UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

AUTOMATION TECHNOLOGY

Mathias Björk

DEVELOPMENT AND TESTING OF IEC 61850 NETWORK INTERFERENCE EQUIPMENT – A CASE STUDY

Master's thesis in Technology for the degree of Master of Science in Technology submitted for inspection, Vaasa, March 26, 2014.

Supervisor	Professor Jarmo Alander
Instructor	Professor Jarmo Alander, D.Sc. (Tech.) Petri Välisuo

FOREWORD

First I want to acknowledge my supervisor and instructors for all their effort in helping me with this project. I would like to thank Mika Ruohonen for his expertise in the IEC 61850 standard along with his contribution to the TehoFPGA project. Finally I would like to thank Olli Rauhala and Staffan Järn for their co–operation and great workmanship in this project.

Vaasa March 26, 2014

Mathias Björk

TABLE OF CONTENTS **page**

SYMBOLS AND ABBREVIATIONS

Greek symbols

λ	Offspring size
μ	Population size

Other symbols

L_{ch}	Chromosome length
loc_i	Locus index
cr_i	Crossover index
R_{cr}	Crossover rate
N_{cr}	Number of crossover points
P_{cr}	Crossover probability parameter
P_m	Mutation probability

Abbreviations

ASIC	Application–specific integrated circuit
CPLD	Complex programmable logic device
CRC	Cyclic redundancy check
DoS	Denial–of–Service
DSP	Digital signal processor
DUT	Device under test
EC	Evolutionary computing
EA	Evolutionary algorithm
EEPROM	Electronically erasable programmable read–only memory
ES	Evolutionary strategies
EP	Evolutionary programming
FCS	Frame check sequence
FPS	Fitness proportional selection
FPGA	Field programmable gate array
FSM	Finite state–machine
GA	Genetic algorithm
GAL	Generic array logic

GMEC	Gosling message exchange time
GOOSE	Generic object–oriented substation event
GP	Genetic programming
HDL	Hardware description language
IC	Integrated circuit
ICBF	Idle cycles between frames
IEC	International Electrotechnical Commission
IED	Intelligent electronic device
IEEE	The institute of Electrical and Electronic Engineers
IO	Input–output
IP	Internet protocol
ISO	International organisation for standards
JTAG	Joint test action group
LAN	Local area network
LC	Logic cell
LE	Logic element
MAC	Media access control
NIC	Network interface card
OSI	Open systems interconnection
PAL	Programmable array logic
PCB	Printed circuit board
PID	Proportional–integral–derivative
PLD	Programmable logic device
PLA	Programmable logic array
RPI	Raspberry PI
SAS	Substation automation system
SFD	Start frame delimiter
SPLD	Simple programmable logic device
SV	Sampled values
TCP	Transmission control protocol
TTL	Time to live
UART	Universal asynchronous receiver/transmitter

USB	Universal serial bus
VHDL	Very high–speed integrated circuit hardware description language
VLAN	Virtual local area network
WAN	Wide area network

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Mathias Björk
Diploomityön nimi: Tapaustutkimus IEC 61850 verkkohäiriölaitteiston kehityksestä ja testauksesta
Valvojan nimi: Professor Jarmo Alander
Ohjaajan nimi: Professor Jarmo Alander, D.Sc. (Tech.) Petri Välisuo
Tutkinto: Diplomi-insinööri
Koulutusohjelma: Sähkö– ja energiatekniikan koulutusohjelma
Suunta: Automaatiotekniikka
Opintojen aloitusvuosi: 2011
Diploomityön valmistumisvuosi: 2014 **Sivumäärä:** 89

TIIVISTELMÄ:

Älykkäissä sähkölaitteiden (IED, intelligent electronic device) lukumäärän kasvaessa sähköasemien automaatiojärjestelmissä (SAS, substation automation system) käytetään enenevässä määrin IEC 61850–standardin mukaista tiedonsiirtoa, jonka useimmiten käytetty tiedonsiirtoprotokolla on GOOSE (Generic Object–Oriented Substation Events). Tämä protokolla asettaa tiettyjä vaatimuksia IED–laitteille ja verkon arkkitehtuurille. Näitä vaatimuksia on seurattava tarkasti jotta voidaan taata turvallinen ja toimiva sähköaseman automaatiojärjestelmä.

Tämän tutkielman päätavoitteena oli sellaisen laitteiston kehittäminen, jolla voidaan testata GOOSE–protokollan avulla kommunikoivien IED–laitteiden käyttöä mahdollisimman huonoissa olosuhteissa. Testauslaitteisto pystyy välittämään ethernet–paketteja gigabitin sekunti nopeudella, jotta voidaan analysoida häiriön vaikutuksia testattavaan laitteeseen (DUT, device under test). Testauslaitteisto pystyy geneettistä algoritmia käyttäen myös etsimään paketin, joka on testattavalle laitteelle kaikkein haitallisin.

Kehitetty laitteisto pystyy tulosten mukaan löytämään heikkouksia testattavissa laitteissa, kun ethernet–paketteja välitetään suurilla nopeuksilla ja häiritsevien pakettien määränpääksi asetetaan jonkin toisen kuin testattavan laitteen osoite. Edellä mainittu onnistuu kuitenkin vain tietyn testattavan laitteen kanssa, ei kaikissa tapauksissa. Tässä tutkielmassa testattavan laitteen toiminta pystyttiin pysäyttämään kokonaan. Tulosten mukaan geneettinen algoritmi ei kyennyt löytämään mitään tiettyä haitallista pakettia, mikä tarkoittaa sitä, että pakettien rakenteella ei ole suurta merkitystä testattavan laitteen toiminnan pysäyttämisessä.

AVAINSANAT: Evoluutiolaskenta, IEC 61850, Ethernet, FPGA

UNIVERSITY OF VAASA
Faculty of Technology
Author: Mathias Björk
Topic of the Thesis: Development and testing of IEC 61850 network interference equipment – a case study
Supervisor: Professor Jarmo Alander
Instructor: Professor Jarmo Alander, D.Sc. (Tech.) Petri Välisuo
Degree: Master of Science in Technology
Degree Programme: Degree Programme in Electrical Engineering
Major of Subject: Automation Technology
Year of Entering the University: 2011
Year of Completing the Thesis: 2014 **Pages:** 89

ABSTRACT:

As the number of intelligent electronic devices (IEDs) is increasing in substation automation systems (SAS), the IEDs have often been extended with network communication conforming with the IEC 61850 standard , where the Generic Object–Oriented Substation Events (GOOSE) communication protocol is mostly used. This protocol sets certain demands on the IEDs and the network architecture, which must be strictly followed to ensure a safe and functional SAS.

The aim of this study is to develop equipment for testing IEDs communicating with the GOOSE protocol in the worst conditions possible. The testing equipment is able to transmit Ethernet packets at the rate of one gigabit per second, in order to analyze the impact of the interference on a device under test (DUT). This testing equipment is also able search for the single most harmful packet for a DUT by using a genetic algorithm.

This study shows that the developed equipment is able to find flaws in DUT's by transmitting Ethernet packets at high speeds, when setting the destination address of the interfering packets to any other address than the physical address of the DUT. However, this only works for a specific DUT, and not in every case. This particular case managed to disable the DUT's functionality completely. This study also shows that the genetic algorithms did not manage to find any specific harmful packet. This shows that the packet structure does not seem to play any role in disabling the functionality of the DUT.

KEYWORDS: Evolutionary computation, IEC 61850, Ethernet, FPGA

VASA UNIVERSITET
Tekniska fakulteten
Författare: Mathias Björk
Titel: Utveckling och testning av störningsutrustning för IEC 61850 nätverk – en fallstudie
Övervakare: Professor Jarmo Alander
Handledare: Professor Jarmo Alander, D.Sc. (Tech.) Petri Välisuo
Examen: Diplomingenjör
Utbildningsprogram: Utbildningsprogrammet för elektro– och energiteknik
Inriktning: Automationsteknik
Intagningsår: 2011
Diplomarbetet färdigställt: 2014 **Sidantal:** 89

REFERAT:

Eftersom antalet intelligenta elektroniska anordningar (IED, intelligent electronic device) ökar inom flera ställverks automationssystem (SAS, substation automations system), så har IED–enheter ofta utökats med kommunikationsmöjligheter som följer IEC 61850–standarden. Av dessa är GOOSE (Generic Object–Oriented Substation Event)–protokollet det mest använda. Detta protokoll ställer hårda krav på IED–enheterna vad gäller kommunikation och nätverksarkitektur. Dessa krav måst följas strikt för att garantera ett säkert och fungerande ställverk.

Syftet med min undersökning är att utveckla testutrustning för IED–enheter som kommunicerar med GOOSE–protokollet inom krävande miljöer. Testutrustningen möjliggjör överföring av Ethernet–paket med en hastighet upp till en gigabit per sekund, vilket kan användas för att analysera störningsinverkan på en DUT–enhet (DUT, device under test). Med hjälp av genetiska algoritmer möjliggjör testutrustningen också sökning av enskilda paket som kan vara skadliga för en DUT–enhet.

Min undersökning visar att testutrustningen som jag utvecklat är kapabel till att hitta brister i DUT–enhetens programvara genom att sända störande paket med höga hastigheter till DUT–enheten. Detta är möjligt när destinationsaddressen hos de störande paketen inte är den samma som DUT–enhetens fysiska adress. Detta gäller dock endast för den specifika DUT–enheten som jag analyserade. Testutrustningen som jag utvecklade lyckades också helt inaktivera DUT–enhetens funktionalitet. Undersökningen visar också att störningspaketens struktur inte spelar någon roll när det gäller inaktivering DUT–enheten, vilket testades med genetiska algoritmer.

NYCKELORD: Evolutionär beräkning, IEC 61850, Ethernet, FPGA

1. INTRODUCTION

As the development of demanding electronics systems grows constantly, the expertise in the area must follow the same pattern to keep up with the increasing demand. Therefore the University of Vaasa has initiated a research project named TehoFPGA (including sub projects named TehoFPGA-I and TehoFPGA-II) to increase the expertise in the area of embedded electronics, where the main focus is on Field-Programmable Gate Arrays (FPGAs). The aim of TehoFPGA is to increase and share the knowledge of FPGAs amongst all participants, which are both universities and companies in Vaasa region. A FPGA-laboratory will be built in Technobotnia, to enable the attendants to participate in the research of the FPGA-technology hence increasing the expertise in this particular field of study. Initially several studies will be performed where the teaching, programming and testing of FPGAs will be analysed to set up a state-of-the-art FPGA-laboratory. (Alander 2012)

This Master's thesis is focusing mainly on the programming and testing of FGPAs, where a genetic algorithm will be implemented and used for network testing in substation automation systems (SAS), to detect flaws and bugs in protection relays communicating with the Generic Object-Oriented Substation Event (GOOSE) protocol derived from the IEC 61850 standard. More specifically this thesis focuses on the impact of interfering network traffic in a SAS, where the developed software is used to interfere with a functional device under test (DUT) in order to break its Ethernet communication. Primarily the software is used to analyse how the transmission rate of the interfering network traffic affects the DUT, and eventually the genetic algorithms are used to search for how the packet structure of the interfering network traffic is affecting the DUT's functionality.

1.1. Related work

Although there seems to be a lack of previous research concerning the specific research subject of this thesis, similar subjects have been researched earlier. Li (2004) proposed

a genetic algorithm for detecting network intrusion to classify rules for denying certain intrusion patterns, hence blocking incoming connections that are classified as threats. Gong, Zulkernine & Abolmaesumi (2005) managed to implement this behaviour, resulting in Java software capable of identifying network intrusions with a genetic algorithm. These two researches can be analysed as complements of this thesis research, since this thesis presumes that the testing device is physically located inside the substation network with IEC 61850 compliant devices.

Kuffel, Ouellette & Forsyth (2010) on the other hand, have studied the impact of abnormal IEC 61850 GOOSE and Sampled Values (SV) protocol data on a DUT, which is very similar to this thesis research. Kuffel et. al. on the other hand did manage to gather important data from intelligent electronics devices (IEDs) which is not possible in a broad range in this thesis, also they did not use genetic algorithm in their research. Furthermore Hor, Crossley & Millar (2007) managed to create a hybrid of a rough set theory and a genetic algorithm (hybrid RS–GA), to obtain additional knowledge from operational data from IEDs. The above mentioned works does relate to this thesis partly, but in this thesis, more emphasis is put on disabling a DUT's functionality using genetic algorithms.

1.2. Outline of the thesis

The introductory theory which is used as a baseline for this thesis is presented in Chapters 2, 3 and 4. Chapter 2 presents the basics of the Ethernet architecture, where the Internet protocol suite is explained along with the Open–System Interconnection (OSI) reference model (to give the reader an understanding of general networking concepts). Chapter 3 on the other hand studies evolutionary computing, and more specifically the structure and principles of genetic algorithms, which will be used in the developed EthGA software presented in Chapter 5. Chapter 4 presents embedded systems and more specifically the Field–Programmable Gate Arrays (FPGAs), on which the EtherTester software have been implemented that serves as a foundation for the EthGA software.

The practical work in this thesis is presented in chapters 5 and 6, where in Chapter 5 the developed EthGA software is presented along with its simulation on a computer and its implementation on a Nios II microprocessor on an Altera Developement and Education board 4 (DE4). The EthGA software is eventually tested and validated in Chapter 6 in a simulated environment where the EtherTester and EthGA is trying to interfere with a DUT, in order to break its Ethernet communication. The results are finally presented in Chapter 7 and discussed in Chapter 8.

2. ETHERNET ARCHITECTURE

This chapter will give the reader a brief introduction to the Ethernet network and its different standards and protocols that have evolved over time. The reader will also be given a short history review of the Ethernet, and later on in Section 2.1 the more important building blocks of the Ethernet are presented, including the *Open Systems Interconnection (OSI) Reference Model* and the Media Access Control (MAC) protocol. The MAC–protocol will be more thoroughly presented in this chapter due to its importance in Ethernet frame transmission(s), whereas specific protocols to the thesis are only briefly presented in Section 2.1.2.

The Ethernet network topology took its toll in 1973 when Bob Metcalfe started extending the use of the Aloha network, which was invented in the late 1960s for radio communication between the Hawaiian Islands. After a few years of research, the paper "Ethernet: Distributed Packet Switching for Local Computer Networks" was presented in 1976, which lead to a U.S patent number 4,063,220 on Ethernet for "Multipoint Data Communication System With Collision Detection". The patent was received by Robert M. Metcalfe, David R. Boggs, Charles P. Thacker, and Butler W. Lampson. Ever since the first patent, the Ethernet have been researched and improved, resulting in more affordable and consumer friendly devices with Ethernet capability. The research lead to the *IEEE 802.3 Standard for Ethernet*, where the newest version is from 2012 (IEEE 802.3:2012 2012; Spurgeon 2000: 3-7).

2.1. Ethernet frames and building blocks

In 1978 the International Organisation for Standards (ISO) developed a standard model for layered network protocol implementation, named the *Open Systems Interconnection (OSI) Reference Model*. The purpose of the OSI model was to enable an easy implementation and communication between different protocol layers. The OSI model is presented in Figure 1, and consists of seven layers where each have their own purpose and implemen-

tation standards. The higher level layers in the OSI model are application specific, and handles the reliability of data transmission and the presentation of the transmitted data (to the user), whereas the lower two layers are more intended for Ethernet transmission and handles the Media Access Control (MAC) protocol.

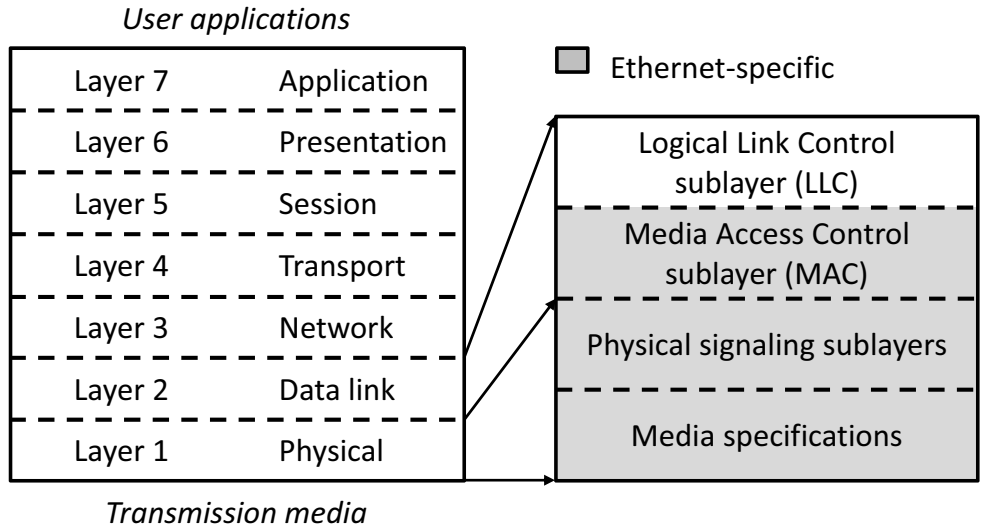


Figure 1. The Open System Interconnection (OSI) Reference Model *(left)* with corresponding Ethernet 802.3 sublayers *(right)* (Adapted from Spurgeon 2000: 13).

A short description of the seven OSI layers presented in Figure 1 is listed below, where the description begins with the lowest layer.

Physical layer

Defines the standards for mechanical, electrical and functional control of data circuits on a physical level including e.g. signal levels, cabling and hardware specifications.

Data link layer

Allows direct communication from station to station on a single link provided by the physical layer, where the communication can be point–to–point or point–to–multipoint depending on the network architecture. This layer receives/transmits

frames, defines standards for the Ethernet frame format and also handles the MAC protocol.

Network layer

Responsible for the communication from station to station across the Internet. The network layer is partly independent from the two lower layers and can exchange data on a higher level of protocols, only by issuing its requests to the link layer. The protocols defined for the network layer (and the upper layers) are independent from the Ethernet standard. A common protocol that is used is the Internet Protocol (IP) that a majority of devices use to communicate with each other.

Transport layer

Handles the end–to–end connection, flow control, congestion control and datagram delivery. Two common protocols used by the transport layer are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), which are used along with the IP protocol of the network layer.

Session layer

Concerns the establishment of reliable communication between applications including opening, managing and closing sessions.

Presentation layer

Encodes the received data on an application specific basis and presents the data to a particular application.

Application layer

The first link between the end–user and the software and provides mechanisms for end–user applications like chat, mail, file–transfer etc.

As the OSI model and the IEEE standards was developed for enabling compatibility between different hardware and software manufacturers, the MAC–protocol also plays an important role in Ethernet communication. The MAC–protocol implements (as previously mentioned) the Data Link Layer, and describes the structure of the Ethernet frames that

are transmitted over Ethernet networks. Figure 2 shows the structure of an *Ethernet frame* and an *Ethernet packet*. (Edwards & Bramante 2009: 374, 379, 403, 435) (Spurgeon 2000: 10-12).

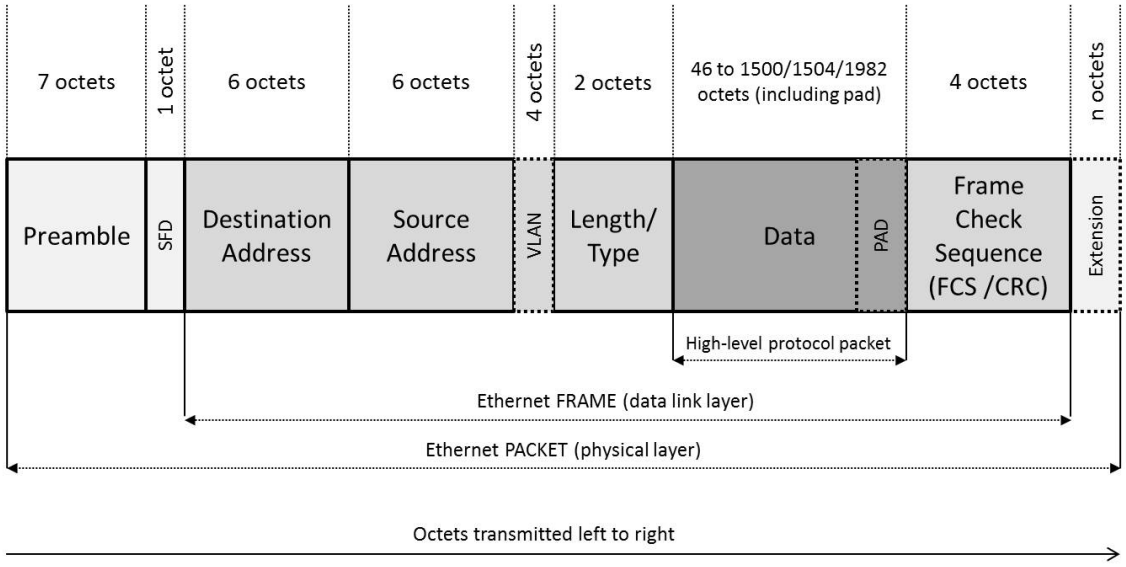


Figure 2. Structure of the Media Access Protocol (MAC) protocol/frame involving fields and their size in octets, where 1 octet equals 8 bits (Adapted from IEEE 802.3:2012: 53).

The Ethernet frame is encapsulated in the Ethernet packet, where the fields *Preamble*, *Start Frame Delimiter (SFD)* and *extension* belongs to the Ethernet packet. According to the IEEE 802.3:2012 standard there are three types of MAC frames that can be transmitted over the Ethernet, where each different frame type affects the total possible size of the frame and its usage:

- Basic frames, used in most cases
- Q–tagged frames, used e.g. by the IEEE 802.1q:2011 standard
- Envelope frames, large frames

The contents of the Ethernet frame along with their uses are described below:

Destination Address

The destination address field defines the receiver(s) of the Ethernet frame that is transmitted. There exists three types of MAC addresses: *Unicast* where the receiver is a single physical device on a Local Area Network (LAN), *Multicast* where the receivers are a specified group of physical devices on a LAN and *Broadcast* where the receivers are every connected physical device on a LAN.

Source Address

The source address is the identifier of the physical device that transmitted the frame and will never change for the sender.

Virtual Local Area Network (VLAN) tag header (optional)

The VLAN tag header can be inserted between the source address field and the length/type field and specifies what part of a virtual local area network the transmitted frame belongs to. The VLAN tag header derived in the IEEE 802.1Q standard is used e.g. in the GOOSE protocol that will be discussed in Section 2.1.2. Frames that include the VLAN tag header are called Q–tagged frames, as described earlier in this section. (Jaakohuhta 2005: 366) (Kriger, Behardien & Retonda-Modiya 2013: 709) (IEEE 802.1q:2011).

Length/Type

A 2 octet field that specifies the length of the data field (in octets) if the decimal value of the Length/Type field is less than or equal to 1500. If the decimal value is greater than or equal to 1536 the Length/Type field defines the higher layer protocol transmitted in the data field. The values for the protocols can be found in the *Public Registered Ethertype List* provided by the IEEE–organisation (IEEE 2013). The values of the most relevant protocols for this thesis are listed below:

- IPv4 Internet protocol 0x0800
- IEC 61850 GOOSE protocol 0x88b8

- IEC 61850 GSE management services protocol 0x88b9
- IEC 61850 Sampled Values Transmission protocol 0x88ba
- IEEE Std 802.1Q – Customer VLAN Tag Type 0x8100
- IEEE Std 802.1Q – Service VLAN Tag Type 0x88a8

Data (including PAD)

The Data field (also known as payload) can hold any sequence of octets up to a maximum length defined by the specific implementation, where the maximum length of the data field is defined by the frame types listed below:

- Basic frames, maximum length is 1500 octets
- Q–tagged frames, maximum length is 1504 octets
- Envelope frames, maximum length is 1982 octets

The data field (usually) holds a higher level protocol, such as the (combined) TCP/IP–protocol or the GOOSE protocol, which also have their own protocol headers and data fields. The *pad* field is used to append information to the data field, if length of the data is less than 46 octets.

FCS (Frame Check Sequence)

The *FCS* or *CRC* (Cyclic redundancy check) holds a polynomial calculated by the transmitter where all the fields of the MAC–frame are included in the calculation (except the FCS field). The receiver then uses the same equation to determine if the frame was corrupted during the transmission, and if the result achieved by the receiver differs from the value of the FCS field, the frame is discarded. This is to ensure that a correct transmission have occurred.

2.1.1. Internet protocol suite

The Internet protocol suite consists of the IP–protocol and the TCP–protocol, and is considered to be the most used standard of the Internet (can also be referred to as the *TCP/IP*

protocol suite). The Internet protocol suite enables data communication (i.e. packet transmission/reception) between nodes that are connected to the Internet, and can be included in many operating systems and devices that implements the Ethernet standard. The Internet protocol suite operates in the five upper layers of the OSI–model shown in Figure 1, enabling the implementation of many different protocols inside the Internet protocol suite. However, the most important ones are the TCP– and IP–protocols. Figure 3 presents the embedding of IP– and TCP–protocols in the MAC–frame, where the underlying protocol is always located in the data field of the parent protocol (Edwards & Bramante 2009: 197-198).

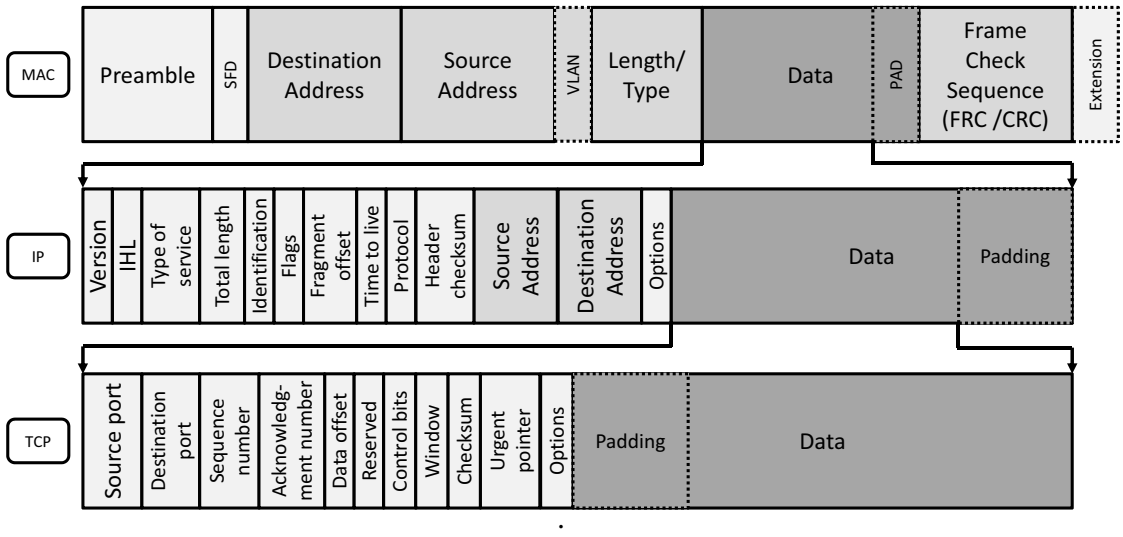


Figure 3. The main part of the Internet protocol suite embedded in a MAC–frame, where the underlying protocol is always located in the data field of the parent protocol. (RFC: 791: 38)(RFC: 793: 15)(IEEE 802.3:2012: 53)

2.1.2. IEC61850 – Communication networks and systems in substations

In order to assure high–speed reliable network communication in time–critical Substation Automation Systems (SAS) between Intelligent Electronic Devices (IEDs), a clear and concrete set of rules and protocols must be used. The IEC 61850 standard was developed to achieve this goal when the first version was published in 2003. The main goal of the IEC 61850 was to enable a minimum set–up effort and auto–configurable IEDs giving the

possibility of IEDs from different manufacturer to communicate with each other (Mackiewicz 2006). Figure 4 shows a conceptual implementation of the IEC 61850 standard in a SAS, where the arrows represent IEC 61850 communication protocols transmitted over the Ethernet in a MAC–frame (Söderbacka 2013: 15-17). The developed EtherTester presented in Section 4.1.2, is later connected to a simulated station bus, as shown in Figure 4. (IEC 61850-1 2003).

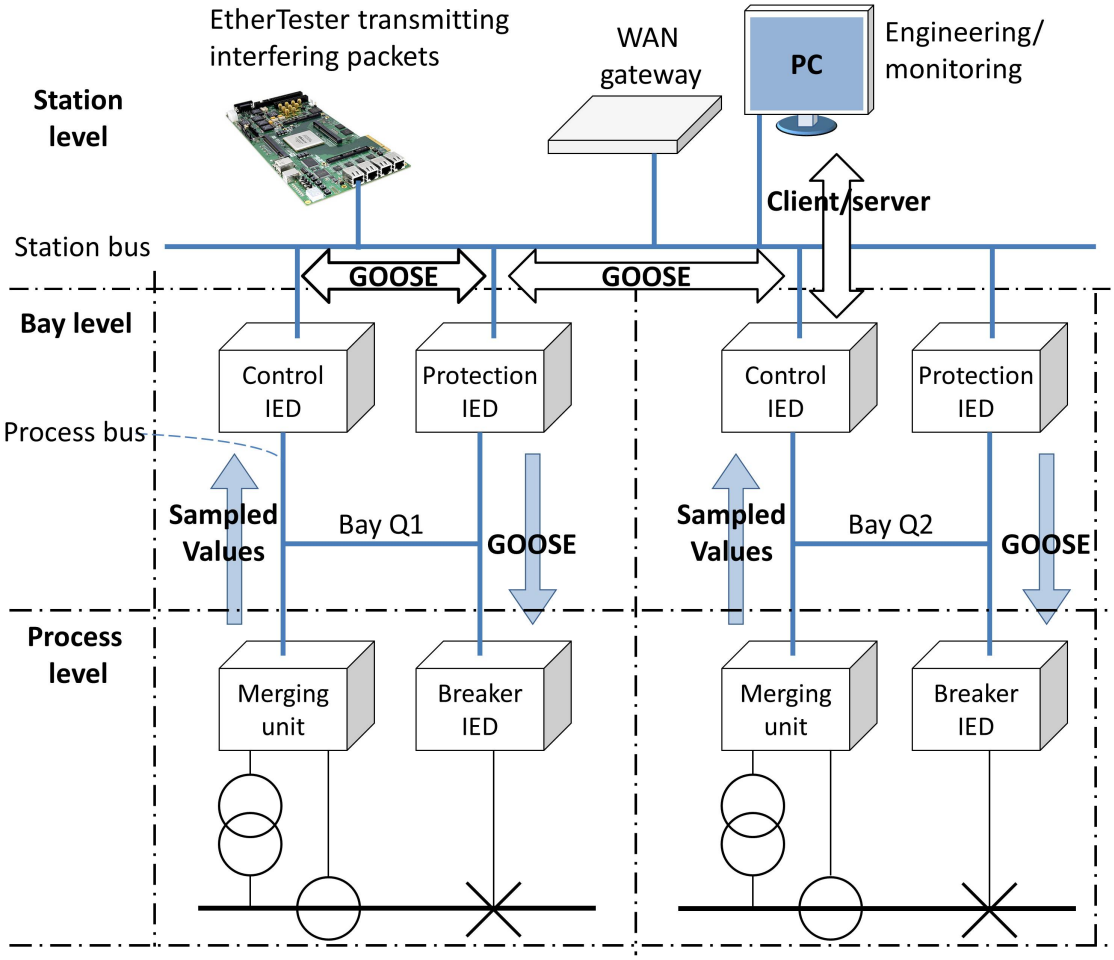


Figure 4. Conceptual implementation of the IEC 61850 standard in a SAS, where the arrows represent IEC 61850 communication protocols and the EtherTester is connected to the station bus. (Adapted from Söderbacka 2013: 15.)

As previously mentioned in Section 2.1.1, nodes supported by the Internet protocol suite that are connected to the Internet can communicate with each other without concerning the type of operating system used. This also applies to IEDs which are connected to the station bus as in Figure 4, shown in the Client/server arrows. However IEDs must also transmit time–critical packets between each other resulting in the Sampled Values (SV) and Generic object–oriented substation event (GOOSE) messages/protocols mapped directly onto the link layer. This time–critical communication proposed by the IEC 61850 standard is the most important target for this thesis and tests will be executed in Chapter 6 to analyse the reliability of the GOOSE communication protocol implementation in IEDs.

3. EVOLUTIONARY COMPUTING

This chapter gives the reader an overview to evolutionary computing, where the most important historical events (in evolutionary computing) are presented along with some of the most common evolutionary algorithms. Section 3.1 introduces the reader with the purpose of these algorithms and also the basic structure of an evolutionary algorithm. Later in the Section 3.2 this thesis will go deeper into the genetic algorithm and thoroughly present its functionality and structure, also a simple example of a genetic algorithm will be evaluated on a 1D–landscape with multiple local optima in Section 5.

Evolutionary computing is a (sub)–field of Computer Science, where the fundamentals of problem solving have arisen from the theories of evolution in nature. These theories were presented by Darwin (1859) in "Origin of the species", and they are the basis of for what is today known as evolutionary algorithms (EA). Evolutionary algorithms can be used in vast amount of ways to solve complex problems by using an improved version of the trial–and–error technique, where the so called samples of the trials are evolved between each run into a better sample until an acceptable result is acquired.

The field of the evolutionary computing reaches all the way back to the 1930s, when Sewell Wright introduced the idea of visualizing an evolutionary system for exploring a multi–peaked fitness landscape and forming sections around peaks of high fitness. The field of study however did not take its toll until the 1960s, when computers could be used as a tool for simulating and modelling evolutionary algorithms.

There were a couple of groups in the 1960s that showed a vast amount of interest in evolutionary algorithms, which also shaped this emerging field by their research. Rechenberg and Schwefel at the Technical University in Berlin, presented the idea of using evolutionary algorithms as a method for function optimisation, which evolved to the principle of evolution strategies (ES). Fogel at UCLA developed an evolutionary framework called evolutionary programming (EP), which was able to enhance the function of state machines over time. At the University of Michigan, Holland used evolutionary systems to develop

robust adapting systems that could handle irregular and changing environments, by including the feedback from the operational environment to alter the system itself. This formed the basis for what is today called genetic algorithms (GA). These three methods (EV, ES, GA) that were developed in the 1960s have been studied ever since, and is researched and improved continuously. Another milestone was in the beginning of the 1990s when Koza (1992) introduced the concept of genetic programming (GP), which evolved from GA into an evolutionary algorithm that could generate computer programs for specific tasks. These computer programs were composed by a multitude of functions that where selected and connected together (using genetic algorithms) to achieve the desired program. There are also a lot of other evolutionary algorithms that have evolved from the previously mentioned ones but they will not be discussed in this thesis. (Eiben & Smith 2007: 1-3) (Jong 2006: 23-29)

3.1. Evolutionary Algorithms

Even though there are many different kinds of evolutionary algorithms (ES, EP, GA, GP, et al.) they still share the same principle regarding their structure and the abstract means of achieving the desired result. They all have an initial population which is stochastically generated, that will be judged (evaluated) for their suitability (fitness) to the environment, where the environment represents the search space of the specific problem. This population will be bred through recombination into a new population where the parents with higher fitness will have a higher chance to breed (i.e. produce an offspring). Recombination mostly occur between a pair of individuals, resulting in a new pair of individuals that holds the properties of the parents. The resulting offspring might also be stochastically mutated to add some randomness to the evolutionary process. After the recombination and/or mutation the offspring will be evaluated again, and the process circles until an appropriate individual in the population is found, or another desirable search criteria is reached. This search criteria could be for example to maximize a function, generate a fast and efficient computer program, search for the shortest path between a number of routes, or to increase the efficiency of a proportional–integral–derivative (PID) controller

(Mirzal, Yoshii & Furukawa u.d.). The pseudo–code for an evolutionary algorithm is presented in Figure 5.

```
BEGIN
    INITIALIZE population stochastically
    EVALUATE each individual from population
    WHILE (TERMINATION CONDITION IS NOT SATISFIED)
        1. SELECT parents
        2. RECOMBINE pairs of parents
        3. MUTATE the resulting offspring
        4. EVALUATE new individuals
        5. SELECT individuals for the next generation
    END WHILE;
END
```

Figure 5. Pseudo–code for an evolutionary algorithm, where the stages are abstract and vary in each different implementation of an evolutionary algorithm. (Adapted from Eiben & Smith 2007: 16.)

The differences between the evolutionary algorithms applies mostly to their application area, but also in the representation of the individuals in the population. An individual in a GA is often represented by a bit–string of length n, which makes it appropriate for this Master's thesis, since Ethernet frames are on a physical level represented by digital voltages i.e. on a software level by binary numbers where n is the length of the Ethernet frame. In ES an individual is represented by real–valued vectors, in EP by a finite state–machine (FSM) and in GP by trees. Another varying aspect in evolutionary algorithms is the way the different stages in Figure 5 are implemented. For example in GA the recombination could be discrete or intermediate whereas an EP can have no recombination at all and the individuals are only evolved by mutation. In this thesis only the stages that apply to genetic algorithms will be presented thoroughly in Section 3.2. (Eiben & Smith 2007)

3.2. Genetic Algorithms

There are many traditional search methods available to address specific problems, when the search space is relatively small and the constraints of the system are within fair ranges. However when the complexity of the system increases and the search space is large, the amount of approaches are limited. This is where the genetic algorithms can be introduced. GAs (and other evolutionary algorithms) are based on the assumption that when randomly combining good sub–solutions, the probability of getting a better new solution is high, which relates good to the idiom *you can't make bricks without straw*. However there is not actual mathematical proofs of the functionality of a GA, and this is one of the main reasons why mathematicians may oppose these search methods. However there have been many successful implementations of GAs where traditional search methods have proven inefficient (Alander 1998: 18).

In genetic algorithms, the samples to be evaluated and evolved are often called as *chromosomes*, they are constructed like bit strings of length n, as shown in Figure 6. The chromosome can be divided in parts called *genes* that represents the variables or different properties of the chromosome. Every single bit in a chromosome is called a *locus* and can take the *allele* values of '0' or '1'. A chromosome has at least one gene in which case the length of the chromosome is the same as the length of the gene, but usually a chromosome contains many genes. A set of chromosomes is called as a *population*. In a population each chromosome have the same structure and the amount of chromosomes is the size of the population. A single chromosome in a population can also be referred to as an *individual*. (Mitchell 1998: 8).

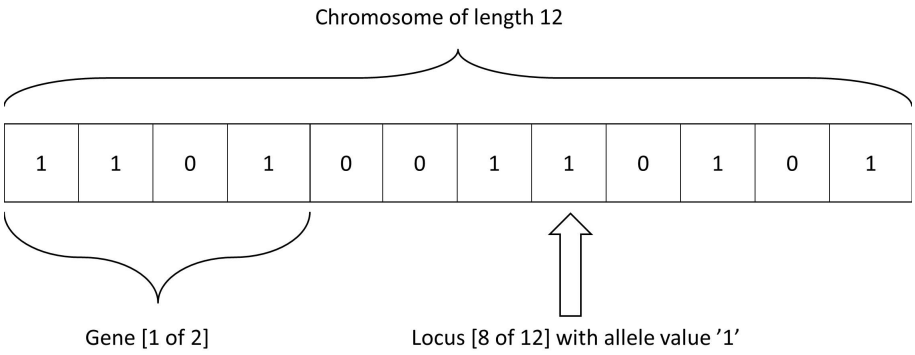


Figure 6. A bit string chromosome of length 12 with 2 genes and 12 loci.

When simulating a genetic algorithm each iteration of the simulation loop is called a *generation*, where as presented in Figure 5, the population is evaluated, selected, recombined and mutated (more on this in sections 3.2.1, 3.2.2 and 3.2.3). For each chromosome in the population to have high chance of breeding, it must be assigned a high score from a fitness function. The fitness function could be for example to maximize a real–valued one–dimensional function, as shown in Equation 1,

$$f(x) = x + |sin(32x)|, \quad 0 \geq x \leq \pi \tag{1}$$

where x is a chromosome in the population, and $f(x)$ gives the fitness of that particular chromosome. In case the function would be known one could normalize the result by dividing it with the maximum value of $f(x)$ hence scaling the fitness between 0 and 1. However this is seldom appropriate in search algorithms since if the algorithm would already know the best result in the search space it would not have to search at all. In most cases the optimum is unknown meaning that $f(x)$ returns an unscaled fitness value, and the higher fitness the better since the task was to maximize the function. Because the maximum fitness is not known the genetic algorithm does not know when to stop, and even though the GA might have found a high fitness value, the GA could be stuck on a local optimum. The search space of the previously mentioned problem is defined in equation 1 where a chromosome can only take the values between 0 and pi, this could

also be defined as the fitness landscape. Since equation 1 only takes one value as an input arguments, the chromosome will only hold one gene, and this particular gene is usually a binary representation of the real number, where the precision of the binary representation is dependent on its length i.e. the gene length. (Mitchell 1998: 8-9)

The are many kinds of operators in genetic algorithms, however only the operators that apply to binary–string representations are presented in this section. Other representations like integer, floating–point or character representations will not be discussed since this thesis will focus on Ethernet traffic which is constructed of bits. The three main operators that are used in most genetic algorithms are listed below, and will be presented in the succeeding sections.

- Mutation
- Recombination
- Selection

These operators where also defined in the pseudo–code for an evolutionary algorithm showed in Figure 5.

3.2.1. Mutation

The mutation operator applies to only one individual at a time, and the outcome of the mutation is a slightly altered individual that still has an unchanged structure. The most common mutation operator for binary–string representation is the bitwise mutation operator, however some other types have occasionally been used but will not be discussed here. The bitwise mutation gives each bit (locus) a probability P_m to be mutated, and if a randomly generated number falls inside that certain probability the bit will be flipped, i.e. the allele value is changed from '0' to '1' or from '1' to '0' (Chambers 2001: 14). Figure 7 shows the original chromosome and the mutated chromosome, where the loci 2, 7 and 8 have been flipped. Note that the indexing loc_i of the loci ranges from 0 to 9, but the length of the chromosome is the number of loci i.e. 10 (Eiben & Smith 2007: 42-43).

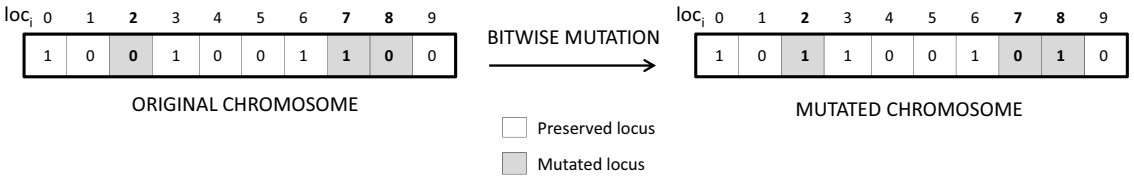


Figure 7. Bitwise mutation of a single–gene chromosome with locus index loc_i from 0 to 9 and chromosome length $L_{ch} = 10$, where the loci 2, 7 and 8 have been flipped. (Adapted from Eiben & Smith 2007: 43.)

3.2.2. Recombination (crossover)

Another operator in the GA is the **recombination**, which in some cases is referred to as **crossover**. The recombination occurs between 2 or more parents and produces an offspring of equal size. Recombination is considered by many to be the most important feature of genetic algorithms, and is the one operator that distinguishes GAs from traditional search operators. The recombination may also implement the crossover parameter R_{cr}, which determines the probability of the crossover. First two parents are selected, secondly a random number is drawn from the range of [0, 1], if the random number is larger than R_{cr} the parents will be recombined. Otherwise the parents will be bred asexually, meaning that the resulting offspring is an identical copy of the parents. Note that this offspring might still be altered through mutation as discussed in the previous section. There are three main recombination operators commonly used for binary–representation chromosomes:

- One–point crossover
- n–point crossover
- Uniform crossover

One–point crossover is the simplest of all crossover operators. The purpose of the one–point crossover is to split two chromosomes at a randomly selected crossover index cr_i in the range of [0, L_{ch}–2], where L_{ch} is the chromosome length. The two chromosomes then swap the information that resides after the selected point, i.e. the chromosomes are exchanging their tails (Goldberg 1989: 12). Figure 8 shows the functionality of an one–point crossover with crossover index cr_i at 2 and a chromosome length L_{ch} of 8 (Eiben & Smith 2007: 47-48).

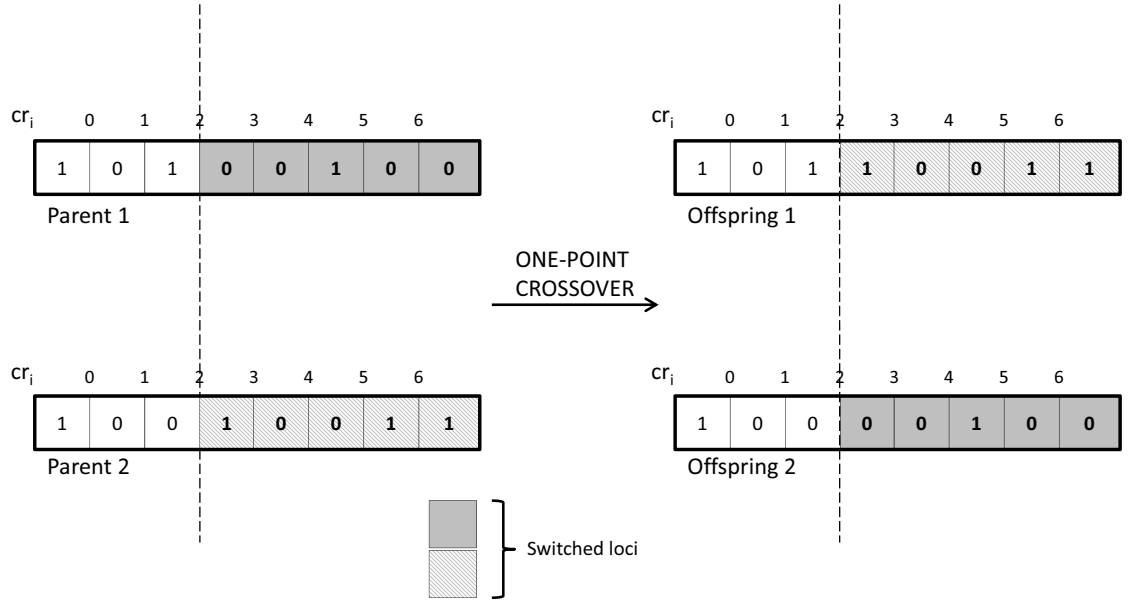


Figure 8. One–point crossover at crossover index $cr_i = 2$ on chromosomes with chromosome length $L_{ch} = 8$. (Adapted from Eiben & Smith 2007: 48.)

A second more generalized form of crossover is the **N–point crossover**, where the chromosome is broken down to $n + 1$ segments that are exchanged between the parents to produce a new offspring. In N–point crossover a number of crossover points cr_N are selected randomly in the range of [0, L_{ch}–2] where L_{ch} is the length of the chromosomes, same as in the one–point crossover (Eiben & Smith 2007: 48). Figure 9 shows a N–point crossover where $cr_i = \{2, 5\}$, $N_{cr} = 2$ and $L_{ch} = 8$.

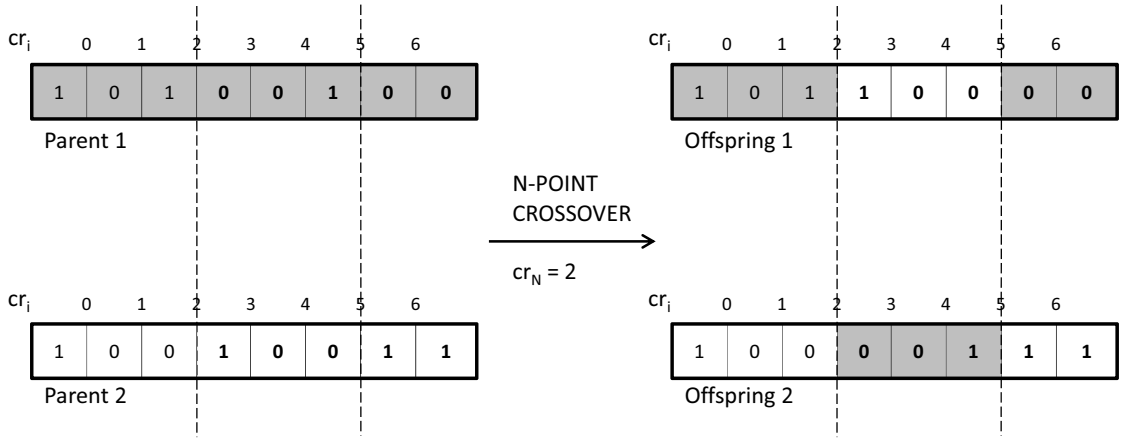


Figure 9. N–point crossover at crossover index $cr_i = \{2, 5\}$ on chromosomes with $L_{ch} = 8$. (Adapted from Eiben & Smith 2007: 48.)

The last type of crossover operator is the uniform crossover, where each locus in the chromosomes are treated individually, as presented in Figure 10. Uniform crossover is implemented by generating uniformly distributed random variables within the range [0, 1] and applying them in a crossover vector of length L_{ch}. A crossover probability parameter P_{cr} is then assigned and if a random variable in the crossover vector falls within that probability, the loci at the index of the random variables are then exchanged (Beasley, Bull & Martin 1993: 2). The crossover probability parameter P_{cr} is often assigned to 0.5 (Eiben & Smith 2007: 48-49).

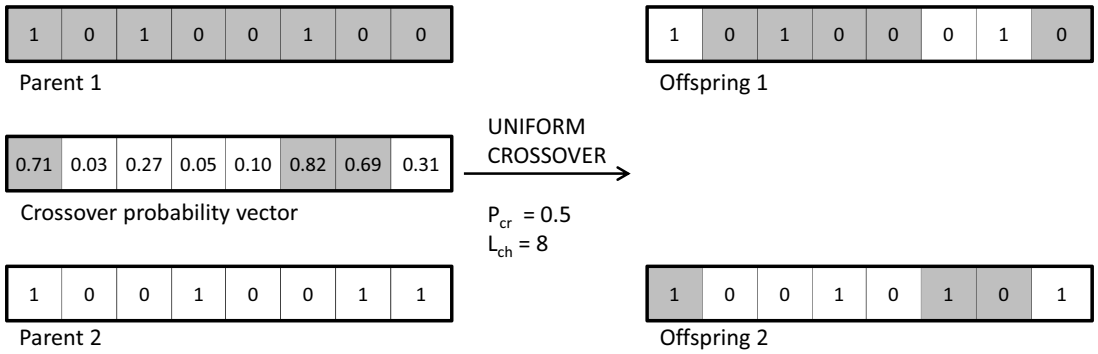


Figure 10. Uniform crossover with crossover probability parameter $P_{cr} = 0.5$. (Adapted from Eiben & Smith 2007: 49.)

The three different kinds of crossover operations that were presented (one–point, N–point, uniform) in Figures 8, 9 and 10 all represents a different kind of strategies. However all crossover operations do work on the principle of randomly mixing parents to produce offspring. The N–point crossover shows a trend of keeping together sections that are located close to each other in the representation, where as when N_{cr} is odd (e.g. in one–point crossover) it shows a strong will of preserving the location of genes in the opposite ends of the chromosome. This phenomena is known as **positional bias** and is the most important property of the N–point crossover. The uniform crossover however shows a tendency of mixing 50 % of the loci on the parents, resulting in what is known as **distributional bias**. Even though it cannot be exactly defined which one of the crossover operators is the most suitable in every case, one must know the different properties of the crossover operators in order to select the most appropriate one (Eiben & Smith 2007: 49).

3.2.3. Selection and GA types

In the previous sections the two different operators mutation and crossover have been discussed, where the maximum number of affected individuals by the operators have been two. In this section the **selection operator** will be presented and the focus will be on the entire population, where new terms arise such as population size μ and offspring size λ, and also two different types of genetic algorithms will be presented.

For every selection operator that is applied, one must also define how the operator is used based on the current GA model. There are mainly two GA models that will be discussed:

- A generational model.
- A steady–state model.

The **generational model** has the property of replacing its entire population with an offspring, called the **next generation** that implies $(\lambda = \mu)$. More thoroughly described each iteration starts with a population of size μ, and by creating a mating pool from the parents, GA operators are applied to the mating pool to create an offspring population where

$(\lambda = \mu)$. The **steady–state model** on the other hand, does not change the entire model at once which implies that $(\lambda < \mu)$, and the so called **generational gap** that occurs is the percentage of the population that is replaced, which equals (λ/μ).

With the GA models subject briefly covered the selection operator types listed below will be presented, where the **parent selection** algorithms applies to the beginning of each generation as showed in Figure 5, and defines the parents that will be used for recombination and/or mutation. The **survival selection** algorithm however define the individuals that will survive for the next generation, and is executed last as shown in Figure 5. It must be mentioned that in some cases, individuals that survive can be the whole bred offspring if $(\lambda = \mu)$.

- Parent selection.
  - — Fitness proportional selection.
  - — Ranking selection.
  - — Tournament selection.
- Survival selection.
  - — Age–based replacement
  - — Fitness–based replacement

The **fitness proportional selection (FPS)** states that the selection probability of an individual depends on the absolute fitness values of the individuals compared to the absolute fitness values of the whole population, where for each individual the probability that it will be selected for mating is equal to equation 2,

$$P_i = f_i / \sum_{i=0}^{\mu} f_i \tag{2}$$

where f_i is the fitness of an individual and P_i is the probability that it will be selected. The FPS however have some problems, one example is that outstanding individuals (i.e.

individuals with very high fitness) will take over the population very quickly, resulting in premature convergence. Another problem with the FPS occurs when the fitness values of the population are close together, meaning that there is no selection pressure.

The **ranking selection** algorithm instead implements selection pressure, by sorting the population on the basis of fitness, and then applying selection probabilities based on the rank of the individuals. The ranking numbers can be mapped to selection probabilities in different ways, e.g. by decreasing the selection probability linearly or exponentially. The common formula for mapping selection probabilities based on rank is described in equation 3,

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)} \tag{3}$$

where $P_{lin-rank}(i)$ is the selection probability of the individual and s is a user–defined parameter where $(1.0 < s \leq 2.0)$ that regulates the selection pressure of the algorithm. The selection pressure of the linear ranking is however limited, since the maximum value of s is 2. This limitation originates from the assumption that an individual of median fitness should have one chance (on average) to reproduce. The exponential ranking algorithm in equation 4 allows instead a higher selection pressure and emphasizes more on selecting an individual of above–average fitness,

$$P_{exp-rank}(i) = \frac{1-e^{-i}}{c} \tag{4}$$

where c is a normalisation factor that is function of the population size, and must be chosen so that the sum of probabilities is unity.

Tournament selection is a popular algorithm due to its simplicity and lean use of computational resources. Simply by choosing K number of individuals randomly and selecting the one with the best fitness, one can effectively select new individuals for the mating pool without performing too complex algorithms. The selection pressure is easily regulated by varying the K parameter, but also by introducing a selection probability parameter P_{sel}.

In most cases $P_{sel} = 1$ leading to deterministic tournaments, but P_{sel} can also be less than one in stochastic versions of the tournament i.e. $P_{sel} < 1$. However, the selection probability parameter is not used in this thesis and will not be discussed further.

Age–based replacement focuses only on the number of generations a particular individual have existed, and for the most simplest cases the whole population is replaced, because the generated new population is of the same size as the old population i.e. $\mu = \lambda$. The age–based replacement can be defined as *Time To Live* (TTL) and defines the number of generations a single individual can exist. In case that the new population produced is smaller than the old population ($\lambda < \mu$), the individuals that will be added to the new population will be the youngest individuals (i.e. those that have existed for the least amount of generations). Nonetheless, the age–based replacement technique is not used in this thesis.

Fitness–based replacement focuses merely on the fitness of the particular individuals as the name states. After breeding, both parents and offspring exist in a common pool, which size is $\lambda + \mu$, from where they can be selected for survival. One can implement **elitism** in a GA, by conserving the most fitted individuals in the population. Another implementation is the **replace worst** scheme, where the least–fitted individuals are selected for replacement. However these two schemes may cause the GA to lead to premature convergence resulting in a stall at a local optimum. (Eiben & Smith 2007: 58-66)

4. EMBEDDED SYSTEMS

This chapter gives the reader an introduction to different kinds of embedded devices available on the market along with their functionality and benefits, where the Field–programmable Gate Arrays (FPGAs) will be more thoroughly discussed. The Altera DE4 developement board will also be introduced along with the EtherTester software (developed by Olli Rauhala) that serves as the foundation for the EthGA software that has been developed by the author.

Embedded systems is an advanced field of technology, where the number of different applications and implementations are vast. In the early days signal processing was achieved using merely analogue components, but as for today most signal processing is handled by digital logic (i.e. with an embedded system). The founding component for an embedded system is shared among all manufactures, namely the transistor, a semi–conductor that was developed in the late 1940s. The invention of the transistor soon led to the development of discrete digital logic gates (Boolean logic) incorporated into integrated circuits (IC), which enabled the manufacturing of programmable logic devices (PLD), processors and application specific integrated circuits (ASIC).

PLDs are often used when the functionality of the system is too complex for a simple discrete logic IC implementation. The benefit of PLDs is re–programmability (in most implementations) that is not possible in a discrete logic system without re–routing the components and/or the wiring on the printed circuit board (PCB). There are mainly three different types of programmable logic: *(a)* the simple programmable logic device (SPLD), *(b)* the complex programmable logic device (CPLD) and *(c)* the field programmable gate array (FPGA) which will be discussed more thoroughly in Section 4.1. The SPLD has a very simple programming capability and was introduced before the CPLD and the FPGA. A PLA–architecture SPLD has cross–points which can be connected through programming, which connects the inputs to the AND gates and the AND gates to the OR gates located inside the IC. However the PLA–architecture can only be programmed once but the IOs can be connected to each other afterwards by re–routing the wires to the IC. There are

also other architectures of the SPLD e.g. the programmable array logic (PAL) and generic array logic (GAL) architectures. The PAL architecture is simpler than the PLA consisting of only one AND plane, and can also be programmed once. The GAL architecture on the other hand implements the electronically erasable programmable read–only memory (EEPROM) that enables re–programmability. A CPLD is an improvement of the SPLD, and can be considered as an implementation of several SPLDs.

Another set of programmable devices in the embedded family are the processors, where the most common types are: *(a)* the Microprocessor (μP), *(b)* the Microcontroller (μC) and *(c)* the Digital signal processor (DSP). The microprocessor is not developed to solve specific problems but to apply to a vast amount of different tasks, resulting in a lack of optimization due to this generalisation. It is integrated in a single circuit that is programmable through software. The microcontroller is an extension of the microprocessor by embedding interfaces (e.g. UART RS–232) and memory registers/controllers inside the microprocessor. It reduces the implementation costs compared to creating a similar microprocessor system with external interfaces added to a PCB, but limits the flexibility due to the limitations embedded inside the IC.

A DSP is an application specific processor that is designed for real–time complex signal processing algorithms, such as the fast fourier transorm (FFT) for example. Hardware multipliers and hardware loops are often embedded in the DSP and can be accessed trough a programming interface. In a standard microprocessor these operations would be performed using shift operations, additions and software loops. (Grout 2008: 21-22)

An application–specific integrated circuit (ASIC) is an IC that is built for a specific task, that could incorporate any of the functions and components previously mentioned in this chapter into a single chip. The ASIC design is sealed after manufacturing resulting in the loss of modification options. However, the ASIC can be optimized by scaling down the power consumption to only consume what the design needs, in contrast to a general–purpose microprocessor. The advantages of an ASIC solution arises when a certain configuration is used vastly, and that particular can ASIC can be manufactured in great num-

bers, hence lowering the production costs. (Deschamps, Bioul & Sutter 2006: 252)

4.1. Field–programmable gate arrays

What differs the FPGAs from the previously mentioned embedded systems in Chapter 4 is the configuration flexibility along with their fairly low power consumption. Since the capacity of the FPGAs are constantly increasing one can program them to perform almost any task that for example a microprocessor or a DSP could perform, and also implement additional functions/tasks besides the processor. It is also possible to perform operations in parallel which enhances the computational speed compared to a microprocessor, where almost all instructions are executed sequentially. FPGAs can often be used as the prototyping platform for products that will eventually be created as ASIC chips, however FPGAs are also often used in the final products, depending on the desired performance, re–programmability, development and production costs. Nowadays FPGAs are found in a vast amount of both consumer and industrial applications such as in automotive systems, wireless communication, signal processing, aerospace, defence, medical imaging and data security only to name a few. (Deschamps et al 2006: 258) (Stavinov 2011: 7)

At the moment the FPGA market holds two leading vendors, namely Altera and Xilinx which in year 2011 held over 90 % share of the total FPGA markets. These two companies both provide a wide range of FPGA families, that offer high flexibility and both low–cost and high–performance solutions. There are also smaller companies e.g. Actel/Microsemi corporation, Lattice Semiconductor Corporation, Achronix Semiconductor and Tabula, which are focusing on different niches with their own FPGA products. (Stavinov 2011: 3-5)

Since all FPGA vendors have their own type of FPGA architectures, this thesis will not go into details at everyone of them. However a generic model of an FPGA architecture is presented in Figure 11, where a logic cell (LC) can be programmed to perform a certain logic operation (e.g. AND/OR/NOT). The LCs can be connected to programmable I/O

cells and other LCs trough programmable interconnect routing channels. The underlying structure of the LCs, I/O cells and routing channels vary between each vendor and FPGA family, along with the functions each object can provide. (Grout 2008: 28-29).

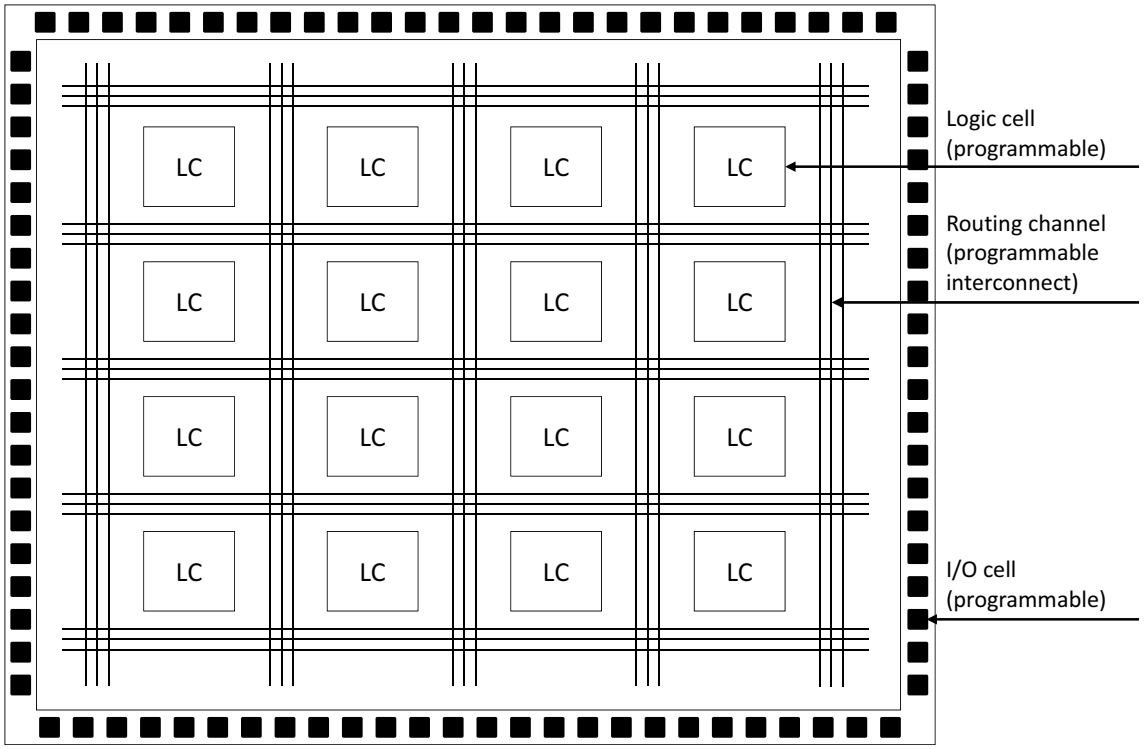


Figure 11. Architecture of a generic FPGA, where the underlying structure of the objects: *(a)* logic cell, *(b)* routing channel and *(c)* I/O cell will vary between different FPGA vendors. (Adapted from Grout 2008: 29.)

Nowadays most embedded software developers use the C programming language to manipulate the functionality of a microprocessor, microcontroller or DSP. A microprocessor follows a sequential workflow where a program pointer is used to determine the instruction to be executed. This implies that the processor will work in a non–parallel manner, executing instructions step–by–step (in most cases). The FPGA on the other hand is programmed by a hardware description language (HDL) which can be used in creating parallel hardware components. The developer creates hardware modules, which serve a specific purpose and are connected together inside a top–level design. There are for the

moment two main HDLs in use: *(a)* VHDL (very high–speed integrated circuit hardware description language) and *(b)* Verilog–HDL. VHDL was developed in 1980 by the United States Department of Defence (DoD) and merged into the IEEE 1076 standard in 1987, whose most recent version is the IEEE Standard 1076–2002. Verilog–HDL was released in 1983 by Gateway Design System Corporation and was also merged into a IEEE standard, namely IEEE 1364–2001. Both VHDL and Verilog–HDL serve the same purpose; to provide the developer a text–based language in which he/she can describe the hardware. The syntax and the structure of the two languages are different. The choice of the language depends partly on the programmers previous experience and personal preference, but also the results of the HDLs impact on the desired FPGA. (Grout 2008: 193-196).

4.1.1. Altera DE4

The Altera Developement and Education Board 4 (DE4) shown in Figure 12 was used in this thesis as the hardware platform for the developed EtherTester and EtherTesterGA software. It features the Altera Stratix IV GX FPGA EP4SGX230C2 with the specification listed in table 1. The DE4 was chosen because it holds 4 Gigabit Ethernet ports, which enables a very high frame transmission rate necessary for the case studies performed in Chapter 6. It also features a 100 MHz oscillator, which frequency can be even further increased by using phase locked loops. The EtherTester software described in Section 4.1.2 runs on the standard 100 MHz clock frequency.

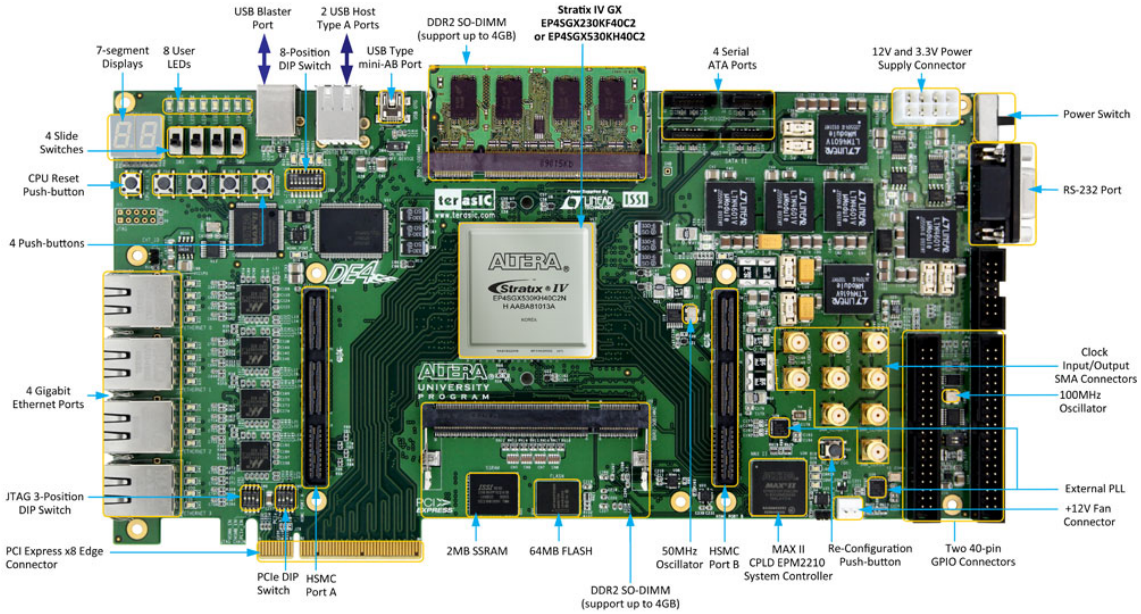


Figure 12. The Altera Developement and Education Board 4. (Terasic Technologies Inc. 2013).

Table 1. Altera Stratix IV GX EP4SGX230C2 specifications. (Terasic Technologies Inc. 2010: 9)

Unit	**Number of**
Logic elements (LEs)	228,000
Total memory	17,133 kb
18x18 multipliers blocks	2
PCI Express hard intellectual property blocks	744
User I/Os	8

4.1.2. EtherTester software

Olli Rauhala have developed the *EtherTester* software for the DE4, which enables it to send, receive and analyse Ethernet frames at gigabit speeds. Figure 13 presents the system description of the EtherTester, where the NIOS processor is the main component that controls the other hardware components. The NIOS processor is a hardware–described

microprocessor developed by Altera, which is embedded into the FPGA and programmed with the embedded C–language (like many other microprocessors). All other components in EtherTester are written in Verilog, and some of them use Altera's own Intellectual Property libraries. The EtherTester is controlled from a host computer with the JTAG/UART (Joint test action group/Universal asynchronous receiver transmitter) interface physically connected over USB. This control type is terminal based using Altera's own software, and the JTAG/UART connection is also required when using Altera's intellectual property cores.

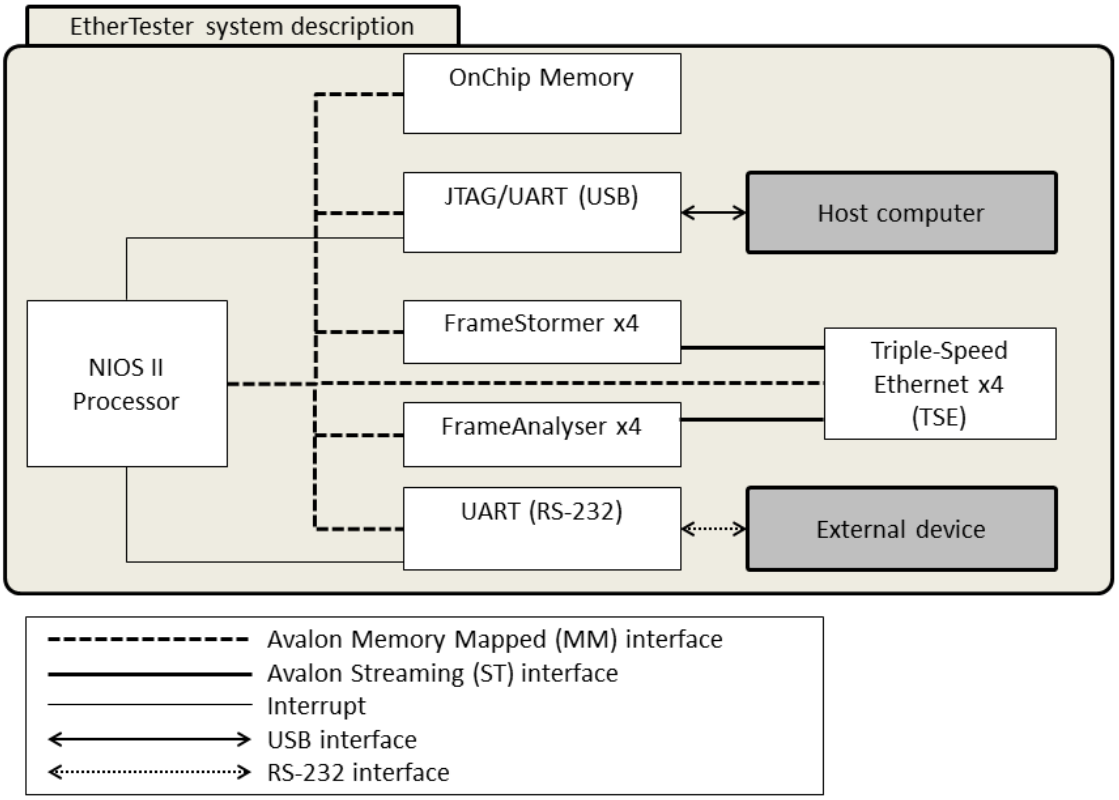


Figure 13. System description of Olli Rauhalas EtherTester software implemented on the DE4, where all components are written in Verilog and the NIOS processor controls the whole system. (Adapted from Rauhala 2013b: 3.)

The FrameStormer components handles all outgoing network traffic on the EtherTester, i.e. intended only for packet transmission. The FrameAnalyser component instead han-

dles all incoming network traffic i.e. packet reception. Both the FrameStormer and the FrameAnalyser components exist in a multiple of four, which is the number of Ethernet ports on the DE4. This enables each port to transmit, receive and analyse frames with a maximum frame size of 1600 octets. This configuration requires about 70 % of the system resources, but it can be reduced by decreasing the number of Ethernet ports in use. The microprocessor can also receive and transmit external commands through the RS–232 interface, this feature is explained more thoroughly in Section 5.1 where the EthGA software is described. The OnChip Memory component is used by the microprocessor, and is necessary for downloading and running C–code in the microprocessor.

5. PROTOTYPING, SIMULATION AND IMPLEMENTATION OF A GENETIC ALGORITHM

This chapter presents the development stage in the practical part of this project; namely to develop, simulate and implement a genetic algorithm, in a NIOS microprocessor instantiated in an FPGA chip, called as the EthGA software. The main purpose of the *EthGA* is to generate Ethernet frames, which will be transmitted to a Device–Under–Test (DUT). The impact of these frames on the DUT will be evaluated by another device that will provide feedback to EthGA. This feedback serves as the fitness score for the GA and as a result the population will evolve by time to generate the most harmful Ethernet frames for the DUT. The complete functionality of EthGA is presented more thoroughly in Section 5.1 and the test cases are presented in Chapter 6.

To produce high quality software, one must understand the concepts and functions that the software will implement. Therefore the prototyping, simulation and implementation mentioned in Figure 14, are needed in system development. Nonetheless this thesis emphasises mostly on the latter phases due to their importance to the project, and the prototyping phase will only be presented briefly.

In the prototyping phase a genetic algorithm named *BasicGA* was developed in Matlab to analyse the behaviour of the GA on a multi–peaked 1–dimensional landscape, to make the author more acquainted with the programming of GAs. Matlab was chosen due its large number of additional packages, matrix operations and graphical presentation system, which might speed up the prototyping phase. The prototype was developed in one week with the properties listed in Table 2, and the fitness function used is presented in equation 5,

$$\mathrm{fit(x)} = (\sin(5 * \mathrm{x} * 0.01) + (\mathrm{x} * 0.001) * 2 + \cos(3 * \mathrm{x} * 0.01)) - \exp((\mathrm{x} * 0.01)/15) \tag{5}$$

where x equals to a particular chromosome in the population.

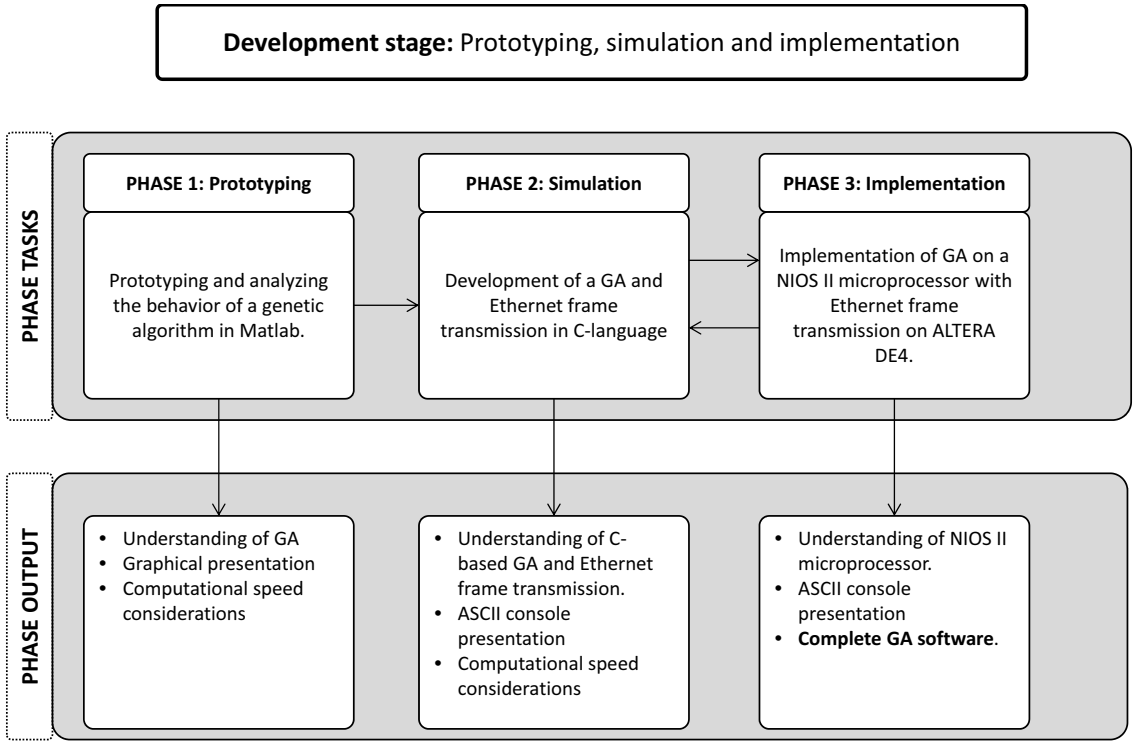


Figure 14. The different phases of the development stage, where the purpose and the outputs of each phase are presented.

Table 2. Properties of the genetic algorithm used for prototyping in Matlab, where the purpose of the algorithm is to find the maximum of a multi–peaked 1–dimensional landscape.

Parameter	Value
Chromosome type	11–bit binary
Population size	20
Selection	Roulette
Crossover	Uniform
Mutation rate	$1\ \%$
Elitism	No
Fitness Landscape	$0 \geq x \leq 2^{11} - 1$
Goal	Max of equation 5
Output presentation	Graphical & numerical

The results from a simulation with the parameters given in Table 2 is shown in Figure 15, with the initial population in the left graph and the final population after 16 runs in the right graph. The algorithm stopped when one of the individuals had obtained the largest possible value from the fitness function within the range of the fitness landscape. However the validity of this test can be questioned, since the 11–bit representation of the chromosome concludes that an individual can only take $2^{11} = 2048$ different values, resulting in a quite large probability of an individual obtaining the best fitness value at the random initialisation. Several tests showed that the time needed to find the best solution, could range between 1 and over 1000 runs. This concludes that fitness of the individuals at initialisation, plays an important role in finding the best solution in the shortest time possible. Multiple tests also showed that with elitism, the algorithm had a high probability of suffering from a premature convergence, leaving the individuals stuck at one of the lower peaks. This could be avoided by increasing the mutation rate, giving the individuals a chance to escape the lower peaks. It can also be concluded that a GA performs well at quickly finding the areas which gives the best fitness values, but not as well in quickly finding the best possible solution.

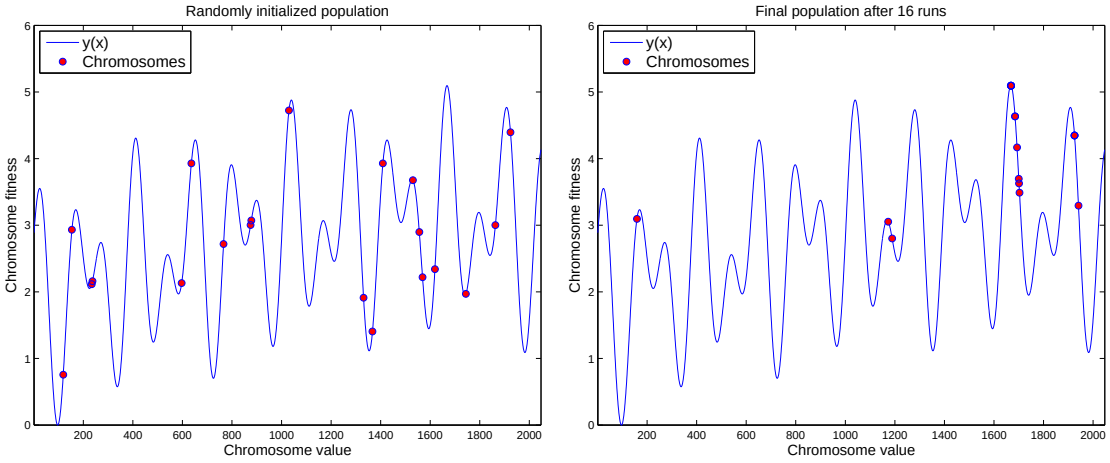


Figure 15. Test results of a BasicGA simulation with the parameters from Table 2, with the randomly initialized population in the left graph and the final population after 16 runs in the right graph.

5.1. Simulation of EthGA on a PC–computer

With the knowledge obtained from the prototyping phase, the development continued to the simulation phase, where the EthGA is written in C–language on a PC–computer (running Windows 7). The C–language was chosen because it is the language that the NIOS microprocessor uses, and it also supports transmission of hand–crafted Ethernet frames using the WinPCap library developed by Riverbed Technology. The pseudo–code for EthGA is presented in Figure 16, which describes the high–level functionality. In the simulation phase, there was no external device attached to the computer, and EthGA simulated the fitness by the so called all–ones problem, where a higher occurrence of '1' bits in a chromosome gives a higher fitness value. The usage of the external device is presented more thoroughly in Chapter 6. What differs EthGA from BasicGa is the functionality of the software: EthGA includes the transmission of Ethernet frames (i.e. individuals) and also a wider selection of crossover operations and parent selection algorithms. A complete list of features in EthGA is showed in Table 3.

```
function MAIN
{
    INITIALIZE Ethernet frame specifications
    INITIALIZE population stochastically OR according to GOOSE/SV/MMS protocols
    WHILE (TERMINATION CONDITION IS NOT SATISFIED)
        1. CALL FITNESS
        2. SELECT parents
        3. CROSSOVER pairs of parents
        4. MUTATE the resulting offspring
    END WHILE;
}

function FITNESS
{
    foreach INDIVIDUAL in POPULATION {
        TRANSMIT individual over Ethernet network
        RECEIVE fitness value from external device
    }
}
```

Figure 16. Pseudo–code for the EthGA C–language software.

Table 3. Features of EthGA C–language software.

Parameter	**Value**
Chromosome range/type	1–1600 bytes (unsigned char)
Affected genes	Selectable in range 1–1600
Population size	PC: 2–200 chromosomes
	FPGA: 2–100 chromosomes
Selection algorithms	Tournament, Ranking
Crossover algorithms	One–point, N–point, Uniform
Mutation rate	0 % – 50 %
Elitism	Yes
Fitness Simulation Algorithm	All–ones
Ethernet frame transmission	PC: Available through WinPCap library
	FPGA: Available through EtherTester software/hardware
Output presentation	ASCII–based console

As the purpose of EthGA is to execute a genetic algorithm and transmit Ethernet frames based on a simulated fitness function, some limitations must be accepted. By recalling from Figure 2 in Section 2.1, the Ethernet frame consists of several fields. A chromosome in EthGA represents the payload in the Ethernet frame, where the other fields are static and initialized in the beginning of EthGA, concluding that the GA–operations are only performed on the payload. Another limitation of EthGA is that every gene in a chromosome has the length of 1 octet (8 bits). Although both crossover and mutation operate on a bit–wise level, and all bits have an equal probability to be altered, the boundaries of the genes set the minimum length of a GA operation. This removes the capability of performing GAs explicitly on data fields shorter than 8 bits, which often occurs when the payload is mapped to a specific protocol. A demonstration of this is presented in Figure 17, where the chromosome have been mapped to the IP–protocol, and the beginning of the IP–header is shown. EthGA however has the ability of performing GA on user–defined

set of genes within the full chromosome range, meaning that GA does not necessary have to be performed on the whole chromosome. This feature is convenient when the user only wants to alter some specific fields within the protocol, and preserve important fields which need a specific value.

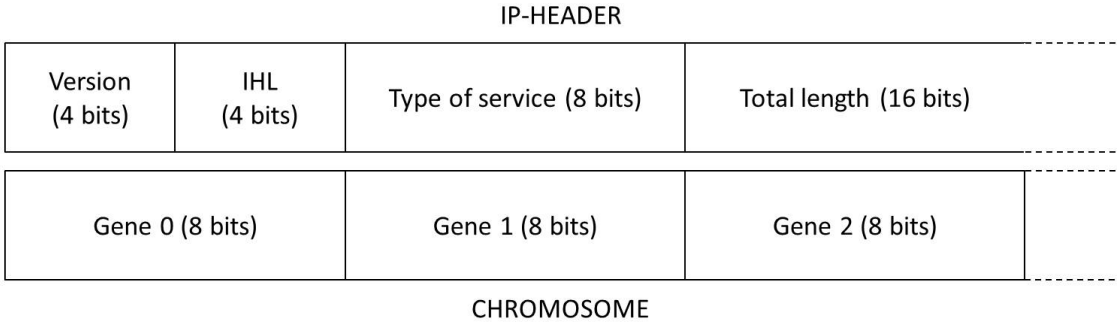


Figure 17. The mapping of a IP–protocol header onto a chromosome with 8–bit gene width.

A minimalistic testing environment was set up to simulate EthGA, as shown in Figure 18. PC1 runs EthGA and transmits broadcast IP–frames constantly over the network, which are observed by PC2. EthGA managed to transmit IP–frames via the WinPCap library to the network interface card (NIC), which in turn passed the received frames onto the physical layer. However the NIC did not always forward the frames, if the frame did not follow the IP–standard the frame was dropped, disabling the possibility of sending malformed frames over the network. This limitation proves that a computer would not be suitable for this thesis, since the DUT must be tested over the widest range of frame–types which is not achievable with a NIC. Therefore the EthGA software is implemented on the NIOS microprocessor on the DE4 board, which can send malformed frames through the FrameStormer component (recall from Figure 13). It is also mentionable that the NIC also have other limitations, that can drop certain frames if they consist of malformed high–level protocol header. In this simulation, no feedback was obtained from the system, and therefore the EthGA simulated the feedback using the all–ones problem.

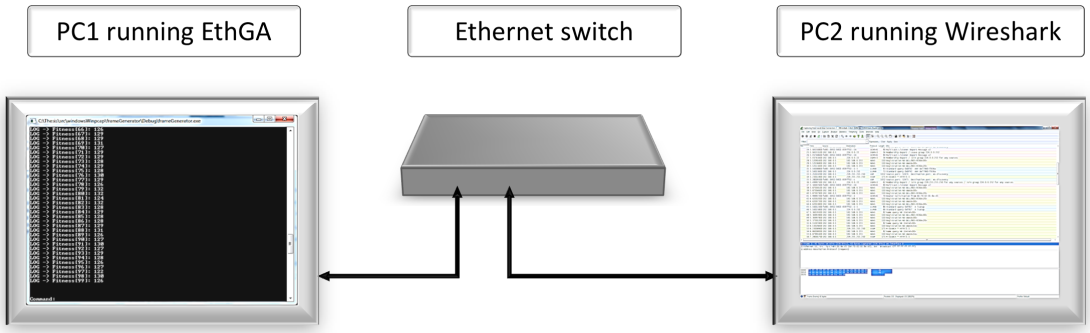


Figure 18. Minimalistic test environment of the simulation phase, where PC1 runs EthGA and transmits broadcast MAC frames over the network, which are received and observed by PC2 with Wireshark.

The user interface (UI) of the EthGA consists of a console–based interface, where the user can enter commands and set parameters of the GA. The user can also choose if he/she wants to print out every individual generation or merely the fitness results at the end of the simulation. The PC version of the EthGA prints directly to the Windows console view, whereas the properties of the FPGA version are presented more thoroughly in the next section.

5.2. Implementation of EthGA on a FPGA/NIOS microprocessor

The FPGA version of EthGA does not differ vastly from the PC version, but still there are some limitations that must be taken into account. Due to the small amount of on–chip memory available on EtherTester (307,200 bytes), the maximum population size in NIOS EthGA must be limited to 30 individuals, otherwise the memory management would fail and the processor would reboot itself. This maximum population size was at first even smaller, but the author managed to increase it to 30 by implementing more advanced memory management, such as finding an optimum between statically initialized memory and dynamically initialized memory.

The advantages of running EthGA in the FPGA instead of a PC computer does overtake the limitations, since EthGA is now able to transmit any kind of Ethernet frame disregarding its contents. This is a result from not having a NIC, and instead using an Ethernet IP–core in co–operation with FrameStormer. Another advantage is the increase of the Ethernet frame transmission rate due to the hardware implementation, which does not need to use any software drivers or libraries which might slow down the transmission rate as the PC version did.

The final implementation of EthGA in the FPGA is shown in Figure 19, where the most important interfaces are presented. EthGA is in this implementation only able to transmit Ethernet frames via Ethernet port 0 (Eth0). The user can send commands and monitor its execution through the USB–based JTAG interface from a console. EthGA can also communicate with an external device through the RS–232 interface. The usage of the external device is explained more thoroughly in Chapter 6 where the test environment is presented. A small simulation environment was also set up as shown in Figure 20 to prove that the configuration works. A user manual for EthGA is presented in Appendix 1, and the EthGA parameters are explained in Appendix 2.

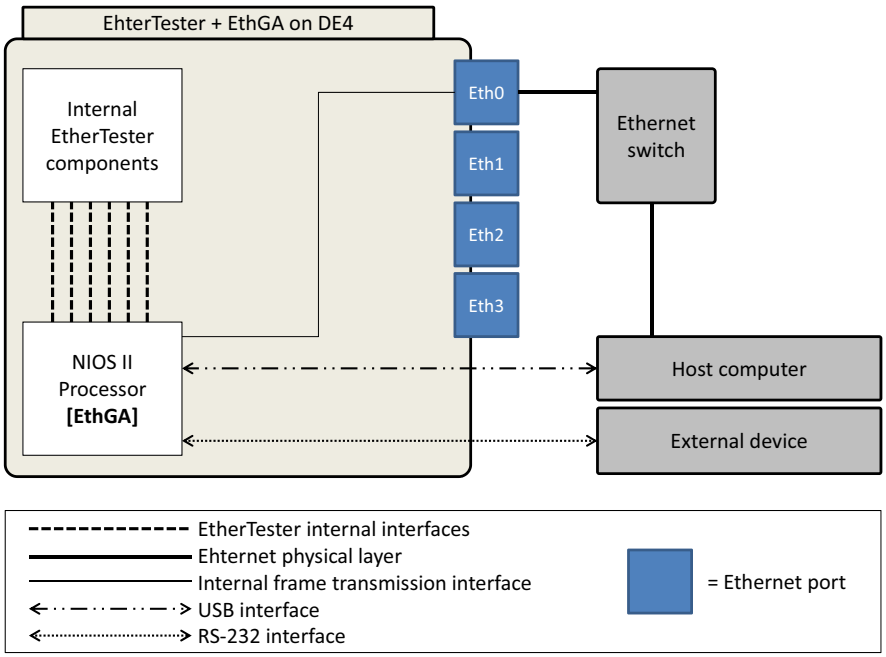


Figure 19. Internal structure of EthGA running on DE4 using Rauhala's EtherTester framework. The user can send commands to EthGA trough the host computer to e.g. enable the operation of EthGA as listed in Figure 16. All frames generated in EthGA are transmitted trough interface Eth0 of the DE4.

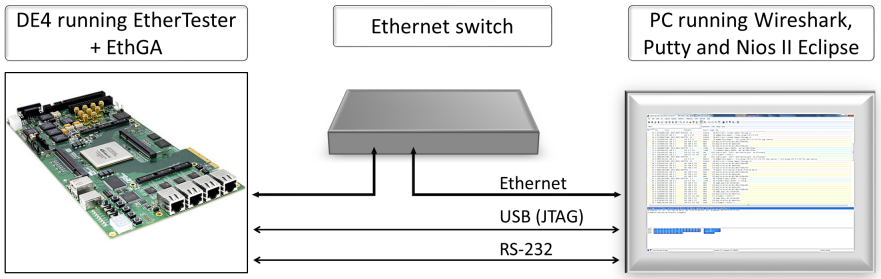


Figure 20. Extended test environment of the simulation phase for EthGA on DE4, where the PC controls and observes the functionality of the EthGA. Wireshark is used to monitor the network traffic, putty to control/monitor the RS–232 traffic and Nios II Eclipse IDE to control/monitor the functionality of EthGA.

6. CASE STUDY

The case studies in this chapter presents the tests performed for validating the functionality of a device under test (the DUT). In Section 6.1 the preliminary system test case is presented, where the basic functionality of the system is analysed to ensure a working test environment. Section 6.2 on the other hand analyses the limits of the system by investigating the transmission speeds of both the interfering packets from EtherTester and the communication between the Raspberry PI (RPI) and the DUT. With the knowledge obtained from case 1 and 2 the testing proceeds to case 3 presented in Section 6.3 where the message exchange time between the RPI and the DUT is analysed while transmitting interfering packets with the EtherTester without any GA operations. Case 4 and 5 in Section 6.4 and 6.5 analyses the effect of packet structure in relation to the message exchange time, where normalization test are executed in case 4 to create a foundation for the GA enabled tests in case 5.

In order to verify the functionality of the DUT, a test environment shown in Figure 21 is constructed. The main idea of this testing environment is that the RPI is used to communicate with the protection relay (i.e. the DUT), and the EtherTester is used to transmit Ethernet packets to the DUT in order to interfere with its communication with the RPI. The RPI incorporates the Gosling library which is explained in more detail further in this chapter. The PC is used to control/monitor the EtherTester through JTAG interface but also to occasionally monitor the network traffic with Wireshark. The general procedure of events with the GAs enabled in the test environment goes as follows:

1. EtherTester initializes a constant packet transmission of interfering packets to the DUT, the transmission rate of the interfering packets is adjustable by modifying the ICBF (Idle cycles between frames) parameter.
2. EtherTester (EthGA) sends a start command via RS–232 to the RPI.
3. RPI (Gosling) transmits n packets to the DUT and receives a response for every packet.

4. RPI (Gosling) calculates the total transmission time of n packets.
5. RPI (Gosling) transmits the result via RS–232 to EtherTester.
6. EtherTester (EthGA) saves the received fitness result and stops transmission.
7. EtherTester (EthGA) moves to the next frame in the population and goes back to stage 1. When the population is run through GA operations are performed on selected genes within the individuals.
8. The procedure continues until the DUT has stopped functioning or the procedure is manually interrupted.

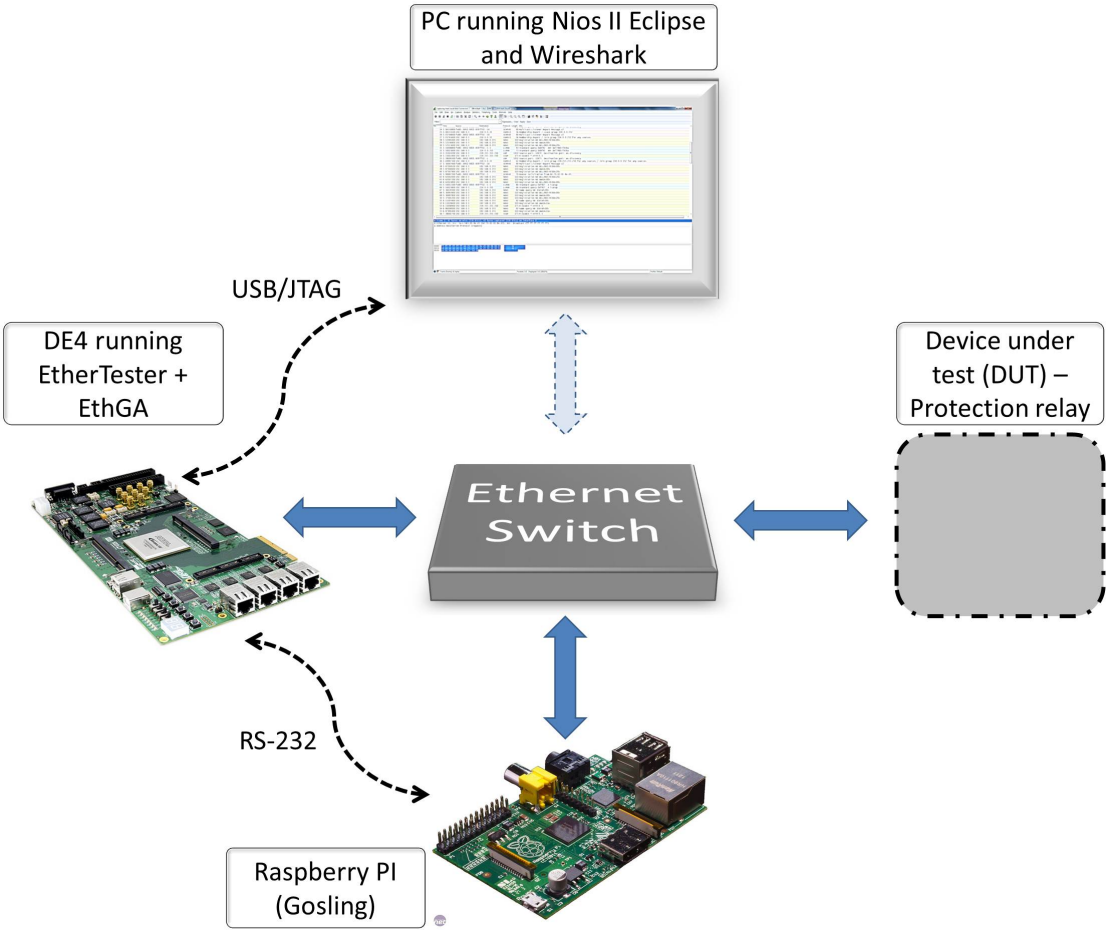


Figure 21. General testing environment, where all devices are connected to an Ethernet switch, and the DE4 board is communicating with the host PC through USB/JTAG and with the RPI trough the RS–232 interface.

As earlier mentioned an FPGA (namely Stratix IV on DE4) is being used to have complete control over the transmitted interfering packets. This set–up is preferred instead of a computer since the system would in that case be depending on third–party drivers and NICs, resulting in less control and also possible filtering of the Ethernet packets. The EtherTester software does no filtering at all and sends exactly the Ethernet packet one tells it to. A big drawback with this testing environment is the RPI, which uses the slow RS–232 interface to communicate with the EtherTester and also has a limited amount of resources compared to a powerful computer. A more appropriate approach would be to implement the functionality of the RPI into the EtherTester, and use hardware accelerators to improve calculation and parsing efficiency. However this proposal was not implemented due to the short time range of this thesis, but is highly recommended as future work. The RPI uses the Gosling library which was developed by Mika Ruohonen to enable a two–way communication link between the RPI and the DUT protection relay. The the DUT was configured by Mika Ruohonen to be able to communicate with the RPI and is in these tests considered as a black box, i.e. the inner functionality of the DUT is not known to the EtherTester.

Another drawback in the testing environment is also the Ethernet switch, since it is responsible for maintaining all network communication in the test set–up. If the switch is not fast enough and cannot handle a heavy load of Ethernet traffic, the communication might be interrupted or malformed packets could occur. However the switch used in the testing environment is a 1 Gb LINKSYS SR2016 switch that routes all incoming packets to every active port on the switch. The switch enables the store–and–forward feature and implements flow control as specified in the IEEE 802.3x standard. (Linksys 2006)

From this chapter and onwards the packets that the EtherTester transmits will always be referred to as *packets* or *interfering packets*, and the packets that the RPI transmits/receives will always be referred to as *messages* or *GOOSE messages*. A GOOSE message is an Ethernet frame that incorporates the GOOSE protocol as specified in the IEC 61850-1 standard, refer to section 2.1.2 for more information. Still to recall the name *EtherTester* refers to the basic Ethernet packet transmission feature of the hardware/software imple-

mented on the DE4, and the name *EthGA* refers to the GA software embedded in the EtherTester, as shown in figure 19.

The most common parameters that are altered in the preceding tests are listed below:

- Idle cycles between frames (ICBF)
- Gosling message exchange count (GMEC)
- Interference level
  - *Interference level 0*: No interfering packets are transmitted (i.e. EtherTester is not used). No GA operations in use.
  - *Interference level 1*: MAC destination address of interfering packets is a unicast address other than the physical address of the DUT. Constant Ethernet frames.
  - *Interference level 2*: MAC destination address of interfering packets is the physical address of the DUT. Constant Ethernet frames.
  - *Interference level 3*: MAC destination address of interfering packets is the physical address of the DUT. GA optimized Ethernet frames.

The ICBF property is a feature of EtherTester that lets the user define how many clock cycles the EtherTester should be idle between the transmission of packets. The actual transmission rate in packets per second is dependent on this parameter but also on the packet length and the clock frequency at which the EtherTester is running. The clock frequency of the EtherTester was set to 100 MHz. The payload length of the interfering packets is 106 octets in every test, resulting in a packet length equal to the messages that the RPI uses to communicate with the DUT. The GMEC parameter defines the number of messages that the RPI uses to communicate with the DUT before reporting the final message exchange time to the EtherTester.

6.1. Case 1: Preliminary system test

The first test executed analysed the behaviour of the whole system environment to ensure that it is functioning according to the specifications given in the previous section. The EtherTester was used to transmit bursts of packets and no GA–operations were applied. Already at this point some bugs where discovered in the Gosling library running on the RPI. The RPI managed to communicate with the DUT when the transmission rate of the interfering packets where low, leading to zero dropped messages. However as the transmission rate of EtherTester increased, the Gosling library will got of sync with the DUT due to the large amount of dropped messages. The Gosling library could not resume communication even when the transmission of the interfering packets was stopped. However the bug in the Gosling library was found by its creator and fixed within a short amount of time. These preliminary results do indeed show that the EtherTester is capable of finding faults in hardware communicating with the IEC 61850 standard, even if the fault was not located in the DUT. With the system fully functioning the tests moved on to the next case where speed tests were performed as presented in the next section.

6.2. Case 2: Speed test

In this case the EtherTester and the RPI were tested separately in order to clarify their behaviour in the system with interference level 0. The first phase was to determine the transmission rate of EtherTester since there is only one option to adjust, the ICBF parameter and not the actual throughput. However there are functions in the EtherTester for analysing the throughput which was used in this test. These rate analysis functions are hardware implemented and performed by analysing the time needed (in clock pulses) for sending a given number of packets. But the activation of the rate analysis is software based along with the preceding calculations which might lead to small measurement errors. The test results are shown in Figure 22 where the relationship between ICBF and actual throughput in megabits per second (Mb/s) is presented. A packet with 106 bytes long payload was transmitted 100,000 times and the time lapsed was measured to calcu-

late the throughput. The results are shown in table 4.

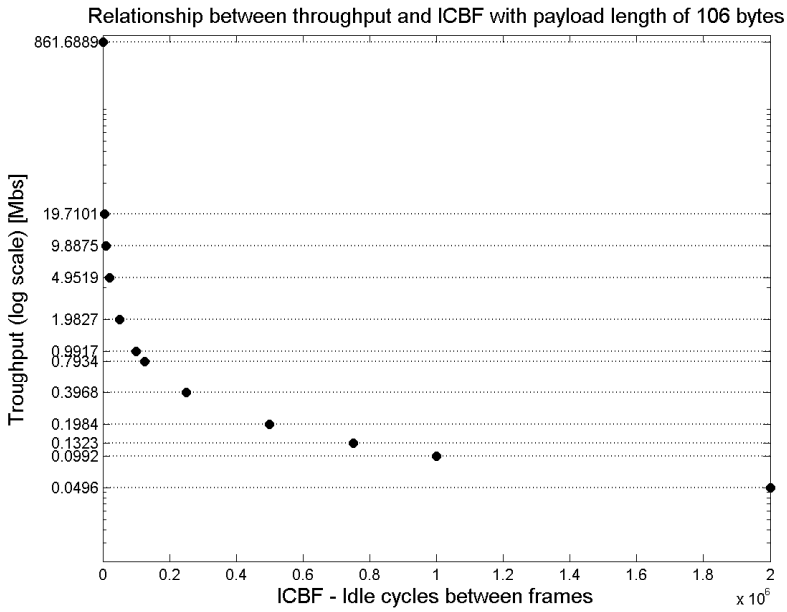


Figure 22. EtherTester speed test showing the relationship between throughput and ICBF. The packet consisted of a 106 byte payload and was transmitted 100,000 times from where an average throughput was calculated.

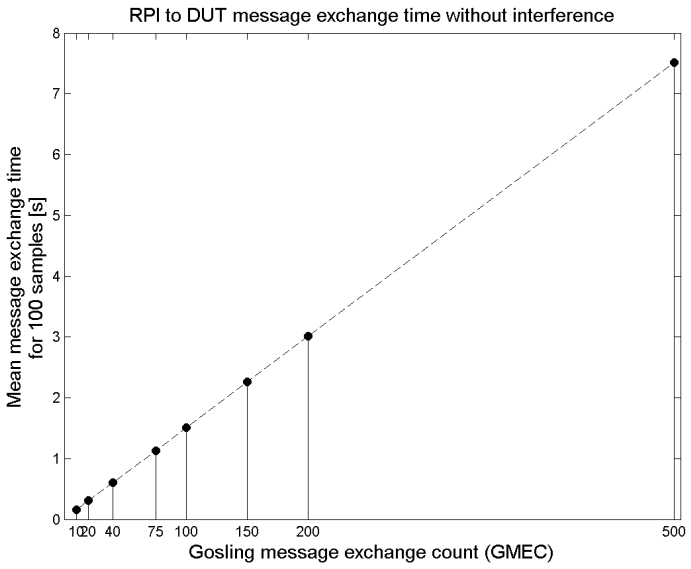


Figure 23. Message exchange time between RPI and the DUT, where the gosling message exchange count (GMEC) values was averaged over 100 samples to analyse the behaviour of the system in its optimal environment (i.e. interference level 0).

Table 4. Relationship between the ICBF parameter, throughput and transmission rate for a payload length of 106 bytes for the EtherTester.

ICBF	Throughput [b/s]	Transmission rate [packets/s]
2000000	5 k	49
1000000	99 k	99
750000	0.13 M	133
500000	0.20 M	199
250000	0.40 M	399
125000	0.80 M	799
100000	1.0 M	999
50000	1.98 M	1998
20000	4.95 M	4991
10000	9.89 M	9967
5000	19.71 M	19869
0	861.69 M	868637

The second phase of this test was to analyse the average message exchange count time between the RPI and the DUT without any interference from the EtherTester. Figure 23 shows that the message exchange time increases linearly with the message count concluding that the system is very stable in its optimal environment.

6.3. Case 3: Burst test

The aim of this burst test was to analyse the behaviour of the system by flooding the switch with Ethernet packets and observing possible fluctuations in the message exchange time between the DUT and the RPI. The results will be compared to the message exchange time without any interference from the EtherTester.

The Ethernet packets all share the same structure and consists only of MAC–frames,

where the type/length field specifies the length of the payload. The payload consisted of ascending numbers starting from zero. This type of MAC–frame was used since it does not belong to any protocol that the DUT in configured to handle. Also the destination MAC address of the interfering packets will be a unicast address other than the physical address of the DUT. This test do not use packet optimisation and will only utilize the basic functionality of EtherTester such as packet transmission at different speeds.

As shown in Figure 25, the EtherTester does not seem to interfere with the DUT at all when the interference is level 1 and GMEC = 10, concluding that the switch is not overloaded and that the DUT ignores these interfering packets since the destination MAC address is wrong. However, when approaching throughput levels between 5 and 10 Mb/s (ICBF between 20,000 and 10,000) the DUT started reporting errors and occasionally restarted itself, and when exceeding the 10 Mb/s troughput (ICBF below 10,000) the DUT failed every time and restarted itself, as shown in Figure 24. To recall, this pattern was only observed when the MAC destination address of the interfering frames is different from the physical address of the DUT, and also only when using EtherTester and RPI simultaneously. Several attempts trying to cause the same error with only EtherTester was executed, showing no errors at all. The reason for this error is unknown since the DUT should not handle packets that are not intended to it, but still the speed of the DUT's Ethernet port is 10 Mb/s and when reaching this limit with the interfering packets it was prone to fail.

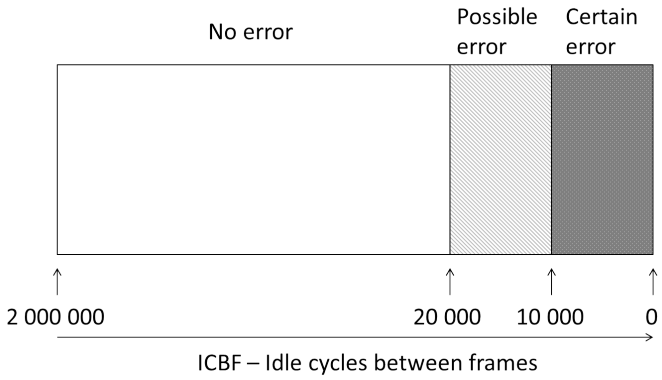


Figure 24. DUT burst test error levels with interference level 1 where the relay operates flawlessly when the ICBF parameter is higher than 20,000 (troughput > 5 Mb/s), but implies an unknown probability of failure with ICBF between 20,000 and 10,000. Eventually the DUT will fail every time ICBF is below 10,000 i.e. when throughput exceeds 10 Mb/s.

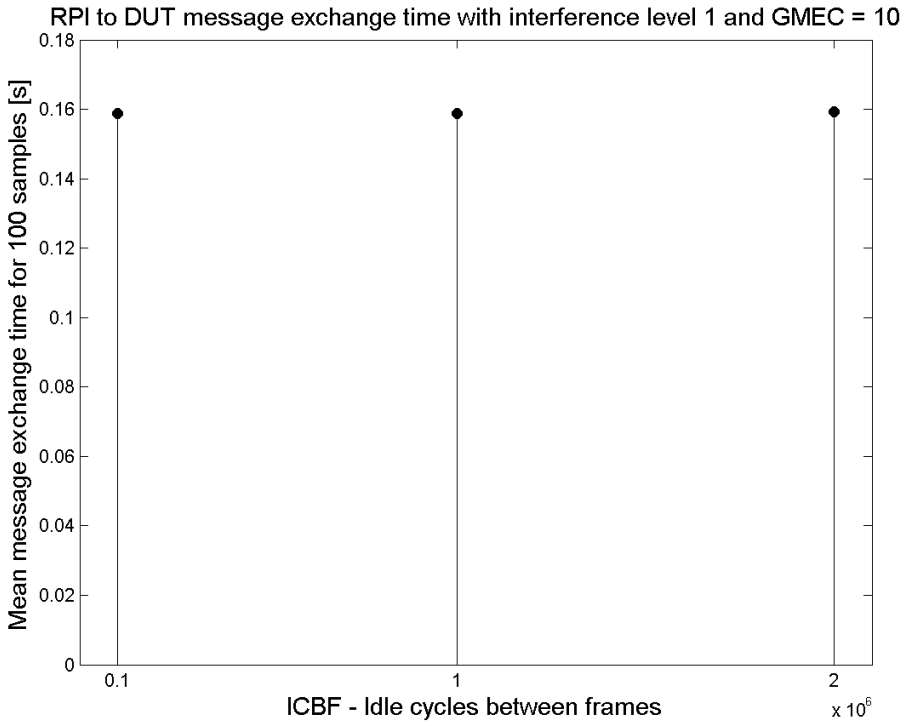


Figure 25. RPI to DUT message exchange time with interference level 1 and GMEC = 10. The interfering packets did not affect the DUT at all when ICBF was above 100,000 i.e. when throughput is below 1 Mb/s.

In addition to the DUT rebooting itself, it showed a behaviour of either enabling safe mode or started functioning normally after a maximum of 13 restarts. Two tests were performed ten times in the *certain error range* (ICBF below 10,000 as shown in Figure 24) with ICBF = 0 presented in Figure 26 and ICBF = 5,000 presented in 27. These results show that there is nearly no difference in the ICBF parameter as long as it is in a certain error range regarding the final stage of the DUT. ICBF = 0 showed a 70 % total failure and ICBF = 5,000 showed a 60 % total failure. A total failure means that the DUT activates safe mode which cuts all communication through the Ethernet and sets the DUT in a passive state. However, the number of samples is relatively small, and should be increased in future work to ensure more reliable statistical results.

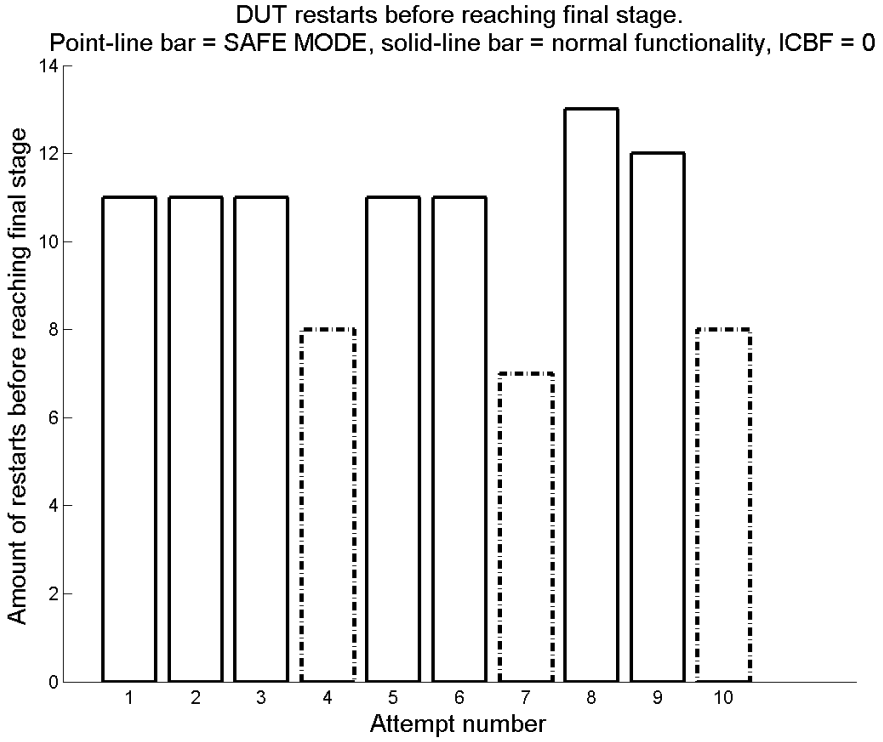


Figure 26. DUT restarts before reaching the final stage with ICBF = 0. A total of 70% of 10 samples indicated safe mode as the final stage. A point–line bar represents safe mode and a solid–line bar represents normal functioning.

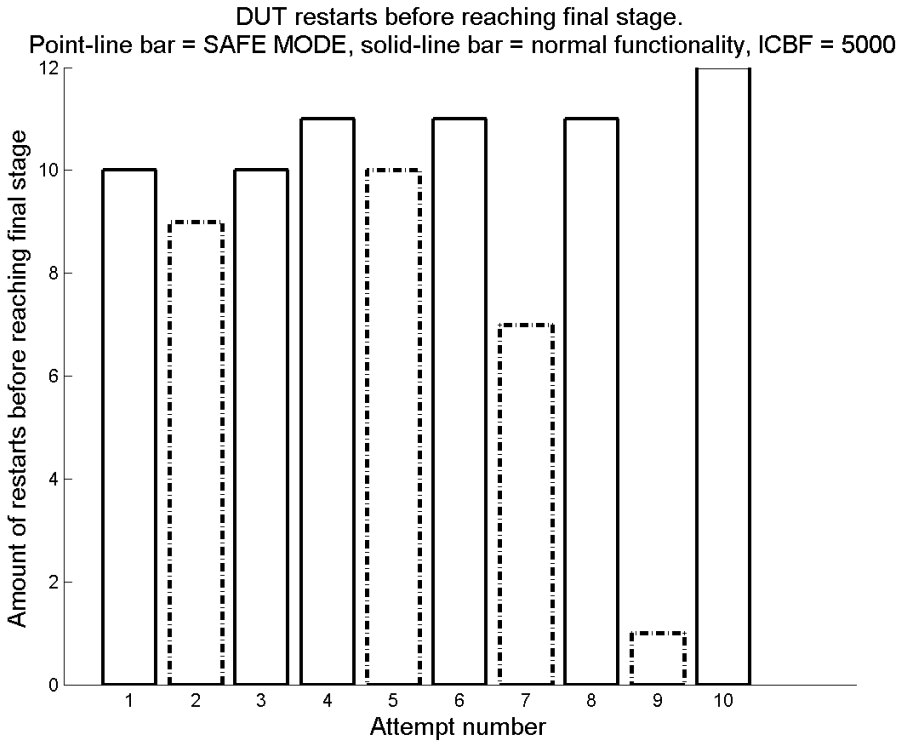


Figure 27. DUT restarts before reaching the final stage with ICBF = 5,000. A total of 60% of 10 samples indicated safe mode as the final stage. A point–line bar represents safe mode and a solid–line bar represents normal functioning.

The results above proves that the developed EtherTester is capable of disabling the functionality of the DUT when it is communicating with the GOOSE protocol in an simulated industry Ethernet network. By having the knowledge obtained that the DUT is not an fail–safe device the tests proceeded to case 4 and 5 where the effect of packet structure in relation to the message exchange time is analysed.

6.4. Case 4: Normalisation test

In order to analyse the effect of packet structure in relation to message exchange time, one need to define normals for the next test to use in comparison when analysing the results. The purpose of this test is to transmit a single interfering packet continuously and analyse

the statistical properties of the results. The single interfering packet is a standard Ethernet LLC packet where the destination MAC address is the physical address of the DUT (i.e. interference level 2). There will also not be any GA operations applied and the results from this test are used to compare with the following tests to analyse whether or not the GAs will have any impact on the message exchange time between DUT and RPI.

Figure 28 shows the mean (top graph) and median (bottom graph) message exchange time for 200 packets with interference level 2, where larger mean peaks occur when ICBF is 1,000,000, 500,000 and 250,000. However the median peaks does only occur when ICBF is 500,000 and 50,000. This is a very interesting phenomena since rational analysis of the testing scenario would imply that the message exchange time should increase for every step the ICBF parameter decreases. By observing the individual outlying tests one can notice oscillation in the message exchange time that does not follow a repeating pattern, as shown on the left graph in Figure 29 where ICBF = 1,000,000. However for the lower message exchange time it oscillates more evenly, as shown on the right graph in Figure 29 where ICBF = 1,062,500, observe that the Y–scale of the left graph in Figure 29 is about 20 times larger than in the right graph. It is still mentionable that when ICBF reached 100,000 the mean message exchange time was above 2 hours for 10 messages, and when ICBF reached 50,000 the communication between the DUT and the RPI eventually stopped resulting in zero transmitted messages.

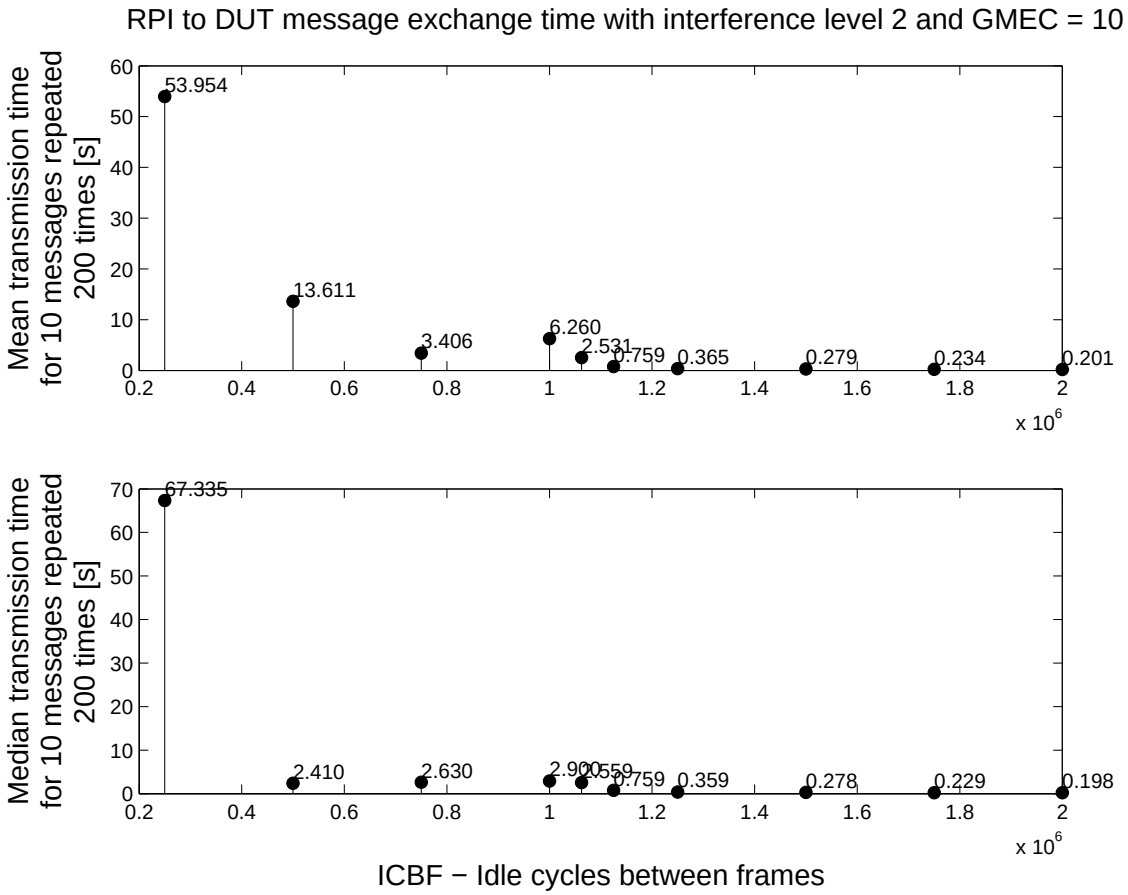
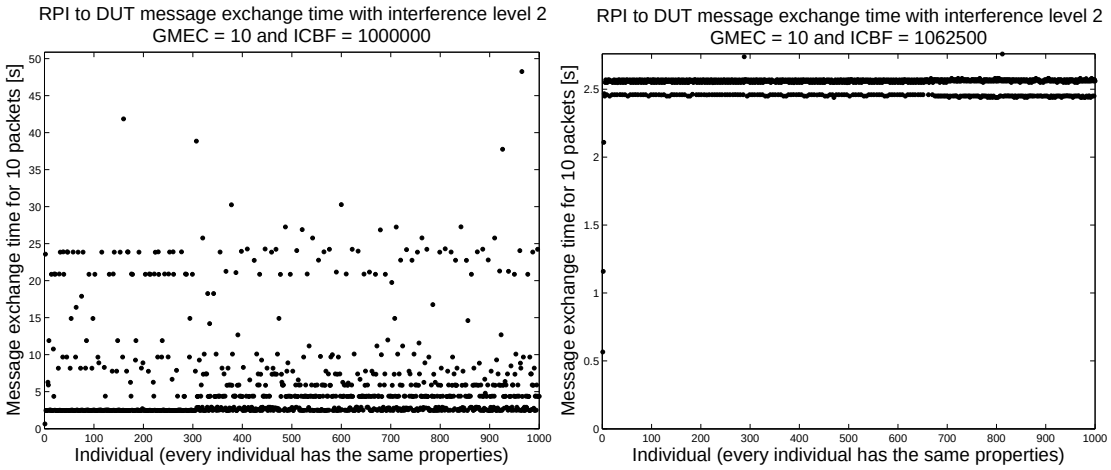


Figure 28. Mean (top graph) and median (bottom graph) message exchange time for 200 packets with interference level 2 where GMEC = 10. Outlying peaks occur when ICBF = {1,000,000, 500,000, 250,000}.

It can also be observed that by merely changing the destination MAC address of the interfering packets to the physical address of the DUT (interference level 2), one can truly interfere with DUT in terms of message exchange time. There were no individuals that could reach as low average message exchange time in Figure 28 as it was observed in Figure 25, where both tests used GMEC = 10. This shows that every Ethernet frame intended to the DUT does consume its resources and therefore increase the message exchange time.



(a) **(b)**

Figure 29. Message exchange time comparison where GMEC = 10. Graph (a) shows a varying oscillation where ICBF = 1,000,000 (i.e. 99 kb/s throughput) and graph (b) shows a more stable oscillation where ICBF = 1,062,500. The y–scale in graph (a) is about 20 times larger than in graph (b).

6.5. Case 5: Effect of packet structure in relation to message exchange time

In this final test the system was used with its full potential by enabling the GA operations on EthGA and also configuring the EtherTester's transmission rate to a point where the mean transmission time between the DUT and the RPI oscillates steadily. The initial population consists of GOOSE packets similar to the ones that the RPI transmits to the DUT but to which an initial mutation is applied with a mutation rate of 7 %. It was noticed that with the mutated initial population a fair diversity in the packets occurred, which resulted in both valid and malformed GOOSE packets as observed with Wireshark. Table 5 shows the parameters for 6 different tests where ICBF is 1,062,500 in every sub–case since it was shown in Section 6.4 that 1,062,500 produces quite small oscillations. Interference level 3 is used in all the sub–cases.

Table 5. EthGA parameters used in case 5 for 6 different tests (sub–cases 5.1 to 5.6).

Sub–case ID	5.1	5.2	5.3	5.4	5.5	5.6
ICBF	1,062,500	1,062,500	1,062,500	1,062,500	1,062,500	1,062,500
GMEC	10	10	10	10	10	10
Population size	25	25	25	50	50	50
Mutation rate	0.001	0.007	0.01	0.001	0.007	0.01
Crossover type	Uniform	Uniform	Uniform	Uniform	Uniform	Uniform
Selection type	Lin.Rank	Lin.Rank	Lin.Rank	Lin.Rank	Lin.Rank	Lin.Rank
Lin.Rank.Para	1.5	1.75	1.99	1.5	1.75	1.99
Elitism	Yes	Yes	Yes	Yes	Yes	Yes
Affected genes	0–120	10–120	63–120	0–120	10–120	63–120

The parameters in Table 5 are altered so that EthGA will cover the most important areas. When the EthGA was simulated on a PC–computer in section 5.1 it was noticed that the uniform crossover type along with linear ranking performs well in relation to tournament selection, and is therefore used. Recall from Section 3.2.3 that by increasing the linear ranking parameter s, the selection pressure rises. The elitism in EthGA works by preserving the two best individuals, and does not apply any GA operations to them. This is used to analyse whether or not the specific individual did actually affect the message exchange time, or if it was due to something else. The affected genes parameter in Table 5 describes on which genes the GA should operate, hence leaving the rest untouched. This was altered since the GOOSE protocol might contain data in the beginning of the packet, that might be vital for the DUT to process it. The mean message exchange time for 6 different tests with parameters from Table 5 is shown in Figure 30, where it can be observed that the mean message exchange time varies a little, probably due to noise. This concludes that EthGA was not able to discover any harmful packet in these tests.

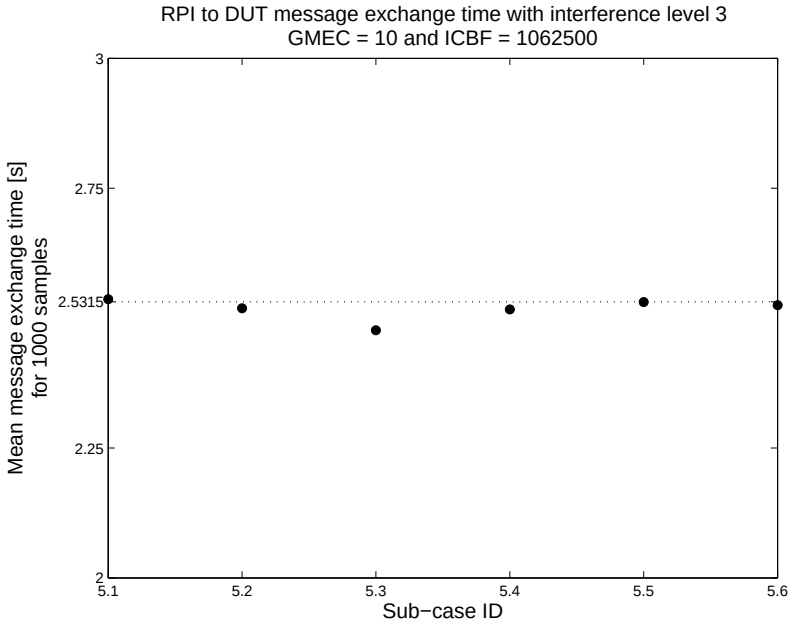


Figure 30. Mean message exchange time where GMEC = 10 and ICBF = 1,062,500 for 6 different test with the parameters listed in Table 5, showing a small variance probably due to noise.

However when altering the ICBF and the GMEC parameter some unexpected results occurred. Three more tests were performed with the parameters in Table 6. Subcase 5.7 is considered as the founding test for finding a good population, and sub–case 5.8 uses this population which is transmitted over and over again without any GA–operations, to analyse if the specific population can recreate the results. Sub–case 5.9 uses the population generated in sub–case 5.7 to improve its results.

Table 6. EthGA parameters used in case 5 for 3 different tests (sub–cases 5.7 to 5.9), where sub–case 5.8 does not use any GA operations. Sub–cases 5.8 and 5.9 uses the final population generated from sub–case 5.7.

Sub–case ID	5.7	5.8	5.9
ICBF	600,000	600,000	600,000
GMEC	20	20	20
Population size	16	16	16
Mutation rate	0.02	N/A	0.004
Crossover type	Uniform	N/A	Uniform
Selection type	Lin.Rank	N/A	Lin.Rank
Lin.Rank.Para	1.9	N/A	1.9
Elitism	Yes	N/A	Yes
Affected genes	10–120	N/A	10–120

Figure 31 shows that the message exchange time unexpectedly oscillates between either three or five different levels. The reasons for this is unknown but could be induced by the DUT switching its operation mode, or the RPI not functioning according to its specification. However, fairly high message exchange times are shown to the right in Figure 31 (where Individual > 8,500), but results could not be reproduce when transmitting the final population over and over again as in sub–case 5.8 presented in Figure 32. This concludes that the interfering packet's structure did not cause these high oscillations. The tests continued in sub–case 5.9 as shown in Figure 33 where the mutation rate was lowered. However, this particular test did not show any results of lengthening the populations message exchange time.

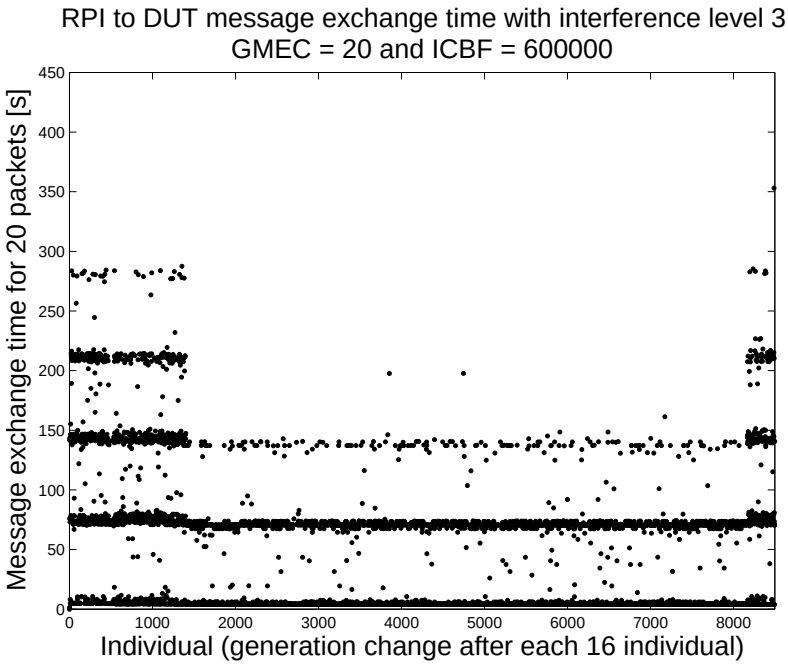


Figure 31. Message exchange time for sub–case 5.7, with parameters according to Table 6.

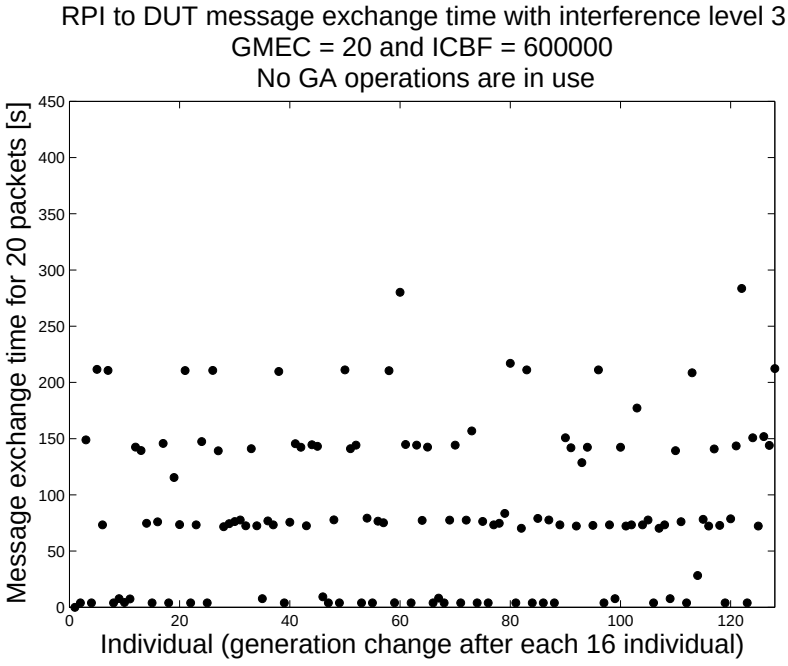


Figure 32. Message exchange time for sub–case 5.8, with parameters according to Table 6. The final population generated from sub–case 5.7 was used in this test and no GA operations were applied.

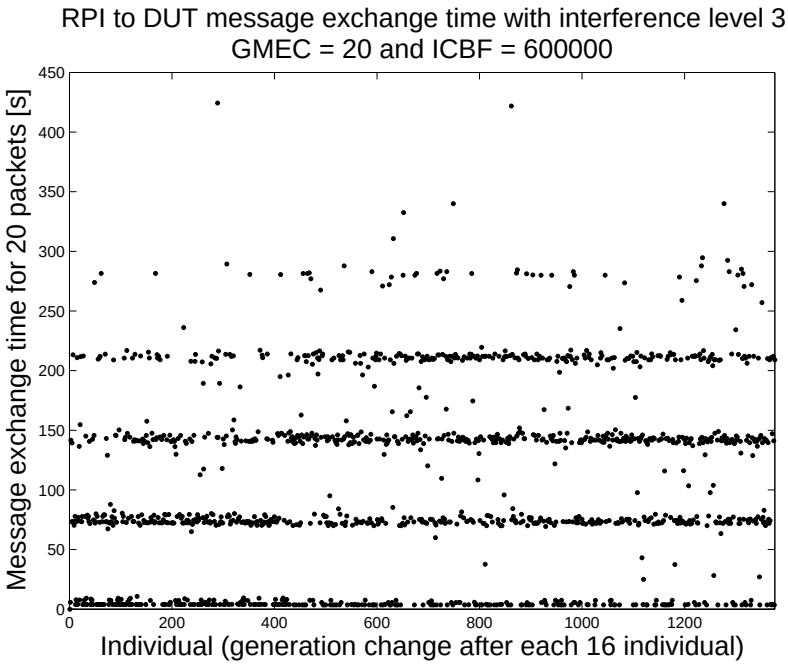


Figure 33. Message exchange time for sub–case 5.9, with parameters according to Table 6. The final population generated from sub–case 5.7 was used in this test.

6.6. Summary

The previously given results in the different cases do imply that the EtherTester (including EthGA) is a valid tool for evaluating the functionality of a DUT communicating within an industrial Ethernet network. The EtherTester was able to generate and transmit interfering packets at high speeds that either slows down the communication between the DUT and the RPI or even disabling the functionality of the DUT completely. It was also shown that the EtherTester is capable of finding bugs in software which are used when communicating with the GOOSE protocol. Even though, the Ethernet interfaces on the DE4 was of gigabit range, the maximum measured throughput achieved with ICBF = 0 was 861 Mb/s as shown in Figure 22. This is the result of using a short payload length (106 octets) which decreases the throughput. The EtherTester managed to put the DUT in error mode and causing it to restart itself if all of the following conditions were met (using interference level 1):

- The destination MAC address of the interfering packets is any other address than the physical address of the DUT.
- The ICBF parameter was set below 10,000 giving a transmission rate above 9.89 Mb/s.
- Communication between the DUT and the RPI was active when the interfering packets were transmitted.

The error mode caused the DUT to restart itself 1–13 times giving approximately a 60–70% probability of enabling safe mode on the DUT disabling its functionality completely. The safe mode is not intended for regular use and was only used during the manufacturing of the DUT for the analysis of occurring bugs. However if the ICBF parameter was set between 10,000 and 20,000 (corresponding to throughputs of 10 Mb/s and 5 Mb/s respectively) the DUT could possibly enter error mode but not always. It was also discovered that when ICBF was set above 20,000 the EtherTester did not interfere the communication between the DUT and the RPI at all when the MAC destination address was different than the physical address of the DUT. The mean message exchange time for 10 GOOSE messages between the DUT and the RPI stayed steadily at 0.16 seconds when the ICBF parameter was between 20,000 and 2,000,000 (5 Mb/s and 5 kb/s).

Eventually the EtherTester did interfere the communication between the DUT and the RPI by using interference level 2 (i.e. the MAC destination address of interfering packets equals to the physical address of the DUT) even if ICBF was set to 2,000,000 giving a transmission rate of 5 kb/s. This did raise the mean message exchange time from 0.16 seconds to 0.199 seconds. However when ICBF was set to 100,000 (throughput = 1 Mb/s) the mean message exchange time rose to above 2 hours for 10 messages, and when with ICBF = 50,000 (throughput = 2 Mb/s) there was no message exchange at all, since the flood from EtherTester did use all the capacity of the DUT.

However, it seems that the structure of the interfering packets does not affect the message exchange time, which was proven in Section 6.5. Some tests did show higher message

exchange time at certain points in the tests, but when elitism was used and the same packet was transmitted again the same message exchange time could not be achieved. This concludes that the system is not deterministic, and the structure of a single packet may not play a significant role in the variance of the message exchange time.

In terms of network failure, the results show that the EtherTester is a valid tool Denial–of–Service (DoS) attacks for the specific DUT used. Whereas the most important parameters that affect this is the MAC destination address of the interfering packets along with their transmission rate. The structure of the payload does not seem to have any effect on the failure of the DUT. However this matter should be investigated further in order to verify this claim.

7. CONCLUSION

The case studies of this thesis show that the developed EtherTester and EthGA software in the TehoFPGA project, are valid tools for validating the functionality of protection and control devices in substation automation systems, communicating with the GOOSE protocol derived from the IEC 61850 standard. The EtherTester managed to break the operation of a DUT in the testing environment merely by transmitting interfering Ethernet packets at high speeds (above 5–10 Mb/s) while the DUT was communicating with a Raspberry PI. However this only occurred when the MAC destination address of the interfering packets was any other address than the physical address of the DUT. The DUT managed to restart itself 1–13 times, showing a probability of 60–70 % that enables safe mode disabling its functionality completely. When using this set–up the EtherTester did not interfere with the DUT at all when using transmission speeds lower than 5 Mb/s.

On the contrary the EtherTester did interfere with the DUT when the MAC destination address of the interfering packets was the same as the physical address of the DUT, regardless of the interfering packets transmission speed. However the interference consisted only of an increase in the message exchange time between the DUT and the Raspberry PI, where communication was no longer achievable when the transmission speed of the interfering packets exceeded 1.98 Mb/s. But still the communication was resumed when disabling the transmission of the interfering packets, concluding that it is not possible to disable the DUT completely with this setup.

Furthermore the results show that the structure of the interfering packets may not play a significant role in terms of interference for the specific DUT used. A genetic algorithm was used to search for a specific packet structure that might interfere with the DUT, but no outstanding individuals were found that did increase the message exchange time between the DUT and the Raspberry PI. However, this matter should be investigated further, and possible improvements are discussed and presented in the next chapter.

8. DISCUSSION

As it has been proved that the EtherTester and EthGA are suitable tools for network testing, this thesis only focused on one specific DUT meaning that it may or may not work on other DUTs as well. However, the results are very clear in terms of disabling the DUT's functionality so the EtherTester can find bugs in devices without any doubt, as it was also verified in Section 6.1 where a bug was discovered in the Gosling library on the RPI. But still further test should be run and improvements should be made in order to consider EtherTester and EthGA as a fully functional network testing tool.

There are some implications on EtherTester not being able to reach the 1 Gb/s packet transmission speed as shown in Section 6.2 where the maximum transmission speed encountered was 0.861 Gb/s. The source of this problem might come from the fact that the packets used for transmission had only a 106 byte payload, resulting more gaps between the packets as defined in the Ethernet standard.

Another difficulty in the test cases was the measurement of the message exchange time with interference level 2 (see Section 6.4), where the message exchange time did not increase with every step that the ICBF decreased, as common sense would imply. One cause to this problem might be that the DUT is operating in cycles, and the packets managed to interfere at an exact state within the cycles causing the message exchange time to oscillate uncontrollably, leading to an increase in the message exchange time with ICBF = 1,000,000 (Ruohonen 2014). As for ICBF = 1,062,500 the oscillation was quite stable and also the mean and median message exchange time were fairly low.

Regarding calculation speeds on the EtherTester, the FPGA gave more than enough computational power needed for processing the genetic algorithm (GA). There was at first doubts that the GA could not keep up since it was written in C and ran on the Nios II processor and needed to be implemented in hardware instead. This would have been the case if the evaluation of the fitness function would have been faster, i.e. calculating and transmitting the message exchange time to the EtherTester from the RPI. If the fitness

evaluation algorithm would be improved (possibly by implementing it on the EtherTester), there might be a strong need to implement the GA on the hardware instead of the Nios II processor. This would imply faster testing enabling more sophisticated tests.

Also it would be useful to understand what is happening inside the DUT since the only parameter that is observed is the message exchange time, which does not tell everything. If one could access the internal software of the DUT the results could be more reliable and the testing would give more precise results.

8.1. Future work

As the EtherTester is still a prototype, it should be improved in the future in order to enable more sophisticated and faster testing. A list of proposed improvements follows:

- Direct modification of transmission rate parameter in Mb/s instead of ICBF.
- Data logging on flash drive instead of showing results in the Eclipse Nios II console view.
- Improved graphical user interface (GUI) with external touch screen.
- The functionality of the RPI embedded directly into the EtherTester for faster fitness evaluation.
- Pattern recognition / clustering algorithm to work in co–operation with the GA.

For convenience only the ICBF parameter could be changed to a transmission rate parameter in Mb/s, however the transmit rate is dependent on the ICBF parameter, the clock frequency of the EtherTester and also the packet length which implies several calculation steps from Mb/s to ICBF. At the moment the EtherTester only manages to print the obtained data on the Nios II console view, making it vulnerable to system failure on either the host computer or on the EtherTester itself, resulting in a loss of the collected data. With data logging on a flash drive the collected data would always be available disregarding any kind of system failure. This could also be extended with an external touch screen

negating the use of a host computer for easy access and increasing the mobility of the EtherTester when moving it between different testing environments.

In addition to usability improvements, there are also two major modifications that could enable more efficient and reliable testing. By embedding the functionality of the RPI (Gosling library) into the EtherTester, one could use hardware accelerators to improve the GOOSE communication between the RPI and the DUT, giving the possibility of performing more tests in less time. This would also eliminate the need for a serial connection between the RPI and the DUT, which is slowing down the response time of the system.

Also as the GAs implemented showed that the structure of a single packet does not seem to have any effect on the message exchange time in this case, one could combine it with a pattern recognition or clustering algorithm, or even use more sophisticated data mining, which would analyse if the order of certain packets does affect the message exchange time. However as any harmful packet sequences are not guaranteed to be found, it is still worth the effort to be able to verify the functionality of the DUT.

Regarding the normalisation and speed test, further testing should be run to get more precise results. There is one important aspect that has not been covered in this thesis, namely the impact of packet length on message exchange time and transmission rate. One would need to analyse the impact from the whole packet range (1 byte payload packet to 1600+ byte packet) to observe possible differences in the message exchange time, it might be that the DUT is responding differently with a change in the packet length.

REFERENCES

Alander, Jarmo (1998). *Geneettisten algoritmien mahdollisuudet*. Teknologian kehittämiskeskus. ISBN 951-53-1392-9.

Alander, Jarmo (2012). *TehoFPGA project plan*. Faculty of Technology, Electrical Engineering and Energy Technology, University of Vaasa.

Beasley, David, David R. Bull & Ralph R. Martin (1993). *An Overview of Genetic Algorithms: Part 2, Research Topics*. *University Computing* 15, 170–181.

Chambers, Lance (2001). *The Practical Handbook of Genetic Algorithms*. Second edition edition. Chapman & Hall/CRC. ISBN 1-58488-2409-9.

Darwin, Charles (1859). *On the Origin of Species by Means of Natural Selection*. Murray.

Deschamps, Jean-Pierre, Gery J. A. Bioul & Gustavo D. Sutter (2006). *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*. Wiley-Interscience. ISBN 0471687839.

Edwards, James & Richard Bramante (2009). *Networking : OSI, TCP/IP, LANs, MANs, WANs, Implementation, Management, and Maintenance*. Wiley. ISBN 9780470402382.

Eiben, Agoston E. & James E. Smith (2007). *Introduction to Evolutionary Computing*. Springer. ISBN 3-540-40184-9.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimisation, and Machine Learning*. Addison-Wesley Publishing Company INC. ISBN 0-201-15767-5.

Gong, Ren Hui, Mohammad Zulkernine & Purang Abolmaesumi (2005). *A software implementation of a genetic algorithm based approach to network intrusion detection*. In: *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference.*

Grout, Ian (2008). *Digital Systems Design with FPGAs and CPLDs*. Newnes Newton. ISBN 9780750683975.

Hor, Chinglai, Peter A. Crossley & Dean L. Millar (2007). *Application of Genetic Algorithm and Rough Set Theory for Knowledge Extraction*. In: *Power Tech, 2007 IEEE Lausanne*.

IEC 61850-1 (2003). *Communication networks and systems in substations – Part 1: Introduction and overview,* 1^{st} edition.

IEEE (2013). *IEEE Public EtherType list* [online]. [Cited 11.11.2013]. Available from World Wide Web: <URL: http://standards.ieee.org/develop/regauth/ethertype/eth.txt>.

IEEE 802.1q:2011 (2011). *IEEE Standard for Local and metropolitan area networks — Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks*.

IEEE 802.3:2012 (2012). *IEEE Standard for Ethernet*. [Cited 19.11.2013]. Available from World Wide Web: <URL: http://standards.ieee.org/about/get/802/802.3.html>.

Jaakohuhta, Hannu (2005). *Lähiverkot - Ethernet*. IT Press. ISBN 951-826-787-1.

Jong, Kenneth A. De (2006). *Evolutionary Computation, a unified approach*. The MIT Press. ISBN 0-262-04194-4.

Koza, John R. (1992). *Genetic Programming, On the Programming of Computers by Means of Natural Selection*. The MIT Press. ISBN 0-262-11170-5.

Kriger, C., S. Behardien & J. Retonda-Modiya (2013). *A Detailed Analysis of the GOOSE Message Structure in an IEC 61850 Standard-Based Substation Automation System*. *INT J COMPUT COMMUN* 8, 708–721.

Kuffel, Rick, Dean Ouellette & Paul Forsyth (2010). *Real Time Simulation and Testing Using IEC 61850*. In: *Modern Electric Power Systems (MEPS), 2010 Proceedings of the International Symposium*.

Li, Wei (2004). *Using Genetic Algorithm for Network Intrusion Detection*. Department of Computer Science and Engineering, Mississippi State University.

Linksys (2006). Wire-speed switching and Linksys reliability in a 16-port Gigabit switch. [Cited 6.3.2014]. Available from World Wide Web: <URL: http://www.andovercg.com/datasheets/linksys-sr2016-switch.pdf>.

Mackiewicz, R.E. (2006). *Overview of IEC 61850 and Benefits*. In: *Power Systems Conference and Exposition, 2006. PSCE '06. 2006 IEEE PES*, pp. 623–630.

Mirzal, Andri, Shinichiro Yoshii & Masashi Furukawa (u.d.). *PID Parameters Optimization by Using Genetic Algorithm*. Graduate School of Information Science and Technology, Hokkaido University Sapporo, Japan.

Mitchell, Melanie (1998). *An introduction to Genetic Algorithms*. The MIT press. ISBN 951-53-1392-9.

Rauhala, Olli (2013a). *Ethernet Testeri Käyttöohje*.

Rauhala, Olli (2013b). *Ethernet Testeri –Pilottiprojektin suunnitelma*.

RFC: 791 (1981). *Internet protocol - Darpa internet program protocol specification*. [Cited 25.11.2013]. Available from World Wide Web: <URL: http://www.ietf.org/rfc/rfc791.txt>.

RFC: 793 (1981). *Transmission control protcol - Darpa internet program protocol specification*. [Cited 25.11.2013]. Available from World Wide Web: <URL: https://www.ietf.org/rfc/rfc793.txt>.

Ruohonen, Mika (2014). E-mail conversation with M.Sc. Mika Ruohonen, Project Researcher at University of Vaasa, 8.1.2014.

Söderbacka, Christian (2013). *THE GOOSE PROTOCOL*. Master's thesis, University of Vaasa.

Spurgeon, Charles E. (2000). *Ethernet: the definitive guide*. O'Reilly & Associates, Inc. ISBN 1-56592-660-9.

Stavinov, Evgeni (2011). *100 Power Tips For FPGA Designers*. Createspace. ISBN 9781461186298.

Terasic Technologies Inc. (2010). *DE4 User Manual* [online]. [Cited 18.11.2013]. Available from World Wide Web: <URL: http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=501&FID=b1388d95cec372035605969d3b3727e4>.

Terasic Technologies Inc. (2013). *Altera Developement and Education Board 4* [online]. [Cited 15.11.2013]. Available from World Wide Web: <URL: http://www.terasic.com.tw/attachment/archive/501/image/layout_1000.jpg>.

APPENDICES

APPENDIX 1. EthGA user manual

This manual describes the general functions of EthGA, and presumes that the user has knowledge in the following areas:

- Nios II build tools for Eclipse
- Quartus Programmer
- EtherTester software
- Altera Terasic DE4 development and education board

Assuming that the user has properly set up the EtherTester project trough the tutorial *Back up manuaali Ethernet Testerille* (Rauhala 2013a) and downloaded both the hardware and EtherTester/EthGA software, the EthGA software is ready to be started. Upon entering the EtherTester software, one only need to type:

GA

to enter the EthGA software and the welcome screen in Figure 34 is showed.

```
--------------------------------------------------------------------------
Welcome to the EtherTestGA software. This software generates MAC frames
randomly or according to a specific protocol (GOOSE/SV) and transmits
them over the Ethernet network from port 0 on the DE4 board. Feedback
for each frame is obtained from an external device thorugh the RS-232
serial port. Genetic algorithms are then applied to the frames
in order to evolve a package that is most harmful for the Device Under Test
(DUT). For a list of commands type 'help'. NOTE! When you set a TYPE
parameter you must set it to a unsigned integer value, correlations shown
by typing 'help -t'
--------------------------------------------------------------------------
```

Figure 34. EthGA welcome screen.

As the user is now in the program, a help menu can be displayed with the:

help

command, as presented in Figure 35

```
EthTestGA GENERAL COMMANDS:
-----------------------------------------------------------------------
COMMAND              ACTION
help                 lists general commands
help -t              lists the available TYPE VALUES (CROSS,SEL,PRINT)
help -p              shows extended info about parameters (instructions follow)
exit                 exits EthTestGA
run -new             runs the GA with a NEW population (must be run first)
run -old             runs the GA from the OLD population
sim -new             simulates the GA with a NEW population (must be run first)
sim -old             simulates the GA from the OLD population
show -para           displays the current GA PARAMETERS
show -pop            displays the current POPULATION
show -pop -w         displays the current POPULATION for wireshark import
show -pop -c         displays the current POPULATION for c-code pasting
show -fit            displays the current FITNESS
show -fit -all       displays every FITNESS value obtained with comma separation
rs232 -run           sends the RUN  command to the Raspberry PI
rs232 -stop          sends the STOP command to the Raspberry PI
rs232 -quit          sends the QUIT command to the Raspberry PI
preset               sets all parameters to a predefined set (instructions follows)
set                  sets a parameter (instructions follows)
-----------------------------------------------------------------------
```

Figure 35. EthGA help screen.

The commands in Figure 35 are self-explanatory, but the most important ones are presented below. In order to start a test, one needs to enter

run -new

in the console. The test imitates a new population with the selected parameters, and an explanation of the parameters can be found in Appendix 2. Recall from Chapter 6 that a Raspberry PI with the Gosling library and a DUT is needed for running a test. The RPI communicates with the DUT and the total message exchange time is transmitted to EthGA for a given number of messages, as where EthGA moves to the next individual

in the population until the last individual in the population is transmitted. Thereafter EthGA performs the selected GA operations on the population and starts over from the first individual. One can now stop the test at any time by typing

a

into the console. Now the EthGA have stored it last population along with all fitness values gathered, with a max count of 10,000 values. If the user wants to continue the test but with new parameters one can first type

set

and follow the instructions to alter the selected parameter. Thereafter the user can enter

run -old

in order to continue the test with the old population. If the user would enter **run -new** again the population would be re-initialized and the old population would be lost. In order to change the destination mac address of the interfering packets, one need to alter one line of C-code. The line is located in the file **GA.c** in the method **RunGA()** where the variable is named **dest**. There are a few things to consider when using the EthGA software, since EthGA is still a prototype and not intended for commercial use:

- EthGA does not support data acquisition in any other way than printing the results onto the console view.
- All existing data is erased when the DE4 board is turned off.
- EthGA needs a tethered host in order to function.
- For every time the DE4 is rebooted, the hardware and software needs to be downloaded again.
- The maximum population size and chromosome length that can be used is defined in the file *parameters.c*.

Additionally as the software is not fully complete, below follows a brief explanation of every file that is involved in the project if the code would need to be altered:

ethcode.c

This is the main file for the whole project and is originally created by Olli Rauhala, and modified by the author. ethcode.c handles the communication between Nios II and the EtherTester hardware, and also the RS-232 communication between the EtherTester and the RPI. The EtherTester software is running in this file, and by typing **GA** in the console view, one enters the EthGA software which is located in the GUI.c file.

GUI.c

All Graphical User Interface (GUI) commands are handled in this file, and is considered at the "main" file for the EthGA software. From here the user can issue commands to EthGA, and also modify parameters and print e.g. the population or the acquired fitness values.

GUI.h

Header file for GUI.c, which includes only function declarations.

parameters.h

This file specifies the maximum size of certain parameters, such as *population size* or *chromosome length*. It is important check the memory usage after altering this file since if the values are set too large, there wont be enough memory left for the stack and the software will not run.

GA.c

This file handles all GA-calculations and contains also the method *RunGA()* which controls the testing and where the user can also modify the mac destination and source address of the interfering packets.

GA.h

Header file for GA.c, which include function declarations for GA.c but also for ethcode.c. This file also defines the structure GA_PARA in which all necessary parameters for EthGA are stored.

generics.c

Contains generic methods for memory allocation and the conversion between enums and unsigned integers.

generics.h

Header file for generics.c, which includes function declarations along with enum declarations.

APPENDIX 2. EthGA parameter explanation

The parameters that can be altered in EthGA is listed in Figure 36, whereas the explanations for parameters 0-7 is listed in Figure 37, the explanations for parameters 8-14 is listed in Figure 38 and the explanations for parameters 15-19 is listed in Figure 39. These explanations can also be showed inside the program by typing:

help -p

in the console view and following the instructions given.

```
LOG -> GA CURRENT PARAMETERS:
-------------------------------------------------
[0 ] Population size..: 25
[1 ] Chromosome length: 106
[2 ] Start gene index.: 10
[3 ] End gene index...: 106
[4 ] Use mutation.....: FALSE (0)
[5 ] Mutation rate....: 0.007000
[6 ] Elitism..........: FALSE (0)
[7 ] Max runs.........: 2
[8 ] Run forever......: TRUE (1)
[9 ] Print type.......: HEX (2)
[10] Use crossover....: FALSE (0)
[11] Crossover type...: UNIFORM (2)
[12] Crossover points.: 3
[13] Use selection....: FALSE (0)
[14] Selection type...: RANKING (2)
[15] Ranking type.....: LINEAR (0)
[16] Tournament size..: 0
[17] Rank lin param...: 1.500000
[18] Protocol type....: RANDOM_1 (1)
[19] ICBF.............: 2000000
-------------------------------------------------
```

Figure 36. EthGA parameters.

```
[0 ] POPULATION SIZE
------------------------------------------------------------------------
The amount of individuals that exists in the population. Where one
individual equals one CHROMOSOME. This value must be an EVEN number
due to the fact that the CROSSOVER operation handles 2 individuals at ones
An individual represents in this software an Ehternet MAC frame.
------------------------------------------------------------------------
[1 ] CHROMOSOME LENGTH
------------------------------------------------------------------------
The length of an individual (chromosome) in octets (bytes). All chromosomes
are of equal length.
------------------------------------------------------------------------
[2 ] START GENE INDEX
------------------------------------------------------------------------
The START position index where the GA performs its operation. This is
useful if the user only wants to alter specific genes instead of the whole
chromosome.
------------------------------------------------------------------------
[3 ] END GENE INDEX
------------------------------------------------------------------------
The END position index where the GA performs its operation. This is
useful if the user only wants to alter specific genes instead of the whole
chromosome.
------------------------------------------------------------------------
[4 ] USE MUTATION
------------------------------------------------------------------------
If true, mutation is applied. If false no mutation is applied.
------------------------------------------------------------------------
[5 ] MUTATION RATE
------------------------------------------------------------------------
The mutation rate of the GA, where a valid mutation rate is between
0% and 50% which in the software is typed as 0.0 and 0.5.
------------------------------------------------------------------------
[6 ] ELITISM
------------------------------------------------------------------------
When enabled, the 2 best individuals are always selected for the next
generation. These top 2 individuals are NOT a part of GA operations.
------------------------------------------------------------------------
[7 ] MAX RUNS
------------------------------------------------------------------------
The amount of RUNS that the GA would perform. In order to calculate the
number of GA generations one should calculate: maxRuns*popSize.
This parameter is in use ONLY if the 'Run forever' parameter is set
to FALSE
```

Figure 37. EthGA parameters 0-7 explanation.

```
--------------------------------------------------------------------------
[8 ] RUN FOREVER
--------------------------------------------------------------------------
When this parameter is set to TRUE the GA will run continuously and
only stop by user input, otherwise when set to FALSE the GA will run
a number of runs specified by the 'Max runs' parameter.
--------------------------------------------------------------------------
[9 ] PRINT TYPE
--------------------------------------------------------------------------
The type in which the population is printed by the command 'show -pop'.
For available print types enter command 'help -t'.
--------------------------------------------------------------------------
[10] USE CROSSOVER
--------------------------------------------------------------------------
If true, crossover is applied. If false no crossover is applied.
--------------------------------------------------------------------------
[11] CROSSOVER TYPE
--------------------------------------------------------------------------
The type of crossover that the GA is performing.
For available crossover types enter command 'help -t'.
--------------------------------------------------------------------------
[12] CROSSOVER POINTS
--------------------------------------------------------------------------
The amount of crossover points enabled.
This parameter is used ONLY if the 'Crossover type' parameter is set
to NPOINT.
--------------------------------------------------------------------------
[13] USE SELECTION
--------------------------------------------------------------------------
If true, selection is applied. If false no selection is applied.
--------------------------------------------------------------------------
[14] SELECTION TYPE
--------------------------------------------------------------------------
The type of selection algorithm that the GA is using.
For available selection types enter command 'help -t'.
--------------------------------------------------------------------------
```

Figure 38. EthGA parameters 8-14 explanation.

```
[15] RANKING TYPE
--------------------------------------------------------------------------
The type of ranking algorithm that the GA is using.
This is ONLY used if the 'Selection type' parameter is set to RANKING.
For available ranking types enter command 'help -t'.
--------------------------------------------------------------------------
[16]TOURNAMENT SIZE
--------------------------------------------------------------------------
The size of the TOURNAMENT selection algorithms, i.e. the number of
individuals that are included in a tournament. This parameter is ONLY
enabled if the 'Selection type parameter is set to TOURNAMENT'.
--------------------------------------------------------------------------
[17]RANK LIN PARAM
--------------------------------------------------------------------------
The 's' value of the linear ranking algorithm. This parameter is ONLY
enabled if the 'Selection type' parameter is set to RANKING and the
'Ranking type' parameter is set to LINEAR.
--------------------------------------------------------------------------
[18]PROTOCOL TYPE
--------------------------------------------------------------------------
This parameter sets the initialization values of the individuals
when they are created. If it is set to GOOSE the individuals tries to
mimic a goose protocol with a small amount of randomness is included.
If it is set to RANDOM the initialized individual (frame) will be completely random.
--------------------------------------------------------------------------
[19]ICBF - IDLE CYCLES BETWEEN FRAMES
--------------------------------------------------------------------------
This parameter defines the amount of clock cycles between the transmission
of each frame, at the clock frequency of 100 MHz concluding that one
clock cycle lasts 10 ns.
--------------------------------------------------------------------------
```

Figure 39. EthGA parameters 15-19 explanation.