ROACH Accelerated BLAST

by

David Macleod

Submitted to the Department of Electrical Engineering,

in fulfilment of the requirements for the degree of

Master of Science in Engineering

at the

University of Cape Town

July 2011

Supervisor: Prof. M.R. Inggs

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. The dissertation is being submitted for the degree of Master of Science in Engineering at the University of Cape Town. The dissertation has not been submitted before for any degree or examination in any other university.

Signature of Author.... Signed by candidate .....

Cape Town

26 July 2011

Abstract

Reconfigurable computing, in recent years, has been taking great strides in becoming part of mainstream computing largely due to the rapid growth in the size of FPGAs and their ability to adapt to certain complex applications efficiently. This dissertation investigates the reuse of application specific hardware developed for radio astronomy in accelerating a popular bioinformatics algorithm.

To showcase the abilities of reconfigurable computing the BLASTN sequence similarity search algorithm for DNA was selected and implemented on a ROACH reconfigurable computer with a Xilinx SX95T FPGA. The implementation divides the work between an x86 computer and the FPGA with communication taking place over 10GbE. The FPGA contains processing elements to perform the seed detection and extension portions of the algorithm while trying to keep the matches as close as possible to the original BLAST.

The results show that NCBI BLAST's runtime is highly dependent on the word size and the length of the query, while ROACH BLAST's runtime is largely unaffected by varying these parameters. This creates conditions where ROACH BLAST outperforms NCBI BLAST and others where NCBI BLAST outperforms ROACH BLAST. With a query length of 361 letters and a word size of 4 ROACH BLAST is 7x faster, however with a query length of 8 letters and a word size of 31 NCBI BLAST is 6x faster when compared one to one, in a production solution multiple queries would be loaded into the FPGA simultaneously.

BLAST's heuristic nature has a clear negative impact on the design since it is unable to take effective advantage of the massive reduction in search space the algorithm provides. However provided the query is long enough or the sensitivity high enough ROACH BLAST will overtake NCBI BLAST demonstrating the huge processing power of the FPGA. The scalability of the FPGA design shows promise with estimated easy scalability to query lengths of 30000 letters across multiple boards, with only minor changes required to support longer queries.

Acknowledgements

I would like to express my gratitude to the following people and organisations who assisted me during this dissertation.

Prof. Michael Inggs for his guidance, supervision and assistance in arranging my studentship at the CSIR.

Prof. Nicola Mulder for her guidance and supervision and for allowing me access to the CBIO group.

The Centre for High Performance Computing and CSIR for awarding me a studentship to fund my studies and for the excellent facilities they provided for me to work with.

Nick Thorne for his assistance with proofreading the dissertation.

Dr. Jeff Chen for his advice and assistance in integrating me into the CSIR system.

The team in the ACE Lab at the CHPC for helping me become familiarised with: the tools in the lab, the CSIR system and for general assistance, specifically, Nick Thorne, Jane Hewitson, Jason Salkinder and Andrew Woods.

UCT's CBIO group for providing me with: data, test sets, guidance on bioinformatics and advice on my dissertation, specifically, Prof. Nicola Mulder, Gerrit Botha and Ayton Meintjes.

This dissertation was funded through a studentship granted by the Human Capital Development Programme of the CHPC, an initiative of the Department of Science and Technology (DST) and managed by the Meraka Institute of the Council for Scientific and Industrial Research (CSIR).

Contents

Declaration i

Abstract iii

Acknowledgements v

List of Symbols xix

Nomenclature xxi

1 Introduction 1

1.1 Background 1

1.2 Scope & Objectives 3

1.3 Dissertation Overview 5

2 Literature Review 11

2.1 Sequence Alignment 12

2.1.1 Smith-Waterman 14

2.1.2 BLAST 15

2.2 Reconfigurable Computing 24

2.2.1 Parallel Computing 25

2.2.2 ROACH 26
2.2.3 BORPH 27
2.3 Conclusion 28

3 Project Environment **31**
3.1 Hardware 31
3.1.1 ROACH 32
3.1.2 Workstation 32
3.1.3 Network Infrastructure 33
3.2 Software 34
3.2.1 ROACH Tool Chain 34
3.2.2 Xilinx ISE Suite 35
3.2.3 Core Extraction 35
3.3 Compilation & Execution 37
3.4 Conclusion 37

4 Architecture **39**
4.1 Considered Designs 40
4.1.1 Global Extension Buffer 40
4.1.2 In Seed Detection Buffered Seeds 41
4.1.3 Large Systolic Array 41
4.1.4 In Array Extension 42
4.1.5 Multi-Level Buffering 44
4.1.6 TCP Networking 45

4.1.7 10Gbe with Push UDP Control 45
4.2 Hardware 46
4.2.1 Decoder 47
4.2.2 Extension Controller 49
4.2.3 Detection Regions 50
4.2.4 Local Controller 51
4.2.5 Seed Detection 51
4.2.6 Arbitrator 56
4.2.7 Extension Units 59
4.2.8 Aggregator 61
4.2.9 10GbE Control 62
4.2.10 10GbE Core 64
4.3 Software 66
4.3.1 Query & Database Input 66
4.3.2 Network I/O 67
4.3.3 Decoding & Expect Filtering 67
4.3.4 Report Generation 68
4.4 Conclusion 68

5 Design Verification **71**

5.1 Simulations 72
5.1.1 C++ Concept Simulations 72
5.1.2 Behavioural Simulation 72
5.1.3 Post-route Simulation 73

5.2 Compilation Strategies 74
5.3 Chip Usage 75
5.3.1 SX95T 75
5.3.2 LX110T 76
5.3.3 Large Virtex 5 FPGAs 76
5.4 Output Verification 77
5.4.1 Targeted Tests 78
5.4.2 Real Data Tests 78
5.5 Conclusion 80

6 Benchmarking **83**

6.1 Methodology 83
6.1.1 Query Selection 84
6.1.2 Database Selection 84
6.1.3 Test Setup 86
6.2 Results 88
6.2.1 Word Size 89
6.2.2 Query Length 89
6.2.3 Similarity 89
6.2.4 Break-even 89
6.2.5 Performance-per-watt 92
6.3 Observations 92
6.3.1 Impact of Word Size 92
6.3.2 Impact of Query Length 95

6.3.3 Query/Database Similarity 98
6.3.4 Impact of the DB length 98
6.3.5 ROACH & NCBI BLAST Break-even 100
6.3.6 Extension Unit Saturation 102
6.3.7 Network Saturation 102
6.3.8 Performance-per-watt 102
6.4 Conclusion 103

7 Conclusions **105**

7.1 Future Work 107
7.1.1 Simultaneous Query Support 107
7.1.2 Hardware Expect Filtering 107
7.1.3 NCBI BLAST Integration 108
7.1.4 Multiple Clock Domains 108
7.1.5 Network Control 109
7.1.6 Larger FPGAs & Cluster 109
7.1.7 BLASTP 110

A ROACH BLAST Instruction Set **111**

B Data Pack **113**

Bibliography **115**

List of Figures

1.1 The structure of the ROACH BLAST core. 7

1.2 Alignments taken from ROACH and NCBI BLAST reports. 8

1.3 ROACH vs NCBI BLAST runtime. 9

2.1 Global and local alignments of the same query and subject DNA. 12

2.2 A dot-plot of a DNA sequence against itself. 14

2.3 Speedup of mpiBLAST over NCBI BLAST. 19

2.4 GPU-BLAST's speed-up relative to a single-threaded CPU as a function of CPU threads. 20

2.5 An illustration of the different approaches for ungapped extension taken by NCBI and Mercury BLAST. 21

2.6 A high level illustration of the operation of RC-BLAST. 22

2.7 The high level architecture of a systolic array based BLAST implementation. 24

2.8 A block diagram of the ROACH architecture. 27

2.9 The BORPH architecture showing the hardware and software domains. . 28

3.1 ROACH v1.0 reconfigurable computer with a Xilinx LX110T FPGA. . . 33

3.2 Fujitsu XG700 CX4 10GbE Switch. 34

4.1 Global database buffer concept. 41

4.2 The basic components of a Seed Detection Element with a 10 line buffer. 42

4.3 A grid systolic array concept for BLAST's seed detection. 43

4.4 A significantly more complex systolic array that implements seed detection and extension. 43

4.5 Multiple levels of buffering to allow any Extension Unit to service a seed produced by any Detection Region. 44

4.6 Top level of the ROACH BLAST architecture. 46

4.7 The structure of the BLAST core. 47

4.8 A flow chart illustrating the operation of the Decoder. 48

4.9 A flow chart for the Extension Controller. 49

4.10 The structure of a Detection Region. 50

4.11 The Local Controller's flow diagram. 52

4.12 A flow chart of the operation of the seed detection process. 53

4.13 The operation of the Seed Detection Array. 54

4.14 An illustration of the operation of the Reference Element. 55

4.15 The priority encoder flow chart. 57

4.16 Extension Unit work allocator flow chart. 58

4.17 A flow chart illustrating the extension process a seed undergoes when allocated to an Extension Unit. 60

4.18 A flow chart illustrating the operation of an Aggregator. 62

4.19 The "pull" network control and buffering. 63

4.20 The output UDP packet payload structure produced by ROACH BLAST. 63

4.21 The network output logic flow chart. 64

4.22 The MATLAB/SimuLink design file for the ROACH BLAST project. . . 65

5.1 A behavioural simulation waveform produced by ISIM. 73

5.2 Alignments taken from ROACH and NCBI BLAST reports. 79

6.1 Test 1, the effect of varying word size on runtime. 90

6.2 Test 2, the effect of query length on runtime. 91

6.3 Test 3, the effect of query/database similarity on runtime. 91

6.4 Test 4, the projected break-even of ROACH and NCBI BLAST. 93

6.5 Test 5, Figure 6.1 power normalised. 94

6.6 The runtime with word size 6 for each of the queries in the test suite arranged in groups of length sorted by similarity. 96

6.7 The runtime with word size 6 for each of the queries in the test suite sorted by length. 97

6.8 The runtime with word size 6 for each of the queries in the test suite sorted by similarity. 99

6.9 Log graphs of the expected word sizes and lengths where ROACH and NCBI BLAST's runtime will be equal. 101

List of Tables

1.1 Xilinx Virtex 5 FPGAs discussed in this dissertation. 6

2.1 The BLOSUM62 scoring matrix showing the amino acids and their associated substitution scores. 13

2.2 The BLAST programs that form the BLAST suite, the query and database they operate on and their function. 16

3.1 The significant differences between the Xilinx SX95T and LX110T FPGAs. 32

5.1 Changes to compilation settings from default. 74

5.2 Chip utilization of the SX95T implementation of ROACH BLAST. . . . 75

5.3 Chip utilization of the LX110T implementation of ROACH BLAST. . . . 76

5.4 Chip utilization of the SX240T and LX330T implementations of ROACH BLAST. 77

6.1 Test queries similarity and length groupings. 85

6.2 The actual and expected runtimes of ROACH BLAST operating on the large database. 100

List of Symbols

w — Word size
s — Detection significance threshold
x — Extension termination threshold
E — Expect
K — Minor constant for the scoring system and database length
λ — Major constant for the scoring system and database length
S — Raw score
S' — Bit score
n — Length of the query in letters
m — Length of the database in letters

Nomenclature

ADC Analogue to Digital Converter

AES Advanced Encryption Standard, a popular encryption algorithm.

ARM A RISC microprocessor architecture and instruction set.

ASIC Application Specific Integrated Circuit

BEE2 Berkeley Emulation Engine 2

BLAST Basic Local Alignment Search Tool, a sequence similarity search tool for DNA and proteins.

BORPH Berkeley Operating system for ReProgrammable Hardware

BRAM Block RAM, dedicated memory structures inside an FPGA.

CASPER Center for Astronomy Signal Processing and Electronics Research

CLB Configurable Logic Block, a unit of configurable logic in an FPGA.

CPLD Complex Programmable Logic Device, a reconfigurable logic device typically smaller than FPGAs.

CPU Central Processing Unit, the main processing unit in a computer, typically used in reference to a von Neumann type processor.

CUDA Compute Unified Device Architecture, a parallel computing architecture developed by nVidia for use with their GPUs.

DAC Digital to Analogue Converter

DCM Digital Clock Manager, a component that manipulates clock signals.

DDR2 Double Data Rate 2, memory capable of performing a read or a write on the falling and rising edge of the clock.

DIMM Dual In-line Memory Module, a memory module with contacts on both sides of the board.

DNA Deoxyribonucleic Acid

DSP Digital Signal Processing

DUST A tool to remove regions of low significance from DNA or protein sequences.

ECC Error Correcting Code

FASTA A sequence alignment program for DNA and proteins.

FIFO First In First Out, a type of memory where data is read out in the same order that it was written in.

FPGA Field Programmable Gate Array, a reconfigurable logic integrated circuit device.

FTP File Transfer Protocol, a protocol to transfer files over a TCP network.

GbE Gigabit Ethernet

GI A number NCBI uses to uniquely identify a sequence record.

GPGPU General Purpose Graphic Processing Unit, a GPU being used in general purpose computing.

GPU Graphics Processing Unit, a device designed for processing 2D and 3D graphics.

HDL Hardware Description Language, a programming language used to describe logic.

HPC High Performance Computing

IBOB Interconnect Break-out Board

ILP Instruction Level Parallelism, mechanisms used to extract parallelism at the instruction level, e.g. pipelining instructions.

KAT Karoo Array Telescope, now named MeerKAT, a radio telescope with 64 antennas in the Karoo.

KATCP Karoo Array Telescope Control Protocol, facilitates the management of the ROACH hardware over a network.

NCBI National Center for Biotechnology Information

NFS Network File System, a protocol allowing users access to a file system over a network.

OPB On-chip Peripheral Bus

PCI-X Peripheral Component Interconnect Extended, an extended version of the PCI bus providing additional bandwidth.

PLL Phase Lock Loop

PSU Power Supply Unit

QDR Quadruple Data Rate, memory capable of performing a read and a write on the falling and rising edge of the clock.

RAM Random Access Memory

ROACH Reconfigurable Open Architecture Hardware, a reconfigurable computer designed for radio astronomy.

SDK Software Development Kit, a set of tools to develop software for a technology.

SKA Square Kilometre Array, a radio telescope project South Africa is bidding to site.

SMP Symmetric Multi-Processor, a type of computer where multiple CPUs share a common memory.

SRAM Static RAM, a type volatile memory that does not need to be refreshed.

SSH Secure Shell, a protocol facilitating remote access to Unix computers.

TCP Transmission Control Protocol, a transport layer protocol belonging to the IP suite.

UDP User Datagram Protocol, a transport layer protocol belonging to the IP suite.

VHDL Very High Speed Integrated Circuit Hardware Description Language, a HDL language.

Chapter 1

Introduction

This dissertation investigates the use of reconfigurable computing in accelerating the BLASTN, nucleotide BLAST, sequence similarity search algorithm. An FPGA implementation of the BLASTN algorithm was developed and its performance was analysed with respect to an equivalent modern software implementation. The introduction begins with a brief background on sequence similarity searches and reconfigurable computing, followed by the projects objectives and ends with an overview of the remaining thesis.

1.1 Background

Due to breakthroughs in DNA sequencing techniques in the 1970s microbiologists now have large databases of sequenced DNA that have not been fully explored. For microbiologists to learn more about a DNA sequence they put it through a process called sequence analysis, which can take many forms depending on what the target of the analysis is. One of the processes applied during sequence analysis is a sequence similarity search which is used to identify the query's relationship to other sequences, ultimately leading to insight into its purpose [45].

One of the earliest and most widely used sequence alignment algorithms to be developed was Smith-Waterman, which is based on the Needleman-Wunsch global alignment algorithm [35]. The Smith-Waterman algorithm performs an exhaustive search to ensure that the best local alignment is found for a given scoring system, however this accuracy has the penalty of long runtimes even when the likelihood of finding a better match is low [19].

Due to this behaviour a heuristic for Smith-Waterman ,BLAST (Basic Local Alignment Search Tool), was developed that employs thresholds at various points in the algorithm giving the microbiologist the ability to manipulate the algorithm to look at areas where significant matches are more likely to be detected [2]. This ability to reduce the search space without having a major impact on the accuracy [10] of the alignments lead to a significant speed-up over Smith-Waterman [19]. Since its conception in 1990 the popularity of BLAST has grown to the point where it is now one of the industry standard similarity search algorithms [10].

Even with the significant speed-up that BLAST provides it remains a bottleneck in the sequence analysis process so work has continued on further accelerating the algorithm. This has lead to a large variety of BLAST implementations developed for conventional CPUs, GPGPUs and FPGAs, each tweaking the algorithm to improve runtime or accuracy.

Until recently FPGAs have been too small to accommodate complete BLAST implementations that are fully parametrisable and true to the original BLAST specification. Homology of results produced by different implementations of the BLAST algorithm are important to microbiologists so they can make direct comparison between results generated from different systems [45]. Due to this requirement of compatibility of results all implementations of the BLAST algorithm must produce matches with a high degree of correlation in at least some conditions to be useful.

One method of reducing the runtime without compromising the BLAST algorithm is to design a custom processing unit that is capable of efficiently processing the data. Reconfigurable computing provides the ideal platform for prototyping such designs at low cost [27].

A reconfigurable computer is a computer that is able to make changes to its control and datapath during runtime [18]. This means that the internal logic the chip consists of can have its connections reordered to create unique logic tailored to a specific application. Early reconfigurable computers consisted of blocks of fixed hardware where the processing was performed connected via reconfigurable logic [3]. This "glue logic" allowed the flow of data through the system to be manipulated but did not provide the designer the flexibility to change the logic where the work was done.

As the size and speed of reconfigurable logic, such as CPLDs and FPGAs, improved they were able to take over more of the processing roles in reconfigurable computers and move away from being mere "glue logic" [28]. Today reconfigurable computers typically contain a fixed logic von Neumann processor to control the board and run the operating system,

with one or more large FPGAs connected to the system bus [40]. In these systems the fixed logic does not perform a large portion of the computation, but rather facilitates easy access to the reconfigurable devices connected to its bus.

Despite reconfigurable computer's logic having flexibility near that of software we have still not seen much penetration into commodity hardware, and thus their role is typically relegated to prototyping platforms [34]. There are a number of reasons for this poor performance in the commodity sector; firstly, FPGAs only became big enough to be generally useful for large scale processing in the past few years and these were the large upper range of the chips. These chips, when compared to the current state-of-the-art fixed logic chips, were significantly more expensive.

Possibly the biggest barrier to the wide adoption of the FPGA remains it accessibility in terms of designing systems for the device. While at a glance designing systems for FPGAs may look similar to software development the reality is that special skills and knowledge, that most software designers do not have, is required [23, 38].

To bridge the gap between hardware and software design many companies have created tool chains to abstract away from the native HDL languages of these devices. Many of these abstractions take the form of C like languages or extensions to the MATLAB/ SimuLink environment.

With the abilities of FPGAs now clear, and mature tool chains to support them there has been a move toward more closely integrating popular von Neumann architectures like x86 and ARM onto the same die as an FPGA. Both of the major FPGA manufactures have announced that their next generation architectures will contain one of those popular architectures paving the way for the FPGA to enter the commodity market [1, 20].

This dissertation seeks to build a scalable BLASTN solution for the ROACH reconfigurable computer demonstrating the reconfigurable computer's flexibility by reusing hardware designed for radio astronomy in bioinformatics and showcasing its performance over software.

1.2 Scope & Objectives

The objective of this dissertation is to design an FPGA based version of the BLASTN algorithm and determine its suitability for implementation on a reconfigurable computer.

The implementation must be compatible with the original BLAST specification and allow the user full control over all parameters. The specific objectives are:

1. Perform a literature review to determine the state various BLASTN implementations.
2. Design and implement an FPGA based version of BLASTN for deployment on the ROACH reconfigurable computer.
3. Analyse the performance of the ROACH implementation compared to a current state-of-the-art software implementation.

The first objective allows for the exploration of various BLAST implementations to assist in determining critical portions of the algorithm, acceptable modifications and deviations from the original, and the identification of successful concepts for incorporation into the design. Due to the heuristic nature of the algorithm there are many implementations across CPUs, GPGPUs an FPGAs to draw inspiration from.

The second objective draws on the knowledge obtained in the literature review to design a new version of BLAST that overcomes the limitations of the identified reconfigurable computer solutions. The output from this design must be true to the original BLAST specification and the user must be able to manipulate the input parameters to the same effect. Once familiarity with the ROACH development tools is gained a ROACH specific version can be implemented. The implementation must be verified and tested for reliability and once this is achieved performance testing can commence.

The performance of the system must be analysed with real data representing a variety of situations so that trends can be established for both the reconfigurable computer and software versions. The software version that will form the baseline for the comparison of performance will be NCBI BLAST, which is the direct successor of the original BLAST but does deviate from the original specification slightly. Unfortunately there is no modern implementation of the original BLAST algorithm, however NCBI BLAST can be configured to produce similar output but is expected to have superior performance.

This dissertation will only investigate the BLASTN algorithm from the BLAST suite. Other algorithms in the suite do have a similar structure to that of BLASTN but this work will not investigate their performance or suitability for reconfigurable computers.

The final design in this dissertation will only be implemented on a single reconfigurable computer, however one of the design goals is to ensure that it is scalable both within

larger reconfigurable computers and clusters of reconfigurable computers. The anticipated performance of clusters of reconfigurable computers will be investigated.

1.3 Dissertation Overview

The dissertation continues as follows:

Chapter 2 provides a literature review of sequence alignment and reconfigurable computing. The sequence alignment section discusses sequence alignment in general and introduces the Smith-Waterman and BLAST algorithms. Smith-Waterman is dynamic programming algorithm that performs local alignment on DNA and proteins [35]. Smith-Waterman is guaranteed to find optimal alignments but is slow and requires large amounts of memory [45, 19]. As a result of Smith-Waterman's negative characteristics heuristic algorithms have been developed, of which BLAST is one.

BLAST is one of the most widely used local alignment search tools due to its high speed and good accuracy, but similarity searches are still a bottleneck in the sequence analysis pipeline [10]. Due to the need for faster BLAST implementations researchers have developed various designs which tweak the algorithm or implement it on alternate hardware architectures. An NCBI BLAST accurate MPI version has been developed with superlinear scaling to tens of thousands of cores [8]. Various GPU implementations have also been designed, some sacrifice speed for correlation with NCBI BLAST and others that prioritise speed [26, 42]. FPGA implementations have also been developed using: bloom filters [25], lookup tables [29] and systolic arrays [44] with some of the designs conforming to NCBI BLAST and others not. In all cases only part of the algorithm is implemented on the FPGA.

Background on the development of FPGAs is discussed starting with Estrin's *Fixed Plus Variable Structure Computer* designed in 1960 [9]. The need for more application specific hardware and how FPGAs can help is covered by a brief review of the technical report *The Landscape of Parallel Computing Research: A View from Berkeley* which illustrates the diminishing returns of traditional microprocessor design [11]. The chapter ends with a description of the basic structure of an FPGA and details the ROACH reconfigurable computer and its operating system BORPH.

Chapter 3 introduces the project development and test environment. ROACH BLAST was designed for implementation on the ROACH board with a Virtex 5 SX95T FPGA

however a design for LX110T version of the ROACH board was also developed. The ROACH board is connected to an x86 workstation via 1GbE and 10GbE. The 1GbE connection provides access to the ROACH boards PowerPC subsystem and BORPH operating system while 10GbE provides high bandwidth low latency access for transferring data directly into the FPGA.

Table 1.1 – Xilinx Virtex 5 FPGAs discussed in this dissertation.

Virtex 5	**SX95T**	**LX110T**	**SX240T**	**LX330T**
Slices	14 720	17 280	37 440	51 840
BRAM (KB)	8 784	5 328	18 576	11 664
DSP48E	640	64	1 056	192

The workstation used during testing is a quad core 3GHz Intel Xeon 5450 with 16GB of DDR2 667 fully buffered ECC RAM. The workstation is running Ubuntu 10.04 64bit and acts as the head node for the ROACH cluster. The ROACH boards network boot through the 1GbE control network from kernel and file system images stored on the head node.

An overview of the tools used in the dissertation is also covered in the chapter. The Matlab/SIMULINK suite is used to design the top level infrastructure for the ROACH board and implement the 10GbE core. From the Xilinx ISE suite System Generator is used to extract and modify the top level netlist produced by the Matlab/SIMULINK suite so that it can be interfaced with VHDL code written in ISE. ISE is used to code the BLAST implementation and generate a binary configuration file for the FPGA.

The binary configuration file is passed back to the ROACH tool chain to be converted into a bof file which can be executed on the ROACH board. The bof file is placed on the workstation in the /boffiles/ directory in the shared file system which the ROACH boards boot off and it is executed through the KATCP system from the workstation. The workstation is also used for benchmarking NCBI BLAST against ROACH BLAST.

Chapter 4 discusses the considered and final architectures of the design developed in this dissertation. Various designs were considered each of which were eliminated due to foreseen problems with their implementation. The initial design was based on the implementation described in the literature review as the systolic array version. The downfall of this design is its lack of flexibility which this dissertation attempts to resolve. An initial design considered implementing a large systolic array that could handle shorter word sizes allowing the user some flexibility, however the design would have a fixed maximum length and require a large number of processing elements of which most would only rarely be used. Ultimately this design was considered too expensive in logic.

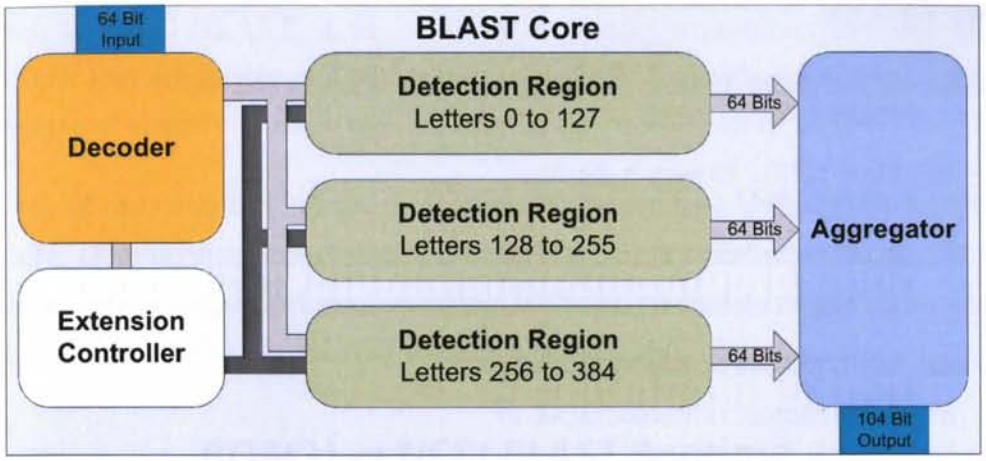


Figure 1.1 – The structure of the ROACH BLAST core, the implementation of ROACH BLAST on an SX95T FPGA contains 3 Detection Regions to process the alignments.

Another design considered involved implementing the first two stages of the BLAST algorithm in a systolic array instead of in two distinct sets of logic. Due to the large disparity in the amount of time spent in stages 1 and 2 and the additional logic required in each element to support both stages it was once again considered too expensive in logic, however this design would have been fully parametrisable and not have the restrictions of the other versions.

The selected design implements stage 1 in a systolic array capable of producing seeds of any length through a multi-element propagation system. Stage 2 is implemented in a similar fashion as described by the systolic array based BLAST in the literature review. The preprocessing and post-processing for the design is implemented on the Xeon workstation and communicates with the ROACH board through 10GbE. The software converts FASTA files into memory mapped binary files and streams the data to the ROACH board. The ROACH board returns indicies to alignments which the software decodes, analyses for significance and filters. The software produces a report similar to NCBI BLAST.

Chapter 5 covers the verification of the final design. This involves: the strategies used to fit the design on the FPGA, simulations, chip usage and output verification. A test design that utilised over 90% of the FPGA slices was used for testing compilation strategies to optimise device usage and speed. It was discovered that enabling global optimization for area had the largest impact producing both smaller and faster designs. Through these tests it was determined that the frequency could be raised by 15MHz over the frequency produced with the standard settings and realise a reduction in size by a few percent.

Due to these experiments a clock frequency of 60MHz was selected for the design with

ROACH BLAST

```
> SRR088923.667671 HWUSI-EAS1591_61H93:8:32:16432:6608 length=94
           Length = 94

 Score = 137 bits (312), Expect = 8e-31
 Identities = 83/87 (95%)

Query:  183 TGCCGAAAGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAACCAATGCCTGATGCGA 242
           ||||||| ||||||||||||||||||||||||||||||||||| ||||||||||||||||
Sbjct:    1 TGCCGAATGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAATCAATGCCTGATGCGA 60

Query:  243 CGCTGTTGCGTCTTATCAGGCCTACAA 269
           ||||||| ||| ||||||||||||||||
Sbjct:   61 CGCTGTCGCGGCTTATCAGGCCTACAA 87
```

NCBI BLAST

```
> SRR088923.667671 HWUSI-EAS1591_61H93:8:32:16432:6608 length=94
Length=94

 Score = 137 bits (312), Expect = 9e-31
 Identities = 83/87 (95%), Gaps = 0/87 (0%) Strand=Plus/Plus

Query 183 TGCCGAAAGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAACCAATGCCTGATGCGA 242
           ||||||| ||||||||||||||||||||||||||||||||||| ||||||||||||||||
Sbjct   1 TGCCGAATGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAATCAATGCCTGATGCGA 60

Query 243 CGCTGTTGCGTCTTATCAGGCCTACAA 269
           ||||||| ||| ||||||||||||||||
Sbjct  61 CGCTGTCGCGGCTTATCAGGCCTACAA 87
```

Figure 1.2 – Alignments taken from ROACH and NCBI BLAST reports. The search was performed against sequence GI Number: 1419508 at a word size of 6.

timing driven placement in reserve for additional logic potentially required after debugging and optimization. Simulations were conducted in software and through the ISIM HDL simulator to verify the behavioural and post-route models of the design. Behavioural simulation of the design proved capable of identifying logical errors while the slower but more thorough post-place and route simulations identified HDL synthesis issues and timing errors.

The chip usage discussion looks at two versions of Xilinx chips, the LX110T and SX95T. The SX95T is comparable to the LX110T except it has more fixed logic and fewer slices. The chapter ends with output verification involving pattern tests to ensure internal checks and control can handle the best and worst case scenarios. Exemplar data is fed through the system to verify the output against the original BLAST algorithm and finally real data is run and compared to NCBI BLAST. The results showed that the device was operating correctly and conformed with the BLAST standard.

Chapter 6 analyses the performance of the system across a variety of situations in

comparison to NCBI BLAST. A set of query sequences were selected to fall into three sets of length and similarity and compared to a 700MB database of short reads and a 7GB database of sequences with various lengths up to millions of nucleotides long.

The first set of tests ran the full set of 45 queries against both databases with word sizes from 4 to 31. The results showed ROACH BLAST has a constant runtime while NCBI BLAST has a exponential runtime increasing as the word size decreases. The break-even point for NCBI and ROACH BLAST was at a word size of 6.

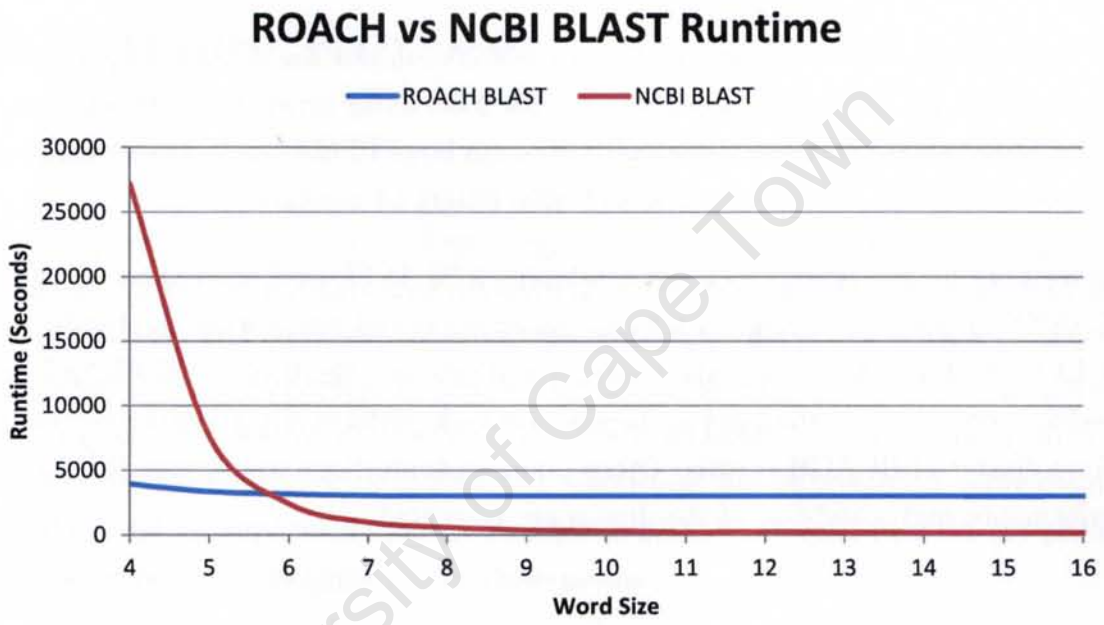


Figure 1.3 – ROACH vs NCBI BLAST runtime. All queries in the dataset used to construct this graph are below 383 letters long, the database subjects represent a large range of lengths. The graph shows ROACH BLAST performs with a near constant runtime while NCBI BLAST's runtime is exponentially effected by word size.

A similarity test was run by dividing the group into three sets of 15 sorted by similarity and run against both databases with a word size of 6. The results showed that similarity had little effect on NCBI and ROACH BLAST.

The third test analysed the impact of the length of the queries and was performed by dividing them into three sets of 15 sorted by length. The results showed that length has little impact on ROACH BLAST but has a linear relationship with NCBI BLAST where runtime increases with query length.

Based on these results a final test was run to determine the break-even point of NCBI and ROACH BLAST from the length of the query and word size. The results were based

on projections of how ROACH BLAST would perform with longer queries, and they showed that if the query is long enough ROACH BLAST will always be faster regardless of word size.

Chapter 7 draws conclusions from the results and observations and further discusses the use of reconfigurable computing in accelerating BLAST. The ROACH BLAST design is capable of outperforming NCBI BLAST with short query lengths when the word size is below 6, however since this sits on the steep portion of NCBI BLAST's exponential runtime curve NCBI BLAST rapidly catches up when the word size is increased.

For ROACH BLAST to always be faster an array supporting 400000 letters would be required, but for the default word size of 11 a 9000 letter array would break-even. The design shows great promise for scalability through large FPGAs and clusters with current technology capable of reaching arrays of 9000 letters on a single board.

Future work for the project includes developing a BLASTP implementation based on the BLASTN architecture designed for this dissertation and integrating the hardware into NCBI BLAST's code to unify the interface and produce identical NCBI BLAST output. Modifications are also proposed to increase network performance and operate the design over a cluster of ROACH boards. Other minor optimisations to improve the usefulness, performance and scalability of the design are presented.

Chapter 2

Literature Review

In this chapter a concise review of reconfigurable computing and sequence alignment is presented to help the reader understand the context of this dissertation. Reconfigurable computing and sequence alignment are both large fields and cannot be covered comprehensively in this chapter. Events leading up to the current state of reconfigurable computing and reasons for the need to accelerate sequence alignment along with an overview of the Smith-Waterman and BLAST algorithms will be presented. More detail is provided on various implementations of BLAST to provide a baseline for the comparison of the implementation designed in this dissertation.

Many computer engineering labs around the world have long predicted the demise of homogeneous computers with large monolithic von Neumann processors forming the base of computation. The need for less versatile processors suited to specific algorithms has become a field of much interest due to recent developments showing evidence of diminishing returns in: ILP, power usage and the ability to address large memories [11].

While the short comings of monolithic processors were becoming apparent GPUs were getting more powerful and their highly parallel, light weight processing element architecture was being better adapted for more general purpose use. As GPGPUs were adopted by the scientific community the power of accelerator cards and application specific hardware became clear [32].

With the scientific community more comfortable with heterogeneous computers there has been a push to develop novel architectures for accelerating some of the more time consuming algorithms. One area that is benefiting from this research is bioinformatics, and more specifically related to this dissertation is the acceleration of BLAST. BLAST

being as popular and widely used as it is has been garnished with a variety of systems designed to accelerate or improve its accuracy. These systems include implementations on: SMP machines, clusters, GPGPUs and reconfigurable computers.

2.1 Sequence Alignment

In general sequence alignment is the pairwise alignment of strings such that an optimal local or global alignment can be found. The optimal alignment is determined by its score which is calculated by applying weightings to various matches and manipulations of the compared strings [35]. Sequence alignment is useful in the analysis of natural language [41] but has its most widespread use in analysing DNA and proteins.

Microbiologists use sequence alignment to identify regions of similarity between compared sequences to assist them in determining the lineage and functionality of DNA and proteins [10]. For these alignments to be useful a degree of flexibility is required to allow for mutations that occurred during evolution to be compensated for [45]. If these subtle mutations cause alignments of related DNA or proteins to be missed important information can be lost. To account for these mutations sequence alignment algorithms allow gaps to be inserted or for substitutions to be made. A gap is a n length break in the alignment between the query and the subject, gaps are illustrated in Figure 2.1 by "-".

GLOBAL ALIGNMENT

```
AGGTCTGGCAGAT--------------------TGAAATCTTCGATGAGAAAGGGCGTTTGTGCTGTTC
||   |||.||||                    ||.|||||||            .||.|.|.||..||
AG---TGGGAGATAAAGAATTATTTACTAACAGATGGAATCTTC-----------AGTCTCTTCTTCTC
```

LOCAL ALIGNMENT

```
CTGGCAGATTGAAATCTTCGATGAGAAAGGGCGTTTGTGCTGTTC
||..|||| ||.|||||||            .||.|.|.||..||
CTAACAGA-TGGAATCTTC-----------AGTCTCTTCTTCTC
```

Figure 2.1 – Global and local alignments of the same query and subject DNA. A "-" represents a gap of arbitrary length, "." a mismatch and "|" a match.

Another method to compensate for slight differences between the query and subject is to allow for substitutions. With DNA alignments this is simple due to the alphabet size being limited to 4 letters with a uniform probability of occurrence. Performing a substitution involves replacing of a single letter in the query with another letter and scoring the modification. This process is of particular importance for protein alignments

since the probability of occurrence of each of the amino acids is not even and some substitutions are better than others [45]. Biologists have developed scoring tables to inform the sequence alignment algorithm as to which are good substitutions and to place emphasis on rarely occurring amino acids [10].

Table 2.1 – The BLOSUM62 scoring matrix showing the amino acids and their associated substitution scores.

C	**9**	-1	-1	-3	0	-3	-3	-3	-4	-3	-3	-3	-3	-1	-1	-1	-1	-2	-2	-2
S	-1	**4**	1	-1	1	0	1	0	0	0	-1	-1	0	-1	-2	-2	-2	-2	-2	-3
T	-1	1	**4**	1	0	-2	0	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-2	-2	-2
P	-3	-1	1	**7**	-1	-2	-2	-1	-1	-1	-2	-2	-1	-2	-3	-3	-2	-4	-3	-4
A	0	1	0	-1	**4**	0	-2	-2	-1	-1	-2	-1	-1	-1	-1	-1	0	-2	-2	-3
G	-3	0	-2	-2	0	**6**	0	-1	-2	-2	-2	-2	-2	-3	-4	-4	-3	-3	-3	-2
N	-3	1	0	-2	-2	0	**6**	1	0	0	1	0	0	-2	-3	-3	-3	-3	-2	-4
D	-3	0	-1	-1	-2	-1	1	**6**	2	0	-1	-2	-1	-3	-3	-4	-3	-3	-3	-4
E	-4	0	-1	-1	-1	-2	0	2	**5**	2	0	0	1	-2	-3	-3	-2	-3	-2	-3
Q	-3	0	-1	-1	-1	-2	0	0	2	**5**	0	1	1	0	-3	-2	-2	-3	-1	-2
H	-3	-1	0	-2	-2	-2	1	-1	0	0	**8**	0	-1	-2	-3	-3	-3	-1	2	-2
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	**5**	2	-1	-3	-2	-3	-3	-2	-3
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	**5**	-1	-3	-2	-2	-3	-2	-3
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**	1	2	1	0	-1	-1
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**	2	3	0	-1	-2
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**	1	0	-1	-2
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	**4**	-1	-1	-3
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**	3	1
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	**7**	2
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-2	-2	-3	1	2	**11**
	C	**S**	**T**	**P**	**A**	**G**	**N**	**D**	**E**	**Q**	**H**	**R**	**K**	**M**	**I**	**L**	**V**	**F**	**Y**	**W**

The process of sequence alignment can be graphically represented by creating a matrix with the query and subject on each axis and marking matching or high scoring intersecting letters[1]. Once sufficient intersections have been marked regions of similarity will be visible by viewing the diagonal lines [45]. Horizontal or vertical offsets between the diagonal lines of matches are gaps, Figure 2.2 illustrates this. Knowing where the similarity between the query and subject is, is only part of the problem, the best alignments need to be selected and this can be done by either looking for global or local alignments.

Global alignment seeks to find the optimal alignment across the compared query and subject resulting in only one alignment. This is not always useful to the microbiologist as they are often interested in regions of similarity, due to this requirement local alignments

[1]This matrix is called a dot-plot and is one of the simplest methods of comparing sequence similarity graphically [45].

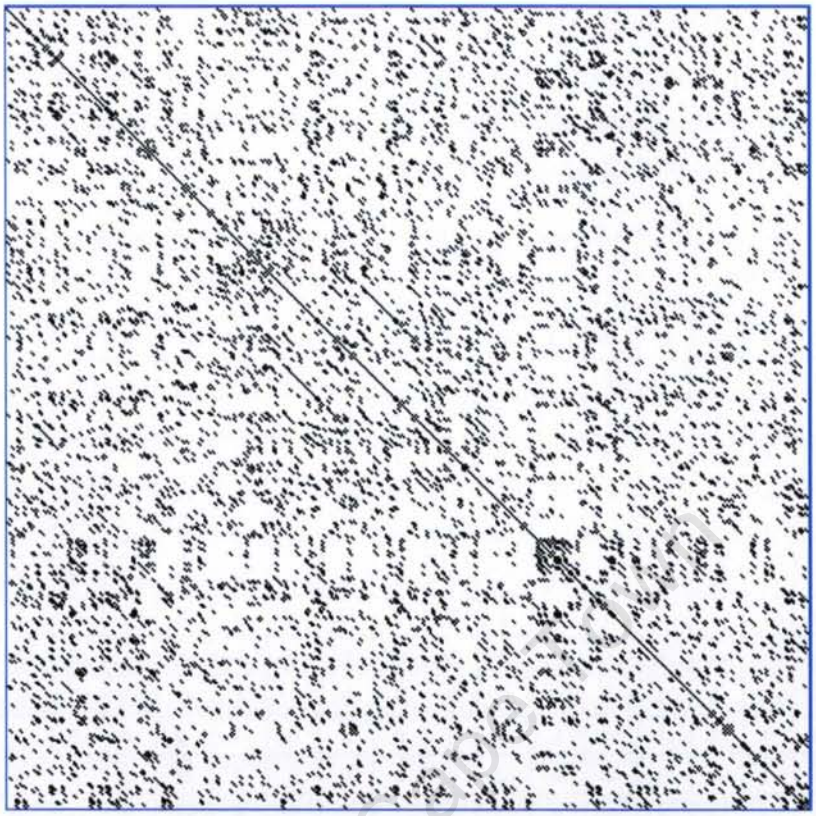


Figure 2.2 – A dot-plot of a DNA sequence against itself. The solid line down the main diagonal represents the perfect alignment of the DNA, the other dots are matching nucleotides in other portions of the DNA.

are often used instead. Local alignments allows for multiple alignments to be detected giving insight into specific regions of the compared DNA or proteins. Figure 2.1 shows a global and local alignment for the same sequences of DNA [10].

It is evident from Figure 2.2 that the search space increases exponentially with the length of the query and subject. This behaviour results in very long run times for long sequence alignments and has resulted in various heuristics, accelerators and highly efficient dynamic programming algorithms being developed.

2.1.1 Smith-Waterman

Smith-Waterman is a dynamic programming implementation for local alignment that was derived from the Needleman-Wunsch global alignment algorithm [35]. Possibly the most desirable aspect of Smith-Waterman is that it is guaranteed to find optimal local alignments [19], however this accuracy negatively impacts the runtime, and in early implementations was severely memory hungry to the point where it limited the length

of possible alignments. Later implementations have included optimizations to improve runtime and memory usage, but Smith-Waterman is still a lengthy process compared to its heuristic alternatives [45].

The basic implementation of the Smith-Waterman algorithm creates a n by m matrix, where n is the length of the query and m the length of the database, which stores the scores of the sub-sequence alignments [12]. As previously stated the size of the matrix grows rapidly with the length of the query and database which causes it to quickly becomes a limiting factor in memory, especially when aligning nucleotide sequences. To resolve this impasse intermediate calculations can be discarded from the matrix reducing its size to only 2 rows. This however will have a negative impact on the performance of the algorithm as additional calculations will be required to reconstruct discarded data when it is needed. This modification effectively alleviates the memory issues of Smith-Waterman and allows even modest computers to perform long alignments [45].

A minor modification that can be adopted to reduce the runtime of Smith-Waterman is to employ an *X-Drop* threshold. The *X-Drop* is the value the score is allowed to drop-off by from its maximum before the algorithm stops looking for a better alignment [45]. This effectively reduces the search space by masking out areas where there is a low probability that a better match will be found, but, when this threshold is used Smith-Waterman is no longer guaranteed to find optimal local alignments. Applying thresholds, such as the *X-Drop*, is a step toward developing heuristic algorithms for Smith-Waterman, of which the focus of this dissertation, BLAST, is one.

2.1.2 BLAST

In 1990 the BLAST algorithm was published as a "rapid sequence comparison" tool which is robust and "amenable to mathematical analysis" that can be adapted to different contexts [2]. Since BLAST's début it has grown in popularity and is now one of the most widely used database search suites. BLAST is well suited to finding very similar short segments and its speed makes it suitable to operate on large databases with multiple queries [45].

The BLAST programs provided by NCBI require the user to reformat the database so that it can be more efficiently handled during analysis. For nucleotide databases the formatter reads the data in from a file in the FASTA format and creates memory mapped binary files which store each letter in 2 bits [13]. The database formatting process can

Table 2.2 – The BLAST programs that form the BLAST suite, the query and database they operate on and their function. Adapted from Schneider [36].

Program	Comparison	Description
BLASTN	Nucleotide - Nucleotide	Compares a nucleotide query against a nucleotide database.
BLASTP	Protein - Protein	Compares an amino acid query against a protein database.
BLASTX	Nucleotide - Protein	Compares the six-frame translation of a nucleotide query against a protein database.
TBLASTN	Protein - Nucleotide	Compares a protein query against a nucleotide database dynamically translated in all six frames.
TBLASTX	Nucleotide - Nucleotide	Compares the six-frame translation of a nucleotide query against the six-frame dynamic translation of a nucleotide database.

have a long runtime with large databases but only needs to be performed once and dramatically improves the efficiency of the operation of the BLAST programs.

The first step in the BLAST algorithm is seeding. Seeding provides an opening window for the algorithm by specifying the number of letters required to start analysis. Every set of consecutive letters with word size w, w-mer, are extracted from the query and analysed for potential substitutions. The query words and their substitutions are scored and a neighbourhood is created. The words in the neighbourhood which have a score higher than the neighbourhood threshold T pass to the next stage [10].

All the words in the passing neighbourhood are compared to all positions in the subject, anywhere there is an exact match an extension begins. When analysing DNA sequences the neighbourhood calculation becomes significantly easier due to the equal probability of occurrence of the letters in the alphabet. Essentially the neighbourhood is all the extracted query words since there are no substitutions to be made and all the scores are the same.

By increasing the word size the number of potential seeds in the neighbourhood are reduced and fewer extensions will be started, while increasing T will raise the quality of the acceptable substitutions further reducing the number of words in the passing neighbourhood. These thresholds act as a filter to prevent the algorithm from attempting to align portions of the query and subject where significant alignments are unlikely to be found.

When an alignment is found between one of the query words and the subject an extension begins in both directions. Extension compares the letters in the aligned query and subject through the scoring table and applies a penalty or reward to the cumulative score for that direction [10]. This process continues resulting in matches, mismatches and gaps being inserted into the alignment until the cumulative score drops off by the *X-Drop* threshold or the end of a sequence is reached. The *X-Drop* threshold specifies the maximum value that the score can drop off by before the extension is terminated [2]. This prevents the extension from continuing when the optimal alignment is likely to have already been found. Once an extension is terminated BLAST has found what it considers to be the likely optimal alignment for the w-mer, but further processing is required to determine if the alignment is statistically significant.

Another threshold, s, is applied to the extended alignment to determine if it will be included in the BLAST report [2]. Not all alignments that pass s are statistically significant so further analysis is required [10]. The scores for alignments produced after extension are known as raw scores and are useless unless detailed knowledge about the scoring system is known [30]. These raw scores need to be normalised and converted into bit scores which take into account K and λ, K and λ are scales for the search space and scoring system [45].

$$S' = \frac{\lambda S - lnK}{ln2} \tag{2.1}$$

Bit scores from alignments can be compared with results from other BLAST runs but their statistical significance still needs to be checked. Consider a long alignment with a low score being detected in a large database, this alignment to the uninitiated may at first appear to be useful based entirely on its length, however this alignment has a low statistical significance and many comparable alignments are likely to be detected. These alignments of statistical low significance need to be removed, to do this the expect value is calculated. The expect value represents the number of alignments with the same length and score that are likely to be found in the database that are not related to the query sentence [10]. Clearly alignments with a high expect are unlikely to be useful as they can arise purely by chance, therefore they are filtered out by the expect threshold. Expect is calculated from the bit score and the lengths of the query and database.

$$E = mn2^{-S'} \tag{2.2}$$

All alignments that pass expect are deemed statistically significant, with lower values of

expect implying a greater biological significance [45].

NCBI BLAST

NCBI BLAST has been continuously developed since BLAST's first release and is considered the de facto standard implementation. The first major version of NCBI BLAST was very close to the original specification and did not support gaps as the statistics of gaped alignments had not been fully worked out [45]. This first NCBI BLAST is the most closely related version of the algorithm implemented in this dissertation, but its development was abandoned for BLAST2 which incorporated a number of improvements, including support for gaps. BLAST2 has also become obsolete and was replaced by BLAST+, BLAST+ overcomes numerous issues in BLAST2 and further improves on BLAST's performance.

BLAST+ moves the design into a C++ development environment and adds support for larger databases [4] and slightly departs from the original specification. For example, later versions of BLAST utilise a "two-hit" method to determine which high scoring w-mers to extend [45]. This involves looking along the same diagonal for at least another high scoring w-mer before beginning extension. Modifications like the one described have the potential to significantly increase the performance of BLAST with little impact on accuracy.

NCBI BLAST allows the user to customise all of the standard parameters discussed in this review and provides additional controls that allow some of the newer performance enhancing, accuracy decreasing modifications to be disabled. Due to NCBI BLAST's good flexibility and industry acceptance it is often used as the baseline for benchmarking new BLAST implementations. NCBI provides standalone binaries of its BLAST for download from their website, they also host a web based version where BLAST searches can be made directly online.

MPI BLAST

mpiBLAST is a cluster implementation of NCBI BLAST that boasts perfect scaling across tens of thousands of cores, superlinear speed-up and output that is an exact match to NCBI BLAST [8]. BLAST is expected to have excellent performance on clusters since alignments are independent of each other thus not requiring inter-thread communication. Furthermore, often a database of queries is compared to the subject database instead of

just a single query. This pervasively parallel nature of the algorithm behaves favourably on large clusters and the analysis of mpiBLAST clearly demonstrates this.

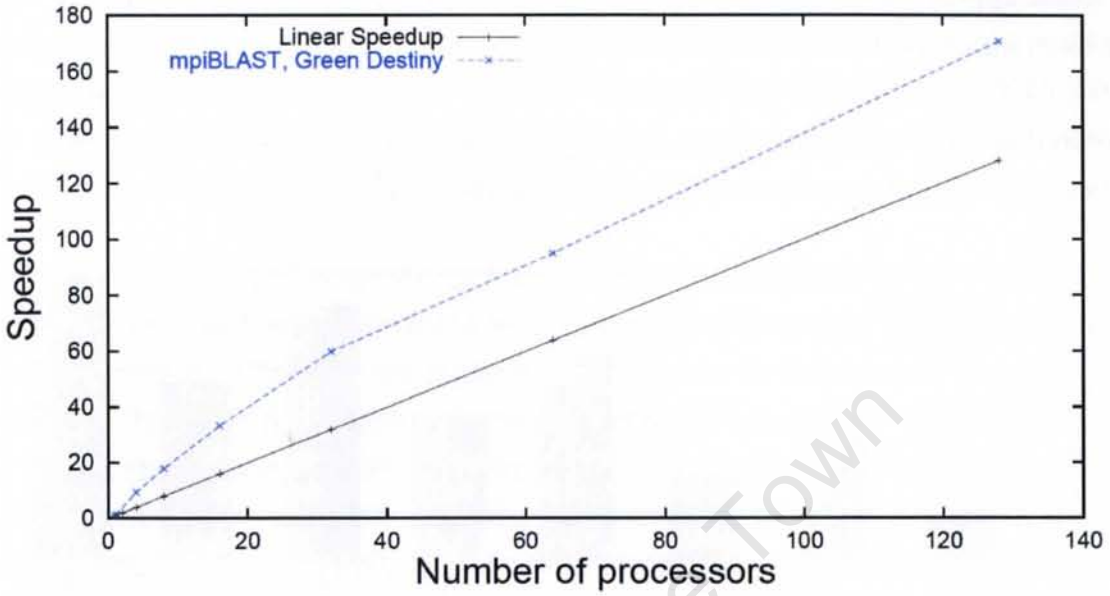


Figure 2.3 – Speedup of mpiBLAST over NCBI BLAST. The graph shows mpiBLAST has a superlinear speed-up represented by the blue dashed line and a plot of linear speed-up for reference. Figure reprinted from Darling [8].

The mpiBLAST implementation divides the database between the compute nodes which has the double effect of reducing disk I/O and reducing the search space each node is required to cover. This design does introduce some communication between the nodes but it does not grow to the point where it becomes a limiting factor [8]. The design of mpiBLAST makes it well suited to operate on efficient low cost clusters and its identical output to NCBI BLAST allows it to be used as a substitute.

GPU-BLAST

A BLAST implementation published in 2011 for nVidia GPGPUs was developed using the CUDA SDK that showed a GPU could be used to realise a speed-up of 3 to 4 times that of NCBI BLAST [42]. Another GPU acceleration paper published in the same year showed a 10x speed-up [26] but at the cost of producing different alignments to NCBI BLAST [42]. It has been observed that biologists are reluctant to use BLAST implementations that produce different output to NCBI BLAST, regardless of the fact it is a heuristic algorithm, thus this GPU implementation will have difficulty finding mainstream applications [42]. For this reason the NCBI BLAST accurate GPU implementation proposed by Vouzis [42] will be discussed in this section.

The GPU implementation was created by embedding GPU acceleration directly into the open source NCBI BLAST code. This lead to the desirable attribute of making the program appear to behave and operate in the same manner as NCBI BLAST, which is something important to BLAST users. Due to GPU acceleration being implemented in the NCBI code special care had to be taken to ensure data structures were properly allocated and placed in appropriate memories as both CPU and GPU threads share these structures to avoid the overhead of duplicating data [42].

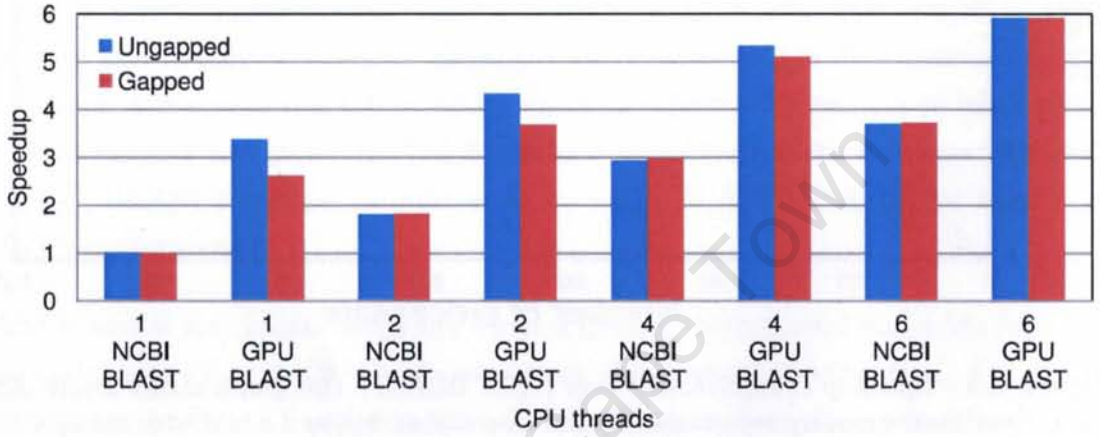


Figure 2.4 – GPU-BLAST's speed-up relative to a single-threaded CPU as a function of CPU threads. The graph shows GPU acceleration consistently provides a good boost in performance over NCBI BLAST. Figure reprinted from Vouzis [42].

The GPU-BLAST architecture assigns a portion of the database subjects to each thread in the GPU, and to ensure resources are not wasted the database subjects must be sorted by length, otherwise a long subject can hold-up unused resources in a warp[2] caused by short subjects finishing quicker [42]. The sorting can be done during the database formatting stage which is executed before NCBI BLAST can perform a run, so this additional step does not add any real overhead. Further, the order in which subjects in the database are submitted for processing has no impact on NCBI BLAST allowing the formatted database to be compatible with both implementations [42].

Mercury BLAST

Mercury BLAST uses an FPGA connected to a host PC via the PCI-X bus to accelerate the word matching and extension stages of the BLAST algorithm. The word matching stage is divided into two sub-stages, a bloom filter and a hash table. Bloom filters are used

[2]A warp is group of hardware threads in the CUDA architecture that simultaneously execute the same instruction [31].

to test membership to a large set. The bloom filter will not miss any valid w-mers but does introduce false positives at a rate dependent on the number of w-mers programmed into it. Bloom filters are well suited to hardware implementations and in this design 16 w-mers are able to be processed in a single cycle. The second sub-stage maps a w-mer to its query position with a lookup table stored in an external SRAM module. This hash table is able to perform a single lookup per cycle and can become a bottleneck so it relies on the bloom filter to keep its load balanced [25].

Ungapped extension of Mercury BLAST also includes modifications to the NCBI BLAST specification to make it more suitable to implementation in hardware. The major difference is that Mercury BLAST does not allow the extension to be dynamically terminated by the *X-Drop*. Instead a fixed window size is applied centred around the w-mer and a single pass trough the window from start to end is made. Some additional checking is imposed to ensure that optimal extensions includes the w-mer and to decide if the extension can pass onto stage 3, gapped extension. If an optimal alignment is detected which ends at either of the window boundaries it is automatically passed onto stage 3 regardless of its score [25].

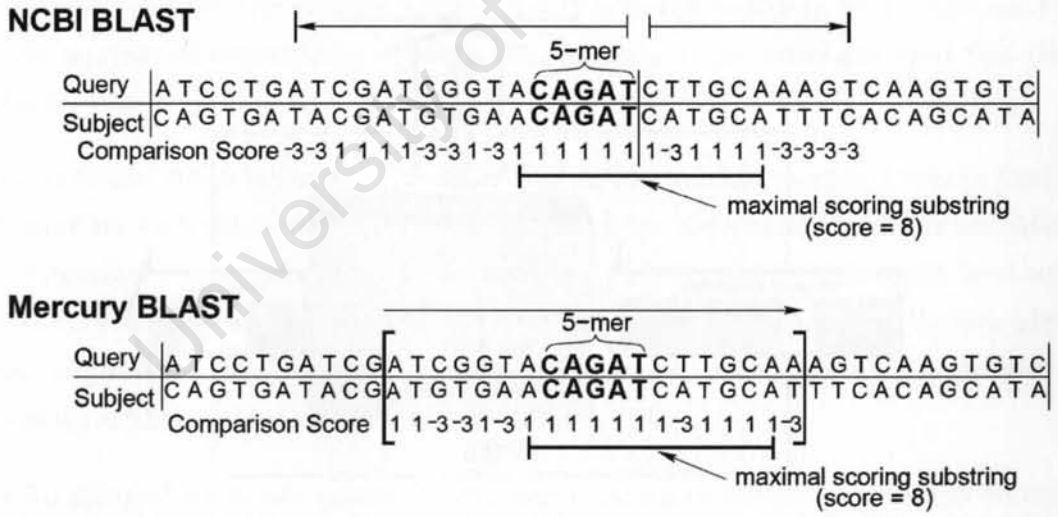


Figure 2.5 – An illustration of the different approaches for ungapped extension taken by NCBI and Mercury BLAST. NCBI BLAST performs extension in both directions starting at the w-mer and continues until it is terminated by the X-Drop. Mercury BLAST applies a fixed extension window and performs a single pass from the start to the end of the window. Figure reprinted from Lancaster [25].

This implementation of BLAST on a Xilinx XC2V6000 FPGA realises a 30x speed-up over a Pentium 4 at 2.8GHz with 1GB of RAM and results that are 99% similar to NCBI BLAST [25].

RC-BLAST

The RC-BLAST implementation began with an in depth study into which portions of the BLASTN algorithm occupied the most time. This was done by identifying slow sections of code and moving them into a single critical code routine. Once the critical areas were identified they were able to determine that the slowest part of the algorithm was finding words, which accounted for 80% of the runtime [29].

From their analysis they determined that it was possible to implement the critical code in hardware using a lookup table that stores the query in 8 byte rows representing the words in the neighbourhood and a 2 bit counter to indicate the number of occurrences of a word. Lookups can then be performed to determine the location of w-mers and the matches are returned to NCBI BLAST for the remainder of the processing [29].

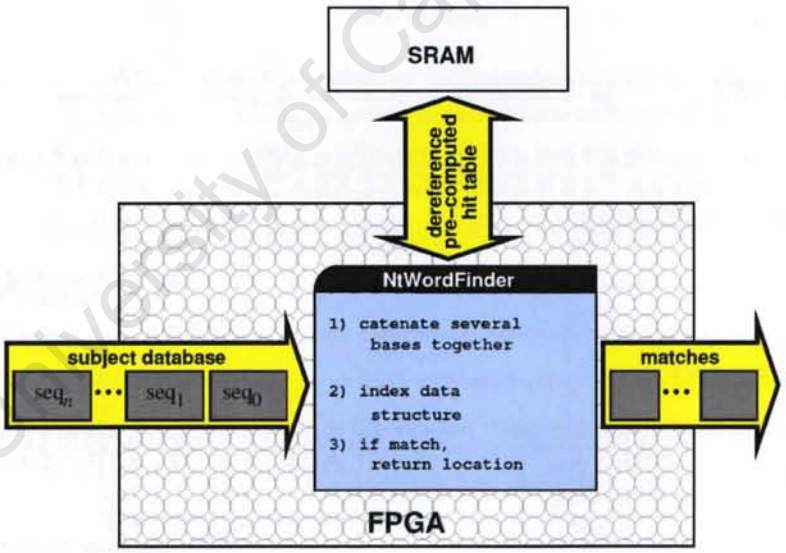


Figure 2.6 – A high level illustration of the operation of RC-BLAST. A pre-computed lookup table for the query is processed and stored in SRAM, subjects from the target database are passed through the FPGA and compared to the lookup table, detected matches are returned to the host for further processing. Figure reprinted from Muriki [29].

While benchmarking is not properly covered in the paper their analysis points to a maximum speed-up of 3.7x NCBI BLAST. The speed-up is limited because the design only accelerates a small portion of the algorithm, however the output from the system produces results identical to NCBI BLAST.

Systolic Array based BLAST

In this FPGA implementation of BLAST the seed detection and extension stages are once again implemented in hardware. This design utilises a systolic array for seed detection, instead of a lookup table and utilises multi-channel extension modules that behave in a similar manner to NCBI BLAST. By eliminating the lookup table there is no longer the need to access slower off chip memory which removes the bottleneck in Mercury BLAST and there is sufficient space on the FPGA to implement additional stages of the algorithm unlike RC-BLAST which is limited to seed detection [44].

While this FPGA implementation is capable of producing output that is the same as NCBI BLAST it can only do so under specific circumstances due to the designs inflexibility. Each of the elements in the systolic array holds a single letter of the query and the database runs over the array one element at a time. Each clock cycle the elements make a comparison between their loaded query letter and the current database letter they are associated with. If a match between the pair is detected control signals are asserted to indicate to the five elements on each side of it that a detection has been found. Each of the elements monitor their ten input signals and their own comparison and if all control lines are asserted then the element knows that it is in the centre of an 11-mer match [44]. Since the number of connections between the elements in the array are fixed this BLAST implementation is limited to a sensitivity with a word size of 11.

An intermediate stage between seed detection and extension merges 11-mers that were found next to each other to help reduce the load on the extension modules. Merging before extension has no effect on the output since two 11-mers which could be combined into a 12-mer would run through the entire extension process and result in two identical optimal alignments, of which one would be discarded. Merging simply prevents the unnecessary additional extensions from beginning [44].

The valid merged seeds are passed to extension modules which have access to a multi-channel bank of memory which stores the query and current database subject. This memory is composed by loading multiple copies of the query and subject into dual ported memory which is then presented as a single multi-channel memory. While this process is wasteful in duplicating the contents of memory it is able to provide the parallelism required for the extension modules to keep up with the seed detection [44].

Due to the simplicity of the design of the systolic array the FPGA is capable of containing 3072 processing elements with the design clocked at 189MHz on XC4VLX160 and realise a speedup of 10x NCBI BLAST [44].

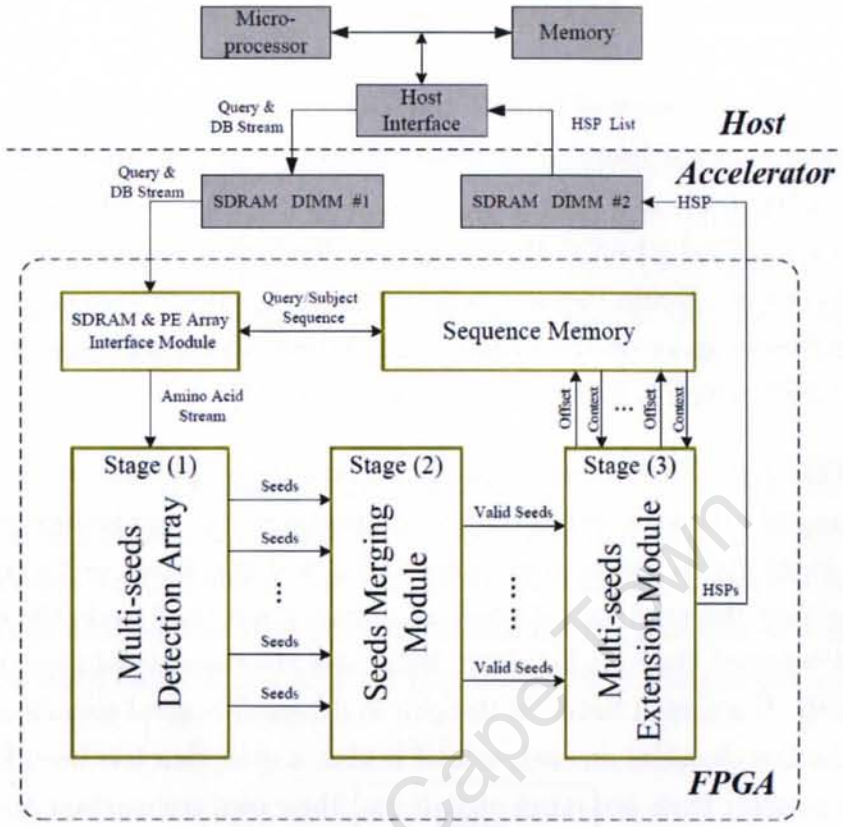


Figure 2.7 – The high level architecture of a systolic array based BLAST implementation. Data is passed to the FPGA from the host and proceeds through the Multi-seeds Detection Array. Detected seeds are merged and passed to the Multi-seeds Extension Module, then back to the host. Figure reprinted from Xia [44].

2.2 Reconfigurable Computing

The first work on reconfigurable computing began in 1959 when John Pasta[3] challenged Gerald Estrin at UCLA to find new ways of organising computer systems. This challenge arose due to Pasta's opinion that computer manufactures were not developing innovative computer architectures capable of solving the vital computational problems of the day. In response to the challenge Estrin presented the design for the first reconfigurable computer, *The Fixed Plus Variable Structure Computer*. The system was divided into a fixed structure part, which contained hard-wired critical circuits and commonly used logic, and a variable structure part which consisted of blocks which could be rewired by hand [9].

Despite Estrin's early success it was not possible to implement custom processor archi-

[3]John Pasta was a highly respected applied mathematician and physicist. He was chairman of the Mathematics and Computer Science Research Advisory Committee to the Atomic Energy Commission [9].

tectures in reconfigurable hardware until the invention of FPGAs in the 1980s. One of the first reconfigurable architectures to be developed using FPGAs was SPLASH, which consisted of 32 Xilinx 3090 chips each with 128kB of RAM. The FPGAs were arranged in a linear array as the devices was intended for use with algorithms that could be mapped into systolic arrays [33].

Current FPGA architectures consist of general purpose structures with a configurable interconnect and specialised fixed structures [28]. The general purpose structure contains Configurable Logic Blocks, CLB, which are used to create arbitrary logic, however some common circuits, like memory and DSP functions, are expensive in CLB usage and fixed logic for these devices is provided. Some flexibility of the fixed devices is facilitated by allowing their interconnect to be manipulated and for them to be bonded together to form larger functions.

2.2.1 Parallel Computing

The Landscape of Parallel Computing Research: A View from Berkeley technical report published in 2006 observed that for the most part software developers were heavily relying on new hardware to overcome the performance issues of their programs while maintaining the traditional von Neumann type processors. During the two years of research by the multidisciplinary team processor manufactures began releasing multi-core architectures that were effectively two of the previous generation chips on the same die. The team investigated the likely outcome this evolutionary approach for parallel architectures and determined that it would face diminishing returns when 16 and 32 processor systems were realised [11].

The research team laid down a number of new "conventional wisdoms" to better reflect the new reality of computing and three "walls" were identified. The walls are areas in microprocessor design where the current generation of generic processors are receiving diminishing returns. The first is the power wall, which states that we are now able to put more transistors on a chip than we have power to turn on. New architectures need to take this into account and more efficient designs need to be produced to work within the power envelope. The second wall is memory, which addresses the fact that accesses are now significantly slower than processing. An access to memory can take hundreds of cycles while floating-point multiplication only takes four. ILP is the third wall which identifies that diminishing returns are being observed in extracting instruction level parallelism. ILP has been relied upon by software developers to increase the performance of their code between generations of processors without having to redesign their systems [11].

These walls point to a future where efficient design with application specific hardware is a viable method for increasing performance over the previous generation's architecture. This new ideology has been widely adopted by academics and industry with recent processors containing on chip GPUs and AES encryption units [21] with manufactures admitting that heterogeneous computing is the way forward.

In this environment FPGAs find a natural fit due to their low development cost compared to ASIC design and their computational density. Traditionally FPGAs excelled at integer operations because of the simple logic required to implement them could easily be synthesized in CLBs while floating point presented a greater problem. However, significant progress has been made in improving the floating point abilities of FPGAs and current architectures have far more potential floating point power than current state-of-the-art conventional CPUs [39].

With the size of FPGAs capable fitting complex designs, the maturing of the tool chain, their potential computation power exceeding conventional CPUs and their ability to fill the gap between software and ASICs reconfigurable computing has found its way into many applications from small low power devices to HPC.

2.2.2 ROACH

ROACH is a Virtex 5 based reconfigurable computer that merges aspects from Berkeley's IBOB and BEE2 platforms. The hardware was developed as an upgrade for the CASPER hardware used in radio astronomy with an intended application being the KAT/SKA project. The board contains either a Xilinx V5LX110T or V5SX95T FPGA connected via the OPB bus to a PowerPC 440EPx embedded processor [14].

The FPGA subsystem contains two Z-DOK+ connectors for ADC or DAC cards providing backwards compatibility with IBOB ADC boards and four CX4 10GbE connectors providing a total of 40Gbps bandwidth. The FPGA also has access to two QDR SRAM modules and one DDR2 DIMM [14].

The PowerPC subsystem provides the primary command and control facility for the board with I/O capabilities to the FPGA. The system's primary component is a AMCC PowerPC 440EPx embedded processor supported by: DDR2 DRAM, 16x32M flash memory, 10/100/1000 Ethernet interface and a Xilinx XC2C256 CPLD to act as an interface to additional I/O. The PowerPC subsystem runs the BORPH operating system which is an extended version of the Linux kernel designed for reconfigurable computers [14].

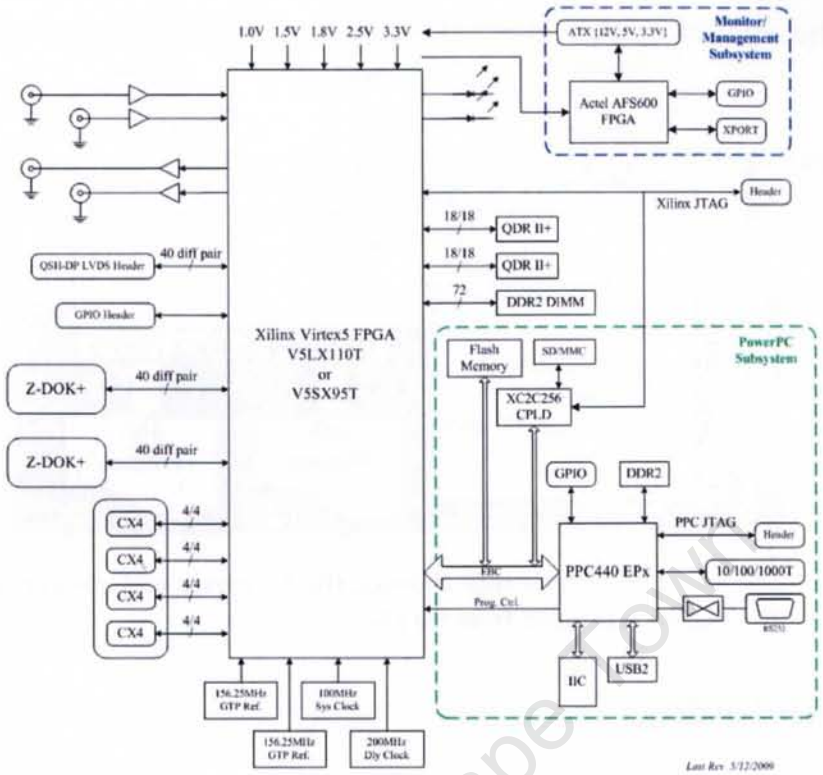


Figure 2.8 – A block diagram of the ROACH architecture showing the FPGA, PowerPC and Monitor/Management subsystems with their associated resources. Figure reprinted from CASPER [14].

2.2.3 BORPH

BORPH is a Linux based operating system designed for the BEE2 reconfigurable computer [37] and ported to ROACH. BORPH provides useful abstractions to allow for easy programming and management of the FPGA by providing the user with hardware processes and shared memories.

The hardware processes allow the user to abstract the programming, execution and control of the FPGA to the same level as conventional processes [37]. A hardware process is created by the execution of a "bof" file, which contains a bit stream for programming the FPGA and interface configuration data. The hardware process creates an interface to shared memory in the /proc/ directory where data can be read or written to the FPGA during runtime through what appears to the user as files. The shared memories are defined during the design time of the FPGA fabric.

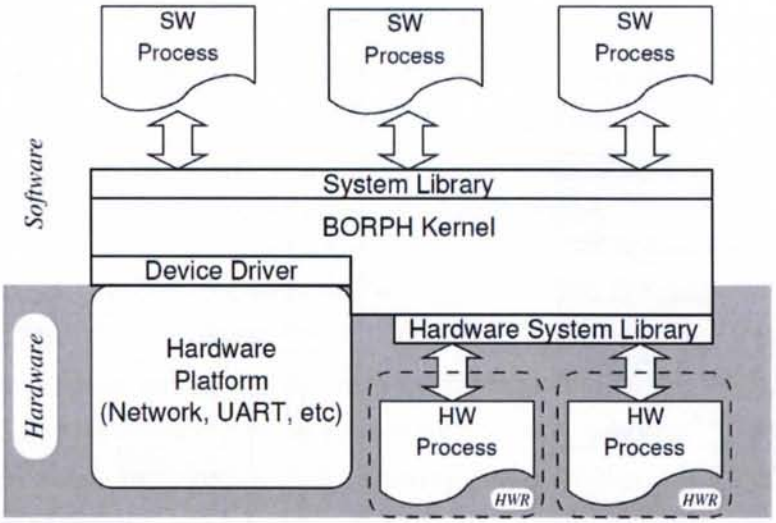


Figure 2.9 – The BORPH architecture showing the hardware and software domains with their interactions. Figure reprinted from So [37].

2.3 Conclusion

The importance of sequence similarity searches in moving forward our understanding of microbiology is evident and with the ever growing databases of sequenced DNA there is still a great demand for faster analysis tools [45]. The early methods of analysis by hand were replaced by slow but accurate algorithms capable of finding the optimal alignments, which lead to the creation of heuristic algorithms to further increase their performance.

The popular BLAST suite has been widely accepted by the bioinformatics community and offers a good compromise between speed and the accuracy of the local alignments it produces [10]. Due to BLAST's success many accelerated versions have been developed for a range of hardware including clusters, GPGPUs and FPGAs.

Although BLAST is a heuristic microbiologists place emphasis on the output of BLAST accelerators being the same as NCBI BLAST, despite this not all the accelerators conform to the standard and therefore are not likely to succeed in the industry. Alternative conforming accelerators often pay the penalty in complexity of the design by not mapping well to hardware thus providing only limited support for the BLAST suite.

In recent years FPGA technology has progressed to the point were the hardware is capable of supporting large complex designs allowing new architectures to be developed to overcome the drawbacks of the previous generation of FPGA based BLAST accelerators. A reconfigurable computer developed for radio astronomy featuring a medium sized Virtex

5 FPGA was used to implement a BLAST accelerator compliant with the original BLAST specification.

In the next chapter the project environment for ROACH BLAST is presented detailing the tools and procedures required to recreate this work.

Chapter 3

Project Environment

This chapter details the project development and test environment providing information on the tools and procedures required to recreate the work. The hardware section presents: the ROACH boards configuration, the specification and setup of the x86 head node for the ROACH cluster and the layout of the 1GbE and 10GbE network infrastructure.

The majority of the work in this dissertation was performed with Xilinx ISE, however the board support package and 10GbE core are extracted from the ROACH tool chain which utilises MATLAB/SimuLink as the development environment. While the extraction process is not difficult the ROACH documentation does not cover it, therefore it is presented here.

Finally, the procedure to compile the project and generate a programming file compatible with BORPH is discussed. The method for executing and interacting with the programming file is also presented.

3.1 Hardware

The hardware utilised in this dissertation consists of the the following: a ROACH board, workstation and network infrastructure. The ROACH board forms the basis of the accelerator, implementing the seed detection and extension stages of the BLAST algorithm.

A x86 workstation drives the ROACH board and provides preprocessing and post-processing support for the accelerator. The ROACH board and workstation are connected

by 1GbE and 10GbE, 1GbE provides access to the PowerPC subsystem to control the reconfigurable computer and the 10GbE network delivers data directly to the FPGA.

3.1.1 ROACH

The ROACH board used is the Xilinx Virtex 5 SX95T variant manufactured by Digicom Electronics. The SX95T ROACH is the primary target for this dissertation but a ROACH BLAST version for the LX110T hardware was also designed. The major difference between these FPGAs is that the SX95T contains more fixed logic and fewer slices than the LX110T, see Table 3.1.

Table 3.1 – The significant differences between the Xilinx SX95T and LX110T FPGAs.

Virtex 5	SX95T	LX110T
Slices	14 720	17 280
BRAM (KB)	8 784	5 328
DSP48E	640	64

The ROACH boards network boot off the *uImage-20091006-mmcfix* kernel image hosted on the head node of the cluster through 1Gbps Ethernet. The ROACH boards share the *filesystem_etch_2010-03-24_sd_shipping.tar.gz* file system is also hosted on the head node. ROACH kernels and file systems are available for download at the following addresses:

Kernel https://casper.berkeley.edu/svn/trunk/roach/sw/binaries/linux/

File System https://casper.berkeley.edu/svn/trunk/roach/sw/binaries/filesystem/

A detailed guide on configuring ROACH boards for network booting is available at: https://casper.berkeley.edu/wiki/ROACH_NFS_guide

3.1.2 Workstation

A Dell PowerEdge 2900 Server is used as the head node of the ROACH cluster and for benchmarking NCBI BLAST. The workstation contains a quad core Xeon X5450 clocked at 3GHz with 16GB of fully buffered DDR2 667 ECC RAM. The workstations file system and operating system is stored internally on a RAID1, mirror, array consisting of two 750Gb 7200rpm SATAII hard drives.

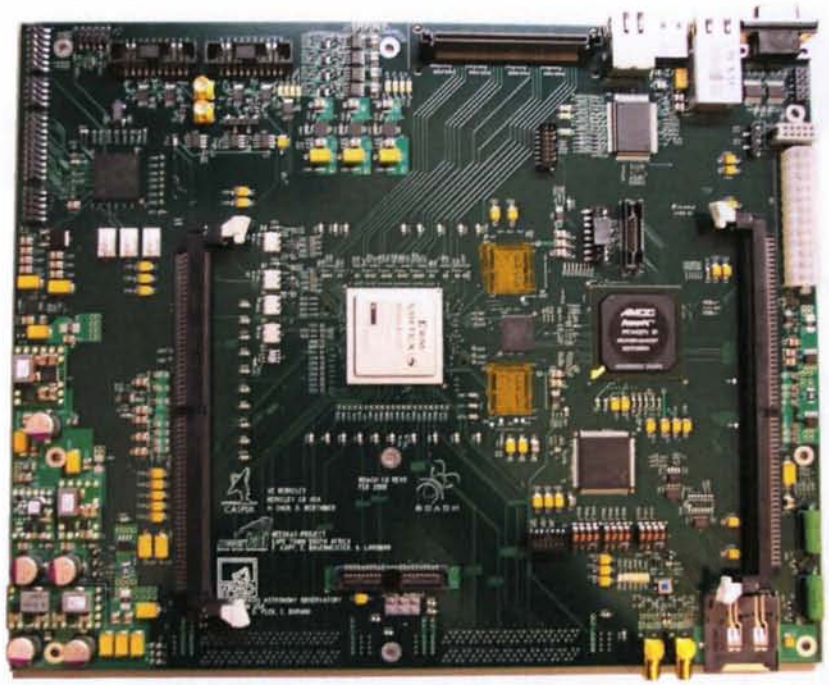


Figure 3.1 – ROACH v1.0 reconfigurable computer with a Xilinx LX110T FPGA, reprinted from CASPER [14].

The workstation is attached to the ROACH PowerPC subsystem network via a Broadcom NetXtreme II BCM5708 Gigabit Ethernet adapter. The 10GbE interface is a provided by a Myricom 10G-PCIE-8B-C adapter connected via a PCI Express x8 bus.

The operating system running on the workstation is Ubuntu server 10.04LTS 64Bit with Linux kernel 2.6.32. The workstation hosts NFS and FTP servers for network booting the ROACH boards and a SSH server for remote access.

3.1.3 Network Infrastructure

The ROACH boards and workstation are connected via two separate star networks, a 1GbE network for control and a 10GbE network for high bandwidth access directly to the FPGA. The ROACH designers warn that there is a bug in the ROACH PHY chip which causes unreliable operation at 1Gbps and that the solution is to connect to them at 100Mbps [43], however none of the ROACH boards used in this project have behaved in this manner. The 1GbE control interfaces in the cluster are connected via a D-Link DGS1008D 1GbE switch.

The 10GbE network utilises the 10GBASE-CX4 copper standard which specifies link lengths of up to 15 meters [7]. The ROACH boards and workstation are connected by

Figure 3.2 – Fujitsu XG700 CX4 10GbE Switch, reprinted from Fujitsu [15].

1m CX4 cables through a Fujitsu XG700 CX4 switch. Through testing it was determined that the 10GbE network has no packet loss.

3.2 Software

The software described in this section is required for the development of ROACH BLAST. ROACH BLAST moves out of the ROACH tool chain, which is primarily for radio astronomy, and is mostly implemented in VHDL written in Xilinx ISE.

3.2.1 ROACH Tool Chain

The ROACH tool chain provides an interface to the developer through the MATLAB/SimuLink environment so that development can be simplified to a process of connecting pre-written blocks representing HDL code that is known to function correctly. The major issue with this environment is that the CASPER blocks are geared towards radio astronomy and were not suitable for the design of ROACH BLAST.

Despite the ROACH tool chain's unsuitability for ROACH BLAST a top level design had to be developed using it. The SimuLink interface consists of blocks and a page where the blocks are connected to form a system. The system can be simulated accurately in SimuLink or converted into a programming file using various Xilinx tools which are automatically called by MATLAB.

The bocks can be broadly divided into three groups, SimuLink blocks, Xilinx blocks and CASPER blocks. SimuLink blocks provide support for simulation and are not converted into hardware. Xilinx blocks target Xilinx FPGAs defining generic hardware structures and configuration functions. The CASPER, or yellow, blocks represent custom hardware written for CASPER reconfigurable computers with SimuLink models for simulation.

The MATLAB/SimuLink environment is used to design the top level architecture in this project. The ROACH BLAST design consists of the ROACH board support package and a 10GbE yellow block. Other SimuLink blocks are connected to the 10GbE block but only serve to suppress warnings and errors generated by MATLAB.

The ROACH tool chain and configuration instructions are available for download at: https://casper.berkeley.edu/wiki/MSSGE_Toolflow_Setup

3.2.2 Xilinx ISE Suite

The Xilinx ISE Suite contains many applications of which only three will be discussed here: ISE Project Navigator, Platform Studio and ISIM.

ISE Project Navigator is the primary development environment in ISE for HDL code and interacts with other applications in the ISE suite to assist in the development process of the project. ISE Suite version 11.4 was used in this dissertation because it is the most recent version of ISE that is compatible with the ROACH tool chain. Newer versions of ISE drop support for the OPB bus which ROACH uses to connect the FPGA and PowerPC.

ISIM is the HDL simulator which was used to verify the HDL code written for this project. ISIM was only used to simulate the HDL produced in this dissertation and was not used to verify the netlist produced by the ROACH tool chain.

Platform Studio forms part of the Embedded Development Kit, EDK, software for designing embedded processor systems. Part of the output from the ROACH tool chain is a Platform Studio project which contains the design produced by the MATLAB/SimuLink environment. Platform Studio is used to manipulate this project to make it compatible with the design environment ROACH BLAST is written in.

3.2.3 Core Extraction

For the HDL designed in ISE to be generated into a programming file compatible with the ROACH hardware the board support package needs to be extracted from the ROACH tool chain. This package contains constraints to pin locations, clock frequencies and logic required for communication with the PowerPC over OPB. The process is as follows:

1. Design the top level for the project using the ROACH tool chain.

2. In the BEE XPS window launched from MATLAB tick all boxes up to and including IP Synthesis, then click run XPS.

3. Open the *.xmp file in Platform Studio produced by the ROACH tool chain after XPS has completed.

4. Platform Studio will prompt the user to upgrade various cores used in the design to the current version, accept these updates.

5. Edit the *.xmp file in plain text outside of Platform Studio and change InsertNoPads to 1. InsertNoPads set to 1 tells Platform Studio not to put I/O buffers on the signals at the edge of the design.

6. Use Platform Studio to make signals external for ports to cores you wish to access from the design written in ISE. You can also use Platform Studio to break connections between components and rewire them. The purpose for the manipulation of the XPS project in this dissertation was to expose signals from the 10GbE core to the code written in ISE.

7. Use Platform Studio to generate a netlist for the project.

The output from this procedure will create a *.ngc file in the "implementation" directory of the Platform Studio project. This netlist contains all the logic required to support the ROACH boards. A constraints file for the design is provided in the "data" directory of the Platform Studio project and together with the netlist will be used to generate a bitstream compatible with the ROACH hardware.

The netlist and constraints file can then be added into the ISE project and interfaced with. The netlist sits under a top level HDL file which exposes signals connected to the physical pins on the FPGA package. Other signals from the imported netlist are diverted into the HDL project.

Since the logic required to support the OPB bus has already been generated and is defined in the netlist this project can be moved onto the current version of ISE. Programming files have been created from ISE 13.1 and worked as expected, however ISE 11.4 is still required to support the ROACH tool chain.

3.3 Compilation & Execution Procedure

A bitstream generated by ISE is not compatible with BORPH, therefore it needs to be converted into the compatible bof file format. The ROACH tool chain contains a simple executable called mkbof which creates a bof file from a binary configuration file. A binary configuration file is generated by ISE's Generate Programming File process.

The process described in Section 3.2.3 which produces the netlist and constraints also produces a file containing configuration information used by mkbof to create the bof file. Mkbof is executed with the *.bin file produced by ISE and the core_info.tab configuration file produced by the ROACH tool chain using the following command:

```
$ mkbof -o <boffile>.bof -s core_info.tab -t 3 <binaryfile>.bin
```

A bof file must be copied into the /boffiles/ directory and set executable on the ROACH board for it to be programmed. As described in the hardware section of this chapter the ROACH boards share a file system hosted on the head node, so distributing a bof file to the ROACH boards is done by simply copying it into the boffiles directory in the NFS shared file system.

The ROACH boards run a KATCP server which allows some of the ROACH boards control to be performed remotely, this saves the user from having to login to BORPH to program or configure the board. A Python script was developed in this dissertation to program the bof file and configure the 10GbE core using the KATCP protocol. All a user needs to do to program the ROACH board is execute this script on the head node of the cluster and wait for the programming to complete.

3.4 Conclusion

ROACH BLAST uses the ROACH hardware for processing computationally intensive portions of the BLAST algorithm and an x86 workstation connected via 10GbE to perform the preprocessing and post-processing tasks. Although ROACH BLAST utilises ROACH hardware it does not rely solely on the ROACH tool chain because the majority of its implementation is written outside the tool chain in VHDL using ISE.

To produce programming files compatible with the ROACH hardware additional procedures need to be followed both before and after compilation. The output from these

procedures is a ROACH compatible bof file that can be programmed via the KATCP protocol running on BORPH.

Chapter 4 discusses the hardware and software architecture of ROACH BLAST describing the design choices and a detailed explanation of the final system.

Chapter 4

Architecture

This chapter presents the hardware and software design of ROACH BLAST with a discussion on the alternate architectures that were considered and the reason for their exclusion. From the literature review it was decided that the design would be based on a systolic array and the FPGA would contain the seed detection and extension stages of the BLAST algorithm.

Early designs suffered from inefficient usage of the FPGA by not utilising fixed resources like the BRAM. Although the design does not map well into BRAMs to not use them was too wasteful, thus design changes were implemented to free up slices through inefficient BRAM usage. Other designs required large amounts of interconnectivity between the processing elements, or contained large amounts of logic that would only rarely be used. Ultimately the design that: made good use of the FPGA resources, was scalable, didn't contain large amounts of rarely used logic and was compliant with the BLAST algorithm was selected.

The basic architecture of the final design consists of Detection Regions which each hold a portion of the query sequence and process alignments independently. The database is fed through the system as a stream of letters to all the Detection Regions arranged in parallel, at peak performance the Detection Regions are able to move a letter in the database through the system every cycle. Inside a Detection Region there is a Seed Detection Array, Extension Units and regional control logic.

The Seed Detection Array in each region holds 128 letters and is capable of operating at 60MHz. The Seed Detection Array compares the query and subject symbols along its full length and detects segments where they match exactly. If the matching segment is

the word size or longer an index to the match is produced and it moves to the extension stage.

Each Detection Region contains eight bidirectional Extension Units to extend the seeds produced by the Seed Detection Array. Each Extension Unit holds the full query and a window of the database in BRAM. The Extension Units are capable of simultaneous extension in both directions, through the use of interleaving memory accesses or duplicating BRAM contents, and process a letter in both directions every cycle.

Preprocessing and post-processing support for the design is managed by a workstation connected to the FPGA via 10GbE. The software: reads and formats the query and database, transmits the data to the FPGA, receives data from the FPGA, decodes the alignments, filters the alignments and writes the report.

4.1 Considered Designs

A number of alternate designs were considered during the process of selecting an appropriate architecture for ROACH BLAST. The design options discussed in this section were all discarded due to likely problems that their implementation would cause, which was most often the inefficient use of the FPGA resources.

4.1.1 Global Extension Buffer

An initial design decision was how to handle the storage of the letters in the database so that they can be used during processing in the extension stage. The first option was to implement a global buffer sitting at the end of the Seed Detection Array where the exiting letters would be stored for extension.

This design has a number of problems, firstly, all the Extension Units would require access to the letters in the Seed Detection Array and global buffer to either perform extension directly off them or copy the data. In both cases the design will not map into BRAMs because of its unstandardised structure, thus requiring the use of slices to implement the memory. Synthesising large memories out of slices is highly inefficient, therefore this approach was abandoned.

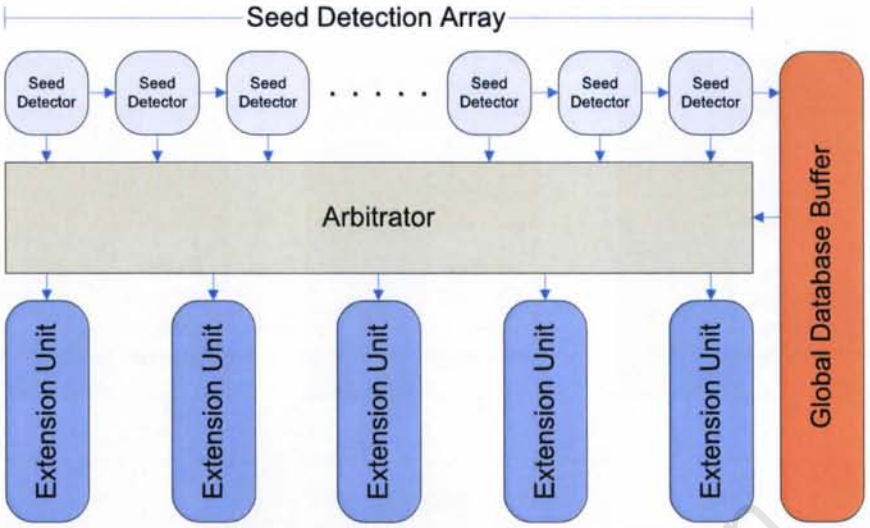


Figure 4.1 – Global database buffer concept. All the Extension Units are required to access the Seed Detectors and the global database buffer through the Arbitrator during extension. Only one copy of the database is required, however the design cannot be synthesised using standard BRAM structures.

4.1.2 In Seed Detection Buffered Seeds

Considerations were made with regard to how the data would transition between the Seed Detection Array and Extension Units. There was concern that the transition could be a bottleneck in the design and to overcome this a buffering facility would be built into each Seed Detection Element. This would prevent the Seed Detection Array from wasting time stalling while waiting for a seed to be transferred to extension.

Each element would have a 10 line memory to store the length of the seed and the number of letters that have moved since it was detected. Based on this information the Arbitrator can decode the seed into a query, database and length index for extension and escalate priority the longer it is delayed. Ultimately the large number of small memories proved inefficient when mapped into the FPGA thus a more centralised buffering scheme was designed.

4.1.3 Large Systolic Array

A large grid of elements each connected to their neighbours would be able to determine if they are on the end of a seed. If they are not they report this to the element below them for continued processing in the next cycle. This process would continue by dropping the seed down a line in the matrix until only the first element of the seed is left. When only

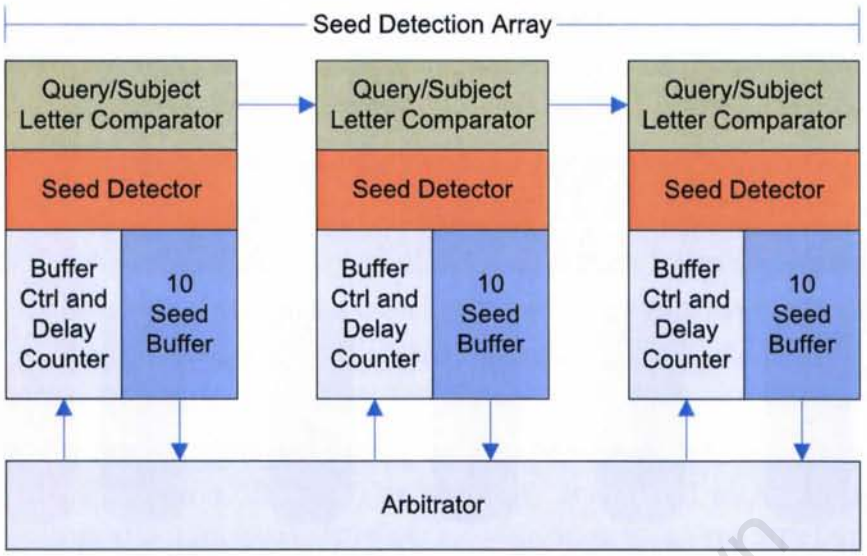


Figure 4.2 – The basic components of a Seed Detection Element with a 10 line buffer. In element buffering would remove the requirement of buffering in the Arbitrator but does not map well to the FPGA's fixed logic.

one element remains, the row in the matrix represents the length of the seed, and the column its starting position in the query. From this point the seed would drop down the remainder of the rows until it exited and gets decoded by the Arbitrator.

While this design is conceptually simple it would require a large number of Detection Elements to process high word sizes. A reality with the design is that the majority of the Seed Detection Elements would be only rarely used, for example a element in the bottom line would only be needed when a seed started at its position in the first line. An architecture containing so many rarely used Detection Elements was considered too inefficient and lead to the design being discarded.

4.1.4 In Array Extension

Instead of separating the seed detection and extension stages, as was done in all other designs, an implementation where the entire system is in a systolic array was considered. Additional logic would be required in each of the Seed Detection Elements to support the scoring system but Extension Units would no longer be necessary.

The design would operate in a similar manner to the final design selected for this dissertation, however minor modifications to support bidirectional extension would be required. Possibly the biggest problem with the design is that it would require the detection array to spend large amounts of time stalled while seeds were being extended.

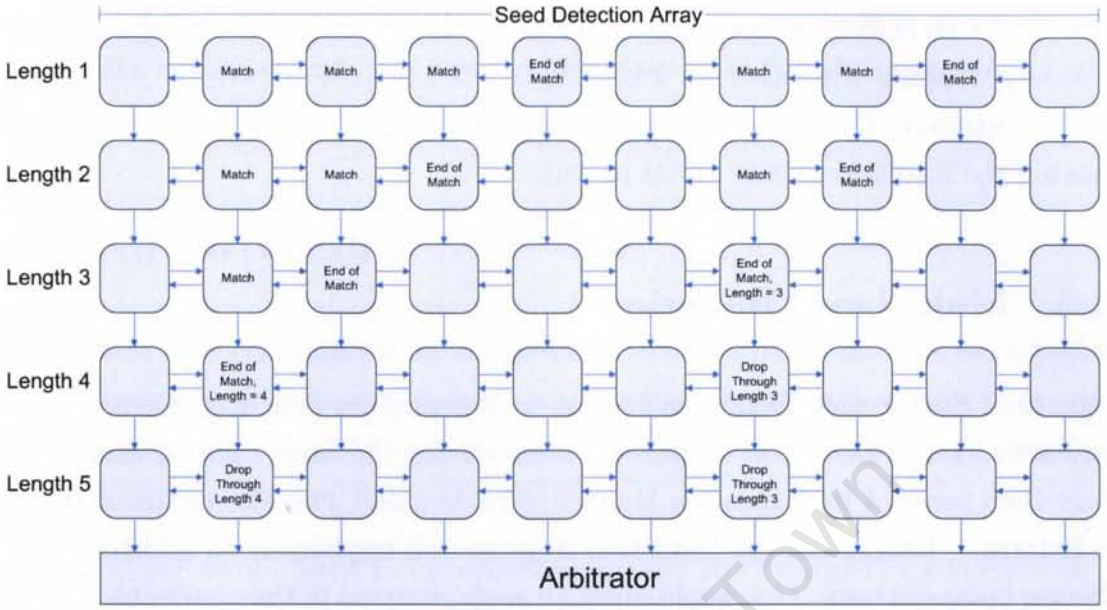


Figure 4.3 – This systolic array only requires each of the elements to communicate with its neighbour and uses the depth at which the seed is reduced to one element to calculate its length.

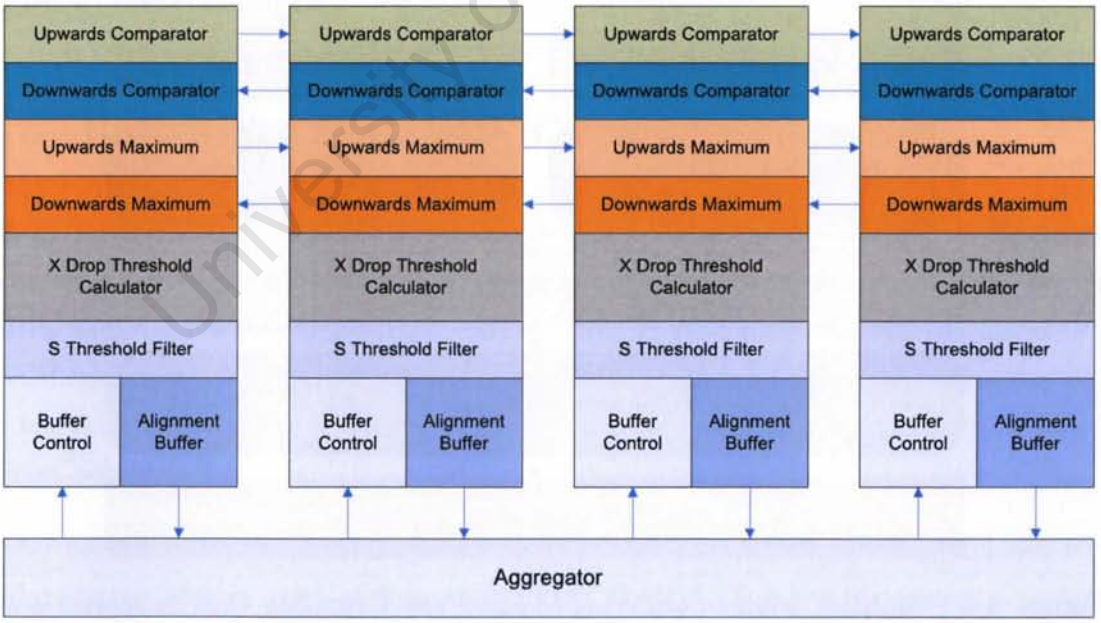


Figure 4.4 – A significantly more complex systolic array that implements seed detection and extension. The upwards and downwards comparators would increment or decrement the score by the user specified penalty and reward until the score dropped off by x. The alignment would have to pass the s threshold before buffering and eventual transmission back to the workstation.

Allowing the extensions to be offloaded from the detection array for processing in dedicated Extension Units is likely to lead to a better performance because the detection array can continue processing while there is a backlog of extensions. Furthermore replicating the logic for extension throughout the detection array is likely to require more logic than removing the Extension Units would provide.

4.1.5 Multi-Level Buffering

In the final design each Detection Region is allocated its own Extension Units. If a query is short or a Detection Region is not producing the same number of seeds as the others then some of its Extension Units will be unused. The alternative is to remove the Extension Units from the Detection Regions and implement an additional level of buffering between them. This would allow all seeds detected in the system to be allocated to any of the Extension Units.

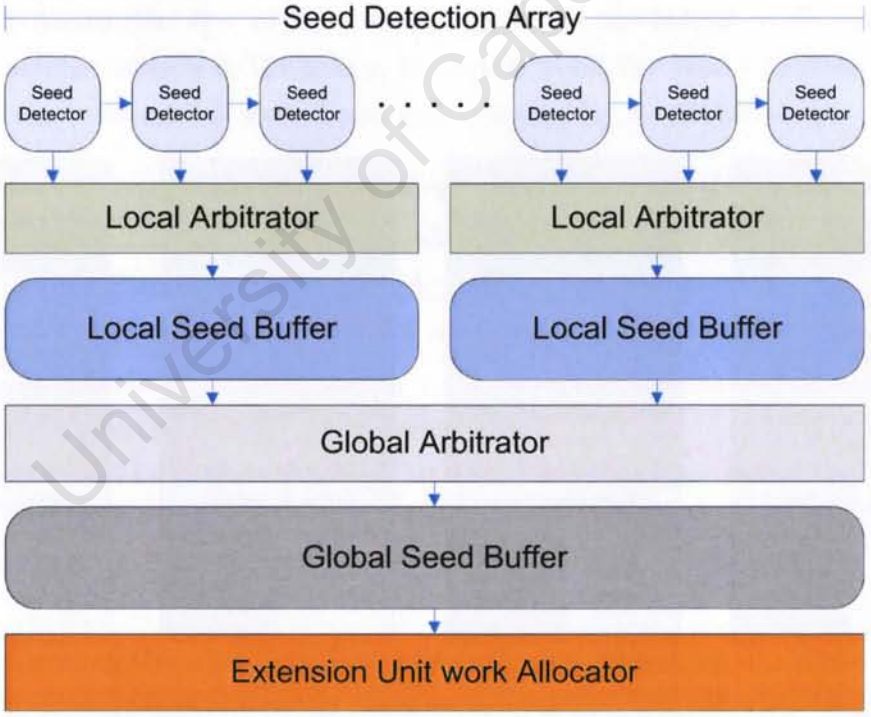


Figure 4.5 – Multiple levels of buffering to allow any Extension Unit to service a seed produced by any Detection Region.

While this design would work well in the version implemented on the SX95T due to the short length of the query it supports, it would not scale well. As larger designs are implemented more Extension Units will be required, and eventually a bottleneck will arise caused by a single module allocating work to all the Extension Units. This is most

likely to become an issue when the word size is low, in this case many of the seeds are rapidly discarded during extension. Under these condition there will more be Extension Units than the system can allocate work to, therefore a parallel allocation architecture is required.

4.1.6 TCP Networking

It is important that communication between the ROACH board and the workstation is able to guarantee in-order packet delivery without loss. A protocol capable of providing this functionality is TCP, however it is expensive to implement in logic and therefore is not a possibility for use over ROACH's 10GbE.

TCP networking would have to be provisioned over the PowerPC subsystem on the ROACH board, which is limited to 1Gbps. While 1Gbps is easily capable of satisfying the bandwidth requirements of transmitting the database to the ROACH board it will create a bottleneck when receiving indices to alignments if ROACH BLAST is run with a high sensitivity. As a result of these bandwidth requirements TCP networking is not an option until filters can be implemented in hardware to reduce the number of alignments returned to the workstation.

4.1.7 10Gbe with Push UDP Control

Since TCP networking is ruled out for this design UDP is the alternative. UDP does not guarantee in-order packet delivery or even packet delivery, therefore support for these features would have to deigned into the system. Fortunately the 10GbE network the ROACH boards are connected to has no packet loss which reduces the complexity of the code requiring only in-order delivery and checking that a packet was not lost so that the user can be notified.

In-order packet delivery can be implemented on top of the UDP protocol, however it was calculated that the amount of time that ROACH BLAST takes to process a packet of data is longer than the time required to ask for and begin receiving a new packet. A method of control to "pull" packets through the network is easier to implement, and since the latency is low, this method is sufficient to satisfy the bandwidth requirements.

The "push" UDP control system would assume that packets are continuously being sent and when buffers approach their full state a signal would be sent to the workstation to

slow down the rate of transmission. The greatest complexity with this design is the logic required to guarantee in-order delivery, and due to the acceptable and simpler alternative method of "pulling" data through the network "push" control was abandoned.

4.2 Hardware

The top level of the ROACH BLAST design contains the ROACH board support netlist, the BLAST core and logic to form an interface between them. The 10GbE core is provided by the ROACH tool chain and is built into the ROACH board support netlist using the method described in Chapter 3. The ROACH board support netlist provides 100MHz and 60MHz clocks with a reset line and 10GbE I/O signals to the BLAST core, the other signals are directed to the edge of the FPGA and constrained to pins by the constraints file produced by the ROACH tool chain.

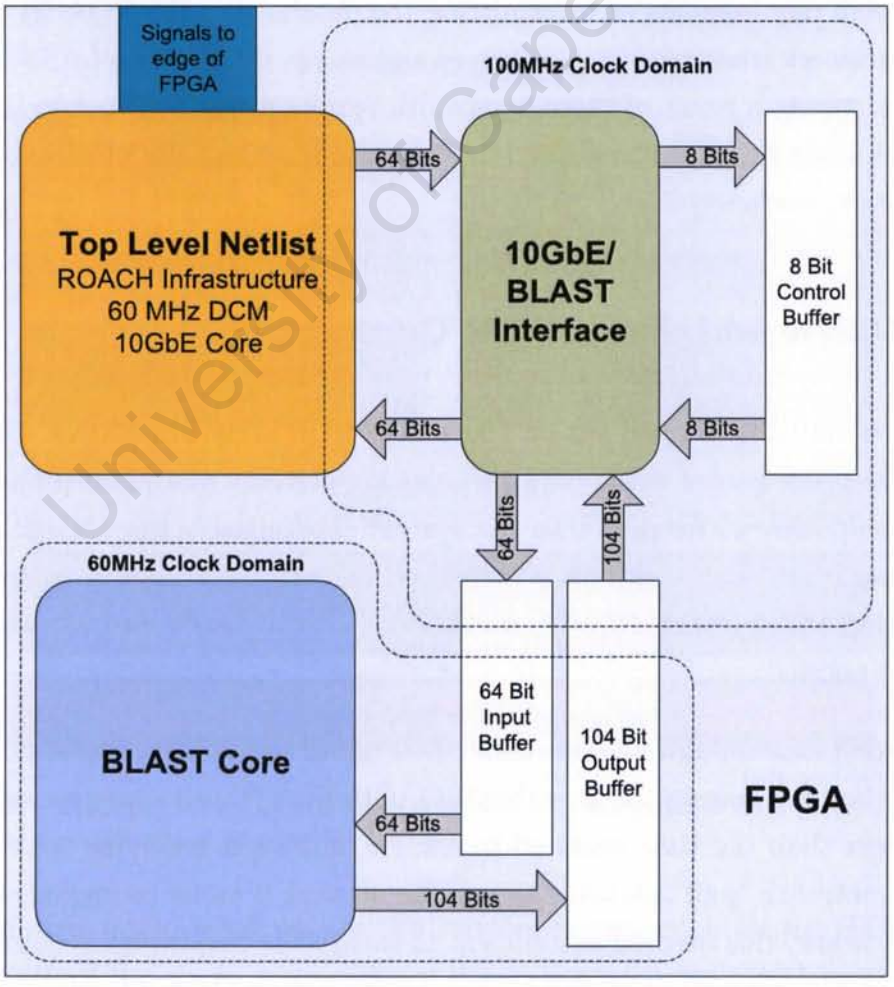


Figure 4.6 – Top level of the ROACH BLAST architecture showing the network infrastructure, network control and BLAST core in the FPGA.

The BLAST core receives its input through a Decoder which converts the frames produced by the 10GbE core into a compatible format and passes the words into the Detection Regions and Extension Controller. The Detection Regions contain the logic which processes the seed detection and extension stages of the BLAST algorithm and returns an index to an alignment with its raw score. All the Detection Regions must accept a new letter from the database into their buffers at the same time. If a Detection Region's buffer becomes full it stops the Decoder from sending new letters to all the Detection Regions until the full buffer has freed-up space for a new letter.

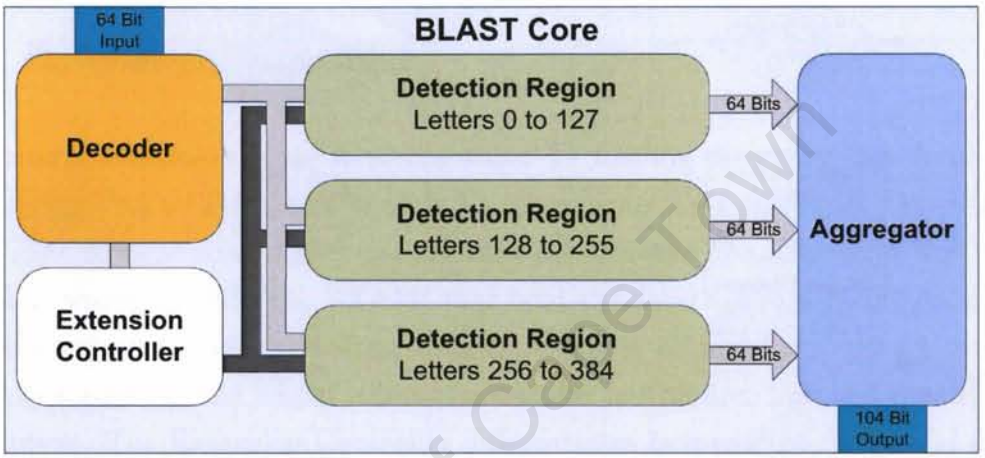


Figure 4.7 – The structure of the BLAST core. The implementation of ROACH BLAST on an SX95T FPGA contains 3 Detection Regions to process the alignments, a Decoder for global control and an Aggregator to collect the alignments from the Detection Regions. The Extension Controller loads query and database letters into the Extension Units inside the Detection Regions.

The alignments are collected from each of the Detection Regions by an Aggregator and placed in an output buffer while waiting to be transmitted back to the workstation. The 10GbE/BLAST interface logic combines the data produced by the BLAST core with control signals and formulates a packet which is written to the 10GbE core and transmitted to the workstation.

4.2.1 Decoder

The Decoder sits at the input of the BLAST core and receives the PLL Locked signal from the 60MHz DCM to reset the system and a 64 bit interface to read the data from the 10GbE control buffer. The first operation the Decoder performs is to break the 64 bit input into the 4 bit words that the instruction set uses. The backoff signal from the Detection Regions interacts with this process to stop new 4 bit words being produced.

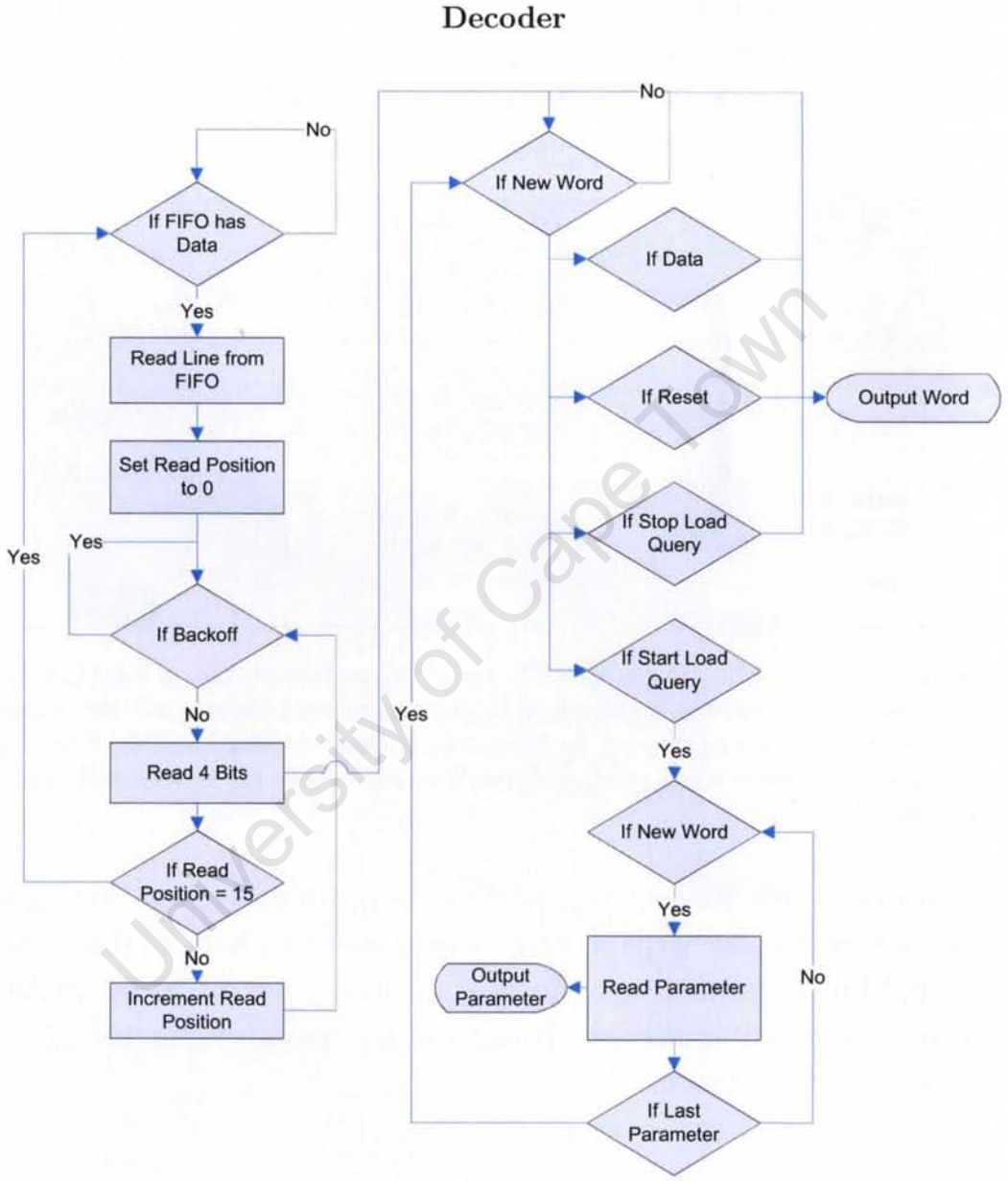


Figure 4.8 – A flow chart illustrating the operation of the Decoder. Data enters the Decoder from the 10GbE controller's input buffer and produces output for the Detection Regions and Extension Controller.

The 4 bit words undergo a decoding process where it is determined if they are instructions, parameters or data. A detailed description of the instruction set is provided in Appendix A. Instructions processed at this level effect the entire BLAST core, however some instructions are not processed here and are forwarded to regional control.

The output from the Decoder produces 4 bit words that have been formatted for compatibility with the remainder of the datapath along with the BLAST algorithm input parameters and a reset line.

4.2.2 Extension Controller

The Extension Controller is the local controller for loading the query and database into the Extension Unit's BRAMs. It accepts its input from the Decoder in 4 bit words and contains logic to determine if they are instructions or data. The Extension Unit BRAMs contain four words per line, the BRAMs that hold the database require the data to wrap around when they are full. The wrap around process is achieved through an overflow of the BRAM's address line which resets the counter to the first line and overwrites the oldest content. The Extension Controller differentiates between the query and database through instructions indicating the beginning and end of the query.

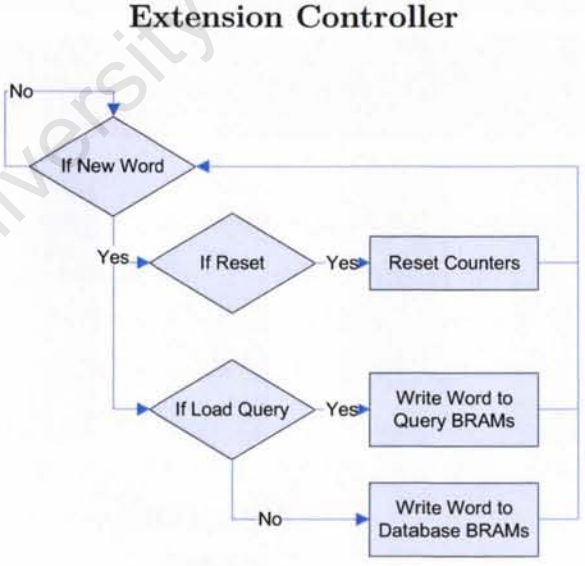


Figure 4.9 – A flow chart for the Extension Controller. Data enters the system from the decoder and is loaded into the Extension Unit's BRAMs.

Apart from writing data directly into the Extension Unit's BRAMs the Extension Controller produces a 31 bit counter to indicate to the Extension Units how many letters of the database have been written into their BRAMs. The Extension Units use this

counter to ensure that they stall for new data instead of reading invalid data out of BRAM.

4.2.3 Detection Regions

Detection Regions divide the query into sections of 128 letters and operate autonomously to perform the seed detection and extension stages of the BLAST algorithm. The need for Detection Regions is borne out of the critical path caused by the Seed Detection Array requesting a stall. The design of the Seed Detection Array requires that each element is able to request a stall, and the asserted stall signal must propagate to all the elements in the detection array and the Local Controller by the next cycle.

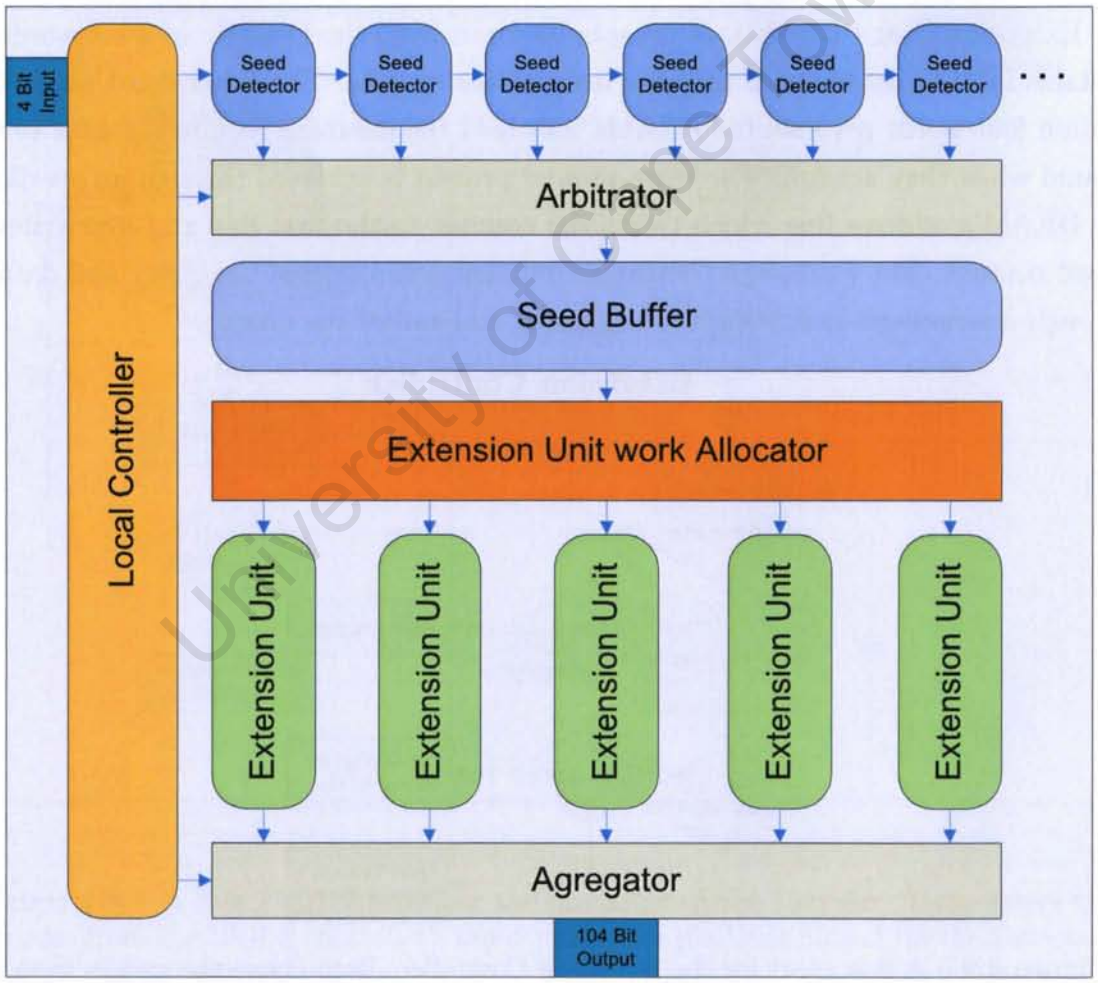


Figure 4.10 – The structure of a Detection Region. The Detection Regions in ROACH BLAST implemented on the SX95T contain 128 Seed Detectors and 8 Extension Units.

Although the Detection Regions were implemented to overcome the limitations imposed by the stall critical path they have the added benefit of improving the scalability of

the design. Since each Detection Region contains all the logic required to perform seed detection and extension on its section of the query they are able to operate autonomously. This allows more Detection Regions to be stamped out without much of an impact on clock speed or routability.

At the input of a Detection Region sits a Local Controller which passes data to the Seed Detection Array and provides control for the region. Below the Seed Detection Array is an Arbitrator which collects the seeds from the detection array and passes them to the Extension Units. The extended alignments are collected from the Extension Units by a Local Aggregator and placed in a buffer at the edge of the region while waiting to be serviced by the Global Aggregator.

4.2.4 Local Controller

A Local Controller sits at the input of each Detection Region and provides it with local control and input buffering. Local Controllers receive 4 bit words from the Decoder and place them into their buffer, if the buffer approaches full they assert a backoff signal to indicate to the Decoder that it must stop sending data or its buffer will be overrun.

The controller reads 4 bit words out of its buffer and decodes them into instructions to control the loading process for queries, reset the design or data to pass through onto the detection array. The controller also indicates to the Detection Region when the last letter of the database has been received so that it can notify its Aggregator to begin checking for the work complete signal.

The Local Controllers perform a similar function to the Extension Controller, however to ensure that extensions can be completed the Extension Controller does not have a buffer and loads letters directly into the BRAMs. This is necessary to prevent the Detection Regions from becoming deadlocked when the input buffers are full and there is not enough data in the BRAMs to complete extension and all the BRAMs are full. Although this is an unlikely situation by simply ensuring that the Detection Region buffers can hold as many letters of the database as there are letters in the query the risk of deadlock can be removed.

4.2.5 Seed Detection

The seed detection stage of the BLAST algorithm is performed in a systolic array where each letter of the query is held in one element. The database letters enter the array at

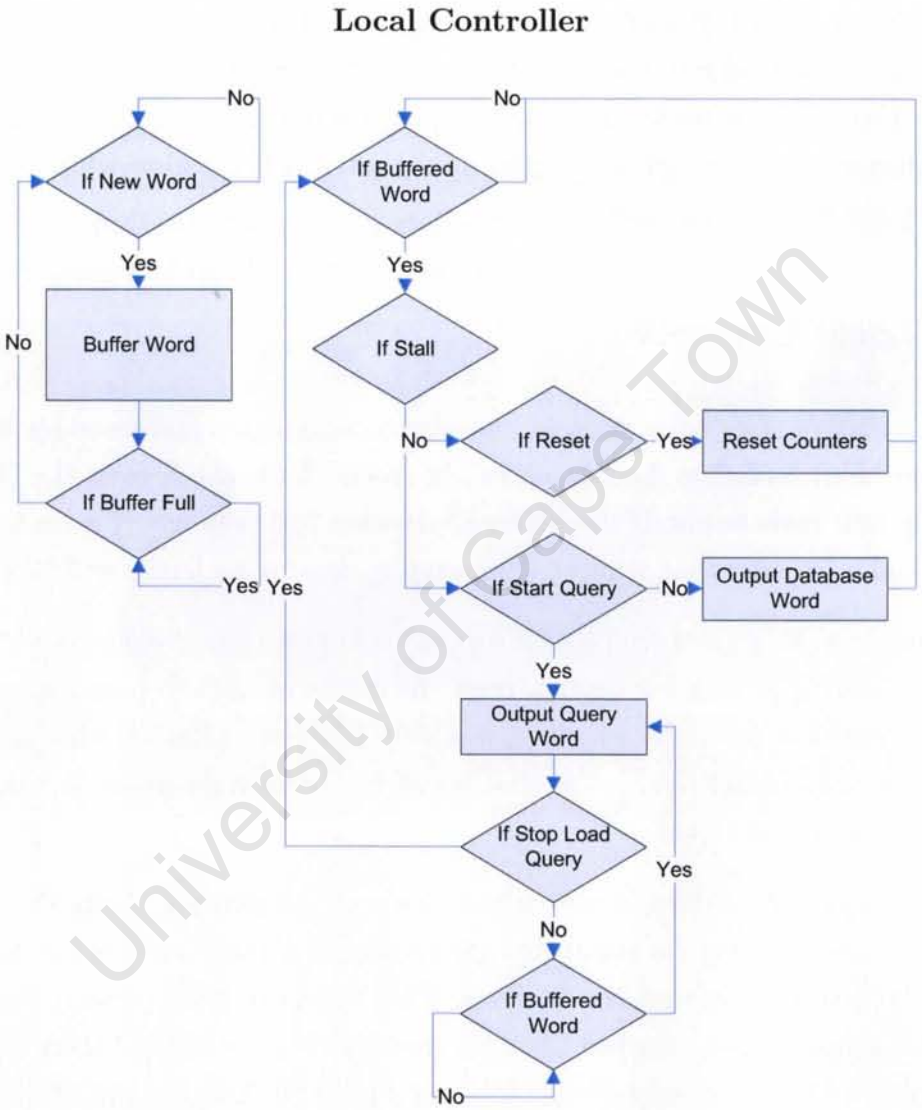


Figure 4.11 – The Local Controller's flow diagram. Data is collected from the Decoder and buffered in the controller. When data is available the controller reads a word from its buffer and processes it as either local control or places it on the Seed Detection Array.

the last letter of the query and move one letter down the array each cycle. During a cycle the elements compare their query letter to the passing database letter and if there is a match they increment their input score by 1 and place it on their output. If there is a mismatch they set their output to 0 and check if their input score is greater or equal to the specified word size. If the input is greater or equal to the word size the input is placed in the elements buffer and the Arbitrator is notified.

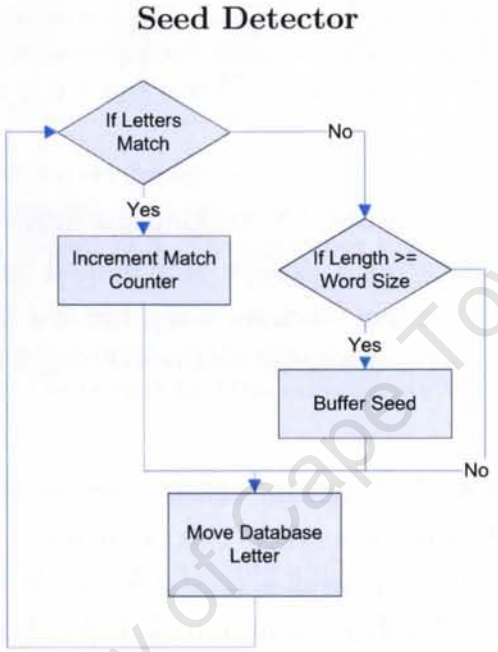


Figure 4.12 – A flow chart of the operation of the seed detection process.

The Arbitrator decodes the position information of the seed from its understanding of the position of the element in the array that produced it and a counter that tracks how many database letters have been placed on the detection array. While this is enough information to decode the position it would rely on the database letters not moving until the seed has been decoded or its database position would be offset by the number of cycles it has been delayed.

To prevent the detection array from stalling while there is a seed waiting to be serviced a delay counter is associated with the length while it is in the Seed Detection Element's buffer. The delay counter is incremented by 1 every time the database moves by one letter. The delay counter is also used to ensure that the seed doesn't spend so long waiting to be serviced that it misses the current database window in the Extension Units' BRAMs'. The delay counter is capped at a 127 "letter moves" delay which upon reaching causes the array to stall until the seed is serviced.

Due to the requirement of many small memories in each Seed Detection Element and

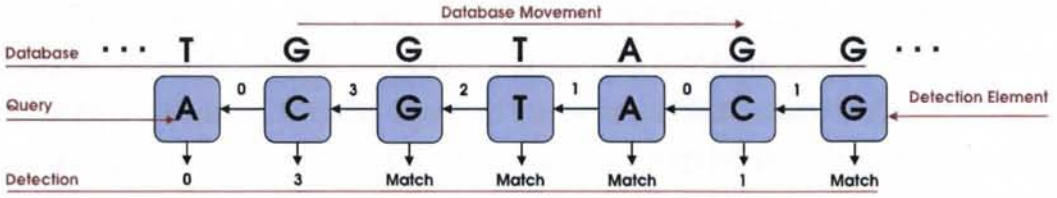


Figure 4.13 – The operation of the Seed Detection Array. The database moves across the elements one letter each cycle while the elements compare their query/subject pairs. If an element detects a match between its letters it increments its input score and outputs it to its neighbour. When an element detects a mismatch between its letters it checks if its input score is greater or equal to the specified word size and if it is the seed is reported to the Arbitrator.

their unstandardised design required for tracking the delay it is not efficient to buffer seeds in each element of the array, therefore only limited buffering is supported. Each element can hold a single seed and track its delay, but if a Detection Element which is already buffering a seed detects a second seed it will cause the array to stall until the first seed is serviced

Although the database moves through the array one cycle at a time the length counter passes through multiple elements in a cycle. This multi-element propagation of the length counter causes problems when the array is long and the signal would have to propagate from one end to the other. To find a compromise between multi-element propagation and clock frequency Reference Elements were introduced to stop the propagation and hold it over so that it can continue the next cycle.

The Reference Elements are placed every 8 Seed Detection Elements in the array and cause a stall whenever their query/subject letters match and there is a good chance that a seed will be found. If the Reference Elements only viewed the length counter in one direction they would have to trigger a stall even if only it and the element next to it found a match to ensure that the seed would be detected in the next cycle. This equates to a 1 in 16 probability of occurrence which is too high and would cause the array to often stall unnecessarily.

To counteract his undesirable behaviour a second track of length counters are setup in the opposite direction so that the Reference Elements can base its decision on the match score of the letters to come. By adding the two length counter scores plus 1 the Reference Element will know the length of the seed, provided it does not fall over multiple Reference Elements, and be able to decide if it is worth causing a stall to detect it or if it should be discarded. When a seed falls over multiple Reference Elements the array will stall to fully process the seed regardless of the specified word size. Each Reference Element that

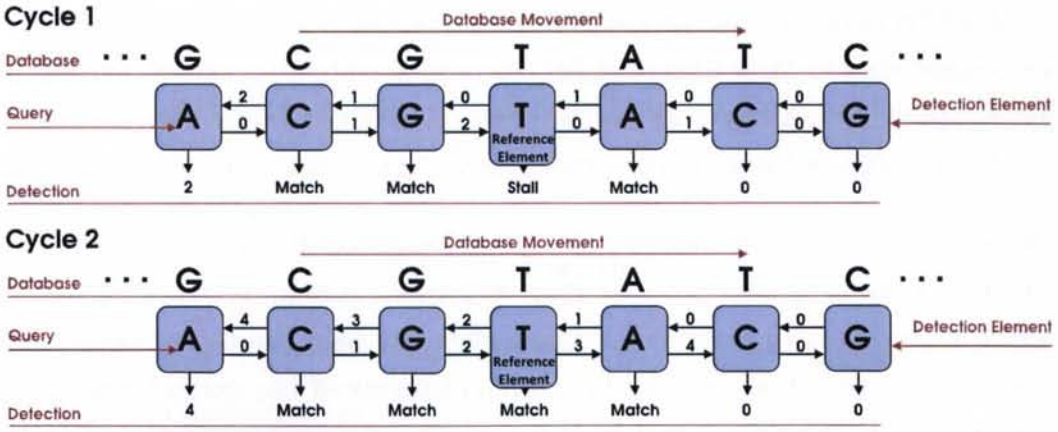


Figure 4.14 – An illustration of the operation of the Reference Element. In the first cycle the Seed Detection Elements score the matches as described in Figure 4.12, while the Reference Element fixes it output at 0. The Reference Element sees input of 1 and 2 and that its letters are a match. The Reference Element knows that there is a 4 letter seed lying across it and that another cycle is required for it to be detected, therefore it raises the stall line so the detection can continue the next cycle. The next cycle it increments and transfers its input scores to the output and the detection of the 4 letter seed completes.

the seed crossed will cause the Seed Detection Array to be stalled by another cycle.

Reference Elements only differ in their stalling ability and otherwise behave in the same manner as the Seed Detection Elements. The other elements in the Seed Detection Array that differ from the Seed Detection Elements lie at the boundaries of the Detection Regions. The database letters enter the Detection Regions at the last letter of the query and length counter signals propagate towards the first letter in each array. There are three types of Detection Regions, beginning, middle and end, which differ in how the detection array is initiated and terminated.

The first element that a database letter enters in the array which holds the last letters of the query is the Global Terminator. This element compares the query/subject letters and either sets its output to 0 or 1. This element is incapable of detecting a seed and does not communicate with the Arbitrator.

The other Detection Regions contain Local Termination Elements which determine if a seed lies over multiple regions. These elements hold the first letter in the Detection Region following them, and when the query/subject letters match they indicate to all other elements in their Detection Region that this is the case. The elements, based on their position in the array and their input, are able to determine if the seed being detected was started in the next Detection Region and if so they ignore the match. This ensures that only one Detection Region will find and extend a seed.

The element preceding the element which holds the first letter of the query is the Global First Element, this element always holds the seed detection termination symbol, which is required to ensure that seeds running to the end of the query are found due to seed indices being generated from a mismatch between the query and database.

An element before the first letter in the other Detection Regions which always causes a mismatch is not necessary however their elements are required to determine if a seed falling on them is worth extending. These elements hold the last seven letters of the database to exit the array and the last seven elements of the query from the previous Detection Region. Matches between these overlapping letters are passed to the First Element in the Detection Region and added to its input score for analysis. If the full overlapping region and the first query/database letters in the region match an extension is started regardless of the word size.

The length reported to the Arbitrator will always be what the First Element saw on its input and never include the score from the overlapping region. This is required for the Arbitrator to decode the position information correctly and the work to discover the true length of the seed will be placed on the Extension Units.

This design requirement causes the ROACH BLAST architecture to potentially introduce seeds with a word size lower than the user specified, which leads to an increased sensitivity in some cases. Despite seeds entering with a shorter length than specified they will still be required to pass all other thresholds and filters to make it into the final report. Any of these seeds that pass all the thresholds and filters can be passed through an additional filter to remove them during report generation.

4.2.6 Arbitrator

The Arbitrator is placed between the Seed Detection Array and the Extension Units facilitating the flow of seeds between them. Each Seed Detection Element presents 2 status bits, 7 length bits and 7 delay bits to the Arbitrator.

The Arbitrator contains four priority encoders which monitor both states of the Detection Element's status bits and looks down the Seed Detection Array from both directions. This process is performed simultaneously to reduce delays and allow a seed to be moved from the detection array to the Arbitrator's buffer every cycle.

The priority encoders each produce the position of the element which they suggest be serviced next, if a priority encoder cannot find an element in need of servicing it returns

an out-of-bounds number. The Arbitrator processes the output through a second level priority encoder which first looks for an element causing a stall and then for elements with data. The Arbitrator records the position of the element that was serviced the previous cycle and if the same element is presented by a priority encoder in the current cycle its status is ignored.

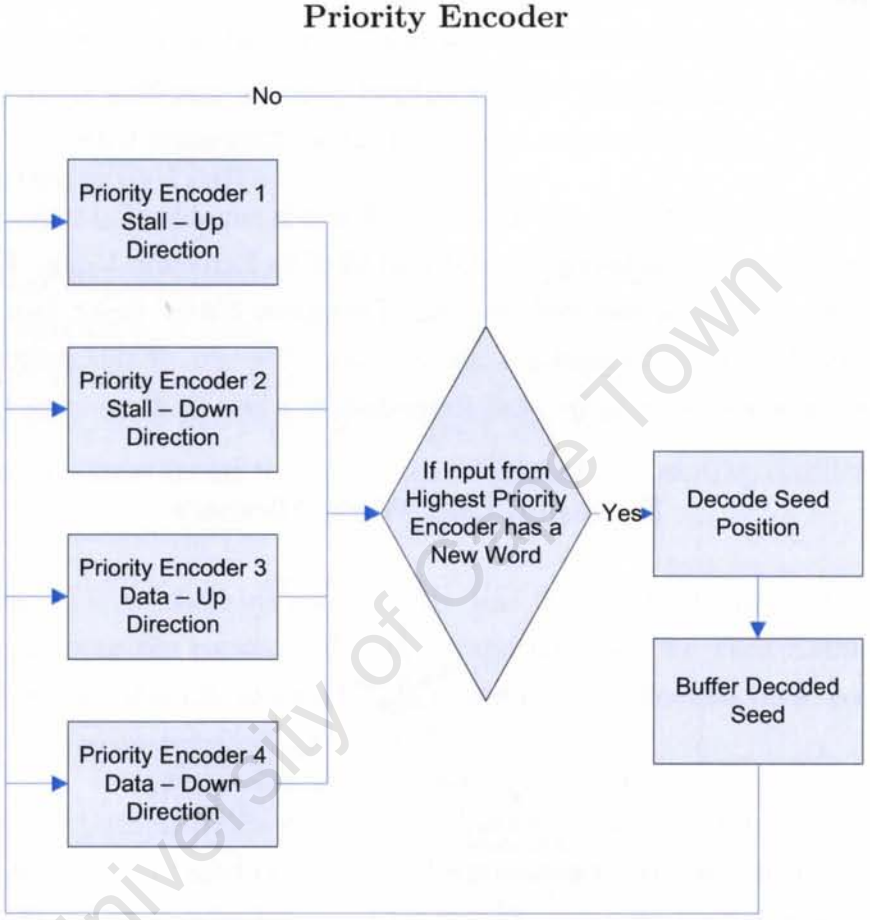


Figure 4.15 – The priority encoder is part of the Arbitrator logic and determines which Seed Detection Element to service next. Four priority encoders select an element for servicing, then a second priority encoder selects the best element proposed by the first round of priority encoders. The selected seed is decoded and buffered.

This procedure is required for the system to move a unique seed every cycle due to the latency in notifying a Seed Detection Element that its seed has been serviced. By implementing a priority encoder in both directions for each priority level a unique seed can be moved every cycle regardless of its priority.

Once a seed has been selected for servicing its Detection Element is notified and it is moved for decoding from a relative position into an absolute position. The seeds absolute query position is calculated by adding the Detection Region's offset to the Seed Detection Element's position and the database position in calculated by combining the

Local Controller's database counter, the delay representing the amount of time the seed has been waiting to be serviced and the position of the Seed Detection Element which produced the seed.

The Arbitrator also contains a lookup table which is used to decode the position of the subject in the database that the seed belongs to. The absolute database position is compared to this table and the subject position is returned. The starting position of the seed in the query and database with its subject position and length are placed in the Arbitrator's FIFO while it waits to be serviced by an Extension Unit.

The logic that allocates work to the Extension Units is capable of starting an extension every cycle by internally registering the status of all of its Extension Units. The Arbitrator places the next seed to be serviced onto the Extension Units' input bus regardless of whether or not there is an Extension Unit available. The aim of this design is to ensure that data is ready and able to be read immediately when an Extension Unit becomes available.

Extension Unit Work Allocator

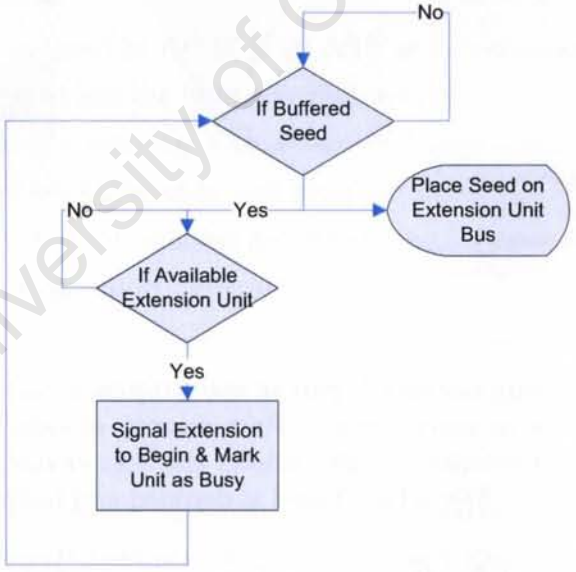


Figure 4.16 – The Extension Unit work allocator forms part of the Arbitrator and manages the process of transferring seeds to Extension Units. When a buffered seed is detected it is placed on the Extension Unit input bus while it waits for an Extension Unit to become available. When an Extension Unit is available the Arbitrator signals the Extension Unit to read the seed and begin extension.

When data is ready to be serviced the Arbitrator searches for an unused Extension Unit in its list, and when one is free raises the begin extension signal associated with the Extension Unit then marks it as busy. When an Extension Unit is finished extending a

seed it signals to the Arbitrator that is is done and the Extension Unit status register is updated.

The Arbitrator also produces a signal to indicate to the Local Aggregator if seeds or extensions are currently being processed. This signal is used by the Aggregator to determine if the system has finished processing so it can indicate that the Detection Region has completed its work.

4.2.7 Extension Units

Each Detection Region contains eight Extension Units which perform simultaneous bidirectional extension of seeds. There are two versions of the Extension Units, one for SX and the other for LX series Virtex 5 FPGAs. The SX Extension Unit contains four BRAMs which hold two copies of the query and database so that each direction of the extension can have access to its own set of BRAMs preventing conflicting memory accesses.

The LX series FPGAs contains significantly less BRAM than their SX counterparts therefore duplicating the copies of the query and database for each Extension Unit is infeasible. However, the LX series FPGAs contain more slices so these resources were put to work to overcome the BRAM limitations.

The LX series Extension Units implement additional logic to determine if accesses to memory will conflict and based on this analysis is able to delay the starting of extension in one direction for up to 2 cycles to ensure that all the memory accesses will be interleaved. This design marginally increases the slice usage which ruled it out for implementation on the SX95T FPGA, which can contain more Extension Units when implementing the SX version.

Extension Units receive their data from the Arbitrator while they are in a wait state, upon receiving the begin extension signal they load the seed sitting on the Extension Unit input bus and encode it for processing. The absolute positions for the boundaries of the seed are calculated and encoded into an address and position format corresponding to the data's position in BRAM.

The Extension Units then progress through a number of states to setup the memory and retrieve the first of the data required for processing. In the LX implementation calculations are performed during this processes to determine if there will be memory

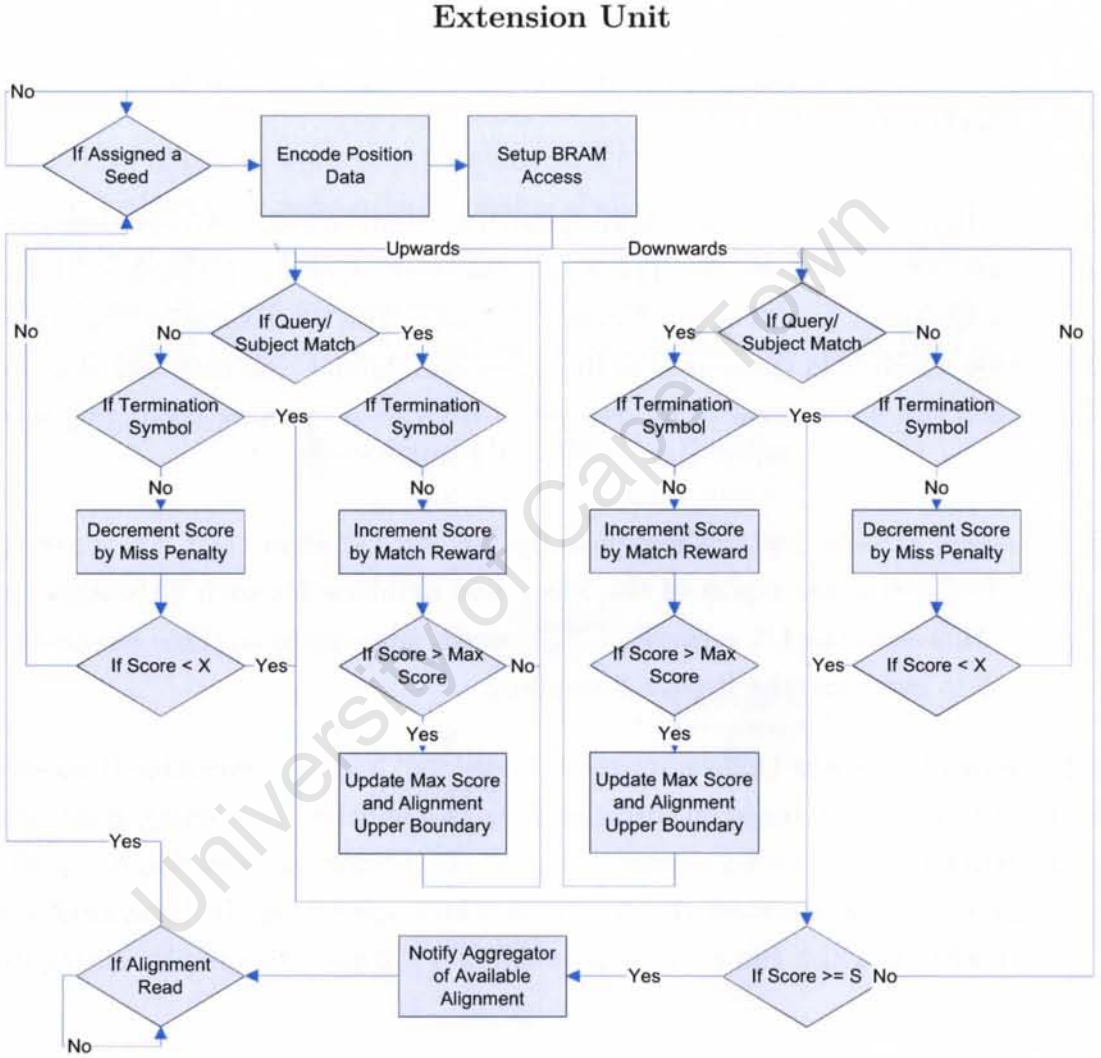


Figure 4.17 – A flow chart illustrating the extension process a seed undergoes when allocated to an Extension Unit.

access conflicts and the appropriate delays are introduced in one direction of the extension to ensure that accesses are interleaved.

With the setup complete extension begins with a letter from the query and subject being compared to each other in both directions every cycle. If the letters match the score is incremented by the user specified match reward, if they mismatch the score is decremented by the mismatch penalty.

The Extension Units track the maximum score and the edge of the alignment in each direction throughout the process, the edge of the alignment is only moved when the score rises above the maximum and not when it reaches the maximum. Extension is terminated when the alignment's score drops off by the specified x value or when the boundary of the query or subject is detected. The boundary of the query and database is indicated by special separator symbols which the Extension Unit recognises and uses to terminate processing.

Raw scores of the alignments are compared to the s cut-off parameter and either discarded, the Extension Unit notifies the Arbitrator that it is free, or decoded into its absolute position after which the Aggregator is notified that there is a alignment ready for servicing. Once the Aggregator has read the alignment the Extension Unit notifies the Arbitrator that it is free and returns to the wait state.

4.2.8 Aggregator

The Aggregator sits at the end of the datapath and collects output from the Extension Units and Detection Regions. The Aggregators in the Detection Regions differ from the Global Aggregator by implementing additional logic to indicate when the Detection Region has completed its work.

Both types of Aggregators monitor status bits which indicate when data is available for servicing, when data is serviced its origin is notified to update its status bit. The data is moved from its originating element into the Aggregator's FIFO where components further down the datapath will read them out for further processing

The additional work done in logic by the Local Aggregator monitors its Local Controller for the transmission done signal, and when received enters a state where it waits for the Arbitrator to signal that it has completed processing the seeds and extensions. The end of transmission instruction is placed 128 letters after the last letter of the database to

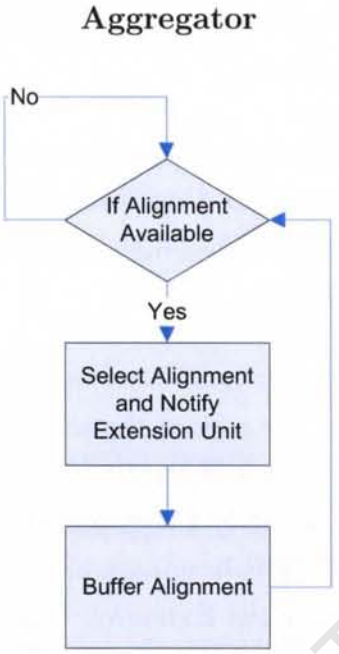


Figure 4.18 – A flow chart illustrating the operation of an Aggregator. The Local Aggregator differs slightly from this implementation with additional logic to determine when its Detection Region has completed processing.

ensure that all possible seeds have been detected by the Seed Detection Array and the elements have been able to signal to the Arbitrator that they have data.

From this point on it can be guaranteed that the work done signal to the Aggregator will only be raised once the last alignment has either been discarded or read by the Aggregator. When the Aggregator receives this signal it generates a zero length alignment, which is invalid and impossible to create through normal processing, and places it onto its output FIFO. The zero length alignment will be used by the network control to determine when all Detection Regions have completed processing.

4.2.9 10GbE Control

The 10GbE control system facilitates the flow of data between the 10GbE core and the BLAST core and runs on a 100MHz clock. It receives data 64 bits at a time and places them into an input buffer capable of storing 32kB of data. When the 10GbE core receives an end of frame signal it generates a control packet to ask for the next packet, this creates a simple control system which guarantees in-order delivery and if a packet gets lost causes the execution to stall thus notifying the user.

Despite the use of the "pulling" data through the network it is capable of delivering data

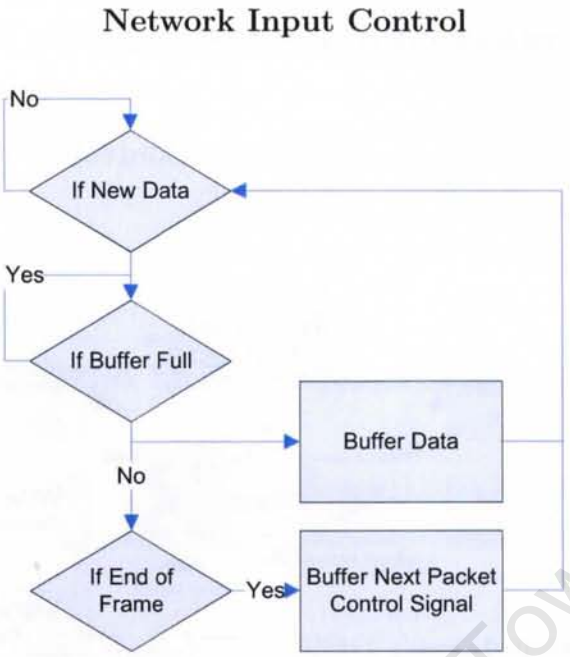


Figure 4.19 – The "pull" network control and buffering. When data arrives it is placed into an input buffer which crosses the clock domain to the BLAST core. Next packet signals are only generated when the last 64 bits of the UDP packet are transferred to the input buffer.

to the BLAST core faster that it can process it under the best case scenario. This results in the input buffer asserting its full signal and slowing down the rate at which packets are requested. Data is read out of the input FIFO by the Decoder in the BLAST core, the FIFO crosses the 100MHz/60MHz clock domain, and next packet symbols are placed in the control FIFO.

ROACH UDP Packet Structure

```
          Upper Integer                                 Lower Integer
63    56 55    48 47    40 39    32 | 31    24 23    16 15     8 7      0 Bits
dddddddd dddddddd dddddddd dddddddd | pppppppp pppppppp pppppppp pppppppp Line 1
------en --ssssss ssssssss ssssssss | -------- ------qq qqqqqqql lllllll1 Line 2

          d  Starting position of the alignment in the database
          p  Alignment's subject position in the database
          q  Starting position of the alignment in the query
          l  Length of the alignment
          s  Alignment Raw Score
          n  Next packet
          e  End of work
          -  Reserved
```

Figure 4.20 – The output UDP packet payload structure produced by ROACH BLAST.

Output from the BLAST core and control data are combined into a 4 integer packet format and written into the 10GbE core's transmit FIFO over two lines. Whenever a packet contains a next packet bit the end of frame flag is set and the UDP packet

is transmitted. Data without control is allowed to full the UDP packet before it is transmitted.

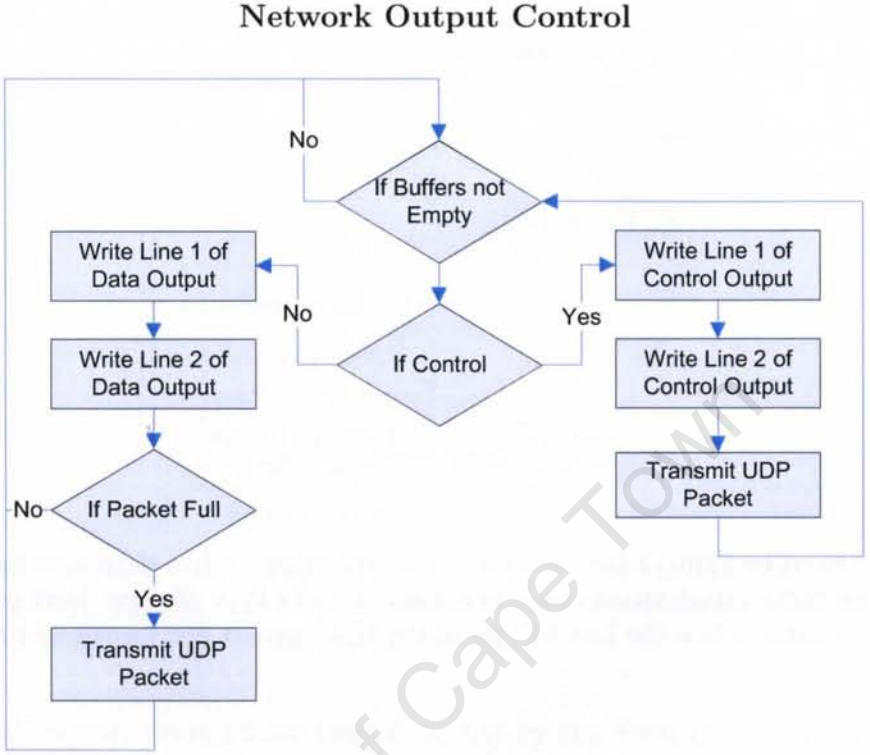


Figure 4.21 – The output logic for the network interleaves the control and data buffers and produces a standardised packet format for the software.

The 10GbE control also performs the important role of determining and indicating to the workstation when the BLAST core has completed processing its work load. The lengths of alignments produced by the BLAST core are checked for a zero value, and when detected a zero length counter is incremented. The value of the zero length counter represents how many Detection Regions have competed their processing. When the value is the same as the number of Detection Regions in the design the end of run bit is set in the packet and the contents of the 10GbE core's transmit FIFO sent.

4.2.10 10GbE Core

The ROACH tool chain provides a 10GbE core with support for UDP that makes efficient used of the FPGA's resources. This 10GbE core was implemented in the ROACH BLAST design by placing its yellow block in the SimuLink model for the top level of the ROACH board.

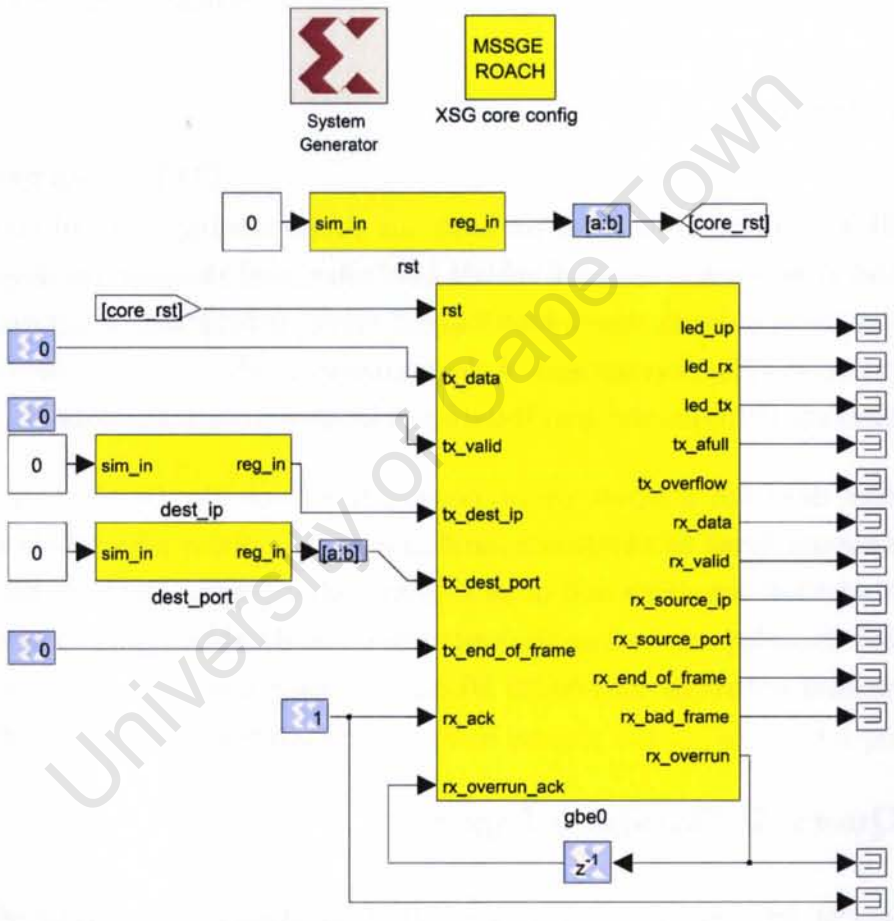


Figure 4.22 – The MATLAB/SimuLink design file for the ROACH BLAST project.

The 10GbE core is programmed during runtime over the OPB bus from BORPH running on the PowerPC subsystem. The configuration information supplied is: its MAC address, IP address, listening port, the IP address to transmit to and the port.

The 10GbE interface inputs and outputs data 64 bits at a time and uses a flag to signal when the payload of a packet has been completely read out of its receive FIFO. The 10GbE core's transmit and receive FIFOs are capable of storing the a UDP packet with jumbo frames enabled. Detailed information on the 10GbE core can be found on the ROACH website [16].

4.3 Software

ROACH BLAST uses software to perform the preprocessing tasks of converting the database and streaming it to the ROACH hardware, and the post-processing tasks of: decoding alignment indices, expect filtering and report generation. The primary focus of this dissertation was the development of the hardware accelerator, therefore only software systems necessary to verify and benchmark the hardware were developed.

This software does not support preprocessing functions like DUST to remove areas of low complexity from the sequences and does not perform gapped extension. It is envisaged that these functions and other features missing from ROACH BLAST will be implemented by either expanding the software or, preferably, integrating the ROACH BLAST accelerator directly into NCBI BLAST's code.

4.3.1 Query & Database Input

Before ROACH BLAST can process a database it must be converted into a binary format that can be memory mapped. This process is similar to NCBI BLAST's database formatting tool and only needs to be run once per database.

The database is provided to ROACH BLAST in FASTA format, which represents the letters in characters with a header line separating the subjects. ROACH BLAST reads the FASTA file and splits the subject header information from the sequence creating a memory mapped file of headers and another of the database letters in character format for decoding indices during report generation. The character file is encoded into another memory mapped file where the letters are transformed into the 4 bit words supported by

ROACH BLAST, the sequences in this file are separated by the 4 bit subject separator words.

If the database contains more than 2 billion letters it is split into sets of 2 billion letters with a 4 bit encoded and character memory mapped file for each set. An index file accompanies the memory mapped files and contains integers representing: the full length of the database, the number of subjects it contains, the position of the last letter in each of the sets, the length of the sets and the number of sequences in each of the sets. This information is used to decode the indices returned from the ROACH board to overcome its limitation of a maximum database length of approximately 2 billion letters supported in hardware.

4.3.2 Network I/O

Transmitting the database to the ROACH hardware is simple due the database formatting process converting it into the compatible format beforehand. The database is preceded by configuration words to setup the parameters and load the query. The words are packed into 8192 byte payloads and transmitted via UDP to the ROACH boards whenever the ROACH hardware asks for a packet.

The software listens for UDP packets sent from the ROACH hardware and places the payload into an integer array for decoding. The architecture specifies that if a packet contains control it will be in the second to last integer of the payload, therefore this integer is checked for the next packet bit and if detected triggers the next packet to be sent. The the data is stripped from the control integer in the packet and placed in the decode buffer.

4.3.3 Decoding & Expect Filtering

Decoding utilises multiple threads to assist the software in keeping up with the hardware. The decode threads read an index payload from the decode buffer and split it into individual alignments. The alignments are partially decoded so that their expect value can be calculated, and if they pass they continue on to be fully decoded, otherwise they are discarded.

The decoded alignments are placed into an array which holds each of the subjects and their alignments. When the end of set bit is received by the software a flag is set to

indicate that when decoding is complete the alignments that have passed expect can undergo further processing. The alignments from each subject are sorted by expect and duplicates are removed, finally the end of set flag is raised to indicate that the next set can be sent or that all the work is done and the report can be generated.

4.3.4 Report Generation

Report generation begins by sorting the subjects by their alignment with the lowest expect, this will be the first alignment in the subjects array. The expect summary is generated off this information, and report generation continues with writing the details of the alignments.

Report generation uses the memory mapped file containing the sequence letters in character format to reconstruct their value from the reduced 4 bit instructions ROACH BLAST uses. This allows for any degenerate codes in the sequences to be correctly placed back into the report and saves the software from having to decode the 4 bit words back into a characters.

Once the details of the alignments are written a summary of the statistics for the run is generated and appended to the report, once ROACH BLAST has written the report the system has completed its work and is ready for the next query.

4.4 Conclusion

The literature review identified that the systolic array approach to seed detection is capable of producing seeds quickly and accurately in a manner that is compatible with the original BLAST specification. With the basic design centred around a systolic array for seed detection various architectures were considered in an attempt to find a well balanced system that maps efficiently to the targeted FPGA.

The final design operates on a 4 bit instruction set with a common datapath for instructions and data. The query is loaded into the BLAST core with each letter placed into an element in the Seed Detection Array. The Seed Detection Array is divided into three 128 element self contained regions each containing 8 Extension Units.

The BLAST core returns indices to alignments which are transmitted back to the workstation over 10GbE for decoding. The software performs the preprocessing tasks of

formatting the database and managing the transmission of data to the ROACH hardware. The post-processing involves decoding the returned indices, additional filtering and report generation.

ROACH BLAST was designed to conform to the original BLAST specification and can produce identical output when run under the same conditions. The next chapter discusses the verification of ROACH BLAST to ensure it is operating as expected and that it does conform to the BLAST algorithm.

Chapter 5

Design Verification

The verification chapter covers a variety of tests to determine if the ROACH BLAST design meets the specifications of the BLAST algorithm and to ensure that is is possible to implement it on the targeted FPGAs. Simulations were conducted to verify the integrity of the design concept and to ensure the components in the design function correctly. These simulations were performed with C++ for the proof of concept tests and ISIM for behavioural and post-route HDL simulation. The simulations were able to identify most faults in the design and produce models of the design which were proven accurate when tested in hardware.

After the design was known to function correctly it was determined how much logic could be implemented on the FPGA to maximise usage. It was discovered that by manipulating ISE's default compilation settings improvements can be made in both speed and chip usage. Through the compilation parameters exploration an additional 15MHz in clock frequency was achieved with a 5% reduction in chip usage.

The final design was verified in hardware against exemplar data consisting of patterns and sequences with known alignments. The pattern tests were used to determine if the system operated correctly under best and worst case and scenarios while contrived query/database sets were used to test boundary conditions.

Real data was run through the system and compared to NCBI BLAST to determine how well the system performed against a modern implementation of the algorithm. NCBI BLAST includes modifications to the original BLAST algorithm which improve its performance, therefore it was expected the output would not match exactly. Despite ROACH BLAST's and NCBI BLAST's differences the alignments were similar, however

an in-depth study into their correlation was not performed since this version of ROACH BLAST is not designed to comply with the NCBI BLAST standard.

5.1 Simulations

The simulations discussed in this section were used to determine if the concept of the Seed Detection Array was sound and to verify that the components written in VHDL were functioning correctly. C++ was used to simulate the Seed Detection Array concept and ISIM to simulate the VHDL in both behavioural and post-route states. The output from the simulations was sufficient to identify and correct faults in the design and resulted in an architecture that works reliably in hardware.

5.1.1 C++ Concept Simulations

The first simulations were conducted before work began on implementing the design to determine if the proposed Seed Detection Array would function correctly. These simulations were performed by writing a C++ program which reads in a query and subject and produces indices to the seeds which are verified by hand and against a conventional seed detection algorithm.

The results from these simulations showed that the multi-element propagation seed detection and merging array was capable of identifying seeds correctly. Based on this positive outcome work began on implementing the design in HDL.

5.1.2 Behavioural Simulation

Behavioural simulation is the fastest simulation that ISIM performs, however it is the least accurate. Behavioural simulation analyses the HDL code before synthesis and serves as first look into whether the code is functioning as intended. These simulations were conducted throughout development to test each of the components in the design and check their boundary conditions.

Apart from datasets designed to probe specific conditions in a component tests were run to verify the system as a whole. These tests were primarily based on sequences and databases randomly generated then slightly modified to interact with boundary

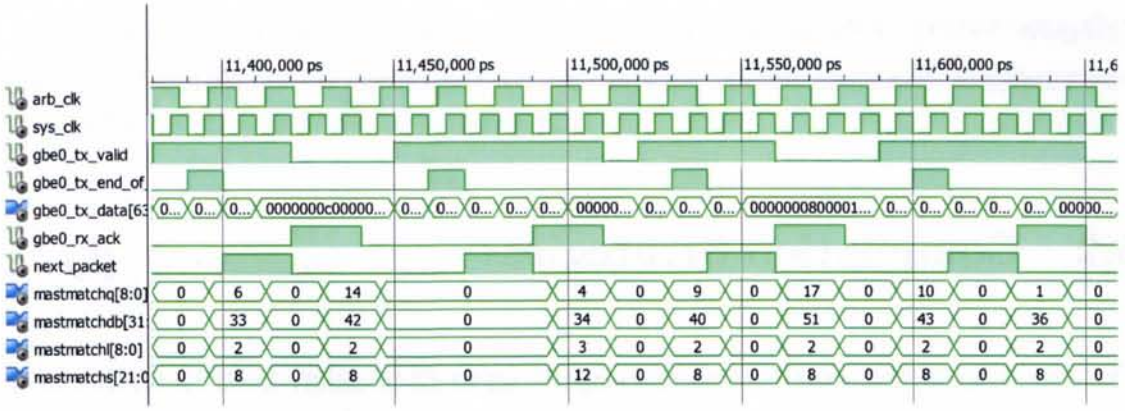


Figure 5.1 – A behavioural simulation waveform produced by ISIM. The waveform shows ROACH BLAST returning indices to detected alignments.

conditions. Queries and databases of complete matches and mismatches were also run to assess the impact on buffers and stalling mechanisms.

Behavioural simulation also proved useful when hardware testing revealed faults in the design, it was fast enough to simulate the conditions under which logical errors would appear, however it was unable to detect errors introduced through compilation due to its simulation model being based on the HDL code. To identify the errors introduced through compilation the more accurate post-route simulation was required.

5.1.3 Post-route Simulation

Post-route simulation produces the most accurate simulations possible with ISIM, these simulations are performed on fully placed and routed projects and take into account the timing requirements of the design. The major problems with the post-route simulations are: the amount of time it takes to generate the model, the speed of simulation and the amount of memory it requires.

To simulate the complete ROACH BLAST design requires approximately 5 hours for the compilation of the test set and approximatively 10 minutes to simulate 10µs while using 16GB of memory. The performance of post-route simulations rules it out for general testing making it primarily useful for confirming the timing of the design and verifying the HDL code has been correctly compiled.

The ROACH BLAST project used post-route simulation to verify timing on one of the query/database sets of random numbers with known alignments. A few timing errors were identified and corrected, otherwise the simulations indicated the design was correct.

Extensive testing of the architecture was not performed in these simulations due to their long runtimes, therefore testing continued with the design implemented in hardware.

5.2 Compilation Strategies

The aim of exploring compilation strategies was to fit as many Seed Detection Elements and Extension Units in the FPGA as possible while maintaining a clock frequency of 60MHz. The design used in the exploration of the strategies was the final architecture without the 10GbE core and before testing in hardware.

ISE presents a large set of options for optimising each stage of the compilation process, broadly, the options provide customisations for the: design's speed, area usage, power usage and speed of compilation. The options are divided between the processes in the compilation chain with some working in conjunction.

Due to the large number of options and the combinations tested only a brief description of the methodology and the selected parameters are discussed. Exploration began by optimising the design for speed and watching the impact on area, to do this options with descriptions indicating they improved performance were tested one at a time. The most promising of these options were tried in conjunction and if chip usage became too high they were combined with options to reduce area. The result of this testing revealed four options that are able to improve performance and keep chip usage reasonable.

Table 5.1 – Changes to compilation settings from default.

Process	Option	Default	Selected
Synthesis	Optimization Effort	Normal	High
Map	Placer Extra Effort	None	Continue on Impossible
	Global Optimization	Off	Area
Place & Route	Extra Effort	None	Continue on Impossible

Global Optimisation in the *Map* process can be set to area, speed or power and by default is off. It was determined that by setting it to speed a significant increase in clock frequency can be achieved with a small drop in area at the cost of a longer compile time. The option also proved useful when the complete design was compiled with timing driven placement. The complete design could not fit on the FPGA when it was set to the speed setting, but on the area setting it not only was able to fit but also meet timing constraints.

The other compilation options changed in the design all increase the runtime their respective process spends searching for a better solution. The complete list of changes from the defaults are presented in Table 5.1.

5.3 Chip Usage

The final design of ROACH BLAST comes in two implementations, one for the Xilinx SX series and the other for the LX series FPGAs. The SX implementation uses almost twice the BRAM of the LX implementation, but is easier to route. It is possible to fit the LX design on the SX95T FPGA with the same number of Detection Regions as the LX110T, however the LX design requires additional slices for routing and is unable to meet timing constraints. The SX design uses additional BRAM to simplify the Extension Units by providing a BRAM for the query and database in both directions of extension. The LX FPGAs cannot afford this additional BRAM usage so a more complex memory access interleaving technique is employed so that only two BRAMs are required per Extension Unit.

5.3.1 SX95T

The most effort was placed on fitting the SX design into the SX95T FPGA due to the lab not containing any LX110T boards. Due to the SX95T containing more BRAM than the design requires the greatest focus was placed on how to use it to simplify the logic and reduce slice usage or improve timing

Table 5.2 – Chip utilization of the SX95T implementation of ROACH BLAST. The design contains three 128 element Detection Arrays with 8 non-interleaved Extension Units per array.

SX95T	Used	Available	Utilization
Slice Registers	29 581	58 880	50%
Slice LUTs	53 804	58 880	91%
Occupied Slices	14 683	14 720	99%
BlockRAM/FIFO	167	244	68%

The result was a architecture that utilises 91% of the slices to implement the logic and an additional 8% for route throughs resulting in 99% chip usage. This design is bounded by the number of slices available and even with the double BRAM usage memory is not a problem.

5.3.2 LX110T

The design was initially created with interleaved memory access in the Extension Units and then modified for the SX implementation, however the interleaved Extension Units were kept up-to-date. SX ROACH BLAST cannot fit on the LX110T under the same conditions as the SX95T, so the number of Extension Units per 128 Seed Detectors would have to be dropped. This would create a bottleneck at extension when the sensitivity of the BLAST algorithm is set high causing the system to slow down. The other option would be to remove an entire Detection Region resulting in large amounts of unused slices in a slice heavy design.

Table 5.3 – Chip utilization of the LX110T implementation of ROACH BLAST. The design contains three 128 element Detection Arrays with 8 interleaved Extension Units per array.

LX110T	Used	Available	Utilization
Slice Registers	30 108	69 120	43%
Slice LUTs	53 746	69 120	77%
Occupied Slices	16 502	17 208	95%
BlockRAM/FIFO	102	148	68%

ROACH BLAST is a much better fit for the LX series of Xilinx FPGAs when interleaved Extension Units are used. Comparing the design implemented on the SX95T and the LX110T FPGAs it is evident that the slice/BRAM usage ratio on the LX110T is superior. The LX110T design requires 9 percentage points more slices than BRAM while the SX95T design requires 23, from this observation it is clear the design maps better to the LX110T than the SX95T resulting is fewer wasted resources.

5.3.3 Large Virtex 5 FPGAs

To determine how well the design scales on Virtex 5 FPGAs the largest SX and LX series chips were selected and a design was compiled for each. The largest SX FPGA is the SX240T and it is capable of fitting seven 128 element Detection Regions with 8 Extension Units per region. The SX240T is 2.5x bigger than the SX95T and fits a 2.3x bigger design with 78% slice LUT usage, thus indicating linear scaling.

The LX330T FPGA contains the most slices in the Virtex 5 family, however it does not maintain the same slice to BRAM ratio of the LX110T. When ROACH BLAST is implemented on the LX330T with interleaved Extension Units 10 Detection Regions

Table 5.4 – Chip utilization of the SX240T and LX330T implementations of ROACH BLAST.

(a) SX240T running the SX implementation of ROACH BLAST with seven 128 element Detection Arrays and 8 Extension Units per array.

SX240T	Used	Available	Utilization
Slice Registers	64 259	149 760	42%
Slice LUTs	117 445	149 760	78%
Occupied Slices	35 821	37 440	95%
BlockRAM/FIFO	366	516	70%

(b) LX330T running the LX implementation of ROACH BLAST with ten 128 element Detection Arrays and 8 Extension Units per array.

LX330T	Used	Available	Utilization
Slice Registers	92 389	207 360	44%
Slice LUTs	187 770	207 360	90%
Occupied Slices	51 471	51 840	99%
BlockRAM/FIFO	185	324	87%

can be accommodated, which is approximately 3.5x more than was implemented on the LX110T. The LX330T contains 3x more slices and 2x more BRAM than the LX110T.

Both of these implementations are not for practical application due to the pin locations not being constrained and minor changes to the code required to support longer queries. For the design to be practical an additional 2 bits must be added to the query index from the Arbitrator onwards and the Local Controller FIFOs each require an additional BRAM. Both of these designs met constraints for the BLAST core but fail to meet a constraint for the XAUI infrastructure, this is likely due to the pin locations not being constrained.

5.4 Output Verification

The output from ROACH BLAST was verified at various stages but in all cases uses the architecture implemented on a ROACH board with a SX95T FPGA. The verification process began by running the same tests performed during behavioural simulation and some additional tests to increase coverage.

Early tests only analysed the output of the ROACH boards with no further processing and were based on contrived data. The purpose of these tests were to determine if the

hardware was functioning as intended and not to verify the complete system. After the hardware to determined to be operating correctly the complete system was compared, with real data, to NCBI BLAST through their output.

5.4.1 Targeted Tests

The aim of these tests were to determine if the boundary conditions, like: back-off and send more requests, FIFO full signals, seed detection over multiple Detection Regions or Reference Elements, in the hardware and the system as a whole were operating correctly. The tests were conducted by loading sequences with known alignments and comparing the output to exemplar alignments. The results of these tests showed the design to be working correctly at the boundary conditions and that the hardware produces output that conforms to the BLAST algorithm.

Further targeted tests consisted of all match sets to test the worst case scenario where the system would continuously generate matches and long alignments fulling buffers and creating a backlog throughout the system. All mismatch sets were run to test the best case scenario where the system would operate at its peak efficiency to determine if the network was capable of supplying data at the required speed. These tests also showed the design to be working as expected. With confidence in the reliability of the system testing moved onto real data with comparisons to NCBI BLAST.

5.4.2 Real Data Tests

Testing with real data primarily provides a facility for measuring performance under real conditions, this is covered in Chapter 6, however it is important to ensure the system works correctly under these conditions and to identify bottlenecks and optimisations that can be made to the design.

Running real data through the system revealed an issue with the software not being able to keep up with the hardware when the sensitivity of the algorithm is set high. Various optimisations in hardware were introduced to reduce the post-processing load in software, and in conjunction with multiple threads the software is able to keep up with the hardware.

Due to the nature of the filtering of alignments in the algorithm it was discovered that large amounts of the data that the workstation is handling during post-processing will

be discarded, and that it is possible to perform the filtering in hardware. The filtering was not implemented in hardware for this version of ROACH BLAST as it would have required too much logic to fit onto the FPGA without reducing the supported query length or number of Extension Units. This optimisation is left for future work as the current short length of the query supported by ROACH BLAST means that the backlog only becomes an issue under the most extreme conditions.

The output of the final ROACH BLAST system was compared to NCBI BLAST with the understanding that they would be slightly different. The first observation was that NCBI BLAST does not specify thresholds in raw scores, while the original algorithm does. Despite this it was possible, through trial an error, to determine a suitable bit score to correspond to the raw score supplied to ROACH BLAST.

ROACH BLAST

```
> SRR088923.667671 HWUSI-EAS1591_61H93:8:32:16432:6608 length=94
           Length = 94

 Score = 137 bits (312), Expect = 8e-31
 Identities = 83/87 (95%)

Query:  183 TGCCGAAAGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAACCAATGCCTGATGCGA 242
           ||||||| |||||||||||||||||||||||||||||||||| ||||||||||||||||
Sbjct:    1 TGCCGAATGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAATCAATGCCTGATGCGA 60

Query:  243 CGCTGTTGCGTCTTATCAGGCCTACAA 269
           ||||||| ||| ||||||||||||||||
Sbjct:   61 CGCTGTCGCGGCTTATCAGGCCTACAA 87
```

NCBI BLAST

```
> SRR088923.667671 HWUSI-EAS1591_61H93:8:32:16432:6608 length=94
Length=94

 Score = 137 bits (312), Expect = 9e-31
 Identities = 83/87 (95%), Gaps = 0/87 (0%) Strand=Plus/Plus

Query 183 TGCCGAAAGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAACCAATGCCTGATGCGA 242
           ||||||| |||||||||||||||||||||||||||||||||| ||||||||||||||||
Sbjct   1 TGCCGAATGTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCAATCAATGCCTGATGCGA 60

Query 243 CGCTGTTGCGTCTTATCAGGCCTACAA 269
           ||||||| ||| ||||||||||||||||
Sbjct  61 CGCTGTCGCGGCTTATCAGGCCTACAA 87
```

Figure 5.2 – Alignments taken from ROACH and NCBI BLAST reports. The search was performed against sequence GI Number: 1419508 at a word size of 6.

With the thresholds set to appropriate equivalents it was possible to achieve alignments that mostly matched, and when the alignments were of the same length and position the raw and bit scores also matched. Despite the correct calculation of the bit score

and all other parameters required for the calculation of expect the numbers did not match, although they were close. ROACH BLAST implements the formula defined by the original BLAST algorithm, therefore it was deduced that NCBI BLAST uses a different method for calculating expect.

These differences are attributed to the evolution of the original BLAST into NCBI BLAST and it is expected that similar modifications can be made to ROACH BLAST. ROACH BLAST, due to an architectural necessity, does depart from the BLAST algorithm when seeds fall across Detection Regions and the word size is larger than 8. This causes additional seeds to be detected and pass into extension, however they must still satisfy all other thresholds to make it into the report.

As a result of this behaviour when expect is set high a large number of these additional alignments will find their way into the report. If this is a significant problem to the microbiologist it can be fixed by additional post-processing. While generating the report a counter can be employed to track the longest exact matching region between the query and subject. If it is found that after writing out the alignment in the report the longest exact matching region is less than the specified word size then the alignment can be discarded and its entry overwritten.

5.5 Conclusion

The ROACH BLAST system was verified throughout its development starting with simulations to test the concept written in C++. The VHDL code was verified through behavioural and post-route simulations and was found to be functioning correctly and to the BLAST specification.

With the ROACH BLAST hardware verified in simulation effort was placed into fitting as large a design as possible into the FPGA . The Virtex 5 SX95T was targeted and in conjunction with various compilation parameters three Detection Regions with 128 Seed Detectors and 8 Extension Units in each were placed in the FPGA.

Finally, the complete system was tested and its output compared to NCBI BLAST. The output differed slightly, however, this was expected due to modifications to the original BLAST algorithm that NCBI BLAST implements. The tests showed that ROACH BLAST functions as designed and that it is capable of producing output that conforms to the original BLAST algorithm.

Next, the verified design underwent benchmarking to determine how well it performed against NCBI BLAST, the de facto standard BLAST implementation.

Chapter 6

Benchmarking

After validation it is important to determine how well ROACH BLAST performs on real data compared to the industry standard implementation, NCBI BLAST. The benchmarking was performed by selecting a representative set of data consisting of two databases and 45 queries so that tests could be designed to probe specific strengths and weaknesses of both versions of BLAST.

The results show that ROACH BLAST's runtime is only effected by the length of the database while NCBI BLAST's runtime is effected by the length of the query and database. ROACH BLAST is shown to outperform NCBI BLAST when the word size is low, but since ROACH BLAST has a constant runtime for a database it is projected that when the query is sufficiently long it will always be faster.

If the runtimes are power normalised ROACH BLAST is able to break-even with NCBI BLAST, when running against the smaller database, however due to the non-linear behaviour of word size on NCBI BLAST, ROACH BLAST is unable to consistently break-even on the large database. Even though ROACH BLAST in these tests was at best only able to beak-even it is important to consider the inefficiency of the ROACH boards PSU and the amount of redundant hardware for radio astronomy on the board. If a reconfigurable computer more suited to the BLAST algorithm was used the performance-per-watt ratio could be greatly improved.

6.1 Methodology

In this section the process for selecting the queries and database is presented and the purpose of each test is described. The data was chosen to diversely represent its structure

not diverse species. The tests analyse the impact of: word size, query length, database length and query/subject similarity.

6.1.1 Query Selection

The query database was derived by extracting all sequences shorter than 384 letters from the *ecoli.fasta* database provided by UCT's CBIO group. 383 letters is the maximum query length supported by the hardware therefore all longer queries had to be discarded from the test. The remaining queries were divided into 3 subsets representing length as follows:

- 0-127bp Short Set
- 128-255bp Medium Set
- 256-383bp Long Set

From each of these sets 15 queries were selected to represent similarity between the query and database for their length. To do this each set was run against both databases and reports were generated for each query. The size of the report was used to judge the queries' similarity to the databases.

From each of the three length subsets 15 queries were selected using the size of the reports as an indicator of similarity. 5 species that displayed high similarity, 5 average similarity and 5 low similarity in both databases were selected. This created the complete set of 45 queries for testing that represent length and similarity across both databases.

By diving the queries into 3 sets along similarity each set would have 5 queries in each of the length sets so that the effect of length would be averaged out in the test, similarity is averaged in the same manner when dividing by length. Through this test set structure it is possible to run tests where a single attribute can be isolated and the others averaged.

6.1.2 Database Selection

The databases were selected based on two criteria, total length of the database and the length of the subjects in the database. The different total lengths of the databases are used to determine the relationship between total length and runtime while the subject's

Table 6.1 – Test queries similarity and length groupings. In each table the queries are grouped in sets from the high to low similarity. The "Short DB" and "Long DB" columns represent the similarity between the query and database with the higher the number the higher the similarity.

Short Set

GI Number	Short DB	Long DB	Length
256032408	6872	1259	21
174434	563	1010	77
43179	467	888	77
174462	451	1122	78
81335926	429	821	121
157881801	96	265	77
24987761	93	221	28
28373677	92	373	35
28373678	92	373	35
42779	91	339	89
270346326	1	0	8
270346327	1	0	8
270346328	1	0	8
270346329	1	0	8
270346330	1	0	8
Average	**617**	**445**	**45**

(a) Queries with lengths between 0-127

Medium Set

GI Number	Short DB	Long DB	Length
1419506	6057	2322	238
1419503	6055	2322	238
1835732	4571	2134	205
1419505	4446	1570	147
1419507	4446	1570	147
209769217	135	146	238
209769219	135	146	238
209769221	135	146	238
209769223	135	146	238
209769225	135	146	238
7330934	1	1	223
7330954	1	3	142
7330957	1	3	160
14278862	1	2	166
42492	1	2	221
Average	**1750**	**711**	**205**

(b) Queries with lengths between 128-255

Long Set

GI Number	Short DB	Long DB	Length
1419508	8165	2778	329
1419517	5342	1923	319
1419524	5309	1897	319
1419518	5306	1892	318
42734	4604	1864	301
209750445	158	212	361
209750443	158	212	361
209750441	158	212	361
209750439	158	212	361
209750437	158	212	361
7330862	1	4	317
7330874	1	6	272
145301781	1	1	286
240119203	1	3	301
42711	1	3	351
Average	**1968**	**762**	**328**

(c) Queries with lengths between 256-383

length is used to determine if there is a negative impact on runtime with a database consisting of short reads, sequences less than 100 letters long.

Based on these criteria UCT's CBIO group provided a 7.3GB database, *prokaryotes_not_ecoli.fasta*, and a 760MB database, *SRR088923.fasta*. The *prokaryotes _not_ecoli.fasta* database contains subjects with a variety of lengths up to millions of letters long. This database was created from the *prokaryotes.fasta* database by removing ecoli sequences because they were used to create the database from which the query sequences were selected. The *SRR088923.fasta* is the short read database with subjects of lengths 94 and 100.

6.1.3 Test Setup

All NCBI BLAST tests are run on a single core of a 3.0GHz Xeon 5450 workstation with 16GB of RAM, ROACH BLAST is driven from the same workstation during its runs. All queries are loaded into ROACH BLAST and processed one at a time regardless of number of unused elements in the detection array, this means that in all of these tests ROACH BLAST is not operating at its peak efficiency. This methodology was selected so that one-to-one comparison between the results from NCBI and ROACH BLAST could be made.

The first test is to determine the impact of word size on NCBI and ROACH BLAST. The test was executed by running all 45 queries against both databases and varying the word size from 4 to 31. An initial run of a single query was permitted for both NCBI and ROACH BLAST so that the software could memory map and load the database files into memory, this run was not timed. By running the full set of queries against both databases the effects of query length, subject length, database length and similarity are all averaged.

The second test analyses the impact of query length, this is performed by dividing the 45 queries into 3 sets based on length. The short query set has an average length of 45 letters, the medium set 205 letters and long set 328 letters. The three test sets were run with a word size of 6 against both databases after memory mapping. A word size of 6 was chosen because the first test indicated it to be the point of intersection for runtime in both databases between NCBI and ROACH BLAST. The sets in this test average the effects of subject length, database length and similarity while fixing the word size.

Test three illustrates the effect that similarity has on runtime. Once again this test was performed by diving the 45 queries into 3 sets as in test two, except by similarity. The

three sets represent high, average and low similarity and were run against both databases with a word size of 6. The sets in this test average query length, subject length and database length.

After the above three tests were complete the behaviour of ROACH BLAST could be projected so a fourth test was run to determine the query length at which ROACH BLAST would break-even with NCBI BLAST for various word sizes. The test queries were selected from the *ecoli.fasta* database and were chosen to have the same runtime in NCBI BLAST as the projected runtime in ROACH BLAST for a given word size. The selected query lengths and word sizes at which break-even is projected to occur in the short read database are as follows:

- w6 - 147bp
- w7 - 301bp
- w8 - 651bp
- w9 - 1051bp
- w10 - 2001bp
- w11 - 9106bp
- w12 - 24196bp
- w13 - 65510bp
- w14 - 204604bp
- w18 - 338534bp
- w20 - 372438bp

These queries were each run 3 times against both databases with NCBI BLAST and the average runtime recorded.

The final test attempts to determine the performance-per-watt of ROACH BLAST compared to NCBI BLAST. The methodology for this test is complex due to the ROACH board requiring a computer to host the databases, drive the board and generate reports, however one workstation can serve multiple ROACH boards.

Based on this consideration it was determined that the head node of the cluster, which would exist in both NCBI and ROACH implementations, could perform ROACH support routines and that only sections of the algorithm executed on the ROACH boards would be scaled by power. Furthermore considerations made with regards to the inefficiency of the PSU powering the ROACH board compared to the workstation, the fact that only part of the ROACH BLAST detection array is being used, that the power-to-watt ratio only makes sense in a cluster and that the NCBI tests are only being run on one core of the Xeon, lends this test to indicating where power is used in the system rather that of definitively indicating performance-per-watt.

To indicate the relative power use of different parts of the ROACH BLAST system only the stages of the algorithm that execute on the ROACH boards are normalised by power. To scale these stages of the algorithm the ratio of power usage between the workstation and ROACH board is measured under load.

6.2 Results

The following subsections contains the results of the tests described in the methodology. The results are presented in graphs with explanations of their content and observations, the data which the graphs are based on are included in the data pack.

In all of the graphs NCBI BLAST is represented by lines and ROACH BLAST by bars. ROACH BLAST's bar graphs are divided into sections representing runtimes for different stages of the algorithm. When analysing the short read databases the database did not need to be divided into multiple sets and could be completed in a single run.

In the short database graphs ROACH BLAST's runtime consists of "ROACH Runtime" and "Data Processing". "ROACH Runtime" is the amount of time the ROACH board was actively processing data while "Data Processing" is the amount of time the workstation required to process any backlog and generate the report.

With the large database the total execution is split into four sets by the software due to limitations of the hardware. These sets are run consecutively with seamless transitions between them. The runtime for each of these sets are represented in the bar graphs with the data processing overhead and report generation at the end.

6.2.1 Word Size

The following graphs illustrate the impact word size has on NCBI and ROACH BLAST. Figure 6.1a shows the impact of word size on the short read database, and Figure 6.1b the impact on the large database.

The graphs clearly indicate that NCBI BLAST performs much better on the less sensitive higher word sizes while ROACH BLAST performs well on the low word sizes. Notice how ROACH BLAST's runtime remains relatively constant while varying the word size and only begins to increase with word sizes less than 6.

6.2.2 Query Length

These graphs illustrate the total runtime of the three length based sets. Figure 6.2a plots the effect query length has on the short read database, and Figure 6.2b the effect on the large database.

In both graphs there is a clear trend with NCBI BLAST's runtime increasing with query length, while ROACH BLAST's runtime remains constant.

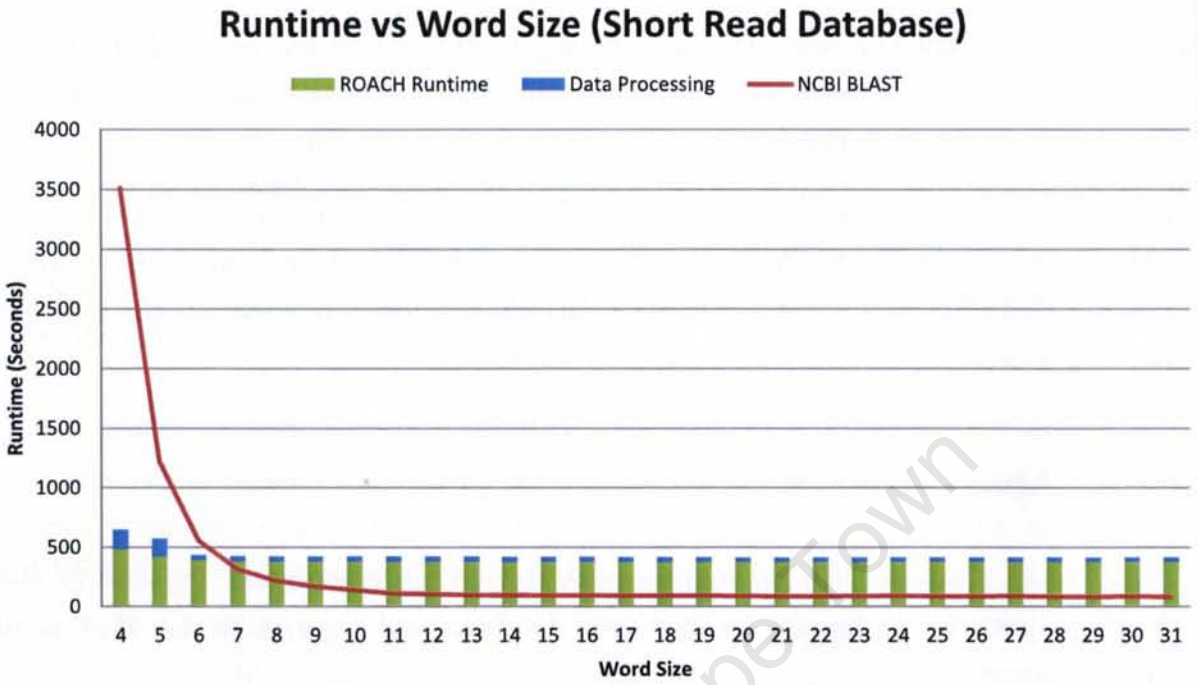
6.2.3 Similarity

The following graphs demonstrate the impact of similarity on ROACH and NCBI BLAST. Figure 6.3a displays the result of varying similarity on the short read database and Figure 6.3b the effect on the large database.

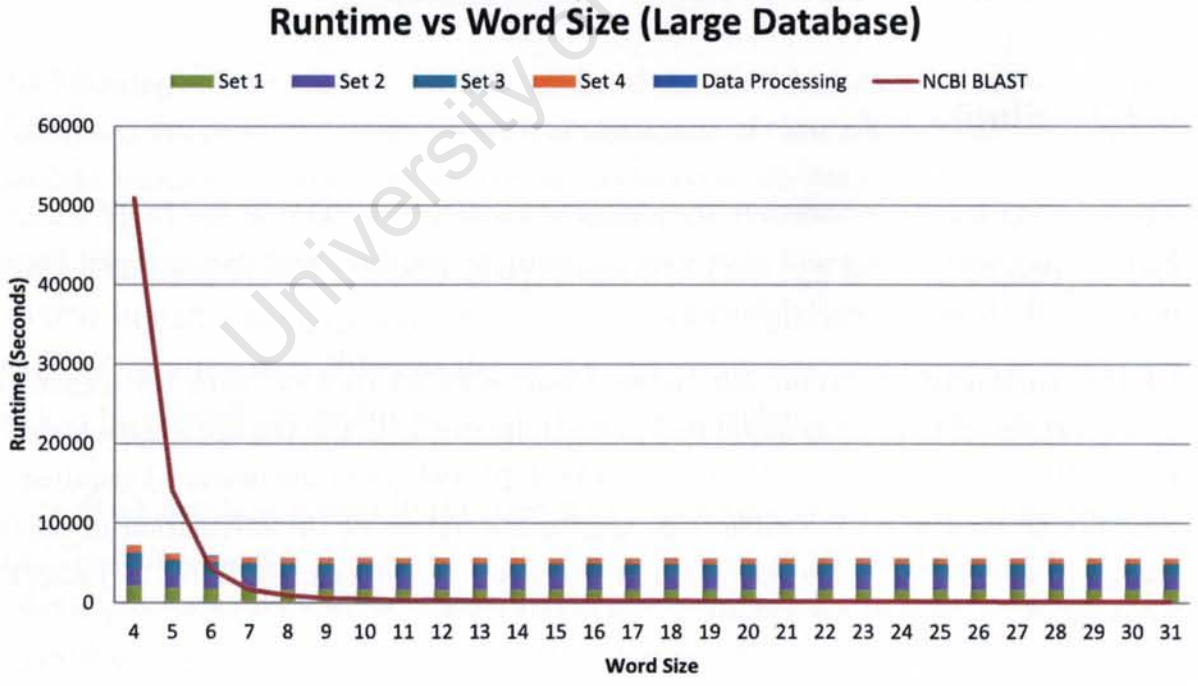
Varying similarity reveals no discernible trend on NCBI BLAST, with the somewhat erratic behaviour most likely attributed to slight differences in the average lengths between the similarity sets. However, from the ROACH BLAST plots the overhead required to generate larger reports is visible, but, due to the nature of the calculation of expect similarity is unlikely to become a significant factor in determining ROACH BLAST's runtime.

6.2.4 Break-even

From the trends established for ROACH BLAST in the above results it is projected that provided the query can fit on the ROACH board it will have a constant runtime

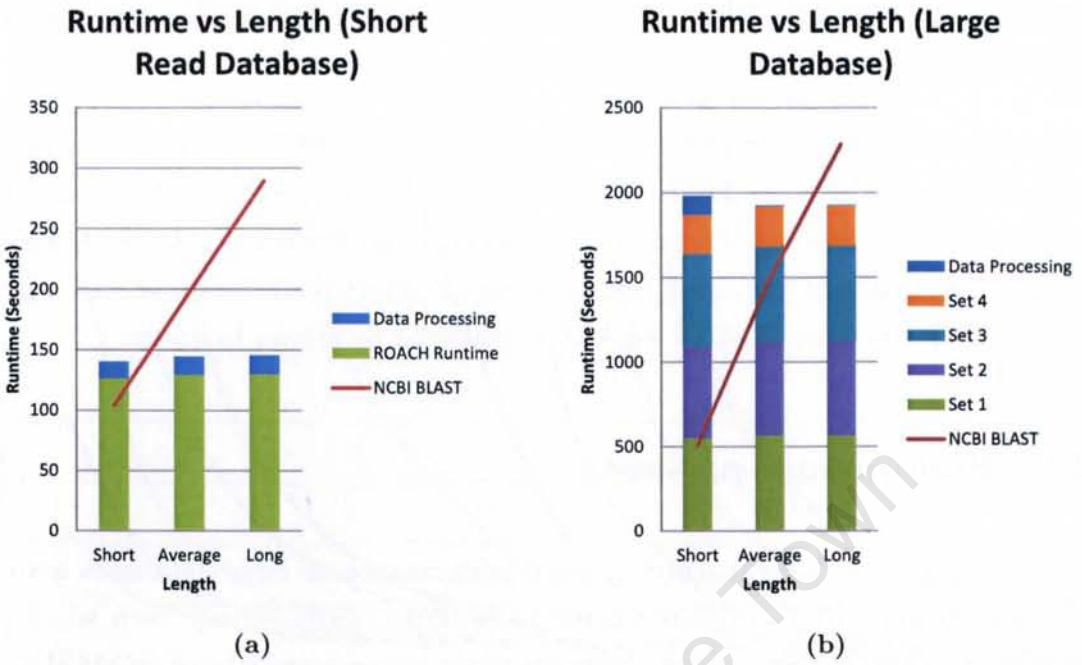


(a)



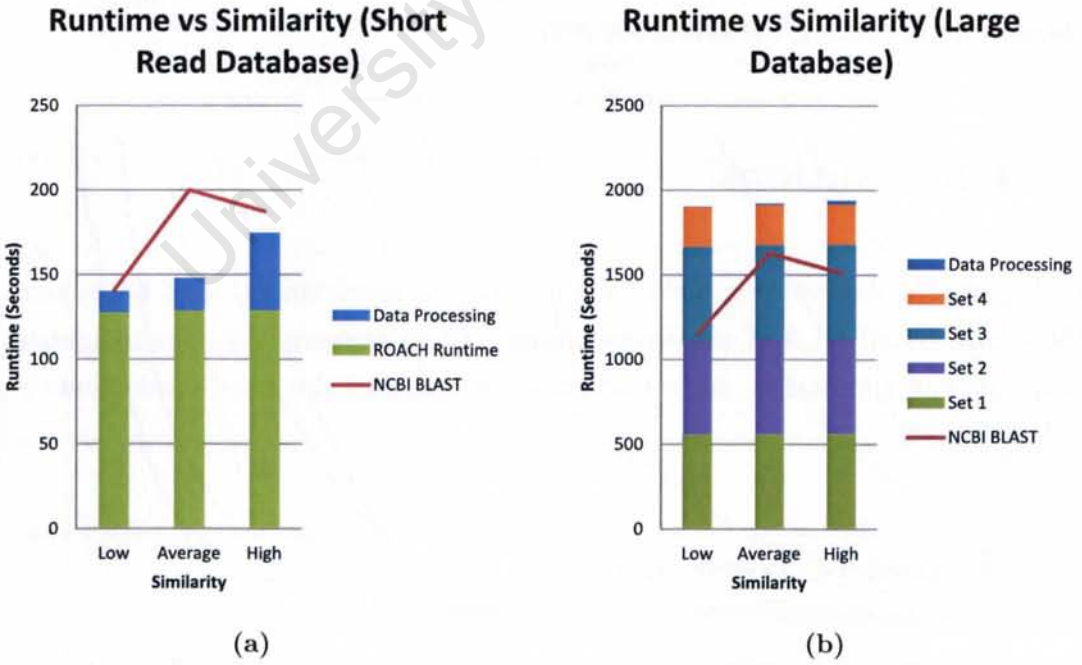
(b)

Figure 6.1 – Test 1, the effect of varying word size on runtime. ROACH BLAST and NCBI BLAST break-even at word size 6, ROACH BLAST outperforms NCBI BLAST on word sizes below 6.



(a) (b)

Figure 6.2 – Test 2, the effect of query length on runtime. NCBI BLAST displays a clear trend of runtime increasing with query length while ROACH BLAST's runtime remains unaffected.



(a) (b)

Figure 6.3 – Test 3, the effect of query/database similarity on runtime. NCBI and ROACH BLAST are both largely unaffected by similarity, however the additional overhead to generate larger reports is visible in ROACH BLAST.

regardless of the query length, subject length, and similarity. Based on these trends Figures 6.4a and 6.4b were constructed to illustrate the word sizes and query lengths at which ROACH and NCBI BLAST would break-even.

From the previous results it was expected that NCBI BLAST would perform better in tests operating on the large database and thus ROACH BLAST would require a longer query length to break-even. Figure 6.4b shows that the intersection is much higher up the exponential part of the curve compared to the short read database, but when the query is sufficiently long ROACH BLAST is expected to always be faster.

6.2.5 Performance-per-watt

Due to NCBI BLAST's comparatively good performance on large databases with word sizes above 6 ROACH BLAST is unable to perform competitively even when power consumption is taken into account. However when power is considered ROACH BLAST is able to break-even with NCBI BLAST on the much smaller short read database.

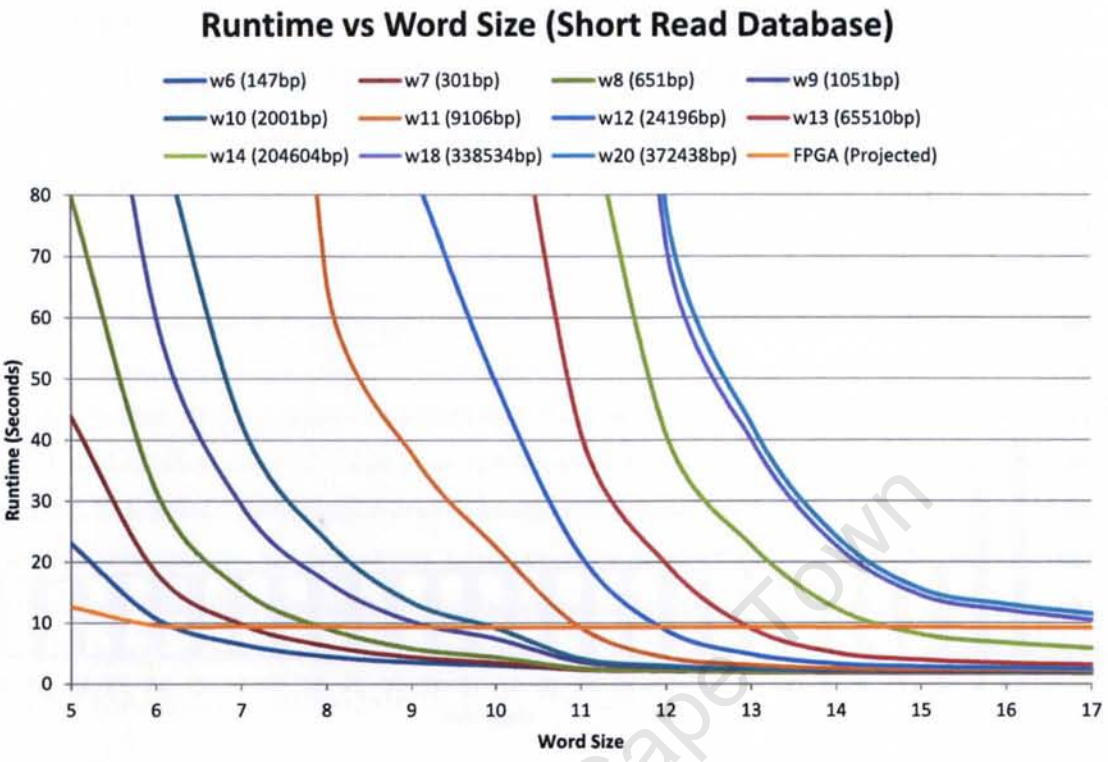
In Figure 6.5a it is clear that the amount of power required to generate the report in the short database represents a significant portion of the total power consumption of ROACH BLAST's execution. At word sizes 4 and 5 report generation requires approximately 4x the power of performing the similarity search.

6.3 Observations

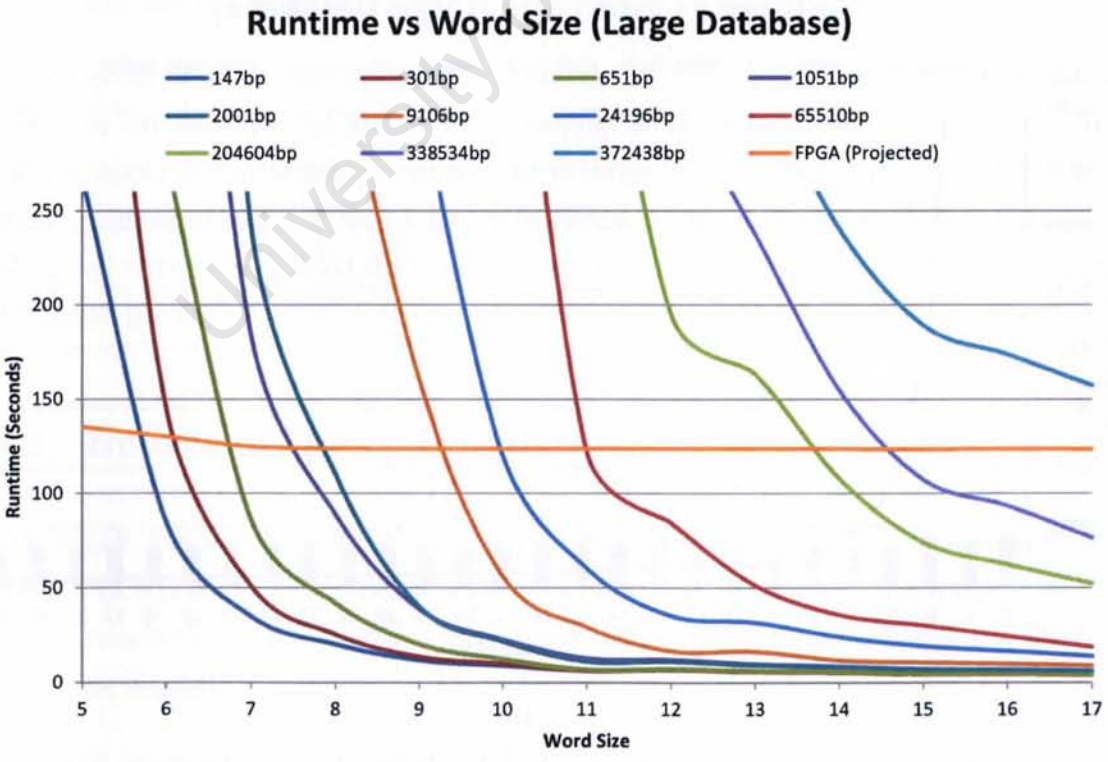
From the results observations can be made about the performance and behaviour of the ROACH and NCBI BLAST implementations. These observations give insight into the consequences of the design choices and the structure of the logic implemented in the FPGA.

6.3.1 Impact of Word Size

The main concern when reducing the word size was that it would lead to large numbers of seeds which would saturate the Extension Units creating a bottleneck causing ROACH BLAST to significantly slow down. This effect was not observed until the word size was below 6 and was much less pronounced than expected.

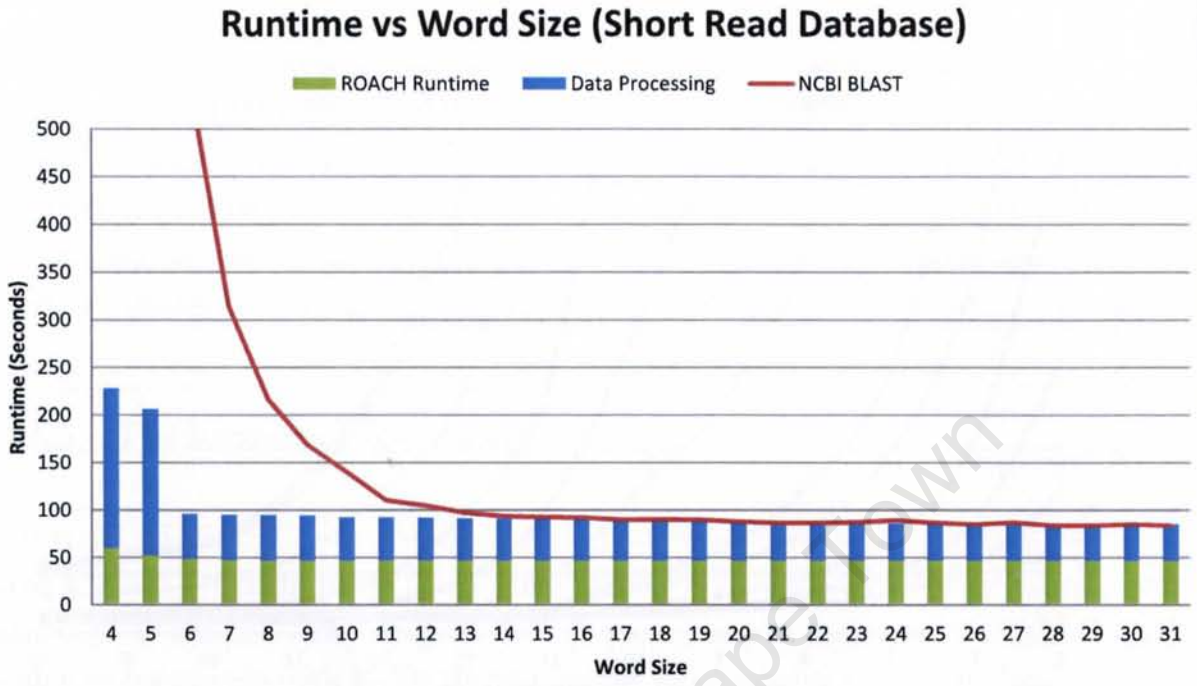


(a)

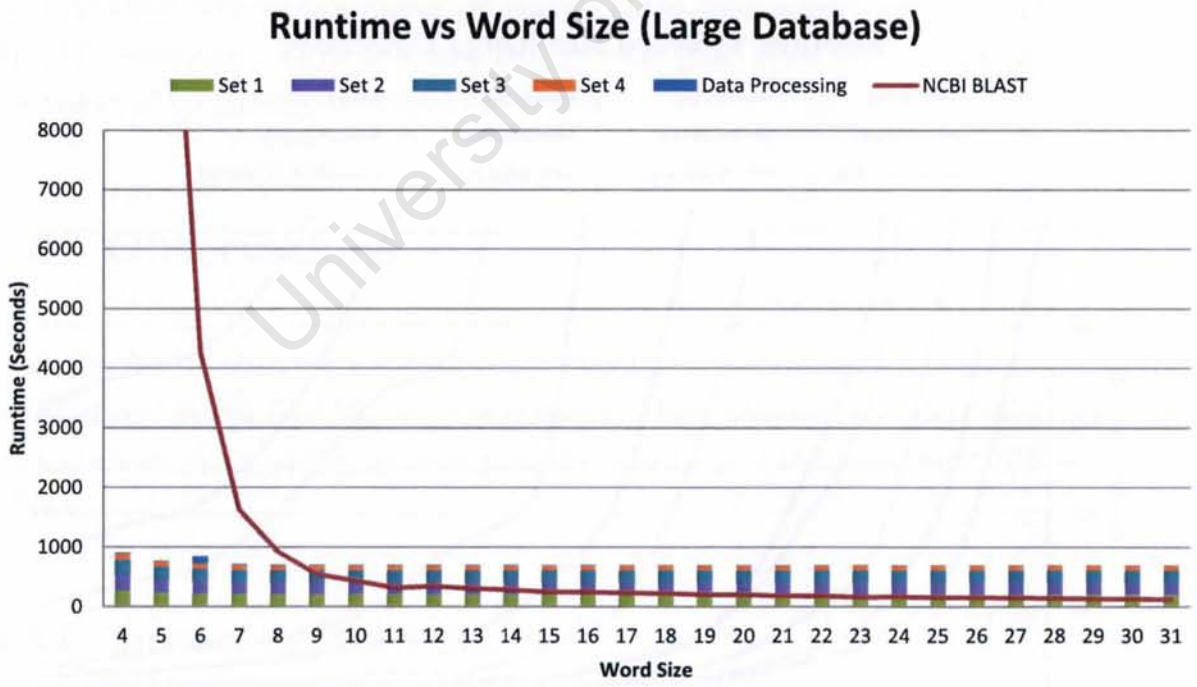


(b)

Figure 6.4 – Test 4, the projected break-even of ROACH and NCBI BLAST.



(a)



(b)

Figure 6.5 – Test 5, Figure 6.1 power normalised. Only stages of the algorithm executed on the ROACH board are scaled.

This was likely due to the poor quality of the seeds detected with very low word sizes terminating quickly during extension by the x parameter. Despite this behaviour more alignments are produced which creates a larger backlog for calculating expect and report generation on the workstation.

Even with the significant increase in backlog the length of the reports do not increase significantly in size due to the filtering based on expect. From this observation it is apparent that at a very low word size the vast majority of alignments returned from the ROACH board are of low significance.

Apart from the slight increase in runtime with word sizes 4 and 5 there is no effect on ROACH BLAST's runtime because under conditions where Extension Units are not saturated the Seed Detection Array becomes the limiting factor. The only variation in data that can cause the detection array to slow down are multi-cycle seed detections created when seeds fall across multiple Reference Elements in the array. This behaviour is caused by long regions of exact matches between the query and subject requiring multiple cycles for the signal to propagate through the array.

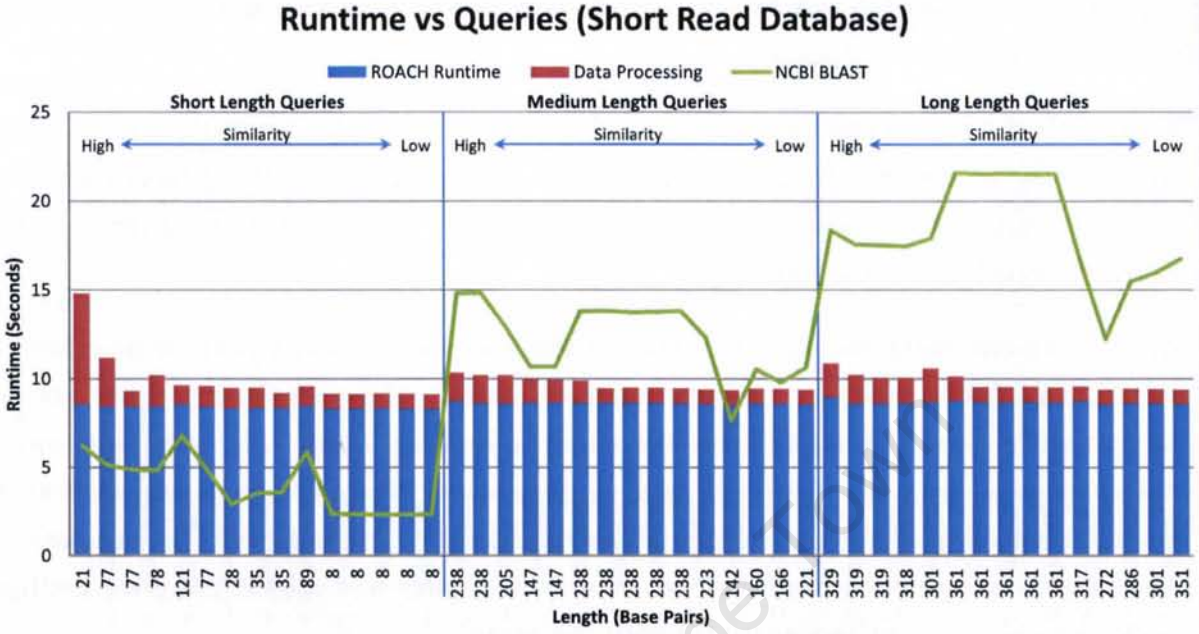
Since word size filters out seeds that are too short and multi-cycle extension is created by long seeds it was expected that word size would have little effect on the detection array, testing bares out this result.

While ROACH BLAST has a consistent runtime when varying word size NCBI BLAST experiences an exponential runtime increasing as the word size is decreased. NCBI BLAST's curve is flat when the word size is large but begins to increase at 11, which is the default word size for NCBI BLAST. Below 11 the runtime rapidly increases and breaks even with ROACH BLAST at 6, below 6 NCBI BLAST's is easily outperformed by ROACH BLAST.

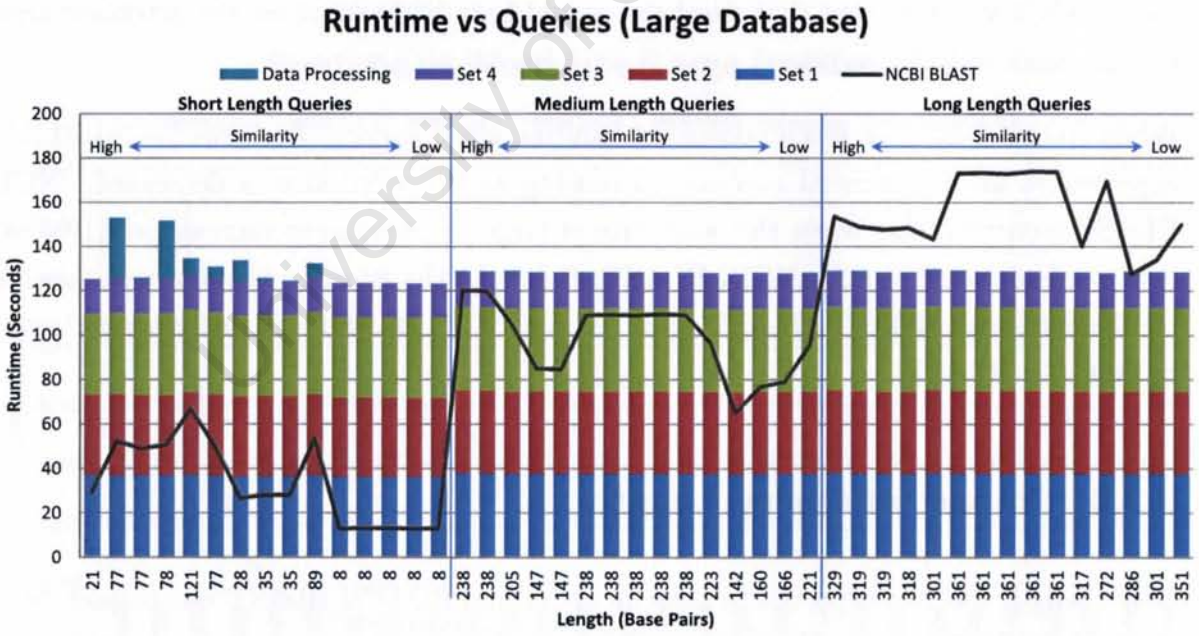
6.3.2 Impact of Query Length

Query length was not expected to have an impact on ROACH BLAST due to each letter in the query requiring an element in the detection array. In effect the longer the query the more parallelism can be extracted, however unused Detection Elements and Detection Regions are wasted.

The results agree with this assertion and display no change in runtime based on length, see Figure 6.2. When the runtime of each query is plotted individually and sorted by length, Figure 6.7, no discernible trend is visible on ROACH BLAST's plot.

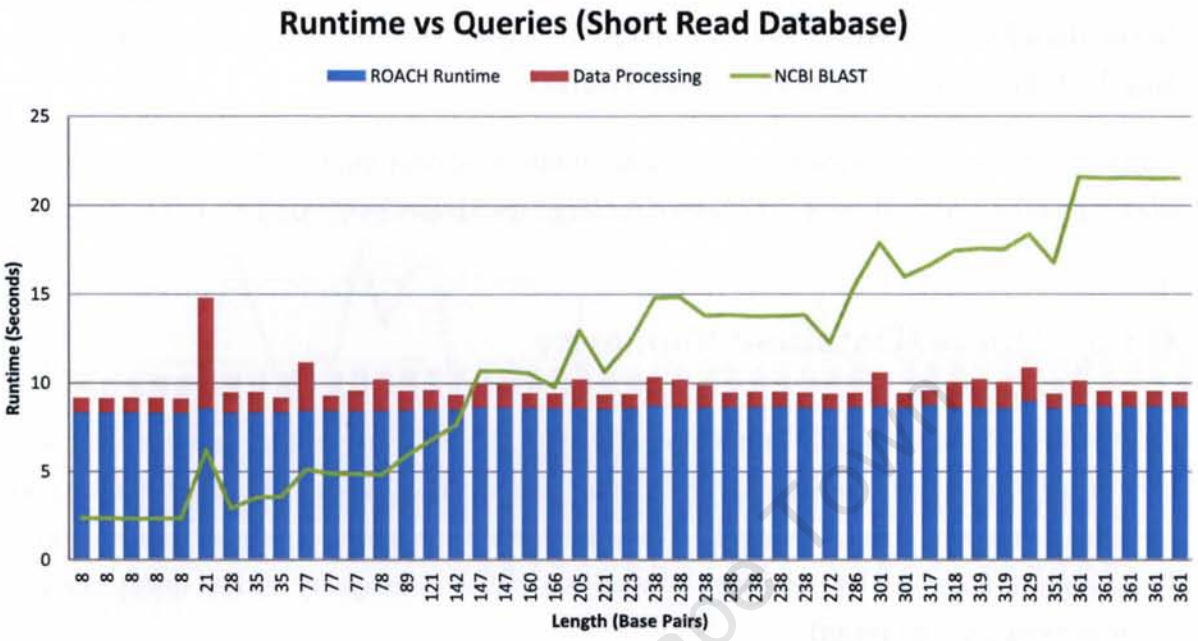


(a)

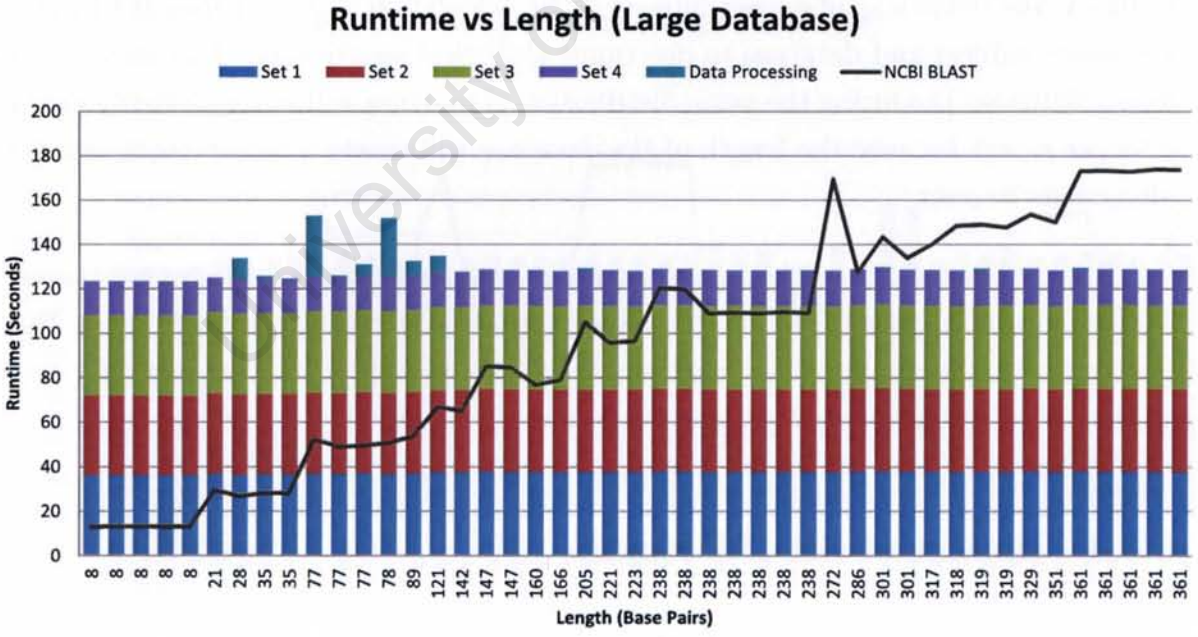


(b)

Figure 6.6 – The runtime with word size 6 for each of the queries in the test suite arranged in groups of length sorted by similarity. The "ripple effect" on the ROACH BLAST's runtime illustrates the impact of similarity, while the groups of length demonstrate length's effect on NCBI BLAST.



(a)



(b)

Figure 6.7 – The runtime with word size 6 for each of the queries in the test suite sorted by length.

NCBI BLAST is effected by query length experiencing an increasing runtime with length, however while there is a clear trend the plot in Figure 6.7 is erratic likely due to differences in similarity between the sets. The results show that NCBI BLAST is approximately 4x faster than ROACH BLAST with a query of length 8 letters, but twice as slow with a length of 361 letters for the short read database.

The same behaviour is visible on the large database except that NCBI BLAST's runtime is offset and on overall performs better on large databases compared to ROACH BLAST.

6.3.3 Query/Database Similarity

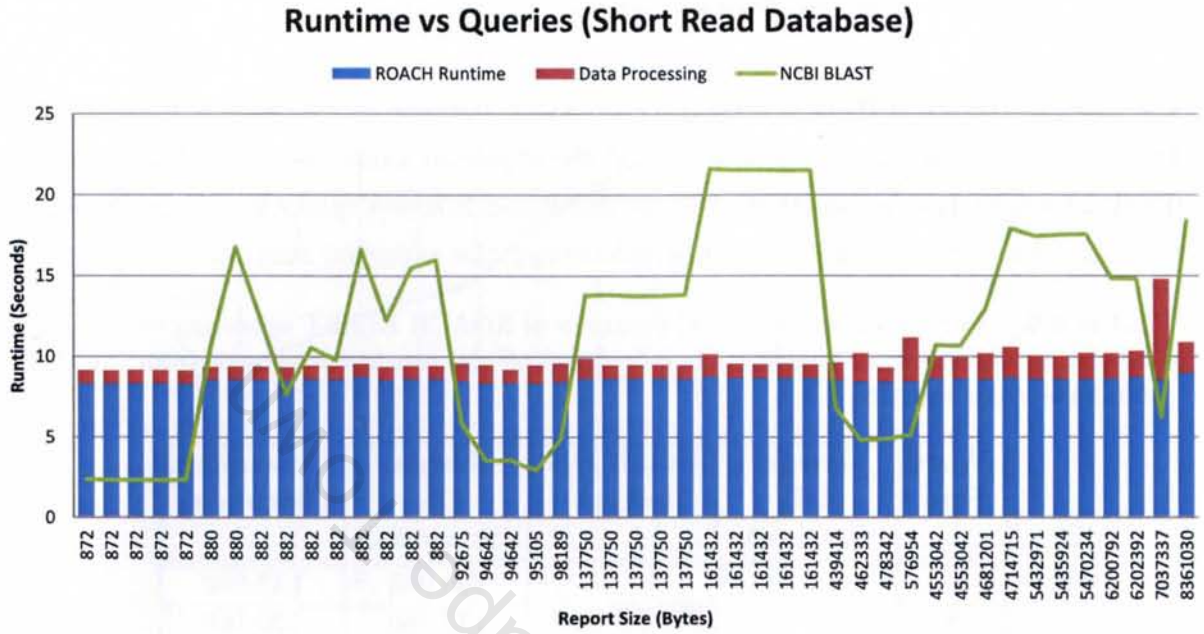
Similarity only has a small impact on ROACH BLAST increasing runtime due to the processing required to generate large reports, however the amount of time spent searching for alignments remains unchanged. The additional time spent on report generation can be hidden in software by starting the analysis of the next query in the query database while generating the report.

Another consideration that must be taken into account when viewing the similarity test results is the behaviour of expect filtering. The calculation of expect uses the length of the query, subject and database to determine statistical significance. This means that a longer database producing the same distribution of matches will not necessarily produce a longer report because the length of the database will create a higher requirement for alignments to pass.

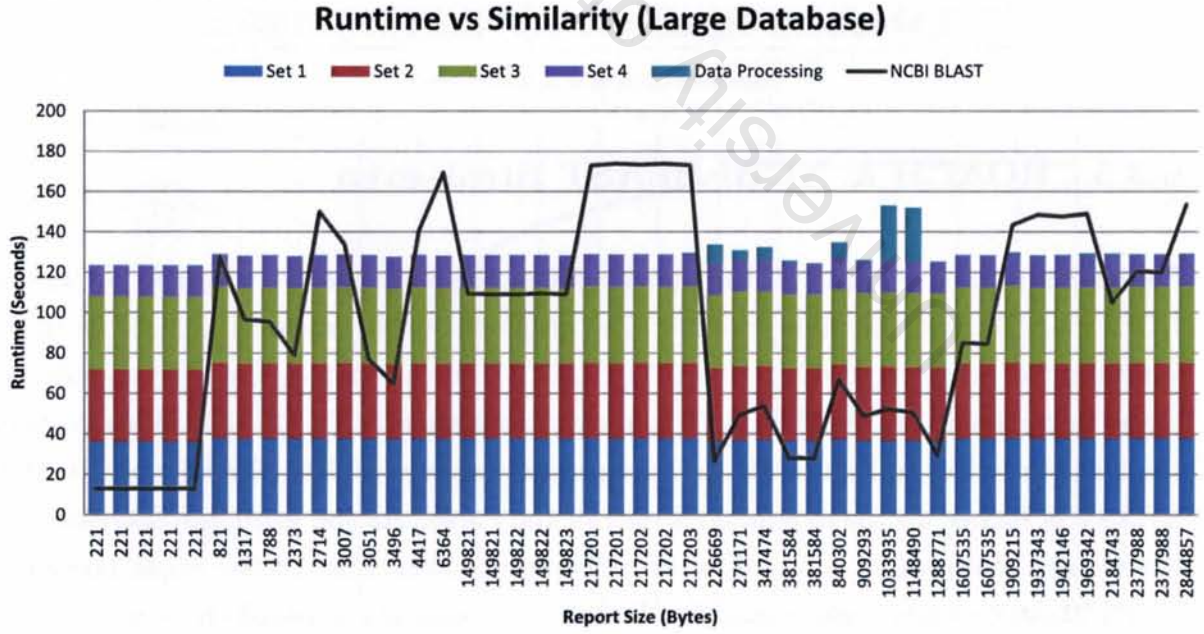
NCBI BLAST is also unaffected by similarity and only displays slight variations in runtime most likely due to the different average lengths between the similarity sets. When the runtime for each query is plotted separately the NCBI BLAST line is erratic due to the much greater significance of query length on runtime.

6.3.4 Impact of the DB length

All the tests were run on two databases, the smaller short read database and the large prokaryotes database. It was expected that the runtime would have a linear relationship with database length, and with ROACH BLAST, this should occur under most circumstances. To test this the expected runtime for the large database was calculated off the short database's runtime at a word size of 6, 8, 10, 12, 14 and 16 using the following formula:



(a)



(b)

Figure 6.8 – The runtime with word size 6 for each of the queries in the test suite sorted by similarity.

$$largeDbRntme = \left(\frac{largeDbLngth}{smallDbLngth} \times smallDbRntme\right) + smallDBDataProc \quad (6.1)$$

The calculations show there is a linear relationship between database length and runtime. The calculated runtime is less than 5% off the expected value, see Table 6.2. The effect database length has on NCBI BLAST is also linear however query length and word size must also be taken into account when calculating the expected runtime.

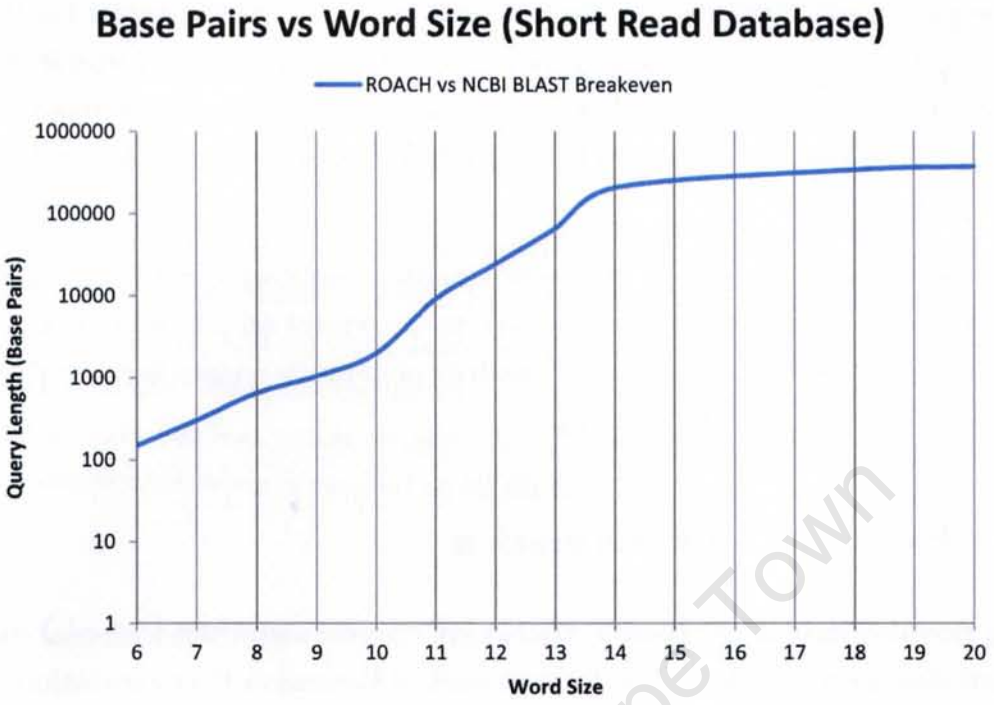
Table 6.2 – The actual and expected runtimes of ROACH BLAST operating on the large database. The expected runtime was calculated off the runtime measured from the short database using equation 6.1.

Word Size	Large DB (letters)	Small DB (letters)	Large DB (Seconds)	Calculated (Seconds)
6	7356385327	4873708	130.083	133.682
8			123.786	130.166
10			123.572	129.927
12			123.552	129.832
14			123.547	129.530
16			123.597	129.833

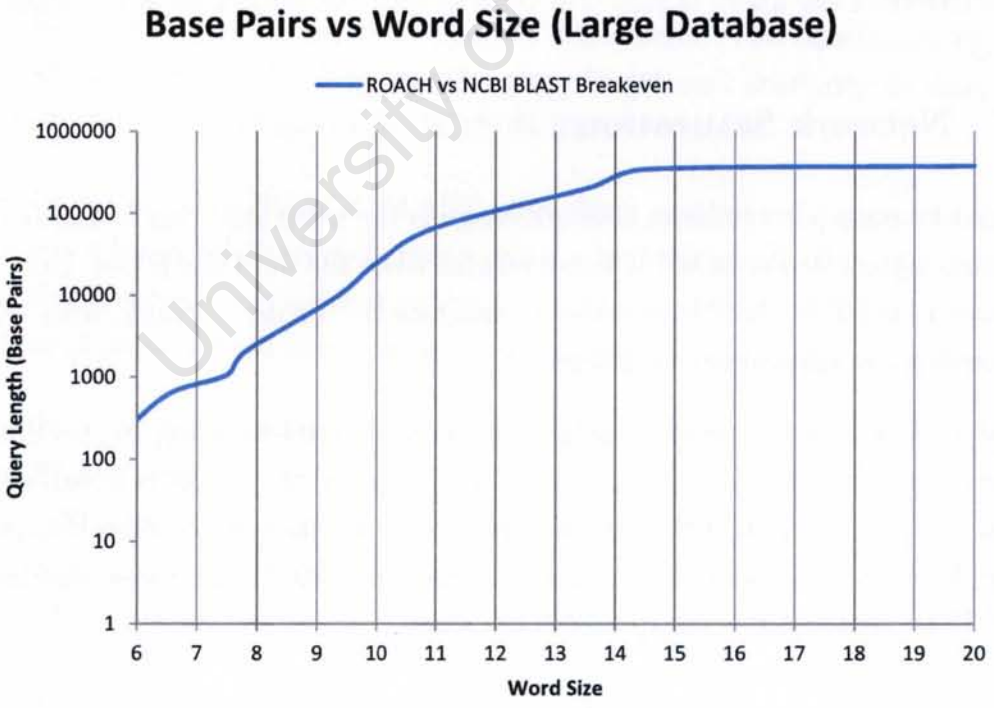
6.3.5 ROACH & NCBI BLAST Break-even

Figure 6.9 plots the length of the query against the word size at which ROACH and NCBI BLAST have the same runtime. The graph shows an exponential increase in the length of the query required for ROACH BLAST to break-even, followed by a period of linear increase until ROACH BLAST is always faster than NCBI BLAST. When viewing the low word sizes in Figure 6.4 the intersection between NCBI BLAST and ROACH BLAST is higher up the exponential part of the curve. As the word size increases the lowest part of the curve is raised pushing the intersection closer to the linear portion of NCBI BLAST's curve, and eventually the lowest portion of the NCBI curve is above the ROACH curve.

For any practical run ROACH BLAST wouldn't be crippled by only using a small part of the architecture when running multiple queries. As many queries as could fit in one run would be packed into the detection array, and while the total runtime for the set would remain unchanged it is effectively divided between the queries.



(a)



(b)

Figure 6.9 – Log graphs of the expected word sizes and lengths where ROACH and NCBI BLAST's runtime will be equal. ROACH BLAST is expected to be faster above the line.

Using the data from the short read set and viewing Figure 6.4 the result of running multiple queries simultaneously can be demonstrated as follows. Consider hardware able to support a 2000 element array with two 1000 letter queries packed into it, the result would move the break-even point from word size 9 to 10. If you packed in six 300 letter queries into the same hardware ROACH BLAST would always be faster than NCBI BLAST.

Tests involving multiple queries processed simultaneously were not performed due the limited length of the detection array possible to implement on the ROACH boards. The hardware does support the simultaneous analysis of multiple queries however the software required for decoding was not designed.

6.3.6 Extension Unit Saturation

Testing revealed that the Extension Units only become saturated with real data when the word size is set very low. The large impact of Extension Unit saturation observed during verification was with the database and query consisting all of the same character, but does not represent a real world scenario. The testing shows that 8 Extension Units per 128 letters of the query is sufficient to deal with the work load of real data.

6.3.7 Network Saturation

The main factor in determining network usage is the word size, this is due to the large number of alignments the ROACH board returns when the word size is low. Even with low word sizes ROACH BLAST was unable to saturate the 10GbE network, with bandwidth usage peaking at approximately 2Gbps.

The network utilisation can be further reduced by implementing expect filtering in hardware which would allow traffic to be routed through the ROACH board's PowerPC subsystem and over 1Gbps Ethernet. By routing the traffic over the PowerPC subsystem TCP could be used instead of UDP increasing the reliability of the system and the 10GbE core could be removed freeing up additional logic.

6.3.8 Performance-per-watt

As discussed in the methodology and results performance-per-watt cannot be properly represented by this test setup. Despite this it is apparent that ROACH BLAST is not

power efficient when it is only able to accommodate short queries. Under the short read database, where the best case performance-per-watt ratio was viewed, ROACH BLAST is only able to break-even with NCBI BLAST. Creating a cluster of ROACH boards to break-even with this database in runtime would consume the same amount of power as the workstation and be considerably more expensive.

For performance-per-watt to be improved for ROACH BLAST better suited reconfigurable computers need to be utilised. Current FPGAs are up to 6 times larger than the FPGA used during testing, and reconfigurable computers with four of these FPGAs on one board are available. By removing the unnecessary hardware from the ROACH board and utilising a more efficient power supply ROACH BLAST could become competitive in power consumption when compared to NCBI BLAST.

6.4 Conclusion

ROACH BLAST was benchmarked against NCBI BLAST because it is one of the most widely accepted BLAST implementations and because there are no modern implementations that remain true the original BLAST algorithm. The benchmarking assessed the impact of word size, database length, query length and similarity to determine the advantages and disadvantages of the ROACH BLAST system.

The results showed that ROACH BLAST has a fixed runtime for a given database, and that there is a linear relationship between runtime and the length of the database. NCBI BLAST also demonstrated a linear relationship between the length of the database and its runtime, however, word size and the query length also have a large impact.

As a result of the different behaviour of ROACH and NCBI BLAST it was found that ROACH BLAST performs well with low word sizes which increase the sensitivity of the algorithm. NCBI BLAST performs well on high less sensitive word sizes, but, if the query is long enough and ROACH BLAST can fit it in its Seed Detection Array then ROACH BLAST will always be faster.

Although this could not be tested in hardware it was confirmed during verification that the design scales well and the benchmarking showed that ROACH BLAST has a constant runtime therefore a reasonably accurate forecast of the word sizes where ROACH and NCBI BLAST break-even was made. Current technology is capable of implementing a 9000 element array in a single reconfigurable computer which is expected to place the

break-even point between word sizes 9 and 11, the break-even word size with a array of 383 elements is 6.

The benchmarked implementation of ROACH BLAST is capable out outperforming NCBI BLAST when the sensitivity is set high but better performance and greater compatibility with NCBI BLAST is possible and is being considered for future work.

Chapter 7

Conclusions

Sequence similarity searches are an important tool for helping microbiologists understand the large quantities of sequenced DNA that has been stored in databases. Due to the size of these databases and the slow speed of similarity search algorithms guaranteed to find optimal alignments many heuristics and accelerators have been developed offering a trade-off between speed and accuracy.

BLAST is one of the most widely used sequence similarity search tools with implementations for PCs, clusters, GPGPUs and FPGAs. FPGAs have displayed promising results in accelerating BLAST, however the implementations often depart from the original BLAST algorithm or are too rigid in their implementation. To address these issues the ROACH BLAST FPGA based accelerator was developed in this dissertation.

The ROACH BLAST system is implemented on a Xilinx Virtex 5 FPGA in the ROACH environment. The ROACH tool chain was not fully utilised due its lack of support for bioinformatics and only the board support package and 10GbE core were extracted. The custom processing core was developed in VHDL using Xilinx's ISE Suite.

ROACH BLAST is based on the systolic array approach proposed by Xia [44], however it attempts to overcome the limitations of their design with a novel architecture. The ROACH BLAST architecture implements the seed detection and ungapped extension stages of the BLASTN algorithm on an FPGA with all other stages running on a workstation connected to the ROACH board via 10GbE. The hardware design consists of Detection Regions which each contain a 128 element Seed Detection Array and 8 bidirectional Extension Units. There is no inter-Detection Region communication in the design facilitating multiple Detection Regions being stamped out within an FPGA or across multiple boards, thus allowing for easy scalability.

Exemplar sequences were passed through the hardware stages of the ROACH BLAST system and verified by hand and with a software implementation. The matches produced by the hardware conformed to the equivalent stages of the BLAST algorithm with additional, yet correct, matches when the word size was set above 7, as was expected.

The hardware was integrated with the driver software and tested against NCBI BLAST. NCBI BLAST implements modifications to the original BLAST design therefore it was expected the output would be slightly different. The output showed that ROACH BLAST produces output similar to NCBI BLAST and with modifications ROACH BLAST can produce output identical to NCBI BLAST.

After ROACH BLAST was determined to be operating correctly a test suite was created from real data designed to probe various aspects of ROACH and NCBI BLAST's design. The results show that ROACH BLAST has a consistent runtime for a given database due the structure of the Seed Detection Array in the FPGA. NCBI BLAST's runtime is affected by a variety of conditions but mainly by word size and query length.

NCBI BLAST is better able to take advantage of higher word sizes and in these cases easily outperforms ROACH BLAST. Despite NCBI BLAST's good performance with high word sizes it is projected that provided the query length is sufficiently long ROACH BLAST will be faster under all conditions. Despite ROACH BLAST's drawbacks it has proven highly efficient with low word sizes by maintaining a constant runtime while NCBI BLAST's runtime increases exponentially.

The BLAST implementation described in this dissertation is well suited to sensitive BLAST runs with word sizes below 6 on short queries of up to 383 letters. As FPGA technology improves larger queries will be supported and the number of word sizes where ROACH BLAST is competitive will increase. Current technology suggests that 9000 letter queries can be supported, if ROACH BLAST was implemented on such hardware it is expected to be competitive at word size 11 on databases of approximatively 750MB, and at word size 10 on 7.5GB databases.

To achieve a 9000 element detection array on a single board four FPGAs 6x larger than ROACH's SX95T would be required, the XC6VHX565T contains 6x the slices and BRAM of the XC5VSX95T chip. Only minor modifications to ROACH BLAST would be required to adapt it to a cluster implementation. In a cluster ROACH BLAST could be setup to support queries hundreds of thousands of letters long and through support for processing multiple queries simultaneously its true power could be realised.

ROACH BLAST's performance is currently limited by its support for only short queries and its adoption will be hindered by its output not conforming to NCBI BLAST. The pace at which FPGA's are growing and the performance of new reconfigurable computers is set to resolve the short query length issue and by integrating ROACH BLAST in NCBI BLAST's open source code conformity in results can be achieved.

7.1 Future Work

During testing modifications that can be made to the ROACH BLAST system to increase performance or conformity to NCBI BLAST were identified. The following modifications are recommended for future versions of ROACH BLAST.

7.1.1 Simultaneous Query Support

Simultaneous query support allows additional queries to be loaded into the Seed Detection Array if there are unused elements. Hardware support for this feature is already in place however the software support was not completed.

The system operates by loading queries into the detection array separated by a query sequence separator symbol defined in the ROACH BLAST instruction set. This symbol causes a mismatch to occur during seeding and terminates an extension when detected preventing alignments from running across multiple query sequences.

The software support required would consist of a lookup table to decode the alignment indices returned from the ROACH board to the appropriate query and mechanisms to track the alignment through the system so the result would land in the appropriate report.

7.1.2 Hardware Expect Filtering

A drawback of the current ROACH BLAST design is the amount of data returned to the workstation that is filtered out after the calculation of expect. The result of this large amount of traffic limits the practical size of clusters to only four boards and places a huge processing strain on the workstation. To overcome these issues expect filtering needs to be performed in hardware on the ROACH boards.

Fortunately the addition of expect filtering to the hardware design can be easily achieved by expanding existing structures. The values required to calculate expect are: λ, k, raw score, database length, query length and subject length of which only subject length and raw score vary between calculations.

Although the expect calculation contains a divide, which is difficult to implement in hardware, pipelined divider cores clocked at a higher frequency than the rest of the design can be implemented to prevent a bottleneck from forming. Further hardware would have to be added to the Arbitrators by expanding the lookup table which decodes the seed's subject position in the database to include its length. The subjects length would have to be passed with the seed through extension and into the Aggregator where expect filtering would occur.

By implementing this change at most 20MB would be returned to the workstation per query compared the hundreds of megabytes that are currently returned. This would alleviate the workload on the workstation and free up network bandwidth for a larger cluster implementation.

7.1.3 NCBI BLAST Integration

Users of BLAST have expressed a need for alternate implementations to produce output which is identical to NCBI BLAST. While the design proposed in this dissertation is accurate to the original BLAST algorithm it does not conform the the current version of NCBI BLAST. Integrating FPGA acceleration into the open source NCBI BLAST code would create a user interface and output which would be indistinguishable to the user and add features to ROACH BLAST which it currently does not support.

The main features this integration would bring are the support for DUST and gaps. DUST is a preprocessing algorithm which is used to remove regions of low significance from the query, and gaps allow related alignments to be combined into a single larger alignment. Apart from the support for DUST and gaps wrapping ROACH BLAST in the familiar NCBI BLAST code would increase microbiologists' willingness to use ROACH BLAST as an alternative to NCBI BLAST.

7.1.4 Multiple Clock Domains

During verification it was discovered that the Extension Unit stage of ROACH BLAST could be clocked close to 100MHz. By placing the Extension Unit stages in their own

clock domain and clocking them at 100MHz it is expected that two or three Extension Units could be removed from each Detection Region without negatively impacting overall performance. The freed up slices and BRAM can be used to implement longer Seed Detection Arrays or other modifications described in this section.

7.1.5 Network Control

Although 10GbE UDP performs well in ROACH BLAST it has the drawbacks of not guaranteeing in-order packet delivery and requires additional slices in the FPGA. Provided expect filtering was applied to the data inside the FPGA the bandwidth requirements would be reduced enough for matches to be returned over the OPB bus to the PowerPC subsystem and transmitted out over 1GbE TCP to the workstation.

Operating the ROACH boards over 1GbE has the added benefit of a large reduction in the cost for the network equipment, and the ability for reliable communication over longer distances. Apart from being expensive the 10GbE implementation the ROACH boards use is only capable of reliable transmission of up to 3 metres over copper cable. It is very important that transmission is reliable over 10GbE for ROACH BLAST since UDP is used and any packet loss will halt ROACH BLAST's execution.

7.1.6 Larger FPGAs & Cluster

The current ROACH BLAST implementation suffers due to the short query length support possible on the ROACH board. The architecture needs to be ported to a larger reconfigurable computer, preferably utilising XC6VHX565T FPGAs. These FPGAs are 6x bigger than the XC5VSX95T used on the ROACH boards, with each chip potentially supporting a 2303 letter query.

Increasing the length of the query allows more parallelism to be extracted and increases the number of queries that can be analysed simultaneously allowing ROACH BLAST to become more competitive with NCBI BLAST. Reconfigurable computers implementing four XC6VHX565T FPGAs on a board could support a 9125 letter query and when expanded to cluster of 16 boards a 147455 element array will be available.

For a cluster implementation to be possible additional hardware would be required to indicate to each reconfigurable computer its position in the array. The three existing types of Detection Regions can easily be used build designs to accommodate the slight

differences required for the reconfigurable computers sitting at the beginning, middle and end of the array. A modification would also have to be made to the Extension Units to only store a window of the query and database around the 128 letters associated with their Detection Region.

A suggested starting point would be to provide 8000 letters on each side of the 128 letters associated with the Detection Region. In the rare case where the extension runs all the way to the end a "partial extension" bit would returned with the index to the software so that the extension can continue on the workstation. This can be implemented without requiring too much additional slice or BRAM usage.

The software would have to be adjusted to serve multiple reconfigurable computers, the least resource intensive method of handling this would be to require all the reconfigurable computers to operate on the same portion of the database at the same time in a similar setup to how the Detection Regions are kept in sync inside the FPGA. This would allow the workstation to broadcast a single database stream to all the reconfigurable computers and provided expect filtering is performed in hardware the bandwidth required to return alignments to the workstation should be sufficient.

7.1.7 BLASTP

The evolution of ROACH BLAST must include support for protein sequences for it to become a comprehensive and useful tool for microbiologists. BLASTP differs from BLASTN in that it has a 20 letter alphabet instead of 4 letters and the probability of occurrence of each of the amino acids is not even. To accommodate the larger alphabet the datapath would have to be increased by 1 bit, which would marginally increase the size of the design. The more complex problem is dealing with the substitution matrix used to calculate and score the neighbourhood during seeding.

The BLASTP architecture is more complex than BLASTN, however its design would be capable of supporting both DNA and protein searches on the same hardware. Furthermore the other algorithms in the BLAST suite provide support for various forms of translation between DNA and proteins but still rely on the underlying BLASTN and BLASTP algorithms to perform the similarity search. Due to this behaviour it will be possible to extend ROACH BLAST to support all algorithms in the BLAST suite with a common accelerator architecture based on BLASTP.

Appendix A

ROACH BLAST Instruction Set

ROACH BLAST utilises a 4 bit instruction set which contains codes to represent both data and instructions.

0000 Database species separator, placed between subjects in the database to prevent seed detection and extensions from running across multiple sequences.

0001 Adenine

0010 Cytosine

0011 Guanine

0100 Thymine/Uracil

0101 Query mask and degenerate codes, creates a mismatch between the query and subject sequences.

0110 Database mask and degenerate codes, creates a mismatch between the query and subject sequences.

0111 Reserved

1000 Counter Reset, resets the database position counter. The database symbols must be preceded by this instruction.

1001 Notify when done, indicates to the system that the whole database has been received and that the system must produce the done signal as soon as processing is complete. This instruction must be placed after 128 "0000"

instructions at the end of the database to ensure the detection array has completed processing.

1010 Reserved

1011 BLAST Core Reset, flushes FIFOs and places all components into their reset state. Three of these instructions must be placed consecutively for the FIFOs to properly reset.

1100 Load Parameters, enters the Decoder into the state where the parameters for the run are loaded. The parameters are loaded 4bits at a time with the lower 4bits preceding the upper 4bits. The parameters are loaded in the following order: word size, S threshold, X threshold, miss penalty and match reward.

1101 Start Loading Query, indicates that the following symbols are part of the query to be loaded into the Seed Detection Array.

1110 Stop Loading Query & Reset, moves the symbols in the Seed Detection Array into the corresponding query registers and resets the database position counters.

1111 Query Terminator, placed in the first element of the Seed Detection Array. Ensures seeds which contain the first letter in the query are detected due to match indices being generated off mismatches. Also acts as the query separator, placed between sequences loaded into the Seed Detection Array for simultaneous analysis.

Appendix B

Data Pack

The data pack accompanying this dissertation contains the following files:

-Report

-Results Data

-Software & Tools

- Casper SDK
- NCBI BLAST Executables

-Source Code

- SimuLink Model for the top level of ROACH BLAST
- ROACH BLAST KATCP Driver
- Xilinx XPS Project
- Xilinx ISE Project
- C++ Project

Bibliography

[1] Altera. Intel atom processor e6x5c series. http://www.altera.com/devices/processor/intel/e6xx/proc-e6x5c.html.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[3] K. Bondalapati and V.K. Prasanna. Reconfigurable computing systems. *Proceedings of the IEEE*, 90(7):1201 – 1217, July 2002.

[4] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. Madden. Blast+: architecture and applications. *BMC Bioinformatics*, 10(1):421, 2009.

[5] UCT Computational Biology Group (CBIO). Paeano. http://cbio.uct.ac.za/arrayportal/Paeano.html.

[6] S. Che, J. Li, J.W. Sheaffer, K. Skadron, and J. Lach. Accelerating compute-intensive applications with gpus and fpgas. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 101 –107, june 2008.

[7] CISCO. Cisco 10gbase x2 modules. Datasheet.

[8] A.E. Darling, L. Carey, and W. Feng. The design, implementation, and evaluation of mpiblast. In *ClusterWorld Conference & Expo*, 2003.

[9] G. Estrin. Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer. *Annals of the History of Computing, IEEE*, 24(4):3 – 9, oct-dec 2002.

[10] A.D. Baxevanis et. al. *Bioinformatics - A Practical Guide to the Analysis of Genes and Proteins.* John Wiley & Sons, 2004.

[11] K. Asanovic et. al. The landscape of parallel computing research: A view from berkeley. Technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, December 2006.

[12] R. Durbin et. al. *Biological Sequence Analysis.* Cambridge University Press, 1998.

[13] M.S. Farrar. *NCBI BLAST Database Format.* NCBI BLAST Database Format, 19700 Helix Drive Ashburn VA 20147, March 2010.

[14] Center for Astronomy Signal Processing and Electronics Research (CASPER). Roach. http://casper.berkeley.edu/wiki/ROACH, October 2009.

[15] Fujitsu. Photos / ethernet switches. Online. http://www.fujitsu.com/us/about/platforms/fcpa/medias/photos/ethernet.html.

[16] D. George. Ten gbe v2. Online, August 2009. https://casper.berkeley.edu/wiki/Ten_GbE_v2.

[17] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '01, pages 642–649, Piscataway, NJ, USA, 2001. IEEE Press.

[18] R. Hartenstein. Trends in reconfigurable logic and reconfigurable computing. In *Electronics, Circuits and Systems, 2002. 9th International Conference on*, volume 2, pages 801– 808, 2002.

[19] T.C. Hodgman. A historical perspective on gene/protein functional assignment. *Bioinformatics*, 16:10–15, 2000.

[20] Xilinx Inc. Xilinx and arm announce development collaboration, October 2010. http://press.xilinx.com/phoenix.zhtml?c=212763&p=irol-newsArticle&ID=1343242.

[21] Intel. Intel microarchitecture codename sandy bridge. Online. http://www.intel.com/technology/architecture-silicon/2ndgen/index.htm.

[22] S. Karlin and S.F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc Natl Acad Sci U S A*, 87:2264–2268, 1990.

[23] J.J. Koo, D. Fernandez, A. Haddad, and W.J. Gross. Evaluation of a high-level-language methodology for high-performance reconfigurable computers. In *Application*

-specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on, pages 30 –35, july 2007.

[24] I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203 –215, feb. 2007.

[25] J.M. Lancaster and A.C. Jacob. Mercury blastn. 2010.

[26] W. Liu, B. Schmidt, and W. Muller-Wittig. Cuda-blastp: Accelerating blastp on cuda-enabled graphics hardware. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, PP(99):1, 2011.

[27] P.L. McMahon. Accelerating genomic sequence alignment using high performance reconfigurable computers. Master's thesis, University of Cape Town, October 2008.

[28] W. Moore and W. Luk, editors. *FPGAs.* Abingdon EE&CS Books, 1991.

[29] K. Muriki, K.D. Underwood, and R. Sass. Rc-blast: towards a portable, cost-effective open source hardware implementation. *Parallel and Distributed Processing Symposium*, 19, April 2005.

[30] NCBI. The statistics of sequence similarity scores. Online. http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html.

[31] NVIDIA. *OpenCL Programming for the CUDA Architecture*, version 2.3 edition, 8 2009.

[32] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879 –899, may 2008.

[33] B. Radunovic. An overview of advances in reconfigurable computing systems. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1999.

[34] J. Ray and J.C. Hoe. High-level modeling and fpga prototyping of microprocessors. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, FPGA '03, pages 100–107, New York, NY, USA, 2003. ACM.

[35] M.S. Rosenberg. *Sequence Alignment: Methods, Models, Concepts, and Strategies.* University of California Press, 2009.

[36] D. Schneider and M. La Rota. The blast suite. Online, 2000. http://www.generationcp.org/genomics/index.php?page=1177.

[37] H.K. So. *BORPH: An Operating System for FPGA-Based Reconfigurable Computers.* PhD thesis, University of California, Berkeley, 2007.

[38] H.K. So and R. Brodersen. A unified hardware/software runtime environment for fpga-based reconfigurable computers using borph. *ACM Trans. Embed. Comput. Syst.*, 7:14:1–14:28, January 2008.

[39] D. Strenski, P. Sundararajan, and R. Wittig. The expanding floating-point performance gap between fpgas and microprocessors. Online, November 2010. http://www.hpcwire.com/hpcwire/2010-11-22/the_expanding_floating-point_performance_gap_between_fpgas_and_microprocessors.html.

[40] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, and P.Y.K. Cheung. Reconfigurable computing: architectures and design methods. *Computers and Digital Techniques, IEEE Proceedings* -, 152(2):193 – 207, March 2005.

[41] S. Vercauteren, B. Lin, and H. De Man. Constructing application-specific heterogeneous embedded architectures from custom hw/sw applications. In *Proceedings of the 33rd annual Design Automation Conference*, DAC '96, pages 521–526, New York, NY, USA, 1996. ACM.

[42] P.D. Vouzis and N.V. Sahinidis. Gpu-blast: using graphics processors to accelerate protein sequence alignment. *Bioinformatics*, 27:182–188, 2011.

[43] D. Werthimer. *Problems with ROACH Gbit Ethernet Port.* CASPER, Berkeley, October 2010.

[44] F. Xia, Y. Dou, and J. Xu. Fpga-based accelerators for blast families with multi-seeds detection and parallel extension. *Bioinformatics and Biomedical Engineering*, pages 58–62, May 2008.

[45] M. Zvelebil and J.O. Baum. *Understanding Bioinformatics.* Garland Science, 2008.