International Journal of Computer and Communication Technology

Volume 4 | Issue 4 Article 5

October 2013

MULTI-GIGABIT PATTERN FOR DATA IN NETWORK SECURITY

Praveen Kumar. Ch
AVN Institute of Engineering & Technology, Hyderabad (A.P), India, pravi457@gmail.com

Prof.P.Vijai Bhaskar
AVN Institute of Engineering & Technology, Hyderabad (A.P), India, pvijaibhaskar@gmail.com

Ravi. Ch
#AVN Institute of Engineering & Technology, Hyderabad (A.P), India, Chini_ravi439@yahoo.co.in

B.Rambhupal Reddy
AVN Institute of Engineering & Technology, Hyderabad (A.P), India, b.rambhupalreddy@gmail.com

MULTI-GIGABIT PATTERN FOR DATA IN NETWORK SECURITY

Praveen Kumar.Ch[1], Prof.P.Vijai Bhaskar[2], Ravi.Ch[3], B.RambhupalReddy4
[#]AVN Institute of Engineering & Technology, Hyderabad (A.P), India

Email: pravi457@gmail.com[1], *Chini_ravi439@yahoo.co.in*[2],b.rambhupalreddy@gmail.com[3],pvijaibhaskar@gmail.co m [4]

***Abstract*:- In the current scenario network security is emerging the world. Matching large sets of patterns against an incoming stream of data is a fundamental task in several fields such as network security or computational biology. High-speed network intrusion detection systems (IDS) rely on efficient pattern matching techniques to analyze the packet payload and make decisions on the significance of the packet body. However, matching the streaming payload bytes against thousands of patterns at multi-gigabit rates is computationally intensive. Various techniques have been proposed in past but the performance of the system is reducing because of multi-gigabit rates.Pattern matching is a significant issue in intrusion detection systems, but by no means the only one. Handling multi-content rules, reordering, and reassembling incoming packets are also significant for system performance. We present two pattern matching techniques to compare incoming packets against intrusion detection search patterns. The first approach, decoded partial CAM (DpCAM), pre-decodes incoming characters, aligns the decoded data, and performs logical AND on them to produce the match signal for each pattern. The second approach, perfect hashing memory (PHmem), uses perfect hashing to determine a unique memory location that contains the search pattern and a comparison between incoming data and memory output to determine the match. The suggested methods have implemented in vhdl coding and we use Xilinx for synthesis.**

I. INTRODUCTION

The proliferation of Internet and networking applications, coupled with the widespread availability of system hacks and viruses have increased the need for network security. Firewalls have been used extensively to prevent access to systems from all but a few, well defined access points (ports), but they cannot eliminate all security threats, nor can they detect attacks when they happen. Stateful inspection firewalls are able to understand details of the protocol that are inspecting by tracking the state of a connection. They actually establish and monitor connections for when it is terminated. However, current network security needs, require a much more efficient analysis and understanding of the application data. Content-based security threats and problems occur more frequently, in an every day basis. Virus and worm inflections, SPAMs (unsolicited e-mails), email spoofing, and dangerous or undesirable data, get more and more annoying and cause innumerable problems. Therefore, next generation firewalls should provide Deep Packet Inspection capabilities, in order to provide protection from these attacks. Such systems check packet header, rely on pattern matching techniques to analyze packet payload, and make decisions on the significance of the packet body, based on the content of the payload. Since string matching is the most computationally intensive part of an NIDS, our proposed architectures exploit the benefits of FPGAs to design efficient string matching systems. The proposed architectures can support between 3 to 10

Gbps throughput, storing an entire NIDS set of patterns in a single device. In this thesis we suggest solutions to maintain high performance and minimize area cost, show also how pattern matching designs can be updated and partially or entirely changed, and advocate that brute force solutions can offer high performance, while require low area. Techniques such as fine-grain pipelining, parallelism, partitioning, and pre-decoding are described, analyzing how they affect performance and resource consumption.

This thesis provides CAM-like architectures and perfect hashing memory (PHmem) for efficient and high-speed string matching. It also evaluates our solutions in terms of performance and cost, discusses its advantages and drawbacks, compares it with related architectures, and presents possible improvements and alternative solutions. Developing VHDL representation of large designs that store hundreds of patterns is a time-consuming procedure. Therefore, it is important to automatically generate the VHDL code of a design that stores a particular set of patterns. This work describes an automatic implementation methodology for the proposed architecture, in order to generate the desired design fast. Objective of this paper is Pattern matching is a significant issue in intrusion detection systems, but by no means the only one. Handling multi content rules, reordering, and reassembling incoming packets are also significant for system performance. In this work, we address the challenge of payload pattern matching in intrusion detection systems. We present two efficient pattern matching techniques to analyze packet payloads at multi gigabit rates and detect hazardous contents. We expand on two approaches and evaluate them targeting the Snort IDS ruleset. The first one is Decoded CAM (DCAM) and uses pre-decoding to exploit pattern similarities and reduce the area cost of the designs. We improve DCAM and decrease the required logic resources by partially matching long patterns. The improved approach is denoted as decoded partial CAM (DpCAM). The second approach perfect hashing memory (PHmem), combines logic and memory for the matching. PHmem utilizes a new perfect hashing technique to hash the incoming data and determine a unique memory location of a possible matching pattern. Subsequently, we read this pattern from memory and compare it against the incoming data. We extend the perfect hashing algorithm in order to guarantee that for any given set a perfect hash function can be generated, and present a theoretical proof of its correctness.

II. LITERATURE SURVEY:

Intrusion detection, in the general sense, identifies anomalous, inappropriate, or incorrect access to a system. There has been much work on dinning the types of intrusions, distinguishing an intrusion from normal activity, and prototyping various intrusion systems. A high-level view of the components necessary to assemble an intrusion system is shown in Figure 2.1. At the center of the system is a component that detects intrusions. Four elements surround the detector that send and receive information. First, the detector has to know what events are classified as intrusions. When a new event occurs, the detector uses information about the current settings of the system as well as information about known intrusions to determine if this event is suspect. If the detector determines that the event is an intrusion, the event can be logged, a countermeasure can be taken, and an alarm can be raised. The potential countermeasures are represented as a database because multiple types of responses are available. An alarm could be signaled or the system could be modified to prevent similar events. When an alarm is triggered, an authority decides what further steps to take.

Feedback from the detector to the database of known intrusions indicates that the ideal detector can discover new intrusions. The event may be an abnormal event or it may be patterned after a similar known intrusion. An authority can be consulted to determine whether the event is deemed an intrusion or not. In the case of data networks, intrusion detection refers to the transfer of unwanted, malicious, or dangerous content over a network, and the system being monitored can be a web server, a database, or a cluster of computers. The intrusion may be as benign as spam or as harmful as a Trojan horse that infects a computer system by reading, writing, or even deleting files.

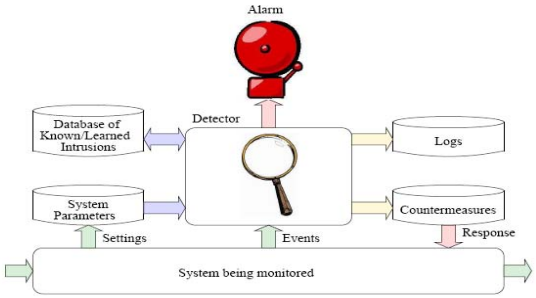


Figure 2.1: An intrusion system

III.THE NEED FOR INTRUSION DETECTION:

According to a recent study by the Computer Security Institute (CSI) and the Federal Bureau of Investigation (FBI), a staggering 70 percent of organizations surveyed reported a security incident. This figure is up from 42 percent reported in 1996. Taking into account organizations' reluctance to admit to incidents or their inability to detect them, the true figure is likely to be higher. E-business has driven organizations to open their networks to wider audiences over the Internet—home and mobile workers, business partners, suppliers, and customers—in order to stay competitive. But such open networks expose the organizations to *intrusions*—attempts to compromise, the confidentiality, integrity, or availability, or to bypass the security mechanisms of a computer system or network. *Intrusion detection* is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusion. But why is intrusion detection necessary? Is it not enough for an organization to use a firewall to control access to its network and maybe a virtual private network (VPN) to secure communications? Deploying firewalls and VPNs is a good thing. A robust firewall policy can minimize the exposure of many networks. Nevertheless, such countermeasures alone are not enough.

1 Attackers Are Getting Smarter
2 Vulnerabilities Are Proliferating
3 "Hacker" Tools Make Attacks Easier
4 Insider Attacks Are Still Predominant

IV.TECHNOLOGY ANALYSIS:

Intrusion detection allows organizations to protect their systems from the threats that come with increasing network connectivity and reliance on information systems. Given the level and nature of modern network security threats, the question for security professionals should not be whether to use intrusion detection, but which intrusion detection features and capabilities to use. IDSs have gained acceptance as a necessary addition to every organization's security infrastructure. When used conscientiously and knowledgeably, IDS products can provide worthwhile indications of malicious activity and spotlight security vulnerabilities, thus providing an additional layer of protection. Without them, network administrators have little chance of knowing about, much less assessing and responding to, malicious and invalid activity. Properly configured, IDSs are especially useful for monitoring the network perimeter for attacks originating from outside and for monitoring host systems for unacceptable insider activity.

2.4.2 Technology Leaders:

The IDS research field is still comparatively young, with most research dating from the 1980s and 1990s, and wide-scale commercial use from the mid-1990s. However, the intrusion-detection market has grown into a significant commercial presence. Gartner Research reported a 73 percent growth in the $153 million IDS software market in 2000. The leader by market share is Internet Security Systems (ISS) with 47 percent. The second largest is Computer Associates with 29 percent. Symantec and Network Associates also have IDS offerings, although they currently have little share and are seeing low growth.

Table 2.1 Leading IDS products

Table 5: Leading IDS Products	
Vendor/Product(s)	**Description**
Cisco Systems, Inc. • Secure IDS (formerly NetRanger) • IDS Host Sensor (Internet: www.cisco.com)	Secure IDS is a NIDS appliance that can also be implemented as software on Cisco's PIX Firewall and Internet Operating System (IOS) routers and as a "blade" in Cisco's switches. Cisco has a strong position in the IDS market because of its presence in the network infrastructure of a majority of organizations. Cisco is winning customers for its IDS, in part through competitive pricing and increased performance. An announced partnership to rebrand and resell Entercept Security Technologies' HIDS gives Cisco a comprehensive IDS solution. The challenge for Cisco will be to create a robust, easy-to-use, central management console that can correlate and report on incidents from host-based and network-based agents.

IV.A.NETWORK INTRUSION DETECTION SYSTEMS (NIDS):

In recent years, Network Intrusion Detection/ Prevention Systems (NIDSs/NIPSs) are more and more necessary for network security. Normally, traditional firewalls only examine packet headers to determine whether to block or pass the packets. Due to busy network traffic and smart attacking schemes, firewalls are not as effective as they used to be. NIDSs/NIPSs are designed to examine not only the headers but also the payload of the packets to match and identify intrusions. Intrusion detection systems can run in one of several modes: intrusion detection or inline NIDS. In intrusion detection mode, the NIDSs monitor the traffic offline and draw the attention of network administrator to suspicious activities by sending alerts. Inline intrusion detection system or Intrusion Prevention System (IPS) actively filters exploits from traffic in real time. It can forge resets, drop packets, or modify the packets in transit to defeat an attack. IPSs have to be extremely fast and reliable to process packets in real-time and should be completely transparent, so there is no need to change the network configuration.

The NIDSs can be further segmented into one of two techniques: anomaly detection or misuse detection (signature based). Anomaly detection is based on searching for discrepancies from the models of normal behavior. These models are obtained by performing a statistical analysis on the history of system calls or by using rule 9 based approaches to specify behavior patterns. Signature based detection is based on searching packets for attack signatures. It is much faster than anomaly detection, but can detect only those attacks that already have signatures. On the other hand, anomaly detection have the advantage of being able to detect previously unknown attacks, however it suffers from a large number of false positives.
There are many signature based NIDSs that require deep packet inspections such as SNORT, Bro. These systems are all open source systems, which allow us to perform a detailed analysis and show their abilities and constraints. Most modern NIDS/NIPSs apply a set of rules that lead to a decision regarding whether an activity is suspicious. They have well over a thousand rules. As the number of known attacks grows, the patterns for these attacks are made into signatures (pattern set). The simple rule structure allows flexibility and convenience in configuring NIDS. However, checking thousands of patterns to see whether it matches becomes a computationally intensive task as the highest network speed increases to several gigabits per second (Gbps). Current high-performance systems can barely process that many rules on a 100 Mbps moderately loaded network. To handle fully loaded gigabit networks, an NIDS must either drop some of the rules or drop some of the packets it analyzes. Neither solution is desirable since they both compromise security

IV.B. SNORT NIDS:

Snort is an open source NIDS that uses a portable library called libcap. Libcap allows the program to examine the network packet for its length, content, and header. Snort can perform traffic analysis, IP packet logging, protocol analysis, and payload content search. Furthermore, Snort can be configured to detect a variety of abnormal packet behaviors, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and OS fingerprinting attempts.

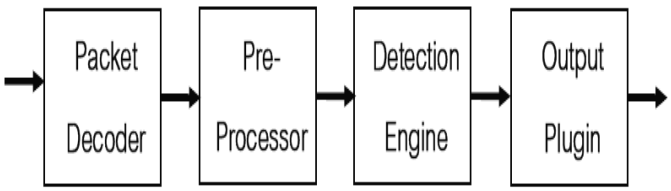


Figure 2.2 illustrates the Snort architecture.

It consists of the following components. When a network packet goes into the system, it is passed to the decoder component. Here the link level information, such as the Ethernet packet header, is removed. Then, the packet enters the pre-processor block, which performs a couple of functions such as packet defragmentation and reassembles the TCP stream, manipulate or examine packets prior to forwarding them to the detection engine. Finally and most importantly, the detection engine performs tests on the packet data forwarded by the preprocessors, using the Snort rules and signatures as a baseline. If suspicious activity is identified by the detection engine, output plug-ins are called to generate administrative alerts, e.g., “drop this packet”, or “log this packet”.

IV.C. PATTERN MATCHING IN SOFTWARE NIDS SOLUTIONS:

At the core of every intrusion detection system is a pattern matching algorithm. From a stream of packets, the algorithm identifies those packets that contain data matching the signatures of a known attack. The intrusion detection system then takes action that could vary from alerting the system administrator to dropping the packet in 12 the case of inline NIDS. The problem of pattern matching is well researched, many algorithms exist and they can be classified into either single pattern string matching or multiple pattern string matching. In single pattern string matching the packet is searched for a single pattern at a time. On the other hand, in multiple pattern string matching the algorithm searches the packet for the set of patterns all at once. Several string pattern matching algorithms have been recently proposed in NIDS especially for SNORT’s open source NIDS. Recently, new pattern matching algorithms are proposed to boost the pattern matching speed of SNORT. For example, the Aho-Corasick-Boyer-Moore (AC_BM) algorithm proposed by Silicon Defense combines the Boyer-Moore and Aho- Corasick algorithms. Another algorithm named Wu-Mander multi-pattern matching (MWM) algorithm. The MWM algorithm improves the Boyer-Moore algorithm by performing a hash on 2-character prefix of the input data, to index into a group of patterns. The MWM algorithm is the default engine of the Snort when the search-set size exceeds 10. When the Snort uses the MWM algorithm, the matching speed becomes much faster than when using the AC and other Boyer-Moore like algorithms.

Finally, Markatos et al. proposed E2xB algorithm, which provides quick negatives when the search pattern does not exist in the incoming data. Compared to Fisk et al., E2xB is

faster, while for large incoming packets and less than 1k-2k rules it outperforms MWM. These algorithms greatly improve SNORT's pattern matching speed to a few hundred Mbps at most, e.g., 50Mbps with the Pentium IV, 250Mbps with the SUN SDA. However, it is still below the line rate needed for network deployment.

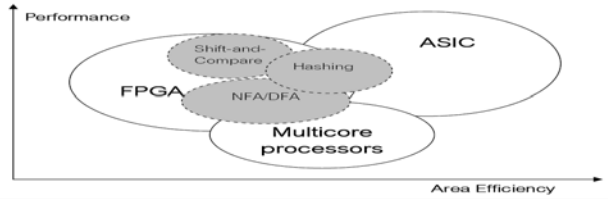


Figure 2.3: Abstract illustration of performance and area efficiency for various hardware pattern matching techniques

IV.D.HARDWARE-BASED PATTERN MATCHING ARCHITECTURES IN NIDS:

Given the processing bandwidth limitations of General purpose processors (GPP), which can serve only a few hundred Mbps throughput, Hardware-based NIDS (Multicore Processors, ASIC or FPGA) as illustrated in Fig. 2.3 is an attractive alternative solution.

ASIC Technique:

Many ASIC intrusion detection systems have been commercially developed. Such systems usually store their rules using large memory blocks, and examine incoming packets in integrated processing engines. In academic research, there are several pattern matching solutions designed for ASIC.

Multi-core processors Technique:

Recently, multi-core processors' implementations are becoming popular for designing NIDS due to flexibility. Different from the traditional single core processors, multi-core processes combine two or more independent processors into a single package. These independent processors can run in parallel hence can provide higher computation power. IBM cell processor has 8 synergistic processor elements, each with 128 KB local memory.

FPGA Technique:

On the other hand, FPGAs are more suitable, because they are reconfigurable; they provide hardware speed and exploit parallelism. An FPGA-based system can be entirely changed with only the reconfiguration overhead, by just keeping the interface constant. Next subsections present some main approaches for hardware based systems in academic researches. Most of them are implemented on FPGA platform.

CAMs & Shift-and-compare:

An easy approach for pattern matching is to use Content Addressable Memories (CAMs) or shift-and-compare. They apply parallel comparators and deep pipelining on different, partially overlapping, positions in the incoming packet. Current FPGAs give designers the opportunity to use integrated block RAMs for constructing regular CAM. Other researchers preferred to use shift-and compare, which leads to designs that operate at higher frequency. Shift-and-compare architecture uses one or more comparators for every matching pattern. Generally, this approach uses FPGA logic cells to store each pattern. Every LUT can store a half-byte (4-bit) of a pattern, and the flip-flops that already exist in logic cells can be used to create a pipeline, without any overhead. The simplicity of the parallel architecture can achieve high throughput when compared to software approaches. The drawback of these methods is the high area cost. To decrease the area cost and achieve a high clock rate, many improvements are proposed.

Nondeterministic/Deterministic Finite Automata:

An alternative approach exploits state machines. The state machines can be implemented on hardware platform to work all together in parallel. There are two main options for implementations of state machines. The first one is using Nondeterministic Finite Automata (NFAs), having multiple active states at a single cycle, while the second is Deterministic Finite automata (DFAs) which allow one active state at a time and result in a potentially larger number of states compared to NFAs. State machines produce designs with low cost, but at a modest throughput. Theoretically, DFA can be exponentially larger than NFA, but in practice often DFAs have, as compared to NFAs, a similar number of states. Sidhu and Prassanna introduced regular expressions and Nondeterministic Finite Automata (NFAs) for finding matches to a given regular expression. Their automata matched one text character per clock cycle. In general, finite automata machines suffer scalability problems. They are complex and hard to implement. Too many states consume too many hardware resources. Every time a new attack is characterized and a signature is added to the database the FA have to be rebuilt again and it requires long reconfiguration time.

V.DECODED CAMs

In the past few years, numerous hardware-based pattern matching solutions have been proposed, most of them using FPGAs, finite automata or hashing approaches. Next, we describe some significant steps forward in IDS pattern matching over the past few years. Simple CAM or discrete comparators structures offer high performance, at high area cost. Using regular expressions (NFAs and DFAs) for pattern matching slightly reduces the area requirements, however, results in significantly lower performance. A technique to substantially increase sharing of character comparators and reduce the design cost is predecoding, applicable to both regular expression and CAM-like approaches. The main idea is that incoming characters are predecoded resulting in each unique character being represented by a single wire. This way, an –character comparator is reduced to an -input AND gate. Yusuf and Luk presented a tree-based CAM structure, representing multiple patterns as a Boolean expression in the form of a binary decision diagram (BDD). In doing so, the area cost is lower than other CAM and NFA approaches.

V.1 BASIC DISCRETE COMPARATOR & COMMON CHARACTER COMPARATOR:

Simple CAM or discrete comparators may provide high performance; however, they are not scalable due to their high area cost. We assumed the simple organization depicted in Figure 3.1(a). The input stream is inserted in a shift register, and the individual entries are fanned out to the pattern comparators. There is one comparator for each pattern, fed from the shift register. This design is simple and regular, and with proper use of pipelining, the circuit can be fast. Its drawback, however, is the

high area cost. To remedy this cost, we suggested *sharing* the character comparators exploiting similarities between patterns

V.2 DECODED CAM:

The Decoded CAM architecture illustrated in Figure 3.2, builds on this idea extending it further by the following observation: instead of keeping a window of input characters in the shift register each of which is compared against multiple search patterns, we can first test for equality of the input for the desired characters, and then delay the partial matching signals. This approach both shares the equality logic for character comparators and replaces the 8-bit wide shift registers used in our initial approach with single bit shift registers for the equality result(s).

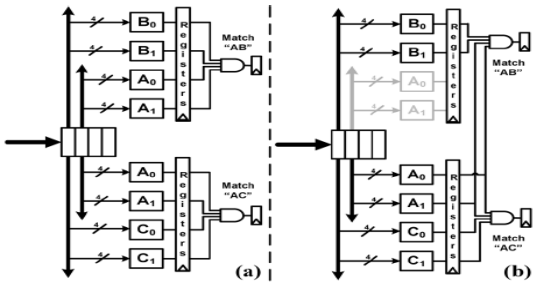


Figure 3.1. Basic discrete comparator structure and its optimized version which shares common character comparators.

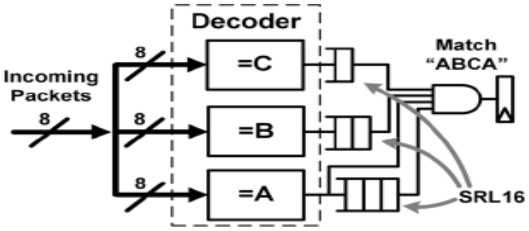


Figure3.2. Decoded CAM: Three comparators provide the equality signals for characters A, B, and C ("A" is shared). To match pattern "ABCA" we have to remember (using shift registers) the matching of character A, B, C, for 3, 2, and 1 cycles, respectively, until the final character is matched.

3.3 DPCAM:

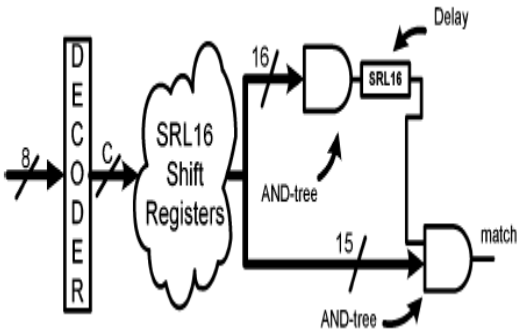


Figure 3.3. DpCAM: Partial matching of long patterns. In this example, a 31-byte pattern is matched. The first 16 bytes are partially matched and the result is properly delayed to feed the second substring comparator. Both substring comparators are fed from the same pool of shifted decoded characters (SRL16s) and therefore sharing of decoded characters is higher.

Long patterns are partially matched in substrings of maximally 16 characters long. The reason is that the AND-tree of a 16 character substring needs only five LUTs, while only a single SRL16 shift register is required to delay each decoded input character. Consequently, a pattern longer than 16 characters is partitioned in smaller substrings which are matched separately. The partial match of each substring is properly delayed and provides input to the AND-tree of the next substring. This way all the substring comparators need decoded characters delayed for no more than 15 cycles.

VI.PERFECT HASHING MEMORY (PHMEM): The alternative pattern matching approach proposed in this paper is the PHmem. Instead of matching each pattern separately, it is more efficient to utilize a hash module to determine which pattern is a possible match, read this pattern from a memory and compare it against the incoming data. Hardware hashing for pattern matching is a technique known for decades.

VI.2 PERFECT HASHING TREE:

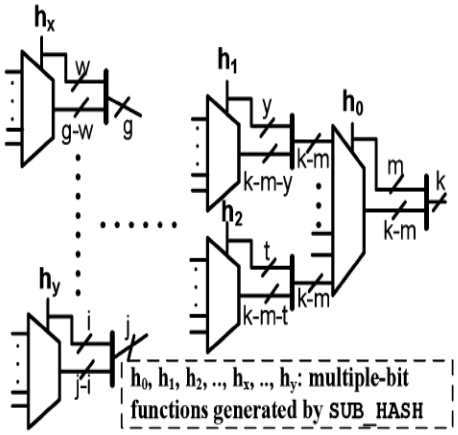
The proposed scheme requires the hash function to generate a different address for each pattern, in other words, requires a perfect hash function which has no collisions for a given set of patterns. Furthermore, the address space would preferably be minimal and equal to the number of patterns. Instead of matching unique pattern prefixes as in, we hash unique substrings in order to distinguish the patterns. To do so, we introduce a perfect hashing method to guarantee that no collisions will occur for a given set.

Generating such a perfect hash function may be difficult and time consuming. In our approach, instead of searching for a single hash function, we search for multiple simpler sub hashes that when put together in a tree-like structure will construct a perfect hash function. The perfect hash tree is created based on the idea of "divide and conquer." Let **A** be a set of unique\ substrings = $\{a_1, a_2 \ldots a_n\}$ and **H (A)** a perfect hash function of **A**, then the perfect hash tree is created according to the following equations:

$$H(A) = h_0(H_1(1^{st} \text{ half of } A), H_2(2^{nd} \text{halfof } A)) \quad (1)$$

$$H_1(1^{st} \text{ half of } A) = h_1(H_{1.1}(1^{st} \text{ quarter of } A), H_{1.2}(2^{nd} \text{ quarter of } A)) \quad (2)$$

and so on for the smaller subsets of the set **A** (until each subset contains a single element). The h_0, h_1 etc., are functions that combine subhashes. The H_1, H_2, $H_{1.1}$, $H_{1.2}$ etc., are perfect hashes of subsets (subhashes).



(b)

To prove that our method generates perfect hash functions, we need to prove the following.

- For any given set **A** of **n** items that can be encoded in $\log_2(n)$ bits, our method generates a function $h:A\rightarrow\{0,1\}$ to split the set in two subsets that can be encoded in $\log_2(n/2)$ bits (that is $\log_2(n) - 1$ bits).
- Based on the first proof, the proposed scheme outputs a perfect hash function for the initial set of patterns.

Proof: By definition, a hash function $H_{|A}$ of set $A = \{a_1, a_2 \ldots a_n\}$ which outputs a different value for each element a_i is perfect

$$H_{|a1} \neq H_{|a2} \neq \ldots H_{|ax}. \quad (3)$$

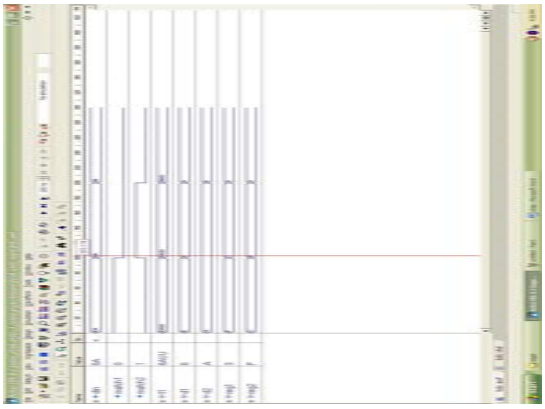
Also, if $h_{|S}$, where $S = A \cup B \cup \ldots \cup N$ and $A \cap B \cap \ldots \cap N = V$ is a hash function that separates the **n** subsets A,B,…N having a different output for elements of different subsets is also perfect, that is

$$h_{|A} \neq h_{|B} \neq \ldots h_{|N}. \quad (4)$$

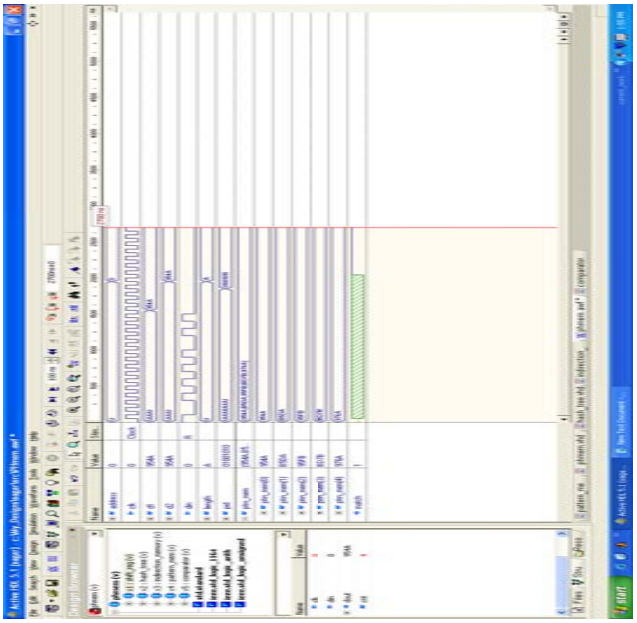
We construct our hash trees based on two facts. First, the "selects" of the multiplexers **h** separate perfectly the subsets of the node. Second, that the inputs of the leaf nodes are perfect hash functions; this is given by the fact that each element differs to any other element at least one bit, therefore, there exists a single bit that separates (perfectly) any pair of elements in the set. Consequently, it must be proven that a node which combines the outputs of perfect hash functions $H_A, H_B, \ldots H_N$ of the subsets **A,B,…..N** using a perfect hash function $h_{|S}$ which separates these subsets, outputs also a perfect hash function H_{node} for the entire set **S**.

VII.Experimental RESULTs & ANALYSIS

Several different architectures are simulated by active-HDL and synthesized in Xilinx. Implementation of simulated and synthesis results of BDC, CCC, DCAM, DpCAM and PHmem structures are shown below.

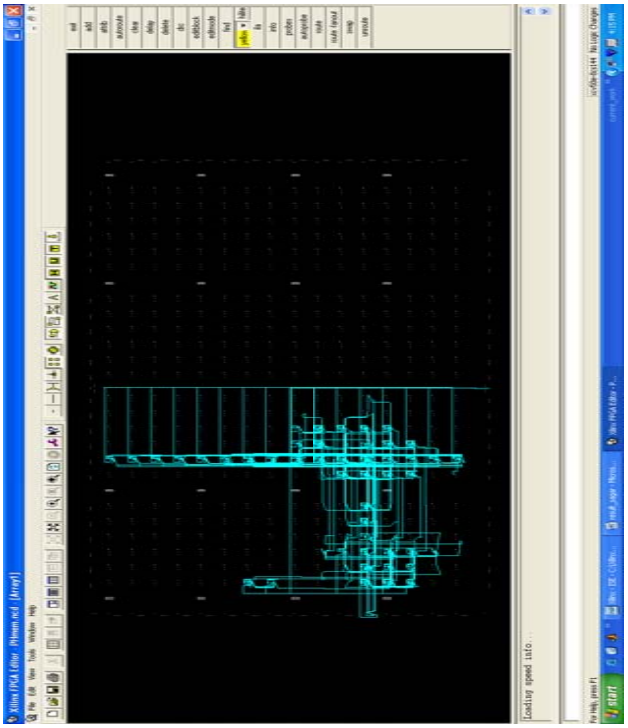


This simulated result screen is for the BDC structure.



This simulated result screen is for the DPCAM structure

FPGA module of PHMEM: The below screen is how the structure of PHMEM will be in the FPGA module.



REFERENCES:[1] I. Sourdis and D. Pnevmatikatos, "Fast, large-scale string match for a 10 Gbps FPGA-based network intrusion detection system," in Proc. Int. Conf. Field Program. Logic Appl,2003.

[2] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching," in Proc. IEEE Symp. Field-Program. Custom Comput. Mach., 2004.

[3] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, "Granidt: Towards gigabit rate network intrusion detection technology," in Proc. Int. Conf. Field Program. Logic Appl., 2002.

[4] Z. K. Baker and V. K. Prasanna, "A methodology for synthesis of efficient intrusion detection systems on FPGAs," in Proc. IEEE Symp. Field-Program. Custom Comput. Mach., 2004, pp. 135–144.

[5] C. R. Clark and D. E. Schimmel, "Scalable parallel pattern-matching on high-speed networks," in Proc. IEEE Symp. Field-Program. Custom Comput. Mach., 2004, pp. 249–257.

[6]Y. H. Cho, S. Navab, and W. Mangione-Smith, "Specialized hardware for deep network packet filtering," in Proc. 12th Int. Conf. Field Program.Logic Appl., 2002, pp. 452–461.

[7] Y. H. Cho and W. H. Mangione-Smith, "Deep packet filter with dedicated logic and read only memories," in Proc. IEEE Symp. Field-Program. Custom Comput. Mach., 2004.

[8] Z. K. Baker and V. K. Prasanna, "Automatic synthesis of efficient intrusion detection systems on FPGAs," in Proc. 14th Int. Conf. Field Program. Logic Appl., 2004, pp. 311–321.

[9] M. Attig, S. Dharmapurikar, and J. Lockwood, "Implementation results of bloom filters for string matching," in Proc. IEEE Symp. Field-Program. Custom Comput. Mach., 2004.

[10] Y. H. Cho and W. H. Mangione-Smith, "Programmable hardware for deep packet filtering on a large signature set," in Proc. Conf. Interaction Between Arch., Circuits, Compilers, 2004.

[11] L. Tan and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," in Proc. 32nd Int. Symp. Comput. Arch. (ISCA), 2005, pp. 112–122.

[12] P. Krishnamurthy, J. Buhler, R. D. Chamberlain, M. A. Franklin, K. Gyang, and J. Lancaster, "Biosequence similarity search on the mercury system," in Proc. 15th IEEE Int. Conf. Appl.-Specific Syst., Arch., Processors (ASAP), 2004, pp. 365–375.

[13] E. Sotiriadis, C. Kozanitis, and A. Dollas, "FPGA based architecture for DNA sequence comparison and database search," presented at the 13th Reconfigurable Arch. Workshop (RAW), Rodos, Greece, 2006.

[14] SNORT, "SNORT official website," 2007. [Online]. Available: http:// www.snort.org

[15] M. Fisk and G. Varghese, "An analysis of fast string matching applied to content-based forwarding and intrusion detection," Univ. California, Tech. Rep. CS2001-0670, 2002.

[16] Z. K. Baker and V. K. Prasanna, "Automatic synthesis of efficient intrusion detection systems on FPGAs," IEEE Trans. Dependable Sec. Comput., vol. 3, no. 4, Oct. 2006.

[17] J. Singaraju, L. Bu, and J. A. Chandy, "A signature match processor architecture for network intrusion detection," in Proc. IEEE Symp. Field- Program. Mach., 2005, pp. 235–242.

[18] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, "Implementation of a content-scanning module for an internet firewall," in Proc. IEEE Symp. Field-Program. Mach., 2003.

[19] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel Bloom filters," IEEE Micro, vol. 24, no. 1, Jan. 2004, pp. 52–61.

[20] G. Papadopoulos and D. Pnevmatikatos, "Hashing + Memory = Low Cost, exact pattern matching," in Proc. Int. Conf. Field Program. Logic Appl., 2005, pp. 39–44.

[21] S. Yusuf and W. Luk, “Bitwise optimized CAM for network intrusion detection systems,” in Proc. Int. Conf. Field Program. Logic Appl., 2005, pp. 444–449.

[22] H.-J. Jung, Z. K. Baker, and V. K. Prasanna, “Performance of FPGA implementation of bit-split architecture for intrusion detection systems,” presented at the Reconfigurable Arch. Workshop IPDPS (RAW), Rodos, Greece, 2006.

[23] Xilinx, San Jose, CA, “VirtexE, Virtex2, Virtex2Pro, and Spartan3 datasheets,” 2006. [Online]. Available: http://www.xilinx.com

[24] F. J. Burkowski, “A hardware hashing scheme in the design of a multiterm string comparator,” *IEEE Trans. Comput.*, vol. 31, no. 9, Sep.1982, pp.825–834.

[25] R. C. Merkle, “Protocols for public key cryptosystems,” in *Proc. IEEE Symp. Security Privacy*, 1980, pp. 122–134.

[26] T. Sproull, G. Brebner, and C. Neely, “Mutable codesign for embedded protocol processing,” in Proc. Int. Conf. Field Program. Logic Appl., 2005, pp. 51–56.