Technical University of Crete

Reconfigurable Architecture Structures for the BLAST DNA Sequencing Algorithm

A DISSERTATION
SUBMITTED
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

of the degree

DOCTOR OF PHILOSOPHY

In the Field of Electronics & Computer Engineering

By

Euripides Sotiriades

Chania 2011

ΣΥΝΟΨΗ

Reconfigurable Architecture Structures for the BLAST DNA Sequencing Algorithm

Η Υπολογιστική Μοριακή Βιολογία ή Βιοπληροφορική είναι ένας από τους πλέον ανταγωνιστικούς κλάδους της επιστήμης του Ηλεκτρονικού Μηχανικού και Μηχανικού Ηλεκτρονικών Υπολογιστών. Τα αποτελέσματα της έρευνας στη Βιοπληροφορική αναμένεται να δώσουν τεράστια ώθηση σε επιστήμες όπως η Βιολογία και η Ιατρική, και να οδηγήσουν σε νέα φάρμακα ή θεραπευτικές μεθόδους. Η Βιοπληροφορική αποτελείται από ένα μεγάλο σύνολο αλγορίθμων, συνήθως υπολογιστικά πολύπλοκων, και τεράστιες συλλογές δεδομένων που αυξάνονται με μεγάλους ρυθμούς. Οι αλγόριθμοι αυτοί έχουν συνήθως διαφορετικές παραλλαγές ανάλογα με τη φύση ή το μέγεθος των δεδομένων. Για την επίλυση οποιουδήποτε προβλήματος Βιοπληροφορικής συνήθως απαιτείται εφαρμογή περισσοτέρων του ενός αλγορίθμων.

Ο αλγόριθμος BLAST, με τον οποίο ασχολούμαστε στη συγκεκριμένη διατριβή, είναι ο πλέον χρησιμοποιούμενος και γνωστός στην κοινότητα της Βιοπληροφορικής. Ο συγκεκριμένος αλγόριθμος χρησιμοποιείται για την αναζήτηση ενός τμήματος του γενετικού υλικού κάποιου οργανισμού (ερώτημα - query) σε μία γενετική βάση δεδομένων. Το αποτέλεσμα του αλγορίθμου είναι ο αριθμός εμφανίσεων του ερωτήματος καθώς και η θέση του μέσα στη βάση, ενώ εξετάζεται και αν υπάρχει κάποιου είδους μετάλλαξη στα δεδομένα ή κακή αποκωδικοποίηση (αλγόριθμος μη ακριβούς ταυτοποίησης).

Η αναδιατασσόμενη λογική (FPGAs) έχει χρησιμοποιηθεί σε σειρά προβλημάτων για την επιτάχυνση του χρόνου εκτέλεσης. Οι FPGAs έχουν χρησιμοποιηθεί σε αλγόριθμους ακριβούς ταυτοποίησης ή Βιοπληροφορικής, αλλά λιγότερο εξελιγμένους από τον BLAST. Στη διατριβή αυτή παρουσιάζεται ένα σύστημα βασισμένο σε αναδιατασσόμενη λογική το οποίο μπορεί να επιλύσει τον αλγόριθμο BLAST ανεξάρτητα με το μέγεθος ή την φύση των δεδομένων. Ο αλγόριθμος έχει μελετηθεί σε βάθος και έχει σχεδιαστεί μία αρχιτεκτονική η οποία εξελίχθηκε σε 4 διαφορετικές εκδόσεις. Η αρχιτεκτονική είναι πρωτότυπη και είναι η μοναδική έως σήμερα που προσφέρει μία εντελώς γενική λύση. Έχει προσομοιωθεί εξαντλητικά και η

ορθή λειτουργία της έχει επιβεβαιωθεί. έναντι των αποτελεσμάτων της κοινά αποδεκτής έκδοσης λογισμικού (NCBI BLAST).

Το τελικό σύστημα υλοποιήθηκε σε εργαστηριακή κλίμακα, κάτι που απαιτούσε την επίλυση ενός μεγάλου αριθμού σημαντικών τεχνικών προβλημάτων.

Η απόδοση του τελικού συστήματος είναι ανάλογη με τη φύση και το μέγεθος των δεδομένων. Η επιτάχυνση κυμαίνεται από μία έως τρεις τάξεις μεγέθους σε σχέση με συμβατικούς υπολογιστές.

ABSTRACT

Reconfigurable Architecture Structures for the BLAST DNA Sequencing Algorithm

Computational Molecular Biology or Bioinformatics is an emerging area for Electronic and Computer Engineering. Bioinformatics research results are expected to have a great impact on Biology and on Medical research, leading to new medicines or treatments for several diseases. The Bioinformatics area consists of several algorithms and datasets, leading to computationally challenging problems. Datasets have exponentially grown in size over the last few years, and the trend continues. The algorithms have several variations, depending on the size and the nature of datasets. Several algorithms are usually combined to solve bioinformatics problems.

The BLAST algorithm is considered to be the most widely used one in the Bioinformatics community, and is used in many Bioinformatics problems, e.g. to find similarity between fragments of genetic data (query) and an organism (database), even if there are mutations or data that are not properly decoded (non exact match algorithm).

Reconfigurable logic has been used in numerous problems to accelerate the execution time of many applications, FPGAs have been previously used to map exact matching algorithms or less sophisticated Bioinformatics algorithms vs. BLAST.

This dissertation presents a system based on reconfigurable logic to implement the BLAST algorithm, regardless of data size or algorithm variation. The BLAST algorithm has been studied in depth and the corresponding architectures have been designed and evolved in four different generations. The architectures are original and unique in offering a completely general solution for all BLAST variations. All architectures have been thoroughly post place and route simulated and the results have been confirmed against results of the most broadly accepted version of software (the NCBI BLAST). In addition, a laboratory prototype system has been build on an off-the-shelf platform and all major technical implementation problems have been solved, including I/O issues.

The TUC BLAST system, which is presented in this work, is one to three orders of magnitude faster than a general purpose computer running the BLAST algorithm.

Acknowledgments

There is a large number of people who have contributed in different ways to this dissertation, either in terms of content or in terms of support.

First, I would like to mention my friend and advisor, Apostolos Dollas. He has taught me all these years (before and during my PhD studies) how to be a good researcher and become a successful PhD student. Without his comments and direction it would be impossible to present this dissertation.

I would like to thank Prof. Dionisios Pnevmatikatos for his interest and his active collaboration at this dissertation. I would also like to thank Prof. Georgios Stamoulis as member of the three-member dissertation committee for his support and useful comments, especially in the first stages of my research.

I want to thank Asst. Prof. Yiannis Papaefstathiou who has also been involved actively at my research, and the remaining members of my committee, Prof. Konstantinos Kalaitzakis, Prof. Manolois Katevenis (of the U of Crete) and Prof. Georgios Stavrakakis for their meaningful remarks.

I also want to thank, Christos Kozanitis for helping me in the development of the first generation of the BLAST architecture; Panagiotis Afratis for his work at BLAST algorithm analysis; Konstantinos Galanakis for his work with the Bloom filters and his professional work with graphics at my presentation; Dimitris Vasilopoulos for the implementation of the TCP/IP FPGA interface which has been used for the experiments; finally, I would like to thank for his great help and active involvement at several stages of this dissertation Grigorios Chrysos.

I would like to thank Kyprianos Papademetriou for his remarks, advice and his fine cooperation all these years my friend and colleague.

Markos Kimionis and a large number of students of MHL provided intellectual and emotional support during these years.

Finally, I would like to thank my family and my wife Athina for making my PhD possible, and for being there when I needed them.

to everyone who supported me all this time,
especially to Athina
and all the members of my family

CONTENTS

LIST OF FIGURES

LIST OF Tables

Chapter 1

Introduction

In this work, a class of computationally intensive problems that arise from the early 70s, Computational Molecular Biology problems or Bioinformatics, was exploited to find how these problems can be mapped to reconfigurable hardware efficiently. The best known and most widely used algorithm of Bioinformatics, BLAST, was selected for that purpose. We studied the algorithm in depth and we designed several different reconfigurable fabric-based architectures. We generalize this architecture in order to implement every variation of the algorithm and for every possible dataset. Designs achieve performance speedup of one to three orders of magnitude faster than a general purpose computer and better performance vs. any other published architecture from the "competition" researchers who have designed similar hardware solutions. An experimental platform was built to test these architectures extensively. In the course of this dissertation, several open problems regarding application specific reconfigurable hardware and data structures for reconfigurable logic were addressed, and several experiments and implementations have been made in actual hardware.

1.1 FPGA Streaming Applications

Since the late 1980s when FPGAs where first introduced, several computationally intensive problems have been mapped into that technology. Reconfigurable hardware proved to be a solution for performance boosting of several algorithms. Several projects[1][2] proved how efficiently this technology can be used; FPGAs offered orders of magnitude better performance vs. general purpose processors in specific problems. Eventually, FPGAs proved not only to be a cost effective rapid system prototyping platform vs. ASIC, but a versatile technology of choice for Image Processing[3][4][5], Automated Target Recognition[6], Data Encryption[8][9][10], Factoring Large Numbers[11], DES [12], Elliptic Curve Cryptography Applications[13], Video Processing [14][15], String Pattern Matching [16], Golomb Ruler Derivation[17][18],

FFT Implementations[19][20], Transitive Closure of Dynamic Graphs[21], Boolean Satisfiability[22], Data Compression[23], Speech Recognition [24], Genetic Algorithms for the Travelling Salesman Problem[25], and Arithmetic Applications [26][27] – to name a few.

Nowadays modern devices offer significant resources in addition to the reconfigurable fabric. Special I/O transceivers, dedicated logic blocks for memory, powerful general purpose processors on chip, special modules for digital signal processing, and fast floating point operations are the best known features of a modern device. Even the reconfigurable fabric has been changed, offering more logic, better routing resources and run time reconfiguration characteristics. In addition, a large collection of functional Intellectual Property cores (IPs) is freely available to the designer through IP generator tools such as the Xilinx Core Generator, or, distributed by designers through web sites such as OpenCores[28].

All these available resources help designers to take up with new applications, with considerable results on network systems, [29][30][31] and especially on network intrusion detection systems [32] [33] [34] [35] [36] [36]. In general, these were the first class of applications that came from a new category, i.e. Data Streaming.

Data streaming applications become much more significant these days due to the technological advances of FPGAs, mostly in the forms of I/O transceivers on a chip and large amount of available memory.

1.2 Bioinformatics

The application of information technology to the field of molecular biology is called *Bioinformatics*. The double-helix form of DNA was discovered in 1953, increasing the ability to manipulate biomolecular sequences and a huge amount of data was generated from laboratories all over the world. Since then several new problems have arisen. Biologists produce enormous amounts of data which has to be stored and organized in several databases, to process them, and to create new algorithms - usually of high complexity. Finally, and after all these processes, biologists get the data that they need in order to have their biological conclusions. Bioinformatics was initially developed since the early 1970s and nowadays it has a tremendous evolution, offering more accurate

and powerful tools to biologists. On the other hand, rapid developments in genomic and other molecular research technologies offer raw data in at rates faster than Moore's law [37][39]. As a result, a geometrically progressing volume of data production has created huge databases containing DNA, RNA, and protein sequences. Such databases include GenBank [40], EMBL[40], PIR[42], GSDB [43], DDBJ [44], EBI [45], and Swiss-Prot[46]. Sequence comparison, especially in DNA or protein databases is one of the most common computations that molecular biologists execute. This is the reason why bioinformatics is a challenging area after almost forty years of development.

1.3 Contribution of this Thesis

This thesis is one of the first systematic approaches to build special-purpose hardware for the computational biologists' algorithm of choice, BLAST. Internationally, there were two additional independent research efforts in this general area, against which this dissertation is compared, with distinct contributions vs. those of other groups. More, specifically, in this dissertation:

- We developed architectures that map the BLAST algorithm for any size of query, any size of data base, and in any of its five variations more efficiently than any other implementation.
- We have studied the algorithm, designed several architectures, built actual reconfigurable logic based hardware on several platforms, ran several experiments and further improved our design.
- We have developed data structures in hardware which were appropriate for BLAST, vs. previous research efforts which were largely in algorithms suitable for simple, systolic array computations. Based on our know-how from the first architecture, successive iterations led to simpler, faster, and more general architectures for the same algorithm.
- We have measured speedup against a general purpose processor that varies between one and three orders of magnitude, depending on input dataset and algorithm variation.
- We showed that this form of computing is more cost effective, both in terms of the platform cost as well as in power requirements vs. general-

purpose computing. This result applies to reconfigurable processors for a general class of bioinformatics problems, and is not specific to BLAST only.

- We have used a high-end platform and fast serial transfer protocol (Gigabit Ethernet), with a proper interface to a PC, in order to prove through experimental testing that I/O does not limit our design in general. This result is in its own right useful, as BLAST was thought to be so I/O intensive that any computational speedup through dedicated hardware would not be very useful due to I/O bottlenecks. Our research shows that there can be reconfigurable logic-based servers that run BLAST in its full complexity and with results rivalling those of grid computers at a fraction of the cost and energy requirements per calculation.
- We have showed with our research that bioinformatics algorithms can be mapped efficiently to reconfigurable hardware and this can be a viable and promising research direction.
- We also give a solution to FPGA streaming problems in which latency is not a limitation, and this approach can be generalized to other fields of applications.

1.4 Thesis Organization

In Chapter 2 we describe the main areas of Bioinformatics with a focus on database search algorithms, and more specifically to BLAST. In Chapter 3 we refer to all previous efforts for high performance architectures for Bioinformatics hardware with emphasis on genetic database search and BLAST algorithm implementations. In Chapters 4 to 6 we describe all the proposed architectures of this work. Chapter 7 includes all system improvements that we have made during the implementation phase and in Chapter 8 system implementations and verification on several FPGA platforms are presented. Chapter 9 refers to implementation issues, performance measurements and comparisons to the state of the art software and hardware, while Chapter 10 concludes and suggests future work.

Chapter 2

Bioinformatics Problems and Algorithms

A brief introduction of bioinformatics problems is given in this chapter in order to increase the readability of this dissertation. Naturally, substantial literature exists and it should be consulted by readers who want a more in-depth knowledge of the field. This chapter also focuses on the specific problem of sequence comparison of genetic data and ends with the algorithm selection and its description.

2.1 Bioinformatics Problems

Bioinformatics consist of several problems of DNA and RNA data manipulation. These problems are separated to six main categories:

a) Sequence comparison: This is the degree of matching between two or more long sequences (comprising typically of characters A,T,P,G, but possibly with as many as twenty characters). Biologists either use character sequence matches on their own or as part of almost any other problem category.

b) Fragment assembly: Biologists try to assemble the complete genome of an organism from parts that came out of a sequencer.

c) Physical mapping problem: This can be considered as fragment assembly on a larger scale. Fragments are much longer, and for this reason assembly techniques are completely different. The aim is to obtain the location of some markers along the original DNA molecule.

d) Phylogenetic tree: Reconstruction of the tree of life in order to understand evolution. It is a complex problem and several methods have been developed. All of them are computationally demanding and several projects spend even millions of CPU hours on this problem.

e) Genome rearrangements: It has been discovered that some organisms are genetically different, not so much at the sequence level, but in the order in which large similar chunks of their DNA appear in their respective genomes. Interesting mathematical models have been developed to study such

differences.

f) RNA structure prediction and protein structure prediction: The understanding of the biological function of molecules is actually at the heart of most problems in computational biology. Because molecules fold in three dimensions and because their function depends on the way they fold, a primary concern of scientists in the past several decades has been the discovery of their three-dimensional structure, in particular for RNA and proteins. This has given rise to methods that try to predict a molecule's structure based on its primary sequence.

In order to solve almost any problem of these categories bioinformatics researchers use a method which consists of several algorithms; a typical number is 4 to 5 and usually 2 or 3 of them are computing intensive.

All these problem categories are very important for biologists and the effectiveness of their solution has significant impact on research in molecular biology, new drug design, on new medical practice development, on genetic engineering, with main emphasis on genetic diseases and mostly on cancer research. All these problem categories have huge datasets and effectively all of the algorithms and methods that have been developed to solve these problems are of high complexity.

The BLAST algorithm is considered to be the most significant among bioinformatics algorithms. BLAST software is considered as one of the most widely used bioinformatics programs [47] and BLAST is a component of many other algorithms [52]. For this reason there exists a common effort to improve the BLAST software. This effort is the end results from the work of many groups, and the coordination and final BLAST release takes place at the National Center for Biotechnology Information (NCBI), where BLAST is the main project of the Institute [40].

Due to the wide use and the global acceptance of the BLAST algorithm from the scientific community, it was selected for speedup through special-purpose architecture in our work.

2.2 Genetic Database Search Algorithms

In 1970 Needleman and Wunch [48] developed an algorithm based on dynamic programming in order to produce an optimal global alignment of two sequences. However, as global alignment produced poor results for the biologists' needs, local alignment algorithms were preferred by the biologists, i.e. algorithms that try to find alignment between sub strings of the given sequences. Smith and Waterman [49] developed an algorithm for local alignment which uses dynamic programming as well and produces optimal results. However, its quadratic time complexity (O(mn) where m is the size of the database and n the size of the query) makes the Smith-Waterman algorithm unattractive as it is computationally too intensive.

Due to the need for faster, if not fully optimal searches, heuristic algorithms were developed for the same problem, such as the FASTA [50] algorithm and subsequently its improvement, BLAST [51]. Both algorithms use a heuristic and provide near optimal local alignment, but their strong statistical background makes them a powerful tool for computational biologists today. At present, BLAST is the most popular algorithm not because it is faster than FASTA, which is arguable, but because its implementation is open source and there is also a web interface at the NCBI website [40]. This interface enables everyone to perform queries against biological databases.

2.3 BLAST Algorithm Description

BLAST is the acronym of Basic Local Alignment Search Tool and it has been introduced by Altschul *et. al*. in 1990 [51]. Table 1 shows the different BLAST programs depending on the nature of the data to be processed (nucleotides have a four letter alphabet, amino acids have a twenty letter alphabet and there are cases of both forms involved in a search). It should be noted that when databases are mentioned, only the set of catalogued sequences are referred to and not any other database feature such as multiple access, fast retrieval etc. The outputs of the algorithm are the positions of substrings of the database and the query that have similarity as well the corresponding score. These pairs of database and query regions are called High Score Pairs (HSP). The score has significant value for biologists because it is used to compute several statistical

variables, the most important of which is the e-value (which is discussed below).

program	*Inputs*
Blastp	Query: amino acid, database: amino acid
Blastn	Query: nucleotide, database: nucleotide
Blastx	Query: translated nucleotide sequence, database: amino acid
Tblastn	Query: amino acid, database: translated nucleotide sequence
Tblastx	Query: translated nucleotide sequence, database: translated nucleotide sequence

Table 1: BLAST Versions

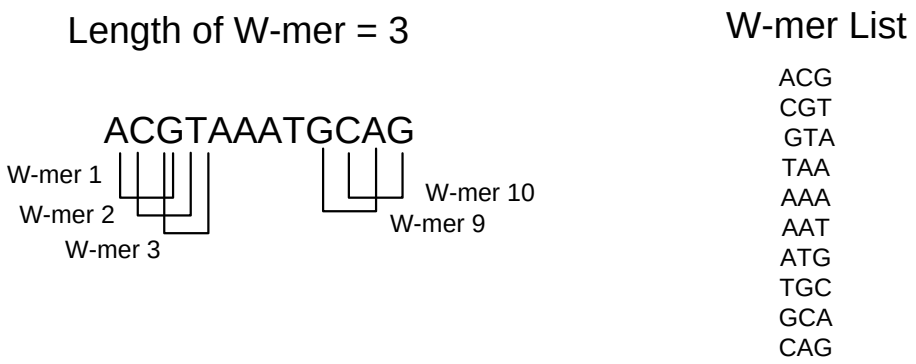

Figure 1: BLAST Algorithm Step 1

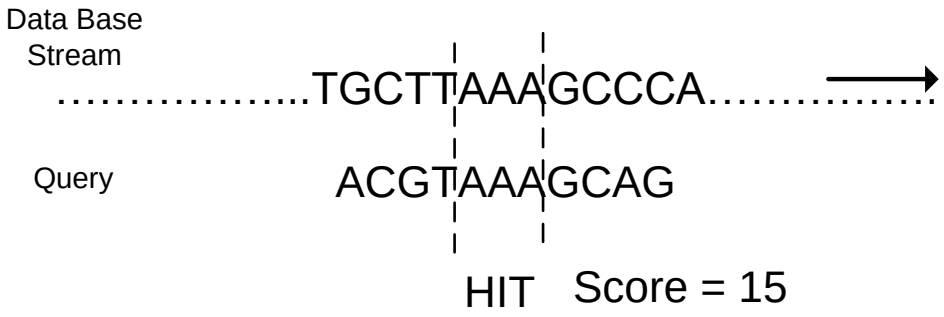

Figure 2: BLAST Algorithm Step 2

The BLAST algorithm comprises of three steps. In the first step the query is compiled to form a list of all the contiguous substrings with length w, which are called W-mers. For example let the string ATGAACCTGAATACTGGGTTACCT be the query sentence of length 24 and let w, the length of W-mers, be equal to 8. The word list will contain 17 W-mers.

ATGAACCT will be the first

TGAACCTG will be the second

GAACCTGA will be the third, etc. and

GGTTACCT will be the last one.

In the *First Step* the complete list of W-mers of the query is created.

The Second step is the search of the database for "hits". After the word list generation, the database sequences are searched for an exact match between any substring of the W-mers list and the database sequence. Every word of the word list that is found in the database is called a hit and it is possible to be part of a High Score Pair (HSP). The list of the generated "hits" is processed in the *third step*, shown in Figure 3.

Each substring which yielded a match in the second step is extended locally in both directions until the score of this substring no longer gets improved under the scoring rules. The scoring scheme typically used for amino acid sequences is derived from the PAM matrices. However, as both the query and the database consist of nucleotide sequences, a simpler scoring scheme is used in BLAST, where each match is scored with +5 and each mismatch is scored with -4. This scheme may produce results slightly different than those with the use of PAM or BLOSUM matrices but biologically it does not have significant impact.

The PAM matrices were introduced by Margaret Dayhoff [77] in 1978 based on 1572 observed mutations in 71 families of closely related proteins. PAM stands for Point Accepted Mutation (PAM) or Percent Accepted Mutation, and is a set of matrices used to score sequence alignments. Each matrix is twenty-by-twenty (for the twenty standard amino acids); which has the score for every pair of proteins.

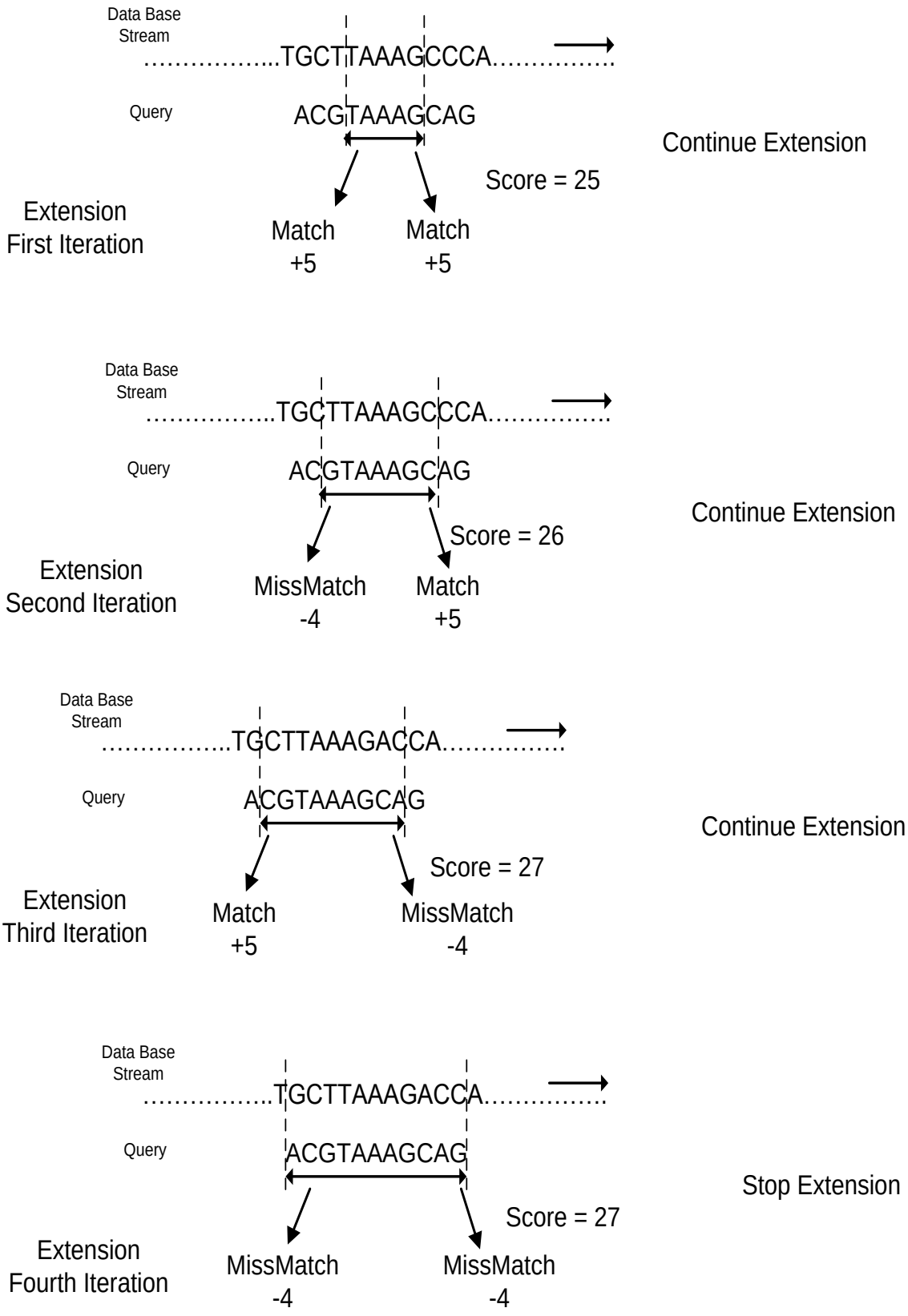

Figure 3: BLAST Algorithm Step 3

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
Ala A	13	6	9	9	5	8	9	12	6	8	6	7	7	4	11	11	11	2	4	9
Arg R	3	17	4	3	2	5	3	2	6	3	2	9	4	1	4	4	3	7	2	2
Asn N	4	4	6	7	2	5	6	4	6	3	2	5	3	2	4	5	4	2	3	3
Asp D	5	4	8	11	1	7	10	5	6	3	2	5	3	1	4	5	5	1	2	3
Cys C	2	1	1	1	52	1	1	2	2	2	1	1	1	1	2	3	2	1	4	2
Gln Q	3	5	5	6	1	10	7	3	7	2	3	5	3	1	4	3	3	1	2	3
Glu E	5	4	7	11	1	9	12	5	6	3	2	5	3	1	4	5	5	1	2	3
Gly G	12	5	10	10	4	7	9	27	5	5	4	6	5	3	8	11	9	2	3	7
His H	2	5	5	4	2	7	4	2	15	2	2	3	2	2	3	3	2	2	3	2
Ile I	3	2	2	2	2	2	2	2	2	10	6	2	6	5	2	3	4	1	3	9
Leu L	6	4	4	3	2	6	4	3	5	15	34	4	20	13	5	4	6	6	7	13
Lys K	6	18	10	8	2	10	8	5	8	5	4	24	9	2	6	8	8	4	3	5
Met M	1	1	1	1	0	1	1	1	1	2	3	2	6	2	1	1	1	1	1	2
Phe F	2	1	2	1	1	1	1	1	3	5	6	1	4	32	1	2	2	4	20	3
Pro P	7	5	5	4	3	5	4	5	5	3	3	4	3	2	20	6	5	1	2	4
Ser S	9	6	8	7	7	6	7	9	6	5	4	7	5	3	9	10	9	4	4	6
Thr T	8	5	6	6	4	5	5	6	4	6	4	6	5	3	6	8	11	2	3	6
Trp W	0	2	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	55	1	0
Tyr Y	1	1	2	1	3	1	1	1	3	2	2	1	2	15	1	2	2	3	31	2
Val V	7	4	4	4	4	4	4	4	5	4	15	10	4	10	5	5	5	72	4	17

Figure 4: PAM 250 Matrix

BLOSUM (BLOcks of Amino Acid SUbstitution Matrix[78] is a substitution matrix used for sequence alignment of proteins. BLOSUM are used to score alignments between evolutionarily divergent protein sequences. BLOSUM is based on local alignments. BLOSUM was first introduced in a paper by Henikoff and Henikoff[78]. Figure 5 shows BLOSUM 62.

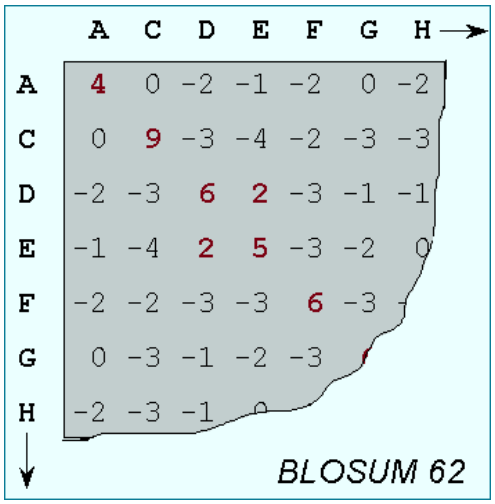

Figure 5: BLOSUM 62(NCBI site)

Detailed description of the scoring scheme for NCBI distribution of BLAST has not been published and for that reason it is not possible to calculate exactly the score as NCBI distribution does.

Chapter 3

State of the Art

3.1 Previous Work on Several Molecular Biology Problems

3.1.1 Previous Work on Genetic Database Search

The reconfigurable hardware community used DNA sequence matching and database search as one of the first problems to show how computationally intensive problems can be solved using FPGAs. The venerable Splash 2 platform was used during the early 1990s by Hoang et. al. [53][54] to solve this problem using the Smith Waterman algorithm. Later Guccione et. al. [55] used Jbits technology and both the Virginia Tech Configurable Computing Laboratory [56] and Nanyang Technological University [57] used run time reconfiguration for the same problem, also implementing the Smith Waterman algorithm.

Common characteristic for all these projects is that they use problem of genetic database search in order to demonstrate the computational power of reconfigurable logic and use all the state of the art technology at the time to do it.

3.1.2 Previous Efforts on BLAST

Few academic approaches for the BLAST algorithm implementation have been done to date. The first is the RC – BLAST project [57] where the designers fully implemented the computationally heavy part of the NCBI BLAST algorithm but overall performance of this project was reported to be poor, even worse than the corresponding software implementation and no further efforts where given for this project. Even so, RC – BLAST remains important to date as the first full BLAST implementation in reconfigurable technology.

A more recent academic effort for DNA sequence matching and database search was presented at 2005 by the CAAD Lab at Boston University. The efforts of this group, though independent of this dissertation, parallel our own work and can be directly

compared to the present thesis. They have implemented the BLAST algorithm for small queries of up to 800 elements [58] and it was extended and implemented later [60], providing significant (one order of magnitude) speedup against a software implementation. This project is called TreeBLAST. An extension of TreeBLAST presented at 2009 [78] was database prefiltering. This method had already been implemented and published from our own TUC research in 2008[79].

The parallel Mercury BLAST [61][62][63]architecture was introduced by Washington University in St. Louis, implementing BLASTn and offering a good speedup against software running on a general purpose computer. Mercury BLAST is a still going project and it seems that Washington University and the inventors intent to commercialize the results of their research [64]. Mercury BLAST also uses BLOOM filters, as TUC for database filtering but they use it for actual BLAST implementation and not for database formation.

Most recently BLAST implementation using reconfigurable logic was presented from IRISA, CNRS at France and the Institute of Computing Technology at Beijing, China [65]. It is a board with reconfigurable logic coupled with a 64GB FLASH memory and it was used to implement the BLASTx/TBLASTn/TBLASTx algorithm.

Multi-seed/ Multi-channel BLAST is the most recent effort from Chinese National University of Defense Technology which is also reports very interesting results for a generic architecture for BLAST algorithm[66][67].

In parallel with academic efforts, commercial efforts have taken place to run efficiently the BLAST algorithm, Timelogic Inc. reports for its system DeCypher [68] very impressive results on BLAST implementation, but without giving many details about data sets that were used to run experiments, the techniques that are used to measure its results, or even details for its system's number of chips or the I/O strategy used. For those reasons their results can not be compared to others. The University of California at Berkeley build the BEE engine as a configurable supercomputer and one of the applications for this project was the BLAST algorithm [69]. The BEE-2 engine was reported to be twice as fast as DeCypher without providing performance details and for this reason its results cannot be compared to others. The latest commercial effort was announced last year (2007) from Silicon Graphics (SGI) and a software company named

Mitrionics announcing very impressive results but following Timelogic policy didn't give any detail at all [69]. The interest of well known companies about BLAST algorithm is an additional proof of its importance, and more over the use of reconfigurable logic for bioinformatics problems was selected as the No 5 trend for 2007 by the bioinformatics community [70] .

Overall, Mercury BLAST, TreeBLAST, FPGA/FLASH and Multi-seed/ Multi-channel BLAST are the most active projects, with very interesting architectures and results and can be considered as "competitors" to TUC BLAST project.

3.2 SW Implementations

Collective efforts by many groups for DNA sequence matching and database search are collected at the National Center for Biotechnology Information (NCBI) [40] which dominates the area of providing open source tools to implement the BLAST algorithm. These implementations have been repeatedly used as computer benchmarks, by major computer manufacturers such as IBM[70], DELL[71], and Apple[73]. IBM uses the BLAST algorithm as a performance benchmark for its pSeries 375 MHz POWER3-II symmetric multiprocessor (SMP) and the 1.1 GHz POWER4 pSeries 690 Model 681, and provides detailed performance information for a large range of queries. DELL uses BLAST as a benchmark for streaming applications to compare four different computers: PowerEdge 3250 (with Intel Itanium processor at 1.5 GHz), PowerEdge 1750 (with Intel Xeon processor at 3.2 GHz), PowerEdge 1850 (with Intel Xeon processor at 3.2 GHz), and PowerEdge 1850 (with Intel Xeon processor at 3.6 GHz); with very large query sizes—94,000 words; 206,000 words; and 510,000 words. DELL does not provide detailed performance information but only relative performance and speed up. Apple also uses BLAST as a performance benchmark of its systems called dual 2 GHz PowerMac G5 and dual 800 MHz PowerMac G4, and they do provide detailed performance information such as execution time for several queries and databases. The use of the NCBI BLAST software as a benchmark for several computers shows the great importance of BLAST algorithm and how computationally demanding it is.

Chapter 4

TUC BLAST Architecture: 1st Generation

The first TUC BLAST generation was build in order to find out if reconfigurable hardware can offer performance boosting for the BLAST algorithm and to understand algorithm and the implementation problems deeply. The target technology was Xilinx Virtex 4. Algorithm study and design of the architecture was work for this thesis, whereas the VHDL modelling which is part of this work was the diploma thesis of C. Kozanitis. Three publications came out of this work [81][82][83].

4.1 Architecture Analysis

The Technical University of Crete (TUC) architecture, described in this chapter, was designed for BLASTn small query implementation (1000 letters) regardless of the data base size. Query sequences can be divided to three cases: small sequence which is between 100 to 2000 characters, medium which is between 2000 and 50000 characters, and large which is between 50000 and 200000 characters. Data base size can also be divided at three cases; small, medium, and large. Small consists of 4.7 × 106 characters, medium is between 5×10^6 and 200×10^6, and large is between 200×10^6 and 4×10^9 characters. NCBI codes consist of several hundreds of files calculating the BLAST algorithm and exporting several numbers which have biological meaning. All these numbers are calculated based on the score of HSP. These calculations produce substantial computing load but the most significant part of the computation power is consumed to find every HSP and extend it, calculating its score. Previous efforts for hardware implementation of BLAST using profiling show that almost 80% of CPU time is spent on these calculations [57].

The TUC architecture is divided into N identical computing machines, each one of which implements all three steps of the algorithm. Input data have a width of 2N bits, and come from N different channels. Every channel drives one of the N computing engines. Every machine has two major subsystems, one for step 2 of the algorithm and one for step 3. The first step of the algorithm (the W-mer calculation) is precalculated

before algorithm is run. The precalculation results are the first inputs for the machine and they are stored in the memory, together with their position in the query. After this procedure the data stream of the database starts to be processed and if a match is found the second component of the architecture is activated and starts to extend the match, thus implementing the third step of the algorithm. The general design of the architecture is shown in Figure 6.

To illustrate in more detail, before each machine starts the database search, its setup mode asks for the precomputation of W-mers, with their position in the query and their loading to the corresponding memories. This procedure takes about 1000 cycles for 1000-character long queries. The input of the system in normal mode (database search) is the database stream, one character for each machine. Only the 10 MSBs of W-mers are stored in memory and at the address which corresponds to their 12 LSBs. The stored bits are called W-mer tags. The width of the memory is 23 bits, 10 for the W-mer tag, 1 for valid, and the remaining 12 to show the position of the corresponding W-mer in the input query.

4.2 Hit Finder Unit

The Hit Finder Unit except for the W-mer memory that was previously described has an input buffer which is 2 bits wide (1 character) and one thousand positions deep, called Future memory. The data stream from the input channel passes through this buffer. As long as there is no hit the buffer operates as a FIFO, getting 2 new bits from the stream in every cycle and driving one shift register (22 bits long) that shifts 2 bits (one letter) per cycle. That shift register has one eleven letter long substring, which is compared with all the W-mers. The 12 LSB of the shift register address the W-mer memory in order to read the W-mer tag. The W-mer tag is compared with the remaining 10 MSBs of the shift register. When a hit is found the Future memory continues to push its data to the shift register and starts to send them at the extension unit as well for the 3rd step of the algorithm. A new comparison is made during every cycle in which the shift register has new data. Conditions for a hit are to have two equal strings in the shift register and the W-mer memory, and the memory content to be valid. Figure 7 shows the Hit Finder Unit architecture. If a second hit comes when the previous is still extended the

whole system goes to a stall mode. The system stops trying to find new hits and signals external devices to stop sending new data. In this case the extension unit operates in the normal mode. The Hit Finder unit stops normal operation but continues to pass the data stream to the extension unit.

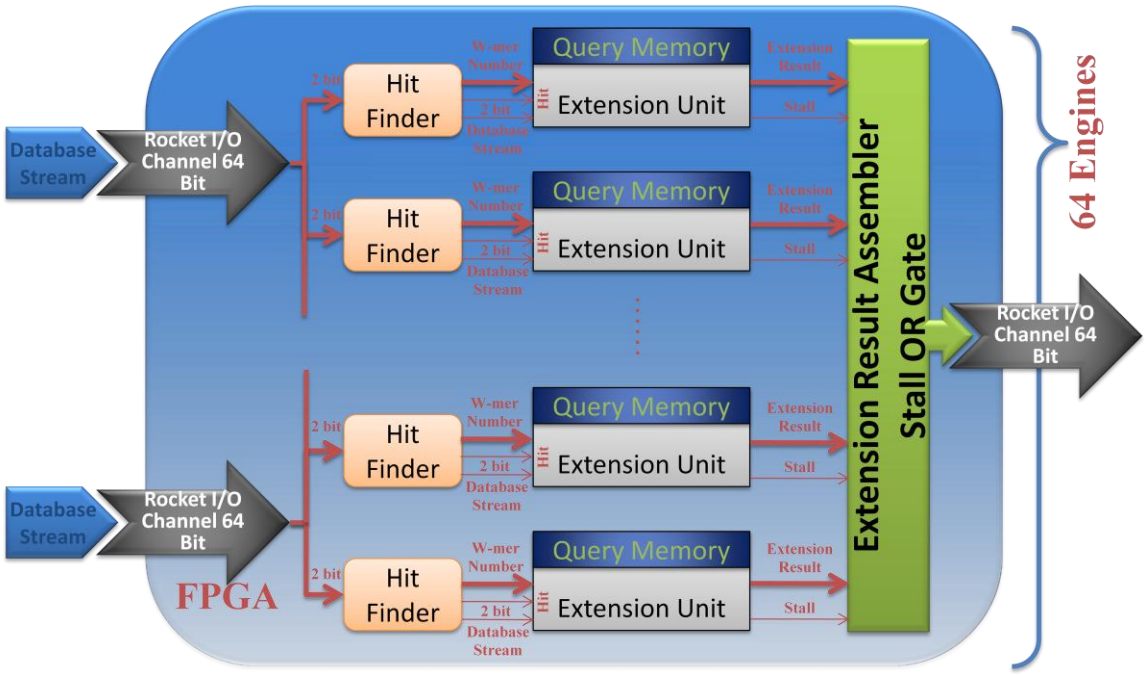

Figure 6: General Architecture Scheme of BLAST Machine 1st Generation

4.3 Extension Unit

The Extension Unit executes two comparisons in every cycle, according to the algorithm. It extents both sides and compares the two pairs of letters. The first pair comes from the query memory and the history memory and the remaining couple comes from the Query memory and the Future memory. The data from the input are buffered in the History and Future memories, as it can be seen in Figure 8. There are also counters and registers that keep several useful data, such as hit position for query and database, its length, and the score (which is the most important result to be calculated). Based on the score all the remaining useful data for biologists (e.g. e-value) can be calculated.

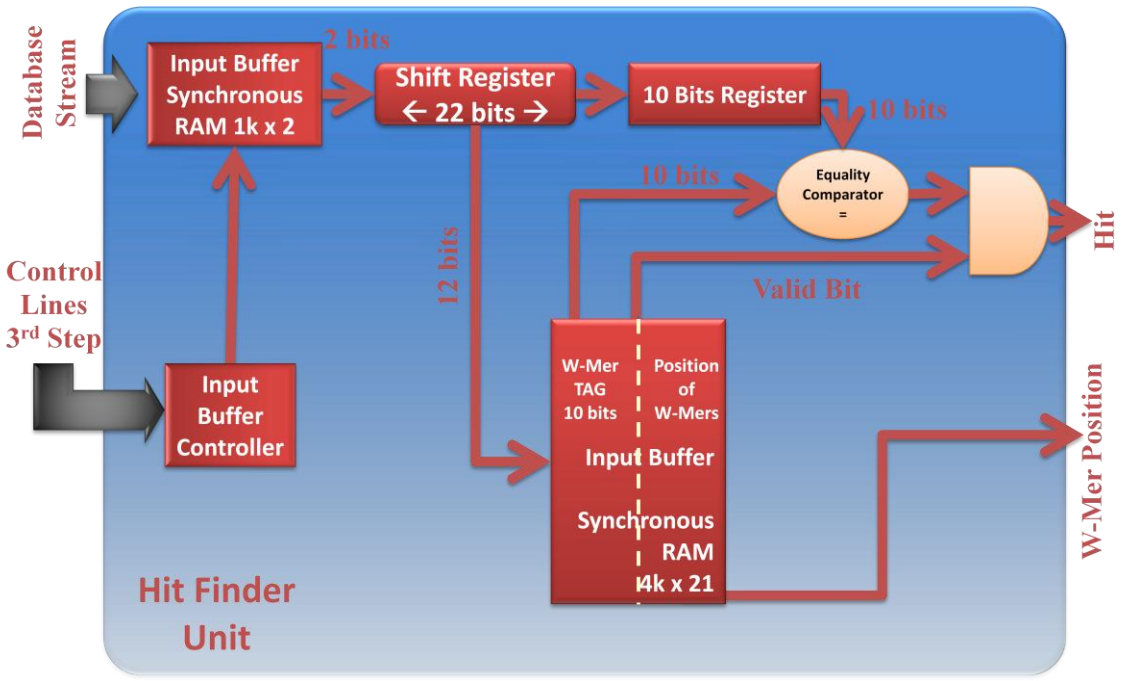

Figure 7: Hit Finder Unit Architecture

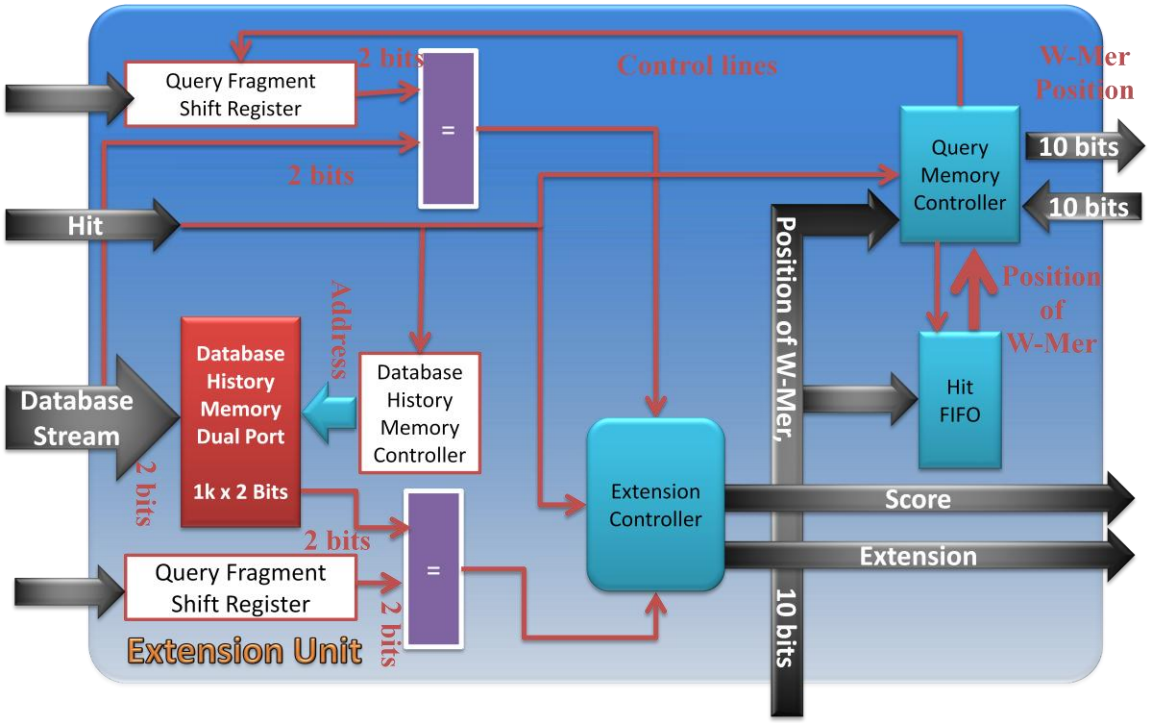

Figure 8: Step 3 Architecture

4.4 Conclusions

This design, of the first generation architecture, offered significant speedups which are presented in the corresponding chapter of this thesis, and proved that it was worth to put further effort on the BLAST algorithm. It also pointed the hard problems of a system design for the BLAST algorithm which were the memory amount limitations, I/O problems and algorithm step 3 efficient mapping at reconfigurable fabric.

Chapter 5

TUC BLAST Architecture: 2nd Generation

The aim of the design of the second TUC BLAST Generation was to build a more flexible architecture for BLASTn variation which could handle dataset of arbitrary size. Limitations of RAM and difficulties of mapping the 3rd step of algorithm at reconfigurable logic were the main problems of the first generation that we had to solve. For that reason every available resourse of the FPGA had to be used. In order to maintain high performance and to get flexibility and a SW/HW system was finally proposed. This is also the first effort we know of in the literature, for which in-depth sizing calculations and SW/HW partitioning were incorporated in the architecture, especially regarding the resources for Step 3 of the algorithm.

5.1 Software Hardware Partitioning

The BLAST algorithm consists of the three steps shown in Chapter 2. The first step is the W-mer calculation which is not computationally intensive and takes a negligible amount of the total execution time. The second step is the comparison step which is computationally the most intensive and must be executed for all the elements of the database, i.e. several billions of elements. The third step of the algorithm is also computationally intensive for those elements which require it, however, it does not need to be executed for all the elements of the database but only for a small percentage of them (the hits from Step 2). According to these characteristics of the algorithm, and considering the target technology, partitioning of the algorithm to hardware and software implementations has to optimize the total execution time.

Therefore, ideally we need a large reconfigurable fabric with sufficient on-board memory for Step 2 of the algorithm, and a fast processor for Step 3 of the algorithm, and sufficient aggregate input bandwidth (output is not an issue) so that the system will not be I/O starved. An initial thought for the target technology for this design was the use of the Virtex 4 family which consists of several units in addition to the reconfigurable fabric

and routing resources. In more detail it consists of 2 Power PC, a large number of Block RAMs (called BRAM), up to 24 Rocket IO transceivers, and DSP units.

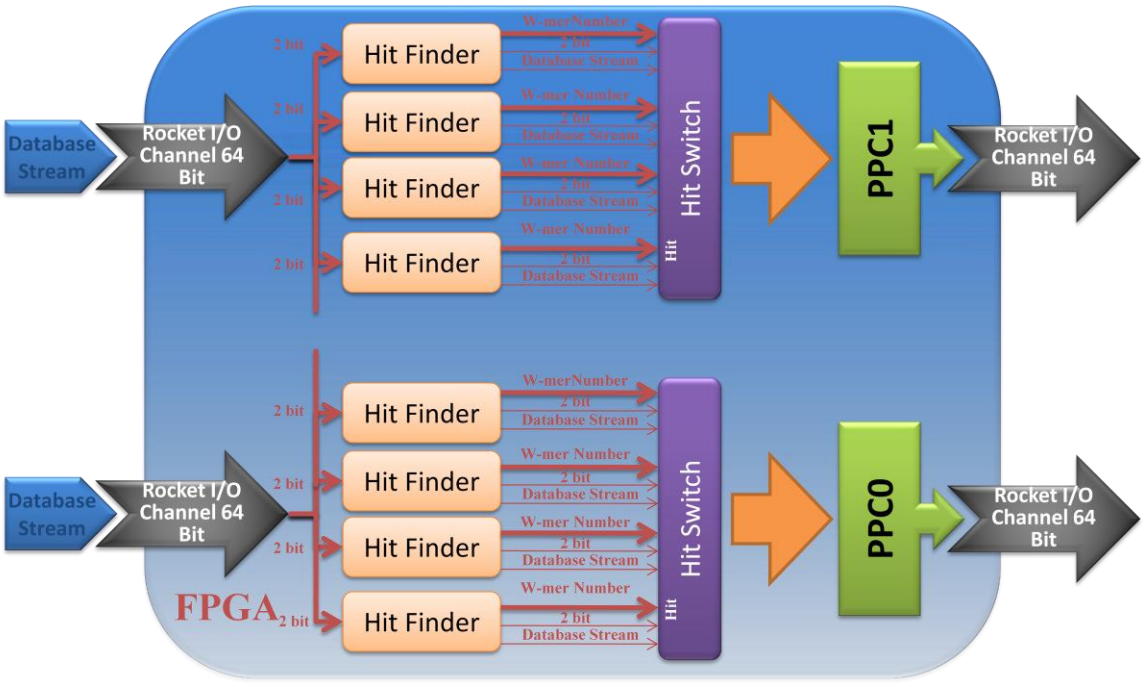

Figure 9: TUC BLAST Machine Architecture 2nd Generation

In the context of present research extensive studies of the algorithm were made. Many runs of the algorithm showed that only a very small number of the comparisons produce a HIT. Depending on the database and the length of the query the comparison may vary but the actual percentage of hits is typically 0.0046%. For every hit and depending on the query size, the execution unit consumes in the worst case time proportional to the size and in the mean case time proportional to one fourth of the query size. Making a rough calculation regarding the extension unit of the first generation architecture for one thousand element queries, such a unit remains 97.5% of the execution time inactive but it consumes 50% of the FPGA BRAM resources which are the critical for computation parallelisation . On the other hand in the same device there are two powerful IBM PowerPC 405 processor cores that can be exploited. Consequently, software/hardware partitioning can be done between Step 2 and Step 3 of the algorithm. Step 1 is the initialization of the hardware. Step 2 is executed on the reconfigurable fabric and its results are processed with software on the embedded Power PC. The general

scheme of the second generation of this architecture is shown in Figure 9. The design overhead for this architecture is the hit switch unit which has to check for hits and to switch properly incoming results from each machine to the Power PC.

5.2 Architecture Analysis

5.2.1 2nd Step Unit

In Figure 10 the new architecture for the second step of the BLAST algorithm is shown. Incoming data are coming from system I/O and have a width of 2 bits. Data are stored to a 1000-position deep FIFO, called Future Memory. The incoming data rate is 2 bits per cycle. The output of the Future Memory FIFO drives the Shift Register and the Data Grouping Unit. The Data Grouping Unit is a unit where data are grouped in 16-bit words and then they are stored in the History Memory. The History Memory collects all data as long as they might be useful for the third step of the algorithm. If a HIT is found the data are driven to the Power PC for extension, else if a HIT is not found at the time window that these data are useful they are overwritten by new data coming from the Future Memory.

The output of the Future Memory also drives a 24-bit Shift Register which has a width of 2 bits and a length of 12 bits. This register contains every possible W-mer and it is compared to the W-mer list. The W-mer list has been implemented as two 4K x 1 bit RAMs each, in which the 14 LSB and the 14 MSB of Shift Register are the input read addresses. If the content of both RAMs is 1 then there is a possible hit. The hit signal activates Control Unit which produces the possible Hit identity and writes it to HIT FIFO. It also produces several control signals, which are not shown in the figure, and collects all required data for extension to the History Memory Unit. When all the data are ready for the extension, the Control Unit signals that is ready for data transfer. When the lower layer of the design signals that is ready for the data transfer, the Control Unit drives the Hit FIFO and the History Memory.

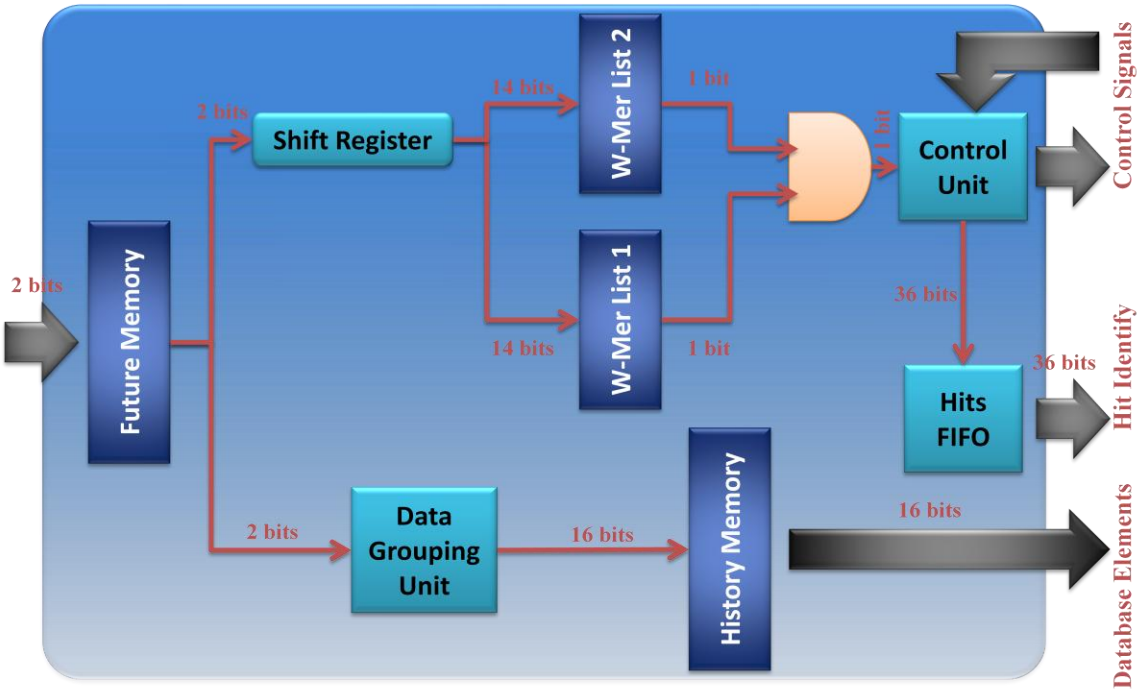

Figure 10: 2nd Step Architecture

The output of each machine goes to a Packaging Unit of the protocol part and then through a Packet Switch and a Packet FIFO to the OPB Bus (i.e. the interface between the reconfigurable fabric and the Power PC) and through it to the Power PC processor. It should be noted that in this implementation the hits that are found are potential hits. This is done because the examined string is not examined against the W-mers but against two parts of the W-mers which are used in effect as a hashing function. With such an implementation the space that is needed for W-mer list is one BRAM or 2 RAMs with 214 address space and not 224 address space which can not be implemented with the total BRAMs of a single device. The tradeoff of increased "false hits" that will be properly weeded out is well worth the thousand-fold reduction of the on-board BRAM memory requirements.

5.2.2 Communication Protocol (Step2 to PowerPC)

The outputs of the Step 2 architecture are inputs to the communication protocol between the reconfigurable fabric and the PowerPC. The communication protocol consists of hardware and software implementation and operates data transfer through the

OPB bus of the target technology. The hardware architecture is shown in Figure 10, where every Step 2 processing unit is connected with a packaging unit and their outputs are concatenated in 32 bit wide packets of variable length. Figure 11 shows the packet which consists of two words as a header and up to 64 packets of data. The length of a packet depends on whether a possible hit follows an earlier hit or not. If the possible hit does not follow an earlier hit then the data packet length will be 64 words of 32-bits of database data and two 32-bits words of hit id, because all the needed data have to be transmitted. If the possible hit follows an earlier hit then some of the required information has already been transmitted to the Power PC and only the remaining information has to be transmitted. The amount of information depends on the distance between two hits and it may need only the header of the packet to be transmitted without any data.

The Packet Switch checks on every cycle if there is a HIT at any machine. When a HIT is detected then it starts to format the transfer packet, forming the header and then it starts to read all the data. All the information produced from the Packet Switch is written directly to the FIFO of the OPB bus. When the switching is complete, it waits for the next HIT. If several HITs are produced in the same cycle, then the HIT that comes from the same processing element is served. With that feature, when a machine has a succession of HIT's, which is quite possible to happen in this algorithm, then the new transfer packet will contain only the needed data and no data stream will be repeated. The remaining units are not starved, as the Power PC becomes the critical resource through which all possible hits will be serviced.

Machine ID **Packet Length** **Hit Id.**
<14 bits> *<14 bits>* *<4 bits>*
Hit Id. *<32 bits>*
Data *<32 bits>*
Data *<32 bits>*
Data *<32 bits>*
Data *<32 bits>*
…….
…….
Data *<32 bits>*
Data *<32 bits>*

Figure 11: Transfer Packet

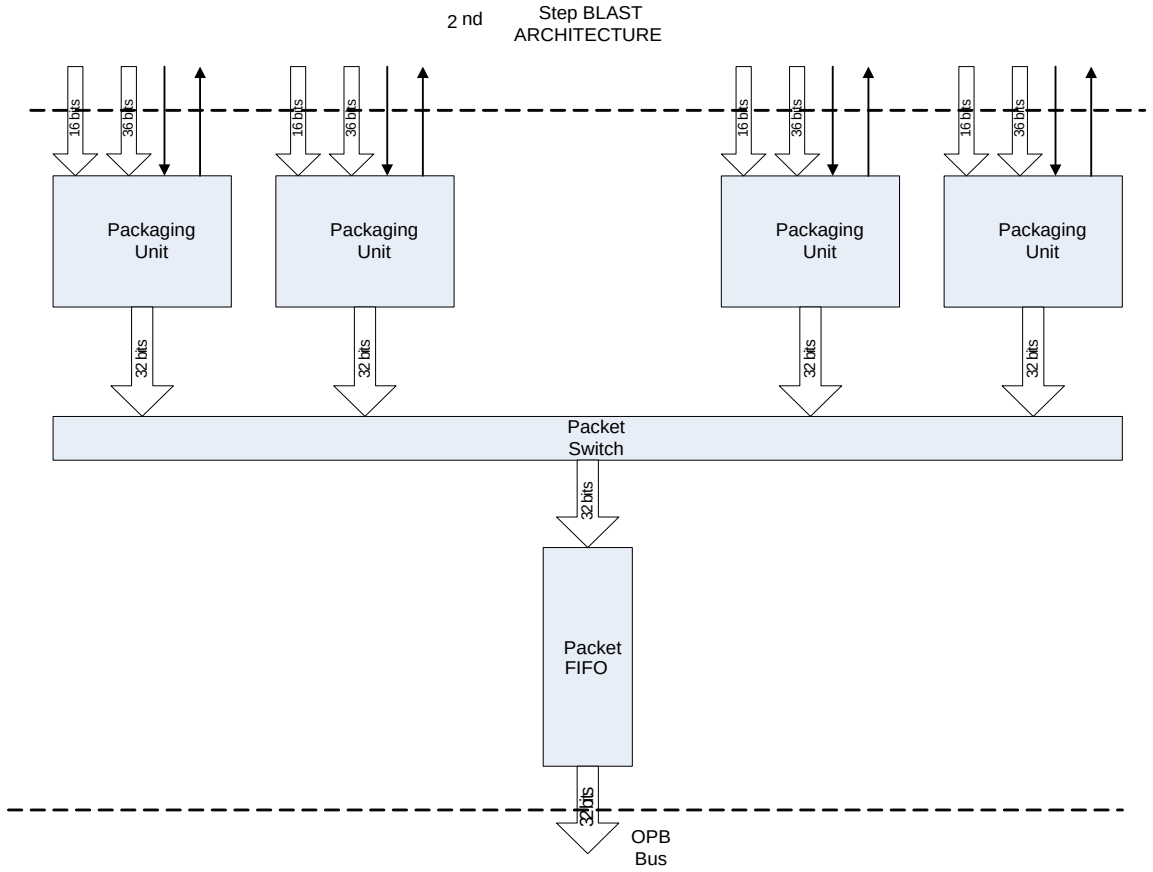

Figure 12: Hardware Part of Protocol Architecture

On the other end of the reconfigurable/fixed processor interface, the PLB bus of the Power PC runs the algorithm shown in Figure 12 to implement the software part of the communication protocol. The processor is activated when a "non-empty" signal is true on the FIFO. Then it reads the first two data words and gets the machine id, the length of the packet and the possible hit id. With this information it reads the data packets that follow, storing them in order in a 1000 letter long array.

Subsequently it calculates, according to possible hit information, the possible hit and determines if it is a HIT or not comparing with the W-mer list. If it is a HIT, then it executes the extension and if not it returns to the start to wait for a new possible hit.

5.3 3rd Step Software Architecture

The Extension Unit is activated when a possible HIT is confirmed as an actual HIT. The Score variable and indices are initialized to the proper values. The Score

variable is initialized to 60 because the W-mer is an exact match of 12 characters. The indices are for the query and for the history memory. The database stream is extended to both directions of the query during every iteration. If the query character matches with the corresponding character of the data stream, then the Score value increased by 5 and if there is a mismatch it is decreased by 4. Comparing 2 sets of characters during every iteration the Score value can increase by 10 for two matches, by 1 for a match and a mismatch, or it can decrease by 8 for two mismatches. If the score value decreases then the extension stops and produces the higher score value as an output.

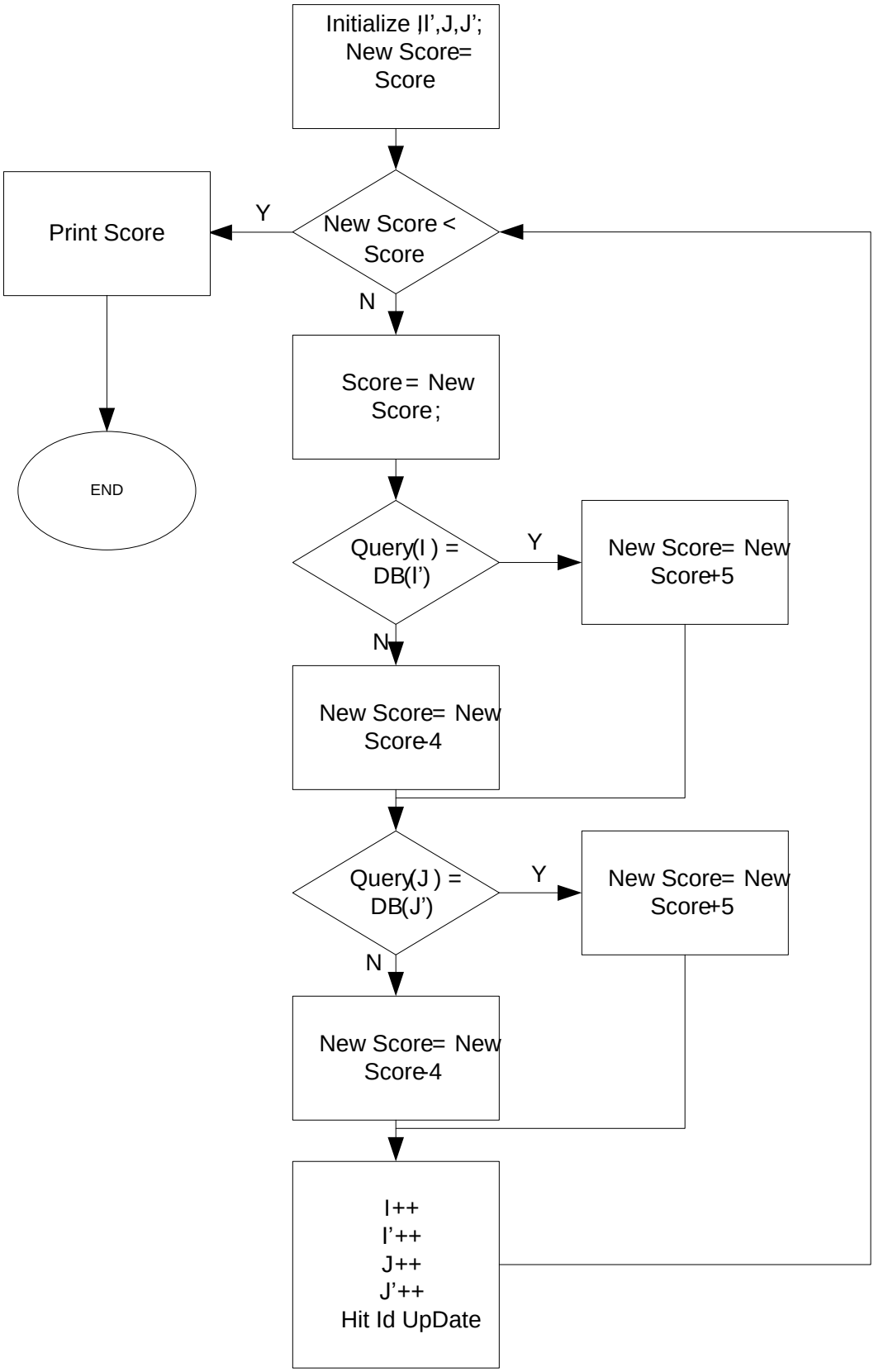

Figure 13: Extension step of BLAST Flowchart

Chapter 6

TUC BLAST Generic Architecture

The second generation of TUC BLAST architecture solved the most significant problems that the first generation had. This architecture was designed for the first variation of the algorithm BLASTn. In order to have a more versatile architecture that could solve the complete set of the BLAST algorithm variations, the architecture should become more generic. The third generation architecture that was developed in the context of this dissertation is the TUC BLAST generic architecture. This work was described at [84].

6.1 Analysis of the BLAST Algorithm Variations

The BLASTn algorithm, which was described in the previous chapters, is one of the five variations of the BLAST algorithm. The BLASTp variant which compares amino acids has a different alphabet of 20 letters instead of 4 and a different scoring scheme (during the third step of the algorithm) than BLASTn. The BLASTp scoring scheme uses matrices that can vary. The twenty letters require 5 bits lack in order to be represented vs. 2 bits that are needed for BLASTn alphabet. The BLASTx, TBLASTn, and TBLASTx algorithms use translated database, queries, or both for comparison. According to this translation each character is represented with 6 bits for the twenty letters alphabet that is used. With 20 amino acids we would expect to require 5 bits to represent them, but due to the translation process we actually need more bits. The translation process is the representation of each amino acid as 3 nucleotides (one triplet), each of which is 2 bits wide and therefore this representation is 6 bits wide. As the database is read and the string comparisons are made it can not be determined even at the bit level when a triplet starts. Therefore a six bit sliding frame scheme is used to examine the 6-bit encoded amino acid against sequences that may possibly start at any bit of the database.

The third step of BLASTx, TBLASTn, and TBLASTx follows the same rules as BLASTp. The typical size for W-mer is also different depending on BLAST version, BLASTn has 11 characters; BLASTp has 3 characters and BLASTx/TBLASTn/

TBLASTx has 6 characters each. Finally, BLASTx/TBLASTn/TBLASTx compares the six-frame conceptual translation products of a nucleic acid or protein sequence against a protein sequence or translation products. Table 2 shows all the differences that are of interest for a general system design.

	BLASTn	***BLASTp***	***BLASTx/TBLASTn/ TBLASTx***
Number of Alphabet letters	***4***	***20***	***20***
Bits Representing an Alphabet letter	***2***	***5***	***6***
3rd step scoring scheme	***+5 similar -4 non similar***	***PAM Matrices BLOSUM Matrices***	***PAM Matrices BLOSUM Matrices***
Size of W-mer	***11***	***3***	***6***
Comparison	***Every character***	***Every character***	***Six bits Frame***

Table 2: Parameter Values for Different Versions of BLAST

An additional extension to the second variant of the TUC architecture is the size of the query. Biologists report that a common length for a query is 1,000 to 10,000 characters but many implementations in software or in other groups examine queries up to 200,000. Whereas this number is probably not of great use for present-day biologists' needs, one cannot preclude such queries and therefore they need to be supported as well.

6.2 Datapath Variation

Considering all these differences and the need to calculate the algorithm for longer queries several changes have been made to the second step of the algorithm. Figure 14 shows the datapath of the previously implemented architecture for BLASTn Step 2. Figures 15, and 16 show respectively the datapaths of the new, general architecture for BLASTn, BLASTp, and BLASTx/TBLASTn/TBLASTx respectively.

The Data Input differs at each datapath depending on the bits that represent an alphabet letter. The BLASTn representation is 2 bits so the input to each processing

element is 2 bits. For BLASTp the representation is 5 bits and consequently the input data are 5 bits for each processing element. For BLASTx/TBLASTn/TBLASTx the representation is 6 bits but the input is 1 bit at a time, due to the six bit frame translation. Each bit is a part of a translated word without an exact bound to which a letter belongs. With that restriction a six bit frame is needed to examine one bit at a time and for that reason the input data is on bit. Despite the obvious datapath similarities in Figures 14, 15, and 16 one can see that we really have three distinct cases, complete with different input bus widths, comparators, and registers, arising from the parameter values shown in Table 2.

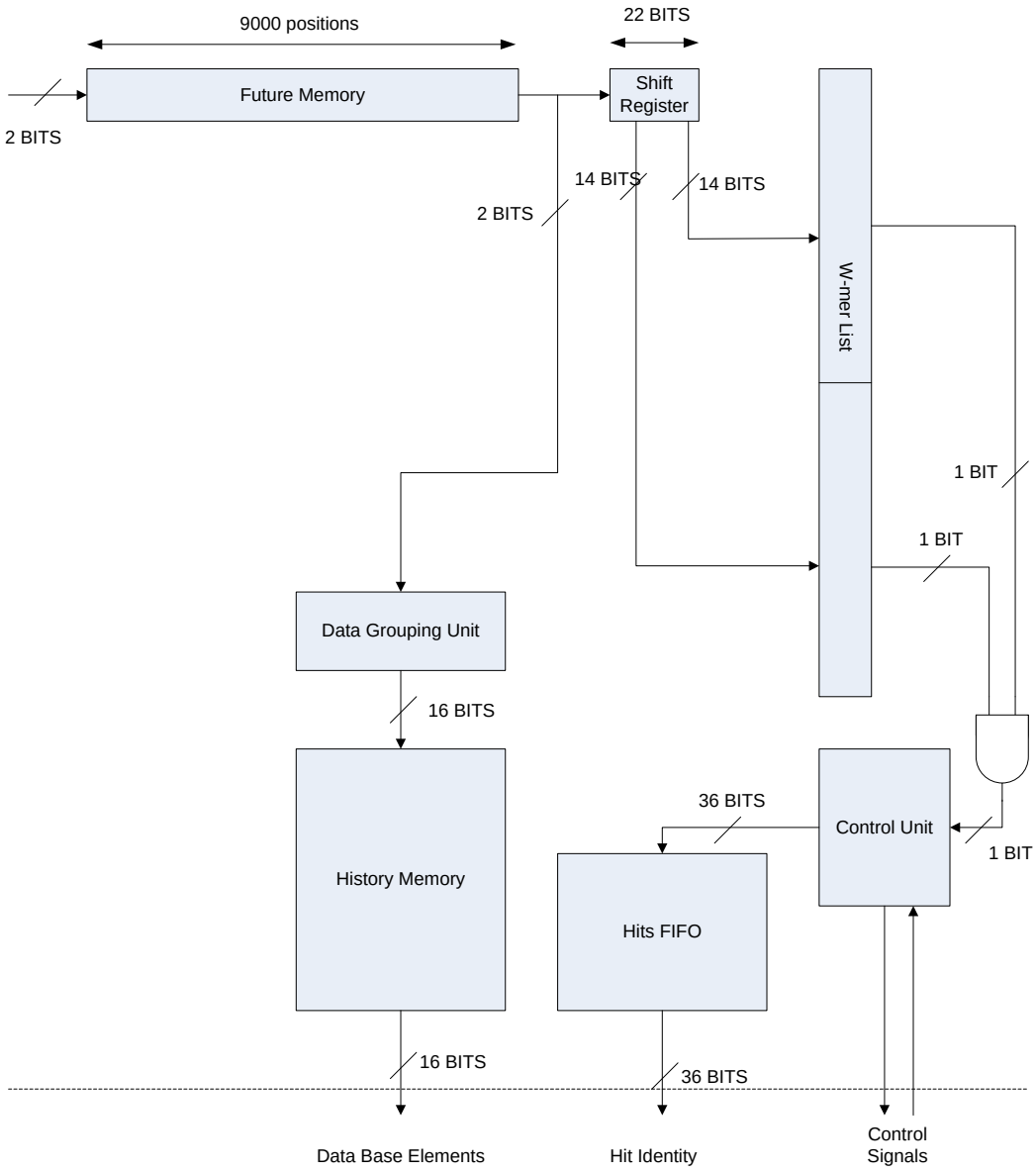

Figure 14: Second BLASTn Step Datapath (Same as the Second Variant of the TUC Architecture)

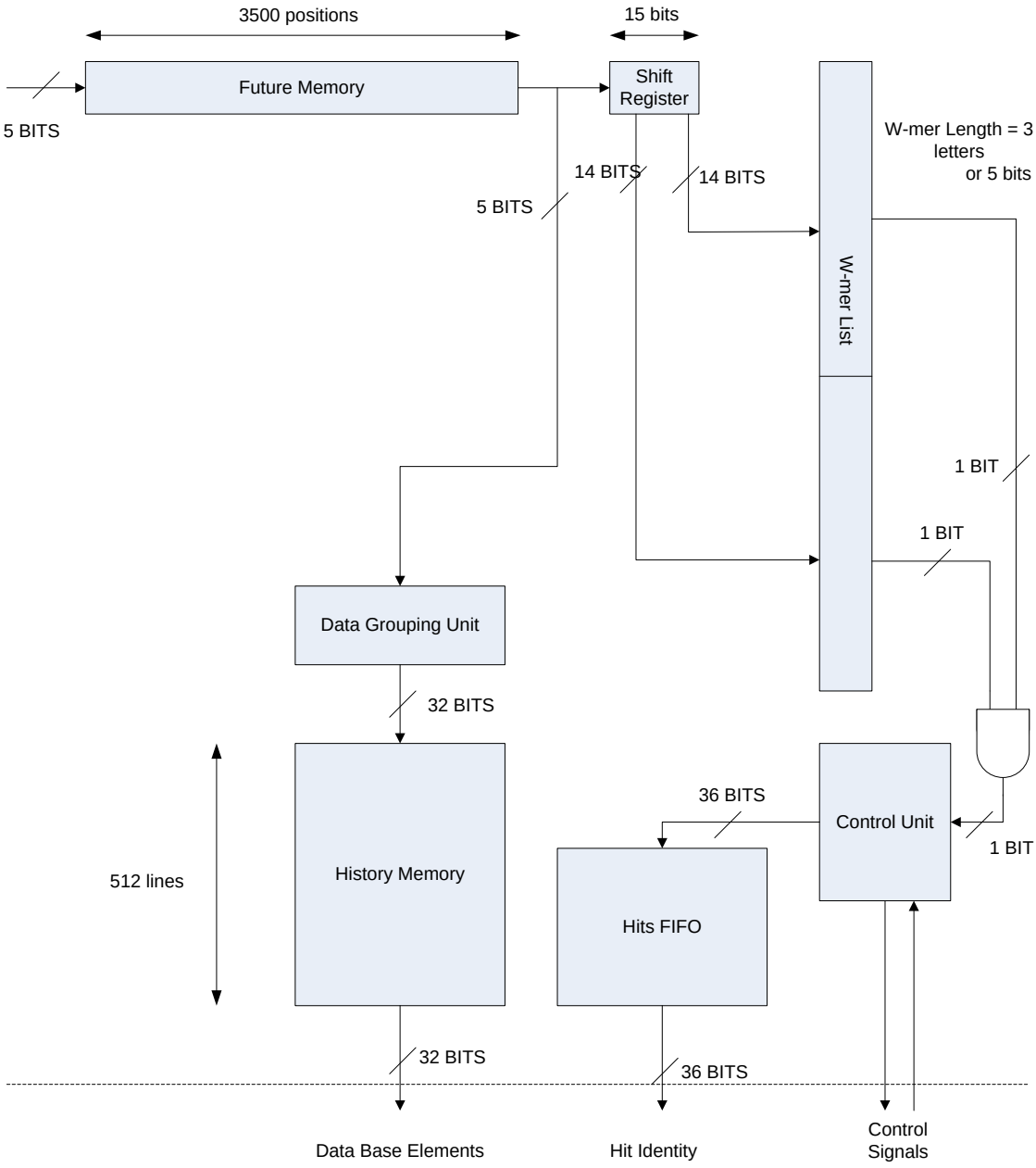

Figure 15: Second BLASTp Step Datapath (New Architecture)

The Future Memory entity is a shift register to buffer the part of the database that has not been examined yet for matches (but will be examined next) and is implemented using one Block RAM (BRAM) of the Xilinx FPGA. The BRAM structure is an on-chip static RAM that offers both high speed and a user-dependent datapath which can be quite wide or quite narrow, leading to a fast, customizable memory. This memory, however, is not as large as off-chip dynamic memory (DRAM), and therefore it is best used for local

data or as a buffer. Depending on the BLAST algorithm version the BRAM is 2, 5 or 1 bits wide to match the input stream. One BRAM can store up to 18Kbits which means 9000 characters for BLASTn, 3500 characters for BLASTp, and 3000 characters for the BLASTx/TBLASTn/TBLASTx version.

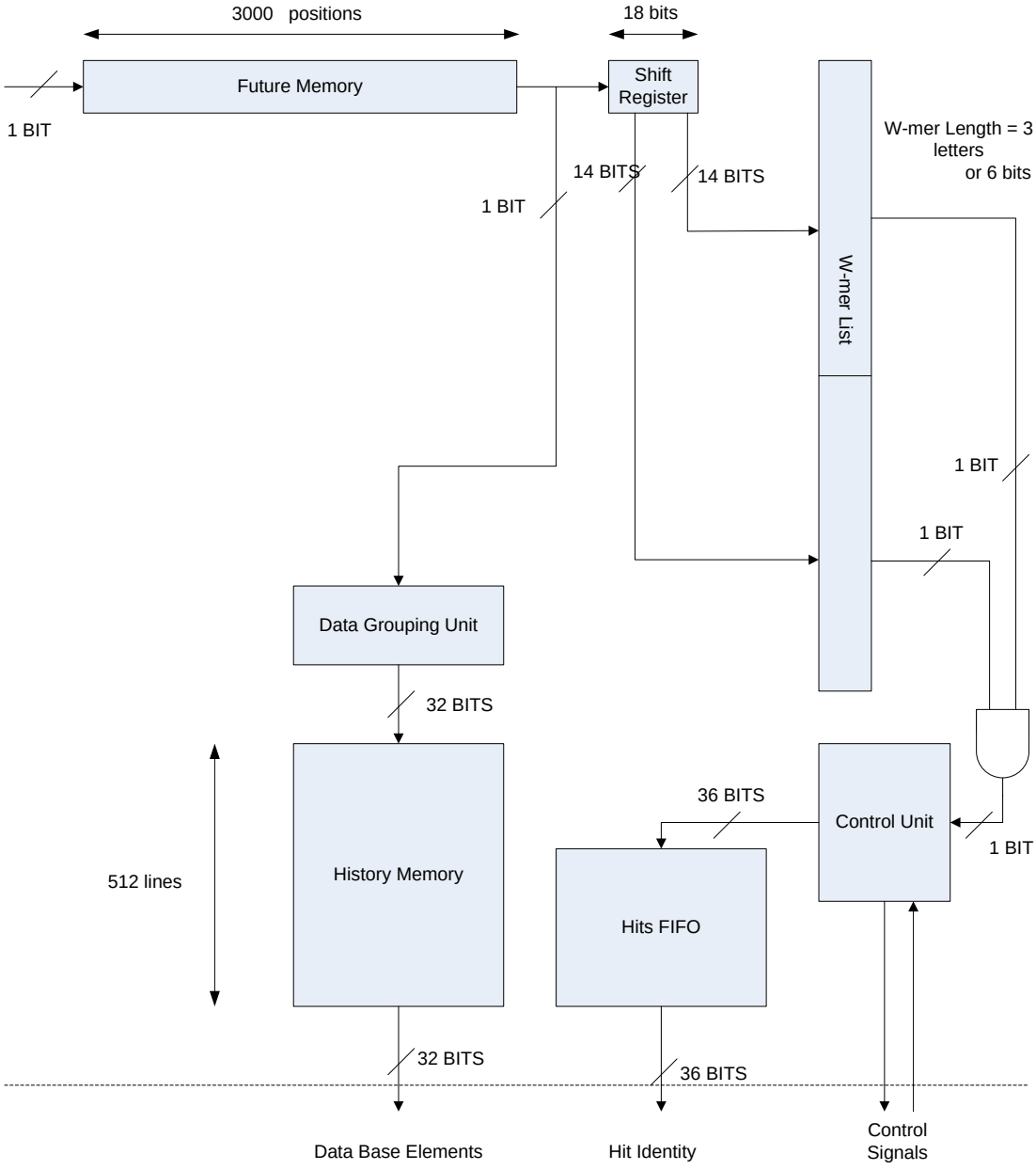

Figure 16: Second BLASTx / TBLASTn / TBLASTx Step Datapath (New Architecture)

The Shift Register that is driven from the Future Memory unit is 2, 5 or 1 bits wide respectively to match the input stream and has 22, 15, or 18 bits depending on W-mer size. The way the shift is done is one character for BLASTn and BLASTp, which

means 2 bits or 5 bits respectively, and one bit for BLASTx/TBLASTn/TBLASTx version.

The Data Grouping Unit concatenates local information to 32-bit words in order to pass those words to the PowerPC embedded processor via the OPB bus, i.e. the standard bus which facilitates communication between the reconfigurable processor and the fixed processor.

The busses which are driven from the Shift Register unit are always 14 bits wide at every version. This width has been chosen to drive one BRAM as one dual memory. For BLASTn it represents 14 bits of the 22 of the W-mer for BLASTp it represents 14 out of 15 bits and for BLASTx/TBLASTn/TBLASTx 14 it represents out of 18 bits of W-mer. Covering more bits of the W-mer would increase the probability of a possible hit to be an actual hit at the expense of less efficient usage of the BRAM. The increase of the probability that a possible hit is an actual hit, however, is not necessarily a goal, especially when it takes valuable resources that can be used for more parallel engines to perform the second step of the algorithm. The reason for this non-obvious conclusion is that if the PowerPC is underutilized, if there are more "false positives" which are weeded out in software the system throughput does not get worse, and if in fact the resources saved (at the expense of false positives during the second step) are used to increase parallelism in the second step of the algorithm, then the system-level throughput *increases*. The compromise of 14 bits was chosen after careful statistical analysis for several data sets, in order to match the reconfigurable processor speed to that of the fixed processor. It should be noted, however, that for different implementation technologies or different clock speeds of either the reconfigurable or the fixed processor, this number can vary. For the 14-bit wide memory the initial calculation of W-mers should be different depending on the version of BLAST algorithm, which means that with some data manipulation at the preprocessing level the general architecture keeps its basic structure without major changes. The remainder of the architecture for the second step remains the same as does the interface to the OPB bus. Whereas the datapath retains some similarities between BLAST versions, the corresponding control units are different, in order to account for W-mer size, alphabet size, and in the case of BLASTx/TBLASTn/TBLASTx for translations. The interface to the Power PC is identical for all versions of the

algorithm and the communication protocol remains also the same, but the PowerPC software to evaluate whether a possible hit is an actual hit and perform the extension (third step of the BLAST algorithm) is different for each version of BLAST.

The third step of the BLAST requires different scoring schemes depending on the version of the algorithm. In addition, different scoring schemes may be chosen by the system user, making this step ideally suited for software execution on the PowerPC. Implementing the third step of BLAST on the Power PC gives the user the flexibility to choose different scoring schemes at every time, as needed.

6.3 Query Variations

The query size in the first two variants of the TUC architecture (both for BLASTn) is 1000 and 5000 elements respectively. These sizes cover the typical size of a query but not every case. The published results of IBM and DELL report performance for BLAST with query sizes of 200,000 and 206,000 elements respectively. Whereas these sizes may have been chosen to optimize system performance, for an architecture to be truly general, arbitrarily high query sizes must be supported as well.

For the general BLAST architecture reported in this work each machine can cover query sizes up to 9,000 elements for BLASTn, 3,500 elements for BLASTp and 3,000 elements for BLASTx/TBLASTn/TBLASTx. Implementing larger query sizes in a single machine, whereas feasible, would place excessive BRAM demands on the design and thus reduce parallelism. The chosen solution was to perform data manipulation of the query as a means to extend the query size. In practical terms this allows for a direct space-time tradeoff and the ability to support arbitrarily large queries while maintaining a more-or-less constant system-level throughput. For example, a query size of BLASTn with size of 18,000 elements, can be calculated in two parallel machines, with the same data input and different initialization to each W-mer list. The first half of the query is calculated in one processing element and the other in the second processing element. Thus, the length of queries is not dependent on the W-mer list capacity but on the Future Memory size. For such an approach the software for the third step implementation on the PowerPC should keep more history data of fewer parallel elements.

For all versions of the BLAST algorithm and the corresponding architectures it is possible that we want either to examine the same query against different parts of the database or different queries for the same part of the database. Therefore there is a part of the datapath that is common to all variants of the TUC architecture, which can be configured accordingly. Figures 17 and 18 show the datapath for each of the two cases. In Figure 17 each of the comparison units for the second step of BLAST has been initialized to the same query. The BIT Selector unit passes different portions of the database or different databases to each processing element. Figure 18 shows how the same implementation is initialized with different queries at each processing element and these queries are examined in parallel against the same database.

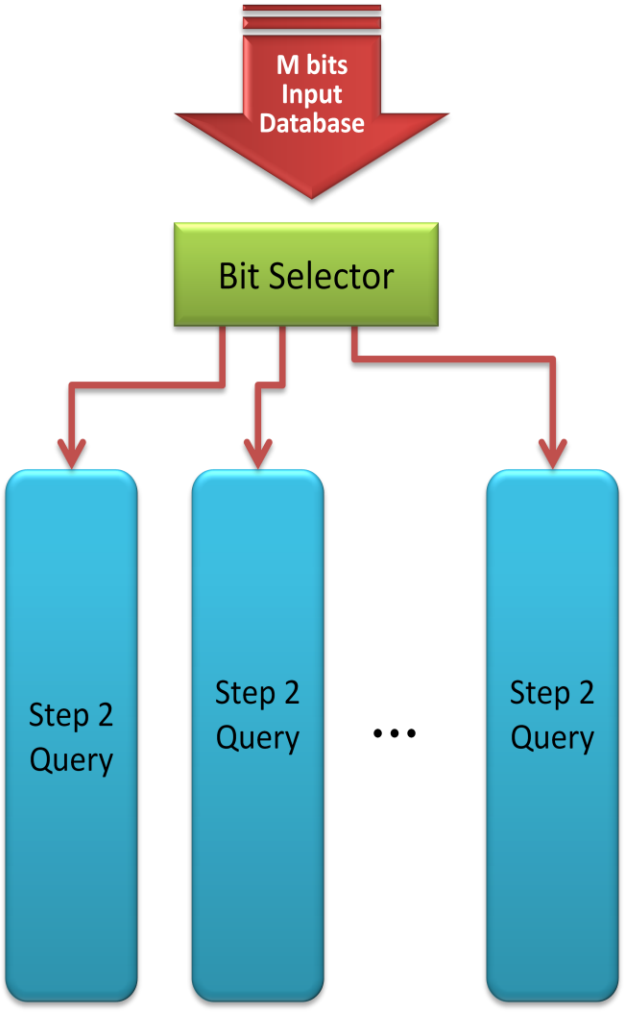

Figure 17: Datapath for One Query and Several Databases

6.4 Conclusions

This general architecture takes advantage of unexploited FPGA resources without using for each general processing element more critical resources such as BRAM. Each processing element spends 4 BRAMs *independently of the version of the algorithm that it implements*. It spends slightly more logic to implement the versions of the algorithm other than BLASTn but without significant resource spending and consequently without clock speed reduction. More detailed results are presented at Chapter 9.

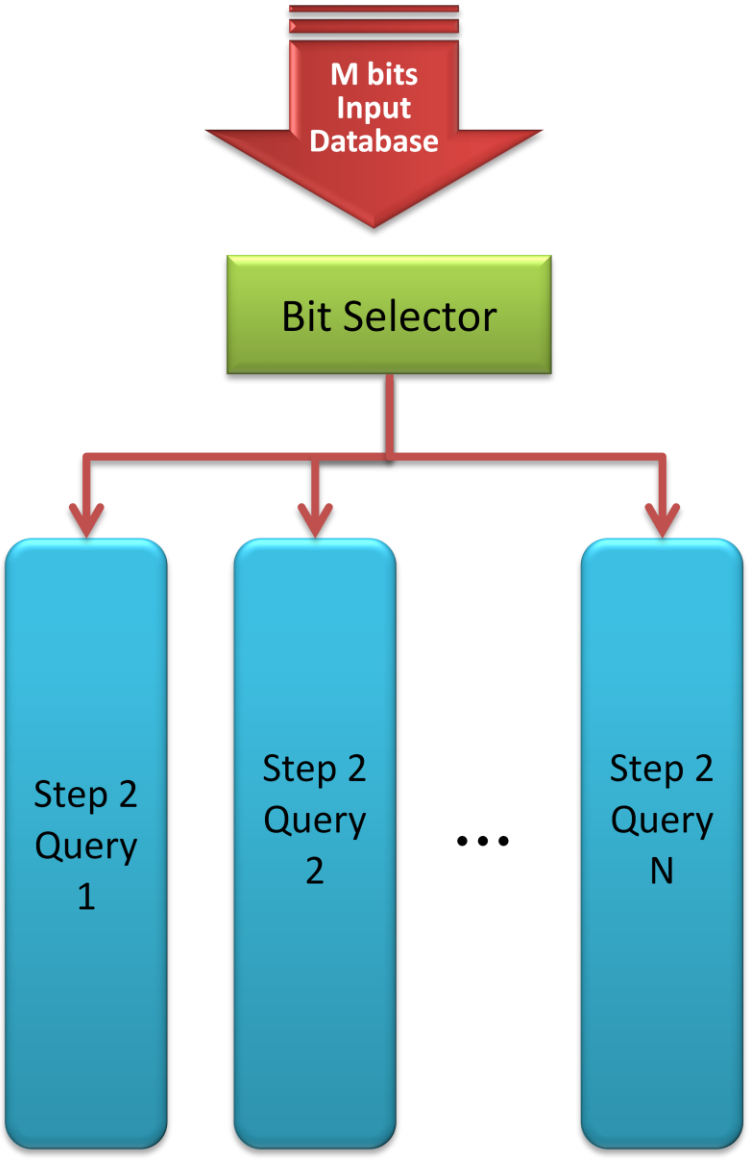

Figure 18: Datapath for Several Queries and a Common Database

Chapter 7

TUC BLAST Generic Architecture V.2

The iterative process of multiple BLAST implementations allowed for in-depth understanding of both algorithmic issues and technology mapping issues. Indeed, several significant changes took place at the system implementation level without changing the system architecture too significantly in order to produce yet a new architecture generation but in order to produce a new version which will be referred as: TUC BLAST Generic Architecture V.2. In this chapter we describe several system improvements at the implementation level.

Sections 7.1 and 7.2 describe architectural improvements that came out as each system was implemented and algorithm mapping to reconfigurable logic was better tuned. Sections 7.3, 7.4, and 7.5 describe improvements that came from algorithm study and better understanding. These three sections are the results of work that has been done with Prof. D. Pnevmaticatos, and fellow researchers Grigorios Crhysos, Panagiotis Afratis and Constantinos Galanakis, and it forms parts of P. Afratis' and C. Galanakis' diploma theses. Our contributions, highlighted here, address every stage of this work but they are mainly at the system architecture level and on how the stand alone system coupled with other processor. From this joint work three papers where published[80][85][86].

7.1 Processor and Switch Change

In order to have a design that can be implemented with any Xilinx device (including that of the XUP5V and the DRC platform) a decision was made to change the hard-core Power PC processor with the soft-core MicroBlaze processor. The Power PC is significantly more powerful than MicroBlaze but to date the maximum number of Power PC processors in a single chip is two. On the other hand an up to date large Xilinx device can have up to 80 [74] MicroBlaze processors running in parallel. In addition, the first

Virtex 5 Xilinx chips did not have any PowerPC processor initially and when the FX series (with PowerPC) was available the MHL has no access to such a device.

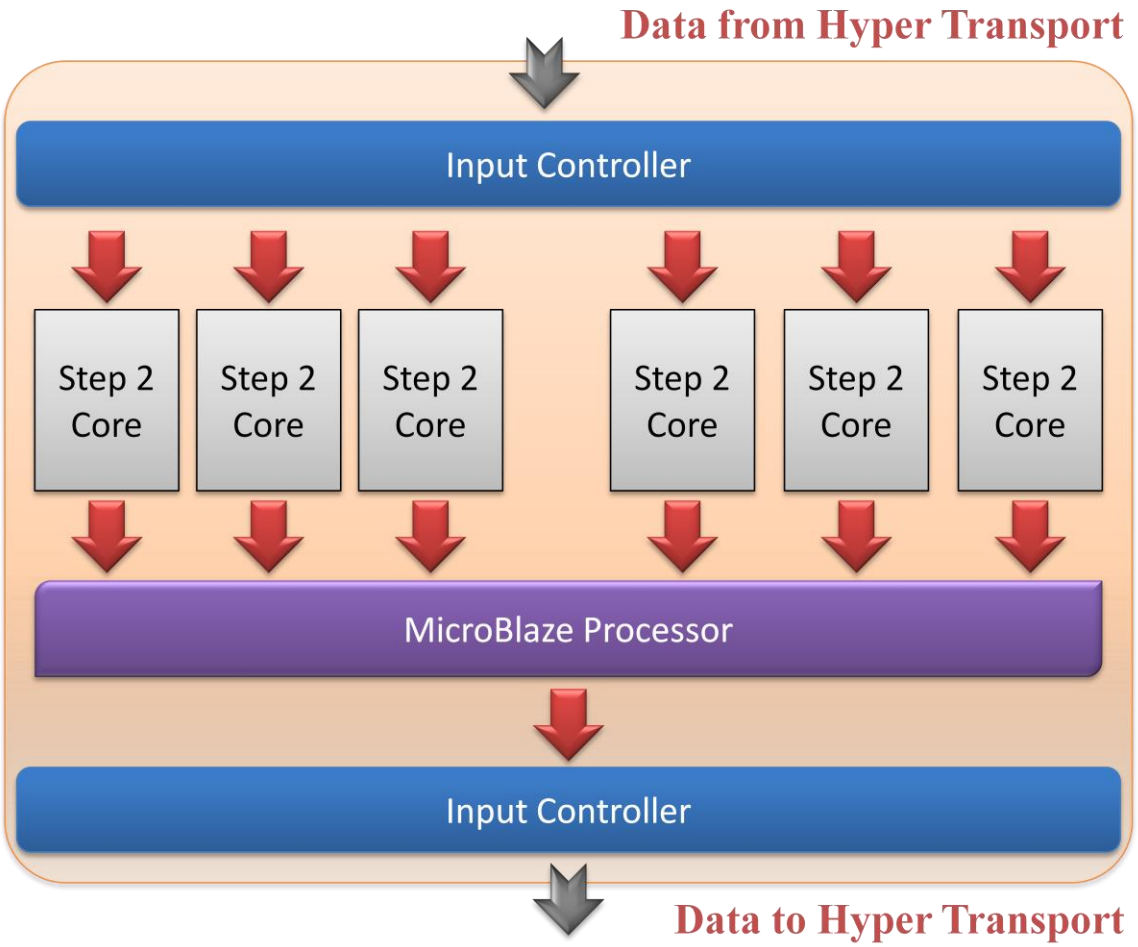

Figure 19: General scheme of MicroBlaze Architecture

The processor change was decided for reasons of design simplification and resulted in changes to the communication between the MicroBlaze processor and the reconfigurable logic design. In the place of the OPB bus, the FSL link which is available to the MicroBlaze processor was used for the communication. This change made the communication faster and simpler. The MicroBlaze can hold up to 8 FSL co-processors. Such a coprocessor can be a single BLAST Step 2 core or several connected with the proper glue logic. Use of FSL link eliminates the need of the switch between the several BLAST step 2 cores and the processor, which proved to be in the critical path, due to the significantly smaller number of these cores in every single processor. The general scheme

(with a single BLAST step 2 core at every FSL) of the architecture after these changes can be shown at Figure 19.

Machine ID *<8 bits>*
Machine ID *<6 bits>* **Packet Length** *<2 bits>*
Packet Length *<8 bits>*
Packet Length *<4 bits>* **Hit Id.** *<4 bits>*
Data *<8 bits>*
Data *<8 bits>*
Data *<8 bits>*
Data *<8 bits>*
…….
…….
Data *<8 bits>*
Data *<8 bits>*

Figure 20: New Communication Protocol over FSL

FSL is 8 bits wide while the OPB was 32 bits. This fact had a minor impact on the way that BLAST algorithm Step 2 core communicates with the processor. The protocol that was described at section 5.2.2 was changed slightly as we changed the width of the words but we kept the order of the bits, Figure 20 illustrates the protocol.

7.2 Future Memory Elimination

The critical resource of the designed architecture is the memory used for FIFOs implementation. As it is well known Xilinx devices have two kinds of memory; distributed memory which is mapped logic to be implemented and Block RAM (BRAM) memory which is mapped to embedded at the device, blocks of RAM. Only BRAM can be used for FIFOs implementation. Memory size is coarse grain with 36 Kb at every BRAM which is equal to 18,000 characters for BLASTn variation. In order to have fewer and consequently larger memory blocks the future memory of the datapath was merged to the FIFO called history memory. Some changes came to the control path of the design as when a HIT is detected Step 2 core does not immediately tries to send data to the MicroBlaze but waits for some cycles (according to query size) to collect the complete set of data that needs for the 3rd step of the algorithm. Figure 21 shows the new datapath for BLASTn variation including the FSL interface.

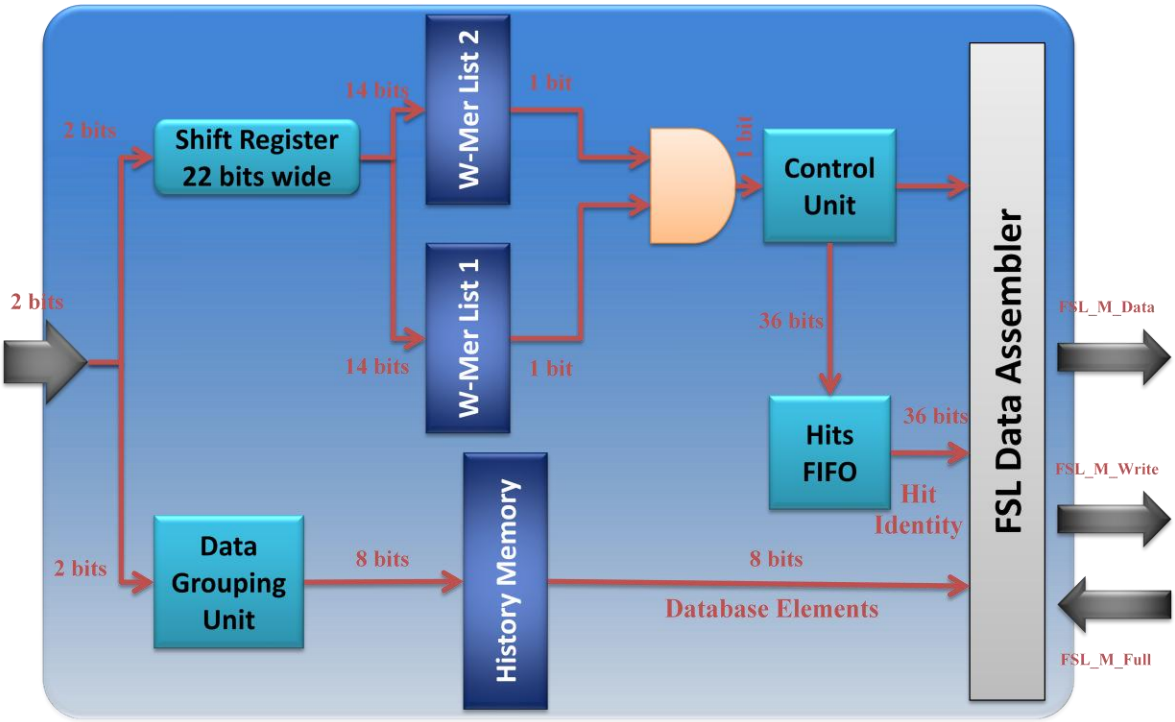

Figure 21: Simplified Datapath with FSL interface

7.3 BLAST Algorithm Further Analysis and Filtering Potential

To analyze the potential of BLAST algorithm, we built a set of software tools that implement BLAST searching. We ran these tools using several data sets that were provided from the NCBI site, and we compared the results against those of the original NCBI software. In our experiments we compared parts of Homo Sapiens (Human) (queries) against Chimpanzee's chromosomes (Pan Troglodytes) genome (database). The data exhibit a high degree of similarity which leads to high hit rate at the second step of the BLAST algorithm. We also used the BioPerf benchmark for BLASTn.

In first place we investigated the load balance between the step 2 and the step 3 of the algorithm at the implementation of the second architecture. In order to do that several software tools were developed which had identical results with the NCBI software and model the newly designed TUC architecture. Through this analysis a fundamental observation was made. Step 3 of the algorithm performs the extension when Step 2 of the algorithm produces a hit. Sections of the database that lead to good matches of the query have more than 2 hits in the same area, and in general they do have a lot of hits when they are examined. Thus, it would be interesting to see if the converse was also true. If

we have an area with a large number of hits does it mean that we have an area in which the BLAST algorithm will have many extensions? Data that have been extracted from the software tools were used and it has been proved that; **if we filter the database and keep the portions that produce many hits we have *all* the portions of the database that we actually need to apply the BLAST algorithm.** The eliminated portions of the database have no biological or other value. In order to state how to do this filtering we follow several steps.

Figure 22: Hit rate distribution for a window of 100 characters over the streaming database input, The two top circled areas are "of interest" i.e. they result in BLAST matches. The top horizontal line represents the optimal threshold (=5) to identify all these areas. Thresholds lees than 5 will produce more candidate regions without identifying more hits (drawn for Threshold=3), while thresholds greater than 5 will miss (some of) the hits reported by BLAST

7.3.1 Prefiltering Window Size

First, we investigated the effect of the window size, i.e. the width of the database region in which we measure the hit rate. Figure 23 plots "Space" (i.e. the resulting percentage of the database that we need to process after prefiltering) versus window size: small values are better since they correspond to smaller input to the full BLAST processing. Since the query size may vary greatly, we express the window size as a percentage of the query length, ranging from 10% up to 100%. Intuitively, larger window

sizes will produce more hits shifting the hit rate upwards. The results in Figure 23 lead to two conclusions. First, regarding window size, space is either unaffected or increases as the window size increases; hence a small window is both more effective and sufficient to capture the necessary information. Second, the effectiveness of pre-filtering varies greatly: we find cases where the results are excellent (the required space is in the range of 3% or less of the database), while totally ineffective in other cases (chromosomes 12 and 13) with space 100%, i.e. the entire database is candidate for match.

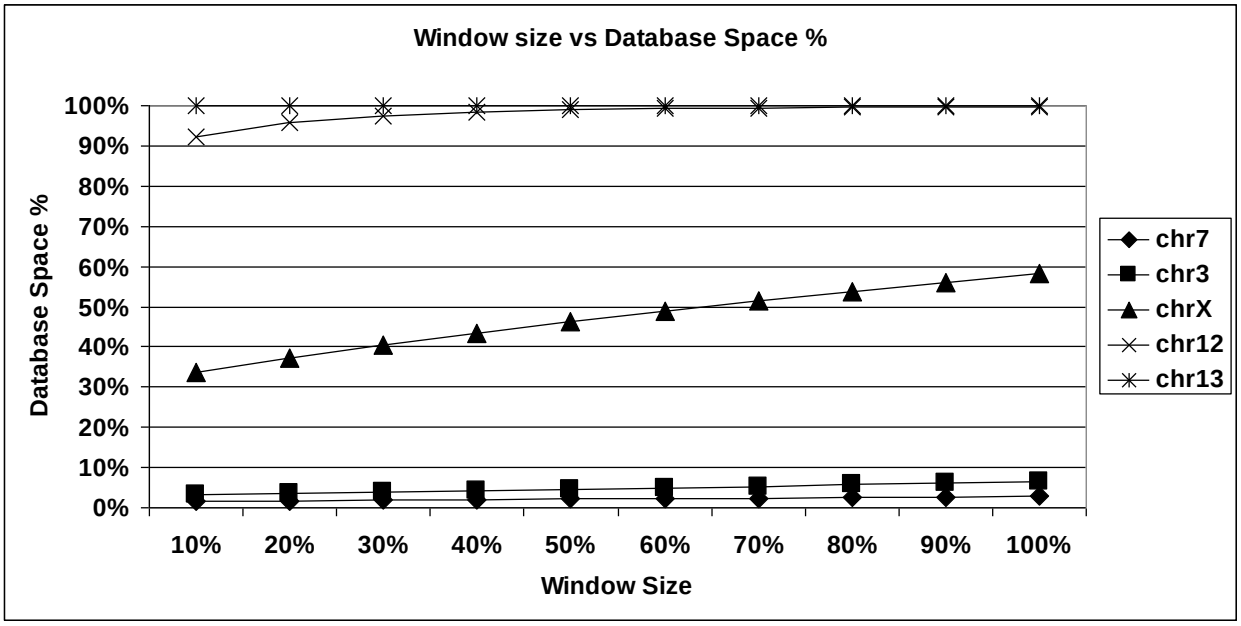

Figure 23: Database Space (%) vs. window Size.

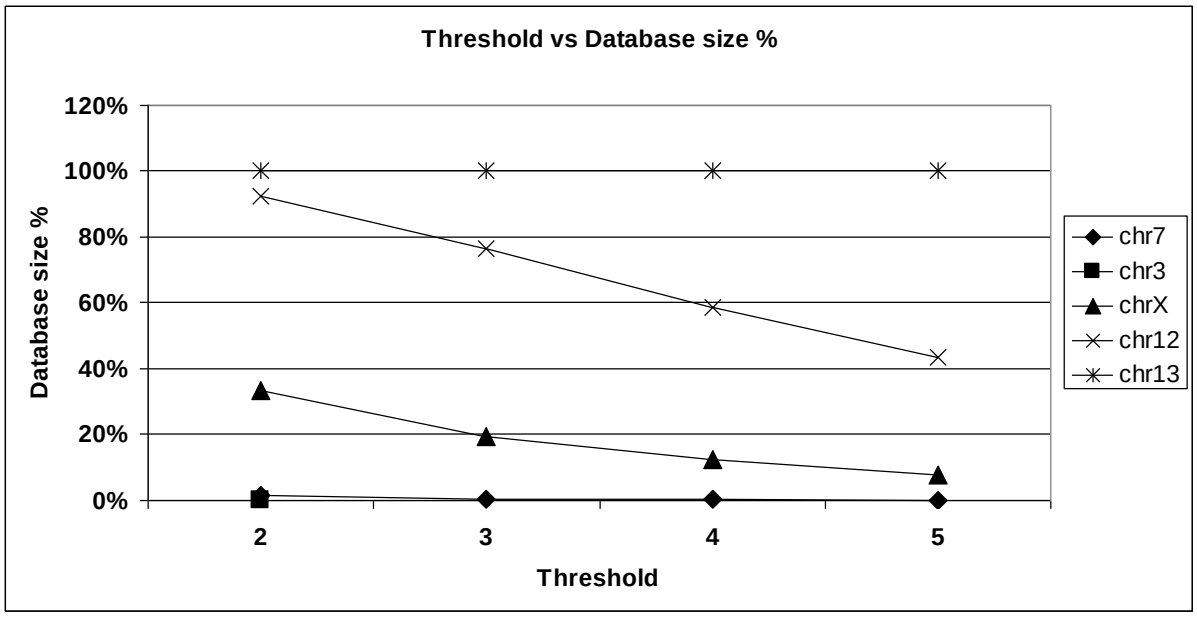

Figure 24: Database Space (%) vs. Threshold.

7.3.2 Filtering Threshold

The other main filtering parameter is the threshold. Figure 24 plots the database space versus a threshold that ranges between two and five. We see that as threshold increases there is a decrease in space, even for some of the "difficult" cases (chromosome 12) identified in the previous paragraph. However, the results for other queries, such as chromosome 13, are insensitive to increasing the threshold. Note that the choice of the threshold value is not straightforward. Setting the threshold too low results in a larger database space that needs to be processed. Setting the threshold too high we risk ignoring portions of the database that will produce actual hits. In the rest of this paper we use a threshold value of 2 based on the following observation: for the BLAST algorithm to begin the extension process we need at least one match. Since there will be at least one extension (otherwise the BLAST extension process stops), we will find another hit for a W-mer overlapping with the first. We tested all our results for all our runs and verified that indeed this threshold identifies all the reported NCBI BLAST results. To safely use larger threshold values we need to further understand the biological significance on the reported results. We believe that setting larger threshold values may omit only the least significant BLAST results while still report the high ranked ones.

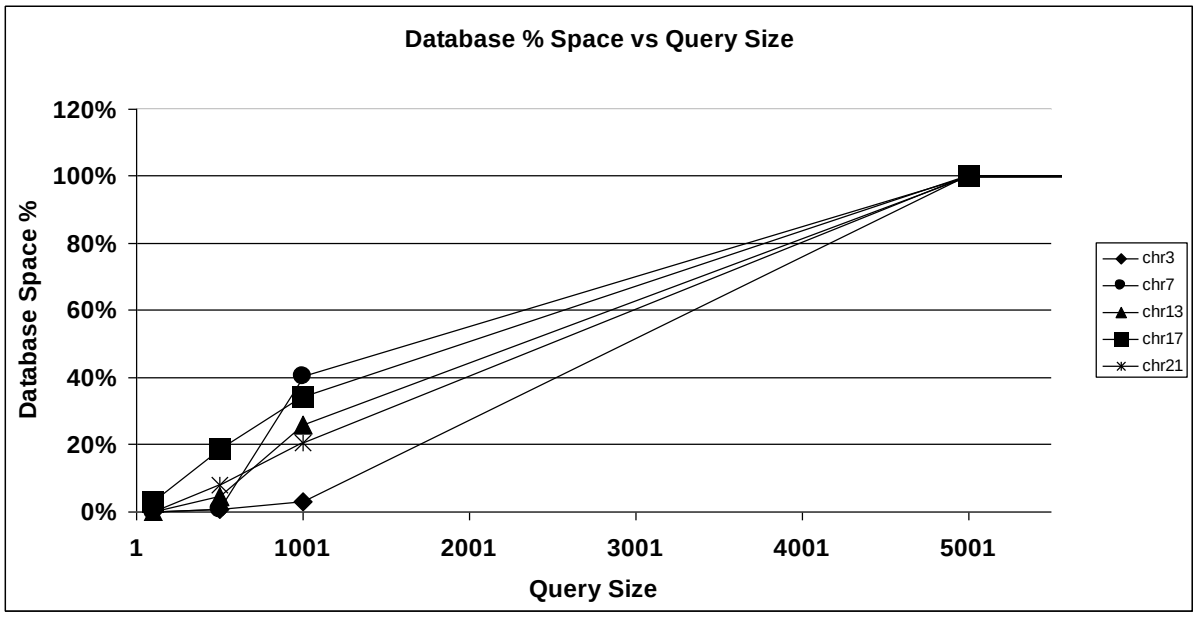

Figure 25: Database Space (%) vs. Query size.

7.3.3 Sensitivity on Query Size

To understand the behavior of the "difficult" cases such as of the chromosome 12 and 13 queries, we analyzed our results and observed that all these cases correspond to very long queries in the order of many thousand characters. In Figure 25 we plot the effect of the query size on the resulting database space that must be searched for the queries that are not amenable to prefiltering. To produce small queries we use a prefix of the original query at a particular size. The trend in Figure 25 is very clear: large queries are not amenable to prefiltering, while small queries show great potential. A possible explanation for this behaviour is that a large query contains more distinct W-mers than a smaller one, so the probability of finding multiple hits between the database and any W-mer is large. Prefiltering works very well for queries a few hundred characters long, and offers no improvement for queries longer than 5 thousand characters.

7.3.4 Partitioned Queries

The results from Figure 25 made clear that long queries, although very useful for biologists, cannot be handled effectively by prefiltering. However, the same results offer the solution to the problem: if the query is partitioned in smaller pieces and is processed

in parallel, we may achieve operation in the effective prefiltering region. Figure 26 evaluates the partitioning potential. Starting with the original query size, we subdivide it to pieces of one thousand, 500, 250 characters and so on, evaluating the resulting database space that we need to search. As indicated from Figure 25, as the query size becomes smaller, the effectiveness of prefiltering increases. The best results are achieved for small sub-queries less than 250 characters, and for all the difficult queries pre-filtering achieves a 5-fold decrease in the space that needs to be explored (space = 20% of the database). More important is the correlation of query and prefiltering potential: given the database and the query, we can determine the effectiveness of prefiltering, and the extent of required partitioning.

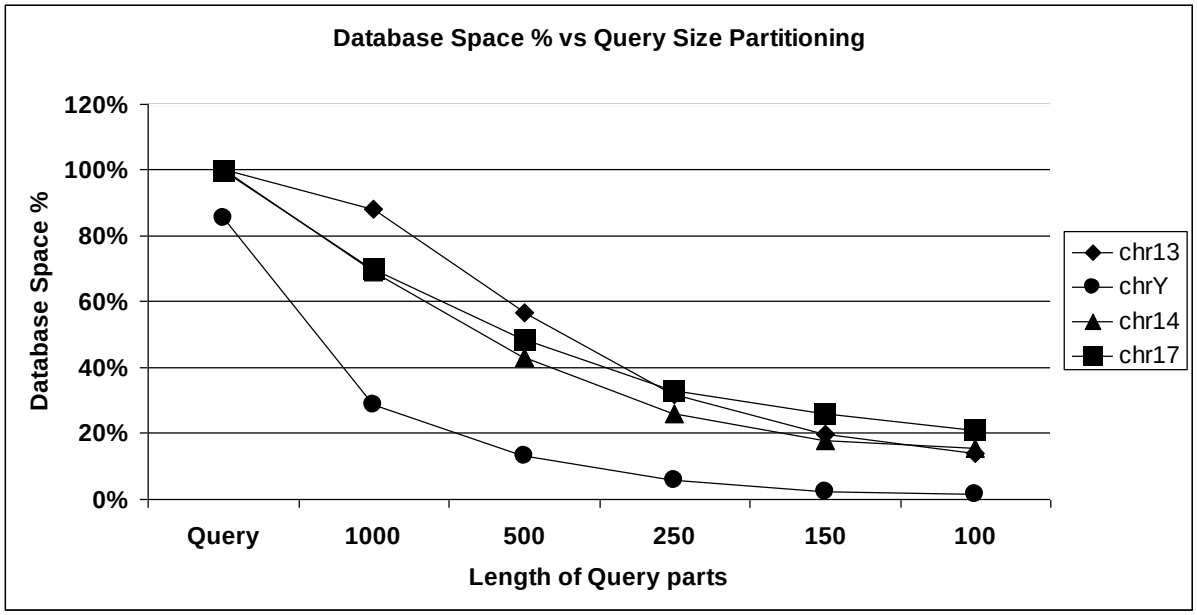

Figure 26: Query partitioning effect to Database Space

7.4 Bloom Filters

All the previously described properties are based on the number of the hits that are produced by the second step of the algorithm. In order to find hits, comparisons should be performed between every W-mer and the complete database. These comparisons are 26 bit-wide (12 characters x 2 bits/character) and their number is almost equal to the size of query: the number of W-mers is equal to (query length – W-mer length + 1). For a 1,000 character query 989 W-mers are produced and need to be

compared to the database input at every location. There are several implementations proposed for this problem.

A memory cache-like scheme was used at all TUC architectures. Using memories has the advantage that the size of the designed hardware is proportional to W-mer size which is constant and not to query size which varies. However, a single memory cannot be implemented due to its size (24 bits address) that can not fit to any reconfigurable device.

Due to hardware implementation limitations, an alternative method is proposed that uses a Bloom Filter [77] to determine the occurrence of a W-mer of a query. By the properties of Bloom filters, this approach can produce false positives, hence we count probable hits instead of actual hits. We attempted to identify the optimum number of the hash functions and the optimum depth of the filter memories while taking into account the implementation idiosyncrasies of the Xilinx FPGAs, and after thorough experimental research we concluded that 4 distinct prime polynomials used as hash functions to address 4 filter memories with address 14 bits was a good implementation trade-off. The 4 hash functions reduce the 24-bit wide W-mers to four 14-bit wide addresses, which are used to address 4 independent, 1-bit wide memories. These memories are initialized by setting to 1 all the locations identified by the hashing of all the query W-mers. On database lookup, a (probable) hit is determined when all the addressed memory locations are set to 1. Those matches are called probable hits because such a match might not necessarily produce a hit in the original second step of BLAST. However, in Bloom filters negatives are always true, so if a match does not exist, it can never be reported falsely. The structure of the Bloom filter is illustrated at Figure 27. Hence a simple lookup in the memory and an AND gate (not shown in Figure 27) identify if the W-mer portion is a sub-match with some W-mer of the query.

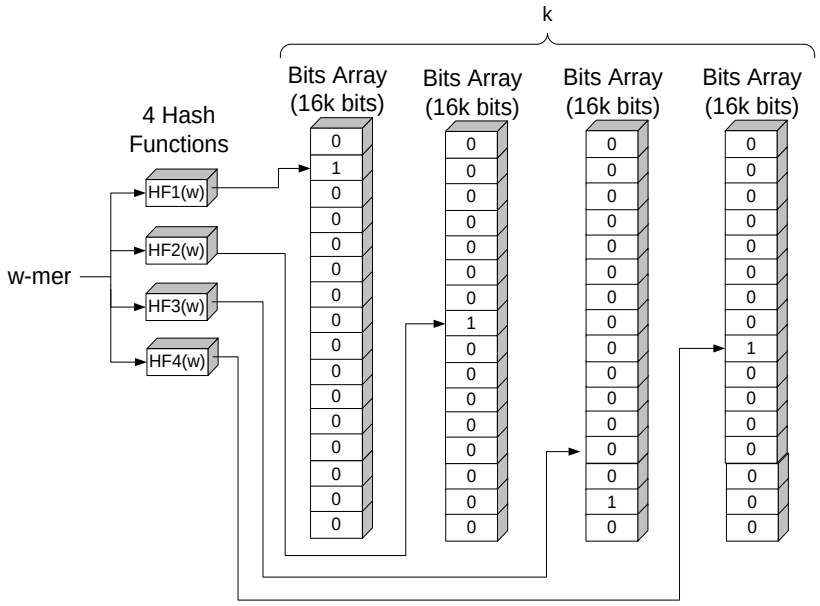

Figure 27: Example of BRAMs preloading

Since Virtex-5 BRAM blocks store 32kbits, are dual ported, and in their 32kx1 configuration need 15 bits for addressing, we combined two independent filter memories into a single BRAM block. In this way, the Bloom Filter can be efficiently implemented using just two embedded BRAM blocks.

7.5 BLAST Database Filter as an Autonomous System

7.5.1 PreBLAST Architecture

The implemented architecture, shown in Figure 28, takes as input the database stream. At every clock cycle a new character (2bits for BLASTn) is inserted in the shift register generating a new word of the database that has to be examined against all the W-mers. This 24-bit word is processed through the Bloom filters with the same hash functions that have been used to initialize the BRAMs four new 15-bit words are produced. The hashed values are used to address the four lookup tables and if all have a '1' stored at these positions, a probable hit is reported.

Figure 29 shows how the probable hits are counted for a certain window size. At every clock cycle the output of the probable hit is inserted in a shift register with length of window size. If a ‘1’ is inserted then the Up/Down counter counts up and when an ‘1’ is shifted out of the register the Up/Down counter counts down. With this simple design the Up/Down counter has always the number of the possible hits for the certain time window.

In this design there is also a position counter which counts the number of the characters that have been processed which is translated to the position of the database which is processed at the certain time. If the value of the Up/Down counter exceeds the predefined threshold then the position of database that this happens is stored in a memory. When the value of the Up/Down counter decreases under the threshold then the position of database is stored again. Consequently every pair of the stored values in the memory is the tagged part of the database.

In order to show the effectiveness of prefiltering we combined it with the Multiprocessor Platform for Embedded systems (M.PL.EM) [76]. We modified the way the MPLEM processor reads the database from its memory. Instead of initializing the memory controller and get the data in the row the memory controller starts from the points where the output memory of the PreBLAST filter shows, up to the points that the “interesting” parts of the database ends.

The MPLEM platform consists of MicroBlaze embedded processors which are not able to run the original NCBI software. For that reason a new software version of BLASTn algorithm was implemented and several experiments where tested. To verify the correctness, we run tests on a fully post-place and route simulation with up to 4 parallel MicroBlaze processors.

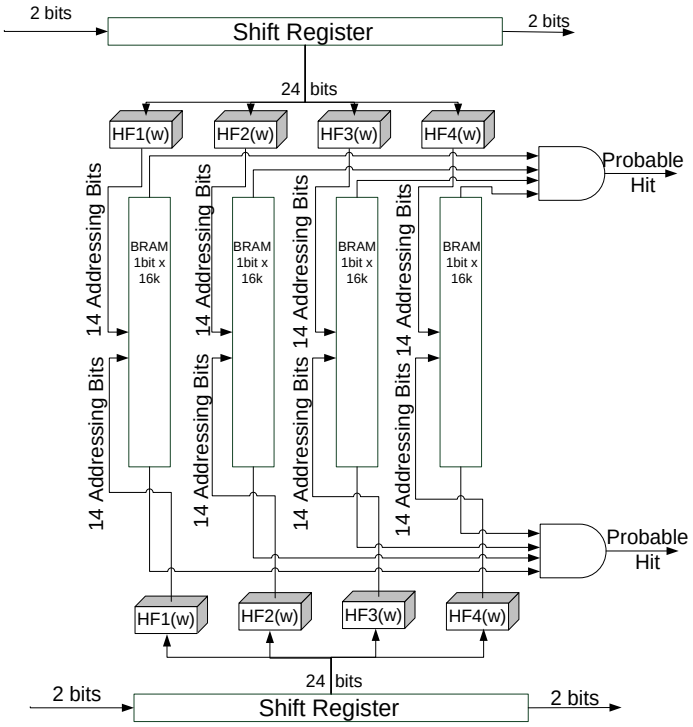

Figure 28: Data path of the designed system

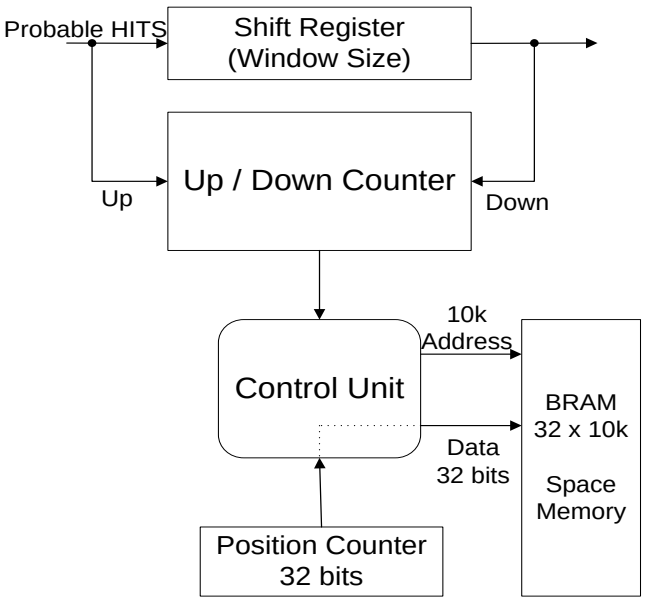

Figure 29: Control path of the designed system

Chapter 8

System Implementation

In this chapter all the systems that have been actually implemented on reconfigurable logic are described. Three platforms have been used to implement TUC BLAST. The XUP Virtex 2P platform was used initially in order to build the actual system for architecture verification. Then a DRC platform with Virtex 4 was used in cooperation with Synective Labs at Sweden but with no success due to problems during the integration phase. Finally a XUP Virtex 5 platform was used which offered enough resources including fast I/O for system building in small scale.

8.1 XUP Virtex 2P Platform

XUP Virtex 2P has a medium size device with XC30V2P with 136 embedded BRAMs and 2 PowerPC processors. The design that was implemented had finally four parallel Step 2 cores connected to 1 PowerPC. There was no input to the FPGA device from PC and RS 232 was used for output. The Xilinx tool Chipscope was used for internal signal checking and debugging through JTAG port. A controller connected with a preloaded ROM was used in order to give Inputs to the step 2 cores. Figure 30 shows the block diagram of the design.

Using this platform, which was available at the time, helped us to build and run several small- and medium-scale tests to debug the design up to the point that it was integrated and to evaluate the performance. The clock speed was 100 Mhz and some results from these implementations are presented in the next chapter.

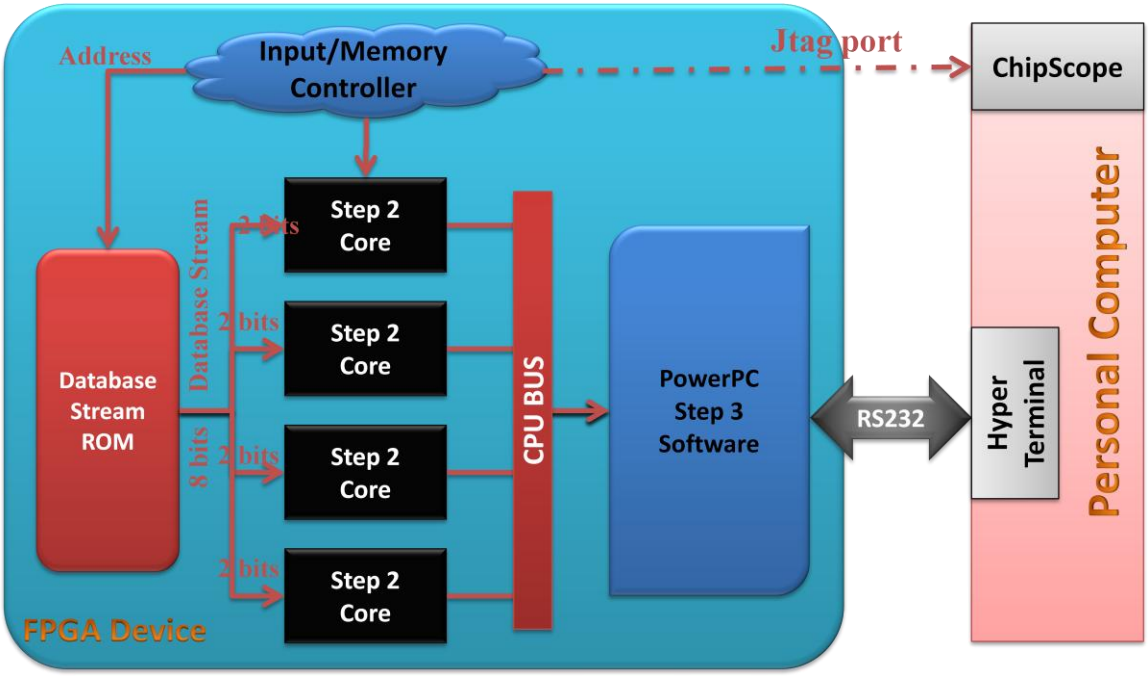

Figure 30: General Scheme of architecture of XUP V2P experiment

8.2 I/O Issues

The XUP Virtex 2P platform was proved to be very good for architecture verification, at least for runs with database up to 40,000 characters. In order to build a system that runs BLAST algorithm for any dataset with performance boosting I/O problem proved to be the bottleneck.

8.2.1 Locally Stored Database

Several solutions have been studied and efforts have been made in order to implement such solutions; one probable solution was the use of a large memory to store the database off-line. With a large database we could pass several queries and measure the overall throughput for several queries over one database. Such an approach bypasses the I/O problem as it adds an overhead time; time to load the database at the memory. The more queries that are loaded at the system the better performance the system has as the overhead gets smaller per query. These results are misleading as they are

i) depended on the data set size (number of queries and data base size)

ii) from biological point of view such a system is half useful. It works very well if we try to find several genes (queries) for example over one spices but it does not work well if we try to find one gene (query) over many spices.

iii) for the proposed TUC architecture such an approach eliminates its generality as this architecture works for every database size for any query and it works either for one database with many queries or for many different databases and for one query.

all these reasons make such an approach less attractive. Such an approach can be effective only if the available amount of memory is substantial percentage of the database size that the biologist is using. If for example Human DNA sequence is about four billion characters a memory of approximately one Gigabyte is needed for one spices database storing.

8.2.2 PCIe Interface

An alternative solution to the I/O problem is the use of PCIe interface. Several Xilinx platforms support such interface at the physical layer design and Xilinx CAD tools provide the higher level layers at the FPGA device. PCIe provides 2.5 Gbps baud rate and 2 Gbps actual data rate per lane. Such a platform is available at MHL. it is the XUP V5 platform which supports a single lane PCIe. A major problem using such a platform to connect to the PC which stores the database and the query is the driver that was needed for the operating system. Such a driver is available at MHL [87] for the Linux Operating System while Microsoft Research at Redmond provides a similar system [88] for Windows.

Both systems provide the driver and a hardware wrapper in order to help the designer to use it. Due to driver problems, both systems had a speed of about 250 Mbps. With such an I/O speed. I/O remains the system performance bottleneck and such a system cannot be competitive to a general purpose high end processor.

8.2.3 HyperTransport Protocol Interface

Several boards with FPGA device and high end I/O are available in the market. Such a board is the DRC platform which connects a general purpose high end processor through HyperTransport protocol with an FPGA. The connection has speed up to 9.6 GBps aggregate bandwidth. Such platform was available at the Synective Labs available with the appropriate software and hardware wrappers in order to encapsulate any design. In order to connect the BLAST designs a communication protocol was developed.

Inputs are the data bus which has width of 32 bits and four control signals Query Start which indicates that the data that are coming will be from query. Query Valid which indicates that the data are valid, and Data base Start and Data Base Valid which work the same way for the data base. When a hit has been found a packet of 8 32-bits words are send to the PC. Control signals are Dataout Start and Dataout Valid which work as the input control signals. The structure of the packet is shown at Figure 31. Due to integration difficulties it was not possible to have a complete system working properly.

<Header of Packet -- with packet id>
<Machine Number>
<Start point at database>
<End point at database>
<Start point at query>
<End point at query>
<score>
<End of Packet-- with packet id>

Figure 31: Result packet Structure

8.2.4 Gigabit Ethernet Interface

An option for fast serial interface between reconfigurable devices and a PC is the use of Ethernet connection. Main manufacturers (including Xilinx) have embedded in the devices tranceivers that can be used for many interfaces including Ethernet. On Xilinx's boards other chips have been added in order to implement the physical layer for such interfaces as the Ethernet and the Aurora. At TUC a complete software/hardware system

have been build called MTP[89] in order to give fast communication between the PC and an FPGA device.

This system is sending ordered UTP packets over IP over Ethernet with a minimum overhead of 5 bytes per packet. This overhead gives a theoretical upper limit of 956.32 Mbps of pure data. Due to limitations of processor and/or network device of the PC the actual speed that has been measured is up to 776.67 Mbps

The query and the database are stored in a standard format called FASTA. Figure 32 shows a part of such a file. In order to send from the PC to the device the dataset both files are modified in a single file. This file is binary and depending of the UTP packet length follows the MTP format for data packets. For example if the MTP packet size is 512 bytes the 5 first bytes are D0000 then it follows with 512 bytes of data and then the next 5 bytes are D0001 and it continues. At the data field the 4 first bytes of the first packet are the length of the query and then the query follows 4 characters per byte for BLASTn. When the query ends then the 4 first bytes of the data field are the length of the data base and then data follows encoded in the same way.

8.3 Universal Interface

Glue logic between Gigabit and wrapper of the interface (in the device) at the reconfigurable device has been build in order to integrate the Gigabit Ethernet Interface with the designed architecture. This logic consists of a FIFO (FIFO_1) and a controller, Gigabit Ethernet Interface receives data in another FIFO (FIFO_2) has a simple control scheme. Figure 32 shows the general scheme of the Glue Logic

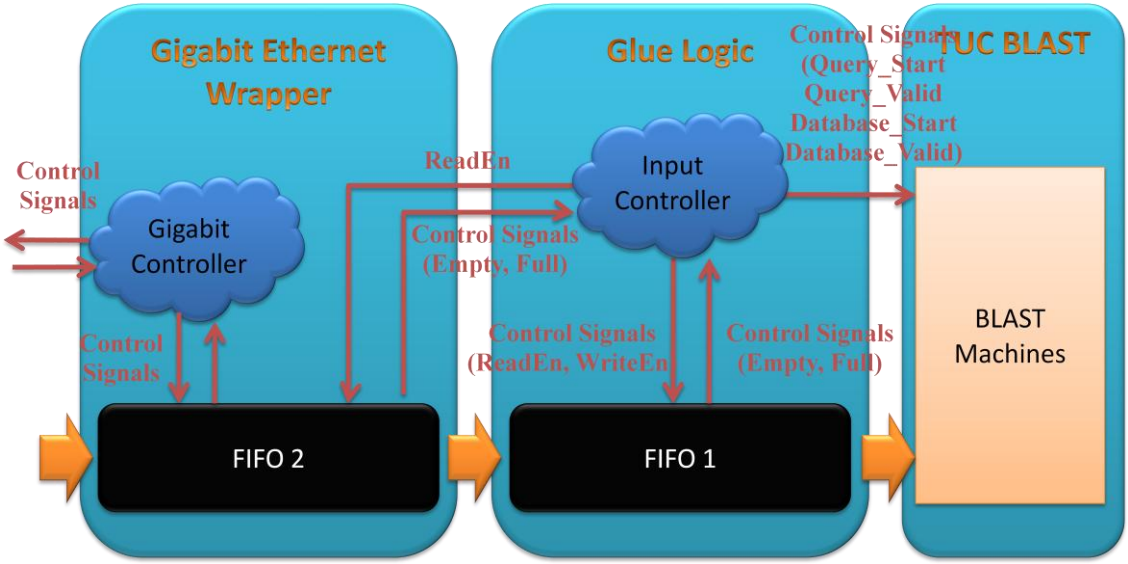

Figure 32: Glue Logic Architecture

FIFO_1 can be an elastic buffer that has different clocks for Input Data and Output Data but also different widths. For example FIFO_2 width can be 8 bits and FIFO_1 can have Input width of 8 bits and Output width of 32 bits. Widths of the FIFO_1 are depending on Width of Input data, rate of incoming data and the consuming rate of the data from BLAST machines.

Design of Glue logic is generic enough to be used with any kind of input wrapper (except Gigabit Ethernet Wrapper) that has an output FIFO with the standard control signals.

8.4 XUP 5V Platform

Digilent's XUP Virtex 5 Platform was available for large scale experiments. XUP 5V has a medium size device XC5V LX110T and is equipped with one lane PCI express Interface and a single Gigabit Ethernet interface. It has 296 BRAMs twice as much as the device of XUP V2P that was used for system building.

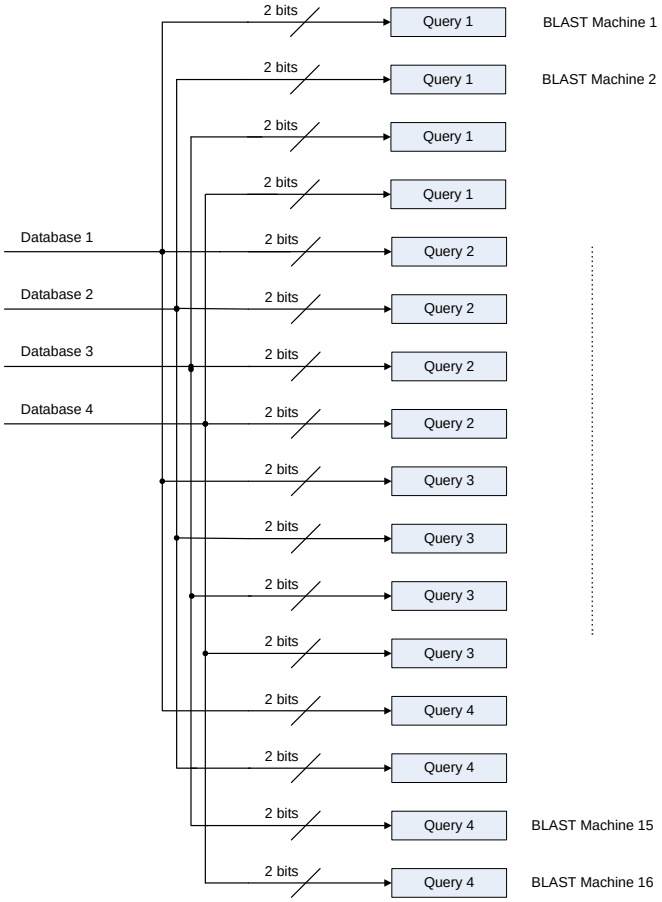

Figure 33: Databases and Queries set-up

In order to execute large scale experiments (e.g. database size of 1 GB) a fast serial interface had to be used. Such solutions were GigaBit Ethernet or PCIe interfaces. According to the environment (drivers, scripts etc) that have been developed to use these interfaces the data rates that have been measured were about 250 Mbps for PCIe interface and 850 Mbps for Gigabit Ethernet. Assuming a clock speed of 125MHz for our design (which is the speed of the Gigabit Ethernet Module) and the BLASTn variation of the algorithm implemented even four parallel BLAST machines need 1Gbps to be fed. Even

the faster Gigabit Ethernet interface will lead to under utilization of the four BLASTn machines for this specific platform. Four parallel BLAST machines would cover less than 25% of such a device. In order to exploit as much as it would be possible of the device resources it was decided to replicate BLAST machine design 16 times and have four different databases for four different queries. Figure 33 shows the set-up for databases and queries.

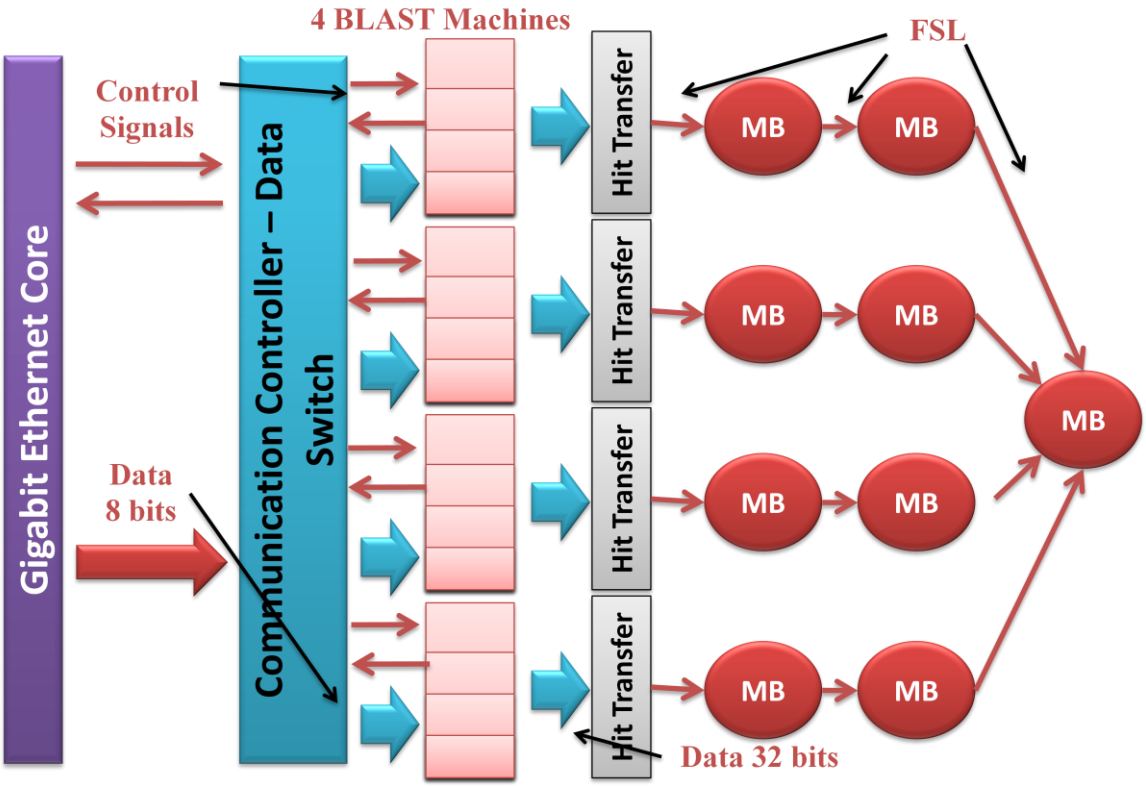

Figure 34: General Scheme of architecture of XUP V5 experiment

We decided that these 16 machines will be organized by four and every quartet to be attached to a Microblaze processor. Due to Microblaze low computing power a second Microblaze is attached in a series and the calculations are divided between these two processors. The outputs of the four quartets are attached to another Microblaze also serial and this one is responsible to give the final result. Figure 34 shows the general scheme of the prototype at XUP Virtex 5 Platform. It has been calculated that the same setting with quartets scheme could be replicated for 66 times to the largest up to date Xilinx FPGA device XC6VSX475T for four different databases and 17 different queries.

8.5 Technology Synopsis

Table 3 presents all the main characteristics of the architectures that were presented in previous sections.

Architecture name	Algorithm Variations	Query Size	Database size	I/O	Microprocessor	Critical Resources:	Technology
1st TUC Generation	BLASTn	up to 1000	any size	No I/O consideration	N/A	BRAM	Virtex 4
2nd TUC Generation	BLASTn	any size	any size	No I/O consideration	Power PC	BRAM. PowerPC	Virtex 4
TUC Generic Generation	BLASTn BLASTp BLASTx TBLASTn TBLASTx	any size	any size	No I/O consideration	Power PC	BRAM, PowerPC	Virtex 5
TUC Generic Generation V.2	BLASTn BLASTp BLASTx TBLASTn TBLASTx	any size	any size	Interface that supports PCIe, Hypertransport, GigE, Implementation with GigE	MicroBlaze	BRAM	Virtex 5

Table 3: Synopsis of Technical Characteristics for the Different TUC BLAST Architectures

Chapter 9

Implementation Issues

In this chapter all performance results that have been reported for software or hardware BLAST implementations in the literature are reported and compared against the performance of our architecture as were as our own software measurements. Due to the changes in technology as this project was implemented many measurements have been taken on several technologies, which were on high end devices at the time of the measurement.

9.1 SW Performance

Several results have been reported in the literature for software implementations of BLAST. IBM has reported several throughput measurements for its system IBM 375 MHz POWER3-II symmetric multiprocessor (SMP) and the 1.1 GHz POWER4 pSeries 690 Model 681[19]. IBM uses a set of several benchmarks depending on algorithm version, query size and database size. More specifically they use certain NCBI databases and a variety of queries that they report. Table 4 shows the set of benchmarks (NCBI files) that IBM used.

	Small Query	***Medium Query***	***Large Query***
BLASTn	ensembl.dna	ensembl.dna	ensembl.dna
BLASTp	drosoph.aa	drosoph.aa	drosoph.aa
BLASTx/TBLASTn/ TBLASTx	drosoph.aa	drosoph.aa	drosoph.aa

Table 4: IBM Single Chip Throughput – Testbench Cases

Tables 5 and 6 show the best throughput for several queries, depending on the BLAST version. Original times and database lengths that IBM provided refer to small queries, up to 2,000 characters, medium queries up to 50,000 characters and large queries up to 200,000 characters. For reasons of comparison Tables 5 and 6 have been formatted in this format.

Query Length (Characters)	*BLASTn Throughput (characters10^6/sec)*	*BLASTp Throughput (characters10^6/sec)*	*BLASTx/TBLASTn TBLASTx Throughput (characters10^6/sec)*
1,000	**187.62**	**15.50**	**4.77**
2,000	**187.62**	**15.50**	**4.77**
5,000	**14.23**	**1.975**	**0.042**
10,000	**14.23**	**1.975**	**0.042**
30,000	**14.23**	**1.975**	**0.042**
50,000	**4.16**	**0.35**	**0.05**
100,000	**4.16**	**0.35**	**0.05**
150,000	**4.16**	**0.35**	**0.05**
200,000	**4.16**	**0.35**	**0.05**

Table 5: IBM Single Chip Throughput – Performance Results

Query Length (Characters)	*BLASTn Throughput (characters 10^6/sec)*	*BLASTp Throughput (characters 10^6/sec)*	*BLASTx/TBLASTn TBLASTx Throughput (characters 10^6/sec)*
1,000	**1201.2**	**48.43**	**18.92**
2,000	**1201.2**	**48.43**	**18.92**
5,000	**159.36**	**9.90**	**1.374**
10,000	**159.36**	**9.90**	**1.374**
30,000	**159.36**	**9.90**	**1.374**
50,000	**53.14**	**1.49**	**0.189**
100,000	**53.14**	**1.49**	**0.189**
150,000	**53.14**	**1.49**	**0.189**
200,000	**53.14**	**1.49**	**0.189**

Table 6: IBM Multiprocessor System Throughput

Besides the IBM-reported results, we conducted our own experiments with several processors depending on the design generation.

9.1.1 TUC Measurements For the evaluation of the 1st generation

We performed runs of BLAST-2.2.12 on a 2GHz Xeon with 2GB main memory running SUSE 9.1 Linux and the CPU usage was profiled. Five NCBI data bases of several sizes for a small query of 1000 letters were executed at the 2GHz Xeon and measured. The same experiment was repeated with a Intel Pentium M 1.7 GHz with 1 GB main memory running Windows XP professional and an Intel P4 2.66 GHz with 1 GB main memory running Windows 2000. For Computers running Windows Intel VTune Performance Analyzer 7.2 was used and every measurement repeated 5 times. The results

of these experiments are respectively on Tables 7, 8 and 9. The averages in the tables are arithmetic averages.

DataBase name	*Database Size (characters)*	*Run Time (sec)*	*Throughput (char 10^6/sec)*
ecoli.nt	**4,662,239**	**0.024**	**194.25**
drosoph.nt	**122,655,632**	**0.482**	**258.33**
month.nt	**386,242,580**	**1.753**	**220.56**
env_nt	**1,061,221,997**	**1.190**	**891.63**
igSeqNt.ftptemp	**44.419,359**	**1.397**	**31.77**
Average	***323,840,361***	***0.968***	***319.25***

Table 7: Measurements on XEON 2 GHz / Linux

DataBase name	*Database Size (characters)*	*Run Time (sec)*	*Throughput (char 10^6/sec))*
ecoli.nt	**4,662,239**	**0.045**	**102.85**
drosoph.nt	**122,655,632**	**0.364**	**337.32**
month.nt	**386,242,580**	**1.303**	**296.50**
env_nt	**1,061,221,997**	**3.670**	**289.19**
igSeqNt.ftptemp	**44,419,359**	**0.174**	**255.43**
Average	***323,840,361***	***1.111***	***256.26***

Table 8: Measurements on Intel M 1.7 GHz / Windows XP

DataBase name	*Database Size (characters)*	*Run Time (sec)*	*Throughput (char 10^6/sec))*
ecoli.nt	**4,662,239**	**0.039**	**118.45**
drosoph.nt	**122,655,632**	**0.309**	**396.32**
month.nt	**386,242,580**	**1.022**	**378.10**
env_nt	**1,061,221,997**	**3.200**	**331.63**
igSeqNt.ftptemp	**44,419,359**	**0.160**	**277.40**
Average	***323,840,361***	***0.946***	***300.38***

Table 9: Measurements at Intel P4 2.66GHz / Windows 2000

9.1.2 TUC Measurements for the evaluation of the 2nd Generation General Architecture

Measurements were made at TUC with original NCBI software BLAST version 2.2.15 for win32-ia32at an Intel Pentium 4 3.0GHz HT, IA-32 architecture, 512 MB RAM with the Microsoft Windows XP Pro operating system. The Intel-developed software tool Vtune Performance Analyzer 8.0 was used for the measurements. The database was NCBI IgSeqNt. for nucleic Acid and BLASTn and igSegProt for proteins

and translated for the other versions of the algorithm. Table 10 shows that for several sizes of the query the respective throughput is depending on query size. The same datasets were used as test benches for the hardware performance evaluation.

Query Length (Characters)	*BLASTn Throughput (char 10^6/sec)*	*BLASTp Throughput (char 10^6/sec)*	*BLASTx/TBLASTn TBLASTx Throughput (char 10^6/sec)*
1,000	**588.75**	**19.80**	**20.97**
2,000	**409.15**	**12.72**	**13.12**
5,000	**218.26**	**5.10**	**4.97**
10,000	**129.32**	**3.02**	**2.75**
30,000	**41.26**	**1.03**	**1.03**
50,000	**36.13**	**0.59**	**0.55**
100,000	**18.73**	**0.26**	**0.21**
150,000	**12.91**	**0.15**	**0.11**
200,000	**10.13**	**0.09**	**0.07**

Table 10: TUC Software Measurements on Intel Pentium 4 @ 3.00 GHz

9.1.3 TUC Measurements for the evaluation of the TUC BLAST Generic Architecture V.2

Lastly, several measurements have been made in order to compare the MicroBlaze version of the hardware on a high end device with an equivalent processor in terms of technology. We used an Intel Core 2 Duo E8400 at 3 GHz with 2 GB RAM and Microsoft Windows XP Professional Version 2002 Service Pack 3. For the measurements we used the latest available version of Intel(R) VTune(TM) Performance Analyzer 9.0. The software version that was used was downloaded from NCBI and is NCBI-BLAST-2.2.19+-win32. The major change at this version is the separation of BLASTx/TBLASTn/TBLASTx. Instead of using the script BLASTall we used different executables that let the user test the different software performances of these variations. Table 11 shows the throughput values for several query sizes at any of the five BLAST variations.

Query Length (Characters)	*BLASTn Throughput (char 10^6/sec)*	*BLASTp Throughput (char 10^6/sec)*	*BLASTx Throughput (char 10^6/sec)*	*TBLASTn Throughput (char 10^6/sec)*	*TBLASTx Throughput (char 10^6/sec)*
1,000	**976.00**	**63.50**	**283.33**	**28.56**	**16.09**
2,000	**815.51**	**42.63**	**166.94**	**14.75**	**9.23**
5,000	**291.73**	**18.80**	**96.89**	**8.27**	**4.45**
10,000	**219.58**	**9.26**	**55.23**	**4.92**	**2.60**
30,000	**106.42**	**2.56**	**16.22**	**1.90**	**0.87**
50,000	**95.48**	**1.68**	**11.38**	**1.09**	**0.58**
100,000	**58.94**	**0.66**	**5.11**	**0.51**	**0.28**
150,000	**32.37**	**0.43**	**3.29**	**0.35**	**0.19**
200,000	**23.35**	**0.32**	**2.49**	**0.26**	**0.14**

Table 11: TUC Software Measurements on Intel Pentium Dual Core 2 @ 3.00 GHz

9.2 HW Performance

RC-BLAST was the first hardware implementation of BLAST but it will not be included in this comparison because the results were not competitive with software implementations even at the time when it was first reported. Table 8 shows the reported data that are available for Mercury BLAST and Treeblast. Mercury BLAST reports throughput of $96x10^6$ characters/sec for a single computing element, and 1400x106 characters/sec for a complete system implementing BLASTn. TreeBlast reports throughput $110x10^6$ characters/sec per processing element but no further data except for relative performance and its queries are small - up to 600 elements for BLASTp (hence the results are optimistic as the entire queries can be mapped in BRAM). For TBLASTn the FPGA/FLASH architecture reports to be two and a half times faster than Timelogic DeCypher with an equivalent performance of $0.0034x10^6$ characters/sec throughput for a query of size 132,000x103. Due to the enormous size of the query no comparison can be made.

	Mercury BLAST	*TreeBlast*
Query Length (Char.)	*BLASTn*	*BLASTp*
	-	
600	-	**110**
5000	**1400**	-

Table 12: Hardware System Reported Throughputs

9.3 TUC Performance

9.3.1 TUC 1st generation

The Technical University of Crete (TUC) 1st generation architecture was designed for BLASTn small query implementation (1000 letters) regardless of the data base size. It has been has been coded in VHDL and exhaustively post place-and-route simulated for the VIRTEX-4 family using the 4VFX140FF1517-11 device.

The first experiment was the measurement of a single machine (N=1) which run at 121,20 MHz and consumed less than 1% of logic recources and 8 BRAMs. More specifically every single machine needs 8 Blocks of BRAM, 5 of which are given to the memory of W-mer, 1 is used for query, 1 for History Memory and 1 for Future Memory. On the other hand it consumed 744 out of 126,336 LUTs. That shows that the critical resource for implementing many parallel machines is the BRAMs and this restricts parallelisation to 69 for the specific device (it has 552 BRAMs divided by 8 BRAMs for each machine). The next implementation was for 60 parallel computing machines (N=60) where exactly 480 BRAMs (or 86%) where used but only 36% of the available LUTs were used. In the last experiment the critical resource BRAMs were exhausted using 552 of them to create 69 parallel computing machines running at 100.36 MHz. As in the previous experiments the percentage of covering LUTs was low, only 42%.

At the experiments above it was assumed that there will be an input data stream of up to 69 characters, 2 bits each in parallel at a speed of 100.39 MHz. For that data stream a 138 bit wide bus is needed, with a speed of 13.86 Gbps.

Number of parallel Machines	***Number of FIFO16/RAMB16s (Total 552)***		***Number of 4 input LUTs (Total 126.336)***	
1	**8**	**1%**	**744**	**>1%**
60	**480**	**86%**	**46,522**	**36%**
69	**552**	**100%**	**53,836**	**42%**

Table 13: Area Demands of TUC Architecture

Number of parallel Machines	*Speed (MHz)*	*Width of Data Stream (characters)*	*Actual Throughput (char 10^6/sec)*
1	**121**	**1**	**121.20**
60	**103**	**60**	**6,192.58**
69	**100**	**69**	**6,924.84**

Table 14: Speed and throughput of TUC Architecture

9.3.2 TUC General architecture

The design and implementation of the TUC BLASTn architecture has been reported in chapter 4 and several throughput figures where projected according to query size, system clock speed, and implementation of parallel processing elements. All measurements are for Xilinx 4 FPGA 4VFX140FF1517-11 device. Due to the use of similar architecture with the same level of parallelism and I/O structure, and without spending more resources for BLASTp and BLASTx/TBLASTn/TBLASTx implementation we have assumed that the clock speed remains unchanged. The key elements of this assumption were verified with post place and route results from the Xilinx tools. The clock speed was calculated according to post place and route timing information of Xilinx software 7.1.03 which includes Device speed data version: "ADVANCED 1.54 2005-05-25" and is 103 MHz. Calculations of throughput are based on a 100MHz clock in all versions, which is a conservative estimate.

For the calculation of performance the first step of the algorithm has been ignored because it is performed off-line and only once for each query. The time needed for the first step is proportional to the query size and is the sum of the time that is needed to create he W-mer list from software plus the time that is needed to load this information to the TUC architecture. Every element of the W-mer list needs one cycle to be loaded and that means that is from 1,000 to 200,000 cycles depending on query size. Considering that the database size is several millions or billions of elements, which need several millions or billions of cycles to be processed, the time that is needed for the execution of the first step of the algorithm is negligible when compared to the total execution time.

Algorithm Step	*No of Units*	*Power PC/Unit*	*LUT/Unit*	*BRAM/Unit*
2nd Step Processing element	**128**	**-**	**720**	**4**
Communication Protocol	**2**	**-**	**1,174**	**9**
3rd Step	**2**	**1**	**-**	**-**
Total		**2**	**94,508**	**521**
Total FPGA Resources		**2**	**126,336**	**552**
Coverage Percentage		**100%**	**74.80%**	**94.38%**

Table 15: FPGA Resources Used in the BLASTn Implementation

Algorithm Step	*No of Units*	*Power PC/Unit*	*LUT/Unit*	*BRAM/Unit*
2nd Step Processing element	**128**	**-**	**750**	**4**
Communication Protocol	**2**	**-**	**1,174**	**9**
3rd Step	**2**	**1**	**-**	**-**
Total		**2**	**97,174**	**521**
Total FPGA Resources		**2**	**126,336**	**552**
Coverage Percentage		**100%**	**74.91%**	**94.38%**

Table 16: FPGA Resources Used in the BLASTp Implementation

Algorithm Step	*No of Units*	*Power PC/Unit*	*LUT/Unit*	*BRAM/Unit*
2nd Step Processing element	**128**	**-**	**780**	**4**
Communication Protocol	**2**	**-**	**1,174**	**9**
3rd Step	**2**	**1**	**-**	**-**
Total		**2**	**101,014**	**521**
Total FPGA Resources		**2**	**126,336**	**552**
Coverage Percentage		**100%**	**79.95%**	**94.38%**

Table 17: FPGA Resources Used in the BLASTx / TBLASTn / TBLASTx Implementation

Tables 15, 16 and 17 show resource allocation of a system implemented on a Xilinx 4 FPGA 4VFX140FF1517-11. The results in these tables show that the critical resources remain identical and highly utilized in all versions of the BLAST algorithm,

whereas the differences lie in data comparison units, control units, shift registers and some local busses. Therefore we can state that not only the architecture is general, but in terms of performance it strikes a good performance balance for all variants of the algorithm.

Query Length (Characters)	*BLASTn Throughput (char 10^6/sec)*	*BLASTp Throughput (char 10^6/sec)*	*BLASTx/TBLASTn TBLASTx Throughput (char 10^6/sec)*
1,000	**8,192**	**8,192**	**1,365**
2,000	**8,192**	**8,192**	**1,365**
5,000	**8,192**	**4,096**	**683**
10,000	**4,096**	**2,688**	**341**
30,000	**2,048**	**896**	**128**
50,000	**1,344**	**512**	**75**
100,000	**640**	**256**	**32**
150,000	**448**	**128**	**21**
200,000	**320**	**128**	**11**

Table 18: TUC General Architecture System Performance

The results in Table 18 show the projection of the TUC System for all major variants of BLAST and for a broad range of query sizes for any size of the database. Although these results are reported as projections, the fact that they come after detailed designs of the critical parts of the architecture and accurate timing simulations thereof means that they correspond to realistic performance that one can expect to achieve, even accounting for I/O issues.

9.3.3 TUC Generic V.2 Architecture

TUC Generic V.2 Architecture was implemented and post place and route simulated for a Virtex 5 XC5VLX330T. The measured period was 7.507ns or 133.20MHz. At this architecture there are 20 MicroBlaze Processor with 8 2nd step processing element each and 1 with 2 due to lack of memory blocks. Speed files that were used are ADVANCED 1.53, STEPPING level 0.

Table 19 show the resources spend for the implementation of BLASTn variation. It is shown that also at this implementation number of BRAMs is the critical resource for the design.

Table 20 shows the projected performance of the design for the several queries and all the algorithm variations. All the numbers have been calculated with clock frequency of 130MHz with the same method as in Table 18.

Algorithm Step	*No of Units*	*Virtex-5 Slices*	*36Kb BRAM*
2nd Step Processing element	**162**	**209**	**2**
Subtotal for 2nd step		**33,858**	**324**
3rd Step MicroBlaze	**21**	**283**	**-**
Subtotal for 3rd step		**5,943**	
Total		**39,801**	**324**
Total FPGA Resources		**51,840**	**324**
Coverage Percentage		**77%**	**100%**

Table 19: FPGA Resources Used in the TUC BLAST Generic Architecture V.2

Query Length (Characters)	*BLASTn Throughput (char 10^6/sec)*	*BLASTp Throughput (char 10^6/sec)*	*BLASTx/TBLASTn TBLASTx Throughput (char 10^6/sec)*
1,000	**21,060**	**21,060**	**3,510**
2,000	**21,060**	**21,060**	**3,510**
5,000	**21,060**	**21,060**	**3,510**
10,000	**21,060**	**10,530**	**1,755**
30,000	**10,530**	**4,212**	**585**
50,000	**5,265**	**2,340**	**351**
100,000	**3,009**	**1,239**	**176**
150,000	**2,106**	**842**	**117**
200,000	**1,620**	**619**	**88**

Table 20: TUC BLAST Generic Architecture V.2 System Performance

9.4 TUC Experimental Measurements

The experimental measurements that are presented were made at the Diglent XUP V5 platform with an XC5V LX110T. As mentioned above at 8.4, 16 parallel BLAST machines were build, grouped at four quartets. Each of the quartets was mapped to device resources as it is shown at Table 21.

Resource	*No of Units*	*Total*	*% Coverage*
Slices	***6,710***	***69,120***	***9.7***
DSP48Es	***7***	***64***	***10.9***
18Kb BRAM	***128***	***148***	***21.6***

Table 21: Device resource spend for each quartet

For the certain platform the clock speed is 125 MHz which is also the clock speed that is needed for the Gigabit Ethernet Interface. Using 1000 long queries the actual throughput of the system is the actual speed of Gigabit Ethernet interface which is about 500 Mbps to 800 Mbps or 250 Mchar to 400 Mchar for BLASTn variation of the algorithm that it was implemented.

Using four replicates of the BLAST machine quartet the processed queries are four for four different databases. While the actual throughput has not been changed the actual computational effort is four times larger. That is the way that DeCypher measure the computational power of their architecture covering the throughput bottleneck. In order to have a direct comparison with this way of measurement the experimental platform has a potential of 1000 KaaMnt/sec to 1600 KaaMnt/sec.

Table 22 shows five different databases and queries sets and actual execution times at a general purpose PC and the experimental platform. FPGA resources that are used for the specific experiment, are according to Table 21 are 20 per cent of the Virtex 5 XC5V LX110T device, which is about one third of the larger Virtex 5 XC5VLX330T. The equivalent of the reconfigurable resources that were used for the specific experiment are the 6.7% of a large high end Virtex 5 XC5VLX330T device.

Dataset	*Query Size*	*DataBase Size*	*Execution Time on Intel Pentium Dual Core 2 @ 3.00 GHz(sec)*		*Execution Time on XUP V5 @125Mhz*	
			Single Query	*4 Queries*	*(cycles)*	*(sec)*
Dataset1	***250***	***854,332***	***0.001***	***0.0035***	***318,706***	***0.0025***
Dataset2	***250***	***1,985,136***	***0.002***	***0.0080***	***743,201***	***0.0059***
Dataset3	***1,000***	***854,332***	***0.001***	***0.0035***	***449,641***	***0.0037***
Dataset4	***1,000***	***1,985,136***	***0.002***	***0.0080***	***1,046,833***	***0.0071***
Dataset5	***5,000***	***1,985,136***	***0.009***	***0.0825***	***1,364,512***	***0.0109***

Table 22: Execution times measured at experimental platform and corresponding run time on Intel Pentium Dual Core 2 @ 3.00 GHz

9.5 Comparisons and Speedups

9.5.1 TUC 1st generation

From Table 6 it can be shown that the fastest system throughput is achieved with the 16 processors Model 681 1.1 system, which has a throughput of 1,201.20 10^6 characters/sec. However the fastest single chip system is IBM Model 681 1.1 with 187.62 10^6 characters/sec. At these measurements I/O issues are not taking into consideration.

Table 23 has the actual throughput for systems implementing BLAST algorithm and in Table 24 the SpeedUp of TUC architecture against the other systems.

System	Actual Throughput (10^6 characters/sec)
2GHz Xeon	**319.25**
1.7 GHz Intel M	**256.26**
2.66 GHz Intel P4	**300.38**
TUC Architecture N=1	**121.20**
TUC Architecture N=60	**6,192.58**
TUC Architecture N=69	**6,924.84**
IBM single chip	**187.62**
IBM System	**1,201.20**

Table 23: 1st TUC Generation Throughput

	SpeedUp of TUC Architecture N=1	SpeedUp of TUC Architecture N=60	SpeedUp of TUC Architecture N=69
2GHz Xeon	**0.38**	**19.39**	**21.69**
1.7 GHz Intel M	**0.47**	**24.16**	**27.02**
2.66 GHz Intel P4	**0.40**	**20.61**	**23.05**
IBM single chip	**0.65**	**33.00**	**36.90**
IBM System (16 chips)	**0.10**	**5.15**	**5.76**

Table 24: 1st TUC Generation SpeedUp

9.5.2 TUC General Architecture

We have compared the TUC performance projections with measurements that have been reported from IBM or measured at TUC with software running on a Intel Pentium 4 and the results are shown in Tables 25, 26, and 27. These figures show that the general TUC architecture can be faster by tens up to more than one thousand times vs. a single chip such as a recent generation Pentium 4, or even hundreds of times faster than the IBM system with 16 1.1 GHz POWER4 pSeries 690 Model 681 processors. The speedup figure shows how many times one system is faster than another, e.g. for a query of 10000 characters, the TUC architecture running BLASTn is close to 32 times faster than a 3GHz Pentium 4 running the same code in software. Such results are most encouraging for hardware-accelerated BLAST execution.

Query Length (Characters)	TUC vs. IBM Single Chip	TUC vs. IBM System	TUC vs. Pentium 4 @ 3GHz
1,000	43.66	6.82	13.91
2,000	43.66	6.82	20.02
5,000	575.69	51.41	37.53
10,000	287.84	25.70	31.67
30,000	143.92	12.85	49.64
50,000	323.08	25.29	37.20
100,000	153.85	12.04	34.17
150,000	107.69	8.43	34.70
200,000	76.92	6.02	32.59

Table 25: Speedup of TUC for BLASTn

Query Length (Characters)	TUC vs. IBM Single Chip	TUC vs. IBM System	TUC vs. Pentium 4 @ 3GHz
1,000	528.52	169.15	413.74
2,000	528.52	169.15	644.03
5,000	2,073.92	413.74	803.14
10,000	1,361.01	271.52	890.07
30,000	453.67	90.51	869.90
50,000	1,462.86	343.62	867.80
100,000	731.43	171.81	984.62
150,000	365.71	85.91	853.33
200,000	365.71	85.91	1,422.22

Table 26: Speed up of TUC for BLASTp

Query Length (Characters)	TUC vs. IBM Single Chip	TUC vs. IBM System	TUC vs. Pentium 4 @ 3GHz
1,000	286.16	72.15	65.09
2,000	286.16	72.15	104.04
5,000	16,261.90	497.09	137.42
10,000	8,119.05	248.18	124.00
30,000	3,047.62	93.16	124.27
50,000	1,500.00	396.83	136.36
100,000	640.00	169.31	152.38
150,000	420.00	111.11	190.91
200,000	220.00	58.20	157.14

Table 27: Speed up of TUC for BLASTx / TBLASTn / TBLASTx

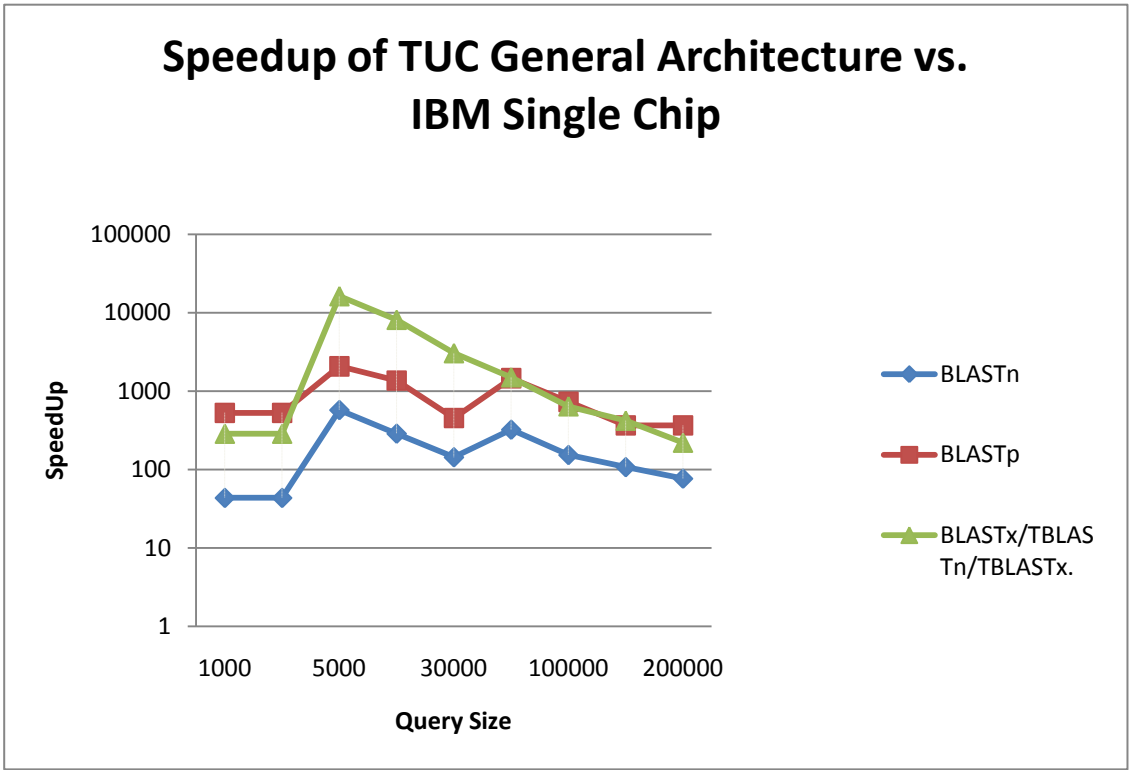

Figure 35: Speedup of TUC General Architecture vs. IBM Single Chip

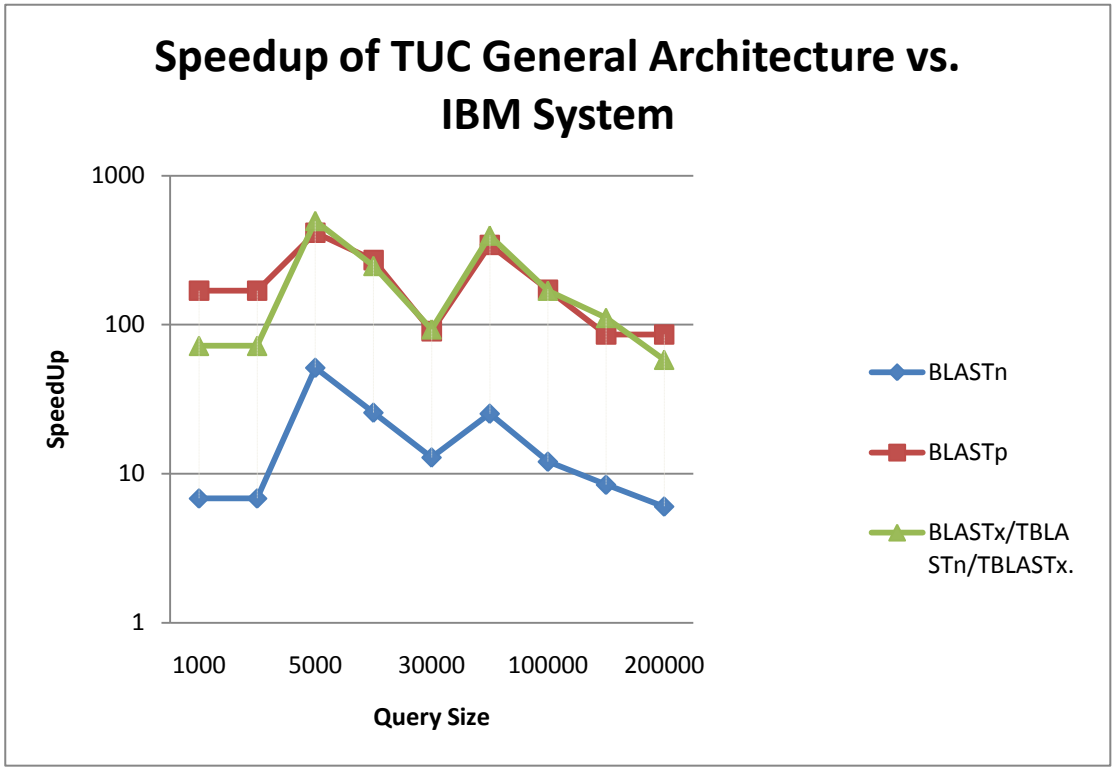

Figure 36: Speedup of TUC General Architecture vs. IBM System

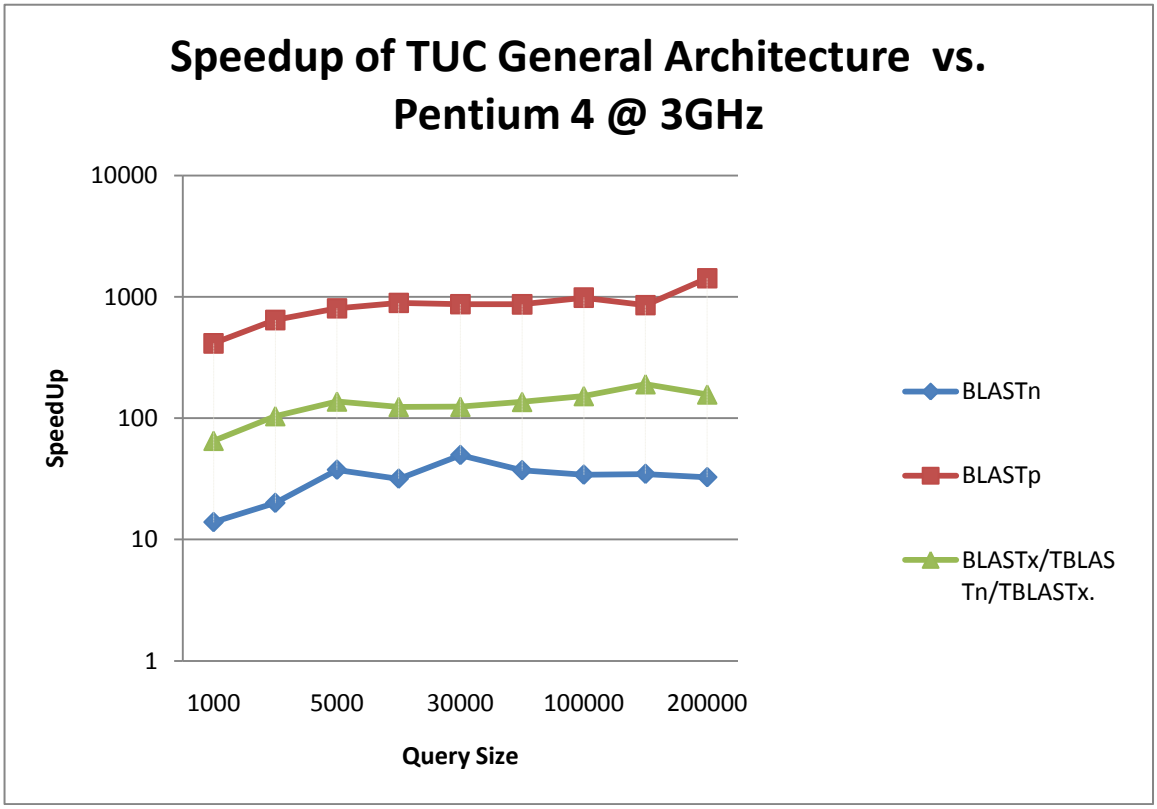

Figure 37: Speedup of TUC General Architecture vs. Pentium 4 @ 3GHz

Figures 35, 36, and 37 show the same speedup results with graphs, according to query size, for the general TUC architecture vs. a single IBM chip. IBM System with 16 processors and a Pentium 4 @ 3GHz respectively for BLASTn, BLASTp and BLASTx/TBLASTn/TBLASTx.

9.5.3 TUC General Architecture Comparison against Other Hardware Systems

Unfortunately it is very difficult to make an apples-to-apples comparison between hardware implementations because there are no sufficient data published for other architectures to determine actual speedup. From the preliminary data reported for other hardware implementations based on similar technology, using the results from Table 11 the TUC general architecture seems to be 5.85 times faster than Mercury Blast system and 74 times faster than TreeBLAST, with all three machines using implementation technology from the same year. We are careful to indicate that this seems to be the case, because hardware performance of other systems can only be inferred by published data and only in aggregate form. Although the following table is based on incomplete data (only aggregates have been published for the two architectures of this comparison), it was deemed appropriate as a rough estimate between the general TUC architecture and the FPGA/FLASH BLAST architecture. We can indirectly surmise some comparison data regarding the DeCypher engine for BLASTx/TBLASTnTBLASTx by converting the TUC throughput to the KaaMnt/sec units in Table 16. The KaaMnt is based on the multiplication on query size with the database size. The actual numbers that are reported for DeCypher are 182 KaaMnt/sec and for FPGA/FLASH BLAST architecture is 451 KaaMnt/sec, but as the details of what these numbers represent are unpublished this comparison should be considered only as a rough estimate.

Architecture	Query Length (Kaa)	BLASTx/TBLASTn TBLASTx Throughput KaaMnt/sec
TUC General Architecture	1	1,365
	2	2,730
	5	3,415
	10	3,410
	30	3,840
	50	3,750
	100	3,200
	150	3,150
	200	2,200
DeCypher	132,000	182
FPGA/FLASH	132,000	451

Table 28: Performance of Hardware Implementations of BLAST

9.5.4 TUC BLAST Generic Architecture V.2

Speed up of the improved architecture is measured only against the general purpose CPU due to the fact that technology of all other chips is quiet old and such a comparison would be unfair against the high end device we use. Table 29 shows the speed ups for the latest version of TUC architecture against latest version of NCBI BLAST software running on a high end PC.

Query Length (Characters)	*BLASTn*	*BLASTp*	*BLASTx*	*TBLASTn*	*TBLASTx*
1,000	21.58	331.65	12.39	122.90	218.15
2,000	25.82	494.02	21.03	237.97	380.28
5,000	72.19	1,120.21	36.23	424.43	788.76
10,000	95.91	1,137.15	31.78	356.71	675.00
30,000	98.95	1,645.31	36.07	307.89	672.41
50,000	55.14	1,392.86	30.84	322.02	605.17
100,000	51.05	1,877.27	34.44	345.10	628.57
150,000	65.06	1,958.14	35.56	334.29	615.79
200,000	69.38	1,934.38	35.34	338.46	628.57

Table 29: Speed up of TUC BLAST Generic Architecture V.2 vs. Intel Core 2 Duo @ 3GHz (single core)

All the measurements have been made running in one core, NCBI can run multithreaded with almost linear speed up in a multi-core processor. Numbers in Table 29 are illustrated at Figure 38.

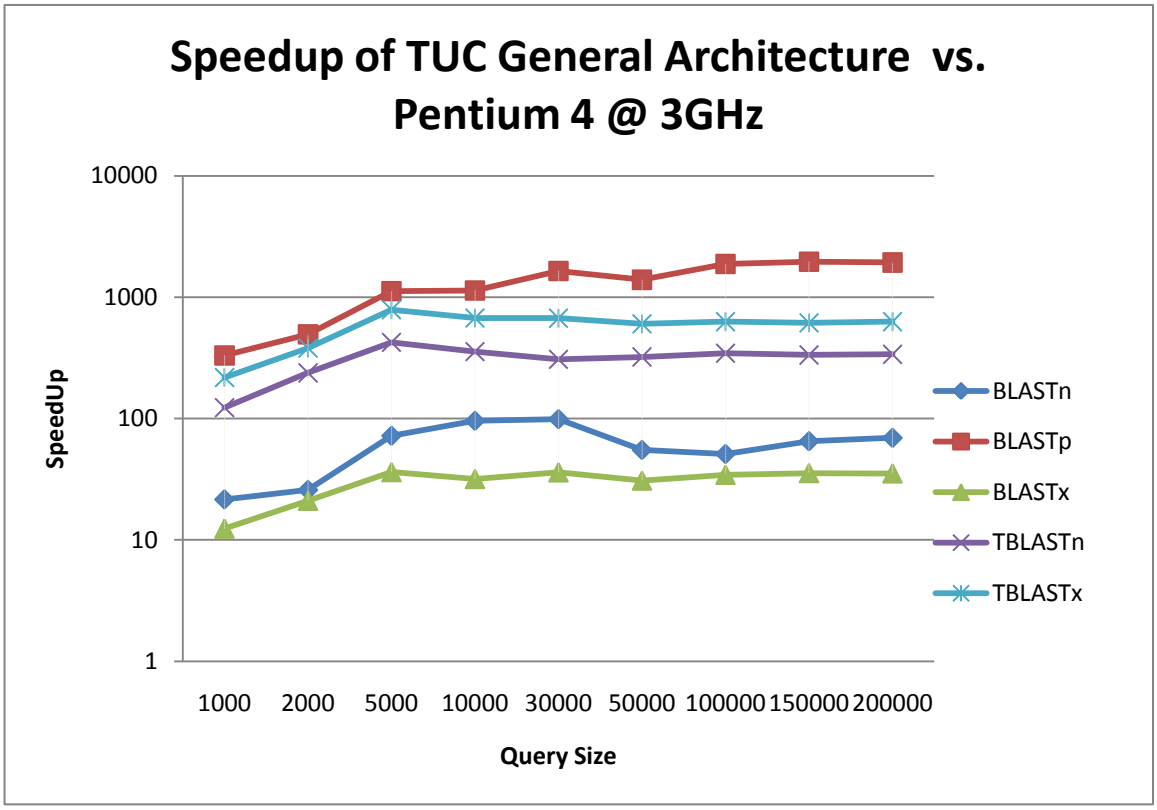

Figure 38: Speed up of TUC BLAST Generic Architecture V.2 vs. Intel Core 2 Duo @ 3GHz (single core)

As mentioned at paragraph 9.4 several actual run were made at the experimental platform. According to Table 22 and Table 29, the Table 30 is formatted. At Table 30 is calculated the speedup of the experimental platform against the Intel Pentium Dual Core 2 @ 3.00 GHz. It is also calculated the expected speedup considering Table 29 and the fact that only the equivalent of 6.7% of the reconfigurable resources of the Virtex 5 XC5VLX330T are used. Virtex 5 XC5VLX330T is the device that was used to calculate Table29.

Dataset	*Query Size*	*DataBase Size*	*Speedup XUP V5 vs. Intel*	*Expected Speedup XUP V5 vs. Intel*
Dataset1	***250***	***854,332***	***1.40***	***1.51***
Dataset2	***250***	***1,985,136***	***1.36***	***1.51***
Dataset3	***1,000***	***854,332***	***1.21***	***1.51***
Dataset4	***1,000***	***1,985,136***	***1.31***	***1.51***
Dataset5	***5,000***	***1,985,136***	***4.51***	***5.05***

Table 30: Run times measured at experimental platform and corresponding run time on Intel Pentium Dual Core 2 @ 3.00 GHz

Actual speedup considered to be close to expected. Differences are due two reasons. For the small Queries of length 250 and 1000 bottleneck is the Data Input rate as the Gigabit link works at 2/3 of the theoretical speed. For the Dataset5 with the 5,000 characters Query, bottleneck proved to be the lack of MicroBlaze computing power, for the 3rd step of the algorithm.

9.6 Results Synopsis

This thesis presents four different architectures to map the BLAST algorithm on reconfigurable logic and the respective performance achieved. There exist several research groups throughout the world which have presented their architectures and their results. As mentioned above, the BLAST algorithm has five different variations and its performance depends on query size. All these results have been presented in the previous sections of this chapter. Tables 31 and 32 summarize the most important results presented in this chapter. Table 31 presents the TUC architectures' performances vs. BLAST variation.

Architecture	*BLASTn Throughput (char 10^6/sec)*	*BLASTp Throughput (char 10^6/sec)*	*BLASTx/TBLASTn TBLASTx Throughput (char 10^6/sec)*
TUC 1st Generation (Query Size up to 1000)	**6,925**	**N/A**	**N/A**
TUC 2nd Generation	**320-8,192**	**N/A**	**N/A**
TUC BLAST Generic Architecture	**320-8,192**	**128-8,192**	**11-1,365**
TUC BLAST Generic Architecture V.2	**1,620-21,060**	**619-21,060**	**88-3,510**

Table 31: TUC BLAST Architectures Performance Synopsis

Table 32 presents the speedup of the execution time achieved for the different TUC BLAST Architectures vs. the execution time of the NCBI BLAST version running on a personal computer. In order to have a fair comparison, NCBI software version and personal computer are of the same time that the TUC Architectures was developed.

Architecture	*BLASTn*	*BLASTp*	*BLASTx/TBLASTn TBLASTx*
TUC 1st Generation (Query Size up to 1000)	**23.05**	**N/A**	**N/A**
TUC 2nd Generation	**13.91-49.64**	**N/A**	**N/A**
TUC BLAST Generic Architecture	**13.91-49.64**	**413.74-1,422.22**	**65.09-190.91**
TUC BLAST Generic Architecture V.2	**21.58-98.95**	**331.65-1,958.14**	**12.39-788.76**

Table 32: TUC BLAST Architectures SpeedUp Synopsis

9.7 Performance Evaluation – Technology Impact

We examine the performance of reconfigurable based hardware for BLAST algorithm since late 2005 when the first architecture was designed. and we evolve this architecture and its mapping to reconfigurable devices. At the same time a large common

effort at NCBI offers significant upgrade of BLAST software from version 2.2.12 then to version 2.2.19+ today.

Query Length (Characters)	*BLASTn*	*BLASTp*	*BLASTx*	*TBLASTn*	*TBLASTx*
1,000	**1.66**	**3.21**	**13.51**	**1.36**	**0.77**
2,000	**1.99**	**3.35**	**12.72**	**1.12**	**0.70**
5,000	**1.34**	**3.69**	**19.49**	**1.66**	**0.90**
10,000	**1.70**	**3.07**	**20.08**	**1.79**	**0.95**
30,000	**2.58**	**2.49**	**15.75**	**1.84**	**0.84**
50,000	**2.64**	**2.85**	**20.69**	**1.98**	**1.05**
100,000	**3.15**	**2.54**	**24.33**	**2.43**	**1.33**
150,000	**2.51**	**2.87**	**29.91**	**3.18**	**1.73**
200,000	**2.31**	**3.56**	**35.57**	**3.71**	**2.00**
Average	**2.21**	**3.07**	**21.34**	**2.12**	**1.14**

Table 33: Speed up of NCBI Software at 2008 technology vs. 2005 technology

At technology level processors are two generations ahead from Pentium 4 to Core 2 architecture, FPGAs that were used in first place were medium/large Virtex 4 and for the latest designs are the larger Virtex 5 devises. Taking into account the above technology evolvement and development of the BLAST implementations we can compare the software evolvement the hardware evolvement.

Table 33 shows that there is a speed up from 2 to 3 to the most of the cases except the BLASTx and TBLASTx where the 3 software variations separate from the software that was running to a Pentium 4 to the up to date Pentium.

The same calculations have been made for the hardware implementations from 2nd generation to TUC BLAST Generic Architecture V.2. At Table 34 it is shown that hardware improvement was 2.5 to 8 times. Improvement was bigger for large queries and this is can be explained to the larger amount of memory that is available to serve larger queries.

Query Length (Characters)	*BLASTn*	*BLASTp*	*BLASTx*	*TBLASTn*	*TBLASTx*
1,000	**2.57**	**2.57**	**2.57**	**2.57**	**2.57**
2,000	**2.57**	**2.57**	**2.57**	**2.57**	**2.57**
5,000	**2.57**	**5.14**	**5.14**	**5.14**	**5.14**
10,000	**5.14**	**3.92**	**5.15**	**5.15**	**5.15**
30,000	**5.14**	**4.70**	**4.57**	**4.57**	**4.57**
50,000	**3.92**	**4.57**	**4.68**	**4.68**	**4.68**
100,000	**4.70**	**4.84**	**5.50**	**5.50**	**5.50**
150,000	**4.70**	**6.58**	**5.57**	**5.57**	**5.57**
200,000	**5.06**	**4.84**	**8.00**	**8.00**	**8.00**
Average	**4.04**	**4.41**	**4.86**	**4.86**	**4.86**

Table 34: Speed up of TUC Hardware Generic Architecture V.2 vs. Generation 1

The same calculations, taking into account the first hardware generation for BLASTn variation with queries up to 1000 characters, are presented at Table 35.

Generation	*HW Throughtput*	*SW Throughtput*	*Speed Up*
1	**6,924**	**319.25**	**21.69**
2	**8,192**	**588.75**	**13.91**
3	**21,960**	**976.00**	**22.50**

Table 35: Speed up of NCBI SW and TUC HW

9.8 Algorithm Sensitivity vs. Performance

Running BLAST software is quite complicated and a lot of parameters are involved. There are a lot of options that have to do with the nature of genetic data and the biological analysis that the user of the software wants to do. One of the major options that a biologist has to determine in order to do genomic analysis with BLAST is its sensitivity. As W-mers decrease in size the algorithm sensitivity increases. Sensitivity is directly linked to the size of the W-mer. W-mer size can vary from 4 to an arbitrary large number, where 28 is the size of W-mer in order to run MegaBLAST. If W-mer length or word_size (argument of BLAST) is set to 4 then the runs of the algorithm would be in the greatest sensitivity and will give more detailed results. If W-mer length sets to 28 then the runs of the algorithm would be in the smaller sensitivity and will give less detailed results.

The smaller size of the W-mer the more computation time is needed to run the same datasets on a general purpose processor. All the measurements that have been taken

are for W-mer with length of 11 which is typical. The TUC architecture performance does not get degraded and in some cases it can be improved when the W-mer length gets smaller. The design of the memory scheme needed for step 2 is proportional to the W-mer length and can become smaller or stay the same. Step 3 is executed on the Microblaze which has only 8 engines for step 2 attached and thus there is no bottleneck. This approach has not been studied in depth with statistical analysis of BLAST algorithm for every length of W-mer and has to be proved experimentally, although some quantitative data are given on order to find out the potential of TUC design.

Several measurements have been taken with an Intel Core 2 Duo E8400 at 3 GHz with 2 GB RAM and Microsoft Windows XP Professional Version 2002 SP3. It was used the latest available version of Intel(R) VTune(TM) Performance Analyzer 9.0. Table 36 Shows for BLASTn and query length 1000 and several W-mers sizes the execution time of the Intel PC, the Virtex 5 and the respective speedup and figure 39 illustrates it.

W-mer Size	***NCBI SW Throughtput (Char 10^6/sec)***	***TUC HW Throughtput (Char 10^6/sec)***	***Speed Up HW vs. SW***
4	**1.13**	**21,060**	**18,582.24**
5	**3.94**		**5,351.76**
6	**13.66**		**1,541.28**
7	**46.37**		**454.17**
8	**96.00**		**219.37**
9	**211.07**		**99.78**
10	**347.58**		**60.59**
11	**976.00**		**21.58**

Table 36: NCBI throughput according to W-mer Length

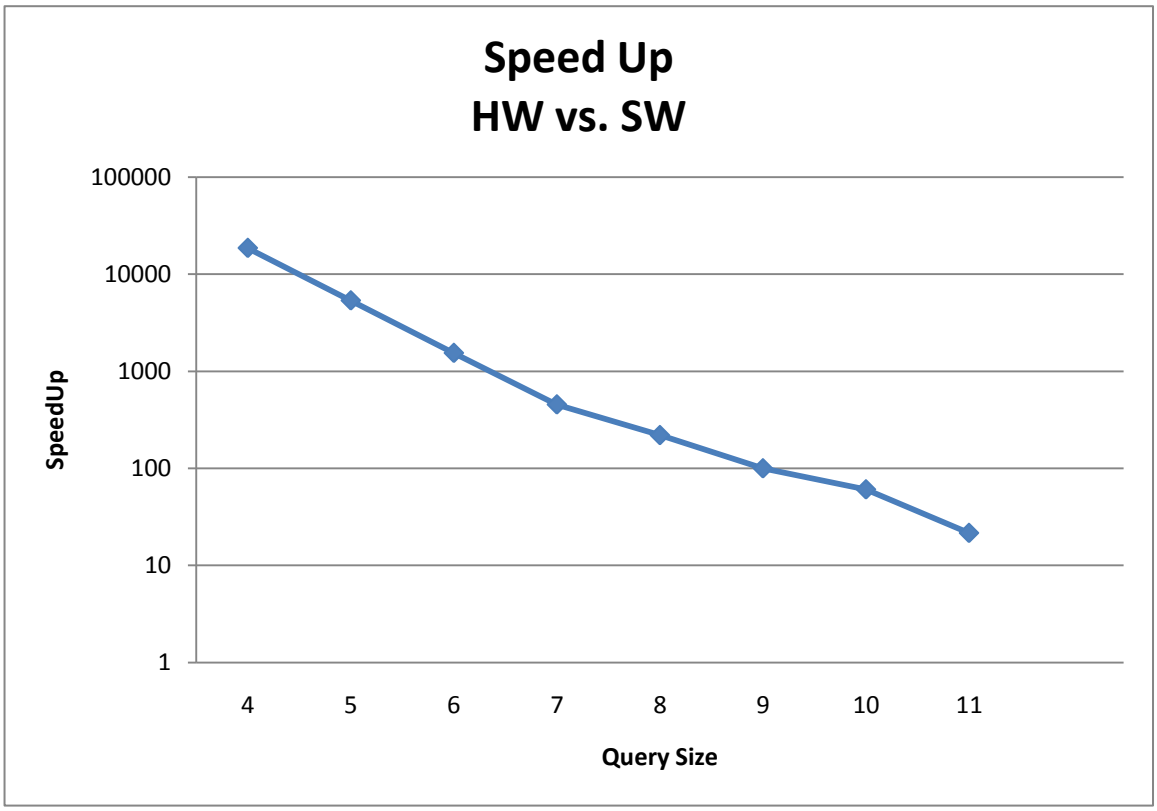

Figure 39: SpeedUp of TUC HW vs NCBI SW according to W-mer length

9.9 Energy consumption

This section presents the energy consumption issues for the TUC architectures resented above; although it was not in our initial intention to develop a low power system. We measured the energy consumption that was needed by the different implemented architectures to scan 10^6 database characters.

A high end board, like the HiTechGlobal HTG-LX330T, which is available at the MHL, is powered by a normal PC power supply of 400 watts. This board can achieve throughputs as mentioned in Table 19 (BLAST Generic Architecture V.2 System Performance). A PC was also used as the base platform for the high end board. The power consumption of the additional PC is also 400 watts. Thus, the whole system consumes in total approximately 800 watts. Table 37 shows the estimated energy consumption for such system per 10^6 database characters. Although these calculations are approximate, they provide a reasonable and realistic estimate of what can be expected in

general, assuming a worst-case scenario both for the PC and for the reconfigurable hardware (i.e. fully used power supply).

Query Length (Characters)	*BLASTn Joule/char 10^6*	*BLASTp Joule/char 10^6*	*BLASTx/TBLASTn TBLASTx Joule/char 10^6*
1,000	**0.0380**	**0.0380**	**0.2279**
2,000	**0.0380**	**0.0380**	**0.2279**
5,000	**0.0380**	**0.0380**	**0.2279**
10,000	**0.0380**	**0.0760**	**0.4558**
30,000	**0.0760**	**0.1899**	**1.3675**
50,000	**0.1519**	**0.3419**	**2.2792**
100,000	**0.2659**	**0.6457**	**4.5455**
150,000	**0.3799**	**0.9501**	**6.8376**
200,000	**0.4938**	**1.2924**	**9.0909**

Table 37: MicroBlaze Based System Energy Consumption

A usual PC consumes 400 watts achieving performance for NCBI BLAST as it was measured at Table 11. Table 38 is calculated using the same metrics for energy consumption.

Query Length (Characters)	*BLASTn Joule/char 10^6*	*BLASTp Joule/char 10^6*	*BLASTx Joule/char 10^6*	*TBLASTn Joule/char 10^6*	*TBLASTx Joule/char 10^6*
1,000	**0.4098**	**6.2992**	**1.4118**	**14.0056**	**24.8602**
2,000	**0.4905**	**9.3831**	**2.3961**	**27.1186**	**43.3369**
5,000	**1.3711**	**21.2766**	**4.1284**	**48.3676**	**89.8876**
10,000	**1.8217**	**43.1965**	**7.2424**	**81.3008**	**153.8462**
30,000	**3.7587**	**156.2500**	**24.6609**	**210.5263**	**459.7701**
50,000	**4.1894**	**238.0952**	**35.1494**	**366.9725**	**689.6552**
100,000	**6.7866**	**606.0606**	**78.2779**	**784.3137**	**1,428.5714**
150,000	**12.3571**	**930.2326**	**121.5805**	**1,142.8571**	**2,105.2632**
200,000	**17.1306**	**1,250.0000**	**160.6426**	**1,538.4615**	**2,857.1429**

Table 38: PC System Energy Consumption

According to Tables 37 and 38, Table 39 shows the energy efficiency (how many times less energy is consumed) of FPGA based system vs. normal PC.

Table 39 and Table 29 show that power consumption is proportional to the speed up divided by a factor of 2. Table 39 shows also that FPGA based system can be up to 2 orders of magnitude more efficient than a general purpose computer. Energy consumption is an issue of great importance for the large Bioinformatics projects. For example, a current study for phylogenomic alignments consists of almost 1,500 genes and requires 2.25 million CPU hours on an IBM BlueGene/L supercomputer. This shows that the energy consumption is one of the most important aspects of the project.

Query Length (Characters)	*BLASTn PC/TUC architecture*	*BLASTp PC/TUC architecture*	*BLASTx PC/TUC architecture*	*TBLASTn PC/TUC architecture*	*TBLASTx PC/TUC architecture*
1,000	**10.79**	**165.83**	**6.19**	**61.45**	**109.07**
2,000	**12.91**	**247.01**	**10.51**	**118.98**	**190.14**
5,000	**36.10**	**560.11**	**18.11**	**212.21**	**394.38**
10,000	**47.96**	**568.57**	**15.89**	**178.35**	**337.50**
30,000	**49.47**	**822.66**	**18.03**	**153.95**	**336.21**
50,000	**27.57**	**696.43**	**15.42**	**161.01**	**302.59**
100,000	**25.53**	**938.64**	**17.22**	**172.55**	**314.29**
150,000	**32.53**	**979.07**	**17.78**	**167.14**	**307.89**
200,000	**34.69**	**967.19**	**17.67**	**169.23**	**314.29**

Table 39: Energy Efficiency of TUC Architecture vs. PC

As Table 39 is calculated on a real base model which considers the performance measurements and not energy or power measurements, it should be stretched as substantial evidence that FPGAs can be energy effective. This result can be applied only to applications that have a certain amount of data to compute, like bioinformatics and not to applications that work occasionally when data are produced, such as network applications. Further research has to be done and more power measurements have to be taken in order to define the problem characteristics which make FPGAs energy efficient platforms.

9.10 Cost effectiveness

The system's cost effectiveness is another important issue to examine. The cost has to be considered into two different aspects: the cost to buy the systems and the operational cost.

The cost to build a high-end FPGA-based system is approximately 8,500 Euros, e.g. 7,500 Euros for the FPGA board and 1,000 Euros for a personal computer. As the performance of the FPGA-based system is 20 to several hundred's times faster than the performance achieved by a personal computer and taking into account that the computer power consumption for the specific program costs from 20,000 Euros up to several hundred's thousands Euros, it can be concluded that the FPGA based system is from 2.5 times cheaper. up to 100 times cheaper vs. the corresponding PC implementing the

BLAST algorithm. These calculations are considered without the cost to develop hardware or software.

The operational cost is mostly in the energy consumption, where FPGAs proved to be more efficient and the administration cost which is much higher for a PC farm than a single, FPGA-based system.

Chapter 10

Conclusions and Future Work

10.1 Conclusions

Concluding this thesis several points can be made according to technology impact, BLAST algorithm and Bioinformatics and hardware implementations.

Technology

FPGAs can offer significant speed up on algorithms in which the datapath can't be trivially mapped to parallel processing. As the devices get bigger and offer more components in addition to the reconfigurable fabric these algorithms can be better mapped to FPGAs by taking advantage of the resources. As in the general purpose processors, in the reconfigurable processors the amount of memory and the way that is organized is critical for system design. Another critical issue for such designs is the I/O, Xilinx devices offer significant resources for this problem but this is the one only aspect of the solution, in addition to the platform which is used and consequently the way in which the device is connected to the "rest of the world".

BLAST Algorithm

This algorithm proved to be a challenging problem with great potential for design of hardware modules. There are 5 variations with many different configurations (W-mer length, scoring scheme etc.) that can be made and a generic hardware solution is impossible to be made without the use of a general purpose processor such as the PowerPC or the MicroBlaze, both of which were used in different implementations in this dissertation. The wide usage of the BLAST algorithm by biologists and the rate of the data that need to be processed, show that a platform with special-purpose hardware running BLAST efficiently can be very valuable to the scientific community.

Bioinformatics

BLAST is the most significant and widely used algorithm of bioinformatics. It's a streaming data algorithm where throughput rather than latency is significant. Bioinformatics algorithms are about one hundred in count, and most of them are computationally intensive. The TUC BLAST implementation shows that such problems with large memory needs, a lot of variations and different configuration or with I/O problems can be mapped efficiently on an FPGA and offer significant performance boosting.

10.2 Towards a Reconfigurable Bioinformatics Processor

As it has been mentioned before, bioinformatics is a research area and industry sector with explosive development. Genetic data generation is enormous and therefore their handling becomes an increasingly difficult problem. A generation of a bioinformatics processor is a challenge that can solve a real-world problem.

For such BioProcessor with today technology two different approaches can been proposed. First, and the most obvious approach is a hardcore BioProcessor which fully implements a special instruction set which is designed to perform faster for the every common instruction that Bioinformatics algorithms perform. The main advantage of such a solution is that the performance for the common case would be very high due to clock speed, as it happens with the DSPs. On the other hand due to the fact that algorithms evolve continuously it is very difficult to develop a generic enough processor.

A coarse-grain reconfigurable processor with specially designed available modules for bioinformatics could be a different approach. In such a device performance can be much better than a generic FPGA but not as fast as a fully custom processor. A reconfigurable part can be included together with a general-purpose hardcore processor.

For both solutions the first step to implement is a very detailed statistical analysis of the characteristics of bioinformatics algorithms (or, at least the major ones), in order to find what are the common operations. Another very significant step is the specification of

a standard interface between such a processor and the outside world which will be of very high speed and will help all applications to run on such a processor easily.

REFERENCES

[1] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", ACM Computing Surveys, Volume 34, Issue 2, Pages: 171 – 210, 2002.

[2] S. Hauck, "The Roles of FPGAs in Reprogrammable Systems" , *Proceedings of the IEEE*, Vol, 86, No, 4, pp, 615-638, April, 1998.

[3] R. Hudson, D. Lehn, P. Athanas, "A run-time reconfigurable engine for image interpolation," In *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1998*, pp,88-95, 15-17 Apr 1998.

[4] M. Shand, L. Moll, "Hardware/software integration in solar polarimetry", *In Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1998*, pp,96-104, 15-17 Apr 1998.

[5] W. Luk, T. Lee, J. Rice, N. Shirazi, P. Cheung, "Reconfigurable Computing for Augmented Reality", *In Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1999*, pp, 136-145, Apr 1999.

[6] M. Rencher, B. Hutchings, "Automated Target Recognition on Splash 2" *In Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1997, for Custom Computing Machines, 1997*, pp, 192-200, Apr 1997.

[7] A. Elbirt, and C. Paar, "An FPGA implementation and performance evaluation of the Serpent block cipher", In *Proceedings of the Eighth ACM/SIGDA International Symposium on FPGAs* 2000, pp 33–40, *2000.*

[8] C. Patterson, "High Performance DES Encryption in Virtex(tm) FPGAs Using Jbits(tm)", In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, 2000*, pp 113-121, 2000.

[9] M. Leong, O. Cheung, K. Tsoi, and P. Leong, "A Bit-Serial Implementation of the International Data Encryption Algorithm IDEA", In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, 2000*, pp 122-131, 2000,

[10] A. Dandalis, V. Prasanna, and J. Rolim, "An Adaptive Cryptographic Engine for IPSec Architectures", In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, 2000*, pp 132-141, 2000.

[11] H. Kim and W. Mangione-Smith, "Factoring large numbers with programmable hardware", In *Proceedings of the Eighth ACM/SIGDA International Symposium on FPGAs ACM/SIGDA International Symposium on FPGAs*, FPGA '00, pp 41–48, 2000.

[12] J. Hauser, J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines 1997*, pp 12-21, 1997.

[13] K. Leung, K. Ma, W. Wong, and P. Leong, "FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor", In *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines* 2000, pp 68-76, 2000.

[14] C. Kachris, N. Bourbakis, A. Dollas: "A Reconfigurable Logic-Based Processor for the SCAN Image and Video Encryption Algorithm", International Journal of Parallel Programming 31(6): 489-506, 2003.

[15] M. Piacentino, G. VanderWal, M. Hansen, "Reconfigurable Elements for a Video Pipeline Processor," *In Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1999*, pp, 82-91, Apr 1999.

[16] M. Weinhardt and W. Luk, "Pipeline Vectorization for Reconfigurable Systems", *In Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1999*, pp,52-62 , Apr 1999.

[17] E. Sotiriades, A. Dollas, and P. Athanas *"Hardware-Software Codesign and Parallel Implementation of a Golomb Ruler Derivation Engine"* Proceedings 8th International IEEE Symposium on Field-Programmable Custom Computing Machines, pp, 227-235, Napa Valley, April 17-19, 2000.

[18] A. Dollas, E. Sotiriades, A. Emmanouilides *"Architecture and Design of GE1, a FCCM for Golomb Ruler Derivation"*, Proceedings, 6th International IEEE Symposium on FPGA's for Custom Computing Machines, pp, 48-56, Napa Valley, April 15-17, 1998.

[19] N. Shirazi, P. Athanas, and A. Abbott, "Implementation of a 2-D Fast Fourier Transform on an FPGA-Based Custom Computing Machine", in Proc, FPL, 1995, pp,282-292.

[20] K. Underwood, R. Sass, and W. Ligon, "Acceleration of a 2D-FFT on an Adaptable Computing Cluster", In *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* 2001, pp, 180-189, 2001.

[21] L. Huelsbergen," A representation for dynamic graphs in reconfigurable hardware and its application to fundamental graph algorithms", In *Proceedings of the 2000 ACM/SIGDA Eighth international Symposium on Field Programmable Gate Arrays* 2000, pp105-115, 2000.

[22] P. Zhong, M. Martonosi, P. Ashar, and S. Malik, "Accelerating Boolean Satisfiability with Configurable Hardware", In *Proceedings of the IEEE Symposium on Fpgas For Custom Computing Machines* 1998, pp 186–195, 1998.

[23] W. Huang, N. Saxena, and E, McCluskey, "A Reliable LZ Data Compressor on Reconfigurable Coprocessors", In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* 2000, pp 249–258, 2000.

[24] P. Stogiannos, A. Dollas, V. Digalakis "A Configurable Logic Based Architecture for Real-Time Continuous Speech Recognition Using Hidden Markov Models", VLSI Signal Processing 24(2-3): 223-240 (2000).

[25] P. Graham and B. Nelson "Genetic algorithms in software and in hardware—A performance analysis of workstations and custom computing machine implementations", In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines 1996, pp* 216–225, 1996.

[26] C. Paar and M. Rosner, "Comparison of arithmetic architectures for Reed-Solomon decoders in reconfigurable hardware", in Proc, FCCM, 1997, pp,219-225,

[27] Y. Li and W. Chu, "Implementation of single precision floating point square root on FPGAs", in Proc, FCCM, 1997, pp,226-233,

[28] www,opencores,org

[29] A. Dollas, D. Pnevmatikatos, N. Aslanides, S. Kavvadias, E. Sotiriades, S. Zogopoulos, K. Papademetriou, N. Chrysos, K. Harteros, E. Antonidakis, N. Petrakis, *"Architecture and Applications of PLATO, a Reconfigurable Active Network Platform,"* Preliminary Proc, FCCM, 2001,

[30] G. Brebner, "Single-Chip Gigabit Mixed-Version IP Router on Virtex-II Pro", in Proc, FCCM, 2002, pp,35-44.

[31] S. Young, P. Alfke, C. Fewer, S. McMillan, B. Blodget, and D. Levi, "A High I/O Reconfigurable Crossbar Switch", in Proc, FCCM, 2003, pp,3-10.

[32] P. Bellows, J. Flidr, T. Lehman, B. Schott, and K. Underwood, "GRIP: A Reconfigurable Architecture for Host-Based Gigabit-Rate Packet Processing", in Proc, FCCM, 2002, pp,121-130.

[33] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West, "ShareStreams: A Scalable Architecture and Hardware Support for High-Speed QoS Packet Schedulers", in Proc, FCCM, 2004, pp,115-124.

[34] Z. Baker and V. Prasanna, "A Methodology for Synthesis of Efficient Intrusion Detection Systems on FPGAs", in Proc, FCCM, 2004, pp,135-144.
[35] I. Sourdis and D. Pnevmatikatos, "Pre-Decoded CAMs for Efficient and High-Speed NIDS Pattern Matching", in Proc, FCCM, 2004, pp,258-267.
[36] Y. Cho and W. Mangione-Smith, "Fast Reconfiguring Deep Packet Filter for 1+ Gigabit Network", in Proc, FCCM, 2005, pp,215-224.
[37] I. Sourdis and D. Pnevmatikatos, "Fast, Large Scale String Matching for a 10 Gbps FPGA-based NIDS", Book chapter in "New Algorithms, Architectures, and Applications for Reconfigurable Computing", Patrick Lysaght and Wolfgang Rosenstiel(Eds,), Chapter 16, pp, 195-207, ISBN 1-4020-3127, Springer, 2005.
[38] P. Higgs, T. Attwood, "Bioinformatics And Molecular Evolution", Blackwell Publishing, 2005.
[39] www.ncbi.nih.gov/Genbank/genbankstats.html
[40] www.ncbi.nih.gov
[41] www.embl-heidelberg.de
[42] pir.georgetown.edu
[43] scop.wehi.edu.au/gsdb/gsdb.html
[44] www.ddbj.nig.ac.jp
[45] www.ebi.ac.uk
[46] http://www.isb-sib.ch/
[47] R. Casey, "BLAST Sequences Aid in Genomics and Proteomics", Business Intelligence Network October 11, 2005, (http://www,b-eye-network,com/view/1730).
[48] B. Needleman, and C. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins,", J. Mol. Biol., vol. 48, pp 443-453, 1970.
[49] T. Smith, and M. Waterman, "Identification Of Common Molecular Subsequences," Elsevier J. Mol. Biol., vol. 147, pp 195-197, 1981.
[50] W. Pearson, and D. Lipman, "Improved tools for biological sequence analysis" Proceedings of the National Academic Science of the USA, vol 85, pages 2444–2448, 1988.
[51] S. Altschul, W. Gish, W. Miller, and E. Myers, "Basic Local Alignment Search Tool" Elsevier J. Mol. Biol., vol. 215, pp 403-410, 1990.
[52] J. Meidanis and J. Setubal, "Introduction to Computational Molecular Biology", PWS Publishing Company, 1997.
[53] D. Hoang et. al. "FPGA Implementation of Systolic Sequence Alignment", Proceedings of the 2nd International Workshop on Field-Programmable Logic and Applications, Lecture Notes in Computer Science 705, pp 183-191, 1992.
[54] D. Hoang "Searching Genetic Databases on Splash 2", Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (FCCM), pp 185-191, 1993.
[55] S. Guccione and E. Keller "Gene Matching Using JBits", Proceedings of the 12th International Conference on Field-Programmable Logic and Applications, Lecture Notes In Computer Science; Vol. 2438, pp 1168-1171, 2002.
[56] K. Puttegowda et. Al. "A Run-Time Reconfigurable System for Gene-Sequence Searching", Proceedings, 16th International Conference on VLSI Design pp 561 – 566, New Delhi 2003.
[57] T. Oliver, B. Schmidt, D. Maskel ""Reconfigurable Architectures for Bio-sequence Database Scanning on FPGAs", IEEE Transactions on Circuits and Systems II, Vol, 52, No, 12, pp, 851-855, 2005.
[58] K. Muriki, K. Underwood, and R. Sass, "RC-BLAST: Towards an open source hardware implementation," In Proceedings of the International Workshop on High Performance Computational Biology (2005).

[59] M. Herbordt, J. Model, Y. Gu, B. Sukhwani, T. VanCourt, "Single Pass, BLAST-Like, Approximate String Matching on FPGAs"14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06), pp, 217-226, 2006.
[60] M. Herbordt, J. Model, B. Sukhwani, Y. Gu, and T. VanCourt, "Single pass streaming BLAST on FPGAs", Parallel Computing, vol. 33, issue 10-11 (Nov, 2007), pp 741-756, 2007.
[61] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, and J. Lancaster, "Biosequence Similarity Search on the Mercury System," In Proc. of the IEEE 15th Int'l Conf, on Application-Specific Systems, Architectures and Processors, September 2004, pp, 365-375
[62] J. Lancaster, J. Buhler, R. Chamberlain, "Acceleration of Ungapped Extension in Mercury BLAST", 7th workshop on media and streaming processors, Barcelona, Spain, November 12, 2005
[63] A. Buhler et al. "Mercury blastn: faster dna sequence comparison using a streaming hardware architecture", RSSI, 2007.
[64] Washington University, "Method and apparatus for performing biosequence similarity searching" International Patent WO/2006/096324, 2006
[65] D. Lavenier, L. Xinchun, G. Georges, "Seed-based Genomic Sequence Comparison using a FPGA/FLASH Accelerator", in Proceedings of IEEE International Conference on Field Programmable Technology, 2006,(FPT '06), pp, 41 - 48, 2006.
[66] F. Xia, Y. Dou and J. Xu, "Families of FPGA-Based Accelerators for BLAST Algorithm with Multi-seeds Detection and Parallel Extension", Bioinformatics Research and Development, Second International Conference, BIRD 2008, pp, 43-57, Vienna, Austria, July 7-9, 2008.
[67] F. Xia, Y. Dou, J. Xu, "FPGA-Based Accelerators for BLAST Families with Multi-Seeds Detection and Parallel Extension," The 2nd International Conference on Bioinformatics and Biomedical Engineering, 2008, ICBBE 2008,, pp,58-62, 16-18 May 2008.
[68] http://www.timelogic.com/benchmark_blast.html
[69] C. Chang "BLAST Implementation on BEE2"Electrical Engineering and Computer Science University of California at Berkeley (2005), http://bee2.eecs.berkeley.edu
[70] http://www.bio-itworld.com/issues/2006/dec-jan/inside-the-box/
[71] C. Sosa et. Al. "Some Practical Suggestions for Performing NCBI BLAST Benchmarks on a pSeries ™ 690 System", http://www,redbooks,ibm,com/abstracts/redp0437,html?Open.
[72] R. Radahakrishnan, R. Ali, G. Kochhar, K. Chadalavada, R. Rajagopalan, "Performance Characterization of BLAST on 32-bit and 64-bit Dell PowerEdge Servers", Dell Power Solutions, February 2005.
[73] http://images.apple.com/acg/pdf/AGBLAST229-PerfData22Jun04.pdf
[74] G. Mplemenos, I. Papaeftathiou, "MPLEM An 80-processor FPGA Based Multiprocessor System", Field-Programmable Custom Computing Machines, 2008, FCCM 2008.
[75] B. Bloom, "Space/time trade-offs in hash coding with allowable errors", Communications of the ACM, vol, 13, issue 7, pp 422–426, 1970.
[76] M. Dayhoff, R. Schwartz, and B. Orcutt, "A model of evolutionary change in proteins," In: Dayhoff,M,O, (ed), Atlas of Protein Sequence and Structure National Biomedical Research Foundation, Washington, DC, pp, 345–352, 1978.
[77] S. Henikoff, J. Henikoff, "Amino Acid Substitution Matrices from Protein Blocks", PNAS 89: 10915–10919, 1992.
[78] J. Park, Y. Qiu and M. Herbordt "CAAD BLASTP: NCBI BLASTP Accelerated with FPGA-Based Pre-Filtering" fccm, 17th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'09), 2009.

[79] P. Afratis, E. Sotiriades, G. Chrysos, S. Fytraki, and D. Pnevmatikatos, "A rate-based prefiltering approach to BLAST acceleration," in Proc,IEEE Conference on Field Programmable Logic and Applications, 2008.

[80] E. Sotiriades, C. Kozanitis, G. Chrysos, A. Dollas "Rapid Phototyping of a System-on-a-Chip for the BLAST Algorithm Implementation", Proceedings, 17th International IEEE Workshop on Rapid System Prototyping RSP-2006, pp, 223-229, Chania, Greece, 14-16 June, 2006, Computer Society Press.

[81] E. Sotiriades, C. Kozanitis, A. Dollas, "FPGA based Architecture of DNA Sequence Comparison and Database Search", Proceedings 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, p 193, ,at the 13th Reconfigurable Architectures Workshop Rhodes, Greece, 25-29 April, 2006

[82] E. Sotiriades, C. Kozanitis, A. Dollas, "Some Initial Results on Hardware BLAST Acceleration with a Reconfigurable Architecture", Proceedings 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, p 251 , at the 5th IEEE International Workshop on High Performance Computational Biology (HiCOMB2006), Rhodes, Greece, 25-29 April, 2006.

[83] E. Sotiriades, A. Dollas "A General Reconfigurable Architecture for the BLAST algorithm", The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, Special Issue on Computing Architectures and Acceleration for Bioinformatics Algorithms, Kluwer Academic Publishers Volume 48, Issue 3 Pages: 189 – 208, September, 2007.

[84] P. Afratis, K. Galanakis, E. Sotiriades, G. Mplemenos, G. Chrysos, Y. Papaefstathiou, D. Pnevmatikatos "Design and Implementation of a Database Filter for BLAST Acceleration" Accepted to be presented at Design Automation & Test in Europe (DATE 2009), Nice France, April 24-29 2009.

[85] P. Afratis, E. Sotiriades, G. Chrysos, S. Fytraki, D. Pnevmatikatos, "Preprocessor for BLAST Algorithm Data", (in Greek), 2nd Greek ECE Student Conference, p,12, Athens, April 2008.

[86] I. Kartsonakis, "Development of PCIe driver for open source system (Linux) and VHDL code for fast serial interface between PC and Virtex-5 Xilinx", Diploma Thesis, Microprocessor Hardware Laboratory, Technical University of Crete, 2009.

[87] R. Bittner, "Bus mastering PCI express in an FPGA," In Proceeding of the ACM/SIGDA international Symposium on Field Programmable Gate Arrays, FPGA '09, pp 273-276, Monterey, California, USA, February 22 - 24, 2009.

[88] D. Vasilopoulos, "Study and Platform Development for High Speed serial Interface with FPGA" Diploma Thesis, Microprocessor Hardware Laboratory, Technical University of Crete, 2009.

Appendix A - Impact of this work

Publications from this work

Journal papers

1. **Euripides Sotiriades**, Apostolos Dollas "A General Reconfigurable Architecture for the BLAST algorithm", The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, Special Issue on Computing Architectures and Acceleration for Bioinformatics Algorithms, Kluwer Academic Publishers Volume 48, Issue 3 Pages: 189 – 208, September, 2007.

Conference papers

2. Panagiotis Afratis, Konstantinos Galanakis, **Euripides Sotiriades**, Georgios-Grigorios Mplemenos, Grigorios Chrysos, Yiannis Papaefstathiou, Dionisios Pnevmatikatos "Design and Implementation of a Database Filter for BLAST Acceleration" Design Automation & Test in Europe (DATE 2009), Nice France, pp166-171, April 24-29 2009.
3. **Euripides Sotiriades**, Christos Kozanitis, Grigorios Chrysos, Apostolos Dollas ***"****Rapid Phototyping of a System-on-a-Chip for the BLAST Algorithm Implementation",* Proceedings, 17th International IEEE Workshop on Rapid System Prototyping RSP-2006, pp, 223-229, Chania, Greece, 14-16 June, 2006, Computer Society Press.
4. **Euripides Sotiriades**, Christos Kozanitis, Apostolos Dollas, "*FPGA based Architecture of DNA Sequence Comparison and Database Search",* Proceedings 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, p 193, ,at the 13th Reconfigurable Architectures Workshop Rhodes, Greece, 25-29 April, 2006.
5. **Euripides Sotiriades**, Christos Kozanitis, Apostolos Dollas, *Some Initial Results on Hardware BLAST Acceleration with a Reconfigurable Architecture*, Proceedings 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, p 251 , at the 5th IEEE International Workshop on High Performance Computational Biology (HiCOMB2006), Rhodes, Greece, 25-29 April, 2006.

Short or Poster Conference papers

6. Panagiotis Afratis, **Euripides Sotiriades**, Grigorios Chrysos, Sotiria Fytraki, Dionisios Pnevmatikatos, "*A Rate-based Prefiltering Approach to BLAST Acceleration*", Accepted at International Conference on Field Programmable Logic and Applications (FPL 2008), Heidelberg, Germany, 08-10 September 2008.
7. Panagiotis Afratis, **Euripides Sotiriades**, Grigorios Chrysos, Sotiria Fytraki, Dionisios Pnevmatikatos, "*Preprocessor for BLAST Algorithm Data*", (in Greek), 2nd Greek ECE Student Conference, p,12, Athens, April 2008.

8. **Euripides Sotiriades**, Apostolos Dollas, "*Design Space Exploration for the BLAST Algorithm Implementation*" Proceedings, 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007), pp, 323-326, Napa Valley, April, 2007 ,

Invited Conference Papers (Unrefereed)

9. **Euripides Sotiriades,** Apostolos Dollas "A General Reconfigurable Architecture for the BLAST algorithm", 1st Hellenic Bioinformatics & Medical Informatics Meeting (HBMIM), Foundation Biological Research Academy of Athens (FBRAA), Athens, October 2007.
10. **Euripides Sotiriades**, "Reconfigurable Hardware for Bioinformatics Algorithms", 1st Cretan Bioinformatics Forum, FORTH, Herakleion, June 19th, 2006.

Papers Citation (no self reference) untul February, 2011: 33 including citations from all "competitive" groups.

Invited Lectures

- "A General Reconfigurable Architecture for the BLAST algorithm", a Computer Engineering Seminar, Yale University, USA, 21st September 2007.
- "Reconfigurable Architectures for the BLAST algorithm", Virginia Tech, USA, 18th September 2007.

Program Committees Participation

- 9th IEEE International Workshop on High Performance Computational Biology (HiCOMB) Atlanta, USA 19 April, 2010.
- Parallel Bio-Computing Workshop (PBC) Wroclaw (Breslau), Poland, September 13–16, 2009.

Industrial Collaborations

Collaboration agreement at February 7th 2008 with Synective Labs AB, Gotherburg, Sweden, for "……*jointly developing of a demonstration unit for the BLAST DNA matching algorithm*……".

Theses

Diploma theses related to this work with C. Kozanitis, P. Afratis, C. Galanakis, D. Vasilopoulos, S. Makropoulos and G. Vastarouhas.

Appendix B – Relative Work

Other publications

Conference papers

11. Nafsika Chrysanthou, Grigorios Chrysos, **Euripides Sotiriades**, Ioannis Papaefstathiou, "Parallel Accelerators For GlimmerHMM Bioinformatics Algorithm", Accepted as regular paper at International Conference Design, Automation and Test in Europe(Date 2011), Grenoble, France, 2011.
12. Nikolaos Alachiotis, **Euripides Sotiriades**, Apostolos Dollas, and Alexandros Stamatakis "Exploring FPGAs for Accelerating the Phylogenetic Likelihood Function" at the 8th IEEE International Workshop on High Performance Computational Biology (HiCOMB2009), Rome Italy, May 25th 2009.
13. Panagiotis Christou, Konstantinos Kyriakoulakos, **Euripides Sotiriadis**, Konstantinos Papadopoulos, Georgios-Grigorios Mplemenos and Ioannis Papaefstathiou, "Low-Power Security Modules optimized for WSNs", 16th International Workshop on Systems, Signals and Image Processing (IWSSIP), Chalkida Greece, 2009.
14. Ioannis Sotiropoulos, **Evripidis Sotiriadis**, Nikolaos Zervos, Ioannis Papaefstathiou "A high-end Binary Search unit for SVM DNA Micro-Arrays", to be presented at Innovations in Information Technology (Innovations'08), Al-Ain United Arab Emirates, December 16-18 2008.
15. Apostolos Dollas, Kyprianos Papademetriou, **Euripides Sotiriades**, Dimitrios Theodoropoulos, Iosif Koidis, George Vernardos, «A Case Study on Rapid Prototyping of Hardware Systems: The Effect of CAD Tool Capabilities, Design Flows, and Design Styles», Proceedings, 15th International IEEE Workshop on Rapid System Prototyping RSP-2004, pp, 180-186, Geneva, Switzerland, June 28-30, 2004, Computer Society Press.
16. Apostolos Dollas, Dionisios Pnevmatikatos, Nikolaos Aslanides, **Euripides Sotiriades**, Stamatis Kavvadias, Sotirios Zogopoulos, "Experimental Testing of PLATO, a Reconfigurable Active ATM Network Node", In Proceedings of the 8th Panhelleic Informatics Conference, pp, 11-17, Nicosia, Cyprus, November, 2001.
17. Apostolos Dollas, Dionisios Pnevmatikatos, Nikolaos Aslanides, Stamatis Kavvadias, **Euripides Sotiriades**, Kyprianos Papademetriou, «Rapid Prototyping of Reusable 4x4 Active ATM Switch Core with the PCI Pamette,» Proceedings, 12th International IEEE Workshop on Rapid System Prototyping RSP-2001, pp, 17-23, June 25-27, 2001, Monterey, CA, Computer Society Press.
18. Apostolos Dollas, Dionisios Pnevmatikatos Nikolaos Aslanides, Stamatis Kavvadias, **Euripides Sotiriades**, Sotirios Zogopoulos, Kyprianos Papademetriou, Nikolaos Chrysos, Konstantinos Harteros, Emmanouil Antonidakis, Nikolaos Petrakis, «Architecture and Applications of PLATO, a

Reconfigurable Active Network Platform,» Preliminary Proceedings, 9th International IEEE Symposium on Field-Programmable Custom Computing Machines, Rohnert Park, CA, April 30 – May 2, 2001, Computer Society Press.

19. **Euripides Sotiriades**, Apostolos Dollas, and Peter Athanas "Hardware-Software Codesign and Parallel Implementation of a Golomb Ruler Derivation Engine" Proceedings 8th International IEEE Symposium on Field-Programmable Custom Computing Machines, pp, 227-235, Napa Valley, April 17-19, 2000.
20. Apostolos Dollas, **Euripides Sotiriades**, Apostolos Emmanouilides "Architecture and Design of GE1, a FCCM for Golomb Ruler Derivation», Proceedings, 6th International IEEE Symposium on FPGA's for Custom Computing Machines, pp, 48-56, Napa Valley, April 15-17, 1998.

Short or Poster Conference Papers

21. Matina Lakka, Athina Desarti, Grigorios Chrysos, **Euripides Sotiriades**, Ioannis Papaefstathiou, Apostolos Dollas, "Reconfigurable Computing IP Cores for Multiple Sequence Alignment", Accepted as short paper for International Conference on Bioinformatics Models, Methods and Algorithms (Bioinformatics 2011),2011.
22. Pavlos Malakonakis, **Euripides Sotiriades**, Apostolos Dollas "GE3: a single chip client-server architecture for Golomb ruler derivation", Presented as short paper at The 2010 International Conference on Field-Programmable Technology (FPT'10),
23. Miltiadis Smerdis, Panagiotis Dagritzikos, Grigorios Chrysos, **Euripides Sotiriades**, Apostolos Dollas, "Reconfigurable Systems for the Zuker and Predator Algorithms for Secondary Structure Prediction of Genetic Data", Short paper at International Conference on Field Programmable Logic and Application (FPL 2010), pp 448-451, August 31 – September 2 2010.
24. Grigorios Chrysos, **Euripides Sotiriades**, Ioannis Papaefstathiou and Apostolos Dollas, "An FPGA based Coprocessor for Gene finding using Interpolated Markov Model (IMM)", Short paper at International Conference on Field Programmable Logic and Application (FPL 2009), Prague, Czech Republic, August 31 – September 2 2009.
25. Pavlos Malakonakis, Miltiadis Smerdis, **Euripides Sotiriades,** Apostolos Dollas "An FPGA-Based Sudoku Solver based on Simulated Annealing Methods" The 2009 International Conference on Field-Programmable Technology (FPT'09) , Sidney Australia, December 9-11 2009, (3rd place in the contest).
26. Grigorios Chrysos, **Euripides Sotiriades**, Ioannis Papaefstathiou and Apostolos Dollas, "A FPGA based Coprocessor for Gene finding using Interpolated Markov Model (IMM)", Short paper at International Conference on Field Programmable Logic and Application (FPL 2009), Prague, Czech Republic, 2009.
27. Nikolaos Alachiotis, Alexandros Stamatakis, **Euripides Sotiriades**, Apostolos Dollas, "A Reconfigurable Architecture for the Phylogenetic Likelihood Function" Short paper at International Conference on Field Programmable Logic and Application (FPL 2009), Prague, Czech Republic, 2009.

28. Georgios-Grigorios Mplemenos, Konstantinos Papadopoulos, Andreas Brokalakis, Grigorios Chrysos, **Euripides Sotiriadis,** Ioannis Papaefstathiou, "RESENSE: Reconfigurable WSN Nodes", Wireless Sensing Showcase, July 2009, London, (3rd place in the contest).
29. **Euripides Sotiriades**, Grigorios Chrysos, Georgios-Grigorios Mplemenos, Panagiotis Afratis, Nikolaos Alachiotis, Panagiotis Dagritzikos, Constantinos Galanakis, Christos Kagiavas, Miltiadis Smerdis, Ioannis Papaefstathiou, Dionisios Pnevmatikatos, Apostolos Dollas "Special purpose processors for performance boosting of Bioinformatics Algorithms", 3rd Conference of the Hellenic Society for Computational Biology and Bioinformatics (CHSCBB 2008), p 42, October 30-31, 2008, CERTH Thessaloniki.
30. Apostolos Dollas, Kyprianos Papademetriou, Stamatis Sotiropoulos, **Euripides Sotiriades**, "A Device for the Communication with the Environment for Persons with Kinetic Disabilities", Proceedings (in Greek), 7th Panhellenic Conference on Medicine and Rehabilitation, pp, 82-83, Chania, Greece, October 25-27, 2003.
31. Apostolos Dollas, **Euripides Sotiriades**, Apostolos Emmanouelides, Lee House, "General Purpose vs, Custom FCCM's: a Comparison of Splash 2, Quickturn RPM, and GE1 for Golomb Ruler Derivation", Proceedings, 6th International IEEE Symposium on FPGA's for Custom Computing Machines, pp, 269-270, Napa Valley, April, 1998.

Invited Lectures

"Reconfigurable Architecture for Golomb Ruler Derivation", Virginia Tech, USA, 6th September 2007.

Theses

Diploma theses relevant to this work were completed by N. Alachiotis, N. Chrysanthou, P. Dagritzikos, A. Desarti, M. Lakka, P. Malakonakis, and M. Smerdis.