Design and Implementation of an Anomaly-based Network Intrusion Detection System Utilizing the DNA Model

RIHAM MAHDY, MAGDY SAEB
Computer Engineering Department
Arab Academy for Science, Technology & Maritime Transport,
School of Engineering, Alexandria
EGYPT

Abstract: - The genetic material that encodes the unique characteristics of each individual such as gender, eye color, and other human features is the well-known DNA. In this work, we introduce an anomaly intrusion detection system, built on the notion of a DNA sequence or gene, which is responsible for the normal network traffic patterns. Subsequently, the system detects suspicious activities by searching the "normal behavior DNA sequence" through string matching. On the other hand, string matching is a computationally intensive task and can be converted into a potential bottleneck without high-speed processing. Furthermore, conventional software-implemented string matching algorithms have not kept pace with the ever increasing network speeds. As a result, we adopt a monitoring phase that is hardware-implemented with the intention that DNA pattern matching is performed at wire-speed. Finally, we provide the details of our FPGA implementation of the bioinformatics-based string matching technique.

Key-Words: - FPGA, anomaly identification, Network Intrusion Detection, DNA computing, pattern matching, bioinformatics.

1 Introduction

The majority of Network Intrusion Detection Systems (NIDS) inspect the network packet payload to check for predefined signature strings starting at an arbitrary location. This process is often referred to as "signature-based deep packet inspection" NID. Other systems construct a model for the normal behavior of the Network and then flag any activity that deviates from this model as a suspicious behavior or anomaly. This process is called "anomaly-based" NID. Some of the previous work in anomaly identification was network-based, such as in [12], which proposed a system that learns the normal range of values for 33 fields of the Ethernet, IP, TCP, UDP, and ICMP protocols. Afterwards, the system checks incoming packets for deviations from this range. Other approaches, called host-based such as in [3], build a user signature by encoding user's command sequences or normal behavior in a DNA strand. Then the system uses a unique variation of Smith-Waterman pair-wise alignment algorithm to compare user's current session with the pre-produced signature. Others, such as in [5], profile processes according to system calls emitted by processes in response to a certain input. Subsequently, it proposes a behavioral distance as a means to detect an attack on one process that causes its behavior to deviate from that of another. The most computationally extensive part in anomaly identification is pattern-matching. Accordingly, previous work in hardware exact pattern-matching architecture was proposed such as in [2]. In this work, a new Content Addressable Memory-based (CAM) architecture, able to match variable-length patterns with fixed-length keywords or "Snort keywords", is suggested. This was achieved at a rate of one character per clock cycle. Matching time is a function of the length of the pattern. Consequently, Nilsen et al. [13] introduced another CAM-based architecture in which the length of each word is independent from the others, in contrast to common CAMs where all words have the same length. In this exertion we present a framework for a bioinformatics-based network anomaly identification system. The system works in two phases. In the first phase, we build the DNA sequence responsible for the normal network traffic rates. A program is developed to capture incoming packets for a predetermined time to construct the basic building structure or codon of the DNA sequence. Once the computer DNA sequence has been created, the second phase sends the new base structures generated by real time network activity to the FPGA pattern matching module. In this module, new base structures are compared to existing structures contained within the DNA sequence. We will show that the proposed system can handle fixed-length patterns at a rate of one character per cycle. Moreover, the system, with a slight modification, can handle variable-length patterns.

In the next few sections we discuss the algorithm, the implementation details of the software application, and the building blocks of the proposed hardware module along with details of its operation. The following sections include background, methodology, and the simulation and implementation results. Finally, we present some statistics after using the 1999 DARPA intrusion detection dataset for training and testing the system [4], and a summary and our conclusions.

2 Background

The process that was summarized in the introduction section is shown below in Figure 1. In the next few lines, we provide the necessary background in order to explain our approach.

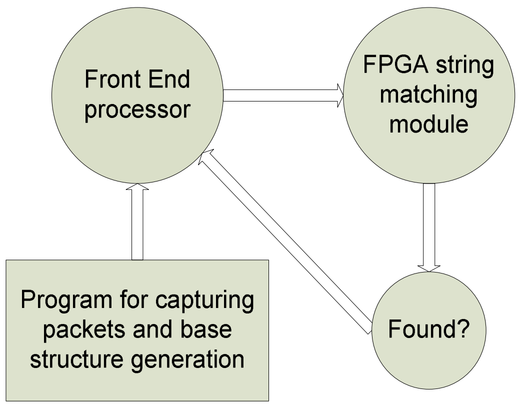

Front End processor
FPGA string matching module
Program for capturing packets and base structure generation
Found?

Figure 1. **The block diagram of the system**

Deoxyribonucleic Acid (DNA) is made of two strands of complementary pairs of nucleotides. Each strand is made of different sequences of four bases (nucleotides) Adenine, Guanine, Cytosine, and Thymine. DNA has tremendous information storage capacity. For example, only 1 gram of DNA contains as much information as 1 trillion CD's [6]. In addition, any DNA sequence can be synthesized in any desirable length. Genes are variable-length combinations of fixed-length combinations or codons of these bases. Different combinations of codons generate different genes. Therefore, unique physical characteristics such as gender, hair color, and height are all encoded in the human gene. Recent research, The Human Genome Project, has shown that certain genes and mutations to other genes are responsible for undesired characteristics and certain conditions negatively affecting human health. These genes are considered DNA anomalies. It has been proven that some external factors can lead to mutations of human genes resulting in diseases such as X-ray and radioactive materials [8].

Yu et al., [16] have proposed that every computer system can be given a DNA characterization that contains all sequences or genes responsible for the important characteristics of a specific monitored system. These characteristics include network traffic, modification of system files, sequence of system calls, and user behavior. In this work, we propose and build a DNA sequence responsible for the monitored system normal network traffic.

Prior to generating the DNA sequence for the network traffic, the structure of the fundamental base must first be determined.
As shown in Figure 2, the proposed base structure is constructed of five different categories of information that are collected for a predetermined time.

Average packet length	Number of TCP packets	Number of UDP packets	Number of ICMP packets	Number of IGMP packets

Figure 2. The Fundamental base structure

Similar to humans, the initial DNA sequences are predetermined on inception and ideally do not cause the system any harm. However, mutations to the DNA are possible, resulting in abnormal behavior [9]. Hu proposed [7] that malicious software or users intruding upon a computer system can have the capability of "mutating a computer's normal characteristics" thereby causing various degrees of damage. However, by recognizing intrusions early enough, the damage can be limited and more rapid recovery is possible. Consequently, the development of an effective intrusion detection system can serve as a deterrent to intrusions as well as prevent hard-to-repair or extensive permanent damage.

3 Methodology

In the following few lines, we provide a description of the proposed system operating steps. The proposed system has two parts:

Part I: Network traffic DNA sequence generation or

the offline training phase.
Part II: Network monitoring or the online monitoring phase.

Part I (TRAINING PHASE)
The aim of this phase is to collect raw data for a certain predefined time that is called the Training Time (T1) in order to produce a number of observations or base structures. The time of each observation is (T2). It is imperative that close structures are not repeated. Therefore, each field in the base structure is rounded to a certain threshold (Th).
Input: T1[training time], T2 [time of one observation], Th[threshold value]
Output: Txt file containing DNA sequence of Network traffic behavior
Algorithm Body:

```
Num of packets=0
Packet len =0
TCP num =0
UDP num =0
ICMP num =0
IGMP num=0
DNA file= file.clear()
for(T1)
    for (T2)
        for each(packet with destination address = to host m/c)
            Num of packets++
            ip= ip header
            packet len = packet len+ ip.total_len
            switch(ip.protocol)
                case 'tcp': TCP num++
                case 'udp': UDP num++
                case 'icmp': ICMP num++
                case 'igmp': IGMP num++
            end switch;
     end for;
    avg packet len = packet len / num of packets
    Round(avg packet,TCP num,UDP num,ICMP num,IGMP num, Th)
    Codon(avg packet,TCP num,UDP num,ICMP num,IGMP num)

        if (DNA file does not contain codon) then
            Add codon to DNA file
        else
            Codon freq++
end for;
```

Part II (MONITORING PHASE)
The aim of this phase is to monitor the rate of traffic and produce codons (observations) and search the DNA file for it, if it does not exist then this behaviour is considered an anomaly (malicious behavior)
Input: T2 [time of one observation] same as in phase I, Th[threshold value] same as in phase I,
Output: codon found or not

```
for (T2)
        for each(packet with destination address = to host m/c)
            Num of packets++
            ip= ip header
            packet len = packet len+ ip.total_len
            switch(ip.protocol)
                case 'tcp': TCP num++
                case 'udp': UDP num++
                case 'icmp': ICMP num++
                case 'igmp': IGMP num++
            end switch;
     end for;
     avg packet len = packet len / num of packets
     Round(avg packet,TCP num,UDP num,ICMP num,IGMP num, Th)
    Codon(avg packet,TCP num,UDP num,ICMP num,IGMP num)
            Send codon to the hardware module for wire speed DNA search
                if (found) then
                    Repeat monitoring
                else
                    It is an attack
                end if
End algorithm.
```

4 Implementation

In this section we describe the software implementation and the hardware search module.

4.1. DNA sequence-producing Software

The software module was developed using Visual Studio .NET EDK and encoded in C# language. It is composed of an observation part that takes the time of one observation and the threshold value and produces the DNA sequence. Later on, this sequence is downloaded onto the CAM-based hardware module for the monitoring phase. The program was developed and tested on a 1.8-GHz Pentium-based machine with 1024 MB memory.

4.2. The hardware monitoring module

The hardware module is used for performing the DNA pattern matching at the monitoring phase. Emulating physical DNA pattern matching that exploits a great level of parallelism O (1), associative memory modules are used to buffer the generated DNA. Each digit in the generated structure field is mapped to two nucleotides or base and each base is represented by two bits. Based on the training data, a five-digit limit was placed for each field in the base structure in order to produce a fixed size codon of 100 bits.

The micro-architecture, along with its operation details, is described using a finite state machine approach (FSM) as shown in Figure 3. The machine operation takes place through four basic states. These are summarized as follows:

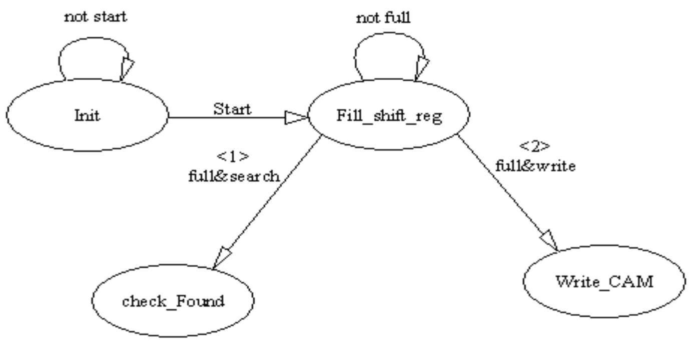


Figure 3. The finite state machine of the micro-architecture (The FPGA string matching module shown in Fig. 1)

The initial state "Init" holds back the execution of the successive states until the "start" signal is triggered and furthermore resets all hardware modules. In the following state "Fill_shift_reg", the 100-bit codon is buffered in a 100-bit shift register that shifts 10 bits each clock cycle. A "full" signal indicates that the 100-bit codon is available in the shift register. A "write" signal transfers execution to "write_CAM" state during which the codon is a part of the normal DNA sequence and hence is buffered in one of the associative memory modules according to the module addressing signal. A "search" signal transfers execution to "check-found" state during which the codon is a new observation that must be checked to ensure that it a normal DNA sequence. The Content-Addressable Memory (CAM) modules are all searched in parallel and a "found" signal indicates if it is an attack or not. Figure 4 shows the top level schematic diagram of the micro-architecture. To handle variable-length patterns, this CAM module can be further subdivided into ten ten-bit modules. The control unit has to accommodate this modification in order to signal end-of-codon.

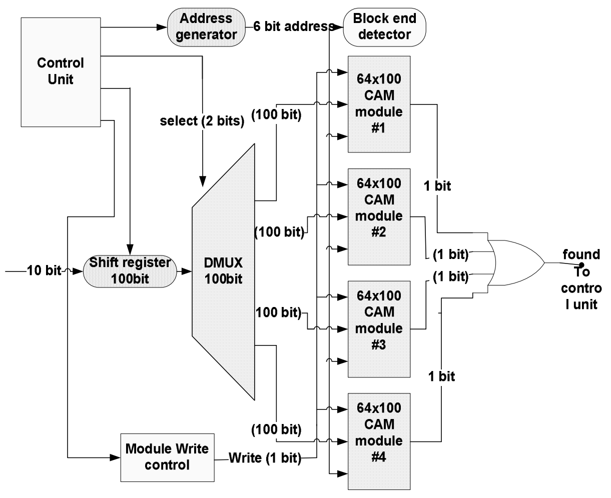


Figure 4: The block diagram of the micro-architecture

5 Software Testing

We performed the system evaluation on the "1999 DARPA off-line IDS evaluation data set" [11]. The set consists of network traffic profiles (*tcpdump* files) that are collected at two points from a simulated local network of an imaginary air force base over a five week period. The simulated system consists of four real "victim" machines running SunOS, Solaris, Linux, and Windows NT, a Cisco router, and a simulation of a local network with hundreds of other hosts and thousands of users and an Internet connection without a firewall. We trained the system on five days of attack-free network traffic (week 1) collected from a sniffer between the router and victim machines (inside.tcpdump). Then it was tested on five days of traffic from the same point (week 4) during which time there were 183 attacks. The test set includes a list of all attacks, labeled with the victim IP address, time, type, and a brief description. We chose a single victim machine (172.16.112.50) and filtered all the packets destined for this machine to build its DNA sequence as it has experienced most of the denial-of-service attacks simulated in week 4.

5.1. Testing results

The dataset contains three types of attacks; remote-to-local, user-to root, and denial-of-service [10]. Our system builds a model for the normal network traffic pattern. Accordingly, we can focus mainly on the denial-of-service attacks. Figure 5 shows the DNA sequence graph for the chosen machine that contains 592 codons (X-axis) and the values for each field in each codon (Y-axis). Codons in this graph are generated with observation time that is equal to ten seconds. The threshold value is equal to ten and repetitions were omitted.

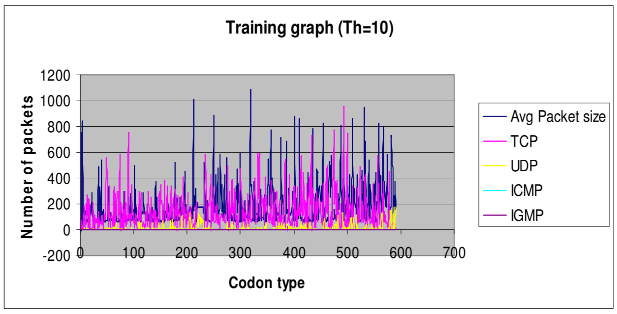


Figure 5. DNA sequence generated from week 1 training (Th=10)

Figure 6 shows codons generated with observation time equals 10 seconds, threshold value equals 1 and repetitions omitted, as the obvious number of codons is nearly six times more because similar sequences are not treated as equal.

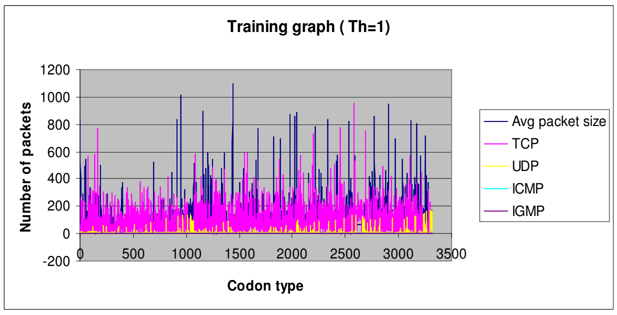


Figure 6. DNA sequence generated from week 1 training (Th=1)

We identified 370 attacking codons at different times on Monday, week 4 from the supplied attack list destined for the selected machine. There were some attacks that lasted for 5 minutes generating up to 30 codons. The system discovered them from the first codon. When a threshold value of 10 was used, only 126 attacking codons were identified with false positive rate equal 7 per day. However, all attacking codons were identified with false positive equal to 453 when the threshold value is set to 1. The system can be adaptable to the level of security needed by assigning different threshold values. Figure 7 shows observations generated from Monday (inside.tcpdump) of week 4, where a *"smurf"* attack shows an abnormal ICMP behavior (codons 324 and 325). Thus, they are marked as attacking codons.

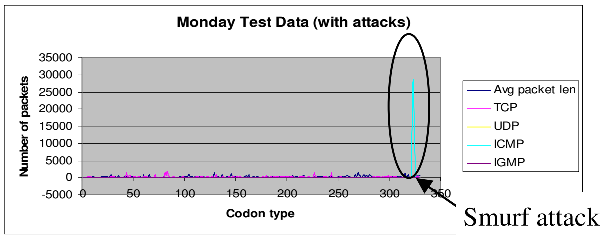
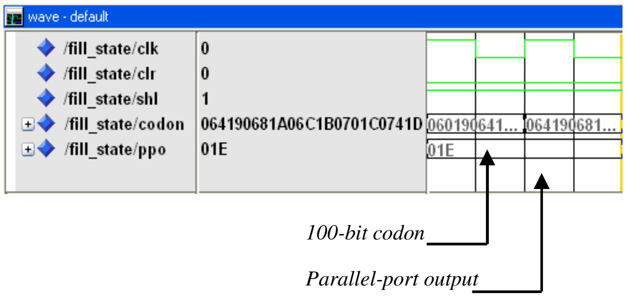


Figure 7. Codons generated from week 4 (Monday) testing data

6 Hardware Simulation

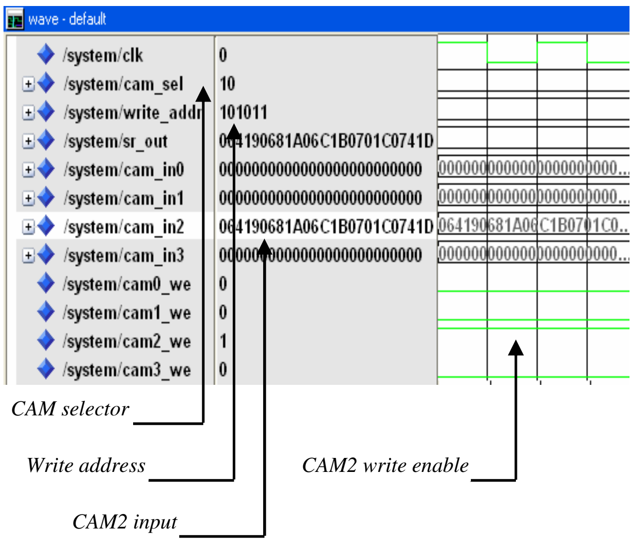
The simulation of the designed micro-architecture is performed on the Logic Simulator of Mentor Graphics Modelsim 6.1. The first operation performed is filling the 100-bit shift register with codons at a rate of 10 bits per clock cycle during the "Fill_shift_reg" state. Figure 8 depicts this operation. The input codon is in this case "064190681A06C1B0701C0741D" and the "Shl" signal is active so the parallel port output is left-shifted each clock cycle until the shift register is completely full after 10 clock cycles.



```
wave - default
/fill_state/clk      0
/fill_state/clr      0
/fill_state/shl      1
/fill_state/codon    064190681A06C1B0701C0741D   060190641...  064190681...
/fill_state/ppo      01E                         01E
```

Figure 8. Simulation of 100-bit codon loading.

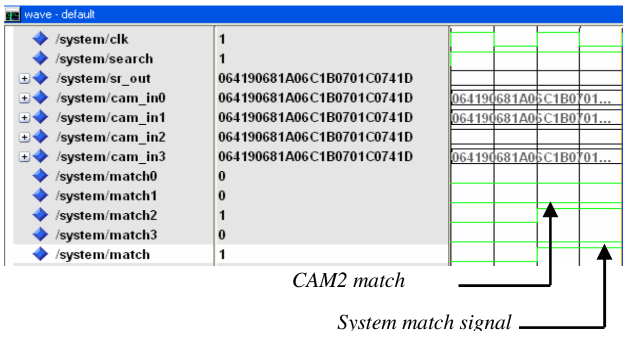
Buffering the 100-bit codon in one of the CAM is shown in Figure 9



```
wave - default
/system/clk          0
/system/cam_sel      10
/system/write_addr   101011
/system/sr_out       064190681A06C1B0701C0741D
/system/cam_in0      0000000000000000000000000   0000000000000000000000000...
/system/cam_in1      0000000000000000000000000   0000000000000000000000000...
/system/cam_in2      064190681A06C1B0701C0741D   064190681A06C1B0701C0..
/system/cam_in3      0000000000000000000000000   0000000000000000000000000...
/system/cam0_we      0
/system/cam1_we      0
/system/cam2_we      1
/system/cam3_we      0
```

Figure 9. Simulation of buffering 100-bit codon in a selected CAM module

modules "cam_sel" signal selects CAM2, "CAM2_we" activates the write operation of CAM2. Figure 10 provides the simulation result for the search operation. The "search" signal provides the codon to search for to all the CAM modules to be searched in parallel, "match2" signal is high indicating that this codon is found in CAM2 and so the whole system "match" signal is high too.



```
wave - default
/system/clk          1
/system/search       1
/system/sr_out       064190681A06C1B0701C0741D
/system/cam_in0      064190681A06C1B0701C0741D   064190681A06C1B0701...
/system/cam_in1      064190681A06C1B0701C0741D   064190681A06C1B0701...
/system/cam_in2      064190681A06C1B0701C0741D
/system/cam_in3      064190681A06C1B0701C0741D   064190681A06C1B0701...
/system/match0       0
/system/match1       0
/system/match2       1
/system/match3       0
/system/match        1
```

Figure 10. Simulation of searching for a codon in the four CAM modules

7 Implementation Results

We have used the Virtex IV FPGA family to implement our design [15]. Most of the hardware implemented string matching algorithms are variations of sequential algorithms. For example, KMP [1] or CAM architectures were developed for snort (IDS) rules [13] and [2]. However, our model is emulation to a single DNA strand pattern-matching process. It searches for fixed-length patterns or codons in variable-length combinations of these codons or Gene. As a result, comparison with other architectures is not practicable. In Appendix A, we provide the implementation results based on the consumed area in CLBs, where CLB is the abbreviation for "Configurable Logic Block". The FPGA is composed of hundreds of the configurable logic blocks that are programmed to perform any logic or sequential operation. In addition, we provide a summary of the implementation timing reports. Moreover, the design floor plan is also provided.

8 Summary and Conclusion

Network behavior anomaly identification is a new alternative in the existing IDS technology. Anomaly identification improves on signature detection by modeling the space of "use" instead of "misuse." It also eliminates the need for frequent signature updates and provides enhanced performance. In this work, we have introduced an anomaly identification system that encodes the network behavior in a DNA strand. The generated DNA sequence specifies the unique behavior of this monitored system. In order to perform packet inspection at wire-speed, we have proposed and implemented an FPGA-based micro-architecture employing associative memory modules. The special features of this (IDS) can be summarized as follows:

- The system has a software interface that performs the offline training that is utilized for generating the DNA sequence.
- This sequence is relatively easy to adapt to new types of benign traffic.
- The system can be adapted to the degree of protection needed through changing the threshold value.
- DNA pattern matching is implemented on FPGA to speed up online processing.
- Searching takes only one clock cycle using associative or content-addressable memory modules emulating organic DNA pattern matching. Ten more clock cycles are required for buffering the entire codon.
- All generated codons are of fixed size of fifty bases that is equal to 100 bits versus organic codons of three bases.

Based on this work, we believe that the proposed DNA-based methodology is a promising new area for IDS research. The implementation of the micro-architecture, shown in the timing reports, provides a maximum path delay of slightly more than four and a half nano seconds. This value is equivalent to about 45-bit delay in a 10 Gigabit Ethernet. We are not counting the buffering delays encountered with each sniffed packet. Considering time slices between consecutive observations, we reason that such a delay is quite acceptable for most of today's fast networks.

References:

[1] Baker, V. Prasanna, "Time and Area Efficient Pattern Matching on FPGAs," Proceedings of the 12th ACM International Symposium on Field Programmable Gate Arrays, 2004.

[2] Bu, John A. Chandy, "FPGA-Based Network Intrusion Detection using Content Addressable Memories," Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004.

[3] Coull S, Joel Branch, Joel Branch, Eric Breimer, "Intrusion Detection: A Bioinformatics Approach," Proceedings of the 19th Annual Computer Security Applications Conference, 2003.

[4] "DARPA Intrusion Detection Evaluation Dataset", [Online source], available at: http://www.ll.mit.edu/IST/ideval/data/data_index.htm

[5] Gao D, Michael K. Reiter, Dawn Song, "Behavioral Distance for Intrusion Detection," Proceedings of the 8th International Symposium On Recent Advances in Intrusion Detection, 2005.

[6] Gehani A, Thomas LaBean, John Reif, "DNA-Based Cryptography," IMACS DNA-Based Computers V, American Mathematical Society, USA, 2000.

[7] Hu S, "An Intrusion Detection Scheme Utilizing Teiresias to Determine a Computer's DNA Sequence Responsible for Network Traffic," Simon Fraser University, 2002.

[8] Jones N.C, P. Pevzner, An Introduction to Bioinformatics Algorithms (Computational Molecular Biology), MIT Press, 2004.

[9] Kim J, Peter Bentley "The Human Immune System and Network Intrusion Detection," University College London, 1999.

[10] Kristopher Kendall, "A Database of Computer attacks for the Evaluation of Intrusion Detection Systems," Masters Thesis, MIT, 1999.

[11] Lippmann, R., et al., "The 1999 DARPA Off-Line Intrusion Detection Evaluation," Computer Networks, vol. 34(4), pp. 579-595, 2000.

[12] Mahoney M.V, Philip K. Chan, "PHAD:

Packet Header Anomaly Detection for Identifying Hostile Network Traffic," Florida Institute of Technology Technical Report CS-2001.
[13] Nilsen G, Jim Torresen, Oddvar Sorasen, "A Variable Word-Width Content Addressable Memory for Fast String Matching," Proceedings of Norchip conference, 2004.
[14] Xilinx, Inc. Synthesis and Verification Guide, ISE 6.2i. Xilinx Inc.
[15] Xilinx, Inc. "Virtex 4 Family Overview", DS112 (v2.0), 2007.
[16] Yu, B., E. Byres, C. Howey, "Monitoring Controller's "DNA Sequence for System Security," BCIT, Burnaby, BC, 2001.

APPENDIX A: IMPLEMENTATION REPORTS

Implementation reports
In this appendix, we provide the details of the implementation reports as they were made available by the Xilinx CAD software [14].

Design Information
Target Device : xc4vlx200
Target Package : ff1513
Target Speed : -12
Design Summary
Number of Slices : 3313 out of 89088 3%
Number of CLBs : 828 out of 22272 3%
Slice Flip Flops : 808 out of 178176 0%
4 input LUTs : 4443 out of 178176 2%
Number of bonded IOBs : 17 out of 964 1%
Total equivalent gate count for design : 4261
Timing Summary
Minimum period : 8.2870ns
Maximum frequency : 120.665MHz
Maximum combinational path delay : 4.565ns

The Floor Plan

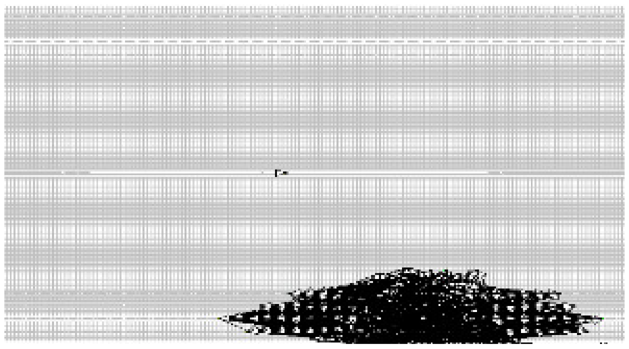
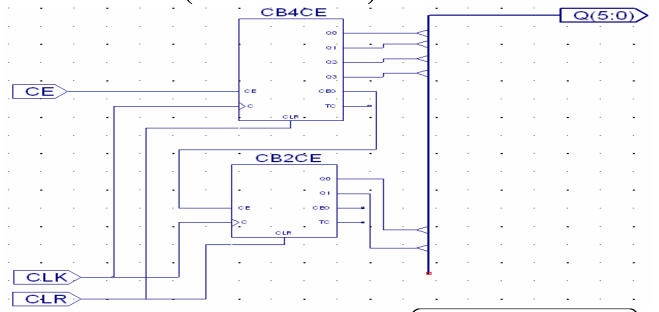


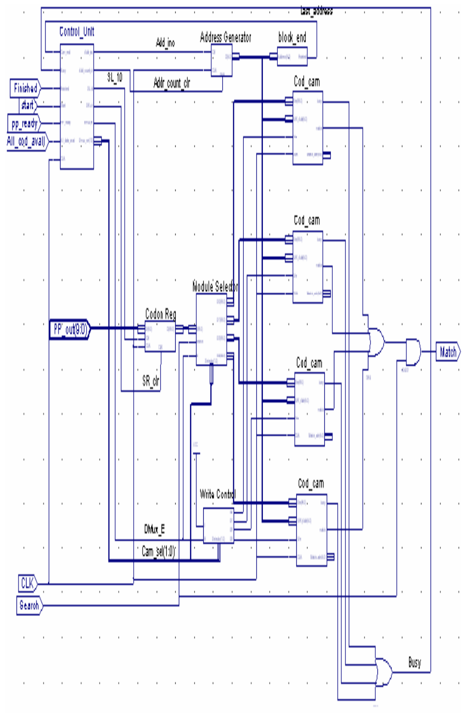
Figure 11. The floor plan

APPENDIX B: (SCHEMATICS)



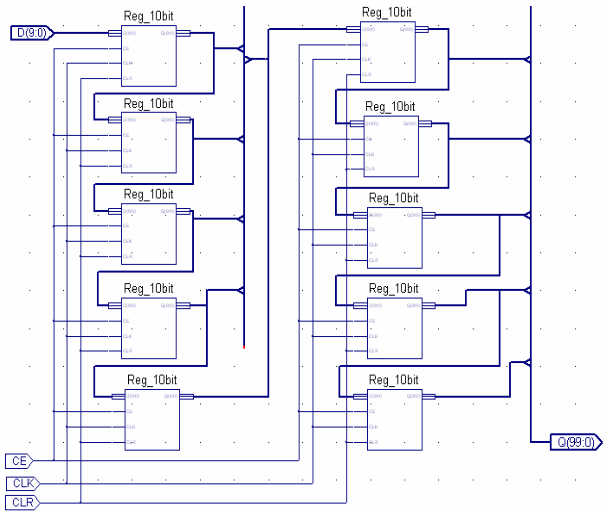
CB4CE: 4-bit binary counter with clock enable

Figure 12. The circuit diagram of the address generator.



- ***Address Generator***: generates next write address
- ***Cod_cam***: CAM module 100x64 (codons buffer)
- ***Codon_Reg***: 100 bit shift register
- ***Module_selector***: selects one of the CAM modules (write) or all the modules (search)
- ***Write_control***: enables module's write operation.
- ***Block_end***: detects CAM module last address.

Figure 13. The circuit diagram of the entire design



Reg_10bit: 10-bit Asynchronous clear loadable register

Figure 14. The circuit diagram of the 100-bit shift register.