基于 Verilog 的正则表达式编译器的实现①

邓凯元

(北京信息科技大学 光电信息与通信工程学院，北京 100101)

摘 要：随着网络带宽的快速增长，正则表达式匹配逐渐成为网络数据处理系统的性能瓶颈。为了获得更高的匹配效率，基于 FPGA 的正则表达式匹配引擎成为近年来的研究热点之一，而将正则表达式高效的转换成硬件描述语言是其中的关键技术。首先分析了正则表达式转换为硬件电路的算法，然后在此算法基础上实现了一个编译器。最后在 Modelsim 平台上进行了仿真，仿真结果证明了编译器的正确性。

关键词：正则表达式；FPGA；模式匹配；Verilog

Implementation of Regular Expression Compiler Based on Verilog

DENG Kai-Yuan

(School of Photoelectric Information and Communication Engineering, Beijing Information Science and Technology University, Beijing 100101, China)

Abstract: With the rapid development of network bandwidth, regular expression matching is becoming the performance bottleneck of networking data processing system. For the purpose of achieving higher matching speed, regular expression matching engine based on FPGA has become one of the recent hot fields. And it is the critical technology how to efficiently transfer regular expression to hardware description language. In this paper, we first analyze the algorithm of converting regular expression to hardware circuit, and then implement a compiler based on this algorithm. Finally we simulate the circuit on the Modelsim platform. The results proved the accuracy of the compiler.

Key words: regular expression; FPGA; pattern matching; Verilog

随着网络带宽的日益增长，带来了近乎无限的共享和机会，但同时也极大增加了网络用户遭受恶意攻击的机会。入侵检测技术是解决网络安全问题的一个重要方案，在入侵检测系统中，正则表达式匹配技术得到了广泛地应用，例如，SNORT[1]，ClamAV[2]，L7-Filter[3]等开源的入侵检测系统都大量的使用了正则表达式描述的规则，并且其数量也越来越多。在网络入侵检测系统中，正则表达式的匹配问题已经成为了系统的性能瓶颈之一。基于软件实现的正则表达式匹配引擎已经无法满足网络带宽高速增长的需求。因此，基于可编程逻辑门阵列（FPGA）实现正则表达式匹配的硬件加速成为了目前研究的热点之一。

则表达式的匹配技术主要有基于 NFA（非确定型自动机）的匹配技术、基于 DFA（确定型自动机）的匹配技术、基于 NFA 和 DFA 的混合匹配技术、位并行技术和过滤型匹配技术[4]。其中 NFA 方案消耗少量的内存，而 FPGA 天然的并发性又可以很好的解决 NFA 带来的回溯问题，基于这些原因，使得在 FPGA 上使用 NFA 实现正则表达式的匹配成为主流方案。在文献[5]中，经过测试表明，对比于软件实现，可以取得两个数量级的加速比。

在实际设计中，如果快速，准确的把给定的正则表达式转换成硬件描述语言并映射到 FPGA 电路是一个关键问题。现实中入侵检测系统的正则表达式规则数量都大于 100 条，显然用人工方法进行转换是非常低效的，也极其容易出错。开发一整套的自动转换工具成为了非常重要且迫切的任务。在本文中，使用 Visual C++开发了编译器，可以快速的将正则表达式转

① 收稿时间:2011-06-11;收到修改稿时间:2011-07-11

换成 verilog 实现，最后在 Modelsim 上进行了仿真测试，测试结果证明了编译器的正确性。

本文下面的章节安排如下：第一部分介绍了正则表达式的硬件构造算法。第二部分介绍了编译器的架构与实现。第三部分以 L7-filter 中的正则表达式规则集，在 Modelsim 上进行了仿真，证明了编译器和算法的正确性。第四部分则是对全文的总结。

1　硬件构造算法

1.1 正则表达式

正则表达式是一种可以匹配一个或多个字符的模式字符串。基本的正则语法由普通字符和一些元字符（meta-characters）组成[6]。正则表达式具有 4 个基本操作：单个字符、“|”、“?”、“*”。其中单个字符由大小写字母、数字和“ε”字符组成，“ε”表示空字符。其他三个操作符含义如下：令 r1，r2 是正则表达式，r1|r2 表示：匹配任何被 r1 或 r2 匹配的字符串；r1·r2 表示：匹配任何前缀字符串被 r1 匹配，剩余字符串被 r2 匹配的字符串；r1*表示：匹配零个或多个被 r1 匹配的字符串。

在现有的系统中，往往引入了更多的符号来表示正则表达式，比如“+”，“?”等等，但以上四种操作符是正则表达式最基本的操作。

1.2 匹配电路构造算法

2001 年，R. Sidhu 与 V. K. Prasanna 两人给出了从正则表达式到基于 FPGA 的非确定型有穷自动机（NFA）的映射算法[5]，该算法给出了正则表达式的四种基本操作（单个字符、r1|r2、r1r2、r1*）转化为 NFA 逻辑电路的基本构造方法（如图 1 所示）。

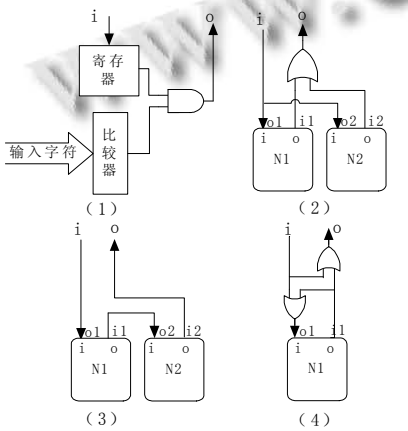

图 1　基本操作的电路结构

如图 1 所示，四个子图分别对应了正则表达式的四个基本操作：单个字符、“|”、“?”、“*”。子图(1)实现了单个字符，寄存器中存放自动机的上一级状态，比较器的输出结果跟输入状态进行“与”操作，输出执行完当前匹配的自动机状态。子图(2)中的 N1 和 N2 表示由子表达式 r1，r2 构造的子电路，不难看出，当 N1，N2 中有一个电路匹配成功，整个电路的输出便是匹配成功，从而实现了正则表达式的“|”操作。同理，子图(3)实现了正则表达式的“?”操作。子图(4)实现了正则表达式的“*”操作。

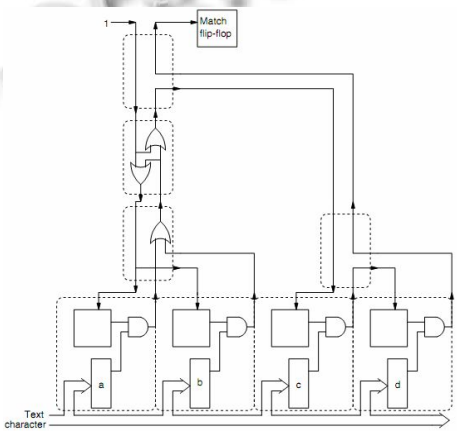

图 2　“((a|b)*)(cd)”对应的电路逻辑

这四种操作递归组合，可以组合成为任意的正则表达式。该算法构造的 NFA 电路每匹配一个字符的开销为一个时钟周期。图 2 表示的是正则表达式“((a|b)*)(cd)”按照算法构造而成的逻辑电路。该电路按照递归方式构造，首先把正则表达式变成后缀形式：“ab|*cd??”。然后依次读入后缀表达式中的内容，当读入“a”时，构造一个单字符 a 的逻辑电路；接着读入“b”，构造一个单字符 b 的逻辑电路；接着读入“|”，构造一个或操作的逻辑电路，并把之前构造好的两个逻辑电路连接起来，作为一个新的逻辑电路。依次类推，读入“*”号和“.”号时，都按图 1 中 给出的构造方法构造电路，最后整个后缀表达式读完得到的电路结构就是需要正则表达式匹配引擎。

2　编译器架构与实现

2.1 整体架构

在系统设计中，采用了 1.2 节中的算法来生成硬件电路，将正则表达式规则转变成硬件电路，下载到

FPGA 的整体流程如图 3 所示。

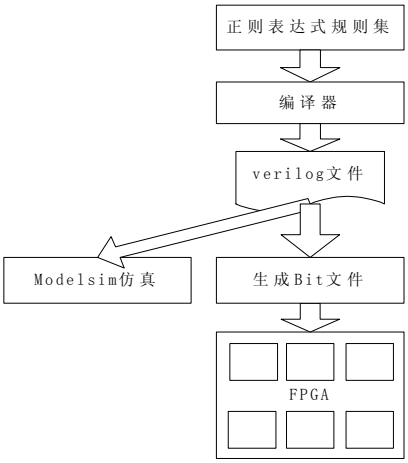

图 3 整体流程

如图 3 所示，将正则表达式转变成匹配引擎需要经过编译，仿真，综合，下载等步骤。

首先，从 SNORT 或者 L7-filter 规则集中抽取出正则表达式规则。然后编译器读取规则文件，根据每条正则表达式规则生成对应的 verilog 文件。这里需要注意的是，SNORT 规则集合 L7-filter 规则集中，有很多复杂的语法，比如字符组，为了适应硬件实现，需要先将规则改写成以 1.1 节中介绍的四种基本操作实现的形式，在 2.2 节中会对规则改写做进一步的说明。Verilog 文件生成以后，需要进行行为仿真，观察输出波形，看是否能够正确产生匹配结果。最后经过工具的综合和布局布线，产生了能下载到 FPGA 上的 Bit 文件，烧写到开发板上。这里需要注意的是，布局布线的过程通常需要消耗几个小时，因此需要在经过行为仿真校验，确保正确之后再进行硬件的综合，以节省时间，提高开发效率。

2.2 编译器的设计与实现

在 2.1 节介绍的整体流程中，编译器的设计是最关键和最重要的部分。编译器需要正确并且高效的把正则表达式规则转换成可被综合的硬件电路描述语言（HDL），综合工具则把生成的 HDL 源文件进一步编译成可以被 FPGA 使用的二进制文件。

实际的入侵检测系统中使用的正则表达式规则很多比较复杂，会出现很多的字符组，转义字符，计数重复等情况。为了适应硬件实现，必须首先把规则改写为可以用 1.1 节中的四种基本操作表示的形式。再做进一步的处理。如图 4 所示，是编译器处理规则的基本流程。

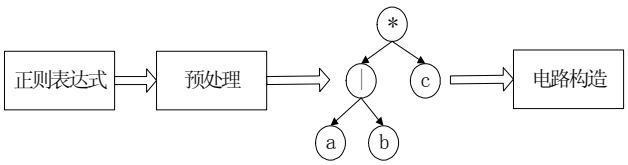

图 4 编译器架构

预处理的过程是把复杂语法改写为简单语法的过程。基本的改写策略如下：遇到“[a-d]”形式的字符组，改写成“a|b|c|d”的形式；遇到“\d”形式的转义字符，改写成“0|1|2|3|4|5|6|7|8|9”的形式；遇到“a+”形式的语法，改写成“aa*”的形式；遇到“a?”的语法，改写成“a|ε”的形式；遇到“a{1,3}”形式的语法，改写成“a|aa|aaa”的形式；遇到“a{3,}”形式的语法，改写成“aaaa*”的形式。除了以上几种情况之外，还有一些语法是硬件无法实现的，比如反向引用，零宽断言[7]等。遇到出现这些语法的规则，预处理器不再处理，并将规则编号通知编译器。

经过预处理器处理的正则表达式，已经全部是由四种基本语法构成。下一步把正则表达式解析成语法树之后，可以按照 1.2 节中的构造方法，递归的构造出硬件电路。递归算法伪代码如下：

```
Procedure Build(TREE)
1 node<-- TREE.NODE
2 case node
3 “|”: left←Build(TREE.LEFT)
4 right←Build(TREE.RIGHT)
5 PLACE_|(left,  right)
6 “·”:left←Build(TREE.LEFT)
7 right←Build(TREE.RIGHT)
8 PLACE_·(left,right)
9 “*”: left <-- Build(TREE.LEFT)
10 PLACE_* (left)
11 char:  PLACE_char(node)
12 End of case
13 End of procedure
```

3 仿真结果

在下载到真正的 FPGA 芯片之前，需要先对实现进行仿真。只有仿真结果完全正确之后，才能开始下

一步的布线工作。由于在开发流程中综合和布局布线所需要的时间通常比较久（经常需要几小时），而仿真需要的时间只有几分钟，进行全面的仿真可以大大节省硬件开发的时间，提高效率。

在设计中，编译器生成的 verilog 文件在 Altera 公司的 Quartus II 平台下进行综合，在 Modelsim 6.5 平台下进行仿真。下面的图显示了对于随机生成的字符串“abcdacdbcdaacdcdefageadgasaddfd7”，正则表达式“((a|b)*)(cd)”经过编译器生成的匹配引擎的仿真结果。

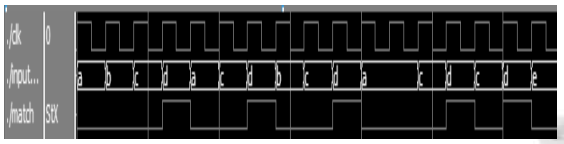

图 5　仿真结果

如图 5 所示，正则表达式“((a|b)*)(cd)”中包含了 1.1 节中提到的四种基本操作。在第一次出现“cd”的时钟周期内显示了匹配成功，共出现了五次匹配，匹配结果完全正确。

测试使用的规则很简单，为了能全面的验证编译器的正确性，采用了实际使用的更大的规则集。规则集提取了 L7-filter 中的正则表达式规则，共 101 条规则。规则的最大长度有 4096 个字符，并有很多的类似“[A-P]”的字符组和“.*”的重复语法。经过测试表明，这 101 条规则能全部编译成功，随机抽取了其中的 20 条进行仿真，仿真结果也完全正确，正确率为 100%。

4　结论与进一步工作

随着网络带宽的快速增长，正则表达式匹配逐渐成为网络数据处理系统的性能瓶颈。为了获得更高的匹配效率，使用硬件实现正则表达式匹配成为近年来的研究热点之一，由于 FPGA 具有易开发，可重复编程等特性，基于 FPGA 的正则表达式匹配研究日益流行。将正则表达式高效的转换成硬件描述语言是其中的关键技术。本文首先分析了正则表达式转换为硬件电路的算法，然后在此算法基础上实现了一个可以把正则表达式转换为逻辑电路的编译器。最后在 Modelsim 平台上进行了仿真，仿真结果证明了编译器的正确性。

目前本文提到的转换算法可以把正则表达式最基本的四种操作：单个字符、“|”、“·”、“*”转换成逻辑电路。虽然其他的一些操作，比如字符组，转义字符等等，通过预处理都可以变成由这四种基本操作组成，但是还有一些操作无法转换，比如零宽断言，反向引用等。这些复杂语法的有效实现是下一步研究的内容。除此之外，编译器只是实现了最基本的转换功能，并没有针对正则表达式的一些特征进行电路面积和时钟频率的优化，未来需要针对这些方向，进一步改进编译器的实现。

参考文献

1 Roesch M. SNORT network intrusion detection system.2007. http://www.snort.org.

2 Tomasz Kojm, Clam AV. antivirus engine.2011.http://www.clamav.net./

3 Filtr L. “Project WWW Page,”2011.http://l7-filter.Sourcefor–ge.net.

4 杨毅夫.面向深度包检测的正则表达式匹配技术.北京:中国科学院计算技术研究所,2009.

5 Sidhu R, Prasanna VK. Fast Regular Expression Matching using FPGAs. The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. Rohnert Park, 2001:227–238.

6 Friedl Jeffrey EF. Mastering Regular Expressions. O’Reilly Media, 2006.

7 Hazel P. PCRE-Perl Compatible Regular Expressions. http://www.pcre.org