High Performance Custom Regular Expression Processing Core

Thianantha Arumugam*, Sakir Sezer*, Dwayne Burns*, Vishalini Vasu*

**Institute of Electronics, Communications and Information Technology,*

Queen's University Belfast

email: tarumugam01@qub.ac.uk, s.sezer@qub.ac.uk, d.c.burns@qub.ac.uk, vvasu01@qub.ac.uk

***Abstract*— This paper discusses a novel hardware based regular expression processing core. Regular expression is widely used as a pattern matching technique to detect presence of malicious content in Internet traffic. In the proposed approach, a regular expression processing core is designed and implemented. The proposed architecture is unique whereby the processing core does not require hardware reconfiguration to support new rules. This processing core is scalable and can be used in a multi-core environment to achieve processing speeds of up to 50 Gbps. A single regular expression processing core is able to operate at 185.46 MHz. The achievable system bandwidth is 1.48 Gbps per core for 8-bit processing.**

***Keywords* – Regular expression, parallel processing, network security, content processing**

I INTRODUCTION

Content processing in network means the processing tasks performed by network elements on the content portion of network traffic. There are several reasons why network elements process Internet traffic content. Some key reasons include network security and monitoring, traffic control and management, QoS and IP based service applications such as billing. Network administrators now need more than firewall to protect their network from vulnerabilities. Both network based and desktop based solutions are required to stop undesirable content from entering and leaving their enterprise. When a new threat has been identified, signature update has to be carried out. With content processing capabilities embedded within each network element, networks are able to respond fast to new threats and traffic restrictions or filtering based on content can be deployed ensuring advanced network security capability.

The list of cyber security threats are on the rise every single day as more and more people and organizations rely on computers and internet connection to perform their tasks. The volume and severity of network based attacks grow substantially every year. A report by Symantec suggests the number of computer viruses reached more than a million for the first time in 2010. A total of 711,000 new viruses have been detected only in 2009 which 468% more compared to same period in 2008 [1].

New malicious code (including viruses) discovered every year also continues its upward trend with 2009 reporting an increase of 71% of new malicious code signatures compared to the previous year. Data also indicates that Trojan continues to be most prevalent malware although it has dropped from 66% to 57%. Similar to Trojan, adware and spyware also show decreasing trend while viruses and worms show an increasing trend [1].

This paper presents the implementation of a unique hardware based regular expression processing core using FPGA technology as prototype. The work explores the possibility of using and implementing a dedicated regular expression processing core that can be used for content processing without requiring reconfiguration of processing core for rule update while requiring a small silicon budget. Section II discusses the background work focusing mainly on significance of a hardware based regular expression processor. Section III presents related work by other research groups. Section IV discusses the design of a single regular expression core and the instructions made available in this processor for regular expression processing. Section V presents the results obtained and performance comparison against other proposed solutions. Section VI summarizes the current work and the future work.

II BACKGROUND

Pattern matching is a process of inspecting and validating a set of data against predefined patterns. This is achieved by tracing a sequence of input events and comparing the inputs against known patterns. For network security based content processing, these patterns or better known as signatures are stored in a database (SNORT, BRO, LINUX-L7). Pattern matching can be categorized to string matching and regular expression matching. String matching is an important text processing tool. The idea of string matching is to locate the appearance of a one dimensional array (of fixed patterns) in an array which is of equal size of larger than the pattern which is the text. It involves finding of one or more occurrences of a pattern in a given text. Brute force or exhaustive search is the most straightforward algorithm for string matching. It attempts to match the pattern in the target text from left to right by using a window which has a predefined size. Almost every editor and computer platform supports the use of string matching. String matching is useful only when the exact string to be matched is known. As the number of patterns and the size of pattern increases, string matching becomes ineffective for pattern matching. This problem gets even worse when task of searching is further complicated by the requirement to search for patterns which have no predefined length. Regular expression is a powerful complex pattern matching tool. Instead of using characters only for pattern recognition, regular expression allows the user to specify patterns using multiple operators ranging from normal character operators to quantifiers that covers repetitions.

A common host processor does regular expression processing by comparing a regular expression against an input in a sequential manner. FPGAs offer massive advantage for regular expression matching requirements. This is due to its inherent parallelism in its logic which allow high throughput even at low clock rates. A good solution to regular expression processing for pattern matching would be dedicated processors with a parallel processing architecture which is able to maintain decent throughput. To achieve this, small and dedicated processing cores must be available. This paper discusses the design and implementation of a single dedicated processing core with wide range of regular expression processing capabilities.

III RELATED WORK

The use of hardware as a method of acceleration for pattern matching has been explored by a number of research groups. These include Bloom filters, CAM/TCAM based circuits, discrete reconfigurable logic, partition technique, finite automata, RAM based FSM, regular expression and multi-core GPU based solutions. Some methods explored by peer research groups are presented in this subsection.

Bloom filter is a data structure that is used to test if a data is a member of a set. This attribute can be used for pattern matching applications. S. Dharmapurikar et al. has used a technique based on Bloom filters for detecting predefined signatures in a packet payload [2]. The basic idea of their work is to isolate all packets that potentially contain predefined signatures. This approach is able to handle thousands of patterns. For each possible pattern length, a unique Bloom filter is implemented. The cost of Bloom filter is independent to the length of the signature. This means, it would consume less hardware resources. Nevertheless, the algorithm of Bloom filter needs to be for a specific application. This means it would be able to handle all the possible input combinations. In summary, Bloom filter is good for occasions that will never fall under false negative category but will not be able to handle applications that might fall under false positive category.

Content Addressable Memory (CAM) or Ternary Content Addressable Memory (TCAM) is a special memory type that implements look-up table function. It works in one clock cycle using dedicated comparison circuitry [3]. This makes CAM implementation one of the best solutions for pattern matching. Content addressable memory can be generally categorized to CAM and TCAM. CAM is the more straightforward component that provides the address of the location matched when pattern matching process is performed. TCAM has a special comparison feature whereby the data compared can be ignored partially. A don’t care condition is used as mask to match more than one possibility. Gokhale et al. have used CAM to implement a set of Snort rules on a Virtex XCV1000E. Besides the payload information, their implementation also checks the header information of packets. They are able to operate at a speed of 68 MHz for a CAM size of 160 x 32-bits [4]. Fang Yu et. al. have implemented pattern matching technique using TCAM that is able to operate at gigabit rate . They use the ClamAv virus database with 1768 patterns where the sizes vary from 6 bytes to 2189 bytes. They are able operate at 2 Gbps rate with a 240KB TCAM [5]. Although these methods of pattern matching are fast, they are very inefficient. For every signature available in a virus database, a unique bank of CAMs or TCAMs has to be implemented to support just one signature. This means, the resource utilization in this method of pattern matching becomes very large. In addition, due to the nature of the implementation of this circuit, the operating speed of the whole device reaches unacceptable level and this affects the performance of the module to maintain the required line rate.

Finite automata simply refer to finite state machine as discussed in section 2.3.2. Finite automata can be classified further to Non-deterministic Finite Automata (NFA) or Deterministic Finite Automata (DFA). Both approaches have been used by researchers to propose solutions for pattern matching. Sidhu and Prasanna from University of Southern California proposed a hardware implementation for pattern matching using regular expression. They use a method to map a circuit directly to a given regular expression [6].Their approach is based on a NFA model. This approach was enhanced further by a research group in Brigham Young University. B. L. Hutchings et. al. have fitted each individual character into a slice of inputs. Their system is comprised of a sequence of 8-bit comparators and flip-flops. The circuit is able to process single character at each clock cycle. They are able to operate this at 31 MHz on a Virtex 1000 FGPA [7].

In addition to NFA, DFA based implementation has also been used. A hybrid approach has been used by James Moscola et. al. from Washington University. They have made a compiler module that translates the regular expressions directly into hardware and also generates corresponding DFA matchers[8]. This model has suffered fan-out problem. This is obvious when the number of regular expression increases.

A number of research groups have been working on pattern matching in hardware using regular expression. Franklin, Carver and Hutchings from Brigham Young University have made a complete NFA based architecture to implement regular expression pattern matching. They suggest string matching performance can be significantly improved using FPGAs. With this consideration, they have made a regular expression to FPGA circuit module generator. The module generator extracts string for the SNORT rule set and generates a regular expression that matches all the extracted strings [7]. Their processing unit is completely time-multiplexed design. In order to process 385 regular expressions, their design requires a total of 20,742 single character comparisons.

Fang Yu et. al. has implemented fast regular expression matching for packet payload scanning applications. They have explored the possibility of implementation a DFA based pattern matcher for regular expression processing. Their findings show that the memory requirements for DFA based pattern matching are extremely high. The length of patterns has and exponentially growing memory costs. They suggest that the signature structures have to be studied further to be rewritten so that memory resource consumption could be kept to a minimum [9].

A. Mitra et. al. presented a way of directly compiling PCRE to FPGA for accelerating SNORT IDS. Their test on a Virtex 4 FPGA revealed a throughput of 12.9 Gbps at 150MHz core speed. Their performance suggests a speed increase of up to 353 times compared to a baseline implementation on an Intel XEON CPU at 3.0GHz [10]. With those performance figures, the calculation shows they had 86 NFAs working parallel (assuming all NFAs are equal in size). Although the throughput figure is very high, this method is not different compared to direct implementations of NFAs to achieve regular expression processing on an FPGA. This technique of regular expression processing requires a configurable hardware and cannot be implemented using standard cell ASIC technology.

Work carried out by all the research groups suggest that a regular expression based content processing solution must be:

- Executed by custom and purpose built processor
- Memory based rule storage
- Able to support large number of rules
- Scalable using multiple instance of the same processing engine for high throughput

This research project has been motivated by these basic requirements and by the shortcomings of related work in this field of research.

IV REGULAR EXPRESSION PROCESSING CORE

There are two approaches of processing NFA based regular expressions. These are breadth-first based regular expression processing and depth-first based regular expression processing. This research mainly focuses on the architecture and implementation of a depth first based regular expression processing core. Each regular expression is processed in a sequential manner against an input stream by progressing through each state in the equivalent regular expression state graph. In the process of progressing through the state graph, the processor has to keep track of previous states (especially the start state and the states that have multiple next state options). In addition, the pointer location of the input stream should also be taken into account. This is critical in the process of backtracking when final state of a branch is reached in a state graph. Stack machine is an ideal computation model for this processor. With a dedicated hardware stack, stack machine based processor architecture is able to store critical states in the state graph to facilitate backtracking. Peer research groups have set throughput performance as their main goal. This work explores the possibility of construction of a regular expression processor which is capable of upholding a decent throughput while requiring a very small silicon budget and maintaining low power consumption. To enable the processing of a wide flavour of regular expressions, the processor supports a number of instructions. These instructions are intended to process the input stream against a given regular expression by manipulating available resources on the processor such as processing unit, stacks and registers.

The proposed regular expression processing core is able to support a wide variety processing options for regular expression. This ranges from normal character processing to complex options such as wildcard and repetitions. Generally, the rules processed by the regular expression processing core can be categorized as static or dynamic rules.

Static matching is a matching process that involves the use of fixed patterns against an input string. This pattern only contains fixed characters which includes all the characters in ASCII table. Examples of such rules found in L7-filter rule set and executable by the regular expression processing core are as follows.

Rule:
JPEG
Regular expression:
/\xff\xd8/
Explanation:
This is one of the simplest rules in the entire L7 rule set. This rule looks for the occurrence of two ASCII characters ("ff" and "d8") as shown in the regular expression above. A match can be produced by the processing unit as soon as it detects these two characters occurring in the input stream in the order defined.

Rule:
postscript
Regular expression:
/%!ps/
Explanation:
This is another rule that uses static matching only. Four static characters are expected as shown in the regular expression above. This is in the order of '%', '!', 'p' and 's'.

Dynamic regular expression rules are used when a given regular expression could match more than one character. This means there is no exact match predictable until the process starts and the inputs are analyzed. Examples of such rules extracted from the L7-filter rule set are as follows.

***Rule*:**
audiogalaxy
***Regular expression*:**
/^(\x45\x5f\xd0\xd5|\x45\x5f.*0.60(6|8)W)
***Explanation*:**
This is a regular expression rule that looks for pattern at the start of the input stream. This is indicated by the anchoring at the start of the pattern. ('^'). There are two options of pattern here. First option is a purely static component. It starts by looking for character x"45". This is then followed by three characters which are x"5f", x"d0" and x"d5". The second matching option for this rule is more complex. It starts similarly by looking for characters x"45" and x"5f". After these stream of characters, a ".*" option is available to match any character except a page break or a line break character. When it finds one of the break characters, backtracking is used to complete the search which consists of static components. The first character wanted in the backtracking operation is '0'. This is followed by '.', '6' and '0' before an option is available to choose between characters '6' or '8'. Finally, it attempts to match character 'W' before signaling a match in the input stream.

Rule:
live365
***Regular expression*:**
/membername.*session.*player/
***Explanation*:**
This rule requires static matching of three components i.e. membername, session and player. Between the static components, there are two wildcard repetition operators that could match any characters with any length until a page break or a line break character is found. Two backtracking operations would be required to complete the processing of this rule.

The main task of the regular expression processing core is to execute program codes which reflect how a regular expression is implemented. While executing the program codes, it mimics the operation of a traditional automata based implementation of a regular expression. The processor successfully executes the programs using a set of instructions made available on this processor and processing modules which perform required tasks. Figure 1 shows the block diagram of the regular expression processing engine which consists of many lightweight processing modules.

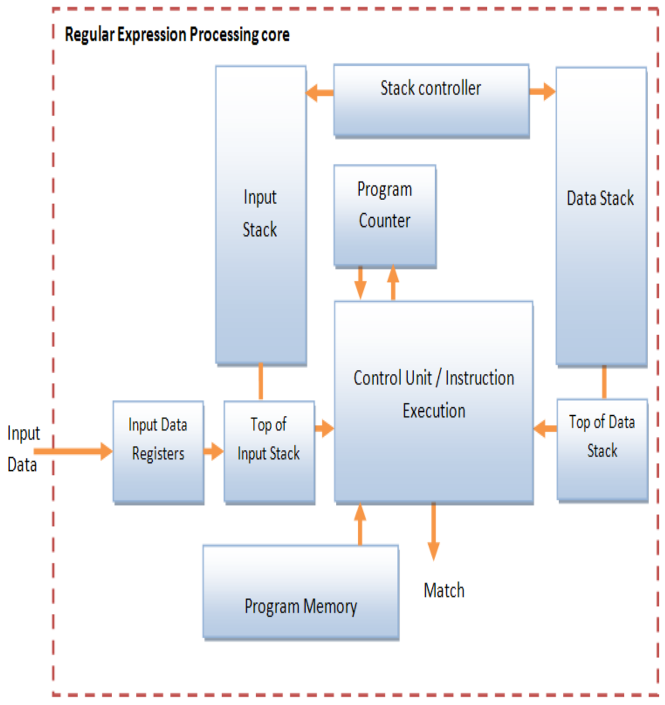

Figure 1: Block Diagram

V RESULTS

This section of the report discusses the performance or the regular expression processing core and performance comparison against other work. The proposed architecture for regular expression processing core was designed using VHDL and implemented on an Altera Stratix IV FPGA. Table below shows the synthesis results obtained for the regular expression processing core on an Altera Stratix IV FPGA technology.

Device	Reg	ALUTs	Freq (MHz)	Speed (Gbps)
EP4SGX230KF40C4ES	5,173	8984	185.46	1.48

Table 1: Synthesis results

A single processing core is made up of only 14k of logic elements. At a maximum operating speed of 185.46 MHz, the core is able to achieve a throughput rate of 1.48 Gbps. The performance comparison in Table 2 shows results obtained by various research groups for regular expression processing. The performance measure is divided to two sections showing the best overall throughput achieved and the normalised throughput achieved.

It is difficult and unreasonable to compare the performance of the regular expression processing core against other results without normalising the figures. Some of the best throughput figures in the table above are achieved by implementing 16-bit, 32-bit and 64-bit based regular expression processing. In addition, parallel regular expression processing and static matching modules are also employed in some cases.

Regular expression processing core proposed and implemented in this research work uses 8-bit input. It supports full regular expression processing unlike some other relevant work which only does string based pattern matching. There is no parallel processing of multiple regular expression rules in this architecture. With only one rule processed at any one time, the core is able to perform at a maximum throughput rate of 1.48 Gbps. This processing core uses memory based rule storage. This means, the rules can be updated for the system without having to reconfigure the entire hardware. This is a major advantage of this architecture compared to related work. The design is portable to ASIC technology where the architecture of the core is maintained while rule update is performed on a dedicated program memory. None of the proposed architecture shown in Table 2 is suitable for regular expression processing when the design is implemented on ASIC technology.

Research Group	Overall (Gbps)	Normalised (Gbps)	Comment
Bispo *et. al.* [11]	2.0	2.0	Static pattern matching + Regular expression
Clark *et. al.*[12]	2.5	0.625	32-bit, string matching only
Franklin *et. al.* [13]	0.4	0.4	String matching only
Sidhu *et. al.* [6]	0.46	0.46	-
Moscola *et. al.* [8]	1.18	0.295	32-bit per cycle
Lin *et.al.* [14]	N/A	N/A	8 -bit
Kumar *et. al.* (projected) [15]	10	N/A	Projected value, multiple rule
Brodie *et. al.* [16]	4	0.5	64-bit per cycle
Mitra *et. al.* [10]	12.9	0.15	Parallel NFAs – multiple rule
Jung *et. al.* [17]	1.4	1.4	String matching only
Regular expression processing core	1.48	1.48	Single rule, 8-bit processing

Table 2: Performance evaluation

VI CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a novel regular expression processing core comprised of many lightweight processing elements. Design and FPGA based implementation using Altera Stratix IV technology has shown that the processing core is able to achieve core speed of 185.46 MHz. Considering one character is consumed every clock cycle under ideal conditions, the proposed solution is able to achieve a maximum throughput of 1.48 Gbps. This performance is achieved with a relatively small processing core and requires very little logic resources. In addition, this solution also has low power consumption in comparison to traditional software based solutions. Since the proposed architecture is small in size at less than 15k logic elements, this number of similar processing cores can be used in parallel to achieve higher data processing speed. The architecture has been validated by manually compiling L7-filter rules and executing IP payload with embedded matching strings.

Considering the largest Altera Stratix IV (EP4SE820) devices are comprised of 325,220 ALUTs, 650,440 registers and 23 Mbits of embedded memory, a single FPGA is capable of accommodating up to 36 regular expression processing cores. Such multi-core architecture is able to achieve throughput levels of up to 50 Gbps (1.48 Gbps x 36) which will outperform any performance figure achieved by peer research groups. Future work on this field will concentrate on using multiple processing cores with a dedicated external program memory. This program memory must be large to support all the required regular expression content processing rules.

References

[1] M. Fossi, E. Johnson, T. Mack, D. Turner, J. Blackbird, M. K. Low, T. Adams, D. McKinney, S. Entwisle and M. P. Laucht, "Symantec Global Internet Security Threat Report: Trends for 2009," *Volume XV, Published April,* 2010.

[2] S. Dharmapurikar, P. Krishnamurthy and D. E. Taylor, "Longest prefix matching using bloom filters," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,* 2003, pp. 201-212.

[3] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *Solid-State Circuits, IEEE Journal of,* vol. 41, pp. 712-727, 2006.

[4] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole and V. Hogsett, "Granidt: Towards gigabit rate network intrusion detection technology," *Field-Programmable Logic and Applications: Reconfigurable Computing is Going Mainstream,* pp. 47-61, 2002.

[5] F. Yu, R. H. Katz and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," in *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on,* 2005, pp. 174-183.

[6] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using FPGAs," in *Field-Programmable Custom Computing Machines, 2001. FCCM'01. the 9th Annual IEEE Symposium on,* 2005, pp. 227-238.

[7] B. L. Hutchings, R. Franklin and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," in *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on,* 2002, pp. 111-120.

[8] J. Moscola, J. Lockwood, R. P. Loui and M. Pachos, "Implementation of a content-scanning module for an internet firewall," in *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on,* 2003, pp. 31-38.

[9] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in *Architecture for Networking and Communications Systems, 2006. ANCS 2006. ACM/IEEE Symposium on,* 2008, pp. 93-102.

[10] A. Mitra, W. Najjar and L. Bhuyan, "Compiling pcre to fpga for accelerating snort ids," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems,* 2007, pp. 127-136.

[11] J. Bispo, I. Sourdis, J. Cardoso and S. Vassiliadis, "Synthesis of regular expressions targeting fpgas: Current status and open issues," *Reconfigurable Computing: Architectures, Tools and Applications,* pp. 179-190, 2007.

[12] C. R. Clark and D. E. Schimmel, "Scalable pattern matching for high speed networks," 2004.

[13] R. Franklin, D. Carver and B. Hutchings, "Assisting Network Intrusion Detection with Reconfigurable Hardware," *IEEE Symposium on FieldProgrammable Custom Computing Machines,* April 2002.

[14] C. H. Lin, C. T. Huang, C. P. Jiang and S. C. Chang, "Optimization ofregular expression pattern matching circuits on FPGA," *Proceedings of the Conference on Design, Automation and Test in Europe,* pp. 12-17, 2006.

[15] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,* 2006, pp. 339-350.

[16] B. C. Brodie, D. E. Taylor and R. K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," 2006.

[17] H. J. Jung, Z. K. Baker and V. K. Prasanna, "Performance of FPGA implementation of bit-split architecture for intrusion detection systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International,* 2006, pp. 8.