结果R_{m-1}。

证明：由 4.2-D 算法证明结果，知

$$\overline{R_{m1-1}^{j}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{58}$$

$$\overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{59}$$

$\overline{L(M3)}$的结果$\overline{R_{m1-1}}$和$\overline{L(M4)}$的的结果$\overline{R_{m2-1}}$按位与

$$R_{m-1}^{j}=\overline{R_{m1-1}^{j}}\ \&\ \overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_0p_1\cdots p_{m1-1}\\\quad or\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{60}$$

恰好为M 的结果。

c） $L(M1)$ 是补，$L(M2)$ 不是补。

令 $M3$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$M2$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M1)=\overline{L(M3)}$，则，$L(M)=L(M1)\bigcap L(M2)\ =\overline{L(M3)}\bigcap L(M2)$，$M$ 的模式串为 $P_m=\overline{p_0p_1\cdots p_{m1-1}}\bigcap(p_0^{'}p_1^{'}\cdots p_{m2-1}^{'})$。

构造方法：首先，按上一章的方法分别构造 $M3$ 和 $M2$，其次，按 4.2-D 的方法构造 $\overline{L(M3)}$，最后把 $\overline{L(M3)}$ 的结果 $\overline{R_{m1-1}}$ 和 $L(M2)$ 的结果 R_{m2-1} 按位与，就是 M 的结果 R_{m-1}。

证明：由 4.2-D 算法证明结果，知 $\overline{L(M3)}$ 的结果

$$\overline{R_{m1-1}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j\neq p_0p_1\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j=p_0p_1\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{61}$$

由 3.2.2 算法证明结果知，$L(M2)$ 的结果

$$R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_j=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_j\neq p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{62}$$

$\overline{L(M3)}$ 的结果 $\overline{R_{m1-1}}$ 和 $\overline{L(M4)}$ 的的结果 $\overline{R_{m2-1}}$ 按位与

$$R_{m-1}^{j}=\overline{R_{m1-1}^{j}}\ \&\ R_{m2-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_j\neq p_0p_1\cdots p_{m1-1}\\\quad and\ t_{j-m2+1}t_{j-m2+2}\cdots t_j=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{63}$$

恰好为 M 的结果。

d） $L(M1)$ 不是补，$L(M2)$ 是补。

令 $M1$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$M3$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M2)=\overline{L(M3)}$，则，$L(M)=L(M1)\bigcap L(M2)\ =L(M1)\bigcap\overline{L(M3)}$，$M$ 的模式串为 $P_m=(p_0p_1\cdots p_{m1-1})\bigcap\overline{p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}}$。

构造方法：首先，按上一章的方法分别构造 $M3$ 和 $M1$，其次，按 4.2-D 的方法构造 $\overline{L(M3)}$，最后把 $\overline{L(M3)}$ 的结果 $\overline{R_{m2-1}}$ 和 $L(M1)$ 的结果 R_{m1-1} 按位与，就是 M 的结果 R_{m-1}。

证明：由 4.2-D 算法证明结果，知 $\overline{L(M3)}$ 的结果

$$\overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_{0}^{'}p_{1}^{'}\cdots p_{m2-1}^{'}\\0\ if\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}=p_{0}^{'}p_{1}^{'}\cdots p_{m2-1}^{'}\end{cases}\quad for\ 0\le j<l. \tag{64}$$

由 3.2.2 算法证明结果，知 $L(M1)$ 的结果

$$R_{m1-1}^{j}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_{0}p_{1}\cdots p_{m1-1}\\0\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}\neq p_{0}p_{1}\cdots p_{m1-1}\end{cases}\quad for\ 0\le j<l. \tag{65}$$

把 $\overline{L(M3)}$ 的结果 $\overline{R_{m2-1}}$ 和 $L(M1)$ 的结果 R_{m1-1} 按位与，得到

$$R_{m-1}^{j}=R_{m1-1}^{j}\mid\overline{R_{m2-1}^{j}}=\begin{cases}1\ if\ t_{j-m1+1}t_{j-m1+2}\cdots t_{j}=p_{0}p_{1}\cdots p_{m1-1}\\\quad and\ t_{j-m2+1}t_{j-m2+2}\cdots t_{j}\neq p_{0}^{'}p_{1}^{'}\cdots p_{m2-1}^{'}\\0\ else\end{cases}\quad for\ 0\le j<l. \tag{66}$$

恰好为 M 的结果。

由以上的证明可知本文提出的 “向量与”算法能够完成正则表达式与正文串 Tn 的匹配，匹配结果存放于与每个正则表达式相对应的最后一级 R_{m-1} 中。

4.2.2 “向量与”算法与 NFA 的等价性证明

在第三章与第四章中证明了“向量与”算法的正确性。表明该算法能正确在处理固定模式串和正则表达式匹配。以上证明仅表明“向量与”算法是正则表达式匹配的充分条件。由于正则表达式的是指一个用来描述符合某个句法规则的字符串集合的单个字符串。一个正则表达式可以匹配最多无限多个字符串。前面的证明说明本文所提出的算法能正确地构造出匹配正则表达式描述的所有字符串。另外还要证明算法构造的自动机所能匹配的字符串集合与正则表达式描述的字符串集合完全相同，也就是要证明“向量与”算法等价于正则表达式。在本章4.1.2中说明正则表达式与NFA是等价的，只需证明“向量与”算法与NFA等价，就可证明“向量与”算法等价于正则表达式。本节一方面完成等价性证明，另一方面在证明过程同时表明该算法可以完成所有的计算任务。通过第六章的实验与仿真过程说明在现有FPGA上一个时钟周期内能够完成所有的计算任务，同时在第六章中给出最差情况下的最大时钟周期和最大吞吐率的理论值[59]。

在参考文献[18]中在以下定理，下面给出相应的非确定型有穷自动机的算法在做正则表达式匹配时的简单算法描述。该定理及算法说明了完成正则表达式匹配所需的所有计算任务。

定理4[18] ：如果L是一个正则语言，则有一个算法，任给 $w\in\Sigma^{*}$，它在 $O(|w|)$ 时间内检查 w 是否在L中。

相应的算法 S_n 简述为：设M = (K,Σ,Δ,s,F)是一台非确定型有穷自动机。每次处理一个正

文字符。在处理n-1个字符后，内部活跃状态集为$S_{n-1}\subseteq K$。在处理第n个字符后，内部活跃状态集变为$S_n=\cup\{E(q):\text{对某个}p\in S_{n-1},(p,t_n,q)\in\Delta\}\subseteq K$，其中$E(q)$表示$\{p:(q,e)\overset{*}{\mapsto}_M(p,e)\}$。在运算过程中如果$S_{n-1}\cap F\neq\phi$，表示正文与L匹配。

为了说明“向量与”算法与上述算法S_n等价，需要证明：上述算法的活跃状态集$S_n\subseteq K$是，本文提出的“向量与”算法在处理第n个正文字符时，对应R_i中为1的集合，即如果第n个字符t_n对应于t_j，则$S_n=\{R_i^j:R_i^j=1,0\le i<m,0\le j<l\}$。

分两步证明：

首先证明：$S_n\subseteq\{R_i^j:R_i^j=1,0\le i<m,0\le j<l\}$。

针对S_n的下标n，使用数学归纳法，证明：

1、基本步骤。n=1，S_1的元素表示正文字符与模式串第一个字符匹配，由前面的证明知，如果输入的第0个正文与模式串第一个字符匹配，即$t_0=p_0$，则“向量与”算法中，

$$\left\{\begin{array}{l}E_0=\left[\overbrace{1,1,\cdots1}^{l}\right]^T\Rightarrow E_0^j=1\\ j=0\\ \left\{\begin{array}{l}C_0^j=\left\{\begin{array}{l}1\ if\ t_j=p_0\\0\ if\ t_j\neq p_0\end{array}\right\}\Rightarrow C_0^j=1\\ t_0=p_0\end{array}\right.\end{array}\right\}\tag{67}$$

$$\overset{\text{定理2}}{\Rightarrow}R_0^j=E_o^j\ \&\ C_0^j=1\&1=1$$

因此$S_1\subseteq\{R_0^j:R_0^j=1,0\le j<l\}$。

如果输入的第0个正文与模式串第一个字符不匹配，则，$S_1=\phi$。另外$\forall0\le i<m,0\le j<l,R_i^j=0\Rightarrow\{R_i^j:R_i^j=1,0\le i<m,0\le j<l\}=\phi$，

$S_1\subseteq\{R_0^j:R_0^j=1,0\le j<l\}$。

综上所述$S_1\subseteq\{R_0^j:R_0^j=1,0\le j<l\}$。

2、归纳假设。设$k\ge1$，并且当$n=k$，有$S_n\subseteq\{R_i^j:R_i^j=1,0\le i<m,0\le j<l\}$成立。

3、归纳步骤。当$n=k+1$时，由归纳假设知：匹配第n-1个正文字符后，有$S_{n-1}\subseteq\{R_{i-1}^{j-1}:R_{i-1}^{j-1}=1,0<i<m,0\le j<l\}$有成立，相应地，匹配第n个正文字符时，$S_n=\cup\{E(q):\text{对某个}p\in S_{n-1},(p,t_n,q)\in\Delta\}\subseteq K$知对于任意的$q\in S_n$。有：

$$
\left.\begin{array}{l}\forall q \in S_n \\ \text{设第} n \text{个字符} t_n \text{对应于} t_j \\ S_n = \cup\{E(q): \text{对某个} p \in S_{n-1}, (p, t_n, q) \in \Delta\}\end{array}\right\}
$$

$$
\Rightarrow \left\{\begin{array}{l}\exists s_{n-1} \in S_{n-1} \Rightarrow \exists i, \text{使得} t_{j-i-1} t_{j-i} \cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \\ t_n = t_j \\ (p, t_n, q) \in \Delta \Rightarrow t_n = p_i\end{array}\right\}
$$

$$
\Rightarrow \left\{\begin{array}{l} t_{j-i-1} t_{j-i} \cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \Rightarrow R_{i-1}^{j-1} = 1 \\ t_j = p_i \end{array}\right\} \tag{68}
$$

$$
\overset{\text{结果使能函数}}{\Rightarrow} \left\{\begin{array}{l} RE_{i-1}^{j} = 1 \\ C_i^j = 1 \end{array}\right\} \Rightarrow \left\{\begin{array}{l} E_i^j = 1 \\ C_i^j = 1 \end{array}\right\} = R_i^j = 1
$$

$$
\Rightarrow S_n \subseteq \{R_i^j : R_i^j = 1, 0 \le i < m, 0 \le j < l\}
$$

综上所述，算法 S_n 的活跃状态集是“向量与”算法结果集 $\{R_i^j : R_i^j = 1, 0 \le i < m, 0 \le j < l\}$ 的子集。

其次证明：$\{R_i^j : R_i^j = 1, 0 \le i < m, 0 \le j < l\} \subseteq S_n$。

证明：因为，

$$
\left.\begin{array}{l}\forall R_i^j = 1 \Rightarrow t_{j-i} t_{j-i+1} \cdots t_j = p_0 p_1 \cdots p_i \\ \text{第} n \text{个字符} t_n \text{对应于} t_j \Rightarrow t_n = t_j\end{array}\right\} \Rightarrow \exists q \in S_n \text{接受字符串} t_0 t_1 \cdots t_n \tag{69}
$$

所以，$\{R_i^j : R_i^j = 1, 0 \le i < m, 0 \le j < l\} \subseteq S_n$。

由以上两步证明，有 $S_n = \{R_i^j : R_i^j = 1, 0 \le i < m, 0 \le j < l\}$ 成立。

通过之前的算法构造、证明和上述证明，可以有以下结论：

1、“向量与”算法的时间复杂度为 $O(|w|)$。通过本节证明知“向量与”算法与定理4是等价算法，而定理4中的算法复杂度是 $O(|w|)$，因此，“向量与”算法的复杂度是 $O(|w|)$。

2、在FPGA 内实现“向量与”算法时，因为“向量与”算法通过把正则表达式算子转化为基本逻辑单元，充分利用了FPGA的计算资源，所以“向量与”算法可以在一个时钟周期内计算出 $\{R_i^j : R_i^j = 1, 0 \le i < m, 0 \le j < l\}$ 集合中的所有的元素。如果以时钟周期为记时单位，“向量与”算法的运行时间复杂度是 $O(1)$。

4.2.3 “向量与”算法构造 NFA 示例

之前的章节，通过理论证明，完成了算法正确性的证明。本节以 SNORT 2.8 中一些规则做为示例，举例说明“向量与”算法的实际应用。为避免重复，这里只说明构造方法，不再证明，证明过程可参考 4.2.1 中的过程。假定每时钟周期输入 32 个正文字符。

例 1：规则取自 snortrules-snapshot-2.8/rules/sql.rules 中一条：

alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 7210
(msg:"SQL SAP MaxDB shell command injection attempt";
flow:established,to_server;
content:"exec_sdbinfo"; nocase;
pcre:"/exec_sdbinfo\s+[\x26\x3b\x7c\x3e\x3c]/i";
metadata:policy balanced-ips drop, policy security-ips drop;
reference:bugtraq,27206; reference:cve,2008-0244; classtype:attempted-admin;
sid:13356; rev:1;)，

其中正则表达式模式串为“exec_sdbinfo\s+[\x26\x3b\x7c\x3e\x3c]”。

1、首先构造公用的解码矩阵，完成每时钟周期处理正文T_{32}的解码任务。输入为32个正文字符，输出为32x256的解码矩阵。

2、针对模式串中第一个字符（ p_0=e）匹配的构造方法：构建字符比较模块，令 $E_0 = \left[\overbrace{1,1,\cdots 1}^{l}\right]^T$ ；因为字符e的编码为101，所以根据e取解码矩阵中的第101列，该列就是输入的32个字符与模式符e的比较结果C_0，即$C_0 = \left[A_{j,k}\right]$,其中 $0 \le j < 32,$，$k = 101$；使用字符比较模块将E_0和C_0映射为R_0；最后使用结果转换函数将R_0映射为RE_0。

3、针对模式串中从第一个字符e到\s之间的所有字符（ p_i其中$0 < i < 12$ ）匹配的构造方法：构建字符比较模块，令$E_i = RE_{i-1}$；同样根据 p_i 从解码矩阵中取得模式比较结果C_i，即$C_i = \left[A_{j,k}\right]$,其中 $0 < \mathrm{i} < 12, 0 \le j < 32, k = p_i$；使用字符比较模块将$E_i$和$C_i$映射为$R_i$；最后使用结果转换函数将$R_i$映射为$RE_i$。这一串的构造最终会生成$RE_{11}$。

4、针对模式串中第12个模式符“\s”和算子“+”。这个组合要求匹配至少一个空白符，等价于至少匹配[\f\n\r\t\v]其中的一个，所以这里要求处理5个字符的或。这里需要构造5个匹配模块，这些字符与输入的32个字符比较结果C_i分别相对应于解码矩阵中的第12、10、13、9、11列。每个模块的E_i都是上一个字符o的输出RE_{11}与本模式符\s的RE_{12}经按位或后的结果，相同的E_i与各自的C_i生成各自R_i后，将这5个R_i按位或后生成\s的R_{12}，最后使用结果转换函数将\s的R_{12}映射为\s的RE_{12}；

5、针对模式串中最后一个模式符[\x26\x3b\x7c\x3e\x3c]。这个模式符与4中类似，还是5个字符的或。这里仍然需要构造5个匹配模块，这些字符与输入的32个字符比较结果C_i分别相对应于解码矩阵中的第38、59、124、62、60列。每个模块的E_i都是上一个字符\s的输出RE_{12}，相同的E_i与各自的C_i生成各自R_i后，将这5个R_i按位或后生成模式符[\x26\x3b\x7c\x3e\x3c]的R_{13}。

6、最后一级生成的R_{13}就是正则表达式串exec_sdbinfo\s+[\x26\x3b\x7c\x3e\x3c]与正文串T_n的匹配结果。

由以上构造可以看出：因为 “+” 算子和[\x26\x3b\x7c\x3e\x3c]的存在，所以对该正则表达式的构造分为了三部分完成，第一部分是构造固定模式串“exec_sdbinfo”，第二部分构造包含“+” 算子的模式串“\s+”，最后构造“[\x26\x3b\x7c\x3e\x3c]”五个字符并的模式符。

例 2：规则取自 snortrules-snapshot-2.8/rules/sql.rules 中一条：

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"SQL generic sql update injection attempt";
flow:established,to_server;
content:"update"; nocase;
pcre:"/[^A-Z\x2B]update[^\n]*set/i";
metadata:policy security-ips drop, service http;
reference:url,www.securiteam.com/securityreviews/5DP0N1P76E.html;
classtype:web-application-attack; sid:13514; rev:5;)，

其中正则表达式模式串为“[^A-Z\x2B]update[^\n]*set”。

1、首先构造公用的解码矩阵，完成每时钟周期处理正文T_{32}的解码任务。输入为32个正文字符，输出为32x256的解码矩阵。

2、针对模式串中第一个模式符[^A-Z\x2B]匹配的构造方法：本模式符需要通配除A-Z和\x2B外的所有字符，因此，需要构建27个字符比较模块。每个模块E_0都是相同的，$E_0=\left[\overbrace{1,1,\cdots 1}^{l}\right]^T$；这些字符与输入的32个字符比较结果$C_i$分别相对应于解码矩阵中的第43列、65列到90列。使用字符比较模块将E_0和C_i映射为R_i，将这27个R_i经按位或非门后生成[^A-Z\x2B]的R_0；最后使用结果转换函数将R_0映射为RE_0。

3、针对模式串中从第一个模式符[^A-Z\x2B]到[^\n]之间的所有字符“update”（p_i其中$0<i<7$）的构造方法：构建字符比较模块，令$E_i=RE_{i-1}$；同样根据p_i从解码矩阵中取得模式比较结果C_i，即$C_i=\left[A_{j,k}\right]$,其中 $0<\mathrm{i}<7,0\le j<32,k=p_i$；使用字符比较模块将$E_i$和$C_i$映射为$R_i$；最后使用结果转换函数将$R_i$映射为$RE_i$。这一串的构造最终会生成$RE_6$。

4、针对模式串中第12个模式符“[^\n]”和算子“*”。这个组合要求匹配零个或多个非换行符，所以这里需要构造一个匹配模块。同时这里是前面不是非与后面是非因此使用4.2.1-B-（d）的构造方法。\n与输入的32个字符比较结果C_i对应于解码矩阵中的第10列。这个模块的$E_i=\left[\overbrace{1,1,\cdots 1}^{l}\right]^T$，由于这里$E_i$的中每个元素都为1，因此

不需要再考虑算子“*”。E_i 与C_i 生成R_i 。这个R_i 按位非后再和RE_6 按位或生成R_7，最后使用结果转换函数将R_7 映射为RE_7；

5、针对模式串中模式符“se”的构造方法：p_i 其中$7 < i < 10$ 构建字符比较模块。同时因为前一级中包含算子“*”，所以令$E_8 = RE_7 \mid RE_6$，同时$E_9 = RE_8$；同样根据p_i 从解码矩阵中取得模式比较结果C_i，即$C_i = [A_{j,k}]$，其中 $7 < \mathrm{i} < 10, 0 \le j < l, k = p_i$；使用字符比较模块将$E_i$ 和C_i 映射为R_i；最后使用结果转换函数将R_i 映射为RE_i。这一串的构造最终会生成RE_9。

6、针对模式串中最后一个模式符“t” 的构造方法：令$E_{10} = RE_9$；同样根据t从解码矩阵中取得模式比较结果C_{10}对应于解码矩阵中的第116列，即$C_{10} = [A_{j,k}]$，其中 $0 \le j < 32, k = 116$；使用字符比较模块将E_{10}和C_{10}映射为R_{10}。

7、最后一级生成的R_{10}就是正则表达式串[^A-Z\x2B]update[^\n]*set与正文串T_n的匹配结果。

4.3 “向量与”算法小结

通过上节的证明可以得到以下结论：

1、“向量与”算法与 NFA 等价。

2、本文提出的“向量与”算法可以根据任一个正则表达式构造一台非确定有穷自动机接受该正则表达式。

3、如果以时钟周期为记时单位，“向量与”算法的运行时间复杂度是$O(1)$。

4、解决了补运算。由于补运算的特殊性，补运算一直是正则表达式匹配算法实现中的一个难点，在各种公开发表的文献中少有解决补运算实现的方案。本文提出了补运算一种解决方案。在后面改进方案中将对补运算的解决做进一步探讨。

第五章 “向量与”算法改进

前面的章节构造与证明了算法的正确性。本章分别对细节进行改进，进一步在资源占用上给出更好的实现方案。

5.1 单字符改进方案

由前两章的构造和证明，“向量与”算法中对每个模式字符的匹配需要构造如下所示的所有部件

1、使能向量 E_i 。

2、模式比较结果 C_i 。

3、结果向量 R_i 。

4、结果使能向量 RE_i 。

实际使用过程中存在大量单字符间的与、或操作，这种情况下如果仍然使用“向量与”算法直接构造这些操作，会占用大量的逻辑资源，降低了芯片内可用模式字符的总量，针对这种情况，本文提出以下解决方案，降低资源占用量。

5.1.1 扩展解码矩阵

这种方案是针对经常使用的字符间与或操作，特别是对单字符的非、某些特殊要求的或操作，如：对所有字母、所有数字、范围和多个字符，对应于 Snort-PCRE 语言中的“., [abc], [a-fA-F0-9], \d, \w, \s,\D, \W, \S, \N”等匹配的要求的条件下，直接使用“向量与”算法，会浪费大量的逻辑资源。如，极端情况（“.”要求匹配所有字符）下，直接使用“向量与”算法，就要求使用 256 组 C_i 、用于生成 R_i 的与门、 R_i ，用于每个模式符的匹配以及大量的或门完成模式符之间的或关系，最终生成“.”所要求计算任务，才能完成该项匹配要求。

我们可以通过以下方法直接扩展解码矩阵来完成该项任务，并大量减少资源占用量：

增加“向量与”算法的解码矩阵中的列数，完成上述常用符号的匹配。也就需要改写解码矩阵的定义：

$$A_{i,j}=\begin{cases}1\ if\ t_i=j\\0\ if\ t_i\neq j\end{cases}\qquad for\ \ 0\le i<l,\ 0\le j<256 \tag{70}$$

通过扩大 j 的取值范围，让增加的每一列分别代表每一种要扩展的情况，引入符号 y_j 来代表第 j 列的模式，扩展解码矩阵的前 256 列依然与解码矩阵中的 256 列相同都代表单个字符。需要扩展的增加相应的列，如：第 256 列代表要匹配“.”，则 $y_{256}=.$， y_{256} 代表了所有字符的集合，这种条件下，无论输入的正文字符是什么，扩展解码矩阵中第 256 列的

每个元素都为 1。因此相应原来解码矩阵中的相等比较$t_i = 256$，就需要改为元素是否在集合的判断$t_j \in y_{256}$。如：第 257 列代表要匹配“[abc]”，则$y_{257} = [abc]$，y_{257}代表了正文字符是 a、b 或 c 中的任意一个都与模式符匹配，这种条件下，输入的正文字符是 a、b 和 c 其中之一时，扩展解码矩阵中第 257 列中相应的元素为 1。因此相应相等比较$t_i = 257$，就需要改为元素是否在集合的判断$t_j \in y_{257}$。于是上述定义就需要改为：

$$A_{i,j} = \begin{cases} 1 \ if \ t_i \in y_j \\ 0 \ if \ t_i \notin y_j \end{cases} \quad for \ 0 \le i < l, \ 0 \le j \tag{71}$$

相应地，对于模式串中所使用的所有模式字符都应该做相应的修改，由p_i改为y_i。由此对于本文定理 1 的证明也做下列修改成为新的定理：

定理 5：如果$C_i = [A_{j,k}]$,其中$0 \le i < m, 0 \le j < l, \mathrm{y_k} = y_i$，A 是解码矩阵有下式成立

$$C_i^j = \begin{cases} 1 \ if \ t_j \in y_i \\ 0 \ if \ t_j \notin y_i \end{cases} \quad for \ 0 \le i < m, \ 0 \le j < l \tag{72}$$

证明：$\forall i, j, C_i = [A_{j,k}]$,其中$0 \le i < m, 0 \le j < l, \mathrm{y_k} = y_i \Rightarrow$

$$\left\{ \begin{array}{l} C_i^j = A_{j,k} \\ \mathrm{y_k} = y_i \\ \\ A_{j,k} = \begin{cases} 1 if \ t_j \in y_k \\ 0 if \ t_j \notin y_k \end{cases} \end{array} \right\} \Rightarrow C_i^j = \begin{cases} 1 if \ t_j \in y_i \\ 0 if \ t_j \notin y_i \end{cases} \quad for \ 0 \le i < m, 0 \le j < l \tag{73}$$

证毕。

根据定理 5 知，通过以上方案生成C_i，完成了一个时钟周期内输入的所有正文 $\mathrm{T_l}$ 与模式串的第 i 个模式的比较操作：判断输入的第 j 个正文字符t_j是否是第 i 个模式y_i的元素，C_i中的每个元素代表了比较结果。

使能向量E_i，原本定义为

$$E_i^j = \begin{cases} 1 if \ i = 0 \ \ or \ \ t_{j-i} t_{j-i+1} \cdots t_{j-1} = p_0 p_1 \cdots p_{i-1} \\ 0 if \ i \ne 0 \ and \ \ t_{j-i} t_{j-i+1} \cdots t_{j-1} \ne p_0 p_1 \cdots p_{i-1} \end{cases} \quad for 0 \le i < m, 0 \le j < l. \tag{74}$$

只需将定义的条件改为

$$E_i^j = \begin{cases} 1 if \ i = 0 \ \ or \ \ \forall k, t_{j-k-1} \in y_{i-k-1} \\ 0 if \ \exists k, t_{j-k-1} \notin y_{i-k-1} \end{cases} \quad for 0 \le i < m, 0 \le j < l, 0 \le k < i. \tag{75}$$

即可。其中取值和结构并不发生变化。

结果向量R_i^j的定义也需要由原来的

$$R_i^j=\begin{cases}1\ if\ t_{j-i}t_{j-i+1}\cdots t_j=p_0p_1\cdots p_i\\0\ if\ t_{j-i}t_{j-i+1}\cdots t_j\neq p_0p_1\cdots p_i\end{cases}\quad for 0\le i<m,0\le j<l. \tag{76}$$

改为

$$R_i^j=\begin{cases}1\ if\ i=0\ \ or\ \forall k,t_{j-k}\in y_{i-k}\\0\ if\ \exists k,t_{j-k}\notin y_{i-k}\end{cases}\quad for\ 0\le i<m,0\le j<l,0\le k\le i. \tag{77}$$

相应地定理 2 的证明也做下列修改成为新的定理

定理 6：如果使能向量E_i与模式比较结果C_i进行位与运算，则得到的结果为结果函数R_i，即$R_i^j=C_i^j\ \&\ E_i^j$。

证明：$\forall i,j,$其中$0\le i<m,0\le j<l$

c) 假设：$E_i^j=0\Rightarrow$

$$\begin{Bmatrix}\overset{等式(75)}{\Rightarrow}\exists k,(0\le k<i),t_{j-k-1}\notin y_{i-k-1}\\\Rightarrow E_i^j\ \&\ C_i^j=0\end{Bmatrix}$$

$$\Rightarrow\begin{Bmatrix}\exists k,(0\le k\le i),t_{j-k}\notin y_{i-k}\overset{等式(77)}{\Rightarrow}R_i^j=0\\E_i^j\ \&\ C_i^j=0\end{Bmatrix} \tag{78}$$

$$\Rightarrow R_i^j=E_i^j\ \&C_i^j.$$

d) 假设：$E_i^j=1\ and\ C_i^j=0\Rightarrow$

$$\begin{Bmatrix}E_i^j=1\overset{等式(75)}{\Rightarrow}\begin{Bmatrix}E_i^j\ \&\ C_i^j=C_i^j\\\forall k(0\le k<i),t_{j-k-1}\in y_{i-k-1}\end{Bmatrix}\\C_i^j=0\overset{定理5}{\Rightarrow}t_j\notin y_i\end{Bmatrix}$$

$$\Rightarrow\begin{Bmatrix}E_i^j\ \&\ C_i^j=C_i^j=0\\\exists k,(0\le k\le i),t_{j-k}\notin y_{i-k}\overset{等式(77)}{\Rightarrow}R_i^j=0\end{Bmatrix} \tag{79}$$

$$\Rightarrow R_i^j=E_i^j\ \&\ C_i^j$$

e) 假设：$E_i^j=1\ and\ C_i^j=1\Rightarrow$

$$\begin{Bmatrix}E_i^j=1\overset{等式(75)}{\Rightarrow}\begin{Bmatrix}E_i^j\ \&\ C_i^j=C_i^j\\\forall k(0\le k<i),t_{j-k-1}\in y_{i-k-1}\end{Bmatrix}\\C_i^j=1\overset{定理5}{\Rightarrow}t_j\in y_i\end{Bmatrix}$$

$$\Rightarrow \left\{ \begin{array}{l} E_i^j \ \& \ C_i^j = C_i^j = 1 \\ \forall k, (0 \le k \le i), t_{j-k} \in y_{i-k} \end{array} \right\} \overset{\text{等式（77）}}{\Rightarrow} R_i^j = 1 \tag{80}$$

$$\Rightarrow R_i^j = E_i^j \ \& C_i^j$$

综上所述，有 $R_i^j = E_i^j \ \& C_i^j$ 成立。

同样，其中取值和结构并不发生变化。

另外，为表明“向量与”算法其它部分取值和结构不受扩展解码矩阵改变的影响，改写第三章算法中 3.3.2 节固定模式串数学归纳法中的归纳步骤证明过程如下所示：

设固定模式串的长度 $|P|$ 为 m+1，则 $P_m = y_0 y_1 \cdots y_m$。根据归纳假设，对于 $y_0 y_1 \cdots y_{m-1}$ 的匹配，R_{m-1} 中记录了匹配结果，

$$\begin{array}{l} R_{m-1}^j = \left\{ \begin{array}{l} 1 \ if \ \forall k, (0 \le k \le m-1), t_{j-k} \in y_{m-1-k} \\ 0 \ if \ \exists k, (0 \le k \le m-1), t_{j-k} \notin y_{m-1-k} \end{array} \right. \\ for \ 0 \le j < l. \end{array} \tag{81}$$

根据构造和结果使能函数知，

$$\begin{array}{l} E_m^j = RE_{m-1}^j = R_{m-1}^{j-1} = \left\{ \begin{array}{l} 1 \ if \ \forall k, t_{j-1-k} \in y_{m-1-k} \\ 0 \ if \ \exists k, t_{j-1-k} \notin y_{m-1-k} \end{array} \right. \\ for \ 0 \le j < l, 0 \le k \le m-1. \end{array} \tag{82}$$

根据扩展解码矩阵 A 和定理 5，固定模式串的最后一个模式符 y_m 与当前时钟周期处理正文 T_l 的匹配结果表示为

$$\begin{array}{l} C_i^j = \left\{ \begin{array}{l} 1 \ if \ t_j \in y_i \\ 0 \ if \ t_j \notin y_i \end{array} \right. \\ for \ \ 0 \le i < m, \ 0 \le j < l \end{array} \tag{83}$$

由等式（82）、（83）和定理 6，有

$$\begin{array}{l} R_m^j = E_m^j \ \& \ C_m^j = \left\{ \begin{array}{l} 1 \ if \ \forall k, (0 \le k \le m), t_{j-k} \in y_{m-k} \\ 0 \ if \ \exists k, (0 \le k \le m), t_{j-k} \notin y_{m-k} \end{array} \right. \\ for \ 0 \le i < m, 0 \le j < l. \end{array} \tag{84}$$

于是，算法结果 R_m 就是固定模式串与正文串 T_n 的匹配结果。

由以上对定义、定理和证明过程的改写，有以下结论成立：

1、“向量与”算法其它部分也和使能向量 E_i 一样，只需要将条件进行修改，取值和结构不会发生改变。因此证明过程也同样成立。

2、“扩展解码矩阵”方案和“向量与”算法的功能是等价的，并且两者的 E_i 、R_i 和 RE_i 的取值也是相同的。

5.1.2 “扩展解码矩阵”和“向量与”算法比较

由5.1.1的方案构造与证明可见，

“扩展解码矩阵”方案与“向量与”算法相比，优点有：

1、通过扩展解码矩阵，在解码矩阵中增加相应的列，就可以完成Snort-PCRE语言中的“., [abc], [a-fA-F0-9], [^abc], \d, \w, \s,\D, \W, \S, \N”单个模式对多个正文的匹配要求。

2、资源占用上，在单个模式对可以匹配多个正文字符的条件下，占用更少的资源，相应增加了可用模式的总量。

3、处理速度上相比，“扩展解码矩阵”方案不低于“向量与”算法。

4、“扩展解码矩阵”方案把所有单字符间并、交、补的影响约束在扩展解码矩阵中，对后续运算不再产生影响，简化了后续运算。

缺点有：

1、资源占用上，实际使用中，如果模式串中无类似的匹配，则资源占用会因解码矩阵的扩展而浪费资源。

2、由于扩展了解码矩阵，本质上是扩大了模式符的数量。导致模式符集比正文字符集大。增加了一定的复杂性。

5.2 整体改进

5.2.1 “矩阵与”算法

为了减少算法对FPGA资源的占用，及提高运算速度。针对“向量与”算法中大量使用了向量与运算，可以改进为矩阵与运算，即把“向量与”算法中大量使用的两输入与门，合并为多输入与门。一方面，因为现代FPGA中使用四输入查找表、六输入查找表、甚至是八输入查找表来实现与、或门，所以相对于使用两输入与、或门而言，使用多输入的与、或门能更好地利用FPGA的特性，从而减少FPGA资源使用；另一方面，因为相对于“向量与”算法使用多级两输入与门实现算法与的连接运算而言，改进为矩阵与运算使用一级多输入与门完成相同的连接运算，可以改进算法整体的时延特性。

为方便说明“矩阵与”算法[63]的实现，本文模式串取值为Snort（snortrules-snapshot-2.8）规则库中dos.rules中所用一条正则表达式串

pcre:"/ftp\x3A\x2F\x2F[\w\x2E\x2F]+[^\x2F]\x3Btype=D/i"。

本算法使用“扩展解码矩阵”方案与“向量与”算法具有相同的解码矩阵，这个解码矩阵可以根据实际要求选择“扩展解码矩阵”方案与“向量与”算法中的一个，相应的算法中使用的模式符也要做相应变化。由上节的证明可知，两者是等价的，因此下面说明过程中不再区分这两种解码矩阵，也不区分这两种模式符，统一称为解码矩阵和模式符，如果影响实现细节将会作出说明。

本文举例的这个正则表达式可分为三大部分：1、不包含通配符的且为正则表达式开

始的固定串“ftp\x3A\x2F\x2F”；2、包含通配符的通配串[\w\x2E\x2F]+；3、不包含通配符的且不在正则表达式开始的固定串[^\x2F]\x3Btype=D。对于三者的匹配有着不同的要求。位于正则表达式开始的固定串，只要有主串输入就需要进行匹配，且在完全匹配时，输出匹配成功为后级提供使能信号，供后级使用。对于包含通配符的通配串，则在前级提供使能信号及匹配成功时，输出匹配成功信号，供后级使用。对于第三种不包含通配符的且不在正则表达式开始位置的固定串，也类似地在前级提供匹配成功且本级完全匹配的情况下才输出匹配成功，再供后级使用。要注意的是：由于必须在单时钟周期内对多个字符并行识别，也就意味着在一个时钟周期内需要对正则表达式有多次并行匹配，因此每级的输出是否成功结果有多个。下面就对三种不同的部分进行详细说明。

一、不包含通配符的且为正则表达式开始的固定串，本例中的模式串为“ftp\x3A\x2F\x2F”，其中包含模式六个字符。由于处于正则表达式的开始因此不需要考虑前级状态，对于每个输入的字符都认为是有效的。另外由于每个时钟周期需要处理多个字符，还需要考虑上一时钟周期的部分匹配情况，与本时钟周期的数据共同完成完整匹配，因此设计的方案中每个时钟周期不但要确定本节拍是否完成匹配，还要生成本时钟周期中部分匹配的状态，传输给下一时钟周期。对于本例的情况，每时钟周期会生成一个5位的部分匹配结果第j位代表是否完成本位所代表的字符之前的所有字符都正确匹配，如果匹配则为“1”，否则为“0”。更进一步本方案的细节设计为：首先根据并行输入的32字节（256位）数据和待匹配的模式串（本例中的模式串长度为6）生成一个5x6位的下三角矩阵、27x6位矩阵和5x5位的上三角矩阵，由这三个矩阵组成（如图19所示）的识别矩阵。

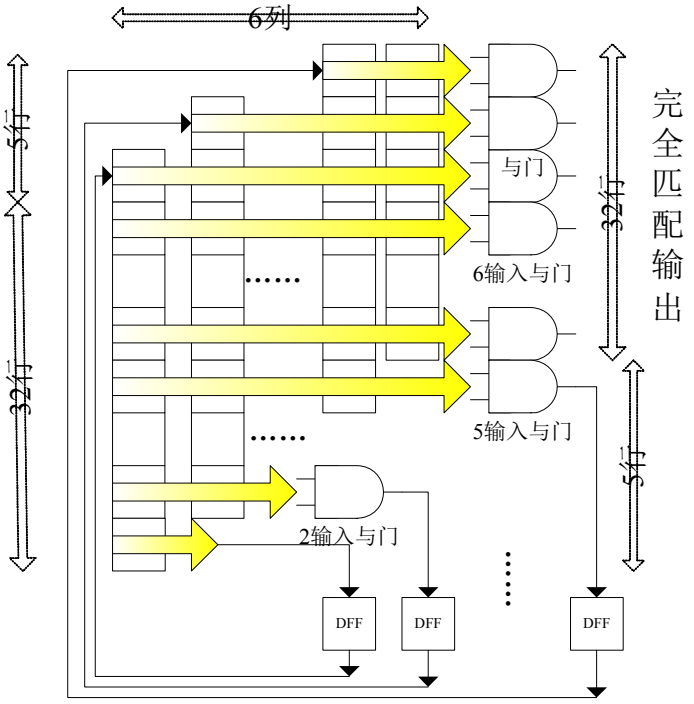

图 19 “矩阵与”识别矩阵示意图

其中每一列有33位。第一列的33比特取值，分别：1、下面32位的取值是根据模式符，从解码矩阵中选择出一列。本例中，为选择“f”所对应的列。2、最上一位的取值为本列中，最下一位经过一个D触发器（DFF）后的结果，代表上一时钟周期最后一个正文字符是否是“f”。第二列的同样有33比特，1、下面32位的取值同样是根据模式符，从解码矩阵中选择出一列。本例中，为选择“t”所对应的列；2、最上一位的取值为本列中最下一位与第一列的倒数第二位与后，经过一个D触发器后的结果，代表上一时钟周期最后两个正文字符是否是“ft”。依此类推，生成前5列的所有位。最后一列只有32位，这32位的取值同样是根据模式符，从解码矩阵中选择出一列。本例中，为选择“\x2F”所对应的列；

在上述矩阵中，三个矩阵的功能分别为：5x5位的上三角矩阵用于确定本时钟周期完成的部分匹配，经过D触发器后，供给下一时钟周期完成全部匹配；27 x 6位的矩阵用于确定本节拍是否完成全部匹配；5x6位的下三角矩阵用于结合上一时钟周期的结果和本时钟周期的数据判断是否完成全部匹配。在匹配过程中，如果是5x5位的上三角矩阵中某一行全为“1”，就表示完成该行表示的部分匹配。如果5x6位的下三角矩阵或24x6位的矩阵某一行全为“1”则表示输入的正文串中出现一次与模式串“ftp\x3A\x2F\x2F”完全匹配的字符串。

二、包含通配符的通配串。结构与一、中的类似，不同之处在于：1、因为是后级，所以需要接受上级输出的匹配结果； 2、因为本级是通配级，所以本级的输出的完全匹配结果也会影响本级的的输出。也就是要求将上级的32位匹配输出与本级匹配结果进行按位或的结果与本级识别矩阵的第一列中下面32位按位与。本例中本级要匹配的为“[\w\x2E\x2F]+”。

由于本例中模式符为“[\w\x2E\x2F]”，因此导致选择“扩展解码矩阵”方案与“向量与”算法时会导致完全不同的结构。选择“向量与”算法时，解码矩阵为32*256位，但会生成62（由\w“匹配字母和数字的字符”导致）个加2个（由“\x2E\x2F”导致）共64个识别矩阵，且需要将64个32位结果按位或，才能生成本模式符所要的结果。选择“扩展解码矩阵”方案时，解码矩阵会因模式符为“[\w\x2E\x2F]”变为32*257位，但只需要一个识别矩阵就可完成对模式符为“[\w\x2E\x2F]”匹配。

三、不包含通配符的且不在正则表达式开始的固定串。结构也与一、的结构类似，不同之处仅在于只需接受上级输入的完全匹配结果，换句话说，只有在前级完全匹配的条件下本级匹配才表示本级的匹配是有效的。也就是要求将上级的32位匹配输出与本级识别矩阵的第一列中下面32位按位与。本例中本级要匹配的为“\x3Btype=D”。

本例完整正则表达式匹配总体示意图，如图20所示

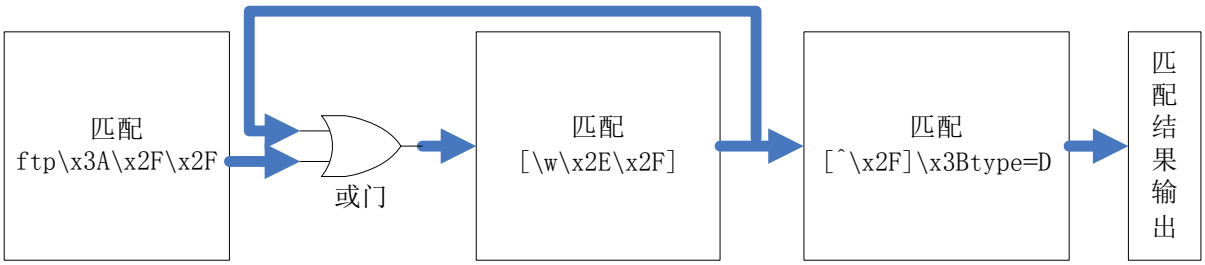

图 20 “矩阵与”总体方案示意图

5.2.2 “矩阵与”算法和“向量与”算法比较

“矩阵与”算法[51]与“向量与”算法相比，优点有：

1、资源占用上，由于两种算法思想上是一致的，差别：在于“矩阵与”算法与“向量与”算法相比去掉了过程中生成的 E_i 、 R_i 和 RE_i ，并将大量的二输入与门变为多输入与门，因此有利于工具对具体实现的资源优化，因此可以与占用更少的资源，相应增加了可用模式的总量。

2、处理速度上相比，“矩阵与”算法有利于工具的优化，因此有可能可以使用更高的时钟频率，因此“矩阵与”算法不低于“向量与”算法。

3、时延特性上相比，相对于“向量与”算法使用多级两输入与门实现算法与的连接运算而言，“矩阵与”运算使用一级多输入与门完成相同的连接运算，可以改进算法整体的时延特性。

缺点有：

灵活性。由于“矩阵与”算法中多输入与门的存在使得算法完成后，识别矩阵不能发生变化，导致正则表达式模式符的关系是固定的，因此不具备可变性。“向量与”算法中使用的都是基本逻辑单元，并且这些基本逻辑单元所组成的模式符识别单元也是相同的，因此，可以通过改变模式符识别单元的连线关系就可组成对不同正则表达式，也就是说，可以通过引入多路选择器等简单的逻辑单元用于改变连线关系，就可完成对芯片重构过程，使得芯片可以支持在线可重构。

5.2.3 补运算改进

通过第四章的构造与证明过程可以看出：虽然“向量与”算法实现了正则表达式中的补运算，但由于补运算的特性为从模式串的第一个字符的使能向量 $E_0 = \left[\overbrace{1,1,\cdots 1}^{l}\right]^T$ 开始，直到算法结果输出 $\overline{R_{m1-1}}$ 为止，必须保证补运算的完整性。导致了1、连接操作的不一致性；2、算法复杂度上升；3、使用了大量资源来完成相应的操作。下面提出一种改进方案解决该问题。

通过前面的“扩展解码矩阵”方案，可以将所有补运算影响限制在扩展解码矩阵中，

对后续的模块不产生影响。

首先改写4.1.1中D的构造过程与证明：

补。设$M1$是一台本算法已实现的非确定型有穷自动机。那么补语言$L(M)=\overline{L(M1)}$。令$M1$的模式串为$P_{m1}=p_0p_1\cdots p_{m1-1}$，则$M$的模式串为$P_m=\overline{p_0p_1\cdots p_{m1-1}}=\overline{p_0}\circ\overline{p_1}\circ\cdots\circ\overline{p_{m-1}}$。

构造方法：

1、使用“扩展解码矩阵”方案，将所有字符的补运算完成在扩展解码矩阵中。在模式符集中增加相应模式符，令$y_i=\overline{p_i}$,其中$0\le i<m$。

2、针对模式串中第一个字符（y_0）匹配的构造方法：构建字符比较模块，令$E_0=\left[\overbrace{1,1,\cdots 1}^{l}\right]^T$；根据$y_0$从扩展解码矩阵中取得模式比较结果$C_0$，即$C_0=\left[A_{j,k}\right]$,其中$0\le j<l,\mathrm{y_k}=y_0$；使用字符比较模块将$E_0$和$C_0$映射为$R_0$；最后使用结果转换函数将$R_0$映射为$RE_0$。

3、针对模式串中除第一个字符与最后一个字符外的所有字符（y_i其中$0<i<m-1$）匹配的构造方法：构建字符比较模块，令$E_i=RE_{i-1}$；同样根据y_i从扩展解码矩阵中取得模式比较结果C_i，即$C_i=\left[A_{j,k}\right]$,其中$0<\mathrm{i}<m-1,0\le j<l,\mathrm{y_k}=y_i$；使用字符比较模块将$E_i$和$C_i$映射为$R_i$；最后使用结果转换函数将$R_i$映射为$RE_i$。

4、针对模式串中最后一个字符匹配（y_{m-1}）的构造方法：构建字符比较模块，令$E_{m-1}=RE_{m-2}$，同样根据y_{m-1}从扩展解码矩阵中取得模式比较结果C_{m-1}；使用字符比较模块将E_{m-1}和C_{m-1}映射为R_{m-1}；

5、最后一级生成的R_{m-1}就是固定模式串与正文串$\mathrm{T_n}$的匹配结果。

$$R_{m-1}^{j}=\begin{cases}1\ if\ \forall k,(0\le k\le m-1),t_{j-k}\in y_{m-1-k}\\0\ if\ \exists k,(0\le k\le m-1),t_{j-k}\notin y_{m-1-k}\end{cases}\quad for\ 0\le j<l. \tag{85}$$

证：通过算法构造过程可以看出：我们可以通过使用固定模式串的长度m，使用数学归纳法来证明“向量与”算法可以完成任意长度的固定模式串匹配。

1、基本步骤。设固定模式串的长度$|P|$为1，即，$|P_m|$=1，则$P_m=y_0$。根据每时钟周期处理正文$\mathrm{T_1}$，同时时钟周期内可生成扩展解码矩阵模块A。根据定理5，有

$$C_0^{j}=\begin{cases}1\ if\ t_j\in y_0\\0\ if\ t_j\notin y_0\end{cases}\quad for\ 0\le j<l \tag{86}$$

另外根据构造知，再由定理6知

$$\left.\begin{array}{l} R_0^j = E_0^j \,\&\, C_0^j \\ E_0 = \left[\overbrace{1,1,\cdots 1}^{l}\right]^T \Rightarrow E_0^j = 1 \end{array}\right\} \Rightarrow R_0^j = C_0^j = \begin{cases} 1 \ if \ t_j \in y_0 \\ 0 \ if \ t_j \notin y_0 \end{cases} \tag{87}$$

$$for \ \ 0 \le j < l$$

于是，算法的结果（R_0）的第个元素确切表示固定模式串与正文串 T_j 的匹配结果。

2、归纳假设。设 m≥1，“向量与”算法能够完成所有长度小于等于 m 的固定模式串匹配。即：

$$R_{m-1}^j = \begin{cases} 1 \, if \ \forall k, t_{j-k} \in y_{m-1-k} \\ 0 \, if \ \exists k, t_{j-k} \notin y_{m-1-k} \end{cases} \tag{88}$$

$$for \ 0 \le j < l, 0 \le k \le m-1.$$

3、归纳步骤。设固定模式串的长度$|P|$为 m+1，则 $P_m = p_0 p_1 \cdots p_m$。根据归纳假设，R_{m-1} 中记录了对于 $p_0 p_1 \cdots p_{m-1}$ 的匹配结果，

$$R_{m-1}^j = \begin{cases} 1 \, if \ \forall k, t_{j-k} \in y_{m-1-k} \\ 0 \, if \ \exists k, t_{j-k} \notin y_{m-1-k} \end{cases} \tag{89}$$

$$for \ 0 \le j < l, 0 \le k \le m-1.$$

根据构造和结果使能函数定理 3 知，

$$E_m^j = RE_{m-1}^j = R_{m-1}^{j-1} = \begin{cases} 1 \, if \ \forall k, t_{j-k-1} \in y_{m-1-k} \\ 0 \, if \ \exists k, t_{j-k-1} \notin y_{m-1-k} \end{cases} \tag{90}$$

$$for \ 0 \le j < l, 0 \le k \le m-1.$$

根据扩展解码矩阵 A 和定理 5，固定模式串的最后一个字符 p_m 与当前时钟周期处理正文 T_l 的匹配结果表示为

$$C_m^j = \begin{cases} 1 \ if \ t_j \in y_m \\ 0 \ if \ t_j \notin y_m \end{cases} \tag{91}$$

$$for \ 0 \le j < l$$

由等式（90）、（91）和定理 6，得到

$$R_m^j = E_m^j \,\&\, C_m^j = \begin{cases} 1 \, if \ \forall k, t_{j-k} \in y_{m-k} \\ 0 \, if \ \exists k, t_{j-k} \notin y_{m-k} \end{cases} \tag{92}$$

$$for \ 0 \le j < l, 0 \le k \le m-1.$$

于是，算法结果 R_m 就是固定模式串与正文串 T_n 的匹配结果

综上所述，“补操作改进”方案能够完成对任意长度的模式符补运算与正文串T_n的匹配，匹配结果存放于与每个固定模式串相对应的最后一级 R_{m-1} 中。

其次改写4.1.1-B-d的构造与证明

设 $L(M1)$ 不是补，$L(M2)$ 是补

令 $M1$ 的模式串为 $P_{m1}=p_0p_1\cdots p_{m1-1}$，$M3$ 的模式串为 $P_{m2}=p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}$，$L(M2)=\overline{L(M3)}$，则，$L(M)=L(M1)\circ L(M2)\quad=L(M1)\circ\overline{L(M3)}$，$M$ 的模式串为 $P_m=(p_0p_1\cdots p_{m1-1})\circ\overline{p_0^{'}p_1^{'}\cdots p_{m2-1}^{'}}=(p_0p_1\cdots p_{m1-1})\circ\overline{p_0^{'}}\circ\overline{p_1^{'}}\circ\cdots\overline{p_{m2-1}^{'}}$

通过使用“扩展解码矩阵”方案中方法扩展解码矩阵，扩大了模式符集，仅针对加入每个字符的补，由原来包含的256个字符的字符集扩展为包含256个字符和256个字符的补的模式集，也就使得 $P_m=y_0y_1\cdots y_{m1-1}y_0^{'}y_1^{'}\cdots y_{m2-1}^{'}$。再使用上面的构造与证明方法，就可得到完成本例的构造和证明。

至此，就完成了对“向量与”算法补操作的改进。

通过以上的改进，“向量与”算法可以有以下结论：

1、不再区分字符与字符的补。由于引入扩展解码矩阵，无论是字符还是字符的补运算都直接在扩展解码矩阵中完成，所以后续运算中不再区分字符和字符的补。

2、连接运算中不再考虑补的影响。“向量与”算法中考虑到引用关系需要先处理补运算，再处理连接运算，因此在处理连接运算时，需要考虑补运算的影响。而补运算改进算法，连接运算位于扩展解码矩阵之后，字符补运算的影响被约束在扩展解码矩阵中，因此连接运算不再需要考虑补运算的影响。

3、最终补运算不再是一种特殊的运算。

5.2.4 补运算改进与“向量与”算法的比较

补运算改进与“向量与”算法相比，优点有：

1、很好地解决了补运算难题，通过扩展解码矩阵，将补运算的影响限制在解码矩阵中，其它部分的运算中不再区分模式符是否是补运算。

2、资源占用上。对于包含大量规则集实现，如SNORT等，其中必然会包含大量字符补运算和字符串的补运算。“向量与”算法会因为补运算的存在，使用大量的非门以及“基础模块3”，因此会占用大量的逻辑单元和产生大量的连线。

3、处理速度上相比。补运算改进与“向量与”算法相比，不再使用非门，因此补运算改进不低于“向量与”算法。

缺点有：

1、资源占用上。由于补运算改进需要使用一个比“向量与”算法大的扩展解码矩阵。如果考虑算法灵活性就需要做出能适应尽可能多情况的扩展解码矩阵，这样会使扩展解码矩阵过大；如果不考虑灵活性，需要构造适合具体应用扩展解码矩阵，则会导致扩展解码矩阵经常变化。

2、同样也因为引入了扩展了解码矩阵，本质上是扩大了模式符的数量。导致模式符集比正文字符集大。增加了一定的复杂性。

5.3 改进后算法示例

在4.2.3节中通过两个例子演示了“向量与”算法构造识别具体正则表达式的自动机。本节同样用改进后的算法来构造识别相同正则表达式自动机。同样假定每时钟周期输入32个字符。

例 1：规则取自 snortrules-snapshot-2.8/rules/sql.rules 中一条：

alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 7210
(msg:"SQL SAP MaxDB shell command injection attempt";
flow:established,to_server;
content:"exec_sdbinfo"; nocase;
pcre:"/exec_sdbinfo\s+[\x26\x3b\x7c\x3e\x3c]/i";
metadata:policy balanced-ips drop, policy security-ips drop;
reference:bugtraq,27206; reference:cve,2008-0244; classtype:attempted-admin;
sid:13356; rev:1;)

其中正则表达式模式串为“exec_sdbinfo\s+[\x26\x3b\x7c\x3e\x3c]”。

1、首先构造公用的扩展解码矩阵，完成每时钟周期处理正文T_{32}的解码任务。输入为32个正文字符，由于本例中有\s和[\x26\x3b\x7c\x3e\x3c]两种模式符，这两种模式符都是分别为5个字符的或。针对这种情况需要构造出输出为32x258的解码矩阵。其中257列是为了识别\s，只要同行中第12、10、13、9、11列有1，则该行的第257行中相同列为1，只在同行中第12、10、13、9、11列全为0情况下，该行的第257行中相同列为0；258列是为了识别[\x26\x3b\x7c\x3e\x3c]，该列中的每个值取值为只要同行中第38、59、124、62、60列有1，则该行的第258行中相同列为1，只在同行中第38、59、124、62、60列全为0情况下，该行的第258行中相同列为0。

2、针对模式串中第一个字符（ p_0=e）匹配的构造方法：构建字符比较模块，令 $E_0 = \left[\overbrace{1,1,\cdots 1}^{l}\right]^T$ ；因为字符e的编码为101，所以根据e取解码矩阵中的第101列，该列就是输入的32个字符与模式符e的比较结果 C_0，即 $C_0 = \left[A_{j,k}\right]$，其中 $0 \le j < 32,$，$k = 101$；使用字符比较模块将 E_0 和 C_0 映射为 R_0；最后使用结果转换函数将 R_0 映射为 RE_0。

3、针对模式串中从第一个字符e到\s之间的所有字符（ p_i 其中$0 < i < 12$ ）匹配的构造方法：构建字符比较模块，令 $E_i = RE_{i-1}$；同样根据 p_i 从解码矩阵中取得模式比较结果 C_i，即 $C_i = \left[A_{j,k}\right]$，其中 $0 < \mathrm{i} < 12, 0 \le j < 32, k = p_i$；使用字符比较模块将 E_i 和 C_i 映射为 R_i；最后使用结果转换函数将 R_i 映射为 RE_i。这一串的构造最终会生成 RE_{11}。

4、针对模式串中第12个模式符“\s”和算子“+”。这个组合要求匹配至少一个空白符，

等价于至少匹配[\f\n\r\t\v]其中的一个，又因为扩展解码矩阵中第257列已经处理了“\s”，所以这里需要取第257列。这里需要构造一个匹配模块，这些字符与输入的32个字符比较结果C_{12}相对应于扩展解码矩阵中的第257列。该模块的E_{12}是上一个字符o的输出RE_{11}与本模式符\s的RE_{12}经按位或后的结果，使用字符比较模块把E_{12}和C_{12}生成R_{12}后，最后使用结果转换函数将\s的R_{12}映射为\s的RE_{12}；

5、针对模式串中最后一个模式符[\x26\x3b\x7c\x3e\x3c]。这个模式符与4中类似。这里仍然只需要构造一个匹配模块，这些字符与输入的32个字符比较结果C_{13}对应于扩展解码矩阵中的第258列。该模块的E_{13}都是上一个字符\s的输出RE_{12}，使用字符比较模块把E_{13}和C_{13}生成R_{13}。

6、最后一级生成的R_{13}就是正则表达式串exec_sdbinfo\s+[\x26\x3b\x7c\x3e\x3c]与正文串T_n的匹配结果。

例 2：规则取自 snortrules-snapshot-2.8/rules/sql.rules 中一条：

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"SQL generic sql update injection attempt";
flow:established,to_server;
content:"update"; nocase;
pcre:"/[^A-Z\x2B]update[^\n]*set/i";
metadata:policy security-ips drop, service http;
reference:url,www.securiteam.com/securityreviews/5DP0N1P76E.html;
classtype:web-application-attack; sid:13514; rev:5;)，其中正则表达式模式串为“[^A-Z\x2B]update[^\n]*set”。

1、首先构造公用的解码矩阵，完成每时钟周期处理正文T_{32}的解码任务。输入为32个正文字符，由于本例中有[^A-Z\x2B]和[^\n]两种模式符，这两种模式符都是分别为模式符的非。针对这种情况需要构造出输出为32x258的解码矩阵。其中257列是为了识别[^A-Z\x2B]，只要同行中第43列、65列到90列只有1，则该行的第257行中相同列为0，只在同行中第43列、65列到90列全为0情况下，该行的第257行中相同列为1；258列是为了识别[^\n]，该列中的每个值取值为只要同行中第10列为1，则该行的第258行中相同列为0，只在同行中第10列为0情况下，该行的第258行中相同列为1。

2、针对模式串中第一个模式符[^A-Z\x2B]匹配的构造方法：本模式符需要通配除A-Z和\x2B外的所有字符。又因为扩展解码矩阵中第257列已经处理了，因此，需要构建一个字符比较模块。该模块$E_0=\left[\overbrace{1,1,\cdots 1}^{l}\right]^T$；这些字符与输入的32个字符比较结果$C_0$对应于解码矩阵中的第257列。使用字符比较模块将$E_0$和$C_0$映射为$R_0$；最后

使用结果转换函数将 R_0 映射为 RE_0 。

3、针对模式串中从第一个模式符[^A-Z\x2B]到[^\n]之间的所有字符“update”（ p_i 其中$0 < i < 7$ ）的构造方法：构建字符比较模块，令 $E_i = RE_{i-1}$；同样根据 p_i 从解码矩阵中取得模式比较结果 C_i ，即 $C_i = [A_{j,k}]$, 其中 $0 < \mathrm{i} < 7, 0 \le j < 32, k = p_i$ ；使用字符比较模块将 E_i 和 C_i 映射为 R_i ；最后使用结果转换函数将 R_i 映射为 RE_i 。这一串的构造最终会生成 RE_6 。

4、针对模式串中第12个模式符“[^\n]”和算子“*”。这个组合要求匹配零个或多个非换行符，同样因为扩展解码矩阵中第258列已经处理“[^\n]”，所以这里需要构造一个匹配模块。参考4.2.1-C中的方法，因此这里令 $E_7 = RE_6 \mid RE_7$ ，这个模式符与输入的32个字符比较结果 C_7 对应于解码矩阵中的第258列。使用字符比较模块把 E_7 和 C_7 生成 R_7 ； 最后使用结果转换函数将 R_7 映射为 RE_7 ；

5、针对模式串中模式符“se”的构造方法： p_i 其中$7 < i < 10$ 构建字符比较模块。同时因为前一级中包含算子“*”，所以令 $E_8 = RE_7 \mid RE_6$ ，同时 $E_9 = RE_8$；同样根据 p_i 从解码矩阵中取得模式比较结果 C_i ，即 $C_i = [A_{j,k}]$, 其中 $7 < \mathrm{i} < 10, 0 \le j < l, k = p_i$ ；使用字符比较模块将 E_i 和 C_i 映射为 R_i ；最后使用结果转换函数将 R_i 映射为 RE_i 。这一串的构造最终会生成 RE_9 。

6、针对模式串中最后一个模式符“t” 的构造方法：令 $E_{10} = RE_9$ ； 同样根据t从解码矩阵中取得模式比较结果 C_{10} 对应于解码矩阵中的第116列，即 $C_{10} = [A_{j,k}]$ ，其中 $0 \le j < 32, k = 116$ ；使用字符比较模块将 E_{10} 和 C_{10} 映射为 R_{10} 。

7、最后一级生成的 R_{10} 就是正则表达式串[^A-Z\x2B]update[^\n]*set与正文串 $\mathrm{T_n}$ 的匹配结果。

5.4 小结

通过本章对“向量与”算法的研究，提出了三种改进方案，其中第一种改进方案通过扩展解码矩阵，本质上扩充了模式符集合，与解码矩阵相比增加了复杂度，但为后续的处理模式符时减少资源消耗打下了基础。第二种改进方案在一定程度上降低了资源的消耗，改善了算法的时延特性，但降低了系统的灵活性。最后一种改进方案是针对“向量与”算法中对补运算符的支持，通过引入第一种方案的扩展解码矩阵，去除了补运算的特殊性，本算法中把正则表达式中补运算实现的难题，转化为正则表达式中常用的连接运算。

5.3节中给出了与4.2.3节中相同的正则表达式的实现方案，通过比较两种实现，可以得到它们之间的差别如下：

1、解码矩阵与扩展解码矩阵。4.2.3中使用的是解码矩阵，其行列数是固定的32x256；

扩展解码矩阵随不同的模式符集有不同的列数，或者随不同的模式符集有更大的列数。增加了资源的占用量。但因为所有的模式符都使用同一个扩展解码矩阵，因此，这里的资源增加会平均到所有模式符中，对总容量影响不大。

2、对于模式符为集合的运算如[\x26\x3b\x7c\x3e\x3c]、\s的实现在4.2.3中需要都构造五个单个模式符运算及把五个结果按位或来支持以上两种运算；在5.3中对这两种运算不仅只需要构造一个模式符就可完成运算，而且不用按位或运算，因此，不仅可以减少资源占用，而且还可减少运算量，还可减少算法的整个时延特性。

3、对于补运算如[^A-Z\x2B]、[^\n]的支持，4.2.3中不仅分别构造了27个、1个模式符运算，而且还使用了27输入的按位或非和非门；5.3中分别只构造了1个模式符运算，并不需要其它运算。因此，不仅可以减少资源占用，而且还可减少运算量。

4、扩展解码矩阵完全解决了正则表达式中单模式符之间的并、交、非运算。把单模式符之间的并、交、非运算约束在扩展解码矩阵中。后续运算只将结果作为单个模式符处理即可。

第六章 性能分析与实验仿真结果

前面的章节构造与证明了本文提出算法的正确性。本章将分别给出算法的吞吐率和资源占用。首先给出理论上的吞吐率和资源占用情况，随后给出在现有 FPGA 上所能达到的实际吞吐率和资源占用。

6.1“向量与”算法性能分析

6.1.1 “向量与”算法的计算占用

要计算“向量与”算法的时钟周期，需要了解“向量与”算法所用到的所有操作。通过“向量与”算法的构造，下面列出“向量与”算法用到的所有与时间相关的操作：

1、计算每个 R_i^j。对每个模式串的匹配都会用到一个与门，用于 $R_i^j = E_i^j \ \& \ C_i^j$。

2、计算每个 RE_i^j。在计算正则表达式中交的情况下，会用到“与”门；在计算正则表达式中并的情况下，会用到“或”门；在计算正则表达式中补的情况下，会用到“非”门；其他情况下只需要连线连接 R_i^{j-1} 和一个D触发器的输出。

$$RE_i^j = \begin{cases} R_i^{j-1} \ if \ 0 < j < l \\ R_i^{l-1} \ if \ j = 0 \end{cases}$$

所有RE_i^k产生于同一时钟周期，RE_i^0比RE_i^k晚一时钟周期， (93)

$$for 0 \le i < m, 0 \le j < l.$$

另外，每个 RE 都会用一个 D 触发器。

3、计算每个 E_i^j。在计算Kleene星号的情况下，会用到“或”门；其他情况下只需要连线连接 RE_{i-1}^j。

4、每个时钟周期内，计算最多 $\min(l, \max(|MP|s)$ 个 R_i^j。

6.1.2 时钟频率与吞吐率分析

这部分的计算使用以下符号及约定：

- l：每时钟周期接受的正文个数。
- t_{and}：实现时，用到的“与”门时延。
- t_{or}：实现时，用到的“或”门时延。
- t_{not}：实现时，用到的“非”门时延。
- $t_{decoder}$：实现时，解码矩阵用到的解码器的时延。
- $\min(a,b)$：取值为 a，b 的较小值。
- $\max(a,b)$：取值为 a，b 的较大值。
- 不失一般情况，可以令 $t_{and} = t_{or} = t_{not}$。

- i：正则表达式中，并出现的数量。
- j：正则表达式中，交出现的数量。
- k：正则表达式中，Kleene 星号出现的数量。
- h：正则表达式中，补出现的数量。

根据6.1.1中的说明，在不考虑线延时的条件下，可以有以下结果：

在FPGA或ASIC中实现电路的最小时钟周期取决于关键路径上所有时延之和。通过第四章算法构造过程及等价性证明可以看出：从数据输入到匹配结果输出的过程中，所有数据都经过类似的运算单元到达结果输出单元或D触发器的输入，这些路径都是类似的，因此我们考虑其中经过最长的路径到达结果输出单元或D触发器的输入的正文字符，就可以得到最小时钟周期。

每时钟周期处理正文T_l依次经过以下路径：

1、解码器生成解码矩阵。产生$t_{decoder}$的时延。

2、到达所有的C_i中。由于使用的是连线，因此不考虑时延。

3、C_0与准备好的E_0经过与门生成R_0。产生t_{and}的时延。

4、如果这个位置上有并运算符，R_0会与另外一条路径上的$R_i^{'}$相或，产生t_{or}的时延；如果有交运算符，R_0会与另外一条路径上的$R_i^{'}$相与，产生t_{and}的时延；如果这个位置上有Kleene星号运算符的开始，R_0会与另外一条路径上的$R_i^{'}$相或，产生t_{or}的时延；如果这个位置上有非运算符，R_0需要生成$\overline{R_0}$，产生t_{not}的时延；最后没有任何运算符则没有时延要求。

5、i从0开始，每次i加1，直到$|MP|$（完成匹配）或l（全部运算到达各自的D触发器的输入）结束循环，到第10步。

6、由R_i生成RE_i。由于使用的是连线，因此不考虑时延。因为其中一位（第$l-1$位）到达D触发器的输入。

7、由RE_i^j生成E_i。由于使用的是连线，因此不考虑时延。

8、E_i与准备好的C_i经过与门生成R_i。产生t_{and}的时延

9、如果这个位置上有并运算符，R_i会与另外一条路径上的$R_i^{'}$相或，产生t_{or}的时延；如果有交运算符，R_i会与另外一条路径上的$R_i^{'}$相与，产生t_{and}的时延；如果这个位置上有Kleene星号运算符的开始，R_i会与另外一条路径上的$R_i^{'}$相或，产生t_{or}的时延；如果这个位置上有非运算符，R_i需要生成$\overline{R_i}$，产生t_{not}的时延；最后没有任何运算符则没有时延要求。

10、 匹配路径完成

由以上路径的描述可知：匹配每个正则表达式的时钟周期最小值为：

$$clock\ cycle = t_{decoder} + \min(l, |MP|) * t_{and} + i * t_{or} + j * t_{and} + k * t_{or} + h * t_{not}$$

$$= t_{decoder} + \min(l, |MP|) * t_{and} + i * t_{and} + j * t_{and} + k * t_{and} + h * t_{and} \quad (94)$$

$$= t_{decoder} + (\min(l, |MP|) + i + j + k + h) * t_{and}$$

因此最大时钟频率为

$$1/(t_{decoder} + (\min(l, |MP|) + i + j + k + h) * t_{and})\,\text{Hz} \quad (95)$$

由于，我们假定字符表中所有字符用8位来表示，即$|\Sigma|$=256，因此每时钟周期计算的正文位数为$8*l$位。另外，我们知道系统的吞吐率就等于系统时钟频率乘以系统每个时钟处理数据位数，因此得到最大吞吐率为

$$8 * l /(t_{decoder} + (\min(l, |MP|) + i + j + k + h) * t_{and})\,\text{bps} \quad (96)$$

多个正则表达式同时匹配的条件下，系统时钟周期的最小值为所有单个时钟周期的最大值，记为$cc_{\max}$。相应地最大时钟频率为$1/cc_{\max}$ Hz，系统最大吞吐率为$8*l/cc_{\max}$ bps。

6.1.3 资源占用分析

假设所有模式串的数量为n。根据4.2.1中的算法构造，可知“向量与”算法必要的资源为：

- 解码矩阵 A。需要l个 8 位到 256 位的解码器。对于扩展解码矩阵，会需要l个 8 位到比 256 位更多位的解码器。
- R_i^j的计算。$R_i^j = E_i^j \,\&\, C_i^j$的数量为$(\sum_i^n |P_m^i|) * l$，即, 所有模式串中字符的总数乘以每时钟周期处理正文数。资源是二输入与门。
- 并的计算。$(\sum_i^n |\text{第i个模式串中并}|) * l$，即, 所有模式串中并操作的总数乘以每时钟周期处理正文数。资源是二输入或门。
- 交的计算。$(\sum_i^n |\text{第i个模式串中交}|) * l$，即, 所有模式串中交操作的总数乘以每时钟周期处理正文数。资源是二输入与门。
- 补的计算。$(\sum_i^n |\text{第i个模式串中补}|) * l$，即, 所有模式串中补操作的总数乘以每时钟周期处理正文数。资源是一输入非门。
- Kleene 星号的计算。$(\sum_i^n |\text{第i个模式串中}Kleene\text{星号}|) * l$，即, 所有模式串中 Kleene 星号操作的总数乘以每时钟周期处理正文数。资源是二输入或门。
- RE_i 的计算。数量为$(\sum_i^n |P_m^i|)$，即, 所有模式串中字符的总数。资源是一位的 D 触发器。
- 其他是大量的连线。

以上是完成“向量与”算法必要的资源。

通过对算法资源占用的分析可以看出“向量与”算法仅使用了逻辑资源，没有使用存储资源。

6.2 实验、仿真结果

6.2.1 环境简介

本文的全部实验的实验环境为：

1、软件使用Altera公司[33]的Quartus II 11.0 sp1。

2、硬件分别使用Altera公司高性能的Stratix II系列中EP2S180芯片和Altera公司低价格Cyclone IV E系列中EP4CE115芯片来验证算法正确性、性能和资源占用。

3、正则表达式模式串来自于SNORT网站提供的规则集snortrules-snapshot-2.8.tar.gz。

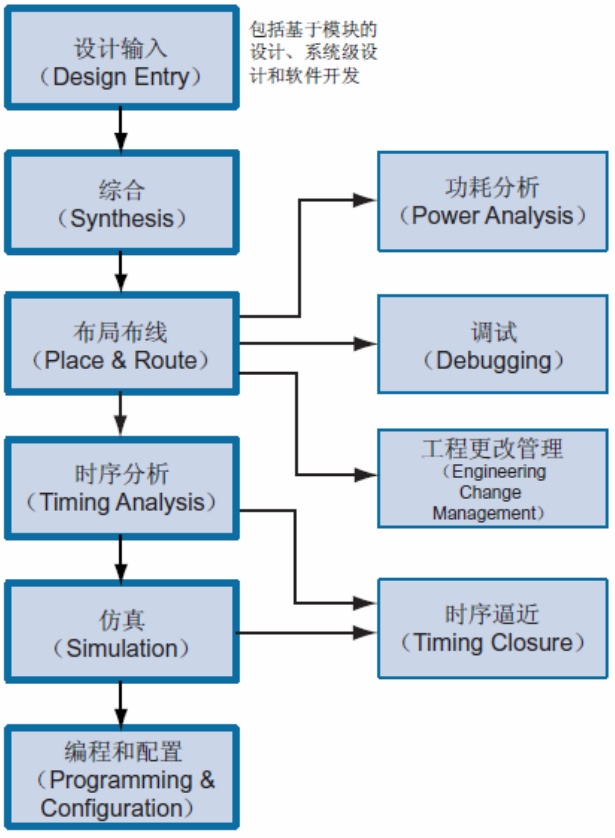

图 21 Quartus II 设计流程

目前世界上有十几家FPGA和FPLD供应公司，Altera和Xilinx两家[64]公司占了60%以上的市场份额，Altera和Xilinx公司的产品各有千秋，不相上下。但就编译速度来讲，Altera公司的Quartus II明显比Xilinx的ISE要快得多。因此，在实验和仿真阶段，主要使用了Quartus II进行，以加快各种实验、仿真的进行，和得到仿真、实验结果。Quartus II是Altera公司的综合性PLD开发软件，支持原理图、VerilogHDL、VHDL以及Altera公司的AHDL等多种设计输入形式。另外该软件也集成了综合器、仿真器、布局器、布线器和时序分析，从而可

以完成从设计输入到硬件配置的完整PLD设计流程。软件提供完整的多平台设计环境，可以在XP、Win7、Linux以及Unix操作系统上运行。提供了完善的用户图形界面，特别适合新手使用，同时也可以使用Tcl脚本运行程序，适合熟练工程师提高工作效率。Quartus II还能与Matlab/simulink相结合方便各种特殊应用地开发。Quartus II含有FPGA和CPLD设计所有阶段的解决方案。Quartus II设计流程的如图21示。

表 5　Stratix II 系列 FPGA 资源表

资源	EP2S15	EP2S30	EP2S60	EP2S90	EP2S130	EP2S180
ALM	6240	13552	24176	36384	53016	71760
可变查找表	12480	27104	48352	72768	106032	143520
等价逻辑单元	15600	33880	60440	90960	132540	179400
M512 存储块	104	202	329	488	699	930
M4K 存储块	78	144	255	408	609	768
M 存储块	0	1	2	4	6	9
存储总量（位）	419328	1369728	2544192	4520488	6747840	9383040
DSP 块	12	16	36	48	63	96
18 位 x18 位乘法器	48	64	144	192	252	384
增强 PLL	2	2	4	4	4	4
快速 PLL	4	4	8	8	8	8
最大可用 I/O	366	500	718	902	1126	1170

Stratix II系列是Altera公司2004年推出的面向高密度、高性能的FPGA（现场可编程门阵列），采用了90nm工艺，是第二代的Stratix系列。设计者使用Stratix II系列的芯片可以达到类似ASIC的密度、性能，但与ASIC相比有更好生产时间，而且，ASIC无法改变内部逻辑，FPGA可以随时改变内部逻辑。Stratix II系列支持的内部时钟最大频率为500MHz，典型值是250Mhz。表5列出了Stratix II系列的所有器件及其片内可用资源情况。

Stratix II系列中的EP2S180具有71760个自适应逻辑模块（ALM），179400个等价逻辑单元（LE），9383040位RAM，96个DSP模块，也是Stratix II系列里最大的芯片。对我们的实验有很大帮助。另外，参考文献中也有人用该系列芯片来实现他们的算法，也方便比较实验结果。

Altera Cyclone IV FPGA拓展了Cyclone FPGA系列的领先优势，为市场提供成本最低、功耗最低并具有收发器的FPGA。Cyclone IV FPGA系列适合对成本敏感的大批量应用，满足越来越大的带宽需求，同时降低了成本。虽然Cyclone系列FPGA支持的内部时钟最大频率不能达到Stratix II系列所能达到的500MHz，但和Stratix相比，逻辑单元的成本下降很多，对成本敏感的应用更多地会考虑选用Cyclone系列的FPGA。表6列出了Cyclone IV系列的所有器件及其片内可用资源情况。

Cyclone IV E系列中的EP4CE115具有114480个逻辑单元（LE），3888K位RAM，266个

DSP模块，是Cyclone IV E系列最大的芯片。

表 6　Cyclone IV 系列 FPGA 资源表

资源	EP4CE6	EP4CE15	EP4CE30	EP4CE55	EP4CE75	EP4CE115
逻辑单元	6272	15408	28848	55856	75408	114480
存储总量（K 位）	270	504	594	2340	2745	3888
18 位 x18 位乘法器	15	56	66	154	200	266
通用 PLL	2	2	4	4	4	4
全局时钟网络	10	20	20	20	20	20
最大可用 I/O	179	343	535	374	426	528

在1998年,Martin Roesch用C语言开发了开放源代码(Open Source)的入侵检测系统Snort，将其定位成简单轻巧的入侵侦测软件。直至今天，Snort已发展成为一个多平台(Multi-Platform)、实时(Real-Time)流量分析，网络IP数据包(Pocket)记录等特性的强大的网络入侵检测/防护系统(Network Intrusion Detection/Prevention SystemNIDS/NIPS)。通过在网络中收集信息，并分析这些信息，查看网络中是否有违反安全策略的行为和遭到攻击，从而扩展了系统管理员的安全管理能力，从而提高信息安全基础结构的完整性。Snort符合通用公共许可(GPL——GUN General Pubic License)，在网上可以免费下载获得Snort。如今Snort已被广泛使用、影响力远布全世界，成为入侵检测/防护系统的实事标准。

表7列出了SNORT规则集中包含规则数量、固定模式串、固定模式符总量、正则模式串数量、正则模式符总量。

表 7　SNORT 规则集数量

时间	规则数量	固定模式数量	固定模式符数量	正则模式数量	正则模式符数量
2007 年七月 V2.6	8145	2927	63953	1687	86024
2006 年八月 V2.4	7000	2558	52841	1504	69127
2006 年四月 V2.4	4392	1537	24258	509	19580
2005 年三月 V2.3	3107	2188	33618	301	9638
2004 年七月 V2.2	2384	1631	20911	157	2269
2004 年二月 V2.1	2162	942	11199	104	1562
2003 年五月 V1.9	2062	909	10692	65	544

表7显示SNORT IDS规则集增长很快，包含了更多的规则和内容描述。特别2006年后，无论是固定模式串还是正则模式数量都迅速增加。

表8列出了SNORT-PCRE常用的元字符说明

表 8　SNORT-PCRE 常用元字符说明

特征	说明
a	所有 ASCII 字符
.	匹配除新行外的任一字符
[abc]	字符集。匹配方括号内的一个字符
[a-f]	范围字符集。
[^abc]	非字符集。匹配除方括号内字符的任一字符
RegExp*	Kleene 星号。匹配 0 或任意次正则表达式
\xFF	匹配 16 进制 FF 表示的 ASCII 字符
\000	匹配 8 进制 000 表示的 ASCII 字符
\d	匹配 0-9 的数字
\D	匹配除 0-9 的数字外的字符
\w	匹配字母和数字的字符
\W	匹配除字母和数字外的字符
\s	匹配空白字符
\S	匹配除空白外的字符
\n	匹配回车符（LF，NL）
\N	匹配除回车符外的字符
\r	匹配换行符
\t	匹配 tab 符
(RegExp)	组操作。方便应用操作符
RegExp1RegExp2	连接。连接正则表达式 1 和正则表达式 2
RegExp1 | RegExp2	并。正则表达式 1 或正则表达式 2

6.2.2 实验、仿真结果

通过之前的算法完成了“向量与”算法正确性和时间复杂度的证明，并通过分析算法得到了“向量与”算法理论上的最大时钟频率、吞吐率及资源占用情况。下面通过实验得到算法的实际运行时可能的最大时钟频率、最大吞吐率和资源占用。

最大时钟频率是指在实例化算法时，在工具中对算法所能加的时钟约束最大值。

最大吞吐率是指上述最大时钟频率乘以每个时钟周期输入的正文位数。

这里用每模式符占用逻辑单元（LE）为标准，评价算法的资源占用。计算方法为：首先在某一型号的FPGA中实例化算法，其次用布局、布线成功后所占用的逻辑单元数量除以实例化算法所用到的模式符数量，最后得到算法在这一型号FPGA上的每模式符占用逻辑单元。逻辑单元（LE）指的是FPGA的基本块，一个逻辑单元包括一个4输入查找列（LUT），

一个1位寄存器和进位链组成。另外，实现正则表达式时，每个正则表达式内包含的模式符个数不同，正则表达式个数作为计量单位显然不合适，但是，正则表达式由模式符组成，模式符是正则表达式的基本单元，因此，在FPGA中实现正则表达式算法时，使用每模式符占用逻辑单元作为衡量算法占用资源的标准是合适的。因为每个FPGA的逻辑单元数量是一定的，所以在知道算法的每模式符占用逻辑单元后，就可知道这个FPGA所能容纳模式符数量。进一步，在统计每个待实现的正则表达式中所包含的模式数量后，就可知道该FPGA能容纳多少条正则表达式。

实验一是在Altera公司的Stratix II系列中EP2S180上测试“向量与”算法，测试目的：

1、在 Stratix II 系列的 FPGA 上，测试算法的时钟频率能否达到 FPGA 支持的最大时钟频率 500MHz。从而在一定程度验证上 6.1.2 中的结论；

2、测试算法的吞吐率；

3、验证算法的资源占用情况；从而得到单个 FPGA 上最大可实现模式符数量；

4、验证输入正文宽度对算法资源占用的影响。

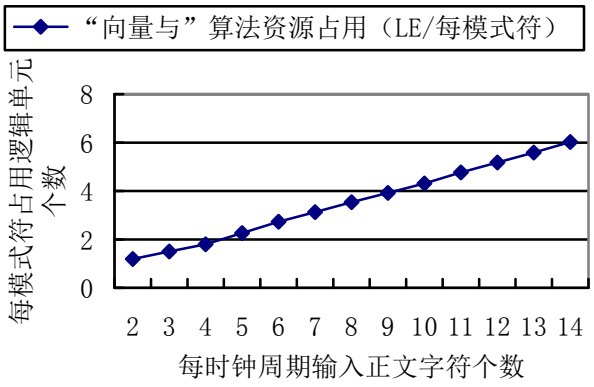

图 22 “向量与”算法资源占用

实验中使用的正则表达式模式串取自从SNORT网站上下载的snortrules-snapshot-2.8.tar.gz文件，实验使用的器件是Altera公司的Stratix II系列中的EP2S180，实验使用的综合、布局、布线工具是Altera公司的Quartus II 11.0 sp1。在以上工具中，例化本文第四章提出的“向量与”算法，分别使用250MHz和500MHz的时钟约束，在工具完成布局、布线完成后查看逻辑单元总占用量。而后用逻辑单元总占用量除以例化时所用的模式符数量得到每模式符占用的逻辑单元数量，从而得到每模式符的资源占用率。

表9是我们的实现结果，列出了一些在每时钟周期正文字符和时钟频率条件下，相应的吞吐率和每模式符逻辑资源占用情况。

表 9 “向量与”算法吞吐率及资源占用

设备	字符数/时钟周期	时钟频率 (MHZ)	吞吐率 (Gbps)	逻辑单元/ 模式符
EP2S180	2	250	4	1.19
EP2S180	3	250	6	1.50
EP2S180	4	250	8	1.80
EP2S180	5	250	10	2.26
EP2S180	6	250	12	2.73
EP2S180	7	250	14	3.13
EP2S180	8	250	16	3.54
EP2S180	16	250	32	6.75
EP2S180	32	250	64	13.14
EP2S180	64	250	128	26.40
EP2S180	128	250	256	52.65
EP2S180	2	500	8	1.19
EP2S180	3	500	12	1.50
EP2S180	4	500	16	1.80
EP2S180	5	500	20	2.26
EP2S180	6	500	24	2.73
EP2S180	7	500	28	3.13
EP2S180	8	500	32	3.54
EP2S180	9	500	36	3.92
EP2S180	10	500	40	4.31
EP2S180	11	500	44	4.77
EP2S180	12	500	48	5.18
EP2S180	13	500	52	5.59
EP2S180	14	500	56	6.02
EP2S180	16	500	64	6.75
EP2S180	20	500	80	8.29
EP2S180	24	500	96	9.94
EP2S180	32	500	128	13.14
EP2S180	40	500	160	16.38
EP2S180	48	500	192	19.85
EP2S180	64	500	256	26.40
EP2S180	128	500	512	52.65

可以从表9中得到以下结论：

1、每时钟周期输入的正文符的数量可以是大于 1 的任意整数。

2、算法所能使用的时钟频率可以使用现阶段 FPGA 器件所能达到的最高频率。在一定程度上验证了 6.1.2 节的中的最大时钟频率。

3、由于本算法可跑在 FPGA 所能的最高频率下，同时，比较两种典型时钟频率下，从算法资源占用情况来看，每模式符占用逻辑单元数量与时钟频率的选择无关。

4、由图 22 可知，每模式符占用逻辑单元数量相对于每时钟周期接受正文字符数大致成线性增长。

5、由于每个 FPGA 的逻辑单元总数量是一定的，所以模式符总量与带宽的增加成反比关系。带宽要求越大，则单个 FPGA 所能容纳的模式符总量越少，反之，带宽要求越小，则单个 FPGA 所能容纳的模式符总量越多。

第二个实验在Cyclone IV E系列中的EP4CE115上测试“矩阵与”算法，测试目的：

1、Cyclone IV E 系列的 FPGA 上，测试算法的吞吐率；

2、测试算法中模式符宽度对算法资源占用的影响；

3、测试算法中输入正文宽度对算法资源占用的影响。

实验中使用的正则表达式模式串取自SNORT网站上下载的snortrules-snapshot-2.8.tar.gz文件，实验使用的器件是Altera公司的Cyclone IV E系列中的EP4CE115，实验使用的器件工具是Altera公司的Quartus II 11.0 sp1。在以上工具中例化本文第四章提出的“向量与”算法，实验时使时钟频率同为250MHz，输入数据宽度分别为8字符（64位）、16字符（128位）、32字符（256位）、64字符（512位），对应链路带宽为16Gbps、32Gbps、64Gbps、128Gbps。在以上条件下，尽可能多地例化模式串，以得到在低成本FPGA上所能使用的模式符总量。表10是我们的实现结果，列出了一些在每时钟周期正文字符和时钟频率条件下，相应的吞吐率和每模式符逻辑资源占用情况。

从实验结果可以看出本文提出的“矩阵与”模式匹配算法很好地支持高速网络中的固定模式匹配，可处理链路带宽为输入位宽与运行算法的频率乘积，达到线速处理能力。

表 10 “矩阵与”算法正文与模式长度对算法资源占用的影响

模式符总量（字节）	**输入数据宽度（字节）**		
模式串长度（字节）	**16**	**32**	**64**
4	33245	15678	7289
5	32774	15189	6814
6	32287	14723	6456
7	31946	14298	5989
8	31342	13856	5547
9	30887	13513	5038
10	30498	13098	4598
11	29864	12675	4115
12	29432	12148	3654
13	28960	11754	3198
14	28441	11219	2786
15	28021	10827	2224
16	27548	10365	1076
吞吐率	32Gbps	64Gbps	128Gbps

用图形化表示表10的数据得到图23：

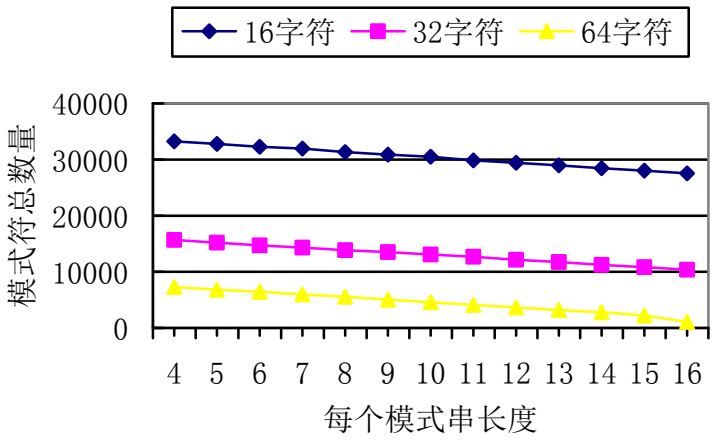

图 23 “矩阵与”算法资源占用

可以直观地得到以下结论：

1、模式符总量会随正文输入宽度增加而显著减少。

2、模式符总量随模式串长度增加略有下降。

根据图23可以看出：在实际应用中，可以根据现实的需要，在性能与容量之间进行一定的平衡。总的来说对吞吐率要求越高，则单片FPGA中所能包含的模式串数量就会越少。反过来，如果在线路带宽需求不高的情况下，单片FPGA中所能包含的模式串数量就可以很大。

由以上两个实验可以看出“向量与”算法不仅能实现在高端FPGA上，而且在低成本的FPGA上同样可以实现。差别仅在于算法受到芯片最大可用时钟频率的影响，所能使用的最大时钟频率不同，从而导致最大吞吐率不同。即使在Cyclone IV上对于SNORT规则集的数据而言本文提出的算法也能达到64Gbps的吞吐率，这个吞吐率已经大于实用的最高网络带宽——40Gbps。另外验证了“向量与”算法只依赖于FPGA中基本逻辑单元，不依赖其它资源。

6. 2. 3 结果对比

本文提出的算法仅使用到了与、或、非门和D触发器等基本的逻辑单元，所以不需要其它器件仅在FPGA上就可实现本文提出的算法。本节针对6.2.2中实验结果与前人工作结果做一些对比。所选择的算法也都是在FPGA上实现高吞吐率正则表达式算法。所比较的主要指标是算法的最大时钟频率、吞吐率和每模式符的资源占用。

由表11可以看出：

1、“矩阵与”算法在资源占用上比其它算法节约 50%；

2、最大吞吐率提高了一个数量级。

3、“矩阵与”算法可以在 FPGA 所能使用的最大时钟频率上运行，其它算法的时钟频率与每时钟周期输入的正文宽度有关。

表 11 各种正则表达式算法资源占用比较

算法	作者	设备	输入宽度（位）	频率 (MHz)	吞吐率 (Mbps)	逻辑单元/模式符
Brute Force	**Cho et al** [66]	EP20K	32	90	2880	10.6
	Sourdis et al [65]	Virtex-1000	32	171.0	5472	16.6
		Virtex2-1000	32	344.0	11008	16.6
		Virtex2-6000	32	252.0	8064	19.4
DFA	**Mos-cola et al** [20]	VirtexE-2000	8	37	296	5.5
		VirtexE-2000	32	37	1184	19.4
NFA	**Hutchings et al** [46]	Virtex-1000	8	30.9	247	2.6
		VirtexE-2000	8	52.5	420	2.6
		VirtexE-2000	8	49.5	396	2.5
Comp. NFA	**Clark et al** [47]	Virtex-1000	8	100.1	801	1.1
		Virtex2-8000	8	253.0	2024	1.7
		Virtex2-8000	32	218.9	7004	3.1
		Virtex2-8000	64	114.2	7310	5.3
		Virtex2P-125	128	129.0	16516	9.7
		Virtex2P-125	256	141.4	36194	31.5
Matrix-And	**Jiang et al**[63]	EP2S180	32	500	16000	1.80
		EP2S180	64	500	32000	3.54
		EP2S180	128	500	64000	6.75
		EP2S180	256	500	128000	12.46
		EP2S180	512	500	256000	23.47
		EP2S180	1024	500	512000	45.23

Shift-Or改进算法[21] 是2009年Hwang发表的仅使用FPGA逻辑实现的固定模式串匹配算法。本文第三章提出的高速固定模式串匹配算法同样仅使用FPGA逻辑实现的固定模式串匹配算法。两者都是在Altera公司[33]的Stratix II系列FPGA上实现。正则表达式模式串也

同样取自SNORT的规则集。因此具有可比性。另外，本文提出的算法不能实现每时钟周期输入一个正文字符。同时Hwang在论文中仅提供了每时钟周期输入一、二、四和八个正文宽度的实现结果，所以只能进行每时钟周期输入二、四和八个正文字符的对比结果。

表 12 两种固定模式串算法资源占用比较

算法	作者	设备	输入宽度（字节）	频率 (MHz)	吞吐率 (Gbps)	逻辑单元/模式符
Shift-Or modified	**Hwang** [21]	EP2S15	1	500	4	0.83
		EP2S15	2	500	8	1.01
		EP2S15	4	500	16	1.78
		EP2S15	8	500	32	3.78
Vector-And	**Jiang et al**[32]	EP2S180	2	500	8	1.19
		EP2S180	4	500	16	1.80
		EP2S180	8	500	32	3.54

综合两者的实验结果，分别列出各自的最大时钟频率、吞吐率和每模式符的资源占用等数据，得到表12。用图24来对比两者的资源占用会更直观。

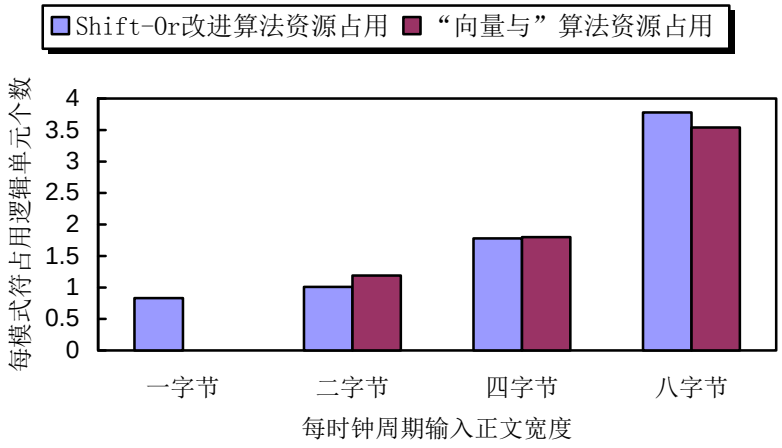

图 24 两种算法资源占用

表12是“向量与”算法和改进Shift-Or算法[21]吞吐率和资源占用比较，通过比较可以看出：资源占用方面，在输入正文字符宽度较小情况下本文提出的算法资源占用高，但随着每时钟周期输入正文字符宽度的增加，“向量与”算法的资源占用增加量较小，增加幅度小于Shift-Or改进算法。另外，吞吐率方面，由于Shift-Or改进算法的特性，通过将多个字符作为一个字符的方法来提高匹配速度，所以Shift-Or改进算法每时钟周期输入正文宽度不能很大，吞吐率最大也就是该文中提出的32Gbps。与之相比“向量与”算法模式串与正文串无关，且每个正文字符间也相互独立，所以每时钟周期输入正文宽度只受器件特性和资源限制，吞吐率可以增加的比较大。根据实验“向量与”算法的吞吐率可以达到512Gbps。

结束语

一、全文总结

模式匹配不仅是计算理论的基础，而且在计算机和网络处理中，有着广泛地应用。现阶段，正则表达式应用在与计算机相关的各种领域内，搜索引擎的基础是正则表达式、数据库的实现中也大量运用正则表达式、在网络安全领域的网络入侵与检测系统中正则表达式的使用量也呈现出迅速增加态势。随着信息爆炸及网络带宽的迅速增加，无论是信息查询的需要还是网络安全的需求，线速地处理网络数据成为了必然要求。无论是基于包内容的网络入侵检测系统（IDS）和入侵防护系统（IPS），还是电子邮件检测系统（ClamAV）、应用层过滤、基于内容的路由、出版/评阅网络和语义网络等应用，都需要线速包内容检测的支持。这些系统与应用的基础就是模式串匹配。现阶段深度包识别面临的主要难题是在满足链路带宽条件下，处理大量模式匹配。例如：在 SNORT　IDS 规则集、Bro 等系统中，每一个模式代表一种恶意流量。所有到达的数据包都要与系统中的所有规则进行匹配，每一个成功的匹配都会触发一种预定义的行为对到达的数据包进行相应地处理。在高速网络中实现正则表达式匹配主要需求为：一、模式匹配速度要满足 40Gbps 以上链路的线速处理。二、单个芯片中容纳的模式符数量要满足并行处理的模式数量要求。

针对以上需求，本文进行了比较详细的研究与实践。众所周知，系统吞吐率为系统时钟频率与每时钟周期处理的数据位数乘积。为了提高系统吞吐率可以通过提高时钟频率和每时钟周期处理的数据位数等两种方法。但现代 FPGA 的最高时钟频率为 500MHz，仅通过提高时钟频率的方法已不能满足现代网络链路的需要，因此还必须要提高每时钟周期处理的数据位数。提高每时钟周期处理的数据位数不但要在同一时钟周期内处理更多的数据，而且还要考虑多个时钟周期间的数据相关性。由此本文提出了一种新的固定模式串匹配算法——“向量与”算法。该算法把模式匹配分为解码矩阵和比较模块链两部分来处理。第一部分为通过设计新的解码矩阵完成一个时钟周期内多个正文的解码，达到生成多个正文与所有模式符比较结果的目的。加快了大量模式符和输入正文字符之间比较操作的运算速度。另一部分为比较模块链，每个模式符比较模块，根据各自需要从上述解码矩阵中提取一列比较结果，该结果与本模块的使能向量按位与生成本模块的部分匹配结果，比较结果经合适的变换后生成下级使能向量。比较模块链既考虑到同一时钟周期内多个数据处理，又考虑到多个时钟周期内数据的相关性。另外，正则表达式模式匹配都是通过 DFA 或 NFA 来实现。假定模式串长度为 m，正文串长度为 n 的条件下，DFA 的计算复杂度是 O(n+m)，而 NFA 的计算复杂度是 O(nm)。一方面，由于 FPGA 的使用开始于上世纪九十年代左右，真正大量应用是在本世纪；另一方面，处理机中核的数量有限，不支持大规模并行计算，因此开始于上世纪六、七十年代正则表达式研究中，大量的研究人员能使用的工具只有软件，于是在研究正则表达式匹配算法时首选的方案是 DFA。由于上述原因，在 DFA 的实现上有大量研究成果。NFA 上研究的人员相对要少得多。随着网络带宽的迅速增

加和实时应用的需求，模式匹配的速度要求越来越高，同时模式串数量也越来越多。自动机的另外一个约束——状态机的存储需求变得越来越重要。理论上 DFA 的状态复杂度是 O(2m)，NFA 的状态复杂度 O(m)。近年来在 DFA 上研究时面临的主要难题是状态爆炸。为了避免该难题的出现，本文选择不会出现状态爆炸的 NFA 来直接实现正则表达式匹配。在固定模式串匹配算法的基础上，针对正则表达式中的并、Kleene 星号、交运算，本文改进“向量与”算法，提出了把正则表达式中的算子转换为相应的逻辑运算方法，从而提高了算法的整体运算速度。

由于正则表达式的是指一个用来描述符合某个句法规则的字符串集合的单个字符串。一个正则表达式可以匹配最多无限多个字符串。在证明本文所提出的算法能正确地构造出匹配正则表达式描述的所有字符串，并且还要证明算法构造的自动机所能匹配的字符串集合与正则表达式描述的字符串集合完全相同时，需要用数学手段来证明。因此在提出算法的过程中，本文在提出和证明了六个定理，并在这六个定理的基础上证明了本文提出的算法正确性和算法与 NFA 的等价性。

通过以上的构造和证明，并在 FPGA（现场可编程门阵列）上验证本文提出的算法，结果表明：本文提出的算法充分地利用了 FPGA 的特性，提高了正则表达式的吞吐率。迄今为止，公开文献中给出的最大吞吐率都在 40Gbps 之下，有资料可查的商用最大吞吐率不超过 20Gbps。在本文最后一章给出了本文提出的算法能达到的最大吞吐率。通过实验表明，在现有 FPGA 中本文提出的算法的吞吐率可以达到 512Gbps。这个吞吐率已经远远超过现代网络中最大带宽。

在完成第一个目标——模式匹配速度要满足 40Gbps 以上链路的线速处理后，针对第二个目标——单个芯片中容纳的模式符要满足并行处理的模式数量要求，对“向量与”算法进行了改进。首先针对经常使用的字符间或操作，特别是一个模式符匹配所有字母、所有数字、某个范围和多个字符中的一个单字符操作，直接使用“向量与”算法，会浪费大量的逻辑资源，提出了扩展解码矩阵方案。一方面，资源占用上，在单个模式对可以匹配多个正文字符的条件下，占用更少的资源，相应增加了可用模式的总量。另一方面，扩展解码矩阵方案把所有单字符间并的影响约束在扩展解码矩阵中，对后续运算不再产生影响，简化了后续运算。其次为了减少算法对 FPGA 资源的占用，及提高运算速度。针对“向量与”算法中大量使用了向量与运算，提出了“矩阵与”算法，即把“向量与”算法中大量使用的两输入与门合并为多输入与门。一方面，因为现代 FPGA 中使用四输入查找表、六输入查找表、甚至是八输入查找表来实现与、或门，所以相对于使用两输入与、或门而言，使用多输入的与、或门能更好地利用 FPGA 的特性，从而减少 FPGA 资源使用；另一方面，因为相对于“向量与”算法使用多级两输入与门实现算法与的连接运算而言，改进为矩阵与运算使用一级多输入与门完成相同的连接运算，可以改进算法整体的时延特性。最后，针对补运算——正则表达式实现算法中的一个难题。公开文献中要么很少提及补运算的实现，要么所有提及的算法都是以大量资源占用的方式来实现补运算。提出了补运算改进算法，本文通过扩展解码矩阵，将补运算的影响限制在解码矩阵中，其它部分的运算

中不再区分模式符是否是补运算，把补运算转化为常用的连接运算，解决了正则表达式中补运算实现的难题。在本文中通过两个例子分别说明了“向量与”算法和补运算改进算法在实现补运算时的资源占用情况，通过例子可以看出补运算改进算法节约了大量资源占用。

本文最后一章给出了“向量与”算法理论上的最小时钟周期、最大时钟频率和最大吞吐率。随后在现有 FPGA 上进行了实验与仿真。得到以下结论：1、每时钟周期输入的正文符的数量可以是大于 1 的任意整数；2、算法所能使用的时钟频率可以使用现阶段 FPGA 器件所能达到的最高频率；3、从算法资源占用情况来看，每模式符占用逻辑单元数量与时钟频率的选择无关；4、每模式符占用逻辑单元数量相对于每时钟周期接受正文字符数大致成线性增长；5、由于每个 FPGA 的逻辑单元总数量是一定的，所以模式符总量与带宽的增加成反比关系。带宽要求越大，则单个 FPGA 所能容纳的模式符总量越少，反之，带宽要求越小，则单个 FPGA 所能容纳的模式符总量越多；6、模式符总量会随正文输入宽度增加而显著减少；7、模式符总量随模式串长度增加略有下降；8、“矩阵与”算法在资源占用上比其它算法节约 50%；9、最大吞吐率提高了一个数量级。10、“矩阵与”算法可以在 FPGA 所能使用的最大时钟频率上运行，前人算法的时钟频率与每时钟周期输入的正文宽度有关。

另外，由于本文提出的算法仅使用 FPGA 逻辑资源，不使用其它资源，因此也不会受 FPGA 与外部设备之间连线及传输的影响。同时也增加了系统的可扩展性，可通过使用更多资源的 FPGA 和增加 FPGA 数量的方式增加整个系统中实现的模式符数量。

二、展　望

论文在正则表达式匹配上，做了一些研究，也得到了一些有意义的结果，完成了算法设计与证明工作。还有一些工作因为时间关系未能做出进一步探索与研究。

1． 数学工具一般性

本文一个重要贡献是提出了一套用于证明 NFA 正确性的数学工具，并用这些工具证明了本文提出的算法。但这套工具是否具有普适性，是否能适用于所有 NFA 类的算法的证明，还有待进一步的应用与证明。

2． 完整应用系统地构成

虽然本文提出和证明的算法，为实际应用提供了核心算法，但距离完整应用还有很长的路要走。

3． 模式串的动态更新

实际系统一个重要指标是模式串的动态更新。在本算法的设计之初就已经关注了动态更新问题，因此将每个模式比较与处理设计的相对独立，相互之间通过连线联接。在更新模式串时只需更改连线关系就可组合成不同的模式串。然而，只做这些还不能完成系统动态更新的要求，还需要做进一步研究，以满足实际环境的需要。

4． 模式串合并

Lin[62]在 2007 年对多个模式串合并进行了初步研究，本文由于时间关系，没能在这个

方面做深入研究。这里只说明初步结论：由于本文中正则表达式是用 NFA 来实现，由于 NFA 的状态复杂度 O(m)，因此在合并多个正则表达式时，有利之处在于如果任意两个表达式之间存在公共子串，则形成的状态数量要小于原有两个表达式状态数之和，一般情况下，状态转移边的数量也会小于原有两个表达式状态转移边之和。不利之处在于，两个表达式合并后要记录匹配路径，在匹配成功时要结合匹配路径才能判断出是否匹配和匹配了哪个正则表达式。

参考文献

[1] Computer Economics, Annual Worldwide Economic Damages from Malware Exceed $13 Billion[EB/OL], http://www.computereconomics.com/article.cfm?id=1225, June 2007.

[2] Mi2g, Digital Attacks Report - SIPS Monthly Intelligence Description, Economic Damage - All Attacks – Yearly [EB/OL], http://www.mi2g.net/cgi/mi2g/sipsgraph.php, September 2004.

[3] Eleanor McKenzie, eHow Contributor, Computer Viruses & How They Affect Our Economy [EB/OL], http://www.ehow.com/info_8739209_computer-viruses-affect-economy.html, updated October 05, 2011.

[4] Fox News: Britain Hires Ex-Hackers to Beef Up Cybersecurity [EB/OL], http://www.foxnews.com/story/0,2933,529094,00.html, June 25, 2009

[5] ClamAV web site [EB/OL]. http://www.clamav.net.

[6] Linux Layer7 Swicthing web site [EB/OL]. http://www.linux-l7sw.org/.

[7] A. Carzaniga, M. J. Rutherford and A. L.Wolf. A Routing Scheme for Content-Based Networking [C]. In Proceedings of IEEE INFOCOM 2004,.

[8] D. S. Rosenblum and A. L. Wolf. A Design Framework for Internet-Scale Event Observation and Notification [C]. In Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97), 1997, 344-360.

[9] A. Carzaniga, D. S. Rosenblum and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service [J]. In ACM Transactions on Computer Systems, 2001, 19(3):332-383.

[10] G. D. Ritchie and F. K. Hanna. Semantic Networks - A General Definition and a Survey [J]. In Information Technology: Research and Development, 1983, 2(4):187-231.

[11] G. Gilder, Telecosm: How Infinite Bandwidth Will Revolutionize Our World [EB/OL], September 2000. Free Press.

[12] M. Roesch, Snort - lightweight intrusion detection for networks [C], in Proceedings of LISA'99: 13th Administration Conference, 1999, 229–238.

[13] SNORT official web site [EB/OL], http://www.snort.org.

[14] Bro NIDS web site [EB/OL]. http://www.icir.org/vern/bro-info.html.

[15] A. V. Aho and M. J. Corasick, Efficient String Matching: an Aid to Bibliographic Search [J], Comm. ACM, June 1975, 18(6), 333–340.

[16] Donald Knuth, James H. Morris, Jr, Vaughan Pratt, Fast pattern matching in strings [J]. SIAM Journal on Computing, 1977, 6 (2): 323–350.

[17] C. R. Clark and D. E. Schimmel, Efficient reconfigurable logic circuit for matching complex network intrusion detection patterns [C], in 13th International Conference on Field Programmable Logic and Applications, 2003, 956–959.

[18] Harrt R. Lewus, Christos H. Papadimitriou 著，张立昂，刘田译，计算理论基础（第 2 版）[M]. 北京：清华大学出版社，ISBN 7-302-03948-8/TP2310, 2000.

[19] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms [J]. IBM J. Res., 1987. 2, 249-260.

[20] Baeza-Tates, R., and Gonnet, G. H., A New Approach to Text Searching [J], Communications of the ACM, 1992, 35(10), 74-82.

[21] W. J. Hwang, C. M. Ou, Y. N. Shih, and C. T. D. Lo, High throughput and low area cost FPGA-based signature match circuit for network Intrusion detection [J], Journal of the Chinese Institute of Engineers, 2009, 32, 397-405.

[22] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm [J]. Commun. ACM , 1977, 20(10), 762-772.

[23] M. Aldwairi, T. Conte, and P. Franzon, Configurable string matching hardware for speeding up intrusion detection [J], Proc. ACM SIGARCH Comput. Arch. News, 2005, 33(1): 99–107.

[24] L. Tan and T. Sherwood, Architectures for bit-split string scanning in intrusion detection [C], Micro, IEEE, 2006, 110–117.

[25] H. J. Jung, Z. K. Baker, and V. K. Prasanna, Performance of FPGA implementation of bit-split architecture for intrusion detection systems [C], presented at the 20th Int. Parallel Distrib. Process. Symp. (IPDPS), Rhodes Island, Greece, 2006.

[26] J. van Lunteren, High-performance pattern-matching for intrusion detection [C], in INFOCOM, 2006, 1–13.

[27] Anat Bremler-Barr, David Hay, Yaron Koral, CompactDFA: Generic State Machine Compression for Scalable Pattern Matching [C], INFOCOM, 2010 Proceedings IEEE, 2010, 1-9.

[28] P. Piyachon and Y. Luo, Compact state machines for high performance pattern matching [C]， in Proc. 41nd IEEE/ACM Des. Autom. Conf. , 2007, 493–496.

[29] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. H. Granidt, Towards gigabit rate network intrusion detection [C], in Proc. the Eleventh Annual ACM/SIGDA International Conference on Field-Programmable Logic and Applications (FPL ’03) , 2002, 404–413.

[30] I. Sourdis and D. Pnevmatikatos, Pre-decoded CAMs for efficient and high-speed NIDS pattern matching [C], in Proc. 12th Annu. IEEE Symp. Field Program. Custom Comput. Mach. (FCCM) , 2004, 258–267.

[31] Yuan Wen , Xingsheng Tang , Lihan Ju , Tianzhou Chen, PeRex: A Power Efficient FPGA-based Architecture for Regular Expression Matching [C], Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications, 2011, 188-193.

[32] Kunpeng Jiang, Huifang Guo, Shengping Zhu, and Julong Lan, Static Patterns Matching for High Speed Networks [C]，2012 年信息计算与应用国际会议（ICICA2012）, 2012

[33] Altera 公司网站 [EB/OL], http://www.altera.com.

[34] Warren S. McCulloch and Walter H. Pitts, A logical calculus of the ideas immanent in nervous activity [J], Bulletin of Mathematical Biology, 1943, 5, 99-115

[35] S. C. KLEENE, Representation of events in nerve nets and finite automata [A], Automata Stutdies, edited by C. E. Shannon and J. McCarthy, Princeton University Press, 1956, 3-41,.

[36] Ken Thompson. Programming Techniques: Regular expression search algorithm [J]. Commun. ACM ,June 1968, 11(6), 419-422.

[37] G. H. Mealy. A method for synthesizing sequential Circuits [J]. Bell System Technical Journal, 1955, 34(5), 1045-1079

[38] E. F. Moore. Gedanken experiments on sequential machines [J]. In: Automata Studies, C. E. Shannon and J. McCarthy(ed.) ,1956, 129-153.

[39] Michael O. Rabin and Dana Scott. Finite Automata and Their Decision Problems [J]. IBM Journal of Research and Development, April 1959, 3(2):114-125.

[40] R. McNaughton and H. Yamada, Regular Expressions and State Graphs for Automata [J], IEEE Transactions on Electronic Computers, 1960, 9, 39–47.

[41] A. Mukhopadhyay, Hardware algorithms for non-numeric computation [J], IEEE Trans. Comput. , 1979, C-28(6), 384–394.

[42] Robert W. Floyd and Jerey D. Ullman. The compilation of regular expressions into integrated circuits [C]. Symposium on Foundations of Computer Science, 1980, 260-269.

[43] M. J. Foster, Avoiding latch formation in regular expression recognizers [J]. IEEE Trans. Comput. , 1989, 38(5), 754–756.

[44] R. Sidhu and V. K. Prasanna, Fast Regular Expression Matching Using FPGAs [C], in Proceedings of the 9th Annual IEEE Symposium on Field- Programmable Custom Computing Machines (FCCM) , 2001, 227–238.

[45] Yi-Hua E. Yang, Weirong Jiang, and Viktor K. Prasanna. Compact architecture for high-throughput regular expression matching on FPGA [C]. In Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '08), ACM, New York, NY, USA , 2008, 30-39.

[46] B. L. Hutchings, R. Franklin, and D. Carver, Assisting Network Intrusion Detection with Reconfigurable Hardware [C], in Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) , 2002, 111–120.

[47] C. R. Clark and D. E. Schimmel, Scalable Parallel Pattern-Matching on High-Speed Networks [C], in IEEE Symposium on Field-Programmable Custom Computing Machines, 2004, 249–257.

[48] C.-H. Lin, C.-T. Huang, C.-P. Jiang, and S.-C. Chang, Optimization of regular expression pattern matching circuits on FPGA [C], Proceedings of the conference on Design, automation and test in Europe, 2006, 12–17.

[49] J. Moscola, Y. H. Cho, and J. W. Lockwood, A scalable hybrid regular expression pattern matcher [C], in Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06) , 2006, 337–338.

[50] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, Implementation of a Content-Scanning Module for an Internet Firewall [C], in Proceedings of the 11th Annual IEEE Symposium

on Field-Programmable Custom Computing Machines (FCCM) , 2003, 31–38.

[51] Norio Yamagaki, Reetinder Sidhu, Satoshi Kamiya, High-speed regular expression matching engine using multi-character nfa [C], Field Programmable Logic and Applications - FPL, 2008 , 131-136.

[52] Hao Wang , Shi Pu , Gabriel Knezek , Jyh-Charn Liu, A modular NFA architecture for regular expression matching [C], Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays, 2010, Monterey, California, USA

[53] Z. K. Baker, H.-J. Jung, and V. K. Prasanna, Regular Expression Software Deceleration For Intrusion Detection Systems [C], in 16th International Conference on Field Programmable Logic and Applications, 2006, 418–425.

[54] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection [C]. In ACM SIGCOMM, September 2006.

[55] Benjamin C. Brodie, David E. Taylor, and Ron K. Cytron. A Scalable Architecture For High-Throughput Regular-Expression Pattern Matching [J]. SIGARCH Comput. Archit. News, 2006, 34(2): 191-202.

[56] 于强，霍红卫，一组提高存储效率的深度包检测算法[J], 软件学报 ISSN 1000-9825,Journal of Software, 2011, 22(1):149-163.

[57] Kunpeng Jiang, Shuqiao Chen, Huifang Guo and Yufeng Li, High-Throughput and scalable matching algorithm [J]，International Journal of Digital Content Technology and its Applications （JDCTA，ISSN 1795-9339）, 2012.

[58] Kunpeng Jiang, Julong Lan and Youjun Bu, High Throughput Constraint Repetition for Regular Expression Matching Algorithm [C]，2012 年信息计算与应用国际会议（ICICA2012）, 2012

[59] Kunpeng Jiang, Julong Lan, Huifang Guo and Tiefeng Li, The Performance Analysis of Vector-And Algorithm [J], Journal of Convergence Information Technology (JCIT，ISSN 1975-9320), 2012.

[60] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to Automata Theory, Languages, and Computation (2nd Edition) [M]. USA: Addison Wesley, November 2000.

[61] Ioannis Sourdis, Designs & Algorithms for Packet and Content Inspection [D], ISBN 978-90-807957-8-5, electronic and computer engineer Technical University of Crete geboren te Corfu, Griekenland, 2007

[62] Cheng-Hung Lin, Chih-Tsun Huang, Chang-Ping Jiang, and Shih-Chieh Chang, Optimization of Pattern Matching Circuits for Regular Expression on FPGA [J], IEEE transactions on very large scale integration (VLSI) systems, 15, 1303-1310, 2007

[63] Kunpeng Jiang,Huifang Guo,Julong Lan, NONDETERMINISTIC FINITE AUTOMATA FOR HIGH-SPEED"，The International Conference on Automatic Control and Artificial

Intelligence (ACAI2012), 2012, 5, 3682-3685.

[64] Xilinx 公司网站 [EB/OL], http://www.xilinx.com.

[65] I. Sourdis, D. Pnevmatikatos, Fast, Large-Scale String Match for a 10 Gbps FPGA-based Network Intrusion Detection System [C], In Proceedings of International Conference on Field Programmable Logic and Applications (FPL), 2003.

[66] Y. H. Cho, S. Navab, W.H. Mangione-Smith, Specialized Hardware for Deep Network Packet Filtering [C], In Proceedings of International Conference on Field Programmable Logic and Applications (FPL), 2002.

作者简历　攻读博士学位期间完成的主要工作

一、 个人简历

姜鲲鹏，男，1972 年 11 月出生，山东黄县人。

1993 年 7 月毕业于信息工程学院　无线电工程专业；

2000 年 9 月入信息工程大学电子技术学院军事装备学专业读硕士研究生，2003 年 6 月获硕士学位；

2006 年 9 月入信息工程大学信息工程学院读博士研究生。

二、攻读博士学位期间发表的学术论文

1．A scheme of online P2P stream indentification, 2011 International Conference on Information Security and Intelligence Control (ISIC’11)，第一作者

2．High Throughput Constraint Repetition for Regular Expression Matching Algorithm，2012 年信息计算与应用国际会议（ICICA2012），第一作者

3．The Performance Analysis of Vector-And Algorithm，Journal of Convergence Information Technology（JCIT，ISSN 1975-9320），第一作者

4．High-Throughput and scalable matching algorithm，International Journal of Digital Content Technology and its Applications （JDCTA，ISSN 1795-9339），第一作者

5．NONDETERMINISTIC FINITE AUTOMATA FOR HIGH-SPEED，The International Conference on Automatic Control and Artificial Intelligence (ACAI2012)，第一作者

6．Static Patterns Matching for High Speed Networks，2012 年信息计算与应用国际会议（ICICA2012），第一作者

7. High Throughput Regular Expression Matching Algorithm，Globecom 2012 - Communication and Information System Security Symposium，第一作者，在投

8．多核处理器片上 Cache 的选择性复制策略．计算机工程．2009.02．排名第二

9．A Method of Dataflow Analysis for OpenMP Programs. IEEE 国际会议 ACAI2012. 2012.03．排名第二

10．A thread-level pipeline parallel model for CMP．IEEE 国际会议 CiSE2009（EI 检索，检索号 20101212799791）．2009.11．排名第三

11. A Novel Multi-Next Hop Routing Algorithm Based on Node Potential，Journal of Computers (JCP, ISSN 1796-203X)，第三作者

12．FPGA 动态局部重构技术研究进展, 信息工程大学学报 2009 年 第 01 期，第三作者

13．可实现三级重构的路由交换设备, 信息工程大学学报 2009 年 第 02 期，第三作者

四、攻读博士学位期间的科研情况

1．融合多层并行处理的网络接入设备一体化硬件实现方法, 发明专利 中华人民共和国国

家知识产权局 2008 年, 发布单位 : CN200810022951.3，排名 4

2．一种组播转发表输出端口的虚拟标识方法, 发明专利 中华人民共和国国家知识产权局 2006 年发明专利 发布单位 : CN200610076504.7，排名 4

3．整包数据的传输方法及传输系统, 发明专利 中华人民共和国国家知识产权局, 2008 年, 发布单位 : CN200810087595.3，排名 6

致　谢

时光荏苒，转眼我的博士学习告一段落。回首这几年，点点滴滴都离不开他人的关心和帮助。在此，谨向所有帮助过我的老师、同学、朋友和亲人表示深深的谢意！

向我的博士生导师兰巨龙教授表示最衷心的感谢和最诚挚的敬意！兰老师严以律己、宽以待人的人格魅力将影响我的一生，他严谨的治学态度、渊博的知识、忘我地工作作风和高尚的个人品质，无时无刻不感染着我、激励着我要再努力、再提高。无论是学习、生活还是工作，在我攻读博士学位的每一个十字路口，他总能耐心地、一针见血地给我点拨迷津和指导，还给予我精神上的鼓励和充满人生智慧的的教诲，我的每一点进步都凝聚着导师的精心培育。与导师共处的日子，将给我今后的工作、科研和生活带来莫大的裨益。

感谢信息技术研究所第三研究部的陈庶樵部长、张建辉部长、伊鹏部长和程东年教授,与他们一起的科研、学习使我的学术水平、科研能力得到了很大的提高。感谢部门中的李玉峰、王雨、卜佑军、张风雨、贺磊、张校辉、于婧、田志等同事们在工作与生活上对我的帮助与支持，与你们共同工作为我的研究工作提供了灵感与基础。感谢我的师兄弟和姐妹们，感谢我的博士同学们，他们是：赵博、万成威、罗文宇、董永吉等，一起的讨论、交流，我们共享思维的灵感与创造的快乐，攻读中的点点滴滴让我们结下终生难忘的友谊。感谢信息技术研究所的陈鸿昶所长、陈志强政委、李印海副所长、张天忠主任，研究生队的梁中彪队长和张晓明政委，感谢你们给予的信任、帮助及所营造的优越学习环境和宽松的科研氛围。

特别感谢我的爱人郭惠芳，是她一方面在工作与研究上的支持、特别是在我论文写作期间做了大量文字工作；另一方面完成了繁重的家务和照顾孩子，是她的理解、关心和支持让我全力以赴，全身心地投入于自己的工作、学业和论文；我的宝贝姜炫伊，你们给予我无上的精神鼓励和理解支持。感谢我的母亲，我的家人，你们在我生命里的所有无私关爱和体贴。

感谢百忙之中参与本论文评议、评阅和答辩的各位专家。