NTNU
Norwegian University of
Science and Technology

Spam Filtering With Approximate Search in FPGA Hardware

Henriette Marie Borgund

Master in Information Security
Submission date: May 2018
Supervisor: Slobodan Petrovic, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Preface

This is a master's thesis at NTNU carried out during the spring semester of 2018. The research problem was presented by Professor Slobodan Petrović in the *Digital Forensics Research Proposals 2017*. The main topic of this thesis is the approximate search algorithm using Levenshtein distance and the implementation of this algorithm onto a Field Programmable Gate Array, FPGA, board. This thesis is written for an audience of technical interest to adapt traditional solutions to the ever expanding area of large text search.

01-06-2018

Acknowledgment

I would like to thank my supervisor, Professor Slobodan Petrović, for his guidance and genuine enthusiasm towards the subject throughout this project. During the thesis writing, he has strictly guided me towards succeeding in my experiments. He has also aided me with the needed software code for this thesis' comparisons. Without this, this thesis would have few results.

I would also thank my superior at work, for allowing me to take some time of to write and finish this thesis. In addition, I would also especially thank my colleagues for encouraging me, and help me through discussing my thesis at coffee brakes. I would also thank my friend Kirsi Helkala for proof reading and guidance during the final days of writing this thesis. Finally I would like to thank my family for motivation and support throughout my studies.

H.M.B

Abstract

Spam poses a complex threat to organizations, business and end users. Large high speed networks require fast spam filtering applications to be able to do search through the large amount of emails that are received. The processing speed of traditional software processors are not increasing as much as the increasing amount of data that are to be processed. When spammers also try to bypass a spam filter by obfuscating the spam keywords, the workload of the processors increases even more.

By outsourcing specific tasks of anti-spam solutions, faster processing times can be achieved. This thesis implements the approximate search algorithm onto a Field Programmable Gate Array, FPGA, board, in order to find obfuscated spam keywords faster than the traditional software implementation. The algorithm has the ability to work in a bit parallel fashion, and hence utilize the parallel feature of hardware.

This thesis compares a traditional software implementation of the anti-spam measure approximate search to a hardware implementation of the algorithm. The results shows that a hardware implementation of this algorithm can give record braking processing times. The proposed FPGA program processed a search text approximately 55 times faster than the software program. To perform these tests, a hardware program was developed in Verilog Hardware Language, HDL. The program takes a pattern input and searches for approximate matches in a search text. The algorithm used is the bit parallel approximate search using Levenshtein distance.

Contents

List of Figures

List of Tables

1 Introduction

Spam is unsolicited email sent in bulk. Often used for marketing purposes, but later years ransomware and phishing are also included in these spam emails. Email is an essential communication tool, but also a basis for distraction and wasted time. An email inbox is an important part of information exchange, but is also the perfect container for spam. Spam continues to represent approximately 50 % of received e-mails. The potential content of a spam email poses a security risk for email users and providers. Spam filtering is used to prevent spam from being exposed to end users. By modifying typical spam keywords, spammers can dodge the spam filter and expose their message to potentially many recipients. The approximate search algorithm can be used to find approximate patterns in a text. That can solve the cumbersome process of detecting the modified spam keywords that are used to disguise the spam email from the anti-spam measure.

This thesis covers in short the topics: spam, Field Programmable Gate Array, FPGA, approximate search using Levenshtein distance and Verilog hardware programming. To this end, various unconstrained and constrained approximate search algorithms have been implemented in a dynamic programming and bit parallel manner in software. The objective of this thesis is to clarify and describe the possible efficiency that can be found when using hardware to accelerate the spam filter program. This thesis aims to code and implement an FPGA based spam filter to detect spam by utilizing the unconstrained approximate string pattern matching using the bit parallel Levenshtein distance algorithm. The task of this thesis is to, by the means of experiments test the accuracy and efficiency of a hardware based spam filter. Hardware has in previous studies showed itself capable of speeding up the processing of large amounts of text in reasonable time.

This first chapter describes the problemand proposed solution. It justifies the motivation and contribution of this theses and propose research questions to be answered throughout this thesis. A section is provided to discuss related work in this field of study before the thesis outline is depicted.

1.1 Problem Description

It is four minutes to nine when the alarm goes off in a security operations centre of a security firm, Mnemonic, in Norway [8]. Traffic from a client to a domain known to spread malicious content is detected by security analysts. An employee at the clients firm has clicked on a link in an email she has received. The email is written in the employee's native language, pretending to be a notice to pickup a package from the postal office signed with the Norwegian post office logo. As the employee was expecting a package, she was not suspicious when clicking on the attachment link, supposedly being the pickup notice. But the attachment contained malware infecting her computer with the ransomware TorrentLocker [8]. The ransomware TorrentLocker has mostly used spam as its means of transportation since its reveal in 2014 [8]. In this specific case, the anti-spam measure

of the company had not stopped the email from reaching the end users email inbox. Ransomware can encrypt not only files on the infected computer, but entire networks. This can be of crucial damage for a company. The client was notified by the security company of the unwanted traffic, and the infected computer was taken off line before doing too much damage, and before spreading. Encrypted files from this type of attack, must be considered lost. The only way to get these files back are by paying the ransom, which helps finance criminal activity and motivate more attacks. A perfectly tuned spam-filter would have stopped this spam from entering this inbox, and would ease the expensive workload of security operation centers.

Malware and viruses are some of the biggest threats to companies. The Norwegian report *Mørketallsundersøkelsen 2016* presents numbers from a survey of Norwegian data crime and information security events [8]. 32% of the organizations included in the survey had been affected by unspecified virus or other malware. 10% of these reported spam as its most serious security issue.

> "Two years from now, spam will be solved."
>
> *Bill Gates*
> *January 24, 2004*

Bill Gates predicted that spam would be solved by 2006. Unfortunately, he could not predict the future in this matter. The growth of spam has from 2004 slightly decreased in percentage, but still comprises well over 50 % of an email account [1]. Email is still the preferred platform for information exchange. Despite the development of new technology that enables other methods to communicate, email is still preferred amongst private users, industry and business networks [9]. Numbers from the Norwegian statistics, SSB, shows that more than 90 % of the Norwegian population used internet connected email solutions [10]. The survey from 2017, actually shows that more people are using internet to send and receive emails, then to read online newspapers or use social media [10].

The technology that enables email was conceived in an era where the security in network based communication was characterized by trust. Email was initially, and still is, an insecure technology, easy to exploit, spoof and manipulate. The email architectural structure has not changed, which means that anti-spam and security measures has to work its way around the structure to function properly. And, since Bill Gates in 2004 stated that spam would be solved in two years, anti-spam measures have been in continuous development. They have helped making sure we are only exposed to emails we are intended to see.

As the anti-spam techniques evolves, so do the spammers. More and more intelligent spam are finding its way into our inbox. Email spam is a perfect marketing place, as it is free and reaches a big crowd. It is, as described in the beginning of this section, also the perfect way of delivering ransomware. One of the countermeasures against spam is the content-based keyword spam filter. One technique to avoid a keyword based spam filter is to obfuscate the typical spam keywords, that the spam filters react to. An example is the word "viagra". Emails containing this word are often filtered by a spam filter, because statistically an email containing this word is a spam email.

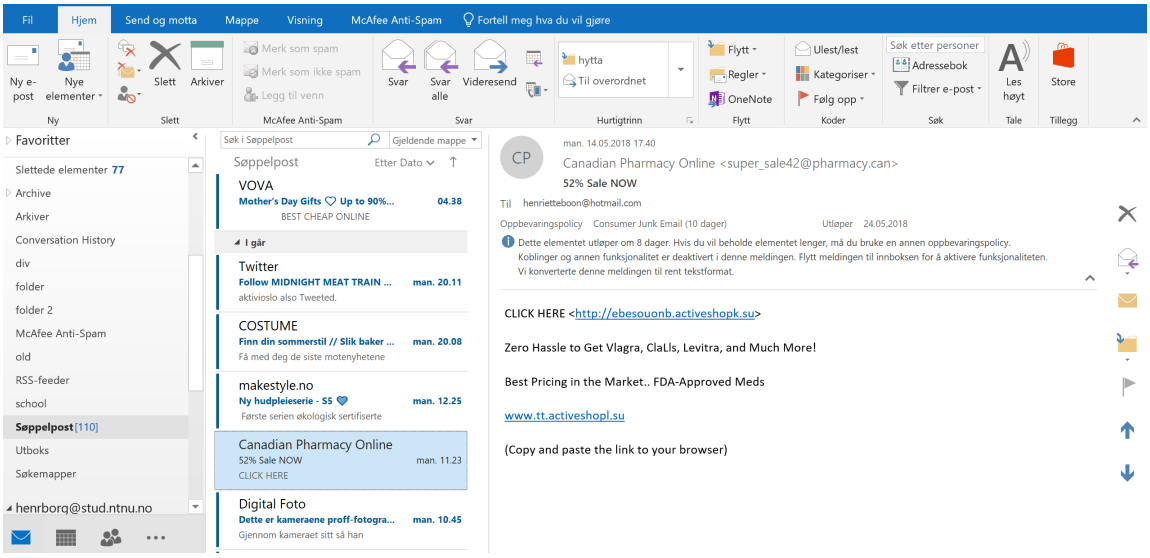

Figure 1: An example of an email inbox filled with spam. In this example, the spam keyword "Viagra" is replaced with "Vlagra".

Intelligent spammers change the word "viagra" to "v1agra" and thus avoids being classified as spam. Figure 1 is an example of an email inbox filled with spam. In that specific example, "viagra" is intentionally misspelled by the spammer to "Vlagra". In the same spam email "Cialis" is also intentionally misspelled as "ClaLls".

These emails are for the end user, normally considered, an annoyance. And in most cases, spam is only annoying. But spam emails are also a security risk. Some spam emails, like the one mentioned in the beginning of this section, are potential hazards for a business, or even for an end user. Email users can, in most cases, manage to sort out spam emails that the spam filter has failed to classify. In those cases where all spam emails are properly classified by the user themselves, it is in the worst case an abuse of time. But in those cases when neither the spam filter nor end users manage to classify a spam email correctly, a security risk is present. Phishing emails that lure confidential information such as log-in details, or other credentials from a user can pose a threat to private users and an even bigger threat to bigger companies. Other spam emails can be of malicious content, attached in files. A malware infected computer can be used to encrypt files for ransom, to steal, delete or change information, or provide a backdoor for hackers to exploit.

Spam filter solutions today are mostly software based solutions [11]. Long traditional text searches are time consuming for most processors. Large high speed networks acquire a fast email filter to process e-mails in a reasonable time and space complexity. Using hardware based solutions can speed up the filtering process [12] [4], and be more time and cost effective for email providers and end users.

1.2 Topics Covered by the Thesis

This thesis is about accelerating spam filtering with approximate search on FPGA hardware. Natural topics to cover in order to solve the problem are spam, its history and current state, and especially spam filtering. Field Programmable Gate Arrays, FPGA's, are also important to understand to fully grasp its functionality in being the hardware basis. The algorithm used to solve the filtering problem is unconstrained approximate search using Levenshtein distance, and is also thoroughly explained throughout this thesis.

1.3 Keywords

Unconstrained approximate search, edit distance, Levenshtein distance, spam filtering , bit parallelism, FPGA, hardware

1.4 Motivation

The aim of this thesis is to determining if hardware solutions should be chosen over software solutions in the development and launch of spam filters. The research proposal was given by professor Slobodan Petrović in digital forensics research proposal collection of 2017 [13]. The author of this thesis had prior little knowledge or experience with either of the topics included in the problem statement, but was motivated to learn and do research from scratch. Engineers with hardware language programming experience are rare, and should, because of the benefits of using hardware, become as common as knowing C or Java.

Text search is a time consuming task, even with the use of computer programs. Digital information is exponentially growing, also in forms of storing digital communication, for example by means of email. The duration of a general text search scales up more with the size of the text, than it scales down with increasing hardware performance [4]. Because of this it makes sense to design, code and implement an algorithm that can utilize bit parallel methods to a hardware based platform. This is to accelerate the spam filtering function, performing record-breaking retrieval times.

1.5 Contribution

This thesis focuses on the implementation and testing of an unconstrained approximate search algorithm on hardware FPGA board. The results obtained in this thesis are relevant for choosing efficient platforms for spam filtering. It is also relevant for further study on accelerating spam filtering with approximate search. With the work of this thesis a code has been developed in hardware description language to solve the bit parallel unconstrained approximate search algorithm. The code is implemented on a small, cheap, FPGA student type board to prove the effectiveness of using hardware over software. A fully developed hardware implementation of the proposed solution will improve the email system and reduce the risk of exposing users to unsolicited emails.

1.6 Research Questions

This thesis will answer the three research questions below.

1. How can approximate search algorithms be implemented in FPGA hardware?
2. What is the accuracy and efficiency of spam detection in hardware compared to classical software implementations?
3. Why should hardware solutions be chosen rather than software solutions?

1.7 Related Work

Search for patterns in text has been a popular problem for decades. Many algorithms have been developed, studied and published [14] [15] [16] [17] [18] [19] [20] and are still relevant in the field of study. Text is despite the digital evolution, the main form of information exchange, and hence the problem is increasingly relevant to researchers. Many of the algorithms studied in the research mentioned above, concern solving the approximate search with dynamic programming. An implementation of the dynamic programming Needleman-Wunch algorithm was implemented on an FPGA in 2007 [12] and proved to greatly speed the performance of this algorithm in spam scanning utilizing hardware as opposed to software. But approximate search algorithms employing bit-parallelism have also been studied[20] [21] [22].

In 1992 two seminal papers of Baeza-Yates and Gonnet [14] and of Wu and Manber [20] in the September 1992 issue of the Communications of the ACM were published. They described the intrinsic parallelism of the bit operations inside a computer word, also referred to as bit-parallelism [17]. Simone Faro and Thierry Leqroc [17] reviews the twenty years of study of bit parallel techniques in their paper *Twenty Years of Bit-Parallelism in String Matching*. The aim of these techniques described in their report, is to speed up the execution time as much as possible in order to make them capable of processing extremely large amounts of data, which represent modern information systems. Fast and efficient search algorithms have been designed and discussed thoroughly in [18] [21].

A few studies have implemented exact search algorithms utilizing bit parallelism [23] onto FPGA boards, successfully accelerating the processing speed of intrusion detection systems.

Chapter 3 discusses related work in more depth. Section 3.1 discusses previous work related to spam. Section 3.2 describes the state of the art of FPGA's and compares it to other types of processors. Section 3.3 goes into details of the literature describing approximate search pattern matching.

1.8 Thesis Outline

This thesis comprises seven chapters:

1. Introduction: This chapter gives an introduction to the topic of this thesis and states the problem description and research questions. It also explains the motivation for the choice of the topic and presents related work.
2. Methodology: This chapter explains the research methods and the methodology implemented as a means to answer the research questions for this study. The chapter reasons the choice of

research approach, research design and what tools are used to produce valid results.

3. Literature review: The literature review presents relevant background theory to follow the thesis. It describes the current situation of email spam and its anti-spam techniques. It gives an introduction to the subject of FPGA and gives an understanding of the bit-parallel unconstrained approximate search using Levenshtein distance.
4. Implementation: This chapter describes the implementation of a spam filtering solution utilizing the algorithm described in Chapter 3 onto an FPGA board. It describes the design flow of hardware design, and how it was utilized in the program development during this thesis.
5. Experiment: This chapter explains the experimental setup and its results. This chapter is written to enable those who wish to replay the experiment to do so.
6. Discussion: This chapter discusses the results from the experiment chapter and literature review and tries to answer the research questions with findings from the results analysis. The three research questions are carefully discussed, as well as the sources of error and validity of the research.
7. Conclusion: This chapter concludes the thesis and points to future work on the topic.

2 Methodology

The previous chapter gave an introduction to the topic and the problem this thesis aims to solve with the research questions. The purpose of this chapter is to explain the research methods and the methodology implemented as a mean to answer the research questions. This chapter explains the choice of research approach, the research design, as well as the advantages and disadvantages of the research tools chosen. This will be followed by a discussion on their ability to produce valid results, meeting the aims and objectives set by this thesis. The chapter then goes on to discuss the sample size and the sampling strategy applied by the author, and the data analysis methods which have been used. It concludes with a brief discussion on the problems and limitations posed by the research methodology, as well as problems encountered during the research.

2.1 Research Approach

The research approach used in this thesis is quantitative. But the approach tends to be more qualitative than quantitative when answering the close-ended questions to the quantitative hypothesis [24]. The quantitative research is an approach for testing objective theories by examining the relationship among variables [24]. The variables in this thesis are hardware and software. To provide a more complete understanding of the research problem, the approach is qualitative in the sense that is uses words to answer the research questions. This thesis uses a deductive approach. Deduction generalizes from the general to the specific. It uses data collection to evaluate hypothesis related to an existing theory [25]. The existing theory in this thesis is that hardware can be used to speed up algorithms traditionally implemented with software. This deductive approach is to falsify or verify the theory [25]. Figure 2 shows the deductive research approach.To compare, the inductive research approach generalizes from the specific to the general, and the abductive research approach generalizes from the interactions between the specific and the general [25].

2.2 Research Design

This thesis makes use of quantitative research strategy in the sense that there will be numeric [25] data produced [25]. A quantitative research strategy is principally applicable for the purpose

Figure 2: Deductive research approach

of this thesis, where two variables are tested against each other in numbers. The thesis will be of an experimental design. Experimental research seeks to determine how a specific treatment influences an outcome [24]. In this thesis the experiment that was carried out seek to determine if converting a bit parallel approximate search algorithm would perform faster on an FPGA board, than on a computer. This thesis also applies explanatory sequential mixed methods by conducting the quantitative research, analyze the results and then build on the result to explain them in more detail with qualitative research [24]. The qualitative research is carried out in Chapter 3.

2.3 Research Methods

The validity and the advantages and disadvantages of the tools used in this thesis is discussed in this section. The specific research methods involve the forms of data collection, analysis and interpretation that is proposed for this study [24]. For the purpose of this research, the author of this thesis decided to use closed-ended questions, predetermined approaches and numeric data. The closed-ended research question is described in Chapter 1 and is the research question "What is the accuracy and efficiency of spam detection in hardware?". The predetermined approach is the coding and development of a hardware spam filter solution. And the numeric data is the testing of the two implementations, software and hardware. This will test and verify the theory that implementation in hardware can be used to speed up programs that utilize invariable algorithms. The experiment uses statistical procedures to measure the effectiveness of the software implementation as the running time varies for each run.

2.3.1 Experiment

An experiment is a scientific tool. It is a try and adjust to try out a theory before knowing if it can be proved before the experiment is undertaken multiple times [26]. Experimental research is called a true experiment. Experimentation is used to try out existing theories or new hypotheses to find support for them or disprove them. A variable, hardware implementation of the algorithm, is manipulated , and the other variable, the software implementation, is controlled [27]. A definition of experimental research, a quasi-experiment, is research where the author has influenced the phenomena to observe the consequences [27]. The phenomenon influenced in this thesis is the classical software implementation of the approximate search algorithm used in spam filtering. Experimental research is important, it often results in improvement of our everyday life. In the case of this thesis, it can help users not to be exposed to unsolicited emails in large high speed networks. The experiment was planned to ensure that the experiment was carried out properly, and that the result reflected the real world.

The data-set used in the experiment is small, but represents a real world email example. The data-set is a collection of words representing *normal* emails and spam emails infused with spam keywords. The keywords used in the data-set for the experiment were chosen from the Symantec email spam report from 2017 [1], to simulate a real state of the art example. But for all practical purposes, these keywords could have been nonsense, and would still deliver valid results. The data-set was converted to hex using ASCII encoding before it was run on the hardware implementation,

and consisted of 477 symbols that made up *normal* words that should not initiate a hit, and spam keywords that should initiate a hit.

Identifying and controlling non-experimental factors in which the author of this thesis did not want to influence the effects, was crucial to draw a valid conclusion [27]. To avoid non experimental factors, the two variables were controlled in the sense that the two implementations worked similar. Prior to the experiment, different types of data-sets and keywords were properly tested to both implementations to assure the accuracy of the two programs. The first data-set used in the program development of the hardware solution, was the same text and pattern as Holub [28] used in his dissertation to describe bit parallel approximate search. This was done to assure that the calculation of the rows, R_i^l, was done correctly by the FPGA. The results showed in a simulation behavior of the hardware implementation, allowed for comparison in a bit-wise manner. This allowed the author to conclude that the use of hardware for the algorithm implementation was the cause for the effect that was measured in the experiment.

The experiment test was run 11 times for each pattern and text size, and the means of these runs were calculated to make the results more reliable. The Algorithm used to carry out the experiment is found in Chapter 5 (Algorithm 1).

To conclude on the found effect of the manipulation, there must be sufficient difference between the manipulated variable and the controlled variable. This is typically determined statistically by comparing results from both variables and deciding if the observed difference could occur randomly or not. In the latter case, the hypothesis of coincidence, *null hypothesis*, is rejected, and it is assumed that the manipulation has had effect. If this is true, the results have *inner validity* [27].

2.3.2 Other Methods

Upon embarking on this research, the author of this thesis initially envisioned designing a real world spam filter, to filter live emails. This approach was soon overruled, as the amount of work would be overwhelming for the author. It would also be prone to less validity of the experiment as more variables that the author could not control would have an impact to the results.

2.4 Program development

To test the two variables against each other, two programs were needed. The classical software to compare against was provided by the supervisor Professor Slobodan Petrović. The hardware program was developed by the author as a part of the research of this thesis. The FPGA board used for implementation was a Nexys4 DDR provided by NTNU. The programming language used was Verilog hardware language. The development method used to program the board used the waterfall method, which consists of setting requirements, design architecture, implement and verify [25]. Criticism of this approach is often in correlation with client-customer conditions [25]. In such a small scale development environment as this research was conducted in, the requirements and the design architecture were well set in the beginning.

2.5 Literature

Most of the literature that was reviewed and used to design the program on to the FPGA board were provided by the thesis supervisor. Other scientific papers that were chosen to aid in this research were found in scientific paper collections. Documentation from Digilent [29] was used during programming as well as implementation and the programming guides were provided by Xilinx [5].

2.6 Methods of Data Analysis

The results of running the test on the software and hardware implementation were analyzed in a mathematical manner with diagrams. The data set used for testing was relatively small, but gave valid results. It was a distinct difference between the two variables. For further and more comprehensive testing, larger datasets could have been tested as well. Testing with larger datasets would make it easier to visualize the effect of utilizing the solution in a real world scenario. But the tests used in this thesis are significant enough to predict the potential result of a larger scale project.

2.7 Problems and Limitations

There were several problems and challenges which the author of this thesis encountered in the process of writing this thesis.

The first challenge was to learn hardware programming language. The author had no prior knowledge with VHDL or Verilog hardware programming language. The first program that was made in the work of this thesis was an advanced led light switch on the FPGA board. Making that led light switch alone was a challenge, but upon completion, it gave the author motivation and insight into the hardware programming conceptualization.

Secondly the author had to familiarize herself with bit parallelized unconstrained approximate pattern matching using Levenshtein distance. This was a more difficult problem to grasp than first expected, and after having programmed what the author thought was an implementation of the algorithm and realizing it was nothing but that algorithm, the author had to start from scratch to truly understand the algorithm.

In addition to the personal learning experience, the author was restricted by time and cost. For the first half of writing this thesis, the delivery date was set to be extended an additional semester. But having the epiphany of the algorithm described above expedited the delivery date by 6 months. This determined the beautification of the hardware programming code, and the size of the test text to be used in the experiment. Also, the board used for development and testing, is a cheap student FPGA board provided by Norwegian University of Science and Technology, NTNU. A more costly industrial board and testing environment would make the results more accurate and more up to scale with a real world example.

In terms of the methodology chosen, some limitations must be mentioned in this context. Firstly, the small size of the data sample used for testing. This size does not represent a real spam filter scenario, but it simulates a real scenario. If the results from this thesis were not so distinct, the

size of the data sample used for testing would be a possible error source. In other words, the generalization from the results could be questionable.

2.8 Conclusion

This chapter has outlined and justified the research methodology applied in this thesis. Because of the nature of the research problem, the author opted for the quantitative strategy, bound by the deductive method. The key research tools were literature review and scientific experiment supplemented by the interpretivist approach. The data set used for testing and experimenting was selected to simulate a real world spam filter scenario. The key results and findings of this thesis are described and discussed in the following chapters.

3 Literature Review

The following chapter presents relevant background theory to illuminate the research topic. Firstly it describes spam, its history, state of the art, and the current situation of anti-spam techniques. Secondly it gives an introduction to Field Programmable Gate Arrays, FPGA, followed by a comparison of different types of processors. Finally a section is devoted to the approximate search problem. This section explains the unconstrained approximate pattern matching using Levenshtein distance, the bit-parallelization of this algorithm, and the simulation of a non-deterministic finite automata, NFA. At the end of this chapter, a recapitulation of the most relevant background theory is provided.

3.1 Spam

With the continuous growth of internet, email spam, also referred to as unsolicited commercial email, is a problem for both email providers and end users. Organizations can utilize spam for economical reasons, because spam is a cheap way of advertising. Most email-users are exposed to unwanted messages in their email inbox. Actually about 50 % of the inbox consists of email spam [1]. The whole point of spam is that the recipient is not specifically intended to be the recipient. The email is structured in a way that can give anonymity, which can be an advantage to advertisers and spammers. Sender data can easily be spoofed, and remote controlled computers can be exploited to send emails in bulk. When email was made, it was not meant to be a solution connected to today's internet. New internet related applications face strict security requirements, which email never had to face. This makes email applicable to spammers. Email spam is a problem for a variety of reasons. It is a waste of network resources and time of end users. Additionally, an email spam can contain malicious content, ransomware and phishing. A report mentioned in [30] found that spam can be characterized to eight general categories. These are investment, adult, finance, products, health, computers, and leisure/travel.

Governments and law are trying to limit spam's damage potential with international and national laws and regulations. Danish supreme court have fined the telecommunication company *Debitel* a ticket of 40 000 danish kroners for mass sending approximately 50 000 commercial text messages and emails [31]. In U.S.A a man was sentenced to nine years of prison for sending millions of unwanted emails [31]. In Norway, the marketing act forbids commercial activity to send

Keyword	Insertion	Deletion	Substitution
shipping	shippping	shpping	shlpping
Invoice	invvoice	invoce	involce
Login	loggin	logn	lugin

Table 1: Obfuscated spam keywords

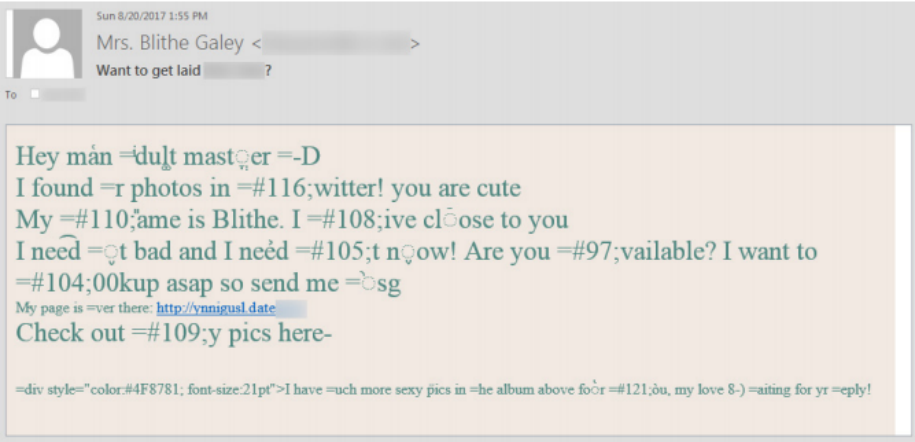

Figure 3: Example of spam e-mail [1]

commercial email to consumers without their prior consent [32]. The new European Union General Data Protection Regulation, GDPR, EU's new privacy law from may 25. 2018, makes the regulations even more stringent [33]. However, the GDPR will not stop spam containing bogus and malware, because these spammers have never, and will never follow a law or regulation. The GDPR will only stop spam emails from organizations and companies that are obliged to comply with the law, where consent is not given clearly [33]. The counterpart to this type of spam, the anonymous spam often comes from criminal minds that remotely control bot nets. Bot nets are a collection of computers where the owner of the computer is lured into installing a type of malware that allows unauthorized people to control and use the computers resources. Bot nets are used to send spam, but also other sorts of digital vandalism.

3.1.1 Spam History

In springtime of 1978, a man named Gary Thuerk wanted everyone to know about his company, namely DCE company [2]. He sent the first unsolicited bulk email to 400 recipients, years before email was colloquial and part of the public domain. He said he thought of it as e-marketing. And now decades later it is the easiest, cheapest and most prevalent use of e-marketing.

> "*I thought of it as e-marketing.*" Gary Thuerk

The linguistic significance of the word spam is a reference from a famous sketch in Monty Pythons TV series the flying circus [30]. A couple at a cafe are trying to order a dish without spam, spam being canned ham in this context. Communication between the waiter and the couple drowns in noise from a group of vikings singing with increasing volume "spam, spam, spam [...]". In the same way as the word spam is used in this sketch to drown normal communication in the cafe, the word spam is used to describe how unsolicited bulk email is drowning normal communication on the internet. Figure 4 shows the last four recipients and the rest of the email sent by Gary Thuerk.

```
YONKE@USC-ISIB
YOUNGBERG@SRI-KA
ZEGERS@SRI-KL
ZOLOTOW@SRI-KL
ZOSEL@LLL-COMP
DIGITAL WILL BE GIVING A PRODUCT PRESENTATION OF THE NEWEST MEMBERS OF THE
DECSYSTEM-20 FAMILY; THE DECSYSTEM-2020, 2020T, 2060, AND 2060T.  THE
DECSYSTEM-20 FAMILY OF COMPUTERS HAS EVOLVED FROM THE TENEX OPERATING SYSTEM
AND THE DECSYSTEM-10 <PDP-10> COMPUTER ARCHITECTURE.  BOTH THE DECSYSTEM-2060T
AND 2020T OFFER FULL ARPANET SUPPORT UNDER THE TOPS-20 OPERATING SYSTEM.
THE DECSYSTEM-2060 IS AN UPWARD EXTENSION OF THE CURRENT DECSYSTEM 2040
AND 2050 FAMILY. THE DECSYSTEM-2020 IS A NEW LOW END MEMBER OF THE
DECSYSTEM-20 FAMILY AND FULLY SOFTWARE COMPATIBLE WITH ALL OF THE OTHER
DECSYSTEM-20 MODELS.

WE INVITE YOU TO COME SEE THE 2020 AND HEAR ABOUT THE DECSYSTEM-20 FAMILY
AT THE TWO PRODUCT PRESENTATIONS WE WILL BE GIVING IN CALIFORNIA THIS
MONTH.  THE LOCATIONS WILL BE:

               TUESDAY, MAY 9, 1978 - 2 PM
                   HYATT HOUSE (NEAR THE L.A. AIRPORT)
                   LOS ANGELES, CA

               THURSDAY, MAY 11, 1978 - 2 PM
                   DUNFEY'S ROYAL COACH
                   SAN MATEO, CA
                   (4 MILES SOUTH OF S.F. AIRPORT AT BAYSHORE, RT 101 AND RT 92)

A 2020 WILL BE THERE FOR YOU TO VIEW. ALSO TERMINALS ON-LINE TO OTHER
DECSYSTEM-20 SYSTEMS THROUGH THE ARPANET. IF YOU ARE UNABLE TO ATTEND,
PLEASE FEEL FREE TO CONTACT THE NEAREST DEC OFFICE
FOR MORE INFORMATION ABOUT THE EXCITING DECSYSTEM-20 FAMILY.
```

Figure 4: The first known spam email sent in 1978 by Gary Thuerk [2]

Drawing parallels to the emails found in today's spam filters, the evolution of spam is not drastic compared to the technological development the world has seen since the 1980's. An example of a modern day spam email is showed in Figure 3. But the use of spam has steadily increased, and has the last ten years steadily yielded fifty percent of the emails sent [1]. The extent of spam grew dramatically from 2003, and email providers and organizations using email had to offer the opportunity for automatic filtering of what was obviously spam. Such spam filters have become more and more effective. Spam fell to its lowest level since 2003 in 2015, and the spam rate has maintained fairly steady since then [1].

3.1.2 Spam Definition

Spam is a buzzword, and an exact international definition of the word is not available. However, the Norwegian Lexicon, defines spam to be bulk mail or other messages that the recipients do not want or have requested [34]. A better and more exact definition of the word spam has been in demand for a while [35] to better align an international anti-spam legislation. The new GDPR discussed in the previous section defines spam to be email sent in bulk that the receiver has not in advanced given consent to receive [33]. Other national legislation, like the one of France, have referred to

spam as the "practice of sending unsolicited emails, in large numbers, to recipients with whom the sender has no previous contact and whose email address was harvested improperly" [30]. In the annual Internet Security Threat Report, ISTR, by Symantec [1], Symantec defines spam to be generally considered as unsolicited email that is sent in bulk and in some cases may not contain malicious threats [1]. Due notice the *may not* contain malicious threats, which implies that most spam seen today contains malicious content.

In this thesis, spam is is referred to with the broad definition that spam is mass mailing of unsolicited messages and is identified by these three general characteristics

1. Spam is an electronic message.
2. Spam is unsolicited.
3. Spam is sent in bulk.

3.1.3 Email Process and Susceptibility to Spam

The technology that email is built upon was developed in a time when network based communication was characterized by trust. Email is an insecure technology that is easy to exploit, spoof and manipulate [9]. The original email infrastructure was not designed for the overwhelming use of email and email end users. Security challenges were, as with all early implementations of internet applications, not considered. These challenges include authentication, integrity, secrecy, confidentiality and bulks of emails.

Figure 5 shows a simplified representation of an email transfer between users. The sender sends an email from his mail user agent, MUA. The email is transferred from MUA to a mail transport agent, MTA. The MTA checks the recipients domain, for example *gmail* uses a domain name server, DNS, to look up the ip-address of the recipients email server provider. DNS sends the ip-address back to the MTA, and the MTA sends the email to the recipient using simple mail transfer protocol [35]. The recipient's MTA delivers the email to the service responsible for delivering and storing email in the recipients inbox. This is the mail delivery agent, MDA. The two most common protocols for fetching email from an inbox is POP3 and IMAP [35]. The recipient's MUA accesses the mailbox using POP3 or IMAP and reads the email.

The SMTP protocol was designed in an environment never intended to be used in today's Internet. Security challenges related to the use of Internet were not considered at that time. SMTP allows spoofing of email sender, and this gives the basic for anonymity and disclaimer [9]. IP addresses are also often spoofed.

3.1.4 Anti-Spam Measures

Various anti-spam measures are used to overcome the problem of spam. No anti-spam measure has proven to be a complete solution to the problem , and each solution has trade-offs between incorrectly classifying a legitimate email as spam, and not classifying spam emails as spam. Respectively called false positives and false negatives. Incorrectly classified spam can be costly in terms of time spent to manually sort an inbox, but also costly when losing important emails that have been sorted to the junk.

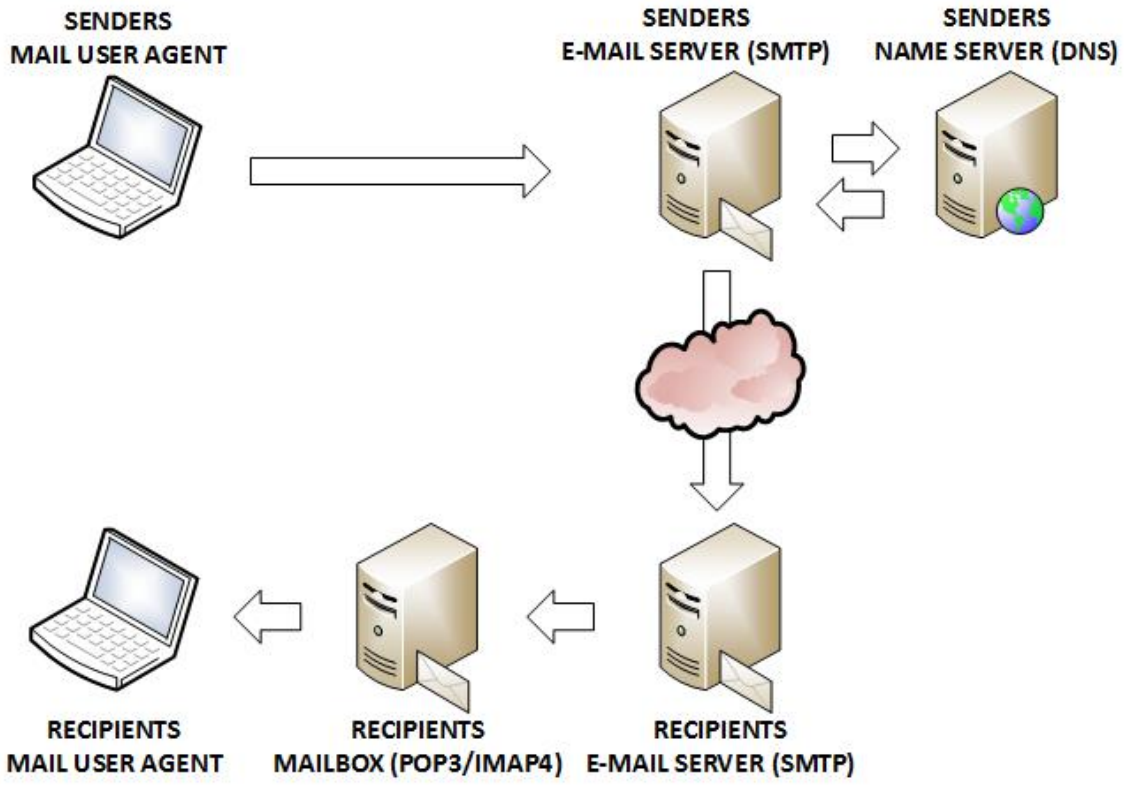

Figure 5: A simplified representation of email transfer between users

Recognizing a spam email is relatively easy for the human brain. However, a glance of the subject or body of the email is normally all it takes to classify the spam email. This is very cumbersome when hundreds of emails are received every day. Besides, spammers have adapted not only to the anti-spam measures. Spam emails written in the native language of the receiver makes the spam more tempting to read.

Besides sorting spam from ham manually, the methods of anti-spam can generally be divided into four groups. These groups are legislative measures, organizational measures, behavioural measures and technical measures [30]. Legislative measures are those measures addressed by law and regulations. Organizational measures are those that can be automated by email administrators. Behavioral measures are those that require actions by individuals. Technical measures can encompass for example a spam filter. The following sections describes these four groups in more detail.

Due to the potential damage of SPAM, authorities begun in the early 2000 to address the issue of spam with legislative measures. As mentioned previously in this thesis, the GDPR that came in may 2018, has restricted the commercial use of spam with strict regulations on consent. A general problem concerning legislative measures to prevent spam, is that an international problem is solved with national laws [30]. Western countries solves the problem of spam a little different than the other countries. The GDPR is bound by the principle that only those that have given consent in advance can receive commercial emails, while in the CAN-SPAM act in the USA bulk email without consent is allowed provided that it emerges clearly who is the sender of the email and it contains the possibility to reserve against future emails from the same sender [36].

Technical measures can comprise IP blocking, TCP blocking, authentication, verification, payment-based, limitation of outgoing emails, address obscuring techniques, reputation-based and filtering [30]. Authentication measures can be SMTP extensions [15], cryptographic authentication or path authentication. Payment-based measures can be CPU-based, memory based and monetary. The next chapter describes the technical measure filtering in more detail.

Spam Filtering

Unfortunately spammers are adaptive, and as such vendors must continually tune their filters. Spam filters are applications which aim to sort out spam from ham. The spam filter program is usually run on a server or in connection to an email client on the personal computer of the end user.

A spam filter is a collection of instructions to determine if an email is solicited or unsolicited, ham or spam. Spam filters are implemented to stop spam from being exposed to the end users. The general problem of spam filtering is to avoid false positives.

A spam filter can be instructed to look for specific keywords, the native tongue, phrases or structures [30]. Filtering measures are the most widely used anti-spam measure. A spam filter is specifically tuned for every user group.

Some filters look only at header information, whilst some inspect content, and in most cases, both. One type of filtering is rule based. Rules can be set manually and automatically. Machine learning algorithms can automatically change rules. The pitfall of rule-based filtering is the ability of spammers to obfuscate the keyword, and thus avoid the filter but still be susceptible to the

end user. Statistical filters are based on the Bayes theorem. Other methods for text classification is support vector machines, artificial neural networks, Markow random field models and more [30].

Scoring content-based filters are simple filters that search for specific words and expressions in an email, and use this when deciding if the email is spam or ham. This type of manual analysis has limitations due to its need to process a large amount of messages to have enough basis for its decision.

A Bayesian filter is based on scoring content-based filters, but is more dynamic and hence, effective. Bayesian filters exploits the probability of an event based on knowledge of conditions that may be related to that event [30]. Thus, the Bayesian filter learns from experience, and can adapt and be trained to better recognize a spam email. But Bayesian filtering is also a source for false positives. The evaluation of anti-spam solutions is measured by effectiveness, accuracy and ease of use [30].

Conclusion

Spam filters are a necessity in an email program. This is probably the best way to reduce spam today. A general flaw of spam filtering is the risk of filtering away a solicited email. Spam filters must apply several techniques in order to be effective. To blacklist sender IP and domains can be effective, but not if the senders are spoofed. To sift out spam only on the basis of keywords is also effective, but can also be avoided by devious spammers that obfuscate the keywords. The most effective way to avoid the spam problem is to utilize all the known techniques and use them together. For example, by sifting out emails based on keywords, also the approximate keywords, analyze the sender of that email and the structure of the email.

3.2 Field Programmable Gate Array, FPGA

A Field Programmable Gate Array, FPGA, is a re-programmable integrated circuit. FPGA's have history from mid 1980's, but have evolved from the simplest gates to complicated integrated circuits [37]. FPGA is very similar to an ASIC, but as opposed to these, the FPGA are flexible in their design. The FPGA allows a hardware designer to design physical circuits on a chip, and change the design at any time. The advantage of an FPGA is that it can do a lot of processes in parallel, which allows the developer to send a lot of data to be processed at the same clock cycle. This makes the processing time for an FPGA potentially very fast.

A drawback of FPGA is that they do not remember their design when power is removed. A separate configuration memory chip that holds the design is needed. When the power is given back to the FPGA a fixed part of the FPGA reads the configuration from the configuration memory chip. After reconfiguration, the FPGA will again be able to perform the function it was designed to do.

An FPGA is not a microprocessor or micro controller. This is due to the fact that an FPGA in its basic form is not able to run software. Only when the FPGA is given a configuration that contains a processor-architecture it has the ability to run software.

Because the FPGA is an integrated circuit, an FPGA manufacturer can decide to add other commonly used integrated circuit components. These fixed parts are called cores [37]. These are for

example Ethernet controllers or even a complete processor-architecture. For different market segments such as defense, medical, communications and robotics the FPGA manufacturer tries to add the most valuable set of these additional cores. Student development boards, like the Nexys4 DDR board, has integrated Ethernet, DDR2 SDRAM, Micro SD Card, JTAG and UART to name a few.

The disadvantages of using an FPGA is the cost, and the high power. Also it is complicated, and hardware description language, HDL, is not easy nor intuitive. The board is volatile and there can be many traps for a hardware designer.

3.2.1 Basic FPGA Architecture and Structure

An FPGA is usually composed of configurable logic blocks, CLB, Inn and Out blocks, I/O and interconnection wires [37]. Figure 6 shows the generic architecture of an FPGA. A CLB is a logic block that can execute both basic and complex combinatorial logic gates. Logic gates are the AND, OR, XOR and NOT. I/O are the interface from internal to external signals. Interconnection wires connect input and output from the CLB and I/O. FPGA devices are perfect for parallelism and bit-level operations with high bandwidth.

Hardware programming language, HDL, is used to program an FPGA. Verilog and VHDL is the two most common programming languages. VHDL is an abbreviation of VHSIC HDL which again is an abbreviation of very-high-speed integrated circuit HDL. The principal difference between hardware programming language and software programming language, is that it while, for example C, describes a sequental system, HDL describes a concurrent system [37]. Software code gives instructions to be read into a processor, while HDL code is synthesized to be combinational logic and memory elements [38].

A CLB consists of programmable look-up tables, LUT's [37]. A LUT is programmed to give an output based on input and pre-programmed tasks. A LUT can be any type of logic wanted by the hardware designer. An example is a LUT with two inputs and one output programmed to emulate Boolean-logic and combinational-logic of the two inputs. This can consist of doing an AND, OR, NAND, NOR, XOR. The output is stored in a register or can be used as input to other LUTs. A combination of LUTs, registers and MUX is what makes CLBs, which the FPGA consists of. A MUX is used to select the registered or no-registered output. Pass transistors are used to connect these logic elements. A hardware description language, like Verilog or VHDL described earlier, can construct the logic for the logic elements, and the pass transistors connect them [38].

Another function of the FPGA is the memory, RAM. The RAM is used for writing, storing, and reading data, variables, and the many other transitory numbers with which any device must deal. In addition to the functions mentioned, an FPGA may have some analog functionality. FPGAs are used in a variety of applications and to make them even more attractive, basic analog input and output is added to the board.

3.2.2 General Purpose Processor, GPP

A central processing unit, CPU, is an essential part of a computer. The responsibility of the CPU is to control every other part of the computer, like the functions of a brain in the human body. A CPU

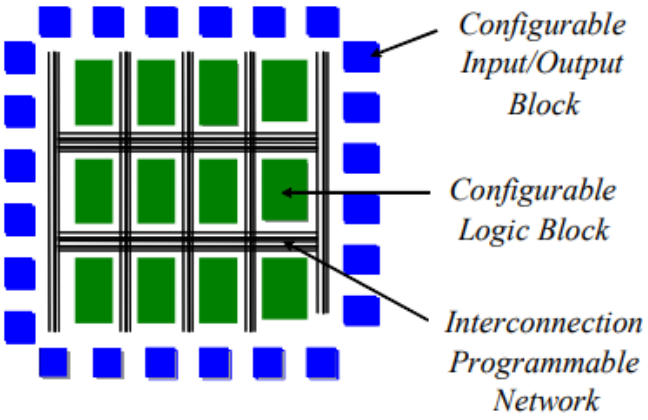

Figure 6: Generic architecture of an FPGA [3]

is a General Processor Unit, GPU, and a General Purpose Processor, GPP, because it is designed to be fairly good at nearly any task it is instructed to do.

The CPU uses lists of instructions to perform [39]. It takes an instruction and executes each instruction in order. This is done in a sequential matter, as opposed to the FPGA described in the previous chapter, which can perform executions in parallel. The total list of instructions to a CPU is what is called a computer program, or software program.

Most up to date processors have 2-3 GHz internal clock. Most cheap, student type, FPGA boards have 100 MHz internal clock, which means that the CPU of a normal personal computer can run 3000 times more cycles per second than that of the FPGA.

3.2.3 Application Specific Integrated Circuit, ASIC

Application Specific Integrated Circuit, ASIC, is another type of integrated circuit. The main difference between an FPGA and an ASIC is the ability to re-program the FPGA. The ASIC can not be reprogrammed. It is designed and built for particular use, and not general purpose. Analog ASICS are not common, but exist. Digital circuit ASICs are what is commonly referred to when speaking of ASIC. ASICs were manufactured from the mid 1990s [7].

The advantages of using ASIC is the small embodiment of the chip [7]. It has the same bit parallel ability as the FPGA, and can improve speed of an integrated circuit. An ASIC does not need much power to perform. But the cost per unit can be high. That is because photo-masks, which are used to make an ASIC, are costly. A photo-mask is a photographic pattern used in making microcircuits, ultraviolet light being shone through the mask onto a photoresist in order to transfer the pattern [7]. It is a pitfall when the desired design result is not achieved after production of the chips, as they are not re-programmable and the entire chip must be reproduced.

There are different ways to design an ASIC. These are gate array design, cell base design, em-

Gate Array design	Is an elementary gate circuit, of wire layers.
Cell base design	Uses wire-layers between blocks. The performance of these are better than the gate array, but is more costly in time and space.
Embedded array design	Uses finished functional blocks.
Standard cell design	Is a general term for gate array, cell base and embedded array.
Structured ASIC design	Is the customization of standard cell design to speed design and production.

Table 2: Different designs approaches of ASIC[7].

bedded array design, standard cell design and structured ASIC design [7]. A description of these designs are showed in Table 2.

3.2.4 FPGA vs. ASIC vs. GPP

When choosing the primary processor for digital design system, three types of processors to consider have been presented in the previous sections. There are more specific types to include in this opposition, but the three mentioned here are a general inclusion of the main domains of processors.

FPGA is a quick way to implement a hardware solution, with the possibility to test and try and retest and retry. The FPGA yields faster production time and cost than ASIC, as well as lower non-recurring engineering, NRE. Routing and time problems can occur with complex or highly interconnected systems. On this basis, FPGA are appropriate to use for small to medium size productions [4].

Traditionally, ASICs are faster, more energy efficient and have achieved more functionality than the first FPGAs. But newer FPGAs, such as the board used in this thesis, or other newer boards by Xilinx[5] or Altera, have come to rival the ASIC solutions [40]. New FPGAs provide significantly reduced power usage, increased speed and lower cost. A hardware designer can now use one FPGA, where previously one would have used ten [40].

FPGA gives flexibility in programming as opposed to ASICS which have fixed design. The flexibility of an FPGA board makes the possibility of malicious modifications during fabrication a lower risk [40]. The advantage of using an FPGA over ASIC in a development setting is the reduced cost due to the fact that the FPGA can be reprogrammed, and a new board does not need to be made when development in a project is made. An FPGA requires more power consumption compared to an ASIC. This is because an FPGA board needs to be bigger in order to be re-configurable. As such FPGA boards are suitable, as mentioned, for small development projects, whereas ASICs are preferred for bigger projects. Whilst ASIC boards consist of mixed signals, linear and digital, FPGA are only digital. Digital design projects that incorporate complex computation, with demanding system performance, are appropriate projects to implement on FPGA architecture [4].

Figure 7 illustrates the pros and cons of the three integrated circuit choices. When design flexibility is the main priority, General Purpose Processors are the best choice. However, if performance, and efficiency is the priority, ASICs and FPGAs are preferable. FPGAs lie in the middle in this com-

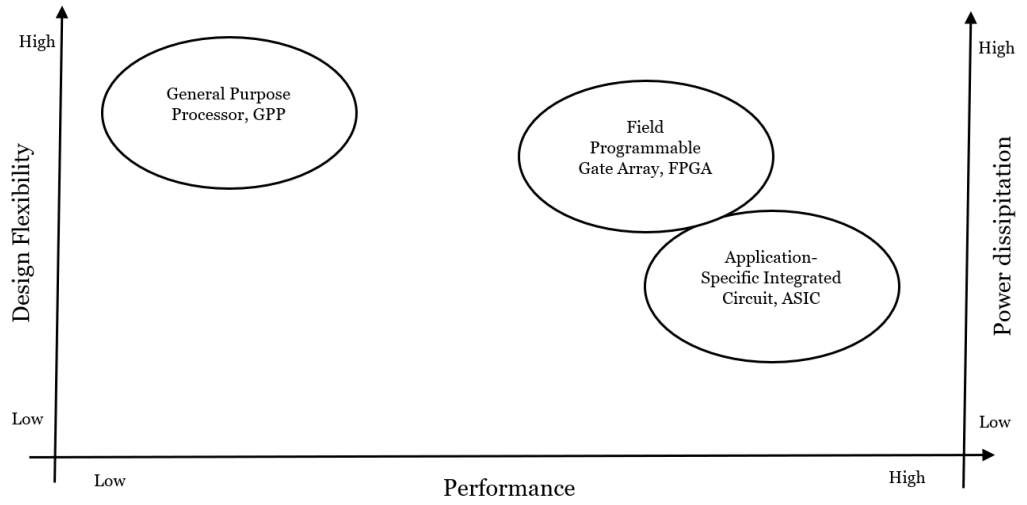

Figure 7: A comparison of different types of processors, Figure adapted from [4]

parison and provide high performance and high design flexibility. They require more power. It can be considered as a compromise between flexibility and performance.

FPGAs and ASICs can process in parallel, whilst GPP are sequential. If the embedded system that is to be designed can utilize bit parallelism, FPGA and ASIC will be of even greater speed efficiency. Knowing that hardware description languages are difficult, and not intuitive, and that the design of such algorithms must often be tested and reconfigured to achieve the best desired results, FPGAs are to be preferred. If a specific acceleration unit, is to be built, flexibility can be traded for performance with an ASIC [4].

Since FPGAs have followed Moore's law to the point [4], system-on-chip, SoC, designs are built on single devices, such as the Nexys4 DDR used in this thesis. These devices can compete with the traditional ASIC devices and offer advantages over the general purpose processors. These advantages are lower production cost per operation per second. It has also been proved in science that they perform better per unit of silicon than processors [4].

String matching algorithms have proven to be optimized several times for approximate text search on general purpose processors [14] [15] [16] [20]. Bit parallelizing the approximate search results in a drastic reduction of execution time [4]. But, the memory subsystems on ASIC and FPGA suffer from a poor memory performance [4].

To summarize, general purpose processors are designed for general purpose computers. The computation speed of a GPP is considerably higher than that of an ASIC or FPGA. The cost of a GPP is also normally higher, than that of an ASIC or FPGA. GPPs are not designed for fast real-time applications [4]. An FPGA is more flexible than an ASIC, and the flexibility is an easy trade for the higher power consumption. An FPGA has higher performance than GPP, and the performance is an

easy trade for the design flexibility when time is of the essence.

3.3 Approximate Search

This section will explain the algorithm used to find approximate matches in the proposed spam filter. The algorithm used in the proposed spam filter is the bit parallel unconstrained approximate search using Levenshtein distance simulating a non-deterministic automata.

3.3.1 Approximate string matching

Approximate string matching, is often referred to as string matching allowing errors [21]. It is also referred to as Fuzzy string searching [41]. Baeza-Yates and Ribeiro-Neto [14] describe approximate string matching in the following way: A text is a sequence of symbols or characters from an alphabet Σ, such that when a query string Q of length m shall be found in a text T of length n, string matching shall count all occurrences of Q in T [14]. Classical string matching algorithms are based on character comparisons [4]. Approximate string matching differs from exact string matching in the essence that it aims to find approximate matches. An approximate match can be a string that is equivalent to the keyword that is used to find a match. If a word can be replaced with another word without changing the meaning of the content, the two words are equivalent. Examples of equivalent words are name variants, special characters, typos, nicknames, phonetic similarities, misspelling and name changes [42]. To use a method that finds equivalent strings has many areas of use. In biology forensics, one could use an approximate string matching to find possible mutations of DNA, and in the musical division, find similar string tones [21]. The approximate string matching problem is defined as finding a sub-string from a given pattern and a given text that has the least edit distance to the pattern. The objective is to find all approximate occurrences of the pattern in the text. An approximate occurrence is the sub-string, in which a set of maximum of edit operations are done to make the pattern become the sub-string [43].

3.3.2 Edit Distance

To convert one word or string to another, changes have to be done to the word unless it is the exactly the same word. Three main types of changes can be applied to a string or word. These are character insertions, deletions and substitutions, and are denoted edit operations. The edit distance can generally be defined as the least amount of changes that must apply to change one word or string to another. An insertion is to add a symbol or a letter. A deletion is to remove a symbol or a letter. A substitution is to exchange a symbol or letter from the text. For example the edit distance of spam and Pamela is four. Because it takes one deletion (s) and three insertions (ela) of symbols to convert spam to pamela. Obviously, with enough insertions, deletions and substitutions, any string can be changed to another. To reduce results of a text search, a set of maximum of edit operations k is set as maximum edit distance [43]. The approximate string matching problem only makes sense when $0 < k < m$, when m is the length of the pattern to be matched. This is because every word can be converted to another by editing all the symbols in a word. When $k = 0$, there is an exact match. One difference from one word to another equals one edit operation [21]. The edit operations are

Insertion	pam → s̲pam
Deletion	spam → _pam
Substitution	spam → spi̲m

Table 3: An example of edit operations

showed in Table 3.

There are many existing models defining a difference[21] [28] [43] , but the focus of this thesis is the Levenshtein distance [28]. The Levenshtein distance is the minimum number of edit operations needed to change one word into another [44]. Levenshtein distance is often referred to as *edit distance*, but the term *edit distance* denotes a wider range of string metrics. The mathematical formula for Levenshtein distance is showed in Equation 3.1. The Levenshtein distance between the prefixes of two strings, a and b, of length i and j (respectively), is given by $lev_{a,b}(i,j)$ [44].

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & if min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{a_i \neq b_i} \end{cases} & otherwise \end{cases} \tag{3.1}$$

3.3.3 Nondeterministic Finite Automata, NFA

The oldest solution to the approximate string matching problem uses dynamic programming [21]. This algorithm uses $\mathcal{O}(mn)$ time, and is theoretically not the most efficient solution to the problem. Despite this, this solution is rediscovered multiple times [11] [28].

An alternative way to consider the approximate string matching problem is to model the search with a non-deterministic finite automation, NFA [21]. This automation was first proposed in [43] and later in [17] [28].

Figure 8 shows the NFA, with $k = 3$ allowing 3 errors. Every row denotes the number of errors seen [21]. A state is active when a pattern match occurrence is found. Horizontal arrow represents a match, vertical arrow represents an insert, solid diagonal arrow represent substitution, and dashed diagonal arrow represents a deletion. The initial self-loop allows an occurrence to start anywhere in the text [21]. A double sided circle is used to denote an end state.

The shift-Or algorithm in Formula 3.3 simulates a run of the NFA for the approximate string matching. In Shift-Or algorithm for the approximate string matching there is for every row of states of NFA one bit vector R_i^l [28]. For each level of states there is one bit vector to depict the set of active states. If $r_i^l = 0$, the final state of the l-th level is active. An active state indicates that the pattern is found with at most l errors ending at position i in the text. An example NFA searching for pattern $P = spam$ is found in Figure 8.

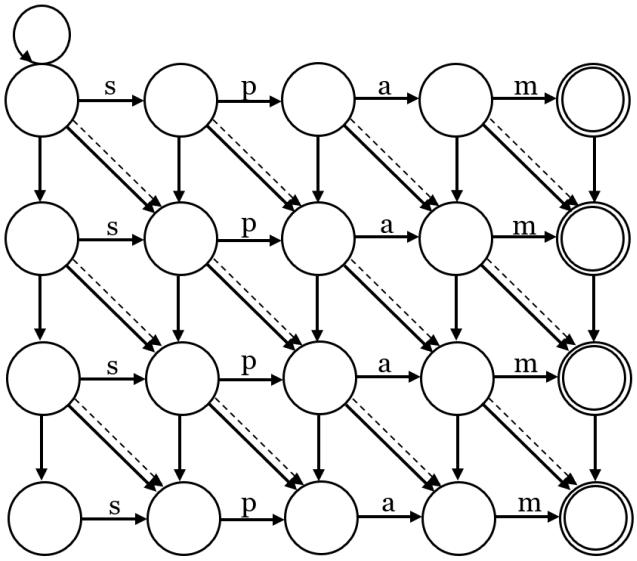

Figure 8: NFA spam

3.3.4 Bit Parallelism

Holub describes bit parallelism as a method that uses bit vectors and benefits from the feature that the same bit-wise operations over groups over bits can be performed at once in parallel over the whole bit vector [28]. Bit parallelism has been used for the purpose of exact string matching [11], the multiple exact string matching [11], the approximate string matching using both Hamming distance [28] and Levenshtein distance [28].

Approximate bit-parallel search

Bit-parallelism has been used a lot in solving the approximate string matching problem. The results are most useful for short patterns [21]. To solve the research question, an approximate bit-parallel search algorithm is needed. The shift- And algorithm and the Shift- Or algorithm, is the same only that the meaning of 0 and 1 is exchanged. For this thesis, the Shift- Or algorithm will be used consequently. The shift- Or algorithm is described by Holub [28] as follows: The Shift- Or algorithm uses matrices R^l, $0 \leq l \leq k$ of size $m \times (n + 1)$ and mask matrix D of size $m \times |\Sigma|$. Each element $r^l_{j,i}$, $0 < j \leq m$, $0 \leq i \leq n$, contains 0, if the edit distance between string $p_1 ... p_j$ and string ending at position i in the text T us $\leq l$, or 1, otherwise. Each element $d_{j,x}$, $0 < j \leq m$, $x \in \Sigma$, contains 0, if $p_j = x$, or 1 otherwise. The matrices are implemented as tables of bit vectors as follows:

$$R_i^l = \begin{bmatrix} r_{1,i}^l \\ r_{1,i}^l \\ \vdots \\ r_{1,i}^l \end{bmatrix} \text{ and } D[x] = \begin{bmatrix} d_{1,i}^l \\ d_{1,i}^l \\ \vdots \\ d_{1,i}^l \end{bmatrix}, 0 \leq i \leq n, 0 \leq l \leq k, x \in \Sigma. \tag{3.2}$$

Approximate String Matching Using Levenshtein Distance

To compute vectors $R^l{}_i$, $0 \leq l \leq k$, $0 \leq i \leq n$ in approximate string matching using Levenshtein distance Formula 3.4 is used. The auxiliary vector V is used to prevent insert transitions leading into final states.

$$\begin{array}{lll} r_{j,0}^l \leftarrow & 0, & 0 < j \leq l, 0 < l \leq k \\ r_{j,0}^l \leftarrow & 1, & 0 < j \leq l, 0 < l \leq k \\ r_{j,0}^l \leftarrow & \textbf{shl}(R_{i-1}^0) \text{ OR } D[t_i], & 0 < j \leq l, 0 < l \leq k \\ r_{j,0}^l \leftarrow & (\textbf{shl}(R_{i-1}^0) \text{ OR } D[t_i]) \text{ AND } \textbf{shl}(R_{i-1}^{l-1} \text{ AND } R_i^{l-1}) \text{ OR } (R_{i-1}^{l-1} \text{ OR } v), & 0 < j \leq l, 0 < l \leq k \end{array} \tag{3.3}$$

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}, \text{ where } v_m = 1 \text{ and } v_j = 0, \forall j, 1 \leq j \leq m. \tag{3.4}$$

In the formula 3.3 **shl**() is the bit-wise operation *left shift* that inserts 0 at the beginning of the vector and operation OR is the bit-wise operation OR. In the same formula, 3.3, the term $\textbf{shl}(R_{i-1}^l)$ OR $D[t_i]$ represents matching, term $\textbf{shl}(R_{i-1}^{l-1})$ represents edit operation replace, term $\textbf{shl}(R_i^{l-1})$ represents edit operation delete - position in pattern P is increased, position in text T is not increased, and edit distance l is increased. The term $\textbf{shl}(R_{i-1}^{l-1})$ represents edit operation insert - position in pattern P is not increased, position in text T is increased, and edit distance l is increased. The term OR V ensures that no *insert* transition leads from any final state [28].

Pattern P is found with at most k differences ending at position i if $r_{m,i}^k = 0$, $0 < l \leq n$. The maximum number of differences of the found string is l, where l is the minimum number such that $r_{m,i}^k = 0$.

D	a	e	m	p	r	s	Σ
s	1	1	1	1	1	0	1
p	1	1	1	0	1	1	1
a	0	1	1	1	1	1	1
m	1	1	0	1	1	1	1
m	1	1	0	1	1	1	1
e	1	0	1	1	1	1	1
r	1	1	1	1	0	1	1

Table 4: Matrix D for pattern P = *spammer*

An example of matrices R^l for searching for pattern P = spam in text T = *spamfromaspammer* with at most $k = 3$ errors is shown in Table 5

The first row of Figure 8 can be considered as an NFA for exact string matching. This row is also the first column in Table 5. The update formula for this NFA is the Shift-Or approach showed in the first row of Algorithm 3.3. Row-wise bit-parallelism, packs each row i of the NFA in a different machine word R_i, with each state represented by a bit. For each new text character, all the transitions of the automation are simulated using bit operations among the $k+1$ bit masks. The update formula to obtain the new R_i^1 values at text position j from the current R_i^1 . The update formula is shown in Algorithm 3.3. R^0 undergoes a simple Shift-or process, while the other rows receive zeroes, which represents active states, from previous rows as well. In the approximate string matching using the Levenshtein distance a new NFA is constructed for every k. An example of mask matrix D for pattern P = *spammer* is shown in Table 4 . The Bit parallel NFA uses one bit vector R for each level of states of NFA.

R^0	-	s	p	a	m	f	r	o	m	a	s	p	a	m	m	e	r
s	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
p	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1
a	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1
m	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1
m	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
e	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
r	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

R^1	-	s	p	a	m	f	r	o	m	a	s	p	a	m	m	e	r
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1
a	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1
m	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1
m	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	0	1
e	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
r	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

R^2	-	s	p	a	m	f	r	o	m	a	s	p	a	m	m	e	r
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1
m	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1
m	1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0	0
e	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0
r	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	0	0

R^3	-	s	p	a	m	f	r	o	m	a	s	p	a	m	m	e	r
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
e	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0
r	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0

Table 5: Matrix R^l for the approximate string matching using the Levenshtein distance (Σ version, $P = spammer$, $k = 3$, *and* $T = spamfromaspammer$.)

3.4 Summary

In this chapter the main topics spam, Field Programmable Gate Array, FPGA and approximate search have been explained. Spam is, despite research and development, and will continue to be a problem. Intelligent spammers finds new ways to surpass anti-spam measures, for example by obfuscating typical spam keywords that are detected by spam filters. An anti-spam solution uses various anti-spam measures in union, to collaborate on the filtering task.

An FPGA is a re-programmable integrated circuit. An FPGA has high performance and processes in a bit parallel manner. An FPGA has most of the advantages of an ASIC, but is more flexible. A GPP is more flexible than the FPGA, but this flexibility is an easy trade for the higher performance of the FPGA. FPGA are programmed in hardware programming language, which is not as intuitive widespread as software programming such as C or Java.

Approximate search finds approximate matches of a pattern in a search text. The oldest solution to the approximate string matching problem uses dynamic programming, but an alternative way to consider the problem is to model the search with a non-deterministic finite automation, NFA. The NFA can be bit parallelized using one bit vector for each level of states of NFA. Bit parallelism uses bit vectors and benefits from the feature that the same bit-wise operations over groups over bits can be performed at once in parallel over the whole bit vector.

4 Implementation

The first step towards a spam filtering design was designing in Verilog a Finite State Machine with approximate search. But despite the theoretical knowledge possessed of the computations of the approximate search algorithm, implementing it onto the FPGA board was a complicated design procedure. In order to identify the bottlenecks of the program, Vivado tools was chosen as a design platform. Vivado gave the possibility of verifying the code in a behavioral simulation using test benches. This chapter describes the implementation process that was undertaken to implement and program the FPGA board.

4.1 FPGA Design Flow

Compared to software design, a longer process is undergone before the design can be used on the FPGA board. For the project carried out in this thesis Vivado 2017.4 [6] design suite was chosen to program the Nexys4 DDR. The entire solution is integrated within a graphical user interface, GUI, known as the Vivado Integrated Design Environment, IDE. The Vivado IDE provides an interface to assemble, implement, and validate the design and the integrated process [6]. The general design flow of a Vivado project is depicted in Figure 9. The hardware description language, HDL, Verilog, was chosen as programming language. When Verilog hardware description language was created, it was created to be similar in syntax to the software programming language C [6]. The object was to enable designers to quickly write descriptions of large circuits in a relatively compact and concise form.

4.1.1 Design Entry

The first step when programming hardware is design entry. The program can be programmed in hardware language, or schematic or a mix of both. As mentioned above, this study was conducted in the Verilog hardware description language.

4.1.2 Simulation

The simulation part of the design flow allows the designer to verify the output of the design prior to programming the FPGA board. Behavioural simulation with test bench has been of great help in the process of writing the code to be used in this thesis. Programming the board directly does not yield the same benefit as analyzing behavioral simulation using a test bench. Figure 10 shows a screen-shot from an analysis of a simulation during development of the program. Being able to analyze bit level computations with a simulation run using a test-bench allows the designer to verify if logical operations of the bits are correctly conducted. Bit-level analysis is crucial in identifying errors when designing complex algorithms.

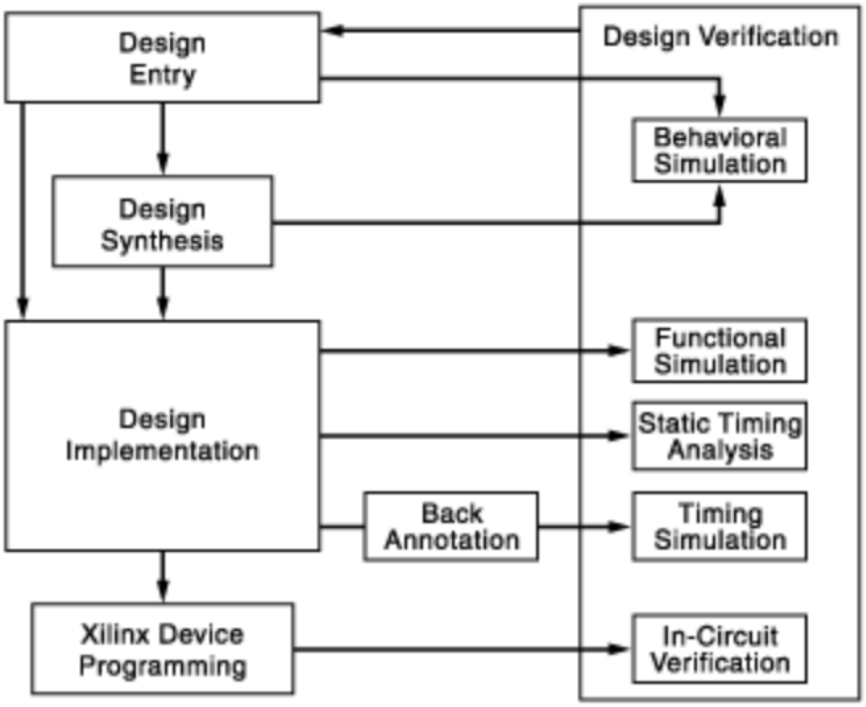

Figure 9: FPGA design flow [5].

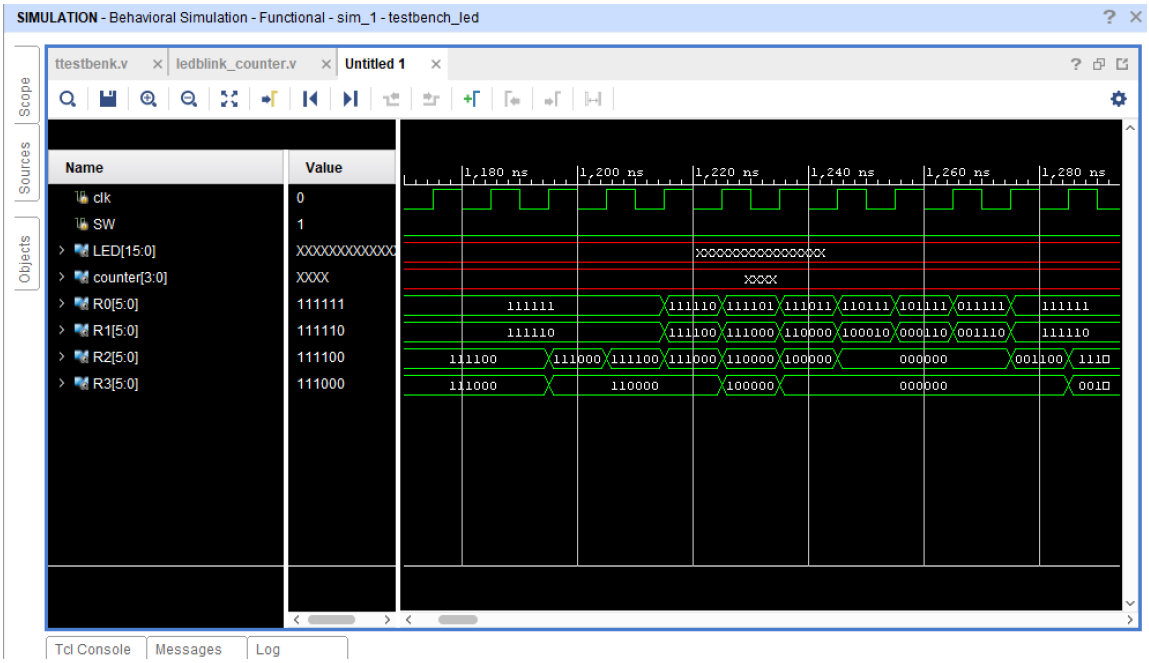

Figure 10: results from run of FPGA behavioural simulation with test-bench

4.1.3 RTL Analysis

Register-transfer level, RTL, analysis allows the designer to see how the tools used interpret the code. In the register transfer level viewer, the synthesized design is depicted in a schematic way. This high-level representation displays gates and elements independently of the targeted FPGA board. The RTL shows the flow of the digital signals between hardware registers, and the logical operations that are performed on these signals [6].

4.1.4 Synthesis

The Vivado design tool performs a global or top-down synthesis of the overall RTL design [6]. When IP cores from the Xilinx IP catalog are used, or block designs from the Vivado IP integrator is used, Vivado utilizes an out-of-context, OOC, design flow. This OOC can also be chosen to synthesize specific modules of a hierarchical RTL design, but this was not chosen for the design flow used in the work of this study.

The Vivado IDE displays the synthesis results and creates report files. Warnings and errors are presented in a messages window, and points the designer directly to the error prone logic in the RTL source files. The Vivado synthesis allows for multiple, concurrent or serial synthesis runs. Choosing multiple synthesis runs creates multiple net-lists. Net-lists can be viewed in the IDE and gives basis for analysis. The synthesis also allows the designer to create constraints for in and out pin planning, timing, floor planing and implementation [6].

4.1.5 Implementation

The design implementation part of the design flow gives the designer access to implementation settings and post-implementation reports. The Vivado implementation is a step wise explanation of placing and routing the net-list, or net-lists onto the resources available on the selected device.

4.1.6 Device Programming

The last step of the FPGA design flow is the device programming. This step generates the bit stream and opens the hardware manager. Hardware manager targets the selected board, and after implementation of the design, the bit stream is loaded onto an FPGA board. Debugging the design in-system is the last error correction of the flow, but the re-programmable quality of the FPGAs allows the designer to rerun the design process if errors are found after device programming.

4.2 System Architecture

The project carried out with the work of this thesis was conducted on a Nexys4 DDR FPGA board by Digilent. Figure 11 is a picture of the board used in this research. The Nexys 4 DDR board, is a student-focused FPGA design kit. Nevertheless, it consists of generous external memories. It is considered a high capacity FPGA due to its collection of USB, Ethernet and other ports. The clock frequency of the Nexys 4 DDR is 100Mhz.

4.2.1 System Overview

To simplify the experiment process, the spam corpus (search text) was stored in ram upon implementation. Prior to implementation the spam corpus was paraphrased to hex using ASCII code.

The first part of the hardware program makes a D-matrix, such as the one described in Chapter 3 Table 4 based on the pattern P. Pattern P is the spam keyword to classify spam from ham. These spam keywords are found in the data set, obscured with up to three insertions, deletions or substitutions.

The second part of the program runs the ASCII stream, one symbol per clock cycle, and calculates all the rows R^{l} based on the shift or algorithm also described in chapter 3. A counter counts and marks hits throughout the search. A hit is found when the most significant bit, MSB, of each R^{l} is 0. To simplify the design process of the hardware program, led lights were used to indicate hits. There are four components in this spam filter these are:

1. Search text (captured emails)
2. Search pattern (spam keywords)
3. Levenshtein distance approximate string matching algorithm function
4. Counter output

4.2.2 Approximate Search Function

This function is the core of the project. The core function of this module is to conduct string matching operation between the pattern stored in RAM on the FPGA board and the email body data set. For simplicity of the project, the spam corpus is stored in RAM on the FPGA board. It will find the

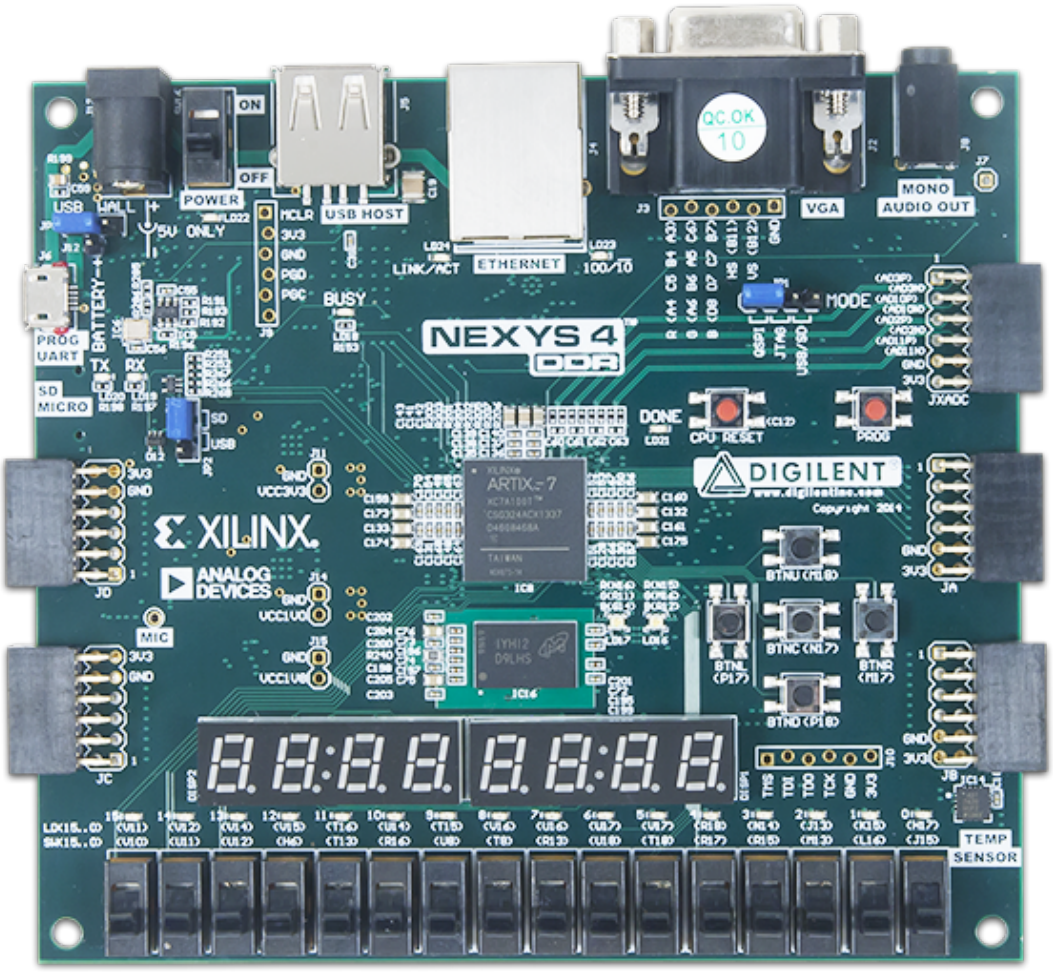

Figure 11: Nexys4 DDR [6]

similarity of these two strings by calculating the distance between the strings and compute the mark from the distance. During computation it will count the number of instances of the pattern found in the text and initiate LED lights to indicate finds. The update formula is presented in Listing 1:

```
if ( rom[i] == 8'h\Sigma )
   begin
   R0temp[i] =  ((R0temp[i-1] << 1) | D_n);
   R1temp[i] = ((R1temp[i-1] << 1) | D_n) & ((R0temp[i-1] &
            R0temp[i]) << 1) & (R0temp[i-1] | V) ;
   R2temp[i] = ((R2temp[i-1] << 1) | D_n) & ((R1temp[i-1] &
            R1temp[i]) << 1) & (R1temp[i-1] | V) ;
   R3temp[i] = ((R3temp[i-1] << 1) | D_n) & ((R2temp[i-1] &
            R2temp[i]) << 1) & (R2temp[i-1] | V) ;
   end
```

The code is a codification of the algorithm found in chapter 3 equation 3.3. The line `((R0temp[i-1] << 1) | D_n)` represents matching, the line `((R1temp[i-1] & R0temp[i])` represents edit operation replace, the line `R0temp[i]) << 1)` represents edit operation delete - position in pattern P is increased, position in text T is not increased, and edit distance l is increased. Term `R0temp[i-1]` represents edit operation insert - position in pattern P is not increased, position in text T is increased, and edit distance l is increased. Term OR V ensures that a final state can not lead to an *insert* transition. An example of the update formula is shown in the following example of R^0 update in Listing 2:

```
R0[0]                                       1111111
(R0temp[i-1] << 1)                          1111110
|
D_s                                         1111110
-------------------------------------------------
R0[1]                                     = 1111110
```

An example of an update of row R^1 from $R^1{}_0$ to $R^1{}_1$ is shown in the following Listing 3:

```
R1[0]                                       1111110
(R1temp[i-1] << 1)                          1111100
|
D_s                                         1111110
-----------------------------------------------------------------
=                                                       1111110   &
     (R0temp[i-1]                           1111111
     &
     R0temp[i])                             1111110
                                          = 1111110
-----------------------------------------------------------------
         << 1)
=                                                       1111100   &
        (R0temp[i-1]                        1111111
        |
        V                                   1000000
-----------------------------------------------------------------
=                                                       1111111   &
-----------------------------------------------------------------
R1[1]                                                 =   1111100
```

5 Experiment

The experiment conducted in the work of this thesis was carried out to help answer the research question of the effectiveness and accuracy of a hardware implementation of a spam filter. No similar programs were found in the literature review. The aim of this chapter is to describe the experimental process to enable the reader to reproduce the results also found in this chapter. The results from this experiment are discussed in Chapter 6.

5.1 Dataset

To test spam filters, large bodies of collected email messages, a corpus, are most frequently used. Each corpus contains messages that are either spam or ham (non spam). The entire corpus is sent through the filter, and count is kept of spam keywords. In the experiment, the patterns used for testing are the spam keywords shipping, invoice and invest. These spam keywords are from the Symantec Internet Security Threat Report from 2017 [1]. The corpus used is a representation of email bodies encompassing *normal*, non-spam, words and spam keywords. The spam keywords are repeated throughout the text, with different types of changes. The changes are a combination of insertions, deletions and substitutions to assure the edit distance function works properly. Using a corpus like the one used in this thesis is the best choice for this experiment due to multiple reasons: It is easy for other researchers to verify the results in this experiment, by replaying the experiment with the same spam keywords in a mix of other normal words. Since the spam filter only classifies based on keywords, this corpus is modified to contain 2 exact keywords, 2 obfuscated keywords and the remaining words are non-spam keywords. The spam corpus consisted of five constructed emails. This amounted to 477 letters. A fragment of the spam corpus used in this experiment is shown in Figure 12.

```
hello my friend can we meat for a dinner this wednesday?
school is very boring lately
buy cheap shipping of pills from our company
```

Figure 12: Example of email bodies used in spam corpus

5.2 Performance Measurement

The common performance measures for evaluating spam filters are spam recall, spam precision, false positive, false negative and accuracy [30]. Spam recall depicts the rate of spam correctly classified [45]. Spam precision will also depict the rate of correctly classified spam, but will take into account the number of non-spam emails mistakenly classified as spam [45]. Accuracy reflects

the whole performance of a spam filter system, both recall and precision [45]. False positives are the non-spam emails mistakenly classified as spam. False negatives are spam emails mistakenly classified as non-spam. These measures are discussed in Chapter 6, but have not been measured explicitly in the experiment. This is because the proposed spam filter solution is proposed as an application system to a bigger spam filter solution. The proposed system is a proof of concept system, concept being applying approximate search algorithm on an FPGA hardware.

In this thesis, the same algorithm is tested for both implementations, and initial tests implicated that they work exactly the same. They give the same result for each run, and classify spam and ham based on the keywords initially set as spam keywords. To give a thorough performance measurement of the algorithm, different algorithms should be tested against each other, but that is a task for future work. This experiment will focus on evaluating the efficiency and processing speed of the two implementations.

5.3 Experimental Results

To illustrate a spam-filter scenario, a few spam words were taken from Symantecs internet report from 2017 [1]. The words chosen were shipping, invest and invoice. In order to avoid a spam filter, but at the same time be intelligible to the recipient, we assumed that the edit distance k, is no bigger than 1 or 2. But to test processing time, an edit distance of $k = 3$ was also tested. The keywords were modified by adding insertion, deletion and substitution errors. A software-based spam filtering system using unconstrained approximate search using Levenshtein distance, developed in C++ , was tested alongside the FPGA-based spam filter to meet the objective of this experiment. The same spam corpus was run through both the software implementation and the hardware implementation. Both software and hardware solutions have the same design flow, to make the experimental results most reliable. In the software implementation, a run timer is added to time the processing time of the approximate search function. Such a function, the timer, is not possible to implement in the hardware version. It is also unnecessary, since all operations in an FPGA follows a clock frequency, and hence the time to execute can be computed precisely. The FPGA board has a clock frequency of 100 MHz compared to 2.5 GHz on the computer used for testing. Search patterns and text are the inputs to the software implementation and the running time is the output, along with where keywords are found in the text. On the hardware implementation running time is not measured, as the program time is dependent of the frequency of the clock, and will run the same amount of clock cycles as the test-bench. To account for the reliability, led-lights on the board were lit when a pattern with k-faults was found.

1. Execute the two implementations, with only one pattern appearing in the search text with $k = 1,2,3$ (edit distance)
2. Iterate the process several times
3. Take the running time of the software implementation and check the FPGA implementation for correctness
4. Double the size of the search text
5. Repeat steps 2 to 4 until the search pattern is five times the size of the first round
6. Repeat steps 1 to 5 with k from one to three

Algorithm 1: Performing the experiment

Figure 13 show the results from the experiments, without taking into account the two different clocks. The horizontal axis indicates the size of the search text, and the vertical axis indicates the running time. Even without taking into account the two different processing clocks of the computer the software is run on and the FPGA the hardware is run on, the processing time is drastically shorter for the hardware implementation.

Figure 14 shows the result from the experiment, showing results from implementation of different edit distance. The results show that the processing time of the hardware implementation stays the same, no matter how long the edit distance is. That means that the rows that are added for each new k are processed in parallel with the first row, and therefore will not affect processing time. However, the processing time of the software implementation increases slightly for every added edit distance k. This means that the difference in processing time for a larger k and a longer search text will be even more significant for long text searches.

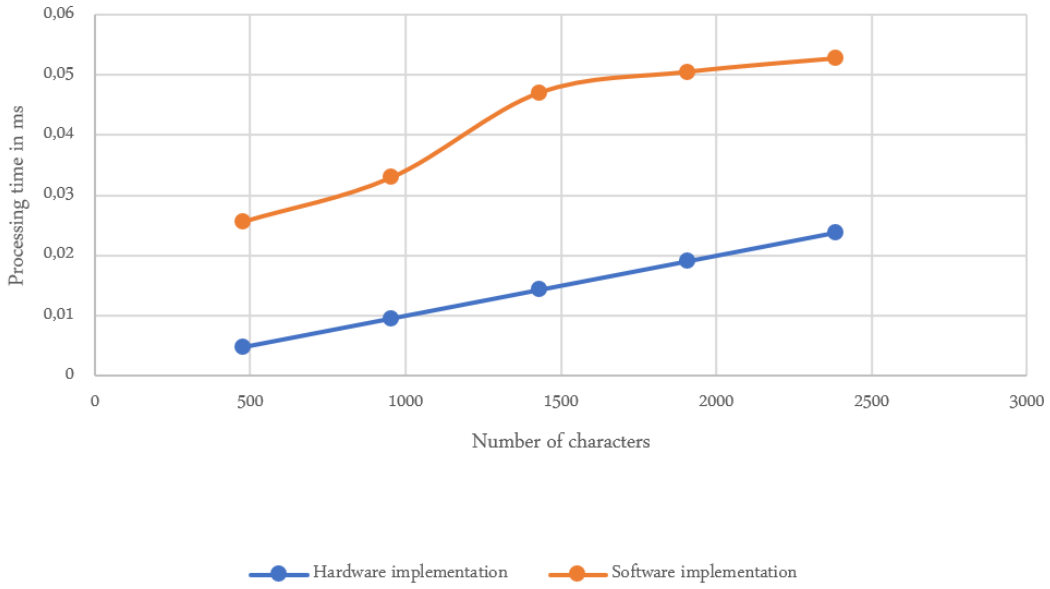

Figure 13: Results from hardware and software implementation of the algorithm with K = 1.

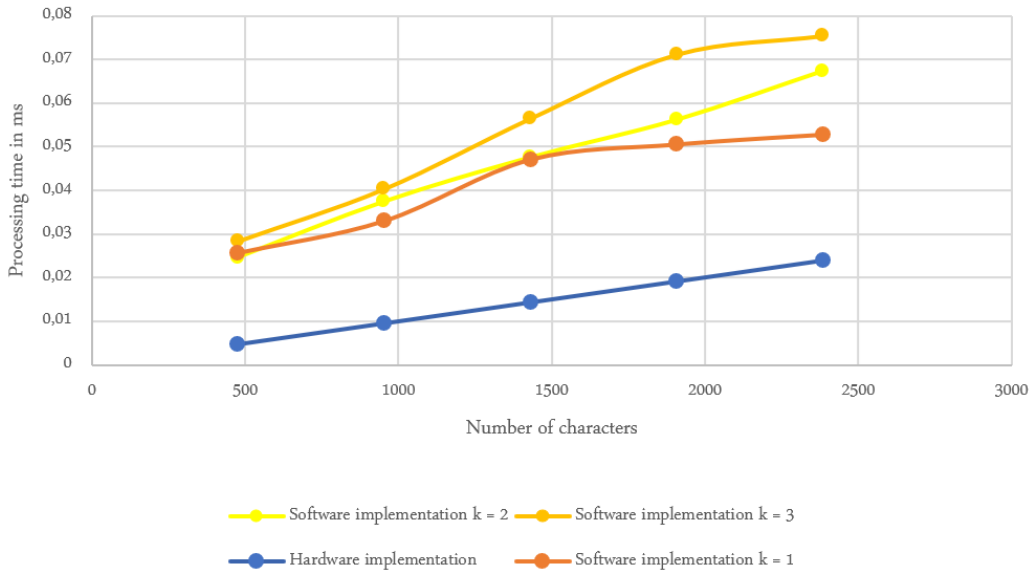

Figure 14: Results from hardware and software implementation of the algorithm with with K = 1, 2 and 3.

6 Discussion

This chapter discusses the findings and results from the previous chapters and relate them to the research problem and questions. The first section discusses the implementation of the algorithm onto the FPGA board. The following section concerns the efficiency of running each of the two programs. The third section discloses the accuracy of the two implementations. Following is a discussion of the results based on the chosen algorithm. Lastly this chapter discusses the relevance of the results of this research, a discussion of the objectives were met and a discussion of sources of error.

6.1 Implementation of Unconstrained Approximate Search using Levenshtein Distance onto FPGA Board

The first research question concerned the implementation of an unconstrained approximate search algorithm using Levenshtein distance on an FPGA board. The results from Chapter 4 and Chapter 5 show that the algorithm, utilizing bit parallelism, can easily be implemented on an FPGA board. Hardware programming design and language differs from any other software language, and learning to program with a hardware language is not like learning a new software language like Java, when one is familiar with C. The time and cost of using a hardware program designer, must be taken in consideration when hardware or software platform is the question. Hardware design and implementation will use more development time, but will cost less in terms of actual hardware. Aside from the learning perspective, and the general lack of hardware programmers, implementing the algorithm onto an FPGA board, such as the one used for this thesis, the Nexys4 DDR, can be achieved. The *field programmable* part of the FPGA makes is easy for a designer to try and test, but can also be the pitfall, when the programmer is satisfied once the desired result is achieved.

The programming language used to program the FPGA board in this thesis, Verilog HDL, is a low level hardware programming language, suitable for beginner levels. The design suite used to design, implement and program the board, is intuitive and aids the researcher in following the hardware design flow. Using these tools to develop solutions to utilize hardware feature alongside a traditional software anti-spam program can be an effective way to accelerate the total anti-spam program without increasing cost and recources.

6.2 Accuracy and Efficiency of Spam Detection in Hardware

This section answers the second research question, addressing the accuracy and efficiency of a hardware based spam filter applying approximate search.

6.2.1 Efficiency

The proposed FPGA-based spam filtering program is able to provide a satisfactory result where it is more than 25 times faster than software-based spam filtering program. With the availability of

enough hardware resources, it is possible to perform every process task in parallel. These processes are the run of each pattern, and each row to each pattern. Running all these processes in parallel, given the same input data, the true parallel solution is no problem. The pipe-lining principle must be applied to each new symbol of the text as the R_i is calculated based on R_{i-1} in the approximate search bit parallel algorithm. Performing this solution will process the text in $\mathcal{O}(m)$ when m is the length of the text. The increased time complexity for the software solution when k increases, is shown in Chapter 5 Figure 14. Compared to the hardware implementation, which does all the R-level processes in parallel, the difference is significant. In an e-mail spam situation, the edit distance k would seldom exceed more than 1 - 3, and therefore this would not have a big impact on the total time complexity. Exceeding the edit distance with more than $k > 3$ will in most cases make the word unintelligible, and hence the obfuscation would loose its value. In another scenario, using larger patterns, this could be a decisive mean. We also notice that the time complexity in the software solution increases more than the hardware solution as the pattern size increases. Figure 13 shows that the software solution uses over 0.025 milliseconds more than the hardware solution for only a little less than 2500 symbols. These results are raw results, without taking into consideration the different clocks.

Considering that the hardware clock is only 100MHz compared to the software clock of 2.5GHz, these processing times are record breaking. Generalizing from these findings, large text searches will use significantly less time and resources with the hardware solution.

To ensure the run time of the software program, the data set was run 11 times for each data set size. The last being five times the size of the first. This was also done on the FPGA board, but gave the exactly the same result on each run. The FPGA board processes according to the time cycle simulated in the behavioural simulation run with test bench, with little to no delay. Due to this, the result from a run of the FPGA board is always equal to the number of symbols in the data corpus times the clock used on the FPGA board. In this thesis, the processing time for one symbol on the hardware implementation is 10ns regardless of what k is. The test corpus consisted of, for the first run, 477 symbols with three distorted words originating from a spam word “shipping”. The distortion was never more than an edit distance of $k = 3$, but this small number of changes in edit distance was visible in the results as seen in Figure 14.

Figure 15 shows the execution time of the software implementation and the calculated execution time of the hardware implementation given the same clock as the computer the software was executed on. The results from Chapter 5 showed, before the mathematical tuning, that the FPGA hardware implementation was far more effective than the software implementation. Taking into account that the computer used in the experiment has a clock frequency of 2.5 GHz while the Nexys4 FPGA board has a clock frequency of 100 MHz the results change drastically. Recalculating the results from Chapter 5 figure 13 simulating a scenario with both software and hardware implementation are using a same speed clock of 2.5GHz, shows in Figure 15 that the FPGA board implementation is by far faster than the computer in means of both time and space complexity. The clock of the computer can do 25 times more processes per second than the FPGA board can. To convert the results from Figure 13 to Figure 15 the results of the hardware implementation is

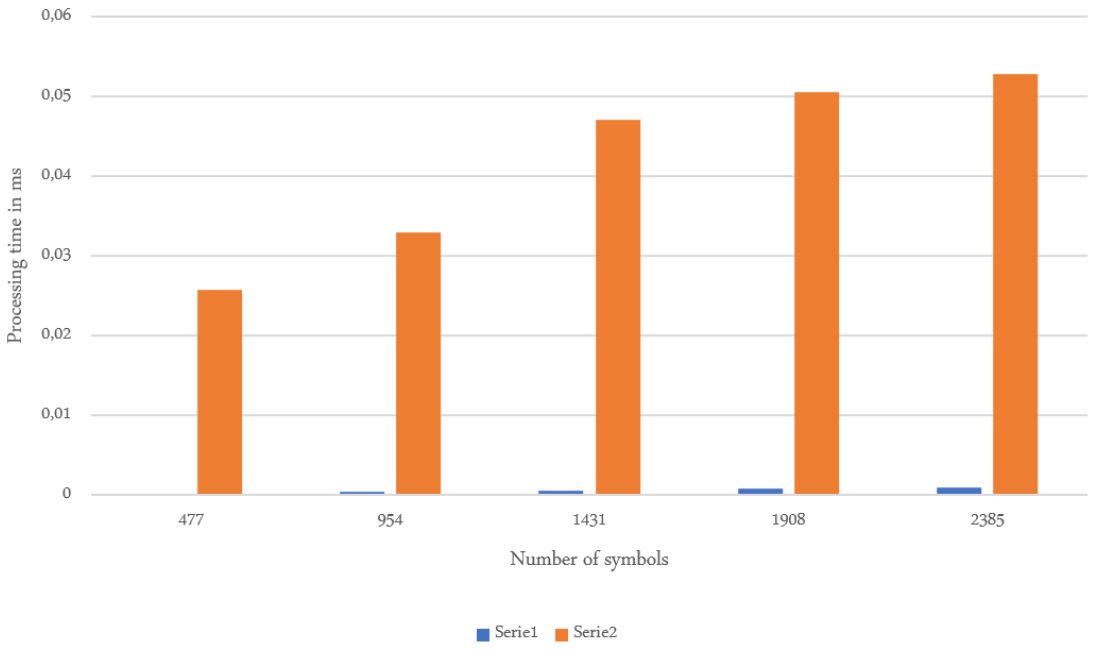

Figure 15: Results from comparison of running speed of the two implementations, in this figure the two different clocks is converted mathematically to depict the real difference of the two runs

divided by 25 to simulate an internal clock of 2.5GHz. Using the results found in the Appendix, Equation 6.1 shows that the processing time of the hardware solution is more than 55 times faster than that of the software implementation. This result is from a test of only a little less than 2500 symbols, and we can only assume the longer text search, a larger time gain will follow.

$$\frac{\text{Processing time (Software)}}{\text{Processing time (Hardware)}} = \frac{0.05276}{0.000954} = 55.3 \tag{6.1}$$

The results from the experiments and discussion are generalized from a very small data corpus, and the results from a bigger corpus would probably portray an even greater gain. As mentioned in Chapter 2 it is difficult to generalize from the results from the run of the software solution since many other processes are running on the computer while testing the program. The core of a GPP will always have many other processes running, and that is the core idea of a GPP. Removing all other processes for a test environment would not simulate a real world scenario either and hence not provide a basis for generalizing from these results. On this basis, the simulation in Figure 15 is a valid representation of the time gain utilizing a specific FPGA implementation rather than a classical software implementation.

6.2.2 Accuracy

The proposed spam filtering program was able to detect all exact and approximate keywords perfectly. This is proved by the result that all spam keywords are successfully classified as spam. The Figure 10 from Chapter 4 shows that the program calculates the algorithm perfectly for every new symbol. The software implementation also proved to perform flawlessly for the same reason. The two implementations can not be set up against each other in terms of accuracy.

Nevertheless, a discussion on the accuracy of the algorithm chosen in this thesis is in its place. The unconstrained approximate search algorithm will always accept all words whose edit distance is less than or equal to k. That implies that if a k larger than three is used, the results, especially for short words, will include false positives. Knowing the two programs behave alike, they will both count double in those cases where the exact word is present. The results in Figure 10 and Figure 16 prove this when k is 3. However, this is the designed behaviour of both implementations and to avoid this, another algorithm or different type of the same algorithm must be considered.

6.3 Hardware vs. Software

The third research question concerned whether or not hardware solutions should be chosen rather than software solutions. The following section is a discussion of the findings in Chapter 3.

In comparison to an FPGA, a conventional ASIC is costly in time and has high-risk factors. The ability to *field program* the FPGA board, removes the risk of running into design defects and costly change in functions. The FPGA board offers a small-size and low-power consumption compared to software running on workstations. The benefits of using hardware to accelerate machine learning algorithms have been recognized in the world [4] [46]. And various algorithms have been implemented in hardware to work with software programs [4].

Hardware design, development and programming is not as straightforward as software engi-

neering. But for specific tasks, like the approximate spam filtering problem that can leverage the parallel processing quality of hardware, the overall effect will repay for the resources used in development. The FPGA board allows the designer to modify and update the whole design at any time. This feature means that new functionality can be added to the board when this is needed.

Hardware processes in a parallel manner as opposed to the sequential software process. The bit parallel approximate search algorithm utilizing Levenshtein distance exploits the parallel manner of a hardware process perfectly, and hence accelerates the traditional processing time of software implementations. Hardware processing is suitable for intensive data processing, as the processes are specific and specifically programmed in advance. In contrast, general purpose processors are not specifically task programmed, and will struggle when large data sets are to be processed.

6.4 Choice of Algorithm

The choice of using unconstrained approximate pattern matching algorithm in the work of this thesis, instead of the constrained one was a choice affected by the time available. The results from running the program, both on software and hardware, illuminated the problem of unconstrained search. As described in Chapter 3 [47] , the introduction of constraints to the approximate search algorithm will limit the false positives generated by the unconstrained algorithm. Instead of only constraining the maximum edit distance k, constraints on the total number of insertions and deletions considered together, and constraints on lengths of runs of deletions and insertions should improve the spam filtering algorithm, by making it more accurate.

Dynamic programming uses arithmetic operations, and bit parallelism uses bit-wise operations. Both of these techniques are used to simulate NFAs, and solve the problem of approximate string matching. Using dynamic programming the computation time is $\mathcal{O}(mn)$ while bit parallelism uses $\mathcal{O}(m)$. Knowing that hardware is perfect for bit parallel processes, the bit parallel technique to solve the approximate search problem was perfect in the context of this thesis.

6.5 Relevance of the Results From This Thesis Given the Problem Description

The results from this thesis have showed that a spam filter solution can be accelerated utilizing the parallel feature of hardware processing, giving record breaking retrieval times. In large high speed networks, where it is crucial that a spam email is classified as spam before it enters the inbox of the end user, such as the case described in the beginning of this thesis, a fast spam filter is important. Large text searches are traditionally time consuming, and large text searches are even more time consuming. The more errors allowed, the more time needed to process the text search. This can be read from the results from Figure 14.

Obfuscating keywords is a known technique by spammers to bypass an anti-spam program, and hence deliver its potentially damaging content to end users. The approximate search algorithm using Levenshtein distance has been proved in this thesis as an appropriate algorithm to efficiently find the obfuscated spam keywords used by spammers.

6.6 Achievement of Objective

The objective of this thesis was to clarify and describe the possible efficiency that can be found when using hardware to accelerate the spam filter program. The task of the author was to code and implement an FPGA based spam filter to detect spam by utilizing the unconstrained approximate string pattern matching using the bit parallel Levenshtein distance algorithm. The hardware solution was used as a mean to test the two variables, hardware and software, opposed each other in an experiment conducted in this research. As this research can be used to determine if hardware solutions should be chosen over software solutions in the development and launch of spam filters the objectives stated in the introduction Chapter 1 are achieved.

6.7 Sources of Error and Validity

The processing time of the software solution varies from each run of the program. That is why the program was run 11 times during the experiment for each text search size. The processing time of the hardware solution does not vary from each run, as it follows the internal clock of the FPGA board. This results in a linear line in Figure 13 and 14. And a more curvy tendency for the results from the software program. It can look like it is starting to be more horizontal as the search text size increases. This could imply that if a larger set of symbols were tested, the curves might cross. That would mean that the software running times are faster than the hardware running times the larger the text search is.

When this was discovered the running times of 5000 symbols was tested on the software board, following the same procedure as during the experiment. This run of the program implied that the theory stated above did not prove to be true. The processing time for this run was even more significant than that of the smallest search text. This test with 5000 symbols was not retested on the FPGA board due to time limitations, and is therefore not included in the results.

7 Conclusion

The consequence of spam for an organization or end user is complex, because spam can be the cause of a variety of issues. Spam on its own is annoying and time consuming to sort out, both manually by the end user, but also for a spam filter. A spam email can contain vicious links and malware attachments. The consequences of spam are thus loss of time, loss of data, direct economical consequences, working hours as well as unknown consequences.

Spam filter solutions today are mostly software based solutions. Long traditional text searches are time consuming for most processors. Large high speed networks acquire a fast email filter solution to process data with a reasonable time and space complexity. This thesis has showed how hardware based solutions can be used speed up the filtering process, and be more time and cost effective for email providers and end users. This answers the first research question.

In this thesis, a spam filter program was written in Verilog hardware description language and implemented onto an FPGA hardware board to test the accuracy and efficiency of a hardware based spam filter. As spammers continue to find intelligent ways to bypass a normal rule-based spam filter, approximate pattern matching algorithms have been proposed as a solution to sift out also those spam emails containing obfuscated spam keywords. The successful development of the Verilog program run on the Nexys 4 DDR, shows that approximate search algorithms can be implemented in FPGA hardware using bit parallel techniques.

The second research question concerned the efficiency and accuracy of spam detection in hardware compared to classical software implementations. The proposed hardware solution implemented on the Nexys 4 DDR, which is a student type FPGA, proved to be approximately 55 times more efficient than the normal personal computer that is four times the price of the FPGA board. Generalizing from the findings from this thesis, the accuracy and efficiency of spam detection in hardware is just as accurate as the software implementation, and many times more efficient in terms of time complexity.

The third research question is why should hardware solutions be chosen rather than software solutions. Hardware solutions are more costly in time and resources to develop, but for high speed network traffic, the gain won from accelerating the processing speed is in most scenarios an easy trade. To conclude, using hardware to accelerate the processing speed of a spam filter using bit parallelism approximate search, is a recommended solution to improve email anti-spam systems.

7.1 Future Work

To continue the work of this thesis the proposed solution should be included in an integrated spam solution utilizing more anti-spam techniques, for example the Naive Bayes theorem for spam keywords selection. To measure the spam filter with spam recall, spam precision and spam accuracy the solution should be integrated within a real program using real time data streams. The solution provided in this thesis is a proof of concept, proving the concept of accelerating the processing speed of a spam filter by utilizing the bit parallel manner of hardware. Putting this concept into a real time scenario using Ethernet to run real time email bodies through the filter will further prove the efficiency of using a hardware based spam filter.

To further clarify the results from this thesis, an industrial hardware FPGA board with a more comprehensive internal clock should be used for experiments. To avoid false positives the approximate search algorithm with constraints should be tested alongside the unconstrained approximate search that is used in this thesis.

To illuminate the source of error discussed in Chapter 6, the experiment with larger data samples should be further tested to show the true difference of large text searches. The test carried out in this thesis cannot determine if the increase depicted in the results will hold for longer data sets.

Bibliography

[1] Nahorney, B. 2017. Internet security threat report: Email threats 2017. Official webpage of symantec. Accessed: 2018-04-05.

[2] Templeton, B. 2017. Reaction to the dec spam of 1978. URL: https://www.templetons.com/brad/spamreact.html.

[3] Monmasson, E. & Cirstea, M. N. 2007. Fpga design methodology for industrial control systems 2014 – a review. *IEEE Transactions on Industrial Electronics*, 54(4), 1824–1842.

[4] Layer, C. 2007. A coprocessor for fast searching in large databases: Associative computing engine.

[5] Webpage, X. 2000. Fpga design flow overview. https://www.xilinx.com/products/boards-and-kits/1-6olhwl.html. Accessed: 2018-04-05.

[6] Xilinx. 2018. Xilinx website. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug904-vivado-implementation.pdf. Accessed: 2018-03-04.

[7] Russel, C. 1017. What is an fpga, what is an asic. https://www.nandland.com/articles/what-is-an-fpga-what-is-an-asic.html. Accessed: 2018-02-29.

[8] NSM. 2016. Mørketallsundersøkelsen 2016, informasjonssikkerhet, personvern og datakriminalitet.

[9] Myndighet, N. S. 2017. Grunnleggende tiltak for sikring av e-post. *NSM Veiledninger*.

[10] Fjørtoft, T. O. 2017. Ni av ti surfer på nettet hver dag. *Teknologi og innovasjon*.

[11] Lueg, C. 2003. Spam and anti-spam measures. *A Look at Potential Impacts*.

[12] E.M.Tamil, M.Y.I.Idris, C. M. & M.Z.Jali. 2007. Needleman wunsch implementation for spam/uce inline filter. *System-on-chip Research Group*.

[13] NTNU. 2017. Ntnu website. http://www.ntnu.no. Accessed: 2018-01-03.

[14] Baeza-Yates, R. & Ribeiro-Neto, B. 1999. *Modern Information Retrieval*. ACM Press / Addison Wesley.

[15] Knuth, D. E. 1997. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley, 2nd edition.

[16] Brisaboa, N., Cerdeira, A., & Navarro, G. 2017. *Managing Compressed Structured Text*. Springer, To appear.

[17] Faro, S. & Lecroq, T. 2012. Twenty years of bit-parallelism in string matching. *University of Catania*.

[18] Navarro, G. & Raffinot, M. Fast and flexible string matching by combining bit-parallelism and suffix automata.

[19] Navarro, G. A guided tour to approximate string matching.

[20] Araújo, M., Navarro, G., & Ziviani, N. 1997. Large text searching allowing errors. In *Proc. 4th South American Workshop on String Processing (WSP)*, 2–20. Carleton University Press.

[21] Navarro, G. & Raffinot, M. 2002. *Flexible Pattern Matching in Strings*. University of Cambridge.

[22] Chitrakar, A. S. & Petrovic, S. 2016. Constrained row-based bit-parallel search in intrusion detection. *Norsk Informasjonssikkerhetskonferanse (NISK)*.

[23] Tripp, G. J. 2006. A parallel "string matching engine" for use in high speed network intrusion detection systems. *Comput Virol 2: 21*, 2–21.

[24] Creswell, J. W. 2013. *Research Design*. SAGE Publications Inc.

[25] Sanders, M. & Thornhill, A. 2012. *Research Methods for Business Students*. Pearson Education Limited, 6th edition.

[26] Svartdal, F. 2017. Eksperiment. i store norske leksikon. https://snl.no/eksperiment. Accessed: 2018-05-15.

[27] Blakstad, O. 2008. Experimental research. https://explorable.com/experimental-research. Accessed: 2018-05-02.

[28] Holub, J. 2000. Simulation of nondeterministic finite automata in pattern matching. Submittet at the faculty of Electrical Engineering, Czech Technical University. Accessed: 2018-02-01.

[29] Digilent. 2017. Nexys4 ddr reference manual. https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual. Accessed: 2018-03-03.

[30] Schryen, G. 2007. *Anti-Spam Measures*. Springer.

[31] Weldeghebriel, L. 2011. 100 kroners bot for hver spam. Accessed: 2018-05-05.

[32] Lovdata. 2009. Lov om kontroll med markedsføring og avtalevilkår mv. (markedsføringsloven). https://lovdata.no/dokument/NL/lov/2009-01-09-2. Accessed: 2018-05-12.

[33] Union, E. 2018. The eu general data protection regulation (gdpr). https://www.eugdpr.org/. Accessed: 2018-05-12.

[34] Dvergsda, H. & Rossen, E. 2018. Spam. i store norske leksikon. https://snl.no/spam. Accessed: 2018-05-08.

[35] Klensin, J. 2008. Simple mail transfer protocol. https://tools.ietf.org/html/rfc5321. Accessed: 2018-04-27.

[36] Giacomo, J. 2016. Understanding email anti spam laws in us, canada, eu. https://www.practicalecommerce.com/Understanding-Email-Anti-spam-Laws-in-U-S-Canada-E-U. Accessed: 2018-05-12.

[37] Brown, S. D. 2001. *Field-programmable gate arrays*. Kluwer.

[38] Trimberger, S. M. 1999. *Field-programmable gate array technology*. Kluwer Acad.

[39] Martindal, J. 2018. What is a cpu. https://www.digitaltrends.com/computing/what-is-a-cpu/. Accessed: 2018-05-03.

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

[41] Yao, B., Li, F., Hadjieleftheriou, M., & Hou, K. 2010. Approximate string search in spatial databases. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 545–556.

[42] Hall, P. A. V. & Dowling, G. R. 1980. Approximate string matching. *Comput. Surv*, 12(4):381–402.

[43] Ukkonen, E. 1985. Algorithms for approximate string matching. information and control. *Information and Control*, 64(1–3):100–118.

[44] Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8), 845–848.

[45] Tan, Y. 2016. *Anti-Spam Techniques Based on Artificial Immune System*. CRC Press.

[46] Ruizhe Zhao, Wayne Luk, X. N. S. H. W. 2017. Hardware acceleration for machine learning. *Computer Society Annual Symposium on VLSI*.

[47] Petrovic, S. A spam filtering scenario using constrained bit-parallel approximate search. *NISlab, Faculty of Information Technology and Electrical Engineering*.

A Testing Data

This is the raw result of running the algorithm presented in chapter 5.

A.1 Test Session

Objective

The Objective of this testing session was to test the running time of the implementation of unconstrained approximate search using Levenshtein distance Σ version on the FPGA board and the software platform. The source code for the two implementations are appended to the digital version of this thesis. The source code for the software implementation is provided by Professor Slobodan Petrović. The program is written in C++. The source code for the hardware implementation is provided by the author of this thesis.

Test environment

The test was carried out on a Hewlet Packard x360 intel core i7 and a Nexys4 DDR FPGA board.

A.1.1 Raw Data

Table 6 shows the running time of one input pattern and a search text of 477 words from the software version. Table 7, 8, 9, 10, depicts the running time of input search text with respectively 954,1431, 1908, 2385 symbols.

A.1.2 Screenshots

R1	R2	R3
0,0217284	0,0240988	0,0335804
0,0205432	0,0237037	0,0248889
0,0280494	0,0284445	0,025284
0,0248884	0,0217284	0,0308148
0,0379259	0,0181728	0,0331852
0,0248889	0,0359506	0,034198
0,025679	0,0237037	0,032
0,028494	0,0229136	0,0237037
0,0225185	0,0213333	0,0248889
0,0217284	0,0233087	0,0248889
0,0260741	0,0276543	0,0244938

Table 6: Results from running the program with 477 symbols.

R1	R2	R3
0,031605	0,0671605	0,0237037
0,0335803	0,0339753	0,0402963
0,0288395	0,0387161	0,0410864
0,0276543	0,0367408	0,0375309
0,0292346	0,0367509	0,0367408
0,0462222	0,021333333	0,0434568
0,0387161	0,0363457	0,0355556
0,0347654	0,038321	0,0426667
0,0264691	0,0375309	0,0422716
0,0312099	0,031605	0,0584692
0,0347654	0,0335803	0,0418766

Table 7: Results from running the program with 954 symbols.

R1	R2	R3
0,116148	0,0454321	0,0553087
0,0462222	0,0466173	0,070321
0,0426667	0,0481975	0,0592593
0,0450371	0,0466173	0,0699269
0,0422716	0,0367408	0,070321
0,0288395	0,0722963	0,0470124
0,044642	0,0454185	0,0537284
0,0347654	0,0410864	0,0545185
0,0327901	0,0371358	0,0359506
0,0454321	0,051358	0,0584692
0,0387161	0,0525432	0,0466173

Table 8: Results from running the program with 1431 symbols.

R1	R2	R3
0,0478025	0,0493827	0,0722963
0,0541235	0,051358	0,0663704
0,0505679	0,0707161	0,0533334
0,0462222	0,0537284	0,107457
0,069926	0,050963	0,0963951
0,0525432	0,0707161	0,0580741
0,051232	0,051358	0,0588642
0,0450371	0,0462222	0,0841482
0,0493827	0,0580741	0,0560988
0,0399013	0,050963	0,0659753
0,0497728	0,0647901	0,064

Table 9: Results from running the program with 1908 symbols.

R1	R2	R3
0,0547778	0,0525432	0,077432
0,0674074	0,0774321	0,082963
0,0481975	0,0647901	0,0612346
0,044642	0,051358	0,0628148
0,0529383	0,0975803	0,0679506
0,051642	0,0553087	0,0825679
0,0458272	0,0596543	0,0628148
0,0462222	0,0920494	0,0809877
0,0442269	0,0568889	0,0896679
0,0624198	0,0809877	0,0908642
0,0620247	0,0529383	0,0707161

Table 10: Results from running the program with 2385 symbols.

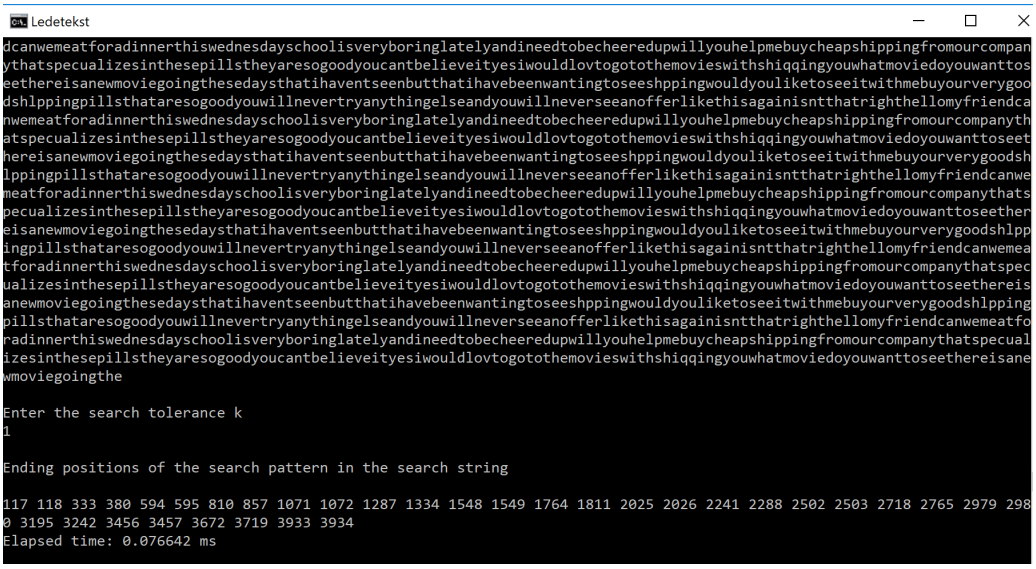

Figure 16: screenshot from run of software program