A Scalable Hybrid Regular Expression Pattern Matcher *

James Moscola, Young H. Cho, John W. Lockwood
Department of Computer Science and Engineering
Washington University, St. Louis, Missouri 63130
{jmm5, young, lockwood}@arl.wustl.edu

1 Introduction

As new Internet technologies emerge, the ability to do full regular expression pattern matching on network packets to support these new technologies is becoming increasingly important. Current technologies that can benefit from high-speed pattern matching are firewalls, network intrusion detection systems, virus scanners, spam filters, and content-based routers. All of these technologies are ruled-based scanners which have rule sets that are continually increasing in size. In combination with increasing network speeds, it is becoming difficult for software based approaches to maintain the required network throughput. Thus is it important to develop high-speed hardware-based pattern matchers.

We present a reconfigurable hardware architecture for searching for regular expression patterns in streaming data. This new architecture is created by combining two popular pattern matching techniques: a pipelined character grid architecture [1], and a regular expression NFA architecture [2, 3, 4]. The resulting hybrid architecture can scale the number of input characters while still maintaining the ability to scan for regular expression patterns.

2 Hybrid Architecture

This section shows how to create the hybrid architecture which is both scalable and capable of scanning for full regular expressions. String detectors in the hybrid architecture consists of chains of pipelined 4-input AND gates. The output of each stage of the pipeline is asserted when all of its input characters are present and the output of the previous stage is asserted. In a system with a single character wide input bus, a pipelined character grid is used to buffer characters as they enter the system. Figure 1 illustrates the pipelined character grid for the hybrid architecture. Figure 2 shows the chain structure required for matching the eleven character pattern "aaabbbcccaa". Because some of the pipelining is moved from the character grid into the pipelined chain, the pipelined character grid can actually be shorter than the length of the pattern.

*This research was sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under Contract number MDA972-03-9-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

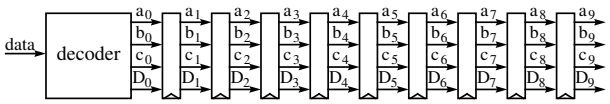

Figure 1. Pipeline for grid/chain hybrid

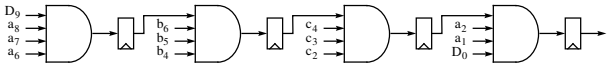

Figure 2. Hybrid Chain for "aaabbbcccaa"

2.1 Regular Expression Pattern Matching

In addition to strings, the hybrid architecture also has the ability to match regular expression patterns. Among the supported regular expression operations are *zero-or-more*, *one-or-none*, *one-or-more*, and *alternation*. An example of each of these operations is shown in figure 3.

Both the *zero-or-more* operation (figure 3a) and the *one-or-more* operation (figure 3b) require a feedback path that allows the hybrid chain to iteratively match repeating substrings. Additionally, these two operations require that additional registers be placed in that feedback path. The additional registers ensure that the pipelined character grid has enough time to receive the characters required for the next iteration of the regular expression operation. Note that the total number of registers required in the feedback path is equal to the number of characters involved in the given regular expression operation. For example, in figure 3a the regular expression "aaa(abc)*cc" contains the *zero-or-more* operation. The number of characters included in the *zero-or-more* operation is three. This means that the total number of registers required in the feedback path is also three.

The regular expression operations in figure 3c is the *one-or-none* operation. This operation does not include a feedback loop, but it does require additional registers similar to the *zero-or-more* and *one-or-more* operations. Again, the number of additional registers required is equal to the number of characters included in the regular expression operation.

The final operation show in figure 3d, the *alternation* operation, is the simplest of the regular expression operations. It requires the addition of only a single OR gate to the hybrid chain. No additional delay registers are required.

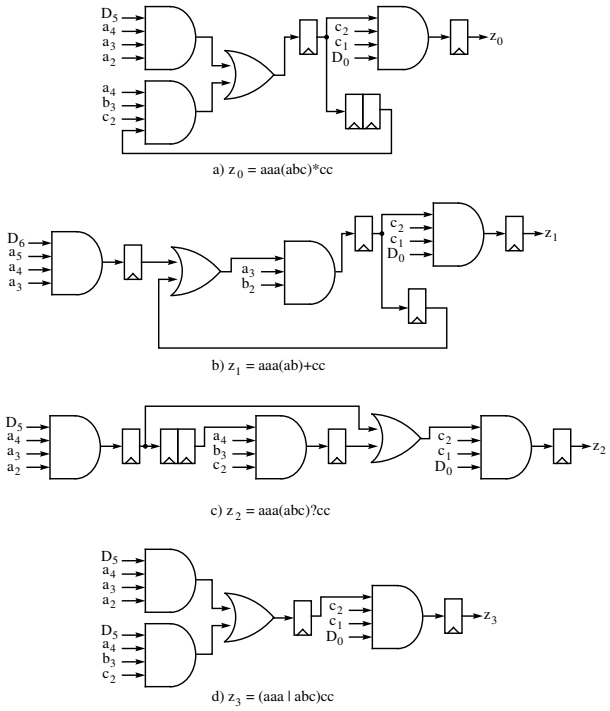

Figure 3. Hybrid regular expression operations

More complex regular expression can be constructed using the basic components for each of the regular expression operations. An example using both the *zero-or-more* operation and the *alternation* operation is shown in figure 4. Notice that the different size substrings in the *alternation* operation require a different number of delay registers. Thus an extra D-flip-flop is inserted at the input of the larger substring.

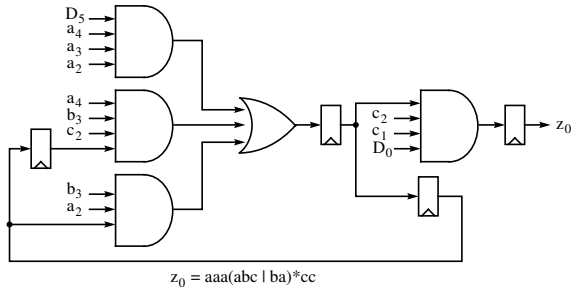

Figure 4. Hybrid chain for "aaa(abc|ba)*cc"

2.2 Scaling the Hybrid Architecture

The hybrid architecture can also be scaled while still maintaining the ability to search for regular expression patterns. Scaling the hybrid architecture requires the replication of logic to search for patterns at all possible starting alignments. This section illustrates how our architecture can be used to scan for regular expressions

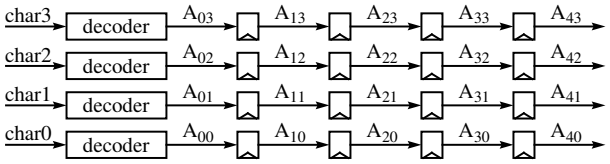

Figure 5. Four character wide pipeline

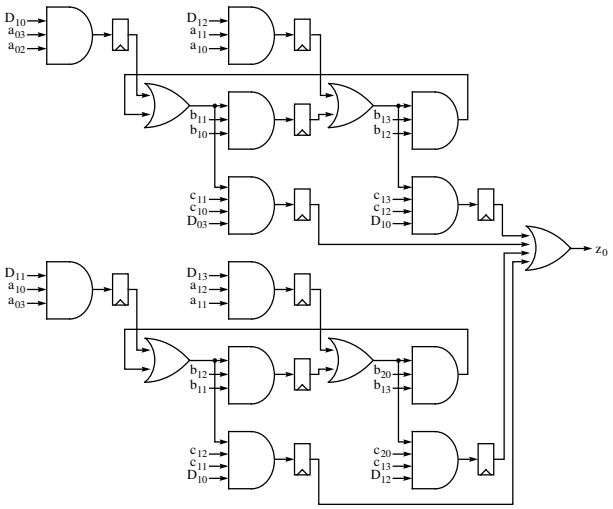

Figure 6. Hybrid chain for "aa(bb)*cc"

on a four-character wide input bus. Though the architecture can scale to larger bus widths. Figure 5 shows the four-character wide pipeline that is used for the scaled regular expression pattern example. The notation A_{ij} represents the alphabet of decoded characters $\{a_{ij}, b_{ij}, c_{ij}, ...\}$. Figure 6 shows the logic required for the regular expression "aa(bb)*cc". Note that similar logic structures can be created for larger and more complex regular expressions.

References

[1] Z. K. Baker and V. K. Prasanna. A Methodology for Synthesis of Efficient Intrusion Detection Systems on FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2004. IEEE.

[2] Y. H. Cho, J. Moscola, and J. W. Lockwood. Context-Free-Grammar based Token Tagger in Reconfigurable Devices. In *International Workshop on Semantics enabled Networks and Services (SeNS)*, Atlanta, GA, Apr 2006. IEEE.

[3] C. R. Clark and D. E. Schimmel. Scalable Parallel Pattern-Matching on High-Speed Networks. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2004. IEEE.

[4] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching using FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2001. IEEE.