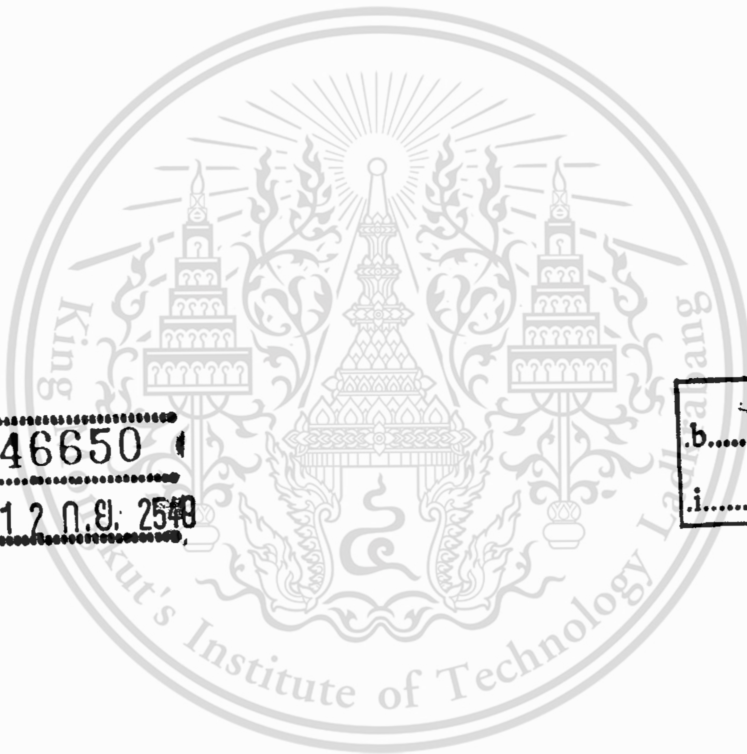
HIGH-THROUGHPUT STRING MATCHING ON RECONFIGURABLE HARDWARE

THINH TRAN NGOC



เลขหมู่..................
เลขทะเบียน 46650
วัน,เดือน,ปี 12 ก.ย. 2549

b. ....
i. ....

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2006

ISBN 974-15-2195-2

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

COPYRIGHT 2006

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

หัวข้อวิทยานิพนธ์	**ไฮทรูพุทสตริงแมทชิ่งสำหรับรีคอนฟิกกิวเรเบิลฮาร์ดแวร์**
ผู้วิจัย	Thinh Tran Ngoc
รหัสนักศึกษา	47060835
ระดับการศึกษา	**วิศวกรรมศาสตรมหาบัณฑิต**
ภาควิชา	**วิศวกรรมคอมพิวเตอร์**
ปี	2549
อาจารย์ผู้ควบคุมวิทยานิพนธ์	**ผศ. ดร. สุรินทร์ กิตติธรกุล**

บทคัดย่อ

ในงานวิจัยนี้ได้นำเสนอการออกแบบฮาร์ดแวร์ 2 แบบ สำหรับระบบการตรวจสอบความคล้ายคลึงกันของสตริง ซึ่งนำมาประยุกต์ใช้งานทางด้านชีววิทยาและด้านความปลอดภัยในระบบเนตเวิร์ก กล่าวถึงงานทางด้านชีววิทยา ได้นำมาใช้งานกับการตรวจสอบความคล้ายคลึงกันของดีเอ็นเอ ซึ่งประยุกต์ให้สามารถทำงานค้นหาดีเอ็นเอในฐานข้อมูลได้พร้อมๆ กันถึง 16 เส้น ระบบสามารถทำงานได้เร็วถึง 1 ล้านล้านคำสั่งต่อวินาที โดยติดต่อเครื่องคอมพิวเตอร์ผ่านพอร์ตยูเอสบี 2.0 ด้วยความเร็วในการส่งข้อมูลจากฐานข้อมูลเพียงแค่ 24 เมกะบิตต่อวินาที ระบบนี้ออกแบบโดยใช้ภาษาเวอร์ริล็อกและใช้เอฟพีจีเอรุ่น Xilinx Virtex-4 XC4VLX60 จากการเปรียบเทียบกับการออกแบบระบบนี้ด้วยซอฟต์แวร์ เมื่อเทียบประสิทธิภาพแล้วระบบสามารถทำงานได้ดีกว่า 100 เท่า และประหยัดเวลาในการส่งข้อมูลมาจากฐานข้อมูลได้มากถึง 16 เท่า ซึ่งจุดนี้เป็นปัญหาคอขวดของระบบอื่นๆ ที่นำมาเปรียบเทียบ

ระบบตรวจสอบการบุกรุกผ่านเนตเวิร์กในส่วนประกอบที่สำคัญส่วนหนึ่งในโครงสร้างพื้นฐานของระบบเนตเวิร์กทั่วไป เนื่องจากว่าวิธีการตรวจสอบการบุกรุกในปัจจุบันทำงานได้ล่าช้าและไม่รองรับกับความเร็วของเนตเวิร์ก ดังนั้นการใช้ฮาร์ดแวร์เข้ามาช่วยจะสามารถเสริมการทำงานได้เร็วยิ่งขึ้น ในงานวิจัยนี้ได้นำเสนอระบบเปรียบเทียบสตริงบนเอฟพีจีเอตามแบบของ Snort NIDS ซึ่งสามารถทำงานได้ดีในระดับหนึ่ง

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Thesis Title	High-Throughput String Matching on Reconfigurable Hardware
Student	Mr. Thinh Tran Ngọc
Student ID.	47060835
Degree	Master of Engineering
Programme	Computer Engineering
Year	2006
Thesis Advisor	Asst. Prof. Dr. Surin Kittitornkun

ABSTRACT

We propose two reconfigurable hardware architectures for string matching in the contexts of *computational biology* and *network security*. In computational biology, a multiple DNA similarity matching system on reconfigurable hardware can search up to sixteen DNA query sequences against a large DNA database. The system can sustain over 1-Tera Operations/Second throughput rate via a USB 2.0 host interface at only 24 Mbps of database transfer rate. Based on the recent processor array design methodology, the system is designed using Verilog hardware description language and configured on a Xilinx Virtex-4 XC4VLX60 FPGA. Compared to software implementation, the array can achieve approximately hundreds folds increase in performance while saving time spent on transferring the database which is the bottleneck of the other FPGA & ASIC systems up to sixteen folds.

In network security, Network Intrusion Detection Systems (NIDS) are becoming critical components of the network infrastructure as they serve as a key line of defense in network protection. However, current methods cannot meet the bandwidth requirements of a moderate sized corporate network. Thus, hardware techniques are desired to speed up the string matching. In this thesis, we introduce a novel string matching architecture for an FPGA based Snort NIDS that can match strings in a throughput-and area-efficient manner.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Acknowledgements Contents

First of all, I would like to thank Assistant Professor Dr. Surin Kittitornkun, my Advisor, and Professor Dr. Shigenori Tomiyama of Tokai University, Japan, my Co-Advisor, for their helpful suggestions and constant supports during this research at KMITL.

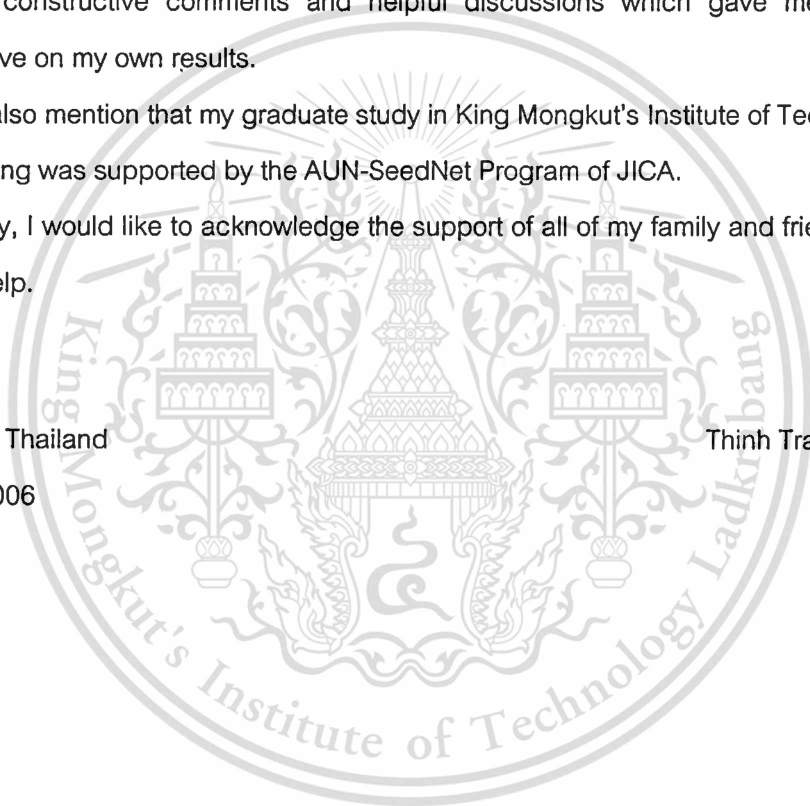
I am also thankful to thesis committee members in Department of Computer Engineering for their constructive comments and helpful discussions which gave me a better perspective on my own results.

I should also mention that my graduate study in King Mongkut's Institute of Technology Ladkrabang was supported by the AUN-SeedNet Program of JICA.

And finally, I would like to acknowledge the support of all of my family and friends for all of their help.

Bangkok, Thailand Thinh Tran Ngoc

March, 2006



This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Contents

Page

บทคัดย่อ III

ABSTRACT IV

Acknowledgements Contents V

Contents VI

List of Tables IX

List of Figures X

Chapter 1 Introduction 1

1.1 Motivation 1

1.2 Existing Approaches 1

1.3 Statement of Problem 2

1.4 Contributions 3

1.5 Organization 3

Chapter 2 Literature Review 4

2.1 DNA Similarity Search 4

2.1.1 Dynamic Algorithms 5

2.1.2 Heuristic Algorithms 7

2.1.3 FPGA Solutions of Smith & Waterman Algorithm 8

2.2 Network Intrusion Detection Systems (NIDS) 10

2.2.1 Snort NIDS 11

2.2.2 Software NIDS Solutions 12

2.2.3 Hardware NIDS Solutions 13

Chapter 3 Multiple DNA Matching 17

3.1 FPGA Design Methodology of Multiple DNA Matching 17

3.1.1 S&W Algorithm for Multiple Similarity Searches 18

3.1.2 Systolic Design Methodology 19

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

Contents (cont.)

Page

3.1.3 FPGA Implementation of Multiple DNA Matchings 21

3.2 Firmware & Software Design for Multiple DNA Matchings System 28

3.2.1 Firmware Design 29

3.2.2 Software Design 32

3.3 Results and Comparison 34

3.3.1 FPGA Throughput 34

3.3.2 Performance Analysis of Multiple DNA Matchings System 38

Chapter 4 Deep Packet Filtering in NIDS 41

4.1 Design Methodology for Deep Packet Filtering in NIDS 41

4.1.1 Match Processor Array 42

4.1.2 Address Calculation Logic 45

4.1.3 Control Unit 46

4.2 Result of our Deep Packet Filtering in NIDS system 47

4.2.1 Comparison of Deep Packet Filtering with Previous Works 48

Chapter 5 Conclusions and Future Works 51

5.1 Conclusions 51

5.2 Future Works 51

Bibliography 53

Appendix A Avnet FPGA Virtex-4 Evaluation Kit 57

A.1 Description 57

A.2 Features 58

A.3 Hardware 59

A.3.1 Virtex-4 FPGA 59

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Contents (cont.)

Page

A.3.2 Clocks 59
A.3.3 Memory 60
A.3.4 DDR SDRAM 60
A.3.5 Flash Memory 60
A.3.6 Universal Serial Bus (USB) 60
A.3.7 10/100 Ethernet 61
A.3.8 RS232 Transceiver 61

Appendix B Implemented Verilog Code 62
B.1 Multiple DNA Matchings 62
B.1.1 Processing Element Hardware Description 62
B.1.2 Systolic Array Hardware Description 64
B.2 Deep Packet Filtering in NIDS 67
B.2.1 Hardware Description of A Processing Element in Match Processor Array 67
B.2.2 Systolic Array Hardware Description of Match Processor Array 68

Appendix C Publication List 70

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

List of Tables

Table Page

3.1 Two-bit Nucleotide Representation 22

3.2 Comparison of FPGA-based DNA search systems 35

3.3 Comparison of our Multiple DNA Matching system with Hokiegene [1] on XC2V6000 FPGA chip 36

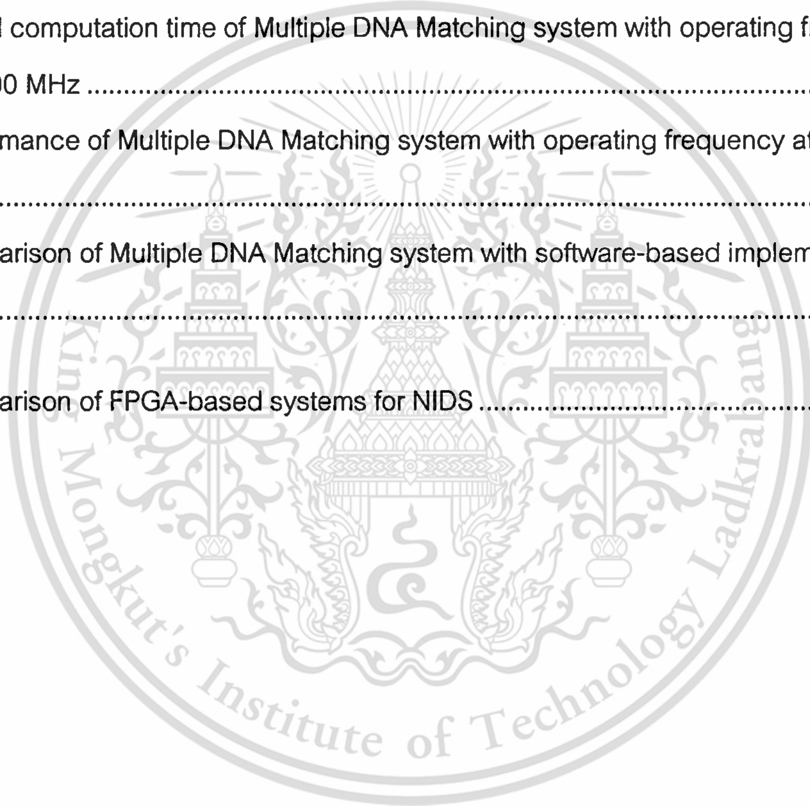
Comparison of our Multiple DNA Matching system with S&W Cell [2] on XCV1000-6 FPGA chip 36

3.5 Actual computation time of Multiple DNA Matching system with operating frequency at 100 MHz 38

3.6 Performance of Multiple DNA Matching system with operating frequency at 100 MHz 39

3.7 Comparison of Multiple DNA Matching system with software-based implementations 40

4.1 Comparison of FPGA-based systems for NIDS 48



This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

List of Figures

Figure Page

2.1 The distance table is generated when comparing query sentence to database sequence ... 6

2.2 Run times of multiple similarity searches of some well-known software with 10 queries ... 8

3.1 The Multiple DNA Matchings System ... 18

3.2 The distance tables are generated when comparing two query sentences "AGA" and "ACT" to database sequence "ACT" ... 19

3.3 a) Dependence Graph m = 4, n = 4 and q = 3; b) A linear systolic array architecture resulted from the space-time mapping in Eq.(3.4) ... 20

3.4 Block Diagram of the FPGA implementation of Multiple DNA Matchings system ... 22

3.5 Microarchitecture of each processing element (PE) in Figure.3.3(b) ... 23

3.6 Our systolic array for multiple DNA matchings, #PEs = m = 5, 200 ... 25

3.7 Simulation waveform of a two-PE array for Multiple DNA Matchings ... 26

3.8 State Machine of Multiple DNA Matchings ... 26

3.9 General Programmable Interface's (GPIF) Place in FX2 USB System ... 28

3.10 GPIF's write waveform design of Cypress FX2 USB ... 31

3.11 GPIF's read waveform design of Cypress FX2 USB ... 32

3.12 User Interface of Multiple DNA Matchings System ... 33

3.13 Data transfer rate of our Multiple DNA Matchings system compares with the other FPGA-based systems when the number of query sequences varies from 1 to 16 ... 37

3.14 Actual performance of Multiple DNA Matching system with difference database sizes ... 40

4.1 Overview of Deep Packet Filtering in NIDS system ... 42

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

List of Figures (cont.)

Figure Page

4.2 Match Processor Array 43

4.3 Example of Match Processor Array 43

4.4 MicroArchitecture of a Processing Element in Match Processor Array 44

4.5 Match Shift Array in Address Calculation Logic 45

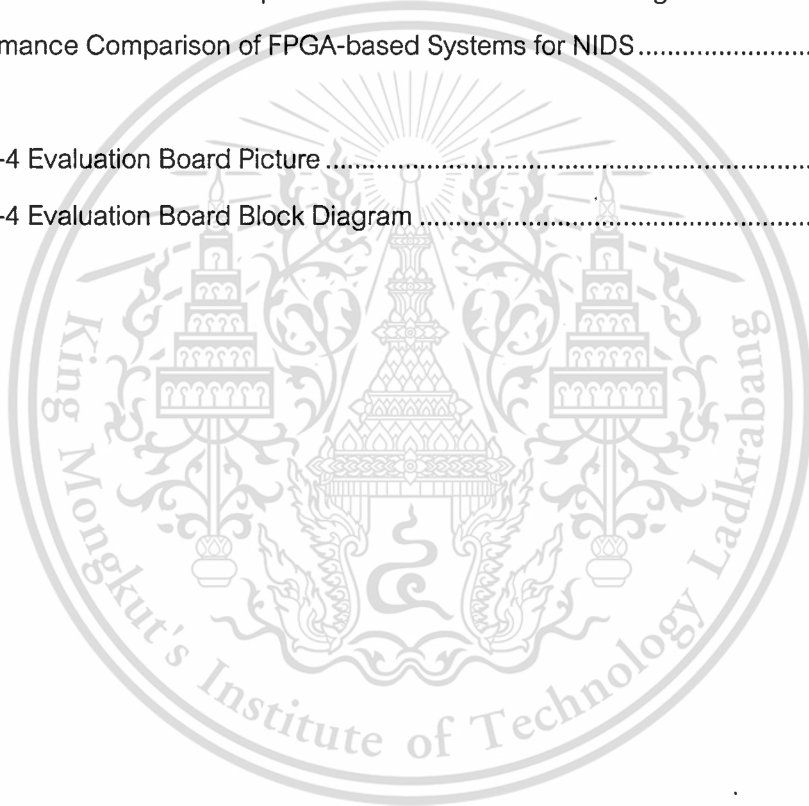
4.6 The Block RAM and the Down Counter for calculating the address of the matched rule 46

4.7 Simulation waveform of comparison rule ABC with AABC string 47

4.8 Performance Comparison of FPGA-based Systems for NIDS 50

A.1 Virtex-4 Evaluation Board Picture 57

A.2 Virtex-4 Evaluation Board Block Diagram 58



This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 1

Introduction

1.1 Motivation

String matching is one of the fundamental problems in computer science, which involves the finding of one or all the occurrences of a pattern string in another string or body of text. In general, string matching can be separated into two categories that are exact string matching and approximate string matching. Exact string matching involves match patterns, where they exist completely, unbroken and with no irrelevant data in between any letters. Some applications are Network Intrusion Detection System (NIDS), text editing, etc. On the other hand, pattern string in approximate string matching rarely matches the text completely. It can be found in numerous applications such as computational biology, image detection, handwriting recognition, etc. Many string matching algorithms which specific to above applications have been proposed in the past. The calculating time in some applications of string matching is the main problem if we only use the general computer to solve it.

Nowadays, to increase the throughput of string matching, people tend to implement string matching algorithms on hardware such as ASIC and FPGA. String matching can get high throughput on hardware because it can exploit parallel and pipelining capability. ASIC is so complex and expensive that it is only suitable for high volume products. FPGA is a low cost device so it is well suited for many applications. Moreover, one of the powerful characteristics of FPGA in comparison with ASIC is its flexibility, i.e. the system can be easily updated or reconfigured at run time.

With these advantages of FPGA, we can apply it for both kinds of string matching. For specific applications, those are multiple DNA matching in computational biology and deep packet detection in NIDS.

1.2 Existing Approaches

On the hardware side, there have been a lot of ASIC and FPGA implementations of the string matching to accelerate the throughput.

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

In computational biology, FPGA implementations of Smith-Waterman (S&W) algorithm [3] include [1, 2, and 4]. Reference [4] is one of the first FPGA implementations using "edit distance" method for similarity search. It can improve significant performance when compared with software. In [1], the authors implemented an enhanced version of a genetic search algorithm (Smith-Waterman). Their system uses Run-Time Reconfiguration (RTR) to optimize the area of chip. Without the need to perform runtime reconfiguration, the design in [2] can decrease the area of chip while increasing throughput significantly. However, all of them focus on comparing one query sequence against the database sequences.

Existing NIDS systems implemented on FPGA are numerous. Previous approaches to string matching on FPGAs are finite automata methods that translate regular expression signatures into hardware implementations [5, 6, 7]. In other words, these methods translate finite automata directly to FPGA circuitry. In [8], a hardwired design was developed to provide high area efficiency and performance by using replicated hardwired 32-bit comparators in a parallel/pipeline structure. Another approach to FPGA-based NIDS is the use of content addressable memories (CAMs). Content addressable memories have long been used for fast string matching against multiple keywords [9, 10]. The Granidt project of Los Alamos National Laboratory implements a fast re-programmable deep packet filter using CAMs and the Snort rule set [11]. Sourdis maps a similar design with a deeper pipeline to increase the filtering rate [12] while Clark et al. make some area improvements [13].

1.3 Statement of Problem

String matching is the most computationally intensive part of many applications. With a powerful reconfigurable architecture, current state-of-the-art FPGAs offers tremendous opportunity to implement string matching at high throughput and low cost. The main challenge faced by researchers is how to maintain high performance while minimizing the cost. To achieve this goal, it calls for effective and efficient design methodology that can optimize both computational capacity and data transfer rate in order to increase overall performance.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

1.4 Contributions

This thesis explores the use of reconfigurable hardware to achieve high throughput of string matching applications. The following use its contributions.

1. It demonstrates and analyzes the use of FPGA-based processor array in string matching systems.
2. It suggests methods for very deep pipelining of input data to increase throughput and decrease the transfer rates of input data.
3. It exploits the runtime reconfigurable potential of FPGA devices to string matching.

1.5 Organization

This thesis is organized in the following manner. Chapter 2 presents the background for the work presented in this thesis. The background discusses related researches and concepts that contributed to this thesis. As one of the major contributions, an approximate string matching implementation on FPGA Virtex-4 Evaluation Kit of Avnet, called multiple DNA matching system, is presented in Chapter 3. Then, Chapter 4 describes the second application of FPGA implementation for exact string matching of deep packet filtering in Network Intrusion Detection System. Finally, Chapter 5 concludes this research work by providing the conclusion and some important issues for future work.

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 2

Literature Review

In this chapter, the notions of DNA similarity search and Network Intrusion Detection System are introduced respectively.

2.1 DNA Similarity Search

Bioinformatics is a field of science that brings together biology, computer science, and information technology, in an effort to increase our understanding of biology. As the field of bioinformatics has developed, it has been labeled with various other names. These names include computational biology, computational genomics, computational molecular biology, and biomedical informatics. Many scientific motivations behind this field are drug development and determination of evolutionary trees of multiple species from their DNA. The field of bioinformatics includes many computationally challenging problems, many of which involve large amounts of data, very complex systems, or both.

Deoxyribonucleic acid (DNA) is the molecule that encodes genetic information in the nucleus of cells. It determines the structure, function, and behavior of the cell. The DNA consists of two strands of linked nucleotides with one of the four bases adenine (A), thymine (T), guanine (G) and cytosine (C). Each base in one strand binds to a specific base in another strand, where A's bind to T's, and G's bind to C's, each of which is called a base-pair. The two strands with bound base pairs constitute a DNA molecule.

Nowadays, database similarity searching has been one of main problems in the field of bioinformatics. It allows us to determine which of the hundreds of thousands of sequences present in the database, such as DNA strands and proteins, are potentially related to a particular sequence of interest.

Most algorithms for similarity search can be categorized into two groups: dynamic and heuristic.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

2.1.1 Dynamic Algorithms

Dynamic programming method for similarity searches makes all pair wise comparisons between the two strings. It achieves high sensitivity so that all the matched and near-matched pairs are detected, however, its computation demanding required strongly limits its use. Dynamic algorithms give optimal solutions, and well known searching algorithms like S&W [3], Needleman-Wunch [14] and Hidden Markov Models are of the dynamic kind.

Smith-Waterman Algorithm

This part discusses the Smith-Waterman algorithm. In this algorithm, a pattern *X* is to be matched against a text *Y*. The pattern *X* consists of m letters, in other words $X = x_1x_2...x_m$ such that for each *i* between 1 and *m*, $x_i \in S$ where *S* is the alphabet. *Y* consists of *n* letters of the same alphabet *S* or, more formally, $Y = y_1y_2...y_n$ with each $y_i \in S$. The pattern *X* is normally shorter or equal in length to the text *Y* ($m \leq n$).

Smith and Waterman devised an algorithm for matching similar patterns. The Smith-Waterman (S&W) algorithm compares a pattern *X* to text *Y* and calculates the penalty required to change *X* into *Y*. Due to the fact that *X* and *Y* may not match exactly, the penalty will take into account the number of insertions, deletions, and substitutions needed to convert the strings to match each other. This penalty is referred to as the ***edit distance***, $d_{i,j}$.

$$d_{i,j} = \min \begin{cases} d_{i-1,j} & + c_{del} \\ d_{i,j-1} & + c_{ins} \\ d_{i-1,j-1} & + c_{sub} \end{cases} \quad (2.1)$$

where

$$\begin{aligned} d_{0,0} &= 0 \\ d_{i,0} &= d_{i-1,0} + c_{del} \\ d_{0,j} &= d_{0,j-1} + c_{ins} \end{aligned}$$

Here c_{sub} is the penalty for substituting characters if a mismatch is found. c_{del} is the gap penalty for a deletion and c_{ins} is the gap penalty for an insertion.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

		A	G	A	C	T	A	G	G
	0	1	2	3	4	5	6	7	8
T	1	2	3	4	5	4	5	6	7
G	2	3	2	3	4	5	6	5	6
C	3	4	3	4	3	4	5	6	7
T	4	5	4	5	4	3	4	5	6
A	5	4	5	4	5	4	3	4	5
A	6	5	6	5	6	5	4	5	6
G	7	6	5	6	7	6	5	4	5
C	8	7	6	7	6	7	6	5	6

Figure 2.1: The distance table is generated when comparing query sentence to database sequence

In genomic databases, four character alphabets are used to represent the four bases in the DNA molecule. These characters are typically denoted *A* for adenine, *T* for thymine, *G* for guanine, and *C* for cytosine. For example, Figure.2.1 shows the distance table generated when comparing query sentence *"TGCTAAGC"* to database sequence *"AGACTAGG"* with the following cost functions:

$$c_{del} = c_{ins} = 1$$

$$c_{sub} = \begin{cases} 0 & , if\ x_i = y_j \\ 2 & , if\ x_i \neq y_j \end{cases}$$

The resulting edit distance is 6 which can be found at the lower right hand corner of the table.

The S&W Algorithm computes all values in the matrix in order to generate the global edit distance in the bottom right cell. For the *n x m* matrix, the complexity to compute all values is *O(n x m).* Because data dependencies exist only on top and to the left of each cell, diagonal values in the S& W matrix can be computed in parallel. The parallelized S&W Algorithm computes up to *n* operations at each step. However, due to the data dependency, the algorithm cannot achieve perfect speedup. Thus, the algorithm computes the matrix in *n+m-1* steps assuming *n* available processing elements.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

2.1.2 Heuristic Algorithms

In addition to the dynamic algorithm, there are a number of other algorithms used to solve the gene-matching problem. Both FASTA [15] and the Basic Local Alignment Search Tool (BLAST) [16] use heuristics to reduce the complexity of the algorithm. Both FASTA and BLAST compare small segments of the query to the database. The assumption in both algorithms is that good matches have a large number of substrings with exact matches. FASTA first computes a position-specific comparison of the genes to generate sets of matches on the same diagonal of the S&W matrix. Using the diagonal with the most matches, the S&W Matrix is computed for a small band surrounding the chosen diagonal. BLAST operates using a similar technique. However, instead of performing a gapless alignment with a specific gene in the database, the algorithm compares the query sequence against a set of genes with common structure, function or evolutionary origin. BLAST associates a higher score when the set of genes having common entries. BLAST then takes the highest matches and computes a small section of the S&W matrix. These heuristic approaches can miss matches and produce false positives. Both FASTA and BLAST improve the speed of the S&W algorithm at the cost of sensitivity in comparing genes. BLAST becomes less precise in finding matches when a family of genes has less in common.

Besides, MegaBlast is a new program using greedy algorithm of Zhang et al. [17] for nucleotide sequence alignment search and concatenates many query sequences to save time spent on scanning the database. It is up to ten times faster than more common sequence similarity software programs and therefore can be used to swiftly compare two large sets of sequences against each other. The disadvantage of MegaBlast is also the very low sensitivity, i.e. significant matches may be missed by the searches.

Software implementations of MegaBlast, BLAST, FASTA and S&W are available for download over the Internet. Figure.2.2 illustrates run times of multiple similarity searches of some well-known software implementations with 10 queries. The lengths of the queries are approximately 150-400 characters, where the database *env_nt* contains over 800 thousands nucleotide sequences and has over 800 millions characters on the

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

platform is a Pentium4 1.8 GHz, 512MB Ram, 80GB hard drive, and Windows 2000 Professional. It is obvious that MegaBlast is much better than S&W.

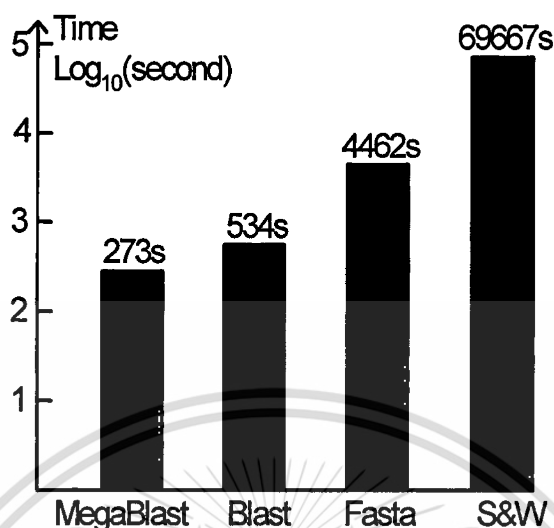


Figure 2.2: Run times of multiple similarity searches of some well-known software with 10 queries

2.1.3 FPGA Solutions of Smith & Waterman Algorithm

In recent years, people tend to implement S&W algorithm on FPGA platform because of the flexibility and reconfigurable characteristic of it. Some Smith-Waterman (S&W) algorithm [3] implementations on FPGA include [1, 2, 4, 18, 19].

In [4], one of first FPGA implementations using *"edit distance"* method for similarity search was proposed. Two systolic arrays for computing the edit distance, bidirectional and unidirectional arrays, were presented end their implementations on Splash2 were described.

In bidirectional method, each Processing Element (PE) computes the distances along a particular diagonal of the distance matrix. The source and target sequences enter the array on opposite ends and flow in opposing directions at the same speed. Comparing sequences of lengths m and n requires at least $2\ max(m+1,n+1)$ processors. In each step, at most half of the PE's are active.

In unidirectional method, data flows through the array in one direction. The source sequence is loaded once and stored in the array. The target sequences are streamed

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

through the array one after another. With this method, the utilization of PE is nearly 100%. In this configuration, each PE computes the distances in one row or column of the distance matrix. At each time step, the PE's compute the distances along the anti-diagonal in the distance matrix.

SPLASH-2 consists of a Sun SPARCstation host, an interface board, and from one to sixteen Splash array boards containing each 14 FPGA processing elements. So the total PEs is 248. This machine was marketed under the WILDFIRE name by Annapolis Micro Systems, Inc.

In [1], a system implements an enhanced version of a genetic search algorithm (Smith-Waterman) using runtime customization and reconfiguration, operating on a single FPGA device, i.e. the characters of query sequence inside PE can change at run time. And some of the optimizations involving fixed constants such as the insert, delete and substitution penalties also make the area of chip as small as possible. This system is able to perform sequence matching at a rate of about a trillion cell updates per second.

In [2], a technique, in which two processing elements are merged into a compact cell, was used to develop a Smith-Waterman systolic processing element design which computes the edit distance between two strings. This cell occupies 3 Xilinx Virtex slices and allows both strings to be loaded into the system without runtime reconfiguration. Using this cell, 4,032 PEs can fit in a Xilinx XCV1000E-6, operating at 202 MHz and achieving a device performance of 814 billion cell updates per second. But when the working design was benchmarked, it had a disappointing performance of approximately 136 B CUPS, which is limited by the simple polling based host interface.

Besides the edit distance method, some people use traditional method of S&W algorithm [18, 19]. The advantage is that they can calculate the local alignment score. The disadvantage is that the area of chip is very large and the complexity is high so that its throughput drops down when comparing with the edit distance method. Moreover, the dynamic range of the score is also too high that the number of bits to encode is the problem.

Yamaguchi et al. [18] implements the system using one off-the-shelf PCI board with one FPGA and a Pentium based computer system. The features of their approach are:

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

1. The system computes the scores of elements in every two clock cycles.
2. The system uses multi-thread for query sequence and database sequence in order to achieve high performance.
3. And the system uses two phase search in order to make up for limited memory bandwidth. In the first phase, database sequences are divided into sub-sequences, because the size of the intermediate results described above is very large, and can not be stored in the internal memory of the FPGA at once. In the second phase, they find the path from the upper left position to the lower right position which gives the best score.

The time for comparing a query sequence of 2,048 elements with a database sequence of 64 million elements is about 34 seconds, which is about 330 times faster than a desktop computer with a 1GHz Pentium III. They also tested system on laptop computer using one PC card with one FPGA (Xilinx XCV300). The performance is about 30 times faster than the desktop computer.

The implementation in [19] considers both linear and affine gap penalties and it is able to compute local alignments. They have described a partitioning strategy to implement database scans with a fixed-size processor array and vary query sequence lengths. Using this method, they can achieve supercomputer performance at low cost on an off-the-shelf FPGA. Their implementation achieves a speedup of approximately 170 for linear gap penalties and 125 for affine gap penalties as compared to a standard desktop computing platform. The system is around three times faster than FPGA implementation presented in [18] on a Virtex XCV2000E.

In commercial products, several companies have produced commercial solutions to the gene matching problem. Time Logic [20] and Compugen [21] have produced products that exploit the use of FPGAs to accelerate their solutions. Both speed up their inner loops of their algorithms by using FPGA boards. The Paracel [22] Gene Matcher2 [23] product uses ASICs and software designed for a Linux distributed system.

2.2 Network Intrusion Detection Systems (NIDS)

Nowadays, network security is becoming more and more important as the internet is growing with a fantastic speed. In fact, any computer that is connected to the network

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

is in danger of being attacked. The traditional firewall techniques are ineffective against the new type of attacks. To prevent the attacks, the security systems must take a more detailed look at the incoming network traffic. Network intrusion detection systems (NIDSs) are one of the primary tools available to help us create a secure network infrastructure. NIDSs monitor incoming network traffic for predefined suspicious activities or data patterns and notify system administrators when malicious traffic is detected so that appropriate action may be taken. NIDSs often rely on exact string matching of packet payloads to detect hostile packets and string matching is the most computationally expensive step of the detection process.

For example, The Snort open-source intrusion detection software suite has well over a thousand rules [24]. Current high-performance systems can barely process that many rules on a 100 Mbps moderately loaded network [25]. 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.

2.2.1 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.

Deep Packet Inspection Rules

Snort uses a set of rules to filter the incoming packets. As the number of known attacks growing, the patterns for these attacks are made into Snort signatures. The simple rule structure allows flexibility and convenience in configuring Snort. However, like the other NIDS, there is a performance disadvantage of having a long list of rules.

Snort uses a detection engine that utilizes modular plug-in architecture. There are three primary subsystems that make up Snort: the packet decoder, the detection engine, and the logging and alerting subsystem. Snort maintains its detection rules in a two dimensional linked list of chain headers and options. These rule chains are

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

searched recursively for each packet in both directions. The detection engine checks only those options that have been set by the rule parser during the run-time. The first rule that matches a decoded packet in the detection engine triggers the action specified in the rule definition and returns.

The structure of a rule consists of a command keyword for handling the matching packet, the header information, and various options for other patterns such as the content string for the payload. Snort can use one or more rule files as its input. Each rule file can contain more than one rule signature with the form at shown as following.

Action Protocol SrcIPAddr/Port Direction DstIPAddr/Port Options

The first part of the rule corresponds to the packet header information. The second part corresponds to the pattern search options applied to the application payload. The most computationally intensive option is called *'content.'* This option is the key to better packet filtering in deep packet inspection firewall.

Following rule is a signature that is used by Snort to detect 'Code Red' worm.

alert TCP 128.142.4.10 any ⟶ 198.162.10.1 80 (msg: "IDS552/web-iis IIS ISAPI Overflow ida"; dsize: >239; flags: A+; uricontent: ".ida?"; classtype: system-or-info-attempt; reference: arachnids,552;)

When the signature is loaded on to Snort, the system will 'alert' the administrator if the packet under examination has matching protocol, IP addresses, ports, and other packet characteristics describe within the parenthesis. Above rule will cause the system to specifically search for the pattern ".ida?" in all the payloads of the packets that match the header specifications in the rule signature.

2.2.2 Software NIDS Solutions

Several string matching algorithms have been recently proposed in NIDS especially for SNORT's open source NIDS. First versions of SNORT used brute force pattern matching, which was very slow, making clear that using a more efficient string matching algorithm, would improve performance. The first implementations that improved SNORT used the Boyer-Moore algorithm [26]. This implementation improved SNORT performance 200-500% [27]. The Boyer-Moore algorithm is one of the most well-known algorithms that uses two heuristics to reduce the number of comparisons. It first aligns

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

the pattern and the incoming data (text), the comparison begins from the right-most character, and in case of mismatch the text is properly shifted.

However, the Boyer-Moore algorithm compares each pattern independently against the incoming data, and hence substrings repeated in more than one patterns are compared multiple times. Fisk et al. [28] introduced Set-wise Boyer Moore-Hospool algorithm, which is an adaptation of Boyer-Moore, and is shown to be faster for matching less than 100 patterns.

Another implementation of SNORT was presented in [29], and used Wu-Mander multi-pattern matching (MWM) algorithm [30]. The MWM algorithm performs a hash on 2-character prefix of the input data, in order to index into a group of patterns. This SNORT implementation is much faster than previous ones.

Finally, Markatos et al. proposed E^2xB algorithm, which provides quick negatives when the search pattern does not exist in the incoming data [31, 32, 33]. Compared to Fisk et al., E^2xB is faster, while for large incoming packets and less than 1k-2k rules it outperforms MWM [33].

All the above software-based approaches can support a few hundred Mbps at most. That's 2-20 times slower compared to recent FPGA-based string matching systems.

2.2.3 Hardware NIDS Solutions

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

Many ASIC intrusion detection systems have been commercially developed [34, 35, 36, 37]. Such systems usually store their rules using large memory blocks, and examine incoming packets in integrated processing engines. Generally, ASICs programmable security co-processors are expensive, complicated, and although they can support higher throughput compared to GPP, they do not achieve impressive performance.

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.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

This characteristic of reconfigurable devices allows updating or changing the rule set, adding new features, even changing systems architecture, without any hardware cost.

2.2.3.1 Nondeterministic/Deterministic Finite Automata

The most common approach is the regular expressions matching, implemented using Finite Automata (NFAs or DFAs) [5, 6, 7]. Regular expressions produce designs with low cost, but at a modest throughput. This approach generates regular expressions for every pattern or group of patterns, and implements them with N/DFA. 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 [7] introduced regular expressions and Nondeterministic Finite Automata (NFAs) for finding matches to a given regular expression. They focused in minimizing the space -$O(n^2)$- required to perform the matching, and their automata matched one text character per clock cycle. Hutchings et al. [5] expanding on Sidhu et al. work, used regular expressions, with more complex syntax and meta-characters such as "?" and ".", to describe patterns extracted from Snort database. Using a sequence of 8-bit character matchers they compose the NFA circuit. Hutchings et al. were the first that mentioned the performance bottleneck that occurs in such systems due to large fan-out. Their solution was to arrange flip-flops in a fan-out tree. They managed to include up to 16,000 characters requiring 2.5-3.4 logic cells per matching character. The operating frequency of the synthesized modules was about 50 MHz on a Virtex XCV2000E.

Moscola, Lockwood et al. used the Field Programmable Port Extender (FPX) platform, to perform string matching for an Internet firewall [6]. They used regular expressions (DFAs) to store the patterns. Each regular expression is parsed and sent through JLex [38] to get a representation of the DFA required to match the expression. Finally, JLex representation is converted to VHDL. Their processing engine processes one byte during every cycle. Incoming packet data is stored in two identical buffers. The first buffer is used to feed the parallel DFA matchers with 8-bit packet data. The second buffer stores the incoming packets until the content scanners indicate whether to output or drop a packet. Moscola et al. finally described a technique to increase processing bandwidth. Incoming packets arrive in 32-bit words, and are dispatched to one of the

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

four content scanners. This implementation can operate at 37 MHz on a Virtex XCV2000E and throughput is 1.184 Gbps.

2.2.3.2 CAMs & Discrete Comparators

Another more straightforward approach for FPGA-based string matching is the use of regular CAM or discrete comparators [8, 11, 12]. Current FPGAs give designers the opportunity to use integrated block RAMs for constructing regular CAM. This is a simple procedure that achieves modest performance, in most cases better than simple N/DFAs architectures. Other researchers preferred to use discrete comparators, which leads to designs that operate at higher frequency. Discrete comparators 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 of a pattern, and the flip-flops that already exist in logic cells can be used to create a pipeline, without any overhead. Both regular CAM and discrete comparators achieve high performance, however, they have increased area cost. To reduce this cost, researchers deployed several techniques that increase sharing.

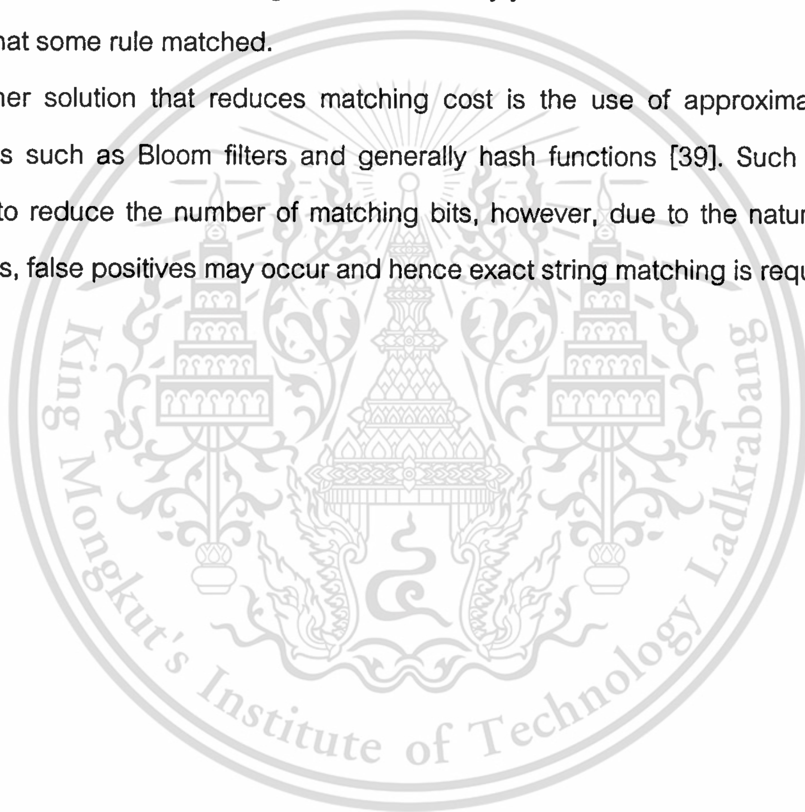
Gokhale, et al. [11] used CAM to implement Snort rules NIDS on a Virtex XCV1000E. They performed both header and payload matching on CAMs, Their hardware runs at 68MHz with 32-bit data every clock cycle, giving a throughput of 2.2 Gbps, and reported a 25-fold improvement on the speed of Snort v1.8 on a 733MHz PIII and an almost 9-fold improvement on a 1 GHz PowerPC G4. Another of CAM implementation, Sourdis et al. [12] mapped a CAMs design with a deeper pipeline to increase the filtering rate. The design uses deep pipeline duplicate comparators, exploits parallelism to increase processing bandwidth, and uses a fast fan-out tree to distribute the incoming data to each comparator. This architecture is able to achieve high performance, but at a significant area cost. The design implemented in a Virtex2-6000 device runs at 250MHz, achieving 8 Gbps throughput. They require about 19-20 logic cells to match a single character, and therefore can store only a few hundreds patterns in a single FPGA.

Closer to our work described in next chapter is the work by Cho, Navab and Mangione-Smith [8]. They designed a deep packet filtering firewall on a FPGA and

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

automatically translated each pattern-matching component into structural VHDL. They presented a block diagram of a complete FPGA-based NIDS, and implemented the content pattern matching unit for more than a hundred signatures. The content match micro-architecture used 4 parallel comparators for every pattern so that the system advances 4 bytes of input packet every clock cycle. In Cho's et al. architecture, incoming packet data is partially matched sequential 4-byte comparators, and finally the results of the four parallel comparators are OR-ed. The design implemented in an Altera EP20K device runs at 90MHz, achieving 2.88 Gbps throughput. They require about 10 logic cells per search pattern character. However, they do not include the fan-out logic, and do not encode the matching rule. Instead they just OR the entire match signals to indicate that some rule matched.

Another solution that reduces matching cost is the use of approximate filtering techniques such as Bloom filters and generally hash functions [39]. Such algorithms succeed to reduce the number of matching bits, however, due to the nature of these techniques, false positives may occur and hence exact string matching is required.



This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

Chapter 3

Multiple DNA Matching

In this chapter, the Multiple DNA Matching system is presented. The Smith & Waterman algorithm is implemented on a FPGA chip in order to increase the performance of similarity searching between DNA query sequences and DNA database.

Our system includes one FPGA board connected to host PC by USB cable as Figure.3.1. In section 1, the main part of system, FPGA implementation is presented. Next, firmware design for USB and simple software on host PC for user's interface are included in section 2. Finally, the results of system and the comparison with previous systems are discussed in last section.

3.1 FPGA Design Methodology of Multiple DNA Matching

In previous chapter, most FPGA implementations of S&W can reduce the computation time by using parallel/pipelined computation such as systolic processor array. As described, they only make comparisons between the two strings at a time. If the query and the database are *m* and *n* characters long, then the computation time complexity of a software implementation and a hardware implementation are $O(mn)$ and $O(m + n)$, respectively. When the database is very large, small changes can be very significant.

However, in reality, data transfer time can make most systems slow down. For example, the theoretical computation performance of [2] can reach up to 814 billion cell updates per second (BCUPS). When the real system was verified, it had a disappointing performance of approximately 136 BCUPS. In addition, to compare a number of query sequences in a very large database, a considerable amount of time is spent on data transfer because the database is read and transferred to the hardware again and again.

This material is reserved for educational use only, not allowed for commercial use.
46650
Forbidden to modify the content, and cite the document when use.

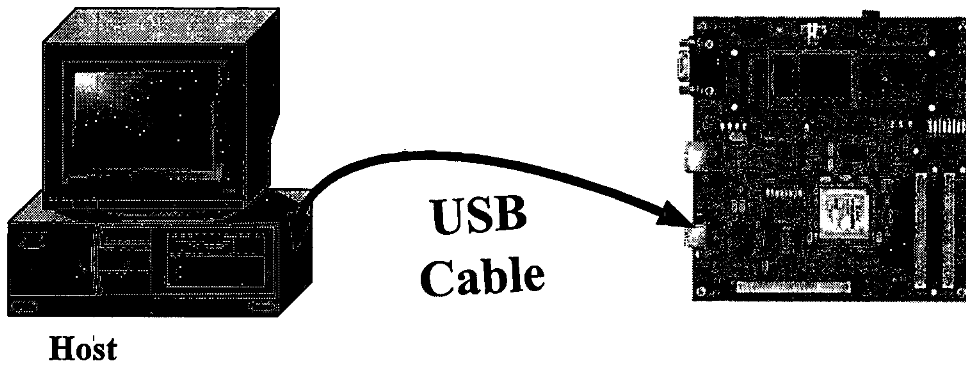


Figure 3.1: The Multiple DNA Matching System

From the problem identified in the previous paragraphs, we will take the advantage of hardware's database transfer time like MegaBlast to reduce the latency and increase the throughput. We use a systolic processor array-like design methodology [40] that is suitable for large FPGA such as Xilinx Virtex-4 LX60. In the processor array, each Processing Element (PE) compares one character of the database to each character of every query sequence every clock cycle.

3.1.1 S&W Algorithm for Multiple Similarity Searches

In multiple similarity searches, the method to calculate each query is also the same with the original formula of S&W algorithm in previous chapter. Let q query sequences be the source sequences of the same length $X_k = x_{1,k}x_{2,k}x_{3,k}...x_{m,k}$, $1 \leq k \leq q$, and $Y = y_1y_2y_3...y_n$ be the database sequence, the edit distance between each query sequence and database sequence is computed according to the following formula:

$$d_{i,j,k} = \min \begin{cases} d_{i-1,j,k} & + c_{del} \\ d_{i,j-1,k} & + c_{ins} \\ d_{i-1,j-1,k} & + c_{sub} \end{cases} \quad (3.1)$$

where

$$\begin{aligned} d_{0,0,k} &= 0 \\ d_{i,0,k} &= d_{i-1,0,k} + c_{del} \\ d_{0,j,k} &= d_{0,j-1,k} + c_{ins} \end{aligned}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

For example, Figure.3.2 shows the distance tables generated when comparing 2 query sentences *"AGA"* and *"ACT"* to database sequence *"ACT"* with the following cost functions:

$$c_{del} = c_{ins} = 1$$

$$c_{sub} = \begin{cases} 0 & , if\ x_{i,k} = y_j \\ 2 & , if\ x_{i,k} \neq y_j \end{cases}$$

The resulting edit distances are 4 and 0 which are found at the lower right hand corner of the tables. Equation (3.1) is suitable for pipelined execution; therefore, systolic array architecture is chosen to implement on FPGA. We use the systolic design methodology described by Kung [41] and a recent improvement by Kittitornkun and Hu [40].

		A	C	T
	0	1	2	3
A	1	0	1	2
G	2	1	2	3
A	3	2	3	**4**

		A	C	T
	0	1	2	3
A	1	0	1	2
C	2	1	0	1
T	3	2	1	**0**

Figure 3.2: The distance tables are generated when comparing two query sentences "AGA" and "ACT" to database sequence "ACT"

3.1.2 Systolic Design Methodology

3.1.2.1 Dependence Graph Design

The Dependence Graph (DG) [41] shows the dependencies of the computations that occur in the algorithm. For example, the DG of the algorithm for $m = 4$, $n = 4$ and $q = 3$ is shown in Figure.3.3.(a). It is a 3-D directed graph according to Eq.(3.1). Each node in the figure represents the assignment of Eq.(3.1). The directed arcs between nodes represent the dependencies between these nodes.

From Figure.3.3.(a), it can be seen that each node for determining the edit distance $d_{i,j,k}$ must provide parameters to the other nodes for the calculation of $d_{i+1,j,k}$; $d_{i,j+1,k}$; and $d_{i+1,j+1,k}$ in every *[i, j]* plane. Each node also passes y_j to the next node along the *k-axis* in every *[i, k]* plane.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

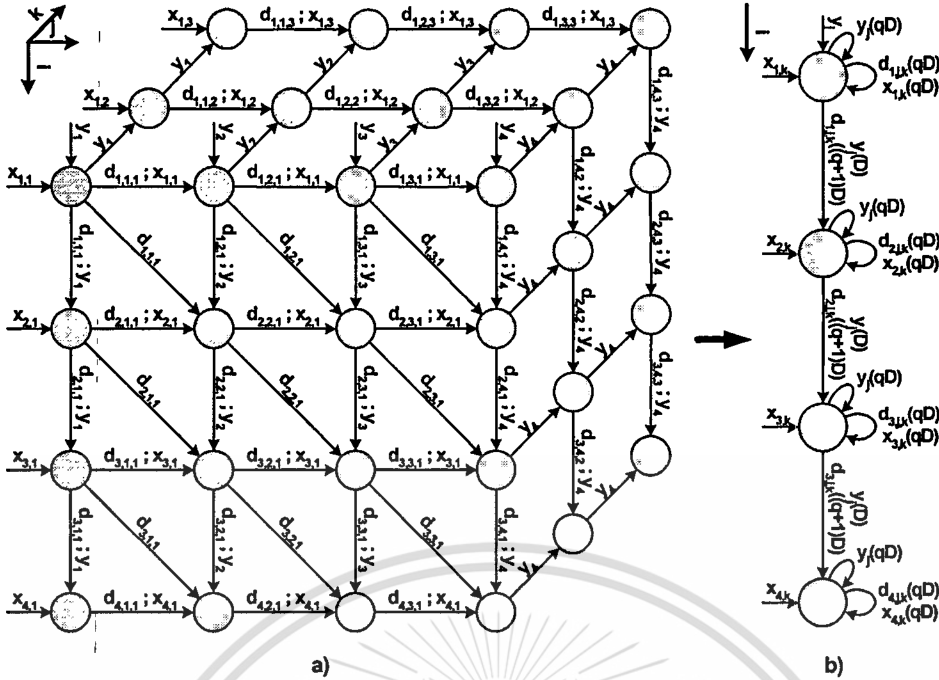


Figure 3.3: a) Dependence Graph m = 4, n = 4 and q = 3; b) A linear systolic array architecture resulted from the space-time mapping in Eq.(3.4)

3.1.2.2 Space-time mapping

The goal of our design is a linear systolic processor array to reduce the number of processors necessary to carry out the computational process. Two important tasks of mapping DG to systolic processor array are processor allocation and scheduling. These can be solved by algebraic projection [40].

This projection is called systolic or space-time mapping.

A dependence matrix D_V is composed of a set of dependence vectors as shown in Figure.3.3.(a):

$$D_V = \begin{bmatrix} \vec{d}_1 & \vec{d}_2 & \vec{d}_3 & \vec{d}_4 \end{bmatrix} \qquad (3.2)$$

where $\vec{d}_1 = [i\ j\ k]^t = [1\ 0\ 0]^t$, $\vec{d}_2 = [1\ 1\ 0]^t$, $\vec{d}_3 = [0\ 1\ 0]^t$, $\vec{d}_4 = [0\ 0\ 1]^t$ are dependence vectors of set of variables $V = \{x_{i,k}, y_j, d_{i,j,k}\}$. The one-D space-time mapping matrix T_1 consists of a scheduling vector s and a processor allocation vector P.

$$T_1 = \begin{bmatrix} \vec{s}^t \\ \vec{P}^t \end{bmatrix} = \begin{bmatrix} s_1 & s_2 & s_3 \\ p_1 & p_2 & p_3 \end{bmatrix} \qquad (3.3)$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

where s_i and p_i are integer numbers. The values of s_i and p_i can be selected using Integer Programming and some heuristic search subject to some design constraints [40]. Based on observation and experiences, we chose $\vec{s}^t = [1 \quad q \quad q]$ and $\vec{p}^t = [1 \quad 0 \quad 0]$. The mapping of D_V results in a delay-edge matrix:

$$\begin{bmatrix} r_1 & r_2 & r_3 & r_4 \\ e_1 & e_2 & e_3 & e_4 \end{bmatrix} = T_1 D_V \qquad (3.4)$$

$$\begin{bmatrix} 1 & q+1 & q & q \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & q & q \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where r_i is the number of registers (delays) of edge e_i which is mapped from the corresponding d_i in Eq.(3.2). Figure.3.3.(b) is a linear systolic array architecture resulting from applying T_1 space-time mapping matrix. The delay 'D' is a pipeline register. For example, in PE2, to calculate $d_{2,j,k}$, we need $d_{2,j-1,k}$ which was calculated q clock cycles before and $d_{1,j,k}$ and $d_{1,j-1,k}$ which were calculated in PE1 one and $q + 1$ clock cycles before, respectively.

3.1.3 FPGA Implementation of Multiple DNA Matching

The FPGA implementation of the Smith-Waterman algorithm consists of four parts, the systolic processor array, the finite-states machines with the up-down counters following, data control, and input-output FIFOs.

The systolic processor array is the main part of system that computes the matrix value for each character in the query string. The state machines follow the systolic processor array to convert the output of the last processing element in the systolic processor array into up-down signals. The up-down signals connect to up-down counters that compute the final edit distance values. The method for receiving and sending data through the system is the final part of the implementation. Figure 3.4 shows the Block Diagram and data flow of FPGA implementation.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

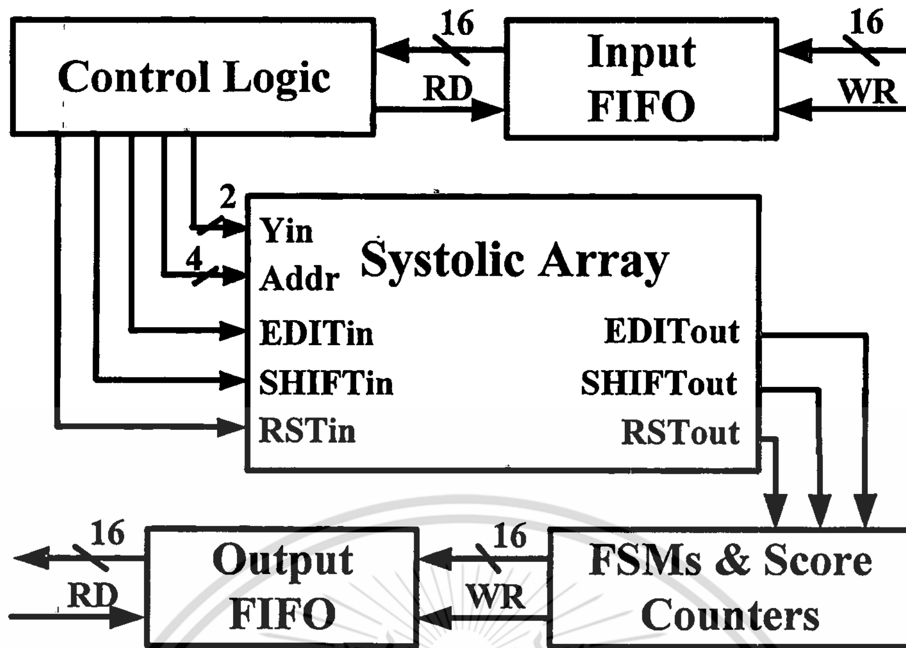


Figure 3.4: Block Diagram of the FPGA implementation of Multiple DNA Matching system

3.1.3.1 Systolic Array Description

Nucleotides are represented by a two-bit value as shown in Table 3.1. An important objective in the design for the hardware is to maximize the number of processing elements that fit in an FPGA.

Table 3.1 Two-bit Nucleotide Representation

A	T	G	C
00	01	10	11

If a query sequence cannot fit in one board, it must span either boards or configurations. To reduce hardware complexity, we pick the following costs, 1 for gap (c_{del}, c_{ins}), 2 for substitution (c_{sub}) if two characters are different or 0 for substitution if two character are same. The values of $d_{i,j-1,k}$ and $d_{i-1,j,k}$ in Eq.(3.1) are restricted to $d_{i-1,j-1,k} \pm 1$ and the equation can be simple to obtain [42]:

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

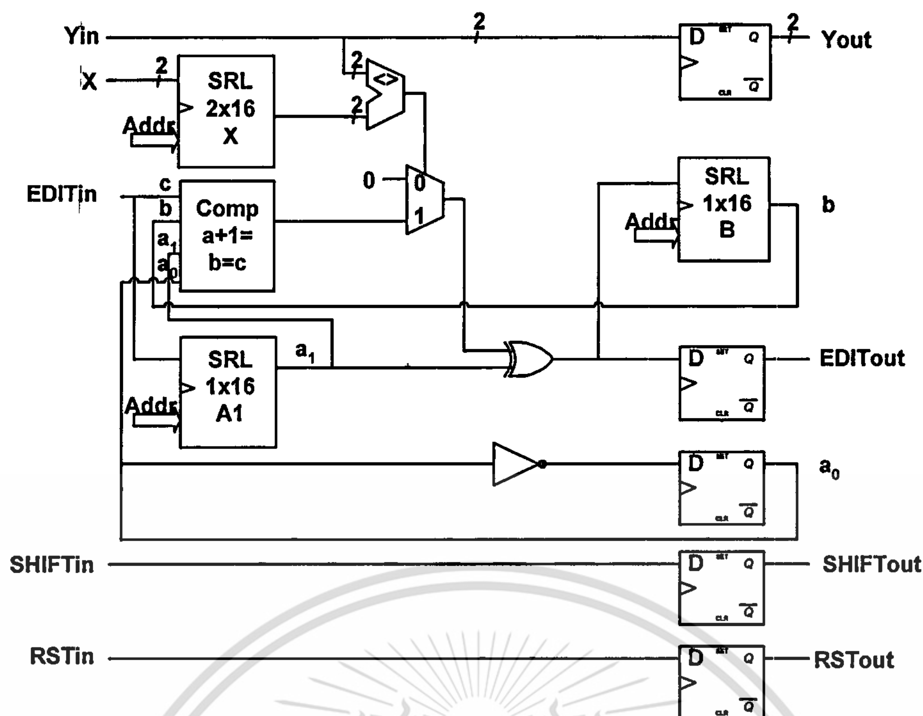


Figure 3.5: Microarchitecture of each processing element (PE) in Figure.3.3(b)

$$d_{i,j,k} = \begin{cases} a & \text{if } (b \text{ or } c) = a-1 \text{ or } (x_{i,k} = y_j) \\ a+2 & \text{if } (b \text{ and } c) = a+1 \text{ and } (x_{i,k} \neq y_j) \end{cases} \quad (3.5)$$

where

$$d = d_{i,j,k}$$

$$a = d_{i-1,j-1,k}$$

$$b = d_{i,j-1,k}$$

$$c = d_{i-1,j,k}$$

These penalty values reduce the amount of data that needs to be stored in the PE and sent to the next PE. Because the insertion and deletion penalty is one, the value of *a* differs from values of *b* and *c* by exactly one. The *d* differs from the *a* by either zero or two. Adjacent cells vary by exactly one, so the least significant bit can be inferred. The intermediate edit distances are computed modulo four, reducing storage space required for the matrix value in the processing elements. Because successive edit

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

distance values vary by exactly one, no information is lost if the values are stored modulo four.

The processing element (PE) shown in Figure.3.5 can compute Equation.(3.5). In each PE, a RAM-based shift register of size 16x2 bits is used to store each character of *q* query sequences X_k, $1 \leq k \leq q$. The value of *q* can be programmed at run time from 1 to 16 depending on input *Addr*.

In order to calculate *d*, inputs *a*, *b* and *c* should be available. *EDITin* is the new value of *c*. It is also stored in a 16x1-bit shift register *A1* and the output of this shift register is the current value of *a*. Value *b* is the output of 16x1-bit shift register *B*. After that, *d* will be stored in a flip-flop to be *c* for the next PE. At the same time, *d* is also stored in shift register *B*. When *SHIFTin* is active, each character of *Y* will pass through one PE, and *SHIFTin* is only active one time in *q* clock cycles. This is an advantage to decrease data transfer rate.

We use Verilog Hardware Description Language for the design. Following here is Verilog example code of some parts inside one PE, the code details are presented in Appendix B.

```
module PE(CLK,RSTin,Yin,EDITin,SHIFTin,Addr,EDITout,Yout,SHIFTout,RSTout);
    ...
    wire [1:0] Xint;
    SRL16E StoreX0(.Q(Xint[0]),.CLK(CLK),.CE(RSTin),.D(Xint[0],
                        .A0(Addr[0]),.A1(Addr[1]),.A2(Addr[2]),.A3(Addr[3]));
    FDRE Y0out(.Q(Yout[0]),.C(clk),.CE(SHIFTin),.D(Yin[0]),.R(RSTin));
    MUXCY MUX1(.DI(1'b0),.CI(Comp abc),.S(XvsY),.O(muxout));
    ...
endmodule
```

From the code, we can see all components of PE using primitive elements to reduce the area of each PE. The main improvement of our design is the use of Loop-Up Table (LUT) as RAM-based shift register (SRL16E) to store up to 16 one-bit values. Furthermore, the design is based on some optimizations for run-time customization [1], i.e. the length of queries and the initialization value of shift registers will be changed at run time. Thus, it occupies only 4.5 Xilinx Virtex-4 slices per PE.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

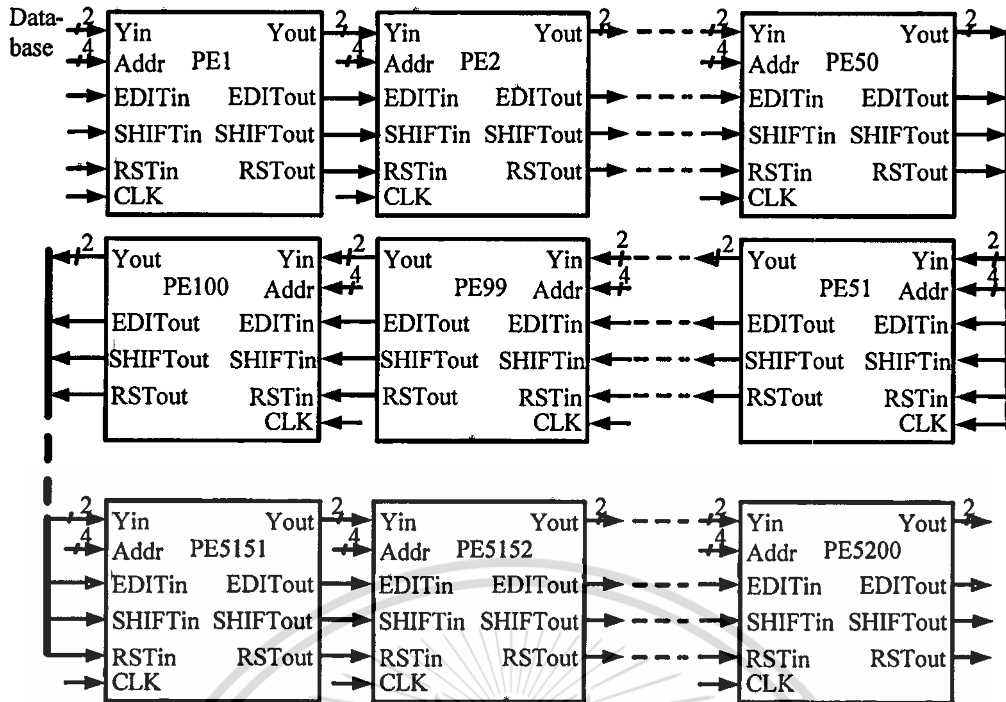


Figure 3.6: Our systolic array for multiple DNA matching, #PEs = m = 5, 200

The linear systolic array is placed in a serpentine pattern as shown in Figure.3.6. With the kind of this architecture, the place and route tools can take full advantage of area of chip in order to place as many PEs as possible.

For simulation, we used ModelSim SE II v5.8a. Figure.3.7 is simulation waveform of a two-PE array for multiple query searching. This waveform is used to check the correct of design with the number of queries be three. From the Figure.3.7, the value of database *Y* in is only change after three clock cycles when the *SHIFTout* signal is active. The result is the last signal EDIT out is taken sampled after the system is started two clock cycles equivalent the number of PE in system.

3.1.3.2 State Machines and Up-down Counters

The final processing element of systolic array sends its local edit distance value into state machines. The number of state machines equals the number of queries in system. Each state machine, in combination with one up-down counter, is responsible for converting the edit distance into the full edit distance value. Figure 3.8 shows the state machine used for this gene matching implementation.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

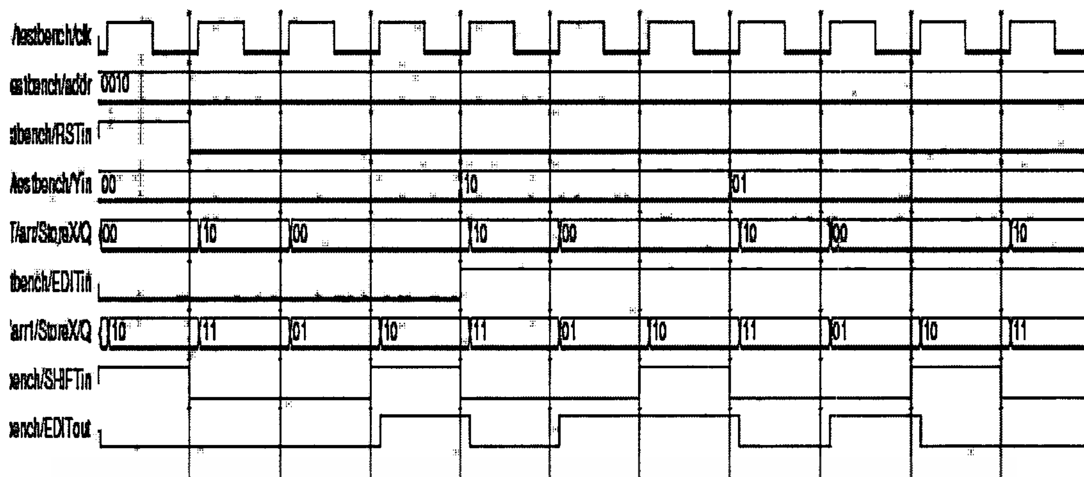


Figure 3.7: Simulation waveform of a two-PE array for Multiple DNA Matching

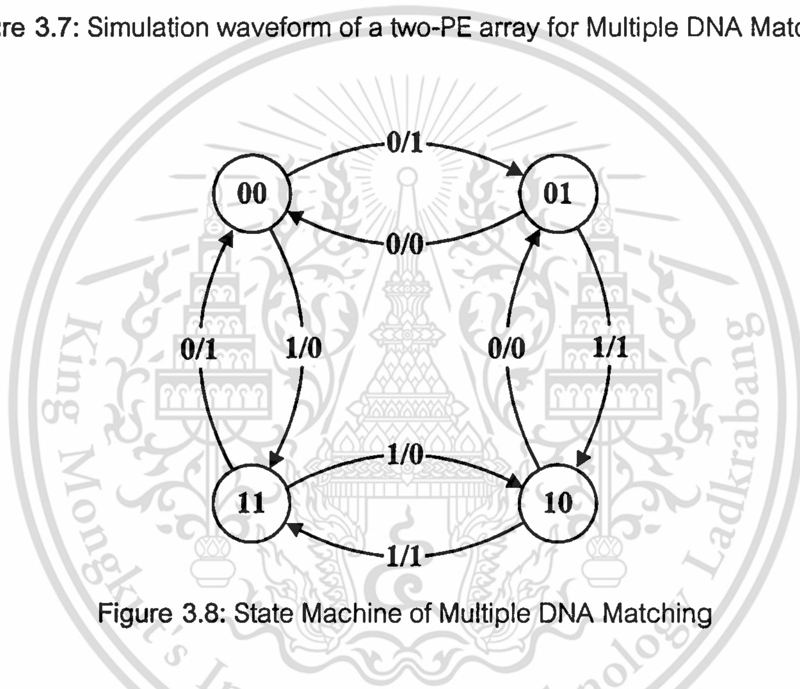


Figure 3.8: State Machine of Multiple DNA Matching

As shown in Figure 3.8, the state machine has four states. Each state represents the four possible intermediate edit distance values. The initial state depends on the size of the query sequence. The edit distance from the last processing element determines the next state of state machines. Only the upper bit of the edit distance is required because the lower bit is inferred. The outer loop in the figure sends an up signal while the inside loop sends the down signal. The up-down signal that is generated is passed to the next stage, the counter.

The counters are up-down counters that calculate the final edit distance value. Each counter receives an up-down signal from the state machine respectively, and

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

outputs the 16-bit edit distance. The initial value of it equals the number characters of query. The counter increases as up-down signal is one and decreases as up-down signal is zero.

3.1.3.3 Data Control

A control logic block was used for reducing the output data width of the input FIFO from 16 bits to 2 bits, so as to allow data to be fed into the systolic array. In addition, it initializes values for some signals of the system. The number of query sequences, q, will decide the values that the control block sends them to systolic array.

• The *RST* signal which resets the systolic array is activated when the system starts up and the new database comes in.

• The *SHIFT* signal which helps database characters pass through every PE is activated at every *q* clock cycles.

• And the *EDIT* signal which is the input value of the first PE in systolic array is toggled at every *2q* clock cycles.

3.1.3.4 The FIFO buffers

In order to use the resources of the FPGA efficiently, knowledge of the speed of the devices is required. Any communication between the FPGA board and the host must travel through the USB interface. An analysis of the design shows that the goal of making the user clock on the FPGA run approximately four times faster than the USB interface was met.

The system used input and output FIFOs constructed from Block RAM to buffer the sequence data transfer between the FPGA board and the host computer. These FIFOs reduce the amount of idle time in the systolic array, since computation time can be smaller than transfer time. The input and output data widths of the FIFOs are both 16 bits. The wide input data bus helps improve IO bandwidth from the host computer to the FIFOs. The clock used by the system is gated by the FIFO input empty flag, representing no valid data available, and the FIFO output full flag, representing no available space to pass data.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Figure 3.9: General Programmable Interface's (GPIF) Place in FX2 USB System

3.2 Firmware & Software Design for Multiple DNA Matching System

This section presents the design for communication between the FPGA board and the host PC. The FPGA board is ADS-XLX-V4-LX-EVL60 board of Avnet as Figure.3.1. This evaluation board has a Xilinx Virtex-4 XCVLX60 FPGA. It also includes a Cypress EZ-USB FX2 USB Microcontroller, CY7C68013-100AC.The EZ-USB FX2 device is a single-chip integrated USB 2.0 transceiver supports full-speed(12 Mbps) and high-speed (480 Mbps) modes. The details of this board are in appendix A.

The DNA Matching system on FPGA chip uses USB interface to connect with PC, so we design the firmware for FX2 USB chip in order to establish data transfer method; and the simple software on PC as a device driver and user interaction.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

3.2.1 Firmware Design

The FX2 interface is a programmable state machine that supports 8-bit or 16-bit parallel data transfers. This interface is called the General Programmable Interface (GPIF) as show in Figure 3.9. The GPIF is controlled by Waveform Descriptors that are created with the Cypress GPIFTool utility and downloaded to the FX2 over the USB cable. The GPIF descriptors are stored in internal RAM and are loaded by the firmware during the initialization process. The GPIF interface is made up of the signals which are connected to Virtex-4 FPGA.

In order to design the firmware, some basic concepts of USB standard are discussed.

USB standard is the half duplex transceiver including 4 types of transfers:

1. *Bulk Transfers*: Bulk data is bursty, traveling in packets of 8, 16, 32 or 64 bytes at full speed or 512 bytes at high speed. Bulk data has guaranteed accuracy; due to an automatic retry mechanism for erroneous data. The host schedules bulk packets when there is available bus time. Bulk transfers are typically used for printer, scanner, or modem data. Bulk data has built-in flow control provided by handshake packets.
2. *Interrupt Transfers*: Interrupt data is like bulk data; it can have packet sizes of 1 through 64 bytes at full speed or up to 1024 bytes at high speed. Interrupt endpoints have an associated polling interval that ensures they will be polled (receive an IN token) by the host on a regular basis.
3. *Isochronous Transfers*: Isochronous data is time-critical and used to stream data like audio and video. An isochronous packet may contain up to 1023 bytes at full speed, or up to 1024 bytes at high speed. Time of delivery is the most important requirement for isochronous data. In every USB frame, a certain amount of USB bandwidth is allocated to isochronous transfers. To lighten the overhead, isochronous transfers have no handshake (ACK/NAK/STALL/NYET), and no retries; error detection is limited to a 16-bit CRC.
4. *Control Transfers*: Control transfers configure and send commands to a device. Because they are so important, they employ the most extensive USB error checking. The host reserves a portion of each USB frame for Control transfers.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

USB data enters and exits FX2 via endpoint buffers. In order to keep up with the high-speed 480 megabit/second transfer rates, FX2 provides double, triple or quad buffering on its large endpoints (EP2, 4, 6, and 8). The CPU 8051 need not participate in high-bandwidth transfers. Instead, dedicated FX2 logic and unified endpoint/interface FIFOs move data on and off the chip at USB 2.0 rates without any CPU 8051 intervention.

The bulk transfer is chosen for designing our system because it is suitable for transferring large amount of data between host PC and the FPGA board. When compared with isochronous, it has NAK to correct data; and compared with interrupt transfer, it is not complicated. Moreover, it should be able to achieve higher throughput rate.

To apply data for computing components on FPGA chip, we get the data from host PC by Endpoint2, called FIFO-Write or OUT data. To collect the scores from FPGA back to host PC, Endpoint6 is used, called FIFO-Read or IN data. A mode of operation called Long Transfer Mode is established for data transfer in order to get the maximum throughput of bulk transfer. In this mode, one counter called the Transaction Counter is loaded with the desired number of transactions (1 to 4,294,967,296). When a FIFO-Read or FIFO-Write waveform is triggered on that FIFO, the GPIF will transfer the specified number of bytes automatically. So it can continuously transfer up to 4GByte data.

Figure 3.10 and 3.11 are the waveform designs of OUT and IN.

To perform a Write Transaction (OUT):

1. Set the action to write and choose the Endpoint2 as output buffer with the appropriate value for the FIFO which is to source the data.
2. Program the FX2 to detect completion of the transaction. As with all GPIF Transactions, one special bit (the DONE bit) signals when the Transaction is complete.
3. Program the FX2 to commit (pass-on) the data from the endpoint to the FIFO. The data can be transferred by AUTOOUT method: CPU is not in the data path; the FX2 automatically commits data from the USB to the FIFO Data bus if the full flag from input FIFO of FPGA chip is not active.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

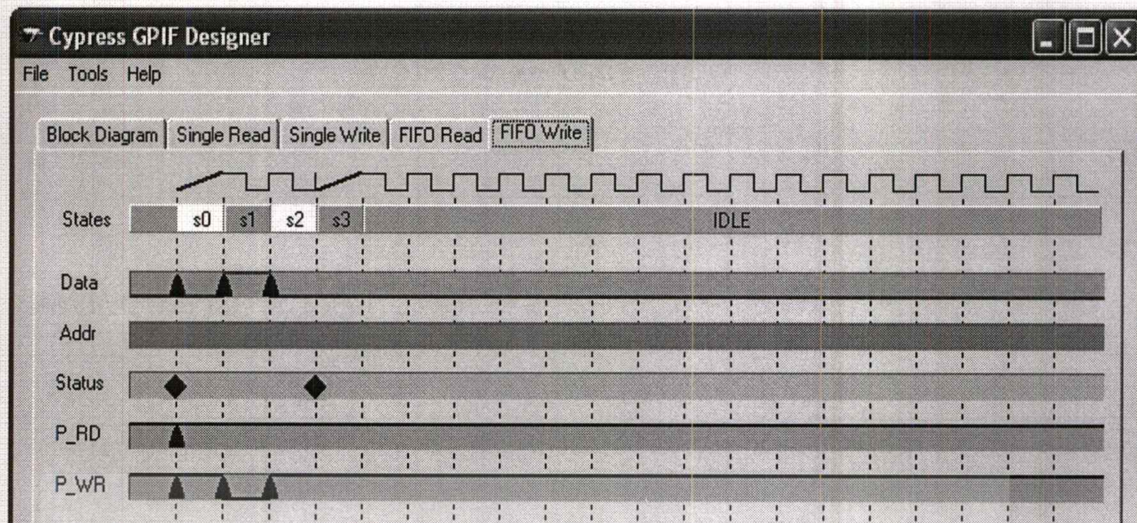


Figure 3.10: GPIF's write (OUT) waveform design of Cypress FX2 USB

To perform a Read Transaction (IN):

1. Set the action to read and choose the Endpoint6 as input buffer with the appropriate value for the FIFO which is to receive the data.
2. Program the FX2 to detect completion of the transaction.
3. Program the FX2 to commit (pass-on) the data from the FIFO to the endpoint. The data can be transferred by AUTOIN method: CPU is not in the data path; the FX2 automatically commits data from the FIFO Data bus to the USB if the empty flag from output FIFO of FPGA chip is not active.

One important note is that the USB standard is half duplex, i.e. at once time, it can not transfer and receive data concurrently. Therefore, the sizes of the input and output FIFOs in FPGA chip are big enough so that data streams for computing are not interrupted.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

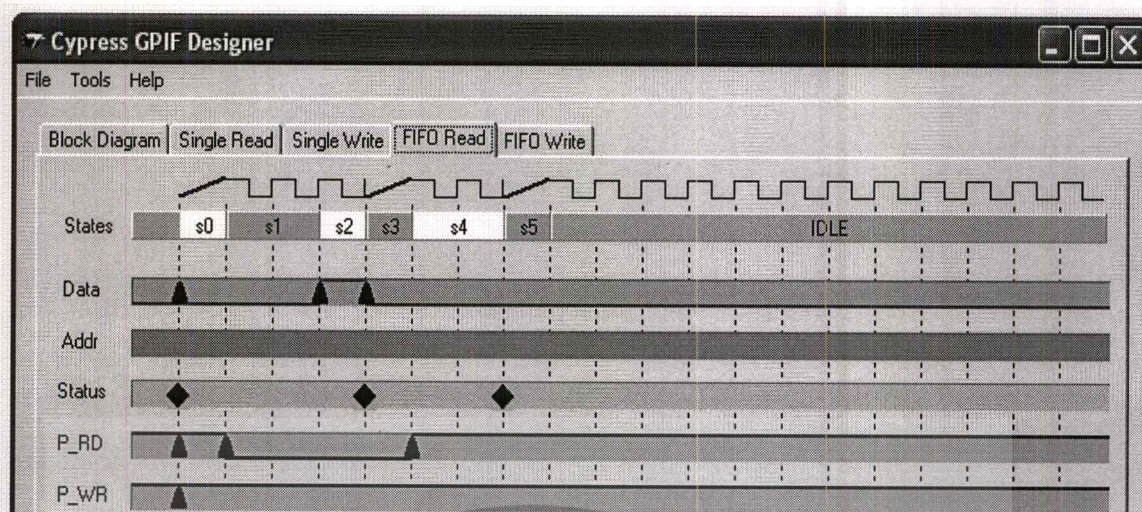


Figure 3.11: GPIF's read (IN) waveform design of Cypress FX2 USB

3.2.2 Software Design

To interface with user, a simple program using C++ is created. Its main function is a device driver that can be used to interface with FX2 EZ-USB. The driver provides a user mode interface to FX2 USB device requests and data transfers. It aids the device or firmware developer. It allows user to test FX2 devices ability to perform standard USB device requests and data transfers. Device drivers typically run in a privileged execution mode called kernel mode.

All USB devices have a Vendor ID (VID) and a Product ID (PID) which are reported to Windows in the device descriptor. Windows uses the VID and PID to find the appropriate device driver.

User mode access to the FX2 USB device driver is through I/O Control calls. A user mode application first gets a handle to the device driver via a call to the Win32 function **CreateFile()**. The user mode application then uses the Win32 function **DeviceIoControl()** to submit an I/O control code and related input and output buffers to the driver through the handle returned by **CreateFile()**.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

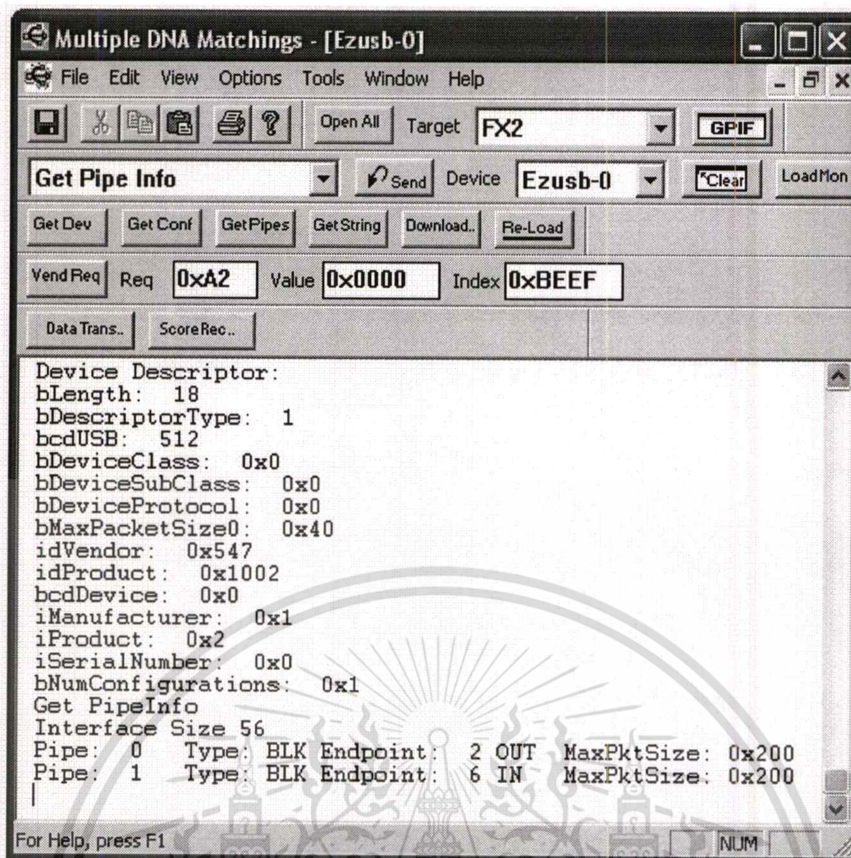


Figure 3.12: User Interface of Multiple DNA Matching System

Our system interface is shown in Figure 3.12. Some of the main functions of user interface are:

- Get Dev: Causes the driver to issue the USB standard device request GET_DESCRIPTOR of type DEVICE to the device.
- Get Conf: Causes the driver to issue the USB standard device request GET_DESCRIPTOR of type CONFIGURATION to the device. This request returns not only the configuration descriptor, but also interface, endpoint and class specific descriptors.
- Get Pipes: Returns an Interface Information structure describing the pipes of the currently selected interface and alternate setting.
- Vend Req: Sends a Vendor or Class specific request to the control endpoint.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

- Download: Downloads data to EZ-USB RAM starting at the specified address. This IOCTL will only download to the EZ-USBs internal RAM.

After configuration FX2 USB, the system can begin to transfer data for computation and receive the scores after finishing.

- Data Transfer: Writes database to the ENDPOINT2 OUT bulk pipe. This IOCTL will block the calling thread until the data transfer completes.
- Score Receive: Reads Scores from the ENDPOINT6 IN bulk or interrupt pipe. This IOCTL will block the calling thread until the data transfer completes.

3.3 Results and Comparison

This section presents the results for the Multiple DNA Matching implementation using FPGA device connected with PC host by USB interface. The first subsection discusses the theoretical results of this implementation and compares the results with other hardware sequence matching systems. The next subsection analyzes the actual results implemented on Avnet board and evaluates the performance of real system.

3.3.1 FPGA Throughput

Our system is synthesized by Xilinx Foundation ISE 6.3i. Then, it is implemented on a Xilinx Virtex-4 XCVLX60 FPGA which has 26,624 slices. Totally, we can fit 5,200 PEs in the Virtex-4 XCVLX60 FPGA chip, the rest area for interface and data control. Our maximum frequency is 195 MHz, reported by Xilinx Timing Analyzer.

Table.3.2 compares our design with systems using Xilinx FPGA chip. Splash 2's [4] system has over a decade old; the result, however, has still been very interesting. It contains 14 PEs in one XC4010 FPGA chip. And the whole system includes 16 boards with totally 272 chips. Both the HokieGene [1] and S&W Cell [2] were the latest reported sequence alignment systems using the edit distance algorithm in recent years. As can be seen from the table, the throughput of HokieGene design is the greatest because it used an XCV6000-4 (33,792 slices) which is bigger than our chip. And the most saving in area is S&W Cell design; it occupies only 3 slices/PE of Virtex Pro XCV1000E (12,288 slices).

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Table 3.2 Comparison of FPGA-based DNA search systems

System	Splash2[4]	Hokiegene[1]	S&W Cell[2]	Ours
# queries (q)	1	1	1	16
#slices/PE	16	4	3	4.5
#slices/char	16	4	3	4.5/16
#PEs/chip	14	7,000	4,032	5,200
#chips/System	272	1	1	1
Model	XC4010	XC2V6000-4	XCV1000E-6	XC4VLX60-10
f(MHz)	N/A	180	202	195
Throughput (BCUPS)	43	1,260	814	1,014
$rate_{data}$ (Mbps)	N/A	360.0	404.0	24.4

All of previous systems only content one character inside one PE. And their systems compare one query sentence against data sentence. Our system can flexibility compare from 1 to 16 queries simultaneously. Our throughput is 1,014 BCUPS, it is comparable with others. Moreover, the number of hardware unit for a character of query sentence is very small; and if comparison up to 16 queries, it is 4.5/16, much lower than others.

For more exact comparison, our system is also implemented on various kinds of Xilinx FPGA chips that are used by recent other references. When our system is implemented on XC2V6000-4 with 33,792 slices, the number of slices per PE is the same as the original design on Virtex-4. The frequency decreases slightly. The remarkable feature is that we can fit approximately the same number of PEs as Hokiegene although our system is more complex than Hokiegene. This allows our throughput approximately the same as Hokiegene's. Table 3.3 shows the comparison with Hokiegene.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Table 3.3 Comparison of our Multiple DNA Matching system with Hokiegene [1] on XC2V6000 FPGA chip

System	No. queries	#Slices /PE	#Slices /char	#PEs /Chip	Freq. (MHz)	Throughput (BCUPS)	$rate_{data}$ (Mbps)
Hokiegene [1]	1	4	4	7,000	180	1,260	360.0
Ours	16	4.5	4.5/16	6,720	176	1,182	22.0

When our system is implemented on XCV1000-6 with 12,288 slices, the number of slices per PE is 5. We can only fit 2400 PEs in chip. The frequency drops down to 167 MHz. The reason of the disappointing features is that our system is placed and routed automatically by Xilinx tools. Conversely, the authors of S&W Cell [2] place and route their design manually. Therefore, they can fit a large number of PEs in XCV1000-6 chip. Their method, however, wastes design time and depends on specific chip. Table 3.4 shows the comparison with S&W Cell.

Table 3.4 Comparison of our Multiple DNA Matching system with S&W Cell [2] on XCV1000-6 FPGA chip

System	No. queries	#Slices /PE	#Slices /char	#PEs /Chip	Freq. (MHz)	Throughput (BCUPS)	$rate_{data}$ (Mbps)
S&W Cell [2]	1	3	3	4,032	202	814	404.0
Ours	16	5	5/16	2,400	167	400	20.9

One more important thing is that most of the previous systems take only 1 clock cycle to compare in every PE, so the database transfer rate ($rate_{data}$) requested for computation is very high. The database request rate is calculated by equation 3.6

$$rate_{data} = [f_{max} \times E] / q \qquad (3.6)$$

where

f_{max} the system clock frequency

E the number of bits to encode each character

q the number of query sequences.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

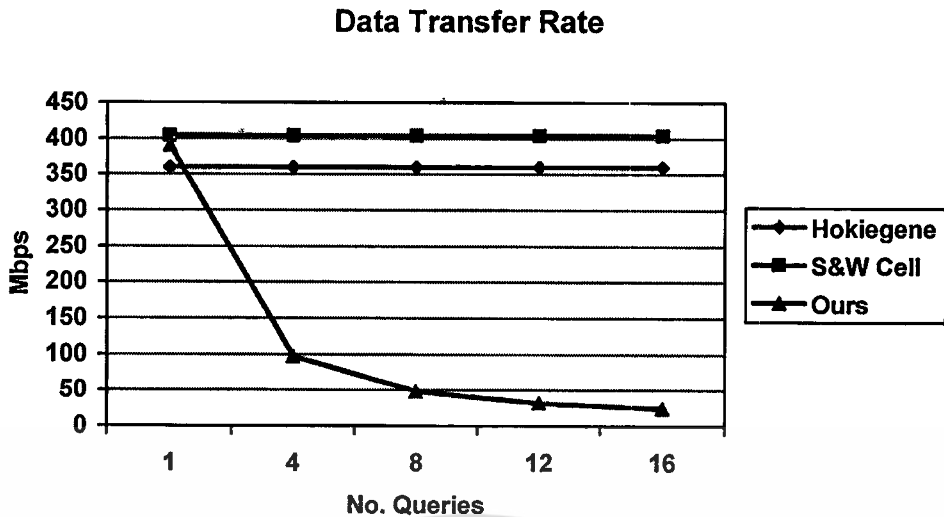


Figure 3.13: Data transfer rate of our Multiple DNA Matching system compares with the other FPGA-based systems when the number of query sequences varies from 1 to 16

For example, the database transfer rate of S&W Cell [2]

$$SWCell_rate_{data} = [f_{max}xE]/q = [202x2]/1 = 404\ Mbps$$

With database transfer rate of 404 Mbps, the system needs expensive FPGA(s) with high-speed interfaces. Nevertheless, the delay of communication can occur everywhere in the host computer, e.g. harddrive speed and I/O interface. Thus, in reality, it can hardly run at the maximum throughput. Meanwhile, our database transfer rate is approximately one-sixteenth of others when the number of query sequences is 16.

$$Our_rate_{data} = [f_{max}xE]/q = [195x2]/16 = 24.375\ Mbps$$

With the number of query sequences varies from 1-16, we can plot the $rate_{data}$ in Figure 3.13. Moreover, with database transfer rate being only 24.375 Mbps, we can use USB 2.0 standard to connect very low-cost FPGA board with a host computer as shown in Figure.3.1.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

3.3.2 Performance Analysis of Multiple DNA Matching System

Our implementation was successfully verified on the AVNet ADS-XLX-V4-LX-EVL60 board which provides a 16-bit wide USB 2.0 interface. The systolic array and all other logic of the implementation operate at the 100 MHz clock. The speed of FX2 USB is 48 MHz with 16-bits wide interface. The host PC is a Pentium IV 1.8 GHz processor with 512MB of RAM.

Table 3.5 Actual computation time of Multiple DNA Matching system with operating frequency at 100 MHz

Data-base	Computation Time (seconds)									
	#q = 1		#q =2		#q =4		#q =8		#q =16	
# chars	theory	real	theory	real	theory	real	theory	real	theory	real
8×10^4	0.0009	0.016	0.0017	0.016	0.0034	0.016	0.0068	0.016	0.0136	0.015
8×10^5	0.008	0.094	0.016	0.094	0.032	0.095	0.064	0.110	0.129	0.139
8×10^6	0.080	0.235	0.160	0.250	0.320	0.390	0.640	0.652	1.281	1.292
8×10^7	0.800	2.453	1.600	2.406	3.200	3.532	6.400	6.530	12.801	13.094
8×10^8	8.000	21.109	16.000	25.156	32.000	35.609	64.000	64.219	128.000	129.620

Table 3.5 shows the results compared between theoretical time and real time when the board run at 100 MHz speed. The numbers of queries that can compare simultaneously in systolic array are 1, 2, 4, 8, 16 queries. The length of each query is 5,200 characters. The sample queries are compared to different databases of 8×10^4, 8×10^5, 8×10^6, 8×10^7, and 8×10^8 characters. The theoretical time in table can be calculated by following the equation.

$$t_{exec} = (l_{data} + l_q - 1) \times q / f \qquad (3.7)$$

where

t_{exec} time to compute edit distance of a database,

l_{data} size of the database sequence,

l_q size of the query sequences,

q the number of query sequences,

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

f the frequency of FPGA.

When the size of database sequence is much greater than the size of query sequences, equation 3.7 can be reduced as equation 3.8.

$$t_{exec} = (l_{data} \times q) / f \qquad (3.8)$$

In Table 3.6, it shows the performance between the actual throughput and theoretical throughput. When the number of queries is 1, the system only runs around 35% of the theoretical computation. When the number of queries is 2, the performance increases to approximately 65%. And when the number of queries is from 4-16, it can reach to 82%-99%. These results, once more, prove that data rate is the bottleneck of most of systems.

Table 3.6 Performance of Multiple DNA Matching system with operating frequency at 100 MHz

Database	**Actual Performance (%)**				
# chars	**#q = 1**	**#q =2**	**#q =4**	**#q =8**	**#q =16**
8×10^4	5.32	10.65	21.30	42.60	90.88
8×10^5	8.57	17.13	33.90	58.56	92.68
8×10^6	34.06	64.04	82.10	98.22	99.14
8×10^7	32.62	66.50	90.61	98.02	97.76
8×10^8	37.90	63.60	89.87	99.66	98.75
Average Throughput (Mbps)	123.21	230.81	330.49	412.94	498.38

To verify the influence of database sizes on performance, the graph in Figure 3.14 shows the actual performance of our system with variety of database sizes. The actual performance is the ratio between real computation time and theoretical computation time. When the size of databases is small, a big portion of real computation time is spent on setting up. Hence, the performance is low. The performance is stable as the size of database is from 8×10^6 characters, i.e. approximate 2MByte. However, when the number of query is 16, the performance is very high regardless of the size of databases. And average throughput is nearly 500 BCUPS.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

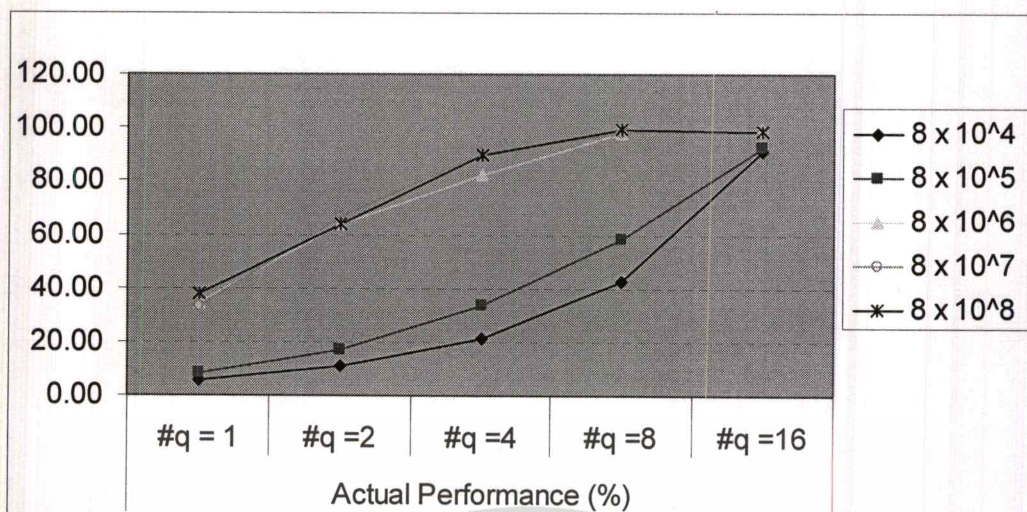


Figure 3.14: Actual performance of Multiple DNA Matching system with difference database sizes

To compare run time with some software-based systems that mentioned in chapter 2, our system is implemented with 10 queries. The length of each query is 400 characters, where the database contains 800 millions characters. The run time of our system is 81.20s. The table 3.7 shows the comparison between our system and other software-based implementations. When compared with the software-based implementation of Smith & Waterman algorithm, our system is remarkable; the speed-up is over 857 times. With the fastest software, MegaBalst, the speed-up is 3.36 times.

Table 3.7 Comparison of Multiple DNA Matching system with software-based implementations

System	MegaBlast	Blast	Fasta	S&W	Ours
Run Time (s)	273.00	534.00	4,462.00	69,667.00	81.20
Speed-up	3.36	6.58	54.95	857.94	1.00

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 4

Deep Packet Filtering in NIDS

In previous chapter, systolic design is applied for DNA similarity search, one kind of approximate string matching. Moreover, this method can also be applied efficiently for exact string matching. Especially in deep packet filtering of NIDS, it helps NIDS check all of incoming packet at network rate. In the first section, design methodology for deep packet filtering of NIDS is proposed. Next section, we present the result of our system on FPGA Virtex-4 and compare with the recently FPGA implementations of NIDS.

4.1 Design Methodology for Deep Packet Filtering in NIDS

Most of previous systems mentioned in chapter 2 compare all of rules in rule set of NIDS system with incoming packets concurrently. Because there are thousands of rules in NIDS, this broadcast method makes the significant increase of fan-out in circuit. The big fan-out affects to frequency of circuit so the performance of whole system decreases. Our system is application of systolic design for string matching. Systolic method has deeper pipeline, and the data dependence is local. Therefore, the system has high frequency and still keeps the high throughput. The performance of whole system is improved significantly.

An architectural overview of our system is shown in Figure.4.1. The system consists of three blocks. A Match Processor Array is the main part of system that stores rules used to compare with incoming packet. An Address Calculation Logic calculates the address of the rules that cause a match. And a control unit controls data flow of system.

The system receives packets from the network in a stream and outputs the address of any signature that is found in the packet. A managing network processor or CPU can use this information to raise a network alert or attempt to terminate the offending connection.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

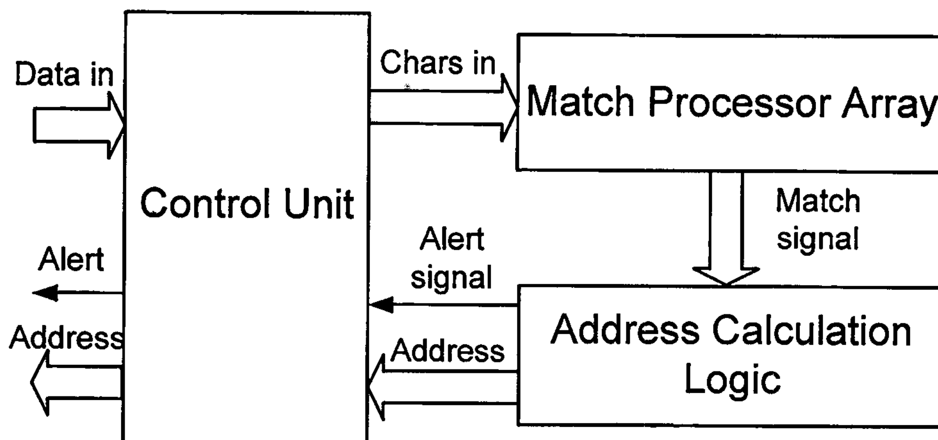


Figure 4.1: Overview of Deep Packet Filtering in NIDS system

4.1.1 Match Processor Array

In this section, we present a novel systolic processor array that uses an array of processing elements to match characters going through. In the latest Snort rules, there are 55 distinct lengths distributed from 1 byte to 122 bytes. For each length, there could be as many as 700 strings or as few as only 1 string. Totally, there are 21,830 bytes in 2,064 rules of Snorts rule set. However, during the analysis, we found that some of the rules look for the same string patterns but with different headers. By combining all the rules with the same payload content, we can eliminate duplicate patterns. Through simple pre-processing, we reduced the number of rules from 2,064 down to 1,519 rules which contain unique patterns.

All of rules are arranged in one linear array by sequential. Let m_1, m_2, ..., m_k be the numbers of character in rule *1, 2, ..., k,* respectively, we have an $1 \times n$ array of processing elements (PEs) where n is the number of characters in the rule set to be matched against. Thus, each PE represents one of the characters in the rule set. So $n = m_1 + m_2 + ... + m_k$, and Match Processor Array is presented in Figure 4.2.

In the Figure 4.2, each Processing Element (PE) contents one character of rule set. When string of characters comes in each PE, it will compare with the character inside the PE. If two characters are the same, the signal match equals 1, and is transferred to the next PE in current rule after delay one clock cycle. When the last PE of current rule have active match signal, that mean current string (packet) has substring same with the rule.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

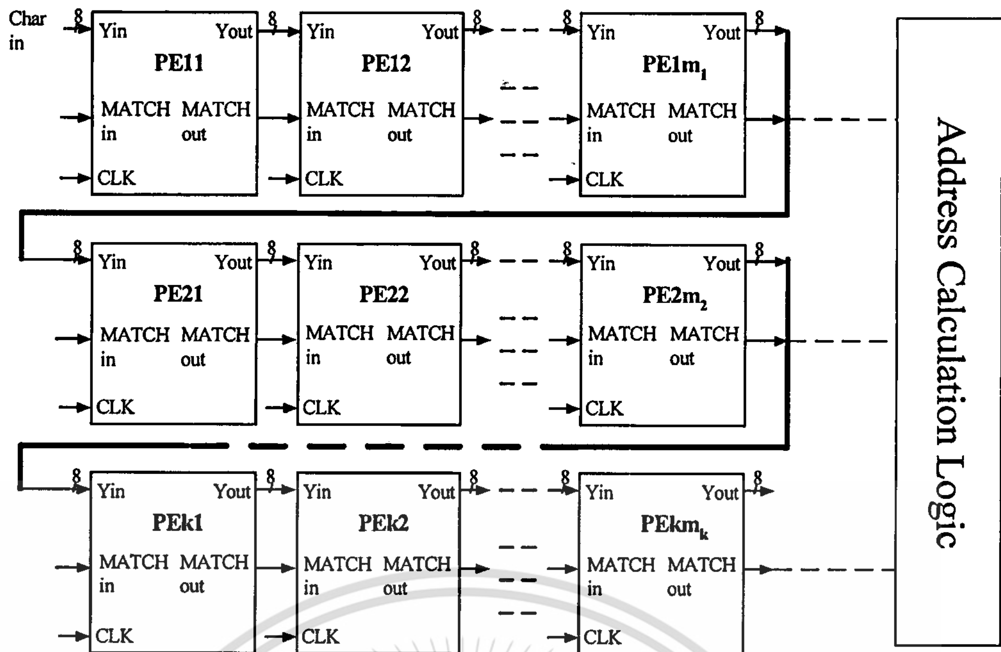


Figure 4.2: Match Processor Array

For example, if the input string is *AABC* and the current rule includes *ABC*, the match signal output in last PE of current rule equals 1 after 6 clock cycles since the first character of input string has come in to the first PE of current rule as the Figure 4.3.

	A (PE1)	B (PE2)	C (PE3)
CLK1	A(1)	-	-
CLK2	**A(1)**	A(0)	-
CLK3	B(0)	A(0)	A(0)
CLK4	C(0)	**B(1)**	A(0)
CLK5	-	C(0)	B(0)
CLK6	-	-	**C(1)**

Figure 4.3: Example of Match Processor Array

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

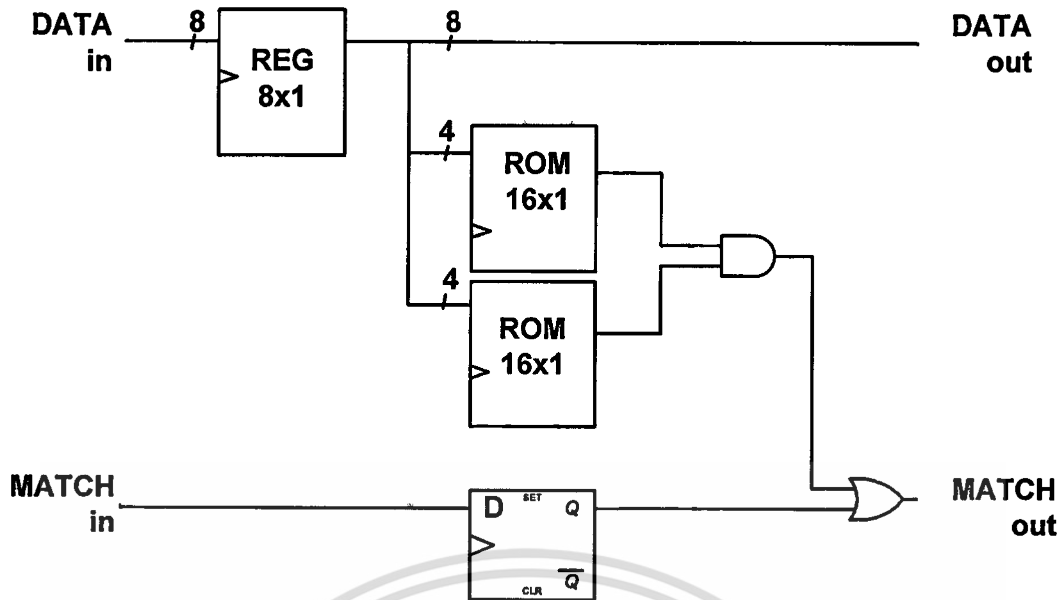


Figure 4.4: MicroArchitecture of a Processing Element in Match Processor Array

Inside each PE as the Figure 4.4, an 8-bit register (REG8x1) is used to transfer each incoming character. In Figure 4.4, ROM is used to store the character of PE. A PE includes 2 ROMs 16x1bit. ROM1 stores the least significant 4-bits of character, and ROM2 stores the most significant 4-bits of character. For example, if we want to store the character A which ASCII code is 41H, we store the value 1 in position 1 of ROM1, and the value 1 in position 4 of ROM2. All of the other values of 2 ROMs store 0. It means that as 1 character needs to be compared come in if it is character A (41H), 2 ROM will have output equal to 1, so the output of AND gate equals to 1, and others case it equals to 0. With this design, we do not need the comparators as the traditional designs, so we can save a lot area of chip. Before the match signal goes out the PE, we use a Flip Flop which delays it one clock cycle to wait for the next incoming character as Figure 4.4.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

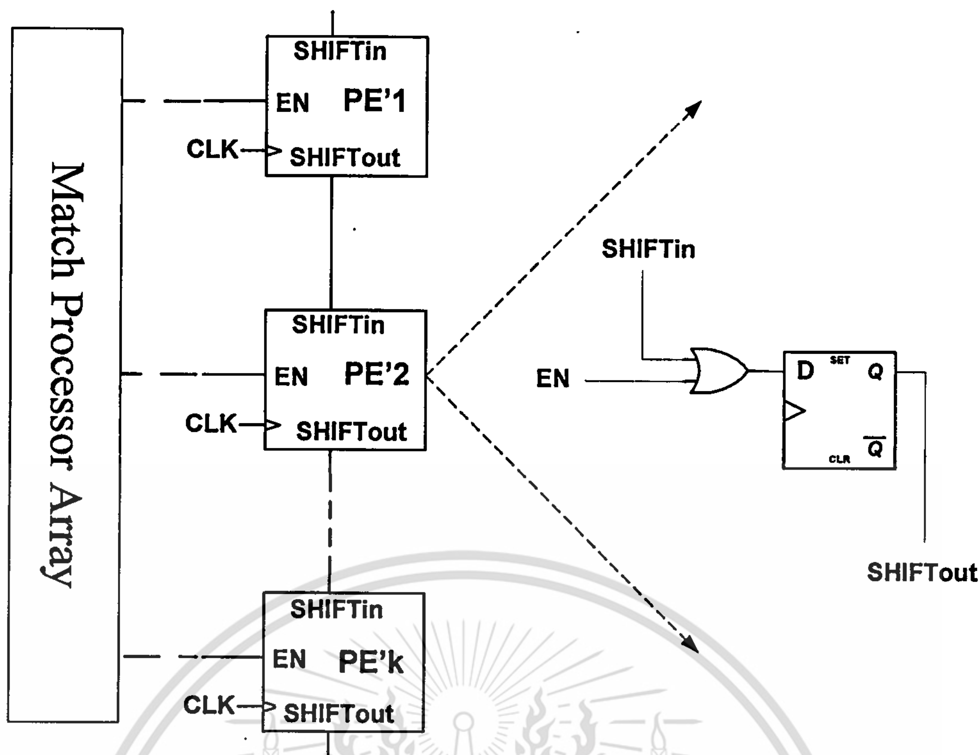


Figure 4.5: Match Shift Array in Address Calculation Logic

4.1.2 Address Calculation Logic

The Match Processor Array outputs a match signal indicating that a match is found. However, for the system to be useful in the context of NIDS, we also need to know which signature causes the match. The Address Calculation Logic is used to find rules causing a match. In order to generate these addresses, we use the match signals generated by the Match Processor Array.

The Address Calculation Logic includes two parts. A Match Shift Array is used to propagate the match signal. A block RAM and a Counter are used to calculate the address of rules.

The Match Shift Array as Figure 4.5 uses same clock source with the Match Processor Array. When the match signal of any rule from the Match Processor Array is active, it will transfer to respective PE in the Match Shift Array. Inside of each PE of the Match Shift Array as Figure 4.5, the SHIFTout is active and propagates this match signal to the end of array.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

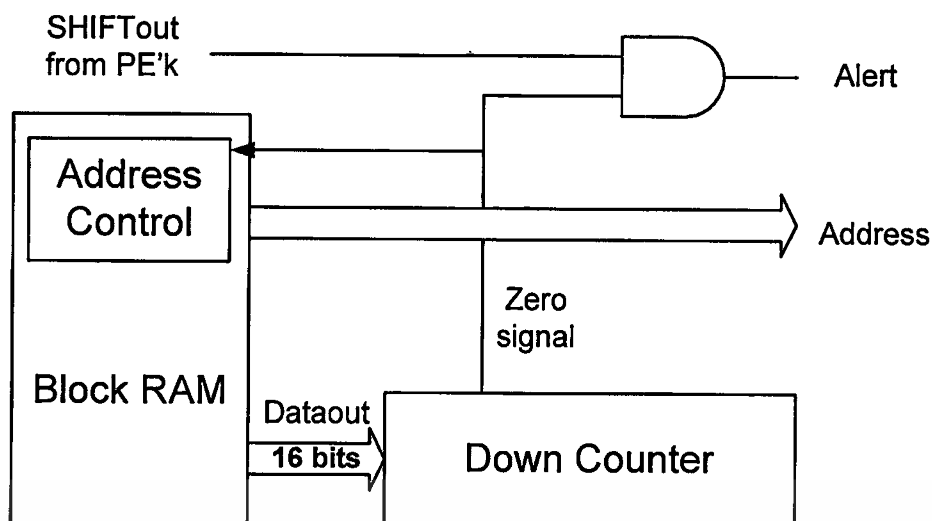


Figure 4.6: The Block RAM and the Down Counter for calculating the address of the matched rule

In Figure 4.6, a block RAM is used to store values for calculating the address of the matched rule. Position *i* in RAM stores a value which equals to the sum of numbers of PEs from the beginning of the Match Processor Array to the last character of rule *i* and the inversed position of rule, *n-i*.

A counter gets the value of RAM to count down. When the counter reaches value 0, the zero signal will be active. The zero signal combines with the signal match (SHIFTout) from Match Shift Array in order to generate the alert signal, and the address of matched rule is the current value of RAM address. Besides, the zero signal also makes the address of RAM increase in the next clock cycle.

4.1.3 Control Unit

A control circuit manages the data flow through the system and also manages flow control of the incoming packet. When an incoming packet is ready to be delivered, the control circuit first resets the Match Processor Array and also resets the address of RAM. The control circuit then takes each byte in the incoming packet and presents it to the Match Processor Array on every clock cycle. When the last character in the packet has arrived, the address of block RAM in Address Calculation Logic is started.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

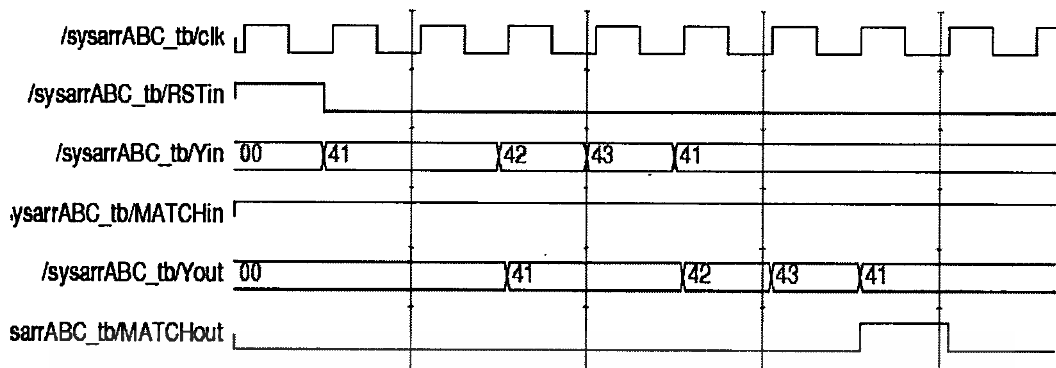


Figure 4.7: Simulation waveform of comparison rule ABC with AABC string

4.2 Result of our Deep Packet Filtering in NIDS system

Our design was coded by Verilog HDL and Xilinx's ISE 6.3i. The design environment was used for all parts of the design flow including synthesis, mapping, and place and route. For simulation, we use ModelSim 5.8a tool, the Figure 4.7 is the waveform simulation of rule *ABC* compare to *AABC* string.

The target chip is Virtex4 XC4VLX60. Totally, we can fit 11,139 characters in the Virtex XCVLX60 FPGA chip. And the maximum frequency reported by Xilinx Timing Analyzer is 343 MHz. The throughput of a design is calculated by multiplying the clock frequency with the data width (8-bits). For a design running at 343 MHz, the throughput is 2.74 Gigabits per second (Gbps).

The latency of system is determined by multiplying the number of characters in system with frequency. With 343 MHz in frequency, the latency of system is 32.5 μs. However, due to the pipelined design, the output overhead can be hidden with a sequence of packets because the input processing for a packet can begin as soon as the matching stage for the previous packet is complete and the match results have been sent to the output stage.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

4.2.1 Comparison of our Deep Packet Filtering in NIDS System with Previous Works

This section compares the results of our Deep Packet Filtering in NIDS system with the results of previous works on FPGA pattern matchers for network security applications.

The common metrics used for comparison are *throughput* and *capacity*. The throughput (in Gigabits per second) of a design is calculated by multiplying the amount of input data (in bits) processed per clock cycle by the maximum clock frequency (in Gigahertz). The capacity of a design is the number of characters that can be programmed into a given FPGA device.

Table 4.1 Comparison of FPGA-based systems for NIDS

System	Device	Max Freq (MHz)	Throughput (Gbits/s)	No. characters	Logic cells/char	Performance (Mb/s/cell)
Moscola et al. [6] (Quad DFA)	VirtexE 2000	37	1.18	.420	34.90	34
Hutchings et al. [5] (NFA)	VirtexE -2000	49.5	0.4	16,028	2.5	160
Gokhale et al. [11] (CAM based)	VirtexE -1000	69	2.2	640	15.2	145
Cho et al. [8] (Discrete comparators)	EP20K	90	2.88	1,611	10.55	273
Sourdis et al. [12] (CAM based)	Virtex2 6000	252	8.06	2,457	19.41	415
Our System (Systolic based)	**VirtexE 2000**	**182**	**1.45**	**7,820**	**4.91**	**295**
	Virtex2 6000	**302**	**2.41**	**14,318**	**4.72**	**510**
	Virtex4 VLX60	**343**	**2.74**	**11,139**	**4.78**	**573**

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

For comparison purposes, a device-neutral metric called logic cells per character (LCs/char) is used. This metric is determined by dividing the total number of logic cells used in a design by the sum of the lengths of all patterns programmed into the design. A logic cell is the fundamental element of both Alteras and Xilinxs devices, and therefore, it is the most proper measure for evaluating the area cost of a design.

Table.4.1 shows the comparison of our work with other recent related work. To be easier in comparison, we also implemented the system on other FPGA chips as VirtexE XCV2000E and Virtex2 XC2V6000. In Table.4.1, the throughput of [12] is 8.06 Gbps, from three to twenty times compared to the others. That's because they use up to 4 pipeline duplicate comparators to process 4 characters per cycle. It is the advantage and also the disadvantage of design; the throughput increases 4 times and the area also increases 4 times. So its logic cells/chars is 19.41.

The implementation of [8] is closer with [12]. It can also compare 4 characters of input packets per clock cycle. However, the system broadcasts data and uses 4 parallel comparators for every rule. Therefore, its frequency is not too high and the throughput is about 2.88 Gbps.

Our system has the highest frequency, 182 MHz – 343 MHz, depending on the kinds of FPGA chips implemented. Because, our system is completely pipelining and the data dependence is mode local. These techniques help our system avoid the high fan-out when matching thousands of rules at the same time. Our throughput is approximately 2.5 Gbps since our system compares only one character per clock cycles.

One other interesting implementation is NFA of Hutchings el al. It occupies only 2.5 logic cells/char. That's because decoded NFA's have very low area cost, due to the centralized decoders, which allow excellent character sharing. It can content up to 16,028 characters of rules. However, the throughput is the weak point of system. Meanwhile, our logic cells/char is medium. With bigger chip, the portion of main parts increases and the portion of interface parts decreases. Thus, our logic cells/char is only 4.72 in XC2V6000.

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

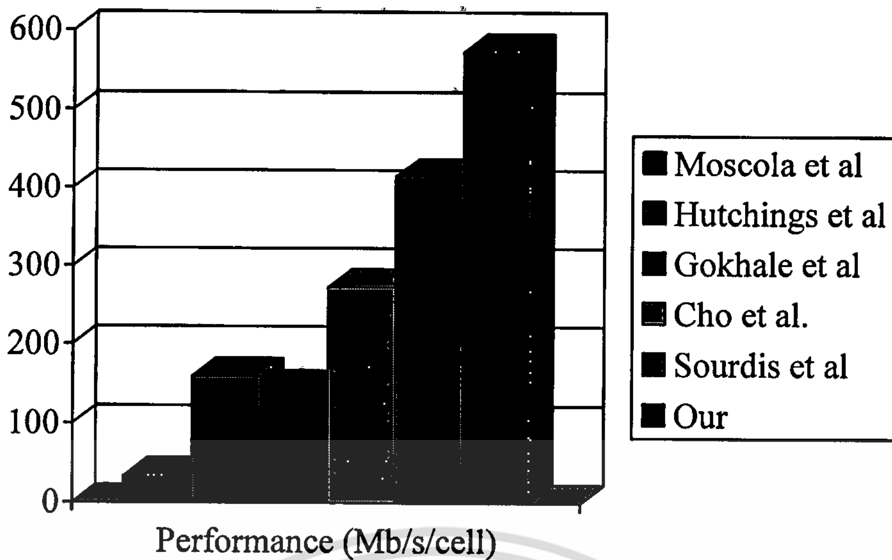


Figure 4.8: Performance Comparison of FPGA-based Systems for NIDS

Since throughput and capacity generally have an inverse relationship, any comparison of designs must consider both metrics. A metric called *performance* is defined as throughput times density. Density is the character density, or the reciprocal of LogicCells/char. Performance increases as throughput and density increase and decreases as throughput and density decrease. Therefore, a design with higher performance provides a better tradeoff between throughput and area, or, in other words, a smaller increase in area as throughput is increased.

In Table 4.1, the performance of the systems implemented by CAM based and Discrete Comparator is better than the performance of DFA/NFA systems. And with the same kinds of chip, our system is always better than the others. On VirtexE, our performance is 295, while the maximum one of other systems is only 160. On Virtex2, our performance is 510 compared with the performance of [12] that is 415. And on the newest FPGA chip, Virtex4, ours is 573, the highest performance as Figure 4.8.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

This thesis presents the design and implementation of high throughput string matching developed on a hardware environment of FPGA boards. These implementations can apply for many applications that use exact or approximate string matching. In this thesis, we proposed it for two fields, bioinformatics and Network Intrusion Detection System.

In bioinformatics, a novel linear systolic processor array architecture implementation on FPGA called Multiple DNA Matching is proposed. It can compare simultaneously up to 16 DNA queries against a large DNA database. The main contribution of this work is the deeper pipeline in systolic array that makes the data rate lower than other systems up to sixteen folds. According to our implementation result on a Xilinx Virtex-4 XC4VLX60 FPGA, the throughput is 1,014 billion cell updates per second with the data rate only 24.375 Mbps. Our system is verified on Avnet Virtex-4 board that runs at 100 MHz using USB 2.0 interface. The actual throughput can get approximately 500 billion cell updates per second, the achievable performance is 96% compared with theoretical throughput at the same frequency.

A new SRAM-base FPGA implementation of Network Intrusion Detection is also proposed. We have used systolic processor technique to design. According to our implementation result, the achievable throughput can be up to 2.74 Gbits/s. This is sufficient to handle intrusion detection on current gigabit networks. And the throughput/area of our system is the best compared with the recent designs.

5.2 Future Works

The multiple DNA matching system developed and presented in this thesis computes the global edit distance for DNA sequences. The design can be modified to compute edit distances for proteins. Proteins have 20 characters so it requires five bits to encode for each character. Therefore, the protein implementation will more complex

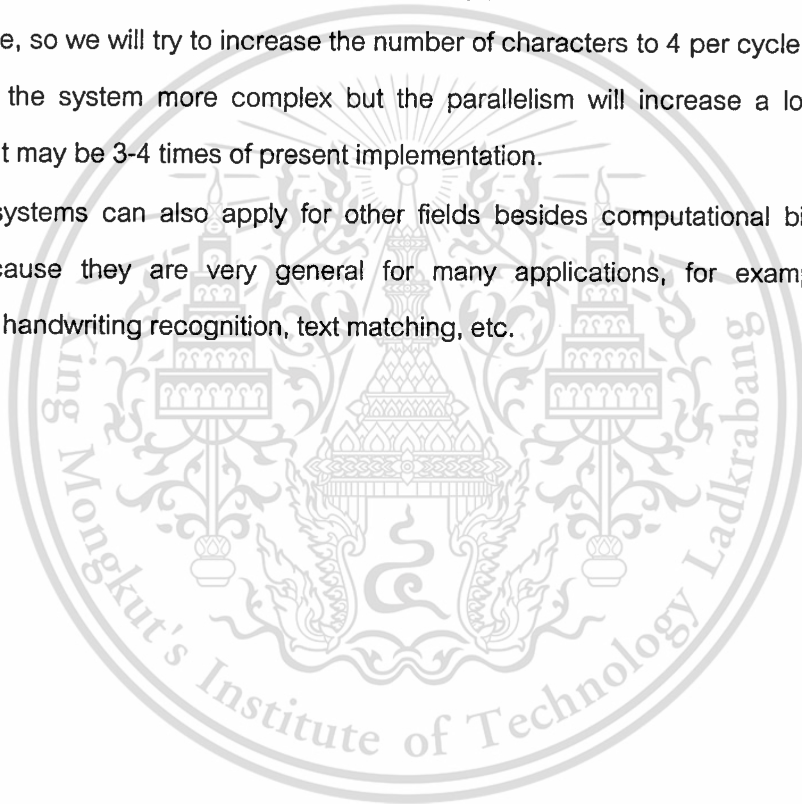
This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

in circuit and require at least three times as much area as the DNA sequence matching implementation.

The system can also develop an implementation for computing local edit distances. The local edit distance is a minimum in the S&W matrix. The local alignment implementation will need the alignment to find which interesting substrings.

As future works for deep packet filtering in NIDS, we will optimize further inside the system by removing some redundant elements and increasing the parallelism of system. In the set of rules, many rules are the substrings of the others. Or two strings have the same certain substring. We can eliminate redundant things to increase the number of rules that can fix on chip. Our present system only processes data flow at one chars per clock cycle, so we will try to increase the number of characters to 4 per cycle. This thing will make the system more complex but the parallelism will increase a lot. And the throughput may be 3-4 times of present implementation.

Our systems can also apply for other fields besides computational biology and NIDS because they are very general for many applications, for example image detection, handwriting recognition, text matching, etc.



This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Bibliography

[1] K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, and P. Athanas, "A run-time reconfigurable system for gene-sequence searching," *Proceedings of the International VLSI Design Conference*, pp. 561–566, 2003.

[2] C.W. Yu, K.H. Kwong, K.H. Lee, and P.H.W. Leong, "A smith-waterman systolic cell," *13th Int. Conference on Field-Programmable Logic and Applications*, pp. 2778:375–384, 2003, Springer-Verlag LNCS.

[3] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequence," Journal of Molecular Biology, pp. 147:196–197, 1981.

[4] D. T. Hoang, "Searching genetic databases on splash 2," *In Proceedings 1993 IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 185–191, 1993.

[5] B. L. Hutchings, R. Franklin, and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," *In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 111–120, 2002.

[6] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, "Implementation of a content-scanning module for an internet firewall," *In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 31–38, April 2003.

[7] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using fpgas," *In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 227–238, April 2001.

[8] Y. H. Cho, S. Navab, and W. H. Mangione-Smith, "Deep network packet filter design for reconfigurable devices," *In 12th Conference on Field Programmable Logic and Applications*, pp. 452–461, September 2002. Springer-Verlag.

[9] T. Moors and A. Cantoni, "Cascading content addressable memories," *IEEE Micro*, pp. 56–66, May/June 1992.

[10] A. Mukhopadhyay, "Hardware algorithms for string processing," *IEEE Computer*, pp. 508–511, 1980.

[11] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, "Granidt: Towards gigabit rate network intrusion detection technology," *In 12th Conference on*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Field Programmable Logic and Applications, pp. 404–413, September 2002. Springer-Verlag.

[12] I. Sourdis and D. Pnevmatikatos, "Fast, large-scale string match for a 10gbps fpga-based network intrusion detection system," *In 13th Conference on Field Programmable Logic and Applications*, September 2003. Springer-Verlag.

[13] C. R. Clark and D. E. Schimmel, "Scalable parallel pattern-matching on high-speed networks," *In IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2004.

[14] S.B. Needleman and C.D. Wunch, "A general method applicable to the search for similarities in amino acid sequences of two proteins," *Journal of Molecular Biology*, pp. 443–453, 1990.

[15] European bioinformatics institute home page, fasta searching program. http://www.ebi.ac.uk/fasta34/

[16] National center for biotechnology information. ncbi blast home page. http://www.ncbi.nlm.nih.gov/blast

[17] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning dna sequences," *Journal of Computational Biology*, pp. 203–214, 2000.

[18] Y.Yamaguchi, T.Maruyama, and A.Konagaya, "High speed homology search with fpgas," *Proceedings of Pacific Symposium on Biocomputing02*, pp. 271–282, 2002.

[19] Tim Oliver, Bertil Schmidt, and Douglas Maskell, "Hyper customized processors for biosequence database scanning on fpgas," *Proc. of the 2005 ACM/SIGDA inter. symp. on Field programmable gate arrays*, 2005.

[20] Decypher bioinformatics acceleration solution. http://www.timelogic.com/decypher intro.html

[21] Compugen. http://www.cgen.com/

[22] Paracel, inc. world wide web site. http://www.paracel.com/

[23] The genematcher2 system datasheet. http://www.paracel.com/products/pdfs/ gm2_datasheet.pdf

[24] M. Roesch. Snort, "Lightweight intrusion detection for networks," *In Proceedings of the USENIX LISA Conference*, Nov. 1999.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

[25] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland, "Characterizing the performance of network intrusion detection sensors," *In Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2003.

[26] R. Boyer and J. Moore, "A fast string match algorithm," *In Commun. ACM, vol. 20, no. 10*, pp.762–772, October 1977.

[27] Snort official web site. http://www.snort.org

[28] M. Fisk and G. Varghese, "Analysis of fast string matching applied to content based forwarding and intrusion detection," *In Techical Report CS2001-0670 (updated version)*, University of California- San Diego, 2002.

[29] Sourcefire. snort 2.0 - detection revised. http://www.snort.org/docs/Snort 20-v4.pdf

[30] S. Wu and U. Mander, "A fast algorithm for multi-pattern searching," *In Techical Report TR-94-17*, University of Arizona, 1994.

[31] E. P. Markatos, S. Antonatos, M. Polychronakis, and K. G. Anagnostakis, "Exb: exclusion-based signature matching for intrusion detection," *In Proceedings of CCN'02*, November 2002.

[32] K. G. Anagnostakis, E. P. Markatos, S. Antonatos, and M. Polychronakis, "E^2xb: a domain-specific string matching algorithm for intrusion detection," *In Proceedings of the 18th IFIP International Information Security Conference*, May 2003.

[33] S. Antonatos, K. G. Anagnostakis, E. P. Markatos, and M. Polychronakis, "Performance analysis of content matching intrusion detection systems," *In Proceedings of the International Symposium on Applications and the Internet*, January 2004.

[34] Safenet inc. safexcel-4850 2004. http://www.safenet-inc.com/Library/3/SafeXcel 4850ProductBrief.pdf, "

[35] Cisco Inc., "Cisco pix 500 series firewalls," 2004.

[36] NetScreen Technologies Inc., "Netscreen-idp 10/100/500/1000 specifications," 2003.

[37] PMC Siera Inc., "Pm2329 classipi network classification processor datasheet," 2001.

[38] E. Berk and C. Ananian, "Jlex: A lexical analyzer generator for java" .

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

[39] M. Attig, S. Dharmapurikar, and J. Lockwood., "Implementation results of bloom filters for string matching," *In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2004.

[40] S. Kittitornkun and Y.H. Hu, "Mapping deep nested do loop dsp algorithms to large scale fpga array structures," *IEEE Trans. on Very Large Scale Integration*, vol. 11, no. 2, pp. 208–217, Feb. 2003.

[41] S.Y. Kung, VLSI Array Processors, *Prentice Hall*, Englewood Cliffs, New Jersey, 1988.

[42] R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," *Chapel Hill Conference on VLSI, Computer Science Press*, pp. 363–376, 1985.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Appendix A

Avnet FPGA Virtex-4 Evaluation Kit

A.1 Description

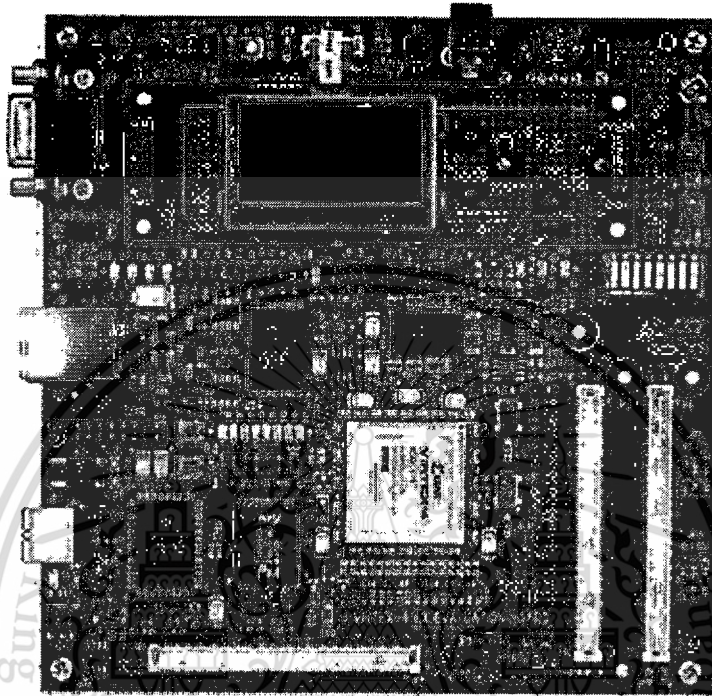


Figure A.1: Virtex-4 Evaluation Board Picture

The Virtex-4 Evaluation Kit provides a platform for engineers designing with the Xilinx Virtex-4 FPGA. The board provides the necessary hardware to not only evaluate the advanced features of the Virtex-4 but also to implement complete user applications.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

A.2 Features

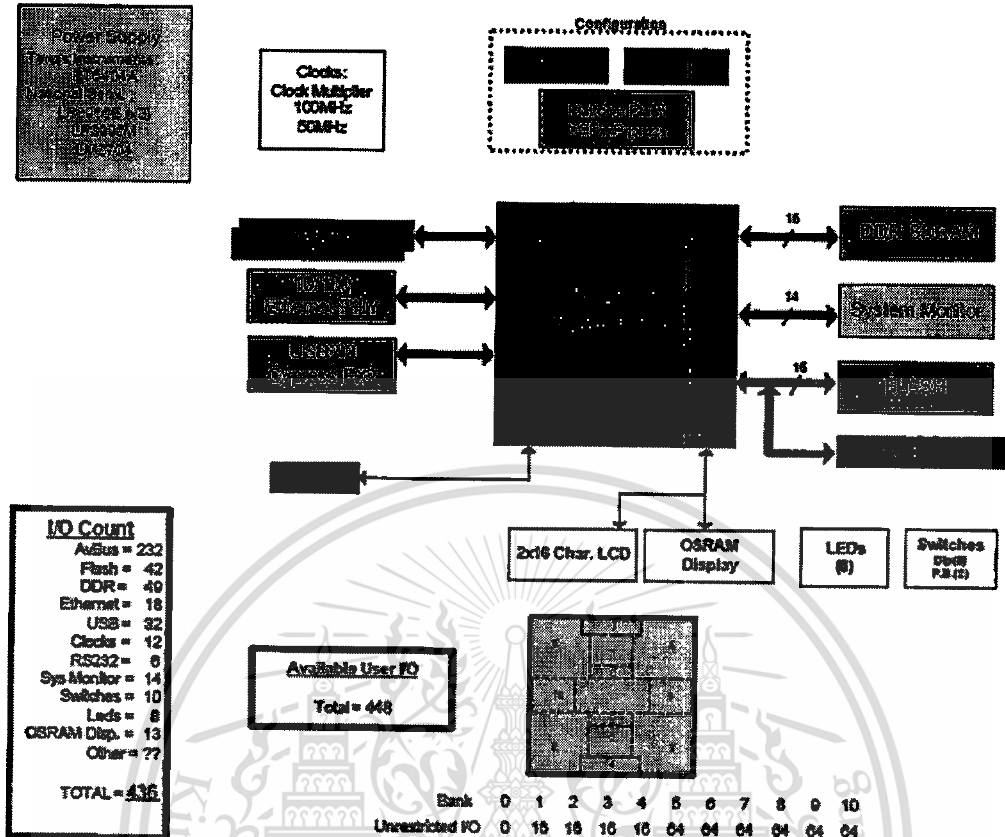


Figure A.2: Virtex-4 Evaluation Board Block Diagram

- FPGA
  - Xilinx Virtex-4 FPGA
  - XC4VLX60-FF668
- Board I/O Connectors
  - Three 140-pin general purpose I/O expansion connectors (AvBus)
  - 30 LVDS pairs
  - Two banks of I/O with selectable output voltage (Vcco)
- Memory
  - Micron DDR SDRAM - 32MB
  - Intel StrataFlash 8MB
- Communication

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

- 10/100 base T Ethernet
- USB 2.0
- RS-232 serial port

- Power
  - 10+ Watt AC/DC +5.0V power supply
  - Texas Instruments 3.3V 6A Module
  - National Linear regulators
- Configuration
  - Xilinx XCFxxP Platform Flash PROM
  - Support for Xilinx Parallel Cable IV
  - Fly-wire support for any Xilinx or compatible cable.

A.3 Hardware

This section of the manual describes the hardware of the Virtex-4 Evaluation board. The hardware was designed with the Virtex-4 FPGA as the focal point. The block diagram is shown in Figure A.2.

A.3.1 Virtex-4 FPGA

The Virtex-4 Evaluation board was designed to support the Virtex-4 FPGA in the 668-pin, flip-chip BGA package (FF668). The FF668 is a versatile package supporting the low to mid-range densities of the LX device including the 4VLX15, 4VLX25, 4VLX40 and 4VLX60. The FF668 package also supports two of the densities of the SX device, the 4VSX25 and 4VSX35. This Virtex-4 Evaluation board is available with device the LX60. The FF668 package has 448 I/Os broken into 10 I/O banks.

The Block Diagram in Figure 10 illustrates the location of the I/O banks on the FF668 package. Bank 0 contains all of the dedicated configuration pins. The ten I/O banks consist of four banks of 16 I/O and six banks of 64 I/O.

A.3.2 Clocks

The available clock sources on the Virtex-4 Evaluation board are shown below.

- Single-ended, 50 MHz Oscillator FPGA pin B17
- Single-ended, 100 MHz Oscillator FPGA pin C13

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

- Differential, Clock Multiplier/Divider FPGA pins C15 (P) and C14 (N)

The 50 MHz oscillator provides the reference clock to the Texas Instruments CDC5801 Low Jitter Clock Multiplier/Divider. The default resistor straps set the CDC5801 in Multiplication Only mode with a multiplier value of 4 providing a 200 MHz clock to the FPGA.

A.3.3 Memory

The Virtex-4 Evaluation board is populated with both high-speed RAM and non-volatile ROM to support various types of applications. The board has 32 Megabytes (MB) of DDR SDRAM and 8 MB of Flash. If additional memory is necessary for development, the Virtex-4 Evaluation board supports the Avnet Communications/Memory Module.

A.3.4 DDR SDRAM

A single Micron DDR SDRAM device, part number MT46V16M16FG-6 makes up the 16-bit data bus. This device provides 32 MB of memory on a single IC and is organized as 4 Megabits x 16 x 4 banks (256 Megabit). The Virtex-4 Evaluation Board can support larger devices with addressing support for up to 128 MB (1 Gigabit). The device has an operating voltage of 2.5V and the interface is JEDEC Standard SSTL 2 (Class I for unidirectional signals, Class II for bidirectional signals). The -6 speed grade supports 6 ns cycle times (DDR333) with a 2 clock read latency.

A.3.5 Flash Memory

Non-volatile data storage is provided in the form of Flash memory. A single Intel StrataFlash device, part number TE28F640J3C120 makes up the 16-bit data bus. This device provides 8 MB of memory on a single IC and is organized as 4Megabits x 16 (64 Megabit). The device has an operating voltage of 3.0V and is compatible with the LVCMOS25 and LVCMOS33 I/O standards of the FPGA. The 64 Megabit device supports 120 ns cycle times.

A.3.6 Universal Serial Bus (USB)

The Virtex-4 Evaluation Board includes a Cypress EZ-USB FX2 USB Microcontroller, part number CY7C68013-100AC. The EZ-USB FX2 device is a single-chip integrated USB 2.0 transceiver, Serial Interface Engine (SIE) and 8051 microcontroller. This device

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

supports full-speed (12 Mbps) and high-speed (480 Mbps) modes, but does not support low-speed mode (1.5 Mbps). The FX2 interface to the Virtex-4 FPGA is a programmable state machine that supports 8- or 16-bit parallel data transfers. This interface is called the General Programmable Interface (GPIF). The GPIF is controlled by Waveform Descriptors that are created with the Cypress GPIFTool utility and downloaded to the FX2 over the USB cable. The GPIF descriptors are stored in internal RAM and are loaded by the firmware during initialization. The GPIF interface is made up of the signals in the following table, which are connected to Virtex-4 FPGA. Some of the additional GPIF pins are connected to the SelectMAP configuration port on the Virtex-4 FPGA. This provides for the development of a FPGA configuration tool, which may be created by Avnet at a later date.

A.3.7 10/100 Ethernet

The on-board Ethernet PHY is a National DP83847ALQA56A DsPHYTER II. The DP83847 is a small, low power physical layer transceiver that only requires a single 3.3V supply. The PHY supports 3.3V signaling levels to the MAC interface, in this case the Virtex-4 FPGA. The PHY is connected to a Pulse RJ-45 jack with integrated magnetic (part number: J0026D01B). The jack also integrates two LEDs to show Link and Receive Activity. Four more LEDs are provided on the board for status indication. These LEDs indicate Link Speed (D8), Transmit Activity (D7), Collision Detect (D6) and Full Duplex operation (D5). The PHY clock is generated from its own 25 MHz crystal. The PHY address is set to binary 00011.

A.3.8 RS232 Transceiver

The RS232 transceiver is a 3222 available from Harris/Intersil (ICL3222CA) and Analog Devices (ADM3222). This transceiver is operating at 3.3V for VCC with an internal charge pump to create the RS232 compatible output levels. This level converter supports two channels. Both channels are connected to the FPGA.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Appendix B

Implemented Verilog Code

This appendix presents some main parts of Verilog code of Multiple DNA Matching system and Deep Packet Filtering in NIDS system.

B.1 Multiple DNA Matching

B.1.1 Processing Element Hardware Description

```
module PE(CLK,Addr,RSTin,Yin,EDITin,SHIFTin,EDITout,Yout,SHIFTout,RSTout);
        input CLK;
        input RSTin;
        input [1:0] Yin;
        input EDITin;
        input SHIFTin;
        input [3:0] Addr;
        output SHIFTout;
        output EDITout;
        output [1:0] Yout;
        output RSTout;

        wire A0,A1,B,dout,muxout,XvsY,comp abc;
        wire [1:0] Xint;
        wire SHIFTout int;

        //store X
        SRL16E StoreX0(.Q(Xint[0]),.CLK(CLK),.CE(RSTin),.D(Xint[0],
        .                     A0(Addr[0]),.A1(Addr[1]),.A2(Addr[2]),.A3(Addr[3]));
        SRL16E StoreX1(.Q(Xint[1]),.CLK(CLK),.CE(RSTin),.D(Xint[1],
                              .A0(Addr[0]),.A1(Addr[1]),.A2(Addr[2]),.A3(Addr[3]));

        //pass the database character through
        FDRE Y0out(.Q(Yout[0]),.C(CLK),.CE(SHIFTin),.D(Yin[0]),.R(RSTin));

        FDRE Y1out(.Q(Yout[1]),.C(CLK),.CE(SHIFTin),.D(Yin[1]),.R(RSTin));
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
//compare database character (Yin) to query character
LUT4 compXY(.O(XvsY),.I0(Xint[0]),.I1(Xint[1]),.I2(Yin[0]),.I3(Yin[1]));
        defparam compXY.INIT = 16'h8421;

//compare a+1, b, and c to determine minimum value to send
LUT4 compABC(.O(comp abc),.I0(A0),.I1(A1),.I2(B),.I3(EDITin));
        defparam compABC.INIT = 16'h6009;

//signal mismatch or send value form CompABC
MUXCY MUX1(.DI(1'b0),.CI(Comp abc),.S(XvsY),.O(muxout));

// send either a + 2 or value from comp1 1
XORCY outd(.O(dout),.CI(A1),.LI(muxout));

//store cell b for use as cell a in next time step
SRL16 StoreA1 (.Q(A1),.CLK(CLK),,.CE(RSTin),.D(EDITin),
                .A0(Addr[0]),.A1(Addr[1]),.A2(Addr[2]),.A3(Addr[3]));

//store B1
SRL16 StoreB (.Q(B),.CLK(CLK),,.CE(RSTin),.D(dout),
.A0(Addr[0]),.A1(Addr[1]),.A2(Addr[2]),.A3(Addr[3]));
// Store A0
FDRE storeA0(.C(CLK),.CE(SHIFTin),.R(RSTin),.D( A0),.Q(A0));

// pass the EN through
FDRSE SHIFT(.Q(SHIFTout int),.C(CLK),.CE(1'b1),.D(SHIFTin),
                .R(1'b0),.S(RSTin));

// pass the Init through
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
        FDRSE synset(.Q(RSTout),.C(CLK),.CE(1'b1),.D(RSTin),.R(1'b0),.S(RSTin));

        assign SHIFTout = SHIFTout int;
endmodule
```

B.1.2 Systolic Array Hardware Description

```
module Sysarr(CLK,Addr,RSTin,Yin,EDITin,SHIFTin,EDITout,Yout,SHIFTout,RSTout);
        input CLK;
        input RSTin;
        input [1:0] Yin;
        input EDITin;
        input SHIFTin;
        input [3:0] Addr;
        output SHIFTout;
        output EDITout;
        output [1:0] Yout;
        output RSTout;

        parameter NumbersPE = 12;
        parameter loop size = NumbersPE/6;
        parameter loop2 size = 2 * loop size;
        parameter loop3 size = 3 * loop size;
        parameter loop4 size = 4 * loop size;
        parameter loop5 size = 5 * loop size;

        wire [NumbersPE:0]Rst int;
        wire [NumbersPE:0]Y1 int;
        wire [NumbersPE:0]Y0 int;
        wire [NumbersPE:0]EDIT int;
        wire [NumbersPE:0]SHIFT int;

        assign Rst int[0]=RSTin;
        assign RSTout=Rst int[NumbersPE];
        assign Y1 int[0]=Yin[1];
        assign Y0 int[0]=Yin[0];
        assign Yout[1]=Y1 int[NumbersPE];
        assign Yout[0]=Y0 int[NumbersPE];
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
assign EDIT int[0]=EDITin;
assign EDITout=EDIT int[NumbersPE];
assign SHIFT int[0]=SHIFTin;
assign SHIFTout=SHIFT int[NumbersPE];

genvar i;
generate
        for(i =1;i<=loop size;i=i+1)
        begin:addbit
                PE arr(.CLK(CLK),.Addr(Addr),.RSTin(Rst int[i-1]),
                        .Yin(Y1 int[i-1],Y0 int[i-1]),
                        .EDITin(EDIT int[i-1]),.SHIFTin(SHIFT int[i-1]),
                        .EDITout(EDIT int[i]),.Yout(Y1 int[i],Y0 int[i]),
                        .SHIFTout(SHIFT int[i]),.RSTout(Rst int[i])
                );
        end
endgenerate

genvar j;
generate
        for(j =1;j<=loop size;j=j+1)
        begin:addbit
                PE arr(.CLK(CLK),.Addr(Addr),.RSTin(Rst int[loop size+j-1]),
                        .Yin(Y1 int[loop size+j-1],Y0 int[loop size+j-1]),
                        .EDITin(EDIT int[loop size+j-1]),
                        .SHIFTin(SHIFT int[loop size+j-1]),
                        .EDITout(EDIT int[loop size+j]),
                        .Yout(Y1 int[loop size+j],Y0 int[loop size+j]),
                        .SHIFTout(SHIFT int[loop size+j]),
                        .RSTout(Rst int[loop size+j])
                );
        end
endgenerate

genvar k;
generate
        for(k =1;k<=loop size;k=k+1)
        begin:addbit
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
                PE arr(.CLK(CLK),.Addr(Addr),.RSTin(Rst int[loop2 size+k-1]),
                        .Yin(Y1 int[loop2 size+k-1],Y0 int[loop2 size+k-1]),
                        .EDITin(EDIT int[loop2 size+k-1]),
                        .SHIFTin(SHIFT int[loop2 size+k-1]),
                        .EDITout(EDIT int[loop2 size+k]),
                        .Yout(Y1 int[loop2 size+k],Y0 int[loop2 size+k]),
                        .SHIFTout(SHIFT int[loop2 size+k]),
                        .RSTout(Rst int[loop2 size+k])
                );
        end
endgenerate

genvar l;
generate
        for(l =1;l<=loop size;l=l+1)
        begin:addbit
                PE arr(.CLK(CLK),.Addr(Addr),.RSTin(Rst int[loop3 size+l-1]),
                        .Yin(Y1 int[loop3 size+l-1],Y0 int[loop3 size+l-1]),
                        .EDITin(EDIT int[loop3 size+l-1]),
                        .SHIFTin(SHIFT int[loop3 size+l-1]),
                        .EDITout(EDIT int[loop3 size+l]),
                        .Yout(Y1 int[loop3 size+l],Y0 int[loop3 size+l]),
                        .SHIFTout(SHIFT int[loop3 size+l]),
                        .RSTout(Rst int[loop3 size+l])
                );
        end
endgenerate

genvar m;
generate
        for(m =1;m<=loop size;m=m+1)
        begin:addbit
                PE arr(.CLK(CLK),.Addr(Addr),.RSTin(Rst int[loop4 size+m-1]),
                        .Yin(Y1 int[loop4 size+m-1],Y0 int[loop4 size+m-1]),
                        .EDITin(EDIT int[loop4 size+m-1]),
                        .SHIFTin(SHIFT int[loop4 size+m-1]),
                        .EDITout(EDIT int[loop4 size+m]),
                        .Yout(Y1 int[loop4 size+m],Y0 int[loop4 size+m]),
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
                    .SHIFTout(SHIFT int[loop4 size+m]),
                    .RSTout(Rst int[loop4 size+m])
                );
        end
    endgenerate

    genvar n;
    generate
        for(n =1;n<=loop size;n=n+1)
        begin:addbit
            PE arr(.CLK(CLK),.Addr(Addr),.RSTin(Rst int[loop5 size+n-1]),
                    .Yin(Y1 int[loop5 size+n-1],Y0 int[loop5 size+n-1]),
                    .EDITin(EDIT int[loop5 size+n-1]),
                    .SHIFTin(SHIFT int[loop5 size+n-1]),
                    .EDITout(EDIT int[loop5 size+n]),
                    .Yout(Y1 int[loop5 size+n],Y0 int[loop5 size+n]),
                    .SHIFTout(SHIFT int[loop5 size+n]),
                    .RSTout(Rst int[loop5 size+n])
                );
        end
    endgenerate
endmodule
```

B.2 Deep Packet Filtering in NIDS

B.2.1 Hardware Description of A Processing Element in Match Processor Array

```
module PE(clk,RSTin,Yin,MATCHin,MATCHout,Yout);;
    input clk;
    input RSTin;
    input [7:0] Yin;
    input MATCHin;
    output MATCHout;
    output [7:0] Yout;

    wire [7:0]char;
    wire MATCH int1,MATCH int2;
    wire and11,and12,andout1;
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
        FDC REG0(.C(clk),.CLR(RSTin),.D(Yin[0]),.Q(char[0]));
        FDC REG1(.C(clk),.CLR(RSTin),.D(Yin[1]),.Q(char[1]));
        FDC REG2(.C(clk),.CLR(RSTin),.D(Yin[2]),.Q(char[2]));
        FDC REG3(.C(clk),.CLR(RSTin),.D(Yin[3]),.Q(char[3]));
        FDC REG4(.C(clk),.CLR(RSTin),.D(Yin[4]),.Q(char[4]));
        FDC REG5(.C(clk),.CLR(RSTin),.D(Yin[5]),.Q(char[5]));
        FDC REG6(.C(clk),.CLR(RSTin),.D(Yin[6]),.Q(char[6]));
        FDC REG7(.C(clk),.CLR(RSTin),.D(Yin[7]),.Q(char[7]));

        ROM16X1 ROM4LSB(.O (and11),
                                .A0 (char[0]),.A1 (char[1]),
                                .A2 (char[2]),.A3 (char[3]));
                defparam ROM4LSB.INIT = 16'h0001;//store 4-bit LSBs of char A

        ROM16X1 ROM4MSB(.O (and12),
                                .A0 (char[0]),.A1 (char[1]),
                                .A2 (char[2]),.A3 (char[3]));
                defparam ROM4MSB.INIT = 16'h0004;//store 4-bit LSBs of char A

        MUXCY AND1(.DI(1'b0),.CI(and12),.S(and11),.O(andout1));
        MUXCY AND2(.DI(1'b0),.CI(MATCH int1),.S(andout1),.O(MATCH int2));

        FDC FF1(.C(clk),.CLR(RSTin),.D(MATCHin),.Q(MATCH int1));
        FDC FF2(.C(clk),.CLR(RSTin),.D(MATCH int2),.Q(MATCHout));
endmodule
```

B.2.2 Systolic Array Hardware Description of Match Processor Array

```
module sysarr(clk,RSTin,Yin,MATCHin,MATCHout,Yout);;
        input clk;
        input RSTin;
        input [7:0] Yin;
        input MATCHin;
        output MATCHout;
        output [7:0] Yout;

        parameter NumbersPE = 11139;
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
wire [NumbersPE:0]MATCH int;
wire [NumbersPE:0]Y0 int;
wire [NumbersPE:0]Y1 int;
wire [NumbersPE:0]Y2 int;
wire [NumbersPE:0]Y3 int;
wire [NumbersPE:0]Y4 int;
wire [NumbersPE:0]Y5 int;
wire [NumbersPE:0]Y6 int;
wire [NumbersPE:0]Y7 int;

assign Yout[0]=Y0 int[NumbersPE];
assign Yout[1]=Y1 int[NumbersPE];
assign Yout[2]=Y2 int[NumbersPE];
assign Yout[3]=Y3 int[NumbersPE];
assign Yout[4]=Y4 int[NumbersPE];
assign Yout[5]=Y5 int[NumbersPE];
assign Yout[6]=Y6 int[NumbersPE];
assign Yout[7]=Y7 int[NumbersPE];
assign MATCH int[0]=MATCHin;
assign MATCHout=MATCH int[NumbersPE];

genvar i;
generate
	for(i =1;i<=NumbersPE;i=i+1)
	begin:addbit
		PE arr(.CLK(CLK),.RSTin(RSTin[i-1]),.MATCHin(MATCH int[i-1]),
			.Yin(Y7 int[i-1],Y6 int[i-1],Y5 int[i-1],Y4 int[i-1], ),
			.MATCHout(MATCH int[i]),
			.Yout(Y7 int[i],Y6 int[i],Y5 int[i], )
		);
	end
endgenerate
endmodule
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

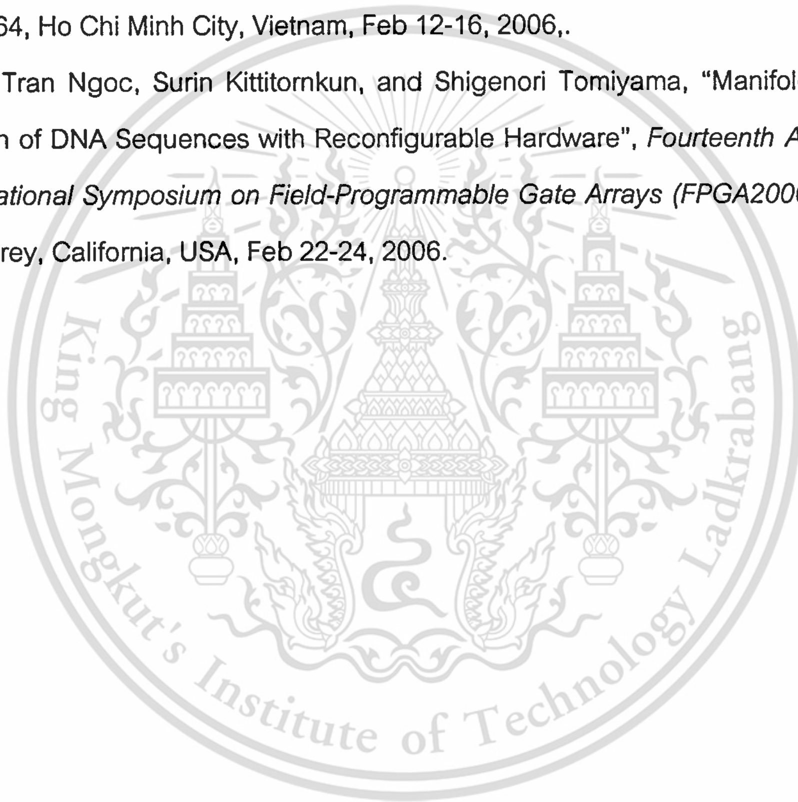
Appendix C

Publication List

[1] Thinh Tran Ngoc, Surin Kitttitornkun, and Yu Hen Hu, "Mass-Similarity Search of Biological Sequences using FPGA", *In Proceedings of the 2005-ECTI International Conference*, pp. 129-132, Pattaya, Thailand, May 12-13, 2005.

[2] Thinh Tran Ngoc, Surin Kittitornkun, and Shigenori Tomiyama, "Multiple DNA Matching with Reconfigurable Hardware", *4th IEEE International Conference on Computer Sciences – Research, Innovation, and Vision for the Future (RIVF'06)*, pp. 161-164, Ho Chi Minh City, Vietnam, Feb 12-16, 2006,.

[3] Thinh Tran Ngoc, Surin Kittitornkun, and Shigenori Tomiyama, "Manifold Similarity Search of DNA Sequences with Reconfigurable Hardware", *Fourteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA2006)*, pp. 231, Monterey, California, USA, Feb 22-24, 2006.



This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

ECTI-CON 2005

The 2005 ECTI International Conference

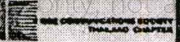
1 00 10 101 01000 1000 101110 1010101010 1

ECTI Association

Proceedings of The 2005 Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI) International Conferenc

May 12-13, 2005

Asia Pattaya Beach Hotel, Pattaya, Cholburi, THAILAND



Technology Ladkrabang

Forbidden to modify the content, and cite the document when use.

MASS-SIMILARITY SEARCH OF BIOLOGICAL SEQUENCES USING FPGA

Thinh Tran Ngoc, Surin Kittitornkun

Dept.of Computer Engineering,
Faculty of Engineering,
King Mongkut's Institute of Technology
Ladkrabang, Bangkok, 10520 Thailand
tnthinh@dit.hcmut.edu.vn,
kksurin@kmitl.ac.th

Yu Hen Hu

Dept. of Electrical and Computer Eng.
University of Wisconsin-Madison,
Madison, WI 53706, USA
hu@engr.wisc.edu

ABSTRACT

In this paper, we present a processor array for the similarity search of many DNA sequences against a large database. Based on a recently proposed *systolic mapping*, it is implemented using a low-cost 400k-gate Xilinx Spartan III XC3S400 FPGA. When comparing to software implementation, the array can achieve approximately hundreds times increasing in performance while saving time spent on transferring the database which is the bottleneck of the other hardware systems up to sixteen folds.

1. INTRODUCTION

Due to endeavors such as the Human Genome Initiative, genetic sequence data are being generated at an ever increasing rate. A fundamental problem in the field of bioinformatics is the comparison of two sequences of biological data, such as DNA strands and proteins. Comparisons of a *set of query sequences against a large database* to find the homology are very common. The Smith-Waterman (S&W) algorithm [1] is a dynamic programming method for homology searches and sequence alignment in genetic databases. It makes all pairwise comparisons between the two strings. It achieves high sensitivity as all the matched and near-matched pairs are detected, however, the computation time required strongly limits its use.

On the hardware side, there have been a lot of ASIC and FPGA implementations of the S&W algorithm to accelerate the throughput. However, all of them can only compare one query sequence against the database. In case if there are many queries, those systems must compare with the same database many times. As a result, the data transfer time from the host computer can be a big portion since the database can be very huge ranging from hundreds of megabytes to some gigabytes. In this paper, we propose an FPGA implementation of the S&W algorithm for DNA searching that can reduce the database access time and increase parallellism by comparing query sequences as many as 16 sequences against the large database.

The paper is organized as follows. Section 2 reviews some background on similarity search and identifies the problem of interest. Next, our design methodology is elaborated. The FPGA implementation and its experimental results are discussed in section 4. Finally, future works are suggested in our conclusion.

We would like to acknowledge the following: AUN-SeedNet for the scholarship and Xilinx, Inc. for donating the software tools.

2. SIMILARITY SEARCH AND PROBLEM IDENTIFICATION

2.1. Similarity Search

Most algorithms for similarity search can be categorized into two groups: dynamic and heuristic. Dynamic algorithms give optimal solutions, and well known searching algorithms like S&W, Needleman-Wunch and Hidden Markov Models are of the dynamic kind. Examples of heuristic algorithms are BLAST [2] and FASTA [3]. On the other hand, heuristic algorithms are statistically driven sequence searching and alignment methods, and might not be as sensitive for all similarity searches as the full dynamic programming algorithms. Beside that, MegaBlast is a new program using greedy algorithm proposed by Zhang et al. [4].

2.2. Problem Identification

Fig.1 illustrates multiple similarity searches of some well-known software with 10 queries. The lengths of the queries are approximately 150-400 characters, where the database *env_nt* contains over 800 thousands nucleotide sequences and has over 800 millions characters on the platform is a Pentium4 1.8 GHz, 512MB Ram, 80GB harddrive, and Windows 2000 Professional. It is obvious that MegaBlast is much better than S&W. That is because MegaBlast concatenates a number of queries to save time spent on transferring the database. Therefore it can be used to swiftly compare two large sets of sequences against each other. The only disadvantage of MegaBlast is its low sensitivity, i.e. significant matches may be missed.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

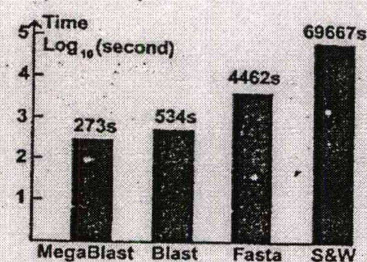


Figure 1: Run times of multiple similarity searches of some well-known software with 10 queries

On the hardware side, many ASIC and FPGA implementations of S&W algorithm have been proposed and improved significantly. Most of them can reduce the computation time by using *parallel/pipelined* computation using systolic processor. As described, they only make comparisons between the two strings at a time. If the query and database are m and n characters long, then the computation time complexity of a software implementation and a hardware implementation are $O(mn)$ and $O(m+n)$, respectively. When the database is very large, small changes can be very significant.

However, in reality, data transfer problem can make most of systems slow down. For example, the computation performance of [5] can reach up to 814 billion cell updates per second (BCUPS). When the real system was verified, it had a disappointing performance of approximately 136 BCUPS. In addition, to compare a number of query sequences in a very large database, a considerable amount of time is spent on data transfer because the database is read and transferred to the hardware again and again.

3. OUR DESIGN METHODOLOGY

From the problem identified in the previous section, we will take the advantage of hardware's parallelism together with simultaneous searching of many query sequences to save database transfer time from MegaBlast to reduce the latency and increase the throughput. The method we use is to widen the ability of *pipelining* [7] that is suitable for FPGA. In the processor array, each Processing Element (PE) will compare one character of database to each character of every query sequence every clock cycle.

This section, we will first describe S&W algorithm and explain our systolic or space-time mapping later.

3.1. Smith-Waterman Algorithm

The S&W algorithm computes the similarity of two entire DNA or protein sequences, or global alignment. The resulting score can be thought of as the total number of mutations to change the query sequence into the database sequence, or the *edit distance*. In multiple similarity search-

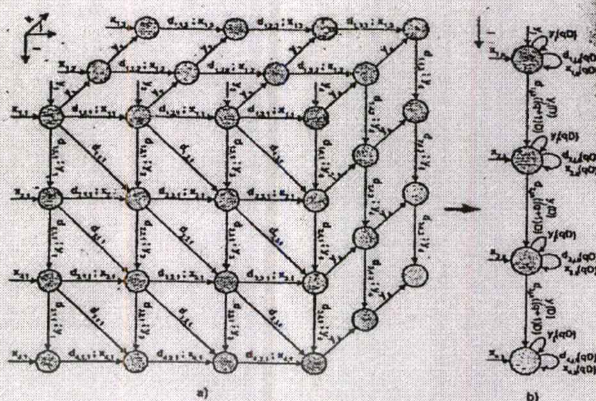


Figure 2: a) Dependence Graph; b) a linear systolic array architecture

ing, method to calculate each query is also the same with the original formula of S&W algorithm. Let q query sequences be the source sequences of the same length $X_k = x_{1,k}x_{2,k}x_{3,k}...x_{m,k}$, $1 \le k \le q$, and $Y = y_1y_2y_3...y_n$ be the database sequence, the edit distance between each query sequence and database sequence is computed according to the following formula:

$$d_{i,j,k} = min \begin{cases} d_{i-1,j,k} & + c_{del} \\ d_{i,j-1,k} & + c_{ins} \\ d_{i-1,j-1,k} & + c_{sub} \end{cases} \quad (1)$$

$d_{0,0,k} = 0; d_{i,0,k} = d_{i-1,0,k} + c_{del}; d_{0,j,k} = d_{0,j-1,k} + c_{ins}$

Here c_{del} is the cost of deleting $x_{i,k}$, c_{ins} is the cost of inserting y_j, and c_{sub} is the cost of substituting y_j for $x_{i,k}$.

3.2. Systolic Design Methodology

Eq.(1) is suitable for pipelined execution; therefore, a systolic array architecture is chosen to implement in FPGA. We use the graph-base design methodology described by Kung [6] and a recent improvement [7].

3.2.1. Dependence Graph Design

The Dependence Graph (DG) [6] shows the dependencies of the computations that occur in the algorithm. For example, the DG of the algorithm for $m = 4$, $n = 4$ and $q = 3$ is shown in Fig.2.a. It is a 3-D directed graph according to Eq.(1). Each node in the figure represents the assignment of Eq.(1). The directed arcs between nodes represent the dependencies between these nodes.

From Fig.2.a, it can be seen that each node for determining the edit distance $d_{i,j,k}$ must provide parameters to the other nodes for the calculation of $d_{i+1,j,k}$; $d_{i,j+1,k}$; and $d_{i+1,j+1,k}$ in every $[i,j]$ plane. Each node also passes y_j to the next node along the k-axis in every $[i,k]$ plane.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

3.2.2. Systolic or space-time mapping

The goal of our design is a linear systolic array processor to reduce the number of processors necessary to carry out the computation process. Two tasks to mapping DG to systolic array processor are *processor allocation* and *scheduling*. These can be solved by algebraic projection [7]. This projection is called systolic or space-time mapping.

A dependence matrix D_V is composed of a set of dependence vectors as shown in Fig.2.a:

$$D_V = \left[\; \vec{d_1} \;\; \vec{d_2} \;\; \vec{d_3} \;\; \vec{d_4} \; \right] \qquad (2)$$

where $V = \{x_{i,k}, y_j, d_{i,j,k}\}$ is set of variables, and $\vec{d_1} = [i\ j\ k]^t = [1\ 0\ 0]^t$, $\vec{d_2} = [1\ 1\ 0]^t$, $\vec{d_3} = [0\ 1\ 0]^t$, $\vec{d_4} = [0\ 0\ 1]^t$. The one-D space-time mapping matrix T_1 consists of a scheduling vector $\vec{s}$ and a processor allocation vector $\vec{P}$.

$$T_1 = \begin{bmatrix} \vec{s}^t \\ \vec{P}^t \end{bmatrix} = \begin{bmatrix} s_1 & s_2 & s_3 \\ p_1 & p_2 & p_3 \end{bmatrix} \qquad (3)$$

where s_i and p_i are integer numbers. The values of s_i and p_i can be selected using Integer Programming and some heuristic search subject to some design constraints [7]. By observation and experiences, we chose $\vec{s}^t = [1\ q\ q]$, and $\vec{P}^t = [1\ 0\ 0]$. The mapping of D_V results in a *delay-edge* matrix:

$$\begin{bmatrix} r_1 & r_2 & r_3 & r_4 \\ e_1 & e_2 & e_3 & e_4 \end{bmatrix} = T_1 D_V \qquad (4)$$

$$\begin{bmatrix} 1 & q+1 & q & q \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & q & q \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where r_i is the number of registers (delays) of edge e_i which is mapped from the corresponding d_i in Eq.(2). Fig.2.b is a linear systolic array architecture resulting from applying T_1 space-time mapping matrix. The delay 'D' is a pipeline register. For example, in PE2, to calculate $d_{2,j,k}$, we need $d_{2,j-1,k}$ which was calculated q clock cycles before and $d_{1,j,k}$ and $d_{1,j-1,k}$ which were calculated in PE1 one and $q+1$ clock cycles before, respectively.

4. EXPERIMENTAL RESULTS

4.1. An FPGA Implementation

To reduce hardware complexity, we pick the costs, 1 for gap (c_{del}, c_{ins}) and 2 for substitution (c_{sub}). The values of $d_{i,j-1,k}$ and $d_{i-1,j,k}$ in Eq.(1) are restricted to $d_{i-1,j-1,k} \pm 1$ and the equation can be simple to obtain:

$$d = \begin{cases} a & ,if\ (b\ or\ c) = a - 1\ or\ (X_{i,k} = Y_j) \\ a + 2 & ,if\ (b\ and\ c) = a + 1\ and\ (X_{i,k} \neq Y_j) \end{cases} \qquad (5)$$

where $d = d_{i,j,k}; a = d_{i-1,j-1,k}; b = d_{i,j-1,k}; c = d_{i-1,j,k}$ [8].

Using Eq.(5), it can be seen that modulo-4 encoding can be used to represent each value. Furthermore, the least significant bit of a and d are always equal, and hence, d can be represented by *only the most significant bit* [9]. In genomic databases, four character alphabets are used to represent the four bases in the DNA molecule. These characters are typically A,T,G, and C. So, each character can be encoded with a two-bit binary number. The processing element (PE) shown in Fig.3 can compute Eq.(5). In each PE, a RAM-based shift register of size 16x2 bits is used to store each character of q query sequences X. The value of q can be programmed at run time from 1 to 16 depending on input *Addr*.

In order to calculate d, inputs a; b and c should be available. *EDITin* is the new value of c. It is also stored in a 16x1-bit shift register $A1$ and the output of this shift register is the current value of a. Value b is the output of 16x1-bit shift register B. After that, d will be stored in a flip-flop to be c for the next PE. At the same time, d is also stored in shift register B.

When *SHIFTout* is active, each character of Y will pass through one PE, and *SHIFTout* is only active one time in q clock cycles. This is an advantage to decrease data transfer speed.

The design is based on some optimizations for run-time customization [9], i.e. the length of queries and the initialization value of shift registers will be changed at run time. Thus, it occupies only 4 Xilinx SpartanIII slices per PE. Fig.4.a is the complete systolic array.

For the simulation and verification, we used Modelsim XE II v5.8a. Fig.5 is simulation waveform of a two-PE array

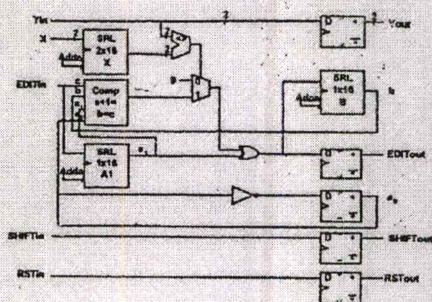


Figure 3: Microarchitecture of each processing element (PE) in Fig.2.b

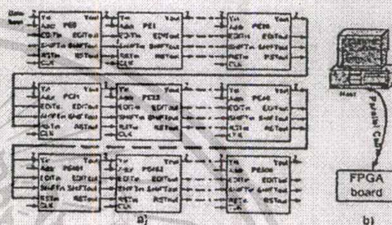


Figure 4: a) Our linear systolic array for multiple query search; b) Communication between Host and FPGA board

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

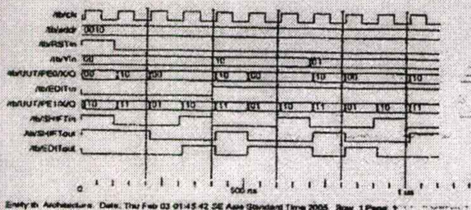


Figure 5: Simulation waveform of a two-PE array

for multiple query search. In Fig.5, *EDITout* is the results of the last PE but it is only the most-significant-bit in two-bit encode. Therefore, to obtain real results, *EDITout* in the last PE will pass to up-down counters with a finite-state machine control. Each counter is respectively the edit distance result. The value of initialization of the counter equals the length of each query. The counter increases if the current output is greater than the previous one, and it decreases otherwise.

4.2. Performance Comparison

We fit 500 PEs on a Spartan III XC3S400 using Verilog HDL to design and Xilinx Foundation ISE 6.3i tools to synthesize and implement. Table.4.2 compares our design to systems using Xilinx FPGA chip. The number of hardware unit for a character of query sentence is 4/16, less than much in compared with others. The throughputs of [5, 9] are noticeable since the FPGA sizes are large to accommodate a lot of PEs. If our design were implemented on a larger FPGA chip, the performance would be nearly similar or even better. Most of the previous systems take only 1 clock cycle to compare in every PE, so the speed of database requested for computation is very large. For example, the data request speed of [5]

$$Speed_{data} = f \times E = 202 \times 2 = 404 \text{ Mbps}$$

where f is the system clock frequency and E is the number of bits to encode each character. With speed of 404 Mbps, these systems need expensive FPGA(s) with high-speed interfaces. Nevertheless, the delay of communication can occur everywhere in the host computer, e.g. harddrive speed and I/O interface. Thus, in reality, they could hardly run at the maximum throughput. Meanwhile, our data request speed is one-sixteenth of others.

$$OurSpeed_{data} = [f \times E]/q = [180 \times 2]/16 = 22.5 \text{ Mbps}$$

Moreover, with data request speed being only 22.5 Mbps, we can use parallel cable to connect very low-cost FPGA device with a host computer as shown in Fig.4.b.

5. CONCLUSION AND FUTURE WORKS

A new SRAM-base FPGA implementation of DNA matching called *Mass-Similarity Search* is proposed. It can compare up to 16 DNA queries against a large database using parallel/pipeline systolic processor array. According to our implementation result, the achievable throughput/cost can be up to 10 folds higher than others with smaller hardware complexity. In addition, the speed required for database transfer to the FPGA can be reduced to 16 times lower.

As future works, we will optimize the system further and test it on a low-cost Spartan III XC3S1500 FPGA device.

Table 1: Comparison of FPGA similarity search systems

System	[10]	[9]	[5]	Ours
# queries (q)	1	1	1	16
# slices/PE	16	4	3	4
#PEs/chip	14	7,000	4,032	500
#chips/System	16	1	1	1
Model	4010	2V6000	V1000E	3S400
Clock(MHz)	N/A	180	202	180
Throughput (BCUPS)	43	1,260	814	90
$Speed_{data}$	N/A	360.0	404.0	22.5
Cost (U$)	N/A	4,606.8	1,722.9	35.8
Throughput/Cost (MCUPS/U$)	N/A	273.5	472.5	2,515.4

6. REFERENCES

[1] T. F. Smith and M. S. Watermann, "Identification of common molecular subsequence," *Journal of Molecular Biology*, pp. 147:196–197, 1981.

[2] "http://www.ncbi.nlm.nih.gov/blast," .

[3] "http://www.ebi.ac.uk/fasta34/," .

[4] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning dna sequences," *Journal of Computational Biology*, pp. 203–214, 2000.

[5] C.W. Yu, K.H. Kwong, K.H. Lee, and P.H.W. Leong, "A smith-waterman systolic cell," *13th Int. Conference on Field-Programmable Logic and Applications*, pp. 2778:375–384, 2003,Springer-Verlag LNCS.

[6] S.Y. Kung, *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[7] S. Kittitornkun and Y.H. Hu, "Mapping deep nested do loop dsp algorithms to large scale fpga array structures," *IEEE Trans. on Very Large Scale Integration*, vol. 11, no. 2, pp. 208–217, Feb. 2003.

[8] R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," *Chapel Hill Conference on VLSI, Computer Science Press*, pp. 363–376, 1985.

[9] K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, and P. Athanas, "A run-time reconfigurable system for gene-sequence searching," *Proceedings of the International VLSI Design Conference*, 2003.

[10] D. T. Hoang, "Searching genetic databases on splash 2," *In Proceedings 1993 IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 185–191, 1993.

This material is reserved for educational [illegible] only, not allowed for commercial use.

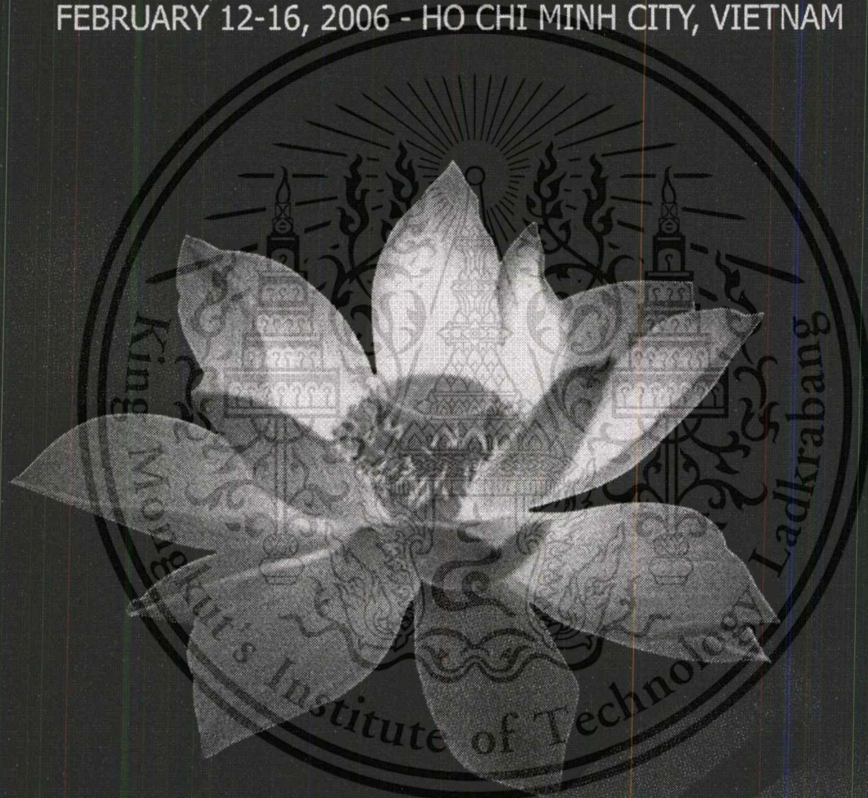
Forbidden to modify the content, and cite the document when use.

RIVF 2006

THE FOURTH IEEE INTERNATIONAL CONFERENCE ON COMPUTER SCIENCES

RESEARCH, INNOVATION AND VISION FOR THE FUTURE

FEBRUARY 12-16, 2006 - HO CHI MINH CITY, VIETNAM



ADDENDUM CONTRIBUTIONS

IEEE

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, Ho Chi Minh city, Vietnam

Multiple DNA Matchings with Reconfigurable Hardware

Thinh Tran Ngoc, Surin Kittitornkun, and Shigenori Tomiyama

Abstract—**This paper presents a reconfigurable processor array for DNA matching of up to sixteen DNA query sequences against a large database. Based on the recently proposed *space-time mapping methodology*, the system can sustain over 1-Tera Operations/Second throughput rate via a USB 2.0 host interface at only 24 Mbps. The processor array is coded in Verilog hardware description language and implemented on a Xilinx Virtex-4 XC4VLX60 FPGA.**

I. Introduction

Nowadays, database similarity searching has been a main problem in the field of bioinformatics. Database similarity searching allows us to determine which of the hundreds of thousands of sequences present in the database, such as DNA strands and proteins, are potentially related to a particular sequence of interest. The Smith-Waterman (S&W) algorithm [1] is a dynamic programming method for similarity searches and sequence alignment in genetic databases. It makes all pairwise comparisons between the two strings. It achieves high sensitivity so that all the matched and near-matched pairs are detected, however, its computation demanding required strongly limits its use.

On the hardware side, there have been a lot of ASIC and FPGA implementations of the S&W algorithm to accelerate the throughput. However, all of them can only compare one query sequence against the database. In case if there are many queries, those systems must compare with the same database many times. As a result, the data transfer time from the host computer can be a big portion since the database can be very huge ranging from hundreds of megabytes to some gigabytes. In this paper, we propose an FPGA implementation of the S&W algorithm for DNA searching that can reduce the database access time and increase parallelism by comparing query sequences as many as 16 sequences against the large database.

The paper is organized as follows. Section II reviews some background on similarity search and identifies the problem of interest. Next, our design methodology is elaborated. The FPGA implementation and its experimental results are discussed in section IV. Finally, section V is our conclusion.

Thinh Tran Ngoc and Surin Kittitornkun are with the Dept. of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, 10520 Thailand. Email: {tnthinh@dit.hcmut.edu.vn, kksurin@kmitl.ac.th}
Shigenori Tomiyama is with the Dept. of Embedded Technology, School of IT and Science, Tokai University, Japan. Email: tomiyama@dt.u-tokai.ac.jp

II. Similarity Search and Problem Identification

A. Similarity Search

Most algorithms for similarity search can be categorized into two groups: dynamic and heuristic. Dynamic algorithms give optimal solutions, and well known searching algorithms like S&W, Needleman-Wunch and Hidden Markov Models are of the dynamic kind. Examples of heuristic algorithms are BLAST [2] and FASTA [3]. On the other hand, heuristic algorithms are statistically driven sequence searching and alignment methods, and might not be as sensitive for all similarity searches as the full dynamic programming algorithms. Beside that, MegaBlast is a new program using greedy algorithm proposed by Zhang et al. [4].

B. Problem Identification

Figure.1 illustrates run times of multiple similarity searches of some well-known software with 10 queries. The lengths of the queries are approximately 150-400 characters, where the database *env_nt* contains over 800 thousands nucleotide sequences and has over 800 millions characters on the platform is a Pentium4 1.8 GHz, 512MB Ram, 80GB harddrive, and Windows 2000 Professional. It is obvious that MegaBlast is much better than S&W. That is because MegaBlast concatenates a number of queries to save time spent on transferring the database. The only disadvantage of MegaBlast is its low sensitivity, i.e. significant matches may be missed.

On the hardware side, many ASIC and FPGA implementations of S&W algorithm have been proposed and improved significantly. Most of them can reduce the computation time by using *parallel/pipelined* computation such as systolic processor array. As described, they only make comparisons between the two strings at a time. If the query and database are m and n characters long, then the computation time complexity of a software implementation and a hardware implementation are $O(mn)$ and $O(m+n)$, respectively. When the database is very large, small changes can be very significant.

However, in reality, data transfer time can make most systems slow down. For example, the theoretical computation performance of [5] can reach up to 814 billion cell updates per second (BCUPS). When the real system was verified, it had a disappointing performance of approximately 136 BCUPS. In addition, to compare a number of query sequences in a very large database, a considerable amount of time is spent on data transfer because the database is read and transferred to the hardware again and again.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, Ho Chi Minh city, Vietnam

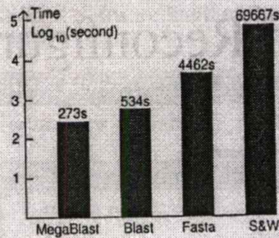


Fig. 1. Run times of multiple similarity searches of some well-known software with 10 queries

		A	C	T
	0	1	2	3
A	1	0	1	2
G	2	1	2	3
A	3	2	3	4

		A	C	T
	0	1	2	3
A	1	0	1	2
C	2	1	0	1
T	3	2	1	0

Fig. 2. Compare two queries "AGA" and "ACT" to database sequence "ACT"

III. Our Design Methodology

From the problem identified in the previous section, we will take the advantage of hardware's parallelism together with simultaneous searching of many query sequences to save database transfer time like MegaBlast We use a systolic processor array-like design methodology [6] that is suitable for large FPGA such as Xilinx Virtex-4 LX60. In the processor array, each Processing Element (PE) will compare one character of database to each character of every query sequence every clock cycle.

A. Smith-Waterman Algorithm

The S&W algorithm computes the similarity of two entire DNA or protein sequences, or global alignment. The resulting score can be thought of as the total number of mutations to change the query sequence into the database sequence, or the *edit distance*. In multiple similarity searching, method to calculate each query is also the same with the original formula of S&W algorithm. Let q query sequences be the source sequences of the same length $X_k = x_{1,k}x_{2,k}x_{3,k}...x_{m,k}$, $1 \le k \le q$, and $Y = y_1y_2y_3...y_n$ be the database sequence, the edit distance between each query sequence and database sequence is computed according to the following formula:

$$d_{i,j,k} = min\begin{cases} d_{i-1,j,k} & + c_{del} \\ d_{i,j-1,k} & + c_{ins} \\ d_{i-1,j-1,k} & + c_{sub} \end{cases} \quad (1)$$

where

$$d_{0,0,k} = 0; d_{i,0,k} = d_{i-1,0,k} + c_{del}; d_{0,j,k} = d_{0,j-1,k} + c_{ins}$$

Here c_{del} is the cost of deleting $x_{i,k}$, c_{ins} is the cost of inserting y_j, and c_{sub} is the cost of substituting y_j for $x_{i,k}$.

In genomic databases, four character alphabets A, T, G, and C are used to represent the four bases in the DNA molecule. For example, Fig.2 shows the distance tables generated when comparing 2 query sentences "AGA" and "ACT" to database sequence "ACT" with the following cost functions:

$$c_{del} = c_{ins} = 1 \qquad c_{sub} = \begin{cases} 0 \ ,if \ x_{i,k} = y_j \\ 2 \ ,if \ x_{i,k} \neq y_j \end{cases}$$

The resulting edit distances are 4 and 0 which are found at the lower right hand corner of the tables.

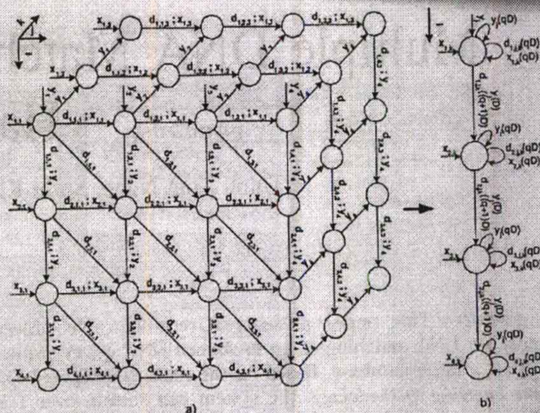


Fig. 3. a) Dependence Graph $m = 4$, $n = 4$ and $q = 3$; b) a linear systolic array architecture resulted from the systolic or space-time mapping in Eq.(4)

B. Space-time Design Methodology

Equation (1) is suitable for pipelined execution; therefore, a systolic array architecture is chosen to implement in FPGA. We use the systolic design methodology described by Kung [7] and a recent improvement by Kittitornkun and Hu [6].

1) Dependence Graph Design: The Dependence Graph (DG) [7] shows the dependencies of the computations that occur in the algorithm. For example, the DG of the algorithm for $m = 4$, $n = 4$ and $q = 3$ is shown in Fig.3.(a). Each node in the figure represents the assignment of Eq.(1). The directed arcs between nodes represent the dependencies between these nodes. From Fig.3.(a), it can be seen that each node for determining the edit distance $d_{i,j,k}$ must provide parameters to the other nodes for the calculation of $d_{i+1,j,k}$; $d_{i,j+1,k}$; and $d_{i+1,j+1,k}$ in every $[i, j]$ plane. Each node also passes y_j to the next node along the k-axis in every $[i, k]$ plane.

2) Space-time mapping: The goal of our design is a linear systolic processor array to reduce the number of processors necessary to carry out the computation process. Two important tasks of mapping DG to systolic processor array are *processor allocation* and *scheduling*. These can be solved by algebraic projection [6]. This projection is called *space-time mapping*.

A dependence matrix D_V is composed of a set of dependence vectors as shown in Fig.3.(a):

$$D_V = \left[\begin{array}{cccc} \vec{d_1} & \vec{d_2} & \vec{d_3} & \vec{d_4} \end{array} \right] \quad (2)$$

where $\vec{d_1} = [i \ j \ k]^t = [1 \ 0 \ 0]^t$, $\vec{d_2} = [1 \ 1 \ 0]^t$, $\vec{d_3} = [0 \ 1 \ 0]^t$, $\vec{d_4} = [0 \ 0 \ 1]^t$ are dependence vectors of set of variables $V = \{x_{i,k}, y_j, d_{i,j,k}\}$. The one-D space-time mapping matrix T_1

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

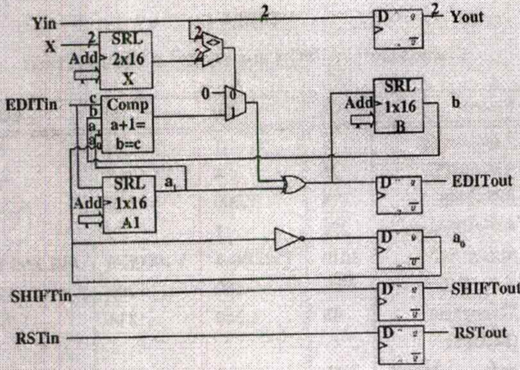


Fig. 4. Microarchitecture of each processing element (PE) in Fig.3(b)

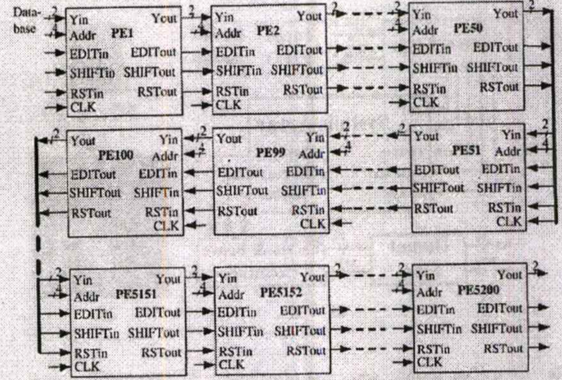


Fig. 5. Our systolic array for multiple DNA matchings, $\#PEs = 5,200$

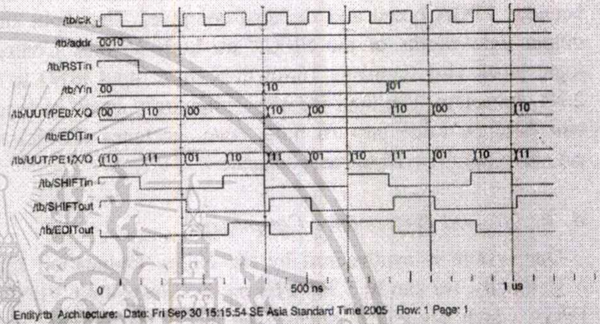


Fig. 6. Simulation waveform of a two-PE array

consists of a scheduling vector $\vec{s}$ and a processor allocation vector $\vec{P}$.

$$T_1 = \begin{bmatrix} \vec{s}^t \\ \vec{P}^t \end{bmatrix} = \begin{bmatrix} s_1 & s_2 & s_3 \\ p_1 & p_2 & p_3 \end{bmatrix} \quad (3)$$

where s_i and p_i are integer numbers. The values of s_i and p_i can be selected using Integer Programming and some heuristic search subject to some design constraints [6]. Based on observation and experiences, we chose $\vec{s}^t = [1\ q\ q]$, and $\vec{P}^t = [1\ 0\ 0]$. The mapping of D_V results in a *delay-edge* matrix:

$$\begin{bmatrix} r_1 & r_2 & r_3 & r_4 \\ e_1 & e_2 & e_3 & e_4 \end{bmatrix} = T_1 D_V \quad (4)$$

$$\begin{bmatrix} 1 & q+1 & q & q \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & q & q \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where r_i is the number of registers (delays) of edge e_i which is mapped from the corresponding $\vec{d_i}$ in Eq.(2). Fig.3.(b) is a linear systolic array architecture resulting from applying T_1 space-time mapping matrix. The delay 'D' is a pipeline register. For example, to calculate $d_{2,j,k}$ in PE2, we need $d_{2,j-1,k}$ which was calculated q clock cycles before and $d_{1,j,k}$ and $d_{1,j-1,k}$ which were calculated in PE1 one and $q+1$ clock cycles before, respectively.

IV. Experimental Results

A. An FPGA Implementation

To reduce hardware complexity, we pick the following costs, 1 for gap (c_{del}, c_{ins}), 2 for substitution (c_{sub}) if two characters are different or 0 for substitution if two character are same. The values of $d_{i,j-1,k}$ and $d_{i-1,j,k}$ in Eq.(1) are restricted to $d_{i-1,j-1,k} \pm 1$ and the equation can be simple to obtain [8]:

$$d = \begin{cases} a & ,if\ (b\ or\ c)\ = a-1\ or\ (x_{i,k} = y_j) \\ a+2 & ,if\ (b\ and\ c) = a+1\ and\ (x_{i,k} \neq y_j) \end{cases} \quad (5)$$

where $d = d_{i,j,k}$; $a = d_{i-1,j-1,k}$; $b = d_{i,j-1,k}$; $c = d_{i-1,j,k}$

Using Eq.(5), it can be seen that modulo-4 encoding can be used to represent each value. Furthermore, the least significant bit of a and d are always equal, and hence, d can be represented by *only the most significant bit* [9]. So, each character of four bases in DNA molecule can be encoded with a two-bit binary number.

The processing element (PE) shown in Fig.4 can compute Eq.(5). The main improvement of our design is the use of Loop-Up Table (LUT) as RAM-based shift register (SRL16E). We use Verilog Hardware Description Language for the design. Following here is Verilog example code of some part inside one PE.

```
module PE(CLK,RSTin,Yin,EDITin,SHIFTin,Addr,EDITout,Yout,SHIFTout,RSTout);
    ...
    wire [1:0] Xint;
    SRL16E StoreX0(.Q(Xint[0]),.CLK(CLK),.CE(RSTin),.D(Xint[0],
                   .A0(Addr[0]),.A1(Addr[1]),.A2(Addr[2]),.A3(Addr[3]));
    FDRE Y0out(.Q(Yout[0]),.C(clk),.CE(SHIFTin),.D(Yin[0]),.R(RSTin));
    MUXCY MUX1(.DI(1'b0),.CI(Comp_abc),.S(XvsY),.O(muxout));
    ...
endmodule
```

From the code, we can see all components of PE using primitive elements to reduce the area of chip. Furthermore, the design is based on some optimizations for run-time customization [9], i.e. the length of queries and the initialization value of shift registers will be changed at run time. Thus, it occupies only 4.5 Xilinx Virtex-4 slices per PE. The linear array is placed in a serpentine pattern as shown in Fig.5. For simulation, we used Modelsim SE II v5.8a. Fig.6 is simulation waveform of a two-PE array for multiple query search.

Figure.7.a shows the system implemented on an AVNet ADS-XLX-V4-LX-EVL60 board as shown in Fig.7.b. There

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, Ho Chi Minh city, Vietnam

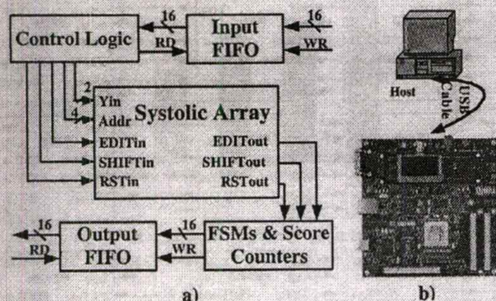


Fig. 7. a) Our systolic array and other logics; b) The overall system

TABLE I

COMPARISON OF FPGA SIMILARITY SEARCH SYSTEMS

System	[10]	[9]	[5]	Ours
# queries (q)	1	1	1	16
# slices/PE	16	4	3	4.5
#PEs/chip	14	7,000	4,032	5,200
#chips/System	272	1	1	1
Model(XC)	4010	2V6000-4	V1000E-6	4VLX60-10
f_{max}(MHz)	N/A	180	202	195
Throughput (BCUPS)	43	1,260	814	1,014
$rate_{data}$(Mbps)	N/A	360.0	404.0	24.375

are input and output FIFOs to buffer the sequence data tranfer between FPGA board and the host computer. The input and output data widths of the FIFOs are both 16 bits. Control logic block also segments 16-bit data from input FIFO to 8 2-bit characters. The final PE of systolic array sends its local edit distance value into 16 finite state machines (FSMs) to calculate the final 16-bit edit distance.

B. Results and Performance Comparison

Our system is synthesized by the Xilinx Foundation ISE 6.3i. Finally, we can fit 5,200 PEs on the Virtex-4 XCVLX60 FPGA which has 26,624 slices. The rest is for interface and control logic. Our maximum clock frequency is 195 MHz, reported by Xilinx Timing Analyzer.

Table.I compares our design with other systems. Other systems can only compare one query against a long data sentence while ours can flexibly compare from 1 to 16 queries simultaneously. We can achieve about the same throughput rate but with much lower $rate_{data}$ than others. For example, the data request rate of [5]

$$rate_{data} = [f_{max} \times E]/q = [202 \times 2]/1 = 404\ Mbps$$

where f_{max} is the system clock frequency and E is the number of bits to encode each character. These systems need expensive FPGA(s) with high-speed interfaces. In reality, they could hardly run at the maximum throughput. Meanwhile, our data request rate is approximately one-sixteenth of others.

$$Our_rate_{data} = [f_{max} \times E]/q = [195 \times 2]/16 = 24.375\ Mbps$$

We could achieve higher computation throughput rate if the FPGA were larger than Virtex-4 LX60. That is because the maximum throughput rate is just a product of $\#PEs$ and f_{max} and $\#PEs$ is proportional to the size of the FPGA. In addition, the size of hardware unit per character of the query string is very small and with comparison up to 16 queries, it is 4.5/16, much lower than others.

Although $f_{max} = 195$ MHz, it operates at 100 MHz clock. The working design is used mainly for verification system with 16 query sequences.

V. CONCLUSION

A novel linear systolic processor array architecture implementation on FPGA called Multiple DNA Matchings is proposed. It can compare simultaneously up to 16 DNA queries against a large DNA database. The main contribution of this work is the deeper pipeline in systolic array that makes the data rate lower than other systems up to sixteen folds. According to our implementation result on a Xilinx Virtex-4 XC4VLX60 FPGA, the achievable throughput is 1.014 trillion cell updates per second with the data rate only 24.375 Mbps.

ACKNOWLEDGMENT

We would like to acknowledge AUN-SeedNet Program of JICA for the scholarship and Xilinx, Inc. for donating the software tools.

REFERENCES

[1] T. F. Smith and M. S. Watermann, "Identification of common molecular subsequence," *Journal of Molecular Biology*, pp. 147:196–197, 1981.

[2] http://www.ncbi.nlm.nih.gov/blast.

[3] http://www.ebi.ac.uk/fasta34/.

[4] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning dna sequences," *Journal of Computational Biology*, pp. 203–214, 2000.

[5] C. Yu, K. Kwong, K. Lee, and P. Leong, "A smith-waterman systolic cell," *13th Int. Conference on Field-Programmable Logic and Applications*, pp. 2778:375–384, 2003,Springer-Verlag LNCS.

[6] S. Kittitornkun and Y. Hu, "Mapping deep nested do loop dsp algorithms to large scale fpga array structures," *IEEE Trans. on Very Large Scale Integration*, vol. 11, no. 2, pp. 208–217, Feb. 2003.

[7] S. Kung, *VLSI Array Processors.* Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[8] R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," *Chapel Hill Conference on VLSI, Computer Science Press*, pp. 363–376, 1985.

[9] K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, and P. Athanas, "A run-time reconfigurable system for gene-sequence searching," *Proceedings of the International VLSI Design Conference*, pp. 561–566, 2003.

[10] D. T. Hoang, "Searching genetic databases on splash 2," *In Proceedings 1993 IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 185–191, 1993.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

FPGA 2006

Fourteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays

Hyatt Regency Monterey
Monterey, California, USA
February 22-24, 2006

King Mongkut's Institute of Technology Ladkrabang

Sponsored by
ACM SIGDA
with support from
Actel, Altera, Synplicity, Trimberger Family Foundation, Xilinx

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Poster Session 3: Applications

High Speed FIR Filter Implementation Using Add and Shift Method

Shahnam Mirzaei, Anup Hosangadi, Ryan Kastner
University of California, Santa Barbara, Santa Barbara, California
Contact email: shahnam@umail.ucsb.edu

Distributed Arithmetic based methods are commonly used to implement Digital Signal Processing (DSP) functions such as filters and transforms. These techniques are very efficient for serial implementation of these functions, but occupy large area when fully parallel implementations for high sample rates are required.

We present a method for implementing high speed Finite Impulse Response (FIR) filters using just registered adders and hardwired shifts. We extensively use common subexpression elimination to reduce the number of adders. Furthermore, we present a new technique to reduce the number of latches required in the design. We compare our designs with those produced by Xilinx CoregenTM and we observe up to 50% reduction in the number of slices for fully parallel implementations. We also observed an average performance improvement of 21.6%.

Manifold Similarity Search of DNA Sequences with Reconfigurable Hardware

Thinh Ngoc Tran, Surin Kittitornkun
Dept.of Computer Engineering, Faculty of Engineering, King Mongkut s Institute of Technology Ladkrabang, Bangkok,10520 Thailand
Contact email: thinh270776@yahoo.com

Shigenori Tomiyama
Dept. of Embedded Technology, School of IT and Science, Tokai University, Japan

This paper presents a reconfigurable processor array for the similarity search of up to sixteen DNA query sequences against a large database. Based on the recently proposed space-time mapping methodology, the processor array is implemented using a Xilinx Virtex-4 XC4VLX60 FPGA on the AVNet ADS-XLX-V4-LX-EVL60 board. The proposed system can sustain a high computation throughput rate with lower data transfer speed from the host database than others up to sixteen folds via a USB 2.0 interface.

Copyright is held by the author/owner(s).
FPGA'06, February 22–24, 2006, Monterey, California, USA.
ACM 1-59593-292-5/06/0002.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Author Biography

1. Personal Data	
Name: **Thinh Tran Ngoc**	Date and Place of Birth: July 27,1976 Ha Bac, Vietnam
Nationality: Vietnamese	Sex: Male
Department, faculty, university: Dept. of Computer Engineering, Faculty of Engineering, King Mongkut's of Institute of Technology Ladkrabang (KMITL)	
Address: KMITL Condominium,Chalongkrung Rd., Ladkrabang, Bangkok, 10520, Thailand	
Mobile: +66-40-13-1417	E-mail: thinh270776@yahoo.com

2. Educational Qualifications

2-1. Academic Qualification

Degree: Bachelor of Engineering Field: Computer Engineering

Year and Duration:1999, 4.5 years

Name of Institution: Ho Chi Minh City University of Technology (HCMUT), Vietnam

4.2004-4.2006: Master Student at King Mongkut's Institute of Technology Lardkrabang (KMITL), Thailand under the JICA Project for AUN/SEED-Net scholarship.

2-2. Research Experience

5.1999-4.2004: Working as a lecturer at HCMUT, Faculty of Information Technology.

Field Research

FPGA technology (Xilinx, Altera), Microprocessor, chip design.

Programming language skills

Assembly, Pascal, C, Visual C++, Visual Basic, Verilog HDL, VHDL.

2-3. List of Publications

(1) Thinh.T.N, S.Kittitornkun, and Y.H.Hu, "Mass-Similarity Search of Biological Sequences using FPGA" The 2nd ECTI Annual Conference, Thailand, May 12-13, 2005.

(2) Thinh.T.N, S.Kittitornkun, and S.Tomiyama, "Multiple DNA Matching with Reconfigurable Hardware", 4th IEEE International Conference on Computer Sciences – Research, Innovation, and Vision for the Future (RIVF'06), Vietnam, Feb 12-16, 2006.

(3) Thinh.T.N, S.Kittitornkun, and S.Tomiyama, "Manifold Similarity Search of DNA Sequences with Reconfigurable Hardware", Fourteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA2006), California, USA, Feb 22-24, 2006.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.