Send Orders of Reprints at reprints@benthamscience.net

46 *The Open Electrical & Electronic Engineering Journal,* 2013, *7,* (Supple 1: M4) 46-50

Open Access

Design of a Regular Expression Matching System Based on Network on Chip

Linhai Cui*, Yusen Qin, Fanyang Kong and Kaihong Yu

Software School, Harbin University of Science and Technology, 150040, Harbin, China

Abstract: This paper presents an efficient method for Regular Expression Matching (REM) by reusing Intellectual Property (IP) cores in a new architecture of Network on Chip (NoC). The method is to design a reusable IP core which consists of many engine cells for REM and to implement each engine cell on a Field Programmable Gate Array (FPGA) as a prototype. To make Finite State Machine (FSM) simpler, a new approach for partitioning a regular expression into several smaller parts is proposed. Each part of a regular expression was matched by an engine cell during matching, and each engine cell communicates with others by routers on a NoC topology. The proposed NoC architecture is a general-purpose design which is suitable for different rule libraries in deep packet inspection (DPI). It can deal with the problem that character self-deplete made the correct regular expression matching missing. A way to use both logic cell and RAM available on FPGA devices is described, and it can make it easier to change the rule library of regular expressions in the RAM. The implementation of the NoC architecture by employing application-specific integrated circuits (ASIC) is finally discussed.

Keywords: Regular expression, FPGA, NOC, Apart-NFA.

1. INTRODUCTION

With the rapid development of network applications, network security becomes more and more important against system intrusion. Many network services now process packets based on payload content, in addition to the structured information found in packet headers. Forwarding packets based on content requires new levels of support in networking equipment. A regular expression, often called a pattern, is an expression that specifies a set of strings. To specify such sets of strings, rules are often more concise than lists of a set's members. Regular expression matching is one of main mechanism that used by Intrusion Detection Systems (IDS) like Snort [1] or simpler ones like L7-filter [2] even some email spam filter. For most string matching, a regular expression is often represented by deterministic finite automata (DFA) or non-deterministic finite automata (NFA). DFA is commonly used to parse regular expressions.

More and more logic cells can be made on a single chip due to the advancement in IC fabrication technologies. This allows us to develop much more complex systems on a chip of silicon, (system on a chip, SoC). A reusable design is a design that shouldn't change much and convenient to reuse. It can be reused in newer design and can make the design cycle shorter. It is possible to match a regular expression by using hardware.

An FPGA is an integrated circuit to be configured by a customer or a designer after manufacturing, and it can be used to implement any logical function that an ASIC could perform. The configuration of FPGA is specified by using a hardware description language (HDL). FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together. Logic blocks can be configured to perform complex combinational functions. In most FPGAs, the logic blocks also include memory elements which may be simple flip-flops or more complete blocks of memory.

It is too difficult to store states of DFA on FPGA due to the complexity of regular expression. The most common method is string based NFA. We found treating regular expression as matching with head of data in most papers is a serious fault, and it will affect REM a lot and probably make mistakes called character self-deplete mistake that will be further discussed in section 4.

A new NoC architecture is proposed for solving the character self-deplete problem. In this architecture, a general-duty reusable design for string matching as an engine cell is provided. This architecture has the advantages of high speed and parallel processing. A new complex reusable form of engine cells within this architecture for effective regular expression matching was introduced.

For saving area, the regular expression to be matched was stored in cache/memory, so does the op-code that controls the engines. Experiments show that each engine cell only cost 48 logic cells in our verification design on FPGA.

A methodology is introduced to generate the regular expression that suit to our architecture, and the regular expression can be rewritten easily without change the circuit because we store it in memory.

2. BACKGROUND

Regular expression gains widespread adoption for deep packet inspection together with the network application. To implement REM in hardware has been widely studied be-

*Address correspondence to this author at the Software School, Harbin University of Science and Technology, 150040, Harbin, China;
Tel: 86-45186397000; E-mail: cuilinhai@hrbust.edu.cn

1874-1290/13

Table 1. Controller Modes and Instruction Set

Meta-Character Example and Description	Op-Code (i_0~i_3)	Mode	Postfix (i_{20}~i_{31})
ab: only ASCII-characters	0001	Controller match 'a' with data stream character input at the this clock, and match 'b' at the next clock	If 'ab' was 2 character or 1 of the pattern (may be only one character here), here store the place in which pattern, and if it was a apart-NFA status over.
[a-z]:a class range of character.	0010	Controller judge if the ASCII value of data stream character input between 'a' and 'z'	Here store the place in which pattern, and if it was a apart-NFA status over.
[^ab]:any character but 'ab'	0100	Controller match 'a' with data stream character input at the this clock, and match 'b' at the next clock	Data that let controller make the result negate and the place in which pattern, and if it was an apart-NFA status over.
(ab)+: match 'ab' once or more and data (a)+ is acceptable.	1000	Controller match 'a' with data stream character input at the this clock, and match 'b' at the next clock. pointer=pointer-n	Data that let controller make the next part of NOC that store the rest of regular expression available. This status must be the end in our structure. Here store the 'n'(pointer backtrack vector).
a*:match 0 times or more.	1001	Controller match 'a' with data stream character	Data that let controller make the next part of NOC that store the rest of regular expression available.

*It matches the preceding element zero or more times

fore. Floyd and Ullman [3] discussed the implementation of NFAs using hardware. Since regular expression can be accepted by FSM. Sidhu and Prasanna [4] proposed an algorithm to implement regular expression matching in regular expression NFA (RE-NFA) architecture on FPGA. Yang and Prasanna [5] improve this architecture process 2 characters per clock cycle, and they got a result which has a concurrent throughput of 14.4Gbps for 760 regular expression matching. It is almost the fastest one in our survey result by RE-NFA. For the L7 with 70 protocol filters, the system throughput is less than 10Mbps, and more than 90% of the CPU time is spent in regular expression matching. These are all explained [6, 7], which transmit the regular expression into a FSM by a state table have 256 columns for ASCII character set. They analyzed and implemented their design using a plausible ASIC, and achieved a 16Gbps throughput.

A System-on-chip (SoC) synchronized by a global clock tree may be more prone to electromagnetic interference (EMI) [8]. On a silicon chip, it's difficult to send a global signal across whole chip within a real-time bounds [9]. For processing deep packet inspection, a design is complicated but each regular expression is simple. GALS (globally asynchronous and locally synchronous) divides a system into smaller, locally decoupled synchronous regions and composes some of them to a localized subsystem. NoC is one of the GALS solution [9], it can supporting reuse of complex cores.

For the NoC architecture, there have been various topologies include mesh, torus, ring, butterfly and irregular interconnection networks [10, 11]. Compared to other topologies, some researchers have suggested that 2-D mesh architecture for NoC will be efficient, ease of implementation, terms of latency, power consumption [12]. However, as NoC architectures are based on packet-switched networks, so that a suitable and efficient principle for design of routers for NoC is important [13].

3. REUSABLE ENGINE CELL DESIGN

As an engine cell, a basic circuit for character matching is designed. As a FSM, it contains a controller and a list memory, and it gets data from the list memory by using a pointer and matching it with input in each clock cycle. Combined with the NoC architecture, this engine cell could process almost all the regular expression Meta-character. This inspiration is from CPU and NoC router.

The controller have several modes for dealing with different Meta-character. It depends on the 32 bit instruction set that described in Table **1**.

The controller's modes are coded by op-code, so does the character status by postfix. Pointer will be added 1 after every cycle. Table **1** shows the way to process meta-characters that discussed in most papers. Other meta-character like (* ?) will be described in section 4.

For the op-code 1010 to 1111, they have some other simpler and easier operations and can be used to deal with other part of regular expression.

We designed a prototype system on a FPGA chip which is produced by Altera to verify the functions of the engine cell. And the experiment results show that the cost and timing of the engine cell are stable and is convenient to reuse.

For snort, we updated the engine cell to a 64 bit code design with 48 bit data, i.e. 6 characters. The experiment results show that it can make the system much faster and more efficient.

The logic circuit based on different length of regular expression will produce different length of combo logic and register, different input delay and output delay, and difficult constraints. For most designs, however, FPGA is only a prototype for verification because FPGA always have a lower clock than that of ASIC.

4. SOLUTION OF APART-NFA ON NOC ARCHITECTURE

In this section, the character self-deplete mistake is discussed first, then a new method for expressing regular expression called Apart-NFA is introduced, and finally a NoC architecture solution is proposed.

4.1. Character Self-Deplete Missing Mistake

Intrusion detection systems do more work than basic firewalls since they look inside the contents of the packets as well as the header fields. This makes things more complex, as we may not have any particular location within the packet to inspect now and we may have to scan the packet's entire contents for the data items we are looking for. Snort is probably the most well known intrusion detection system. This operates by using a set of intrusion detection rules—the first part will check for packets on the basis of the header fields and then we may look inside selected packets for strings need to be matched.

Snort system allows the use of regular expressions to perform matching. These are useful because we can use a single regular expression to match multiple variants of the pattern which we are looking for. Regular expression matching can be more complex to implement, particularly for us to build a hardware implementation rather than a software one. The requirement for regular expressions has been increasing over the last few years and a large number of Snort rules now use them.

Traditional RE-NFA structure has some disadvantages. For most regular expression in snort, the matching begins with "^" which means the start position of a line or a packet. But some of the regular expressions need to match the payload of data stream. For an example of L7-filter "msnmassager.pat" it needs to match the whole packet: "\x03\x9a\x89\x22\x31\x31\x31\.\x30\x30\x20\x42\x65\x74\ x61\x20|\xe2\x3c\x69\x1e\x1c\xe9". Traditional way for RE-NFA circuit to deal with the data stream may encounter a problem because the stream matching circuits do not have any latch to hold the data. However, if there were latches, they shouldn't keep one clock one character matching.

If traditional RE-NFA circuit is used for matching in IDS, the problem mentioned above will affects the matching a lot and would make the IDS faulty.

In Snort, 23.5% patterns begin without "^". This is a mistake that would make matching system work improperly. We describe an architecture that could support regular expression like this consummate.

4.2. Apart-NFA

NFA is unusual in that it may has more than one node active at any time, and there may be multiple edges leaving a node that are enabled on the same input data item. NFA is not so efficient to implement in software, as the current state of the NFA will be the set of nodes that are currently active. Looking at each of these nodes to see which edges are enabled and hence which set of nodes will next be active. An NFA can be implemented quite efficiently in hardware as each node can be implemented as a flip-flop and the edges can be implemented as comparators, logic and routing resources to link the nodes together. Implementing NFA in hardware can improve resource utilization and performance significantly. Early designers used a comparator per NFA edge, and this requires routing resources to take the input data to lots of distributed comparators, which needs large numbers of comparisons against the same input byte values. This matching system was improved on by replacing the distributed comparators with a global 8–256 decoder which can generates 256 logic signals that identifies the presence or absence of each data byte value on the input stream.

NFA based architectures support concurrently active states without state explosion, and are much better suited to handle constraint repetitions and overlapped matching for future generation of critical applications. However, their parallel processing nature often leads to poor runtime performance on sequential execution machines. The inherently parallel processing architecture of FPGA makes it well suited for NFA implementations. In conventional designs, libraries of building blocks (template Verilog/VHDL codes) are first created for different regular expression primitives. A regular expression based rule is then parsed into those primitives, and their template codes are assembled into one system design file for compilation and synthesis [14].

For most regular expression matching, our engine cell circuit can't process the characteristics of traditional NFA for one state has several next states. We proposed a new NFA that is suitable for our new NoC architecture based on the basic IC design rules. The processing procedure is to make the engine cell uniform and then to transfer the regular expression into a traditional NFA. We parsed the regular expressions at the points that make backtrack. Then each part of regular expression is just treated as a DFA for our engine cell.

The regular expressions like [a-z][0-9] are treated as a deterministic state in our engine cell, and so does (a|b). We parsed a regular expression into two level, apart-level and group level.

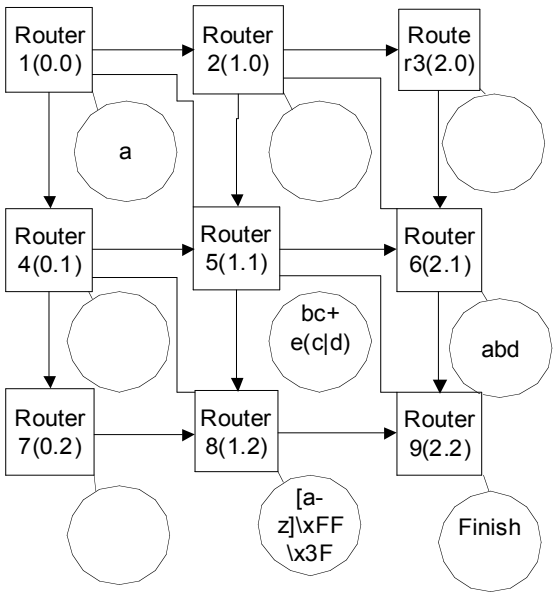
Every group must not be in the situation that one state has several next states but the end state. So every part within a group of one regular expression in the data stream could be simple and convenient for our engine cell to match in a high-speed and parallel way.

For most regular expressions in Snort, the way to partition a regular expression into several simple parts is proved to be efficient to complete our design in time.

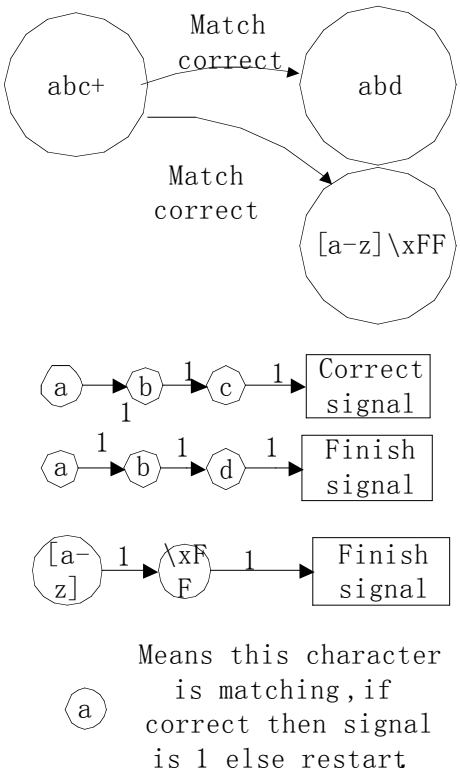
4.3. NOC Architecture Solution

For the Apart-NFA, after transferring a regular expression into a NFA, we parse it into several parts. And each part is stored in an IP core's RAM/CACHE. Whole regular expression is connected by a NoC router algorithm. Because IPs are connected by a topology that improved from mesh, so we called it mesh-pro.

Fig. (**1a**) shows a 3 level of 3X3 mesh-pro model. Cores of first level match the regular expression's head to the data stream. The first level includes the first column and the first row. Like a tree or a hash table, the matching performed by the first level cores using a common prefix will improve the speed of matching in parallel at the first time. When a prefix is matched, router will send a data that include several addresses of routers and which group level of the regular expression will be matched next. The system proposed in this paper is a globally asynchronous locally synchronous (GALS) system, so the frequency of router is higher than that of the cores in the system. When a router received information and wake up the cores, the cores should have been ready before data comes to them.



(a) A easy mesh-pro model for regular expression "abc+ ((abd) | ([a-z]\xFF))" "/a*e(c|d)*[xyz]\x3F"



(b) A complex example regular expression that '(abc)+((abd)|([a-z]\xFF))'

Fig. (1). A simple NoC architecture model.

When a regular expression like "(abc)+…" was matched, a core will send a router 2 data. One is the address of itself and the group level number of the regular expression. The other is the address of next router and the next group level number of the same regular expression. The data were stored in RAM of the core.

For a regular expression which has several stars in middle part of it like "a b*c*d", the previous state of the FSM will give the addresses of each cores and they will be matched at the same time. For the example above, the router will give the addresses of routers to the cores that is needed to match "b", "c", and "d" after the "a" in the regular expression is matched.

For meta-character "?", what we need to do is only let the core which will perform matching in next state of apart-level being alive.

Now let's talk about self-deplete mistake in more detail. Fig. (**1b**) shows the method. The first character of a regular expression was matched by the first level in the topology. The first level of the model always works if necessary for the matching. For the above example, it refers to that if the "a" character being matched incorrectly, the head core which performs matching if "a" is still working. In this way, we not only solve the problem of self-deplete mistake, but also improve the speed of matching as we described before.

Furthermore, as shown in Fig. (**1a**), we improved the topology of mesh by adding a diagonal in the model based on the previous structure and the analysis of experimental results. Due to the cores in the model are divided into different levels in the topology, a diagonal could shorten the transmission time of data.

5. EXPERIMENTAL RESULTS

We test the functionality of engine cell on Altera DE2 (EP2C35F672C6) with low performance RAM. The engine cell worked with all kinds of meta-character in Table **1** for the test case. When the data stream for test be input, the experimental results show that the return of the device is correct.

Simulations are performed by using a NoC simulator (NIRGAM). We tried the traditional 2D-mesh topology first for our design. It has 12.88G bps throughput and 13.3311 average cycle latency with 64 cores (only 38 cores in use). For, it has a 17.9447G bps throughput but some test cases have 7 to 38.7004 cycle latency with 38 cores in use. It shows that our mesh-pro topology with diagonal and only one-way down to the next level has bigger throughput but longer latency.

We changed the routing algorithm for mesh-pro topology with 2-way diagonal and a new grouping method of group level of regular expression. For a test case of 754 rules in snort, it has over 19.98G bps throughput and 7.6-8 cycle latency with 48 cores in use. It shows that our mesh-pro topology with 2-way diagonal and a new grouping method of group level of regular expression is better than the previous topology both in throughput and latency.

6. CONCLUSIONS

A new NoC architecture for regular expression matching with a new improved topology called mesh-pro and a general reusable design is proposed in this paper. The method takes the advantages of REM implementing in hardware and a new NoC architecture. It solves the communication problem of multiple cores in NoC architecture for regular expression matching and improves the throughout` of the system. The experiment results show that this model can process 6 character or meta-character per clock, and the throughput is over 19.98G bps for 754 REMs.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGEMENT

This work is supported by Natural Science Foundation of Heilongjiang Province under Grant Nos. F201205, Science and Technology Research Funds of Education Department in Heilongjiang Province under Grant Nos. 12531132.

REFERENCES

[1] SNORT. Official Web Site, Available from: [http://www.snort.org]

[2] J. Levandoski, E. Sommer, and M. Strait, "Application Layer Packet Classifier for Linux." Available from: http://l7-filter.source-forge.net/

[3] R. W. Floyd, and J. D. Ullman, "The Compilation of Regular Expressions into Integrated Circuits," In: *Foundation of computer Science, 1980; 21st, Annual Symposium*, Oct 13-15, 1980, NY: USA 1980. vol. 29, no. 3, pp. 603-622, July 1982.

[4] R. Sidhu, and V. K. Prasanna, "Fast Regular Expression Matching Using FPGAs," In: *IEEE Symposium on Field Programmable Custom Computing Machines,* April 2001.

[5] Yi-Hua, E. Yang, W. Jiang, and V. K. Prasanna, "Compact architecture for high-throughput regular expression matching on FPGA," *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems,* 2008, pp. 227-238.

[6] Y. Fang, C. Zhifeng, D. Yanlei, T.V. Lakshman, and H. K. Randy, "Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection" UCB Tech. Report, EECS-2005-8, 2006.

[7] B. C. Brodie, D. E. Taylor, and R. K. Cytron, "A Scalable Architecture For High Throughput Regular Expression Pattern Matching," In: *33rd International Symposium on Computer Architecture* (ISCA'06), 2006.

[8] S. Kumar, A. Jantsch, J-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology" *IEEE Computer,* 2002, pp. 117-124.

[9] A, Jantsch, and H. Tenhunen, *Network on Chips*. Kluwer Academic Publishers: Boston, 2003.

[10] J. Dally, and B. Towles, *Principles and Practices of Interconnection Networks. Morgan Kaufmann*: California, 2004.

[11] P. Pratim Pande, C. Grecu, M. Jones, A. Ivanov and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures" *IEEE Transactions on Computers,* vol. 54, no. 8, pp. 1025-1040, 2005.

[12] A. Agarwal, C. Iskander, H. Multisystems and R. Shankar, "Survey of network on chip (NoC) architectures & contributions," *Journal Engineering Computing Architecture*, vol. 3, no. 1, 2009.

[13] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip" *IEEE Proc. on Computers and Digital Techniques,* vol. 150, no. 5, pp. 294-302, September, 2003.

[14] Hao Wang, Shi Pu, G. Knezek, and Jyh-Charn Liu, "A modular NFA architecture for regular expression matching" FPGA'10, Monterey, California, USA, February, 2010.

Received: February 18, 2012 Revised: March 29, 2012 Accepted: October 20, 2012