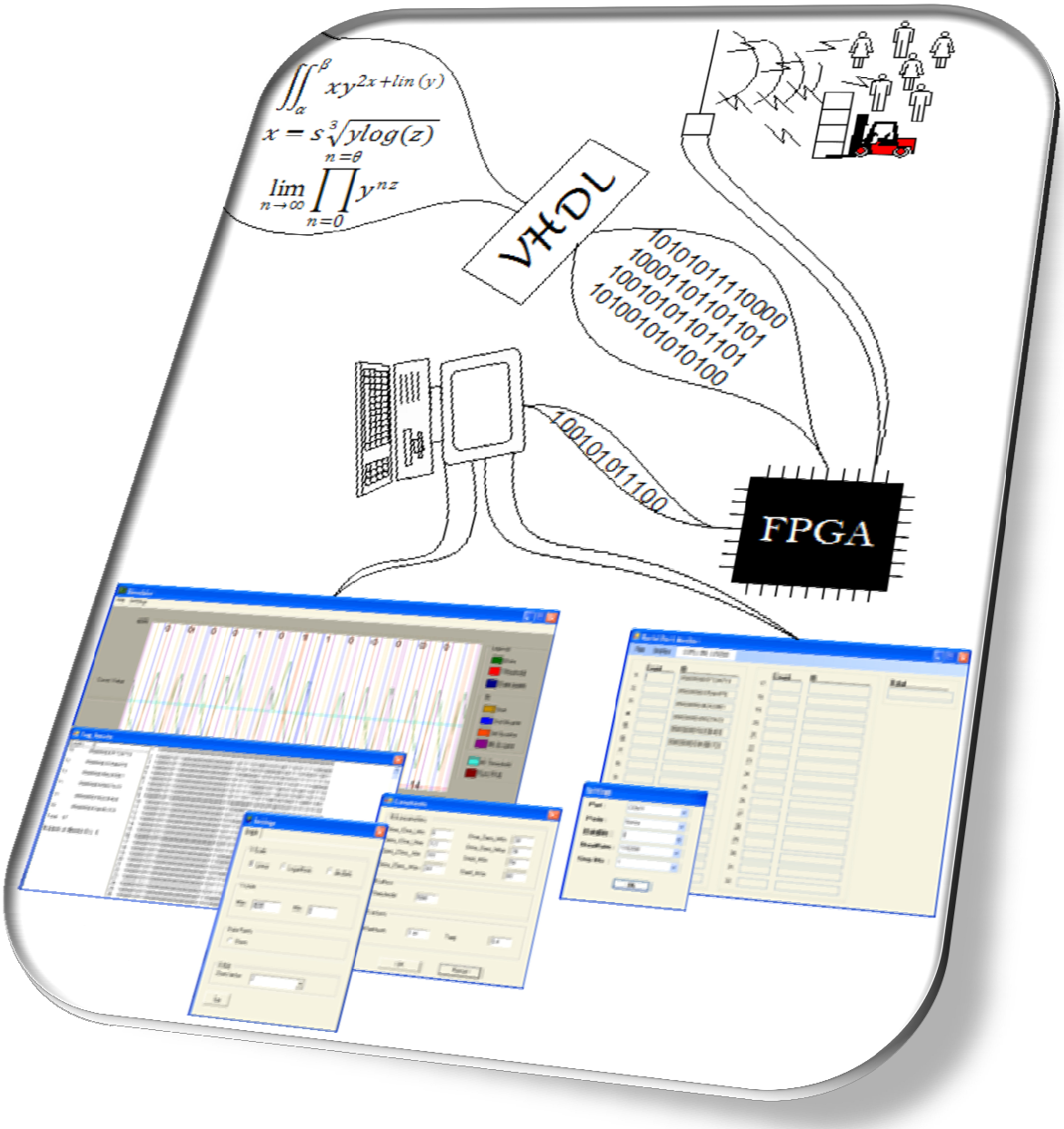
FPGA IMPLEMENTATION OF AN ENHANCED DIGITAL DETECTION ALGORITHM FOR MEDIUM RANGE RFID READERS



FPGA implementation of an enhanced digital detection algorithm for medium range RFID readers

Francois Dominicus Muller

A dissertation submitted in partial fulfilment of the requirements
for the degree
Magister Ingeneriae in Computer Engineering

Supervisor: Prof W.C. Venter (PhD)
2008
Potchefstroom

Preface

Engineering is a multidisciplinary environment challenging the way individuals' reason in order to find feasible solutions to problems. The origins of these problems are not always as clear-cut, or even as complex as they are made to be.

The development of digital systems, parallel and dual processing, has improved to such an extend in the past decade, that ideas thought to be impossible even five years back, are now considered to be old technology.

The development of the hardware needed for this project makes great use of this idea. Whereas previously simple analogue filtering was insufficient to retrieve wanted signals from within extensive white noise levels, modern digital signal processing techniques in conjunction with simple analogue filtering are implemented to retrieve signals for the purpose of this project.

If the research done in order to complete this study has one lesson to teach, it would most certainly be the relativity of the manifestations concerning the origins of different problems.

Executive Summary

The School of Electrical, Electronic and Computer Engineering of the North-West University is conducting research about RFID (radio frequency identification) medium range reader systems for an international company, iPico. The focus area of the present research is the development of a robust tag detection algorithm for noisy environments.

During the past three years a digital detection algorithm was developed. This digital detection algorithm delivered significant improvements in detection of RFIDs over its analogue counterpart, especially in noisy environments. However, the digital detection algorithm was found to be very sensitive with regard to data rate deviations.

Although the latter algorithm improved the detection of RFIDs, ghost (absent) tags were now also detected. The objectives of this project are, to develop an enhanced detection algorithm which is less sensitive to frequency deviations and to eliminate the appearance of the so called ghost tags.

The proposed enhanced algorithm will be implemented on a FPGA (field programmable gate array), more specific the Altera Cyclone EP1CT144C6 FPGA.

Dedication

Dedicated to my Heavenly Father,

In gratitude for my talents, exceptional Parents, supporting family and amazing friends with which He blessed me. Without these blessings this dissertation would not have been feasible. For the lessons He taught me once again, that no man is an island and to see the light in the world when everything seems dark.

Acknowledgements

I would like to express my gratitude to the following people in no particular order;

Prof. W.C. Venter, for his insights in programming.

Mr. C. Voster, for always answering my endless questions, regarding the digital detection algorithm.

Mr G. Oosthuizen, for his help in familiarizing me with the hardware used.

Janny Whitelock, for doing all the work you have done on such short notice, for excellent work delivered and for your patience with my mistakes.

My mother and father, for all the phone calls and financial support.

Ruan van den Heever, for helping with the untangling of my non-sequential thought line, grammar corrections and coping with my semantics.

Chapter I Background

Broad Overview

Chapter II Project History

Technical Insight

Chapter III Literature Study

Supporting Study

Chapter IV Simulation Tools

Design Tools

Chapter V ….Middleware

Evaluation Tools

Chapter VI….Data Rate Sensitivity

Addressing Objective Two

Chapter VII….Algorithm II and the VHDL Implementation

Addressing Objective One

Chapter IIX….Testing

Critical Evaluation

Chapter IX….Final Word

Ending Arguments

*Please refer to appendix H for a CD containing all necessary extra information, documentation and supporting software.

Table of contents

PREFACE ..... **III**
EXECUTIVE SUMMARY ..... **IV**
DEDICATION ..... **V**
ACKNOWLEDGEMENTS ..... **VI**
TABLE OF CONTENTS ..... **VIII**
LIST OF FIGURES ..... **XII**
LIST OF TABLES ..... **XIV**
1 BACKGROUND **1**
1.1 INTRODUCTION 1
1.2 TAGS 1
1.2.1 Passive tags *2*
1.2.2 Active tags *3*
1.2.3 Tag comparison *3*
1.3 TAG IDENTIFICATION NUMBERS (IDS) 4
1.3.1 RO (read-only) *4*
1.3.2 WORM (write once, read many) *4*
1.3.3 RW (read-write) *4*
1.4 READERS 5
1.5 ADVANTAGES OF RFIDS 6
1.6 SYSTEM COMPOSITION FOR THIS RESEARCH 7
1.7 PROBLEM STATEMENT 8
1.8 PURPOSE OF THIS RESEARCH 9
1.9 ISSUES TO BE ADDRESSED 9
1.10 DISSERTATION LAYOUT 10
1.11 CONCLUSION 11
2 PROJECT HISTORY **12**
2.1 INTRODUCTION 12
3 LITERATURE STUDY **15**
3.1 INTRODUCTION 15
3.2 PASSIVE TAGS 16
3.2.1 Tag Specifications *16*
3.2.2 RF Transmission Encoding *17*
3.2.3 Tag ID Packets *19*
3.2.4 CRC (cyclic redundancy check) *20*
3.3 READER 23

3.3.1 Analogue Reader Hardware and Operation *23*
3.3.2 Digital Reader Hardware and Operation *27*
3.3.3 Digital Reader Algorithm *29*

4 SIMULATION TOOLS 35

4.1 INTRODUCTION 35
4.2 SIMULATION DATA 35
4.3 GRAFIEKE 37
4.3.1 Program's functionalities *37*
4.3.2 Errors present in "GrafiekE" *41*
4.3.3 Limitations *42*
4.3.4 Conclusion *44*
4.4 NEW SIMULATION SOFTWARE (SIMULATOR) 44
4.4.1 Introduction *44*
4.4.2 Algorithm extraction *44*
4.4.3 User interface *46*
4.4.4 Interface *48*
4.4.5 Conclusion *49*

5 MIDDLEWARE (GHOST TAGS) 51

5.1 INTRODUCTION 51
5.2 MULTI-THREADED TTY (MTTY) 52
5.3 MULTI-THREADED TTY VERSION 2 (MTTY2) 54
5.4 GHOST TAGS 55
5.5 SERIAL PORT MONITOR – THE NEW MIDDLEWARE 57
5.5.1 Main window *57*
5.5.2 Settings *58*
5.5.3 Threads *59*
5.5.4 Test *59*

6 FREQUENCY SENSITIVITY 61

6.1 INTRODUCTION 61
6.2 BIT DETECTION METHOD 62
6.3 ALGORITHM IMPROVEMENTS 65
6.4 TEST 69
6.5 CONCLUSION 69

7 ALGORITHM III'S VHDL IMPLEMENTATION 71

7.1 INTRODUCTION 71

7.2 VHDL BASICS 71
7.3 VHDL IMPLEMENTATION OF ALGORITHM 73
7.3.1 Filter entity 74
7.3.2 Limit entity 76
7.3.3 Sync detect entity 77
7.3.4 ID detect entity 80
7.3.5 CRC entity 81
7.4 SUPPORT ENTITIES 84
7.4.1 RFID detector entity 84
7.4.2 ADC entity 86
7.4.3 Pack entity 88
7.4.4 Store (FIFO) entity 89
7.4.5 RFID UART entity 92
7.5 CONCLUSION 94
8 TEST 95
8.1 INTRODUCTION 95
8.2 C++ VS VHDL 95
8.2.1 Phase one - Method 96
8.2.2 Phase one - Results 96
8.2.3 Phase one - Conclusion 96
8.2.4 Phase two - Method 97
8.2.5 Phase two - Results 99
8.2.6 Phase two - Conclusion 100
8.3 C++ DETECTION CAPABILITY 100
8.3.1 Method 101
8.3.2 Results 101
8.3.3 Conclusion 102
8.4 TRIO TEST 102
8.4.1 Method 103
8.4.2 Results 103
8.4.3 Conclusion 104
9 FINAL WORDS 105
9.1 CONCLUSIONS 105
9.2 FUTURE RECOMMENDATIONS 105
9.2.1 Analogue signal 106
9.2.2 Digital filter 107
9.2.3 Threshold 107

9.2.4 ID threshold *107*
9.2.5 Algorithm *108*
9.2.6 ID storage *109*

REFERENCE **110**
APPENDIX A. **A.1**
APPENDIX B. **B.1**
APPENDIX C. **C.1**
APPENDIX D. **D.1**
APPENDIX E. **E.1**
APPENDIX F. **F.1**
APPENDIX G. **G.1**

List of figures

FIGURE 1.1 RFID SYSTEM COMPONENTS [1] 1
FIGURE 1.2 READING OF AN RFID TAGS [1] 2
FIGURE 1.3 READER'S DUTY CYCLE 5
FIGURE 3.1 IPICO'S PASSIVE CREDIT CARD TAG. [3] 16
FIGURE 3.2 DATA ENCODING TECHNIQUES [9] 19
FIGURE 3.3 BASIC STRUCTURE OF ID PACKED [3] 19
FIGURE 3.4 A PACKET OF INFORMATION [13][14] 20
FIGURE 3.5 EXAMPLE OF MODULO-2 BINARY DIVISION [13][14][18] 22
FIGURE 3.6 ANALOGUE READER DETECTION FLOW DIAGRAM [16] 24
FIGURE 3.7 ANALOGUE READER DETECTION, SIGNALS [16] 25
FIGURE 3.8 MEDIUM RANGE READER.[3] 26
FIGURE 3.9 ANALOGUE READER'S READING CAPABILITY [3] 27
FIGURE 3.10 CALCULATION OF NYQUIST SAMPLING FREQUENCY 29
FIGURE 3.11 THRESHOLD CHANGE AND START DETECTION 31
FIGURE 3.12 START DETECTION AND SYNCHRONIZATION BIT DETECTION [11][16 32
FIGURE 3.13 SYNCHRONIZATION WEIGHT EQUATION [16] 33
FIGURE 3.14 DETECTION OF DATA BITS [11][16] 34
FIGURE 3.15 NUMBER OF POSSIBLE VALID IDS 34
FIGURE 4.1 SETUP TO OBTAINED SAMPLED DATA 37
FIGURE 4.2 DETECTION SEQUENCE: (A) START OF DETECTION SEQUENCE, (B) SYNCHRONIZATION STAGE COMPLETE AND DATA DETECTION STAGE STARTED AND (C) END OF DETECTION SEQUENCE, VALID ID DETECTED 39
FIGURE 4.3 GRAFIEKE PARAMETER CHANGE FUNCTION 41
FIGURE 4.4 ALGORITHM'S FIVE MAIN PARTS 45
FIGURE 4.5 SIMULATOR 46
FIGURE 4.6 (A) ARRAY DECLARED, (B) ROW-WISE IN MEMORY AND (C) COLUMN-WISE IN MEMORY 49
FIGURE 5.1 MULTI-THREADED TTY, VERSION 1 52
FIGURE 5.2 ID TAG PACKET COMPOSITION TO RFID MIDDLEWARE 53
FIGURE 5.3 MULTI-THREADED TTY, VERSION 2 54
FIGURE 5.4 (A) TEST 1 RESULTS, (B) TEST 2, SECTION 1 RESULTS AND (C) TEST 2, SECTION 2 RESULTS 56
FIGURE 5.5 SERIAL PORT MONITOR, MAIN FORM 57
FIGURE 5.6 SERIAL PORT MONITOR, SETTINGS FORM 59
FIGURE 6.1 DATA SAMPLES FOR 128 KBITS/S DATA TEMPO AND 3.072 MHZ SAMPLING FREQUENCY, ... 62
FIGURE 6.2 ALIGNMENT CALCULATIONS 63
FIGURE 6.3 AUTO ADJUST TECHNIQUE (A) CALCULATION OF *SAMPLE COUNT* VALUE (B) EFFECT OF AUTO ADJUST 64
FIGURE 6.4 SYNCHRONIZATION START METHODS, (A) OLD IMPLEMENTATIONS, (B) NEW IMPLEMENTATION 66
FIGURE 6.5 LAST SYNCHRONIZATION BIT TO START BIT PROBLEM 67

FIGURE 6.6 AFFECT OF FREQUENCY DEVIATION ON SYNCHRONIZATION BITS 68
FIGURE 6.7 ALGORITHM'S ABILITY TO HANDLE FREQUENCY DEVIATION 69
FIGURE 7.1 VHDL SYNTAX'S BASIC COMPONENTS [6],[29] 72
FIGURE 7.2 VHDL, ALGORITHM ENTITIES, SUPPORTING ENTITIES AND EXTERNAL HARDWARE 74
FIGURE 7.3 FILTER ENTITY BASIC DESIGN 75
FIGURE 7.4 LIMIT ENTITY BASIC DESIGN 77
FIGURE 7.5 SYNC DETECT ENTITY BASIC DESIGN 78
FIGURE 7.6 DIVISION ENTITIES' SIGNALS 79
FIGURE 7.7 ID DETECT ENTITY BASIC DESIGN 81
FIGURE 7.8 CRC ENTITY BASIC DESIGN 82
FIGURE 7.9 CRC CHECK ENTITY BASIC PROCESS DESIGN 83
FIGURE 7.10 RFID DETECTOR ENTITY BASIC DESIGN 85
FIGURE 7.11 ADC ENTITY BASIC DESIGN 87
FIGURE 7.12 PACK ENTITY BASIC DESIGN 89
FIGURE 7.13 STORE ENTITY BASIC DESIGN 90
FIGURE 7.14 FIFO SATURATION 91
FIGURE 7.15 RFID UART ENTITY BASIC DESIGN 93
FIGURE 8.1 THE FREQUENCY ABILITIES OF THE DIFFERENT IMPLEMENTATION OF THE ALGORITHM 96
FIGURE 8.2 SAMPLED NOISE SIGNAL 97
FIGURE 8.3 FPGA SETUP FOR TESTING 98
FIGURE 8.4 DETECTION CAPABILITIES OF C++ VS VHDL, NOISE EXCLUDED 99
FIGURE 8.5 VHDL CORRELATION TO C++ DETECTION CAPABILITY, NOISE EXCLUDED 99
FIGURE 8.6 DETECTION CAPABILITIES OF C++ VS VHDL, WITH NOISE 100
FIGURE 8.7 VHDL CORRELATION TO C++ DETECTION CAPABILITY, WITH NOISE 100
FIGURE 8.8 C++ ALGORITHMS' COMPARISON, NOISE EXCLUDED 101
FIGURE 8.9 C++ ALGORITHMS' COMPARISON, WITH NOISE 102
FIGURE 8.10 TRIO TEST SETUP 103
FIGURE 8.11 DETECTION CAPABILITIES OF TRIO TEST, NOISE EXCLUDED 103
FIGURE 8.12 DETECTION CAPABILITIES OF TRIO TEST, WITH NOISE 104
FIGURE 9.1 ID THRESHOLD ERROR 108

List of tables

TABLE 1.1 RFID TAG COMPARISON [1][5] 3
TABLE 3.1 IPICO'S TAG SPECIFICATIONS [3] 17
TABLE 3.2 A FEW INTERNATIONAL STANDARD CRC GENERATOR POLYNOMIALS [13][14] 23

Abbreviations

AC	alternating current
ADC	analogue-to-digital converter
CPU	central processing unit
CRC	cyclic redundancy check
csv	comma separated values
dB	decibels
DC	direct current
DLL	dynamic link library
DSP	digital signal processing
EDMA	enhanced direct memory access
FIFO	first - in - first – out
FIR	finite impulse response
FPGA	field programmable gate array
GUI	graphical user interface
HDL	hardware description language
IC	integrated circuit
ID	identification
IDE	integrated development environment
Inf	infinite
I/O	input / output
IP rating	ingress protection rating [15], [28]
LCD	liquid crystal display
MFC	Microsoft foundation class
MTTTY	Mutli-threaded TTY
PC	personal computer
PCM	pulse code modulation
PPD	peak pulse detection
PSU	power supply unit
RF	radio frequency
RFID	radio frequency identification
RO	read-only
RTL	register transfer level

RW	read-write
S/N	signal to noise
SRAM	static random access memory
TTL	transistor-transistor logic
TTY	teletyping [22]
txt	text
UHF	ultra - high frequency
UART	universal asynchronous receiver – transmitter [11]
VB	visual basic
VHDL	VHSIC (very high speed integrated circuit) HDL
WORM	write once, read many

1 • Chapter

1 Background

1.1 Introduction

RFID systems are composed of three mandatory components:

- tags
- readers and,
- RFID middleware

Although the system described is not a complete system as described in [1], it is an accurate model for the purpose of the project. The focus of this project falls on the interaction and mechanism of the reader and the tags and not the commercial application of RFID systems.

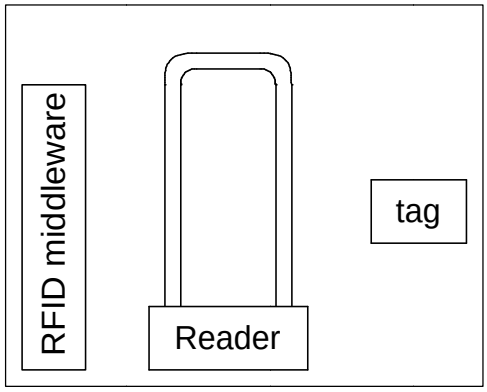


Figure 1.1 RFID system components [1]

A tag contains an ID (identification) number which is read by a reader where after it is transferred to the RFID middleware. Middleware can either be a database or, as implemented during the course of this research a user interface device only displaying the extracted IDs. User interface devices vary from hand held units with LCD's (liquid crystal display) to computer screens. Figure 1.1 depicts the latter idea in the simplest of ways.

1.2 Tags

A tag is a device that can be placed on an item which is intended to be identifiable or monitored and come in many different shapes and sizes. Items may be commodities,

import and export bulk, live stock and even humans. Different tags can be categorised in different ways. The most common method is to distinguish tags by type of power supply it contains.

When classified in this manner, two types of tags are generally found in the industry:

- passive tags and,
- active tags[1][3][5]

In order to put the remaining discussion in this chapter into perspective, refer to the following depiction which, when compared to figure 1.1, is slightly more detailed.

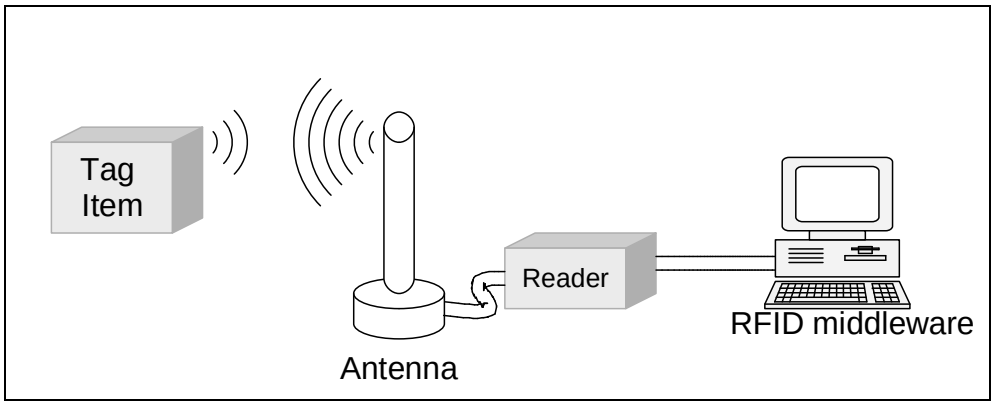


Figure 1.2 Reading of an RFID tags [1]

1.2.1 Passive tags

Passive tags have no built in power supply. These tags house an IC (integrated circuit) programmed with a specific ID and an antenna. The housing is generally made of plastics but may be a different material. [1][5] Passive tags are powered by the magnetic field induced by the radio waves propagating from the antenna situated at or in the tag reader. An onboard capacitor is charged after this AC signal is rectified and powers the electronic circuit onboard the tag. The same antenna used to pick up the propagating radio waves from the reader is also implemented for transmission. [1][3][5]

1.2.2 Active tags

An active tag houses an onboard power supply. These tags therefore require more maintenance than passive tags in that batteries need to be replaced regularly. Simple operation costs are hereby increased. [1][3][5]

The basic working of an RFID system implementing active tags does not differ much from a system implementing passive tags. The most important difference is that a reader utilized in a system implementing active tags does not transmit an energizing signal as does those utilized in systems implementing passive tags. Active tags are mostly also larger than passive tags. [1][3][5]

In retrospect to passive tags, the IDs of active tags are transmitted continuously when activated, by the user, and is then detectable by a reader when in range. [1][3]

1.2.3 Tag comparison

Table 1.1 RFID tag comparison [1][5]

Passive tag	**Active tag**
Physical size smaller	Physical size larger
Cheaper to manufacture	More expensive to manufacture
No maintenance (no battery replacement)	Maintenance needed (battery replacement required)
More expensive reader	Less expensive reader
Physical size of reader larger	Physical size of reader smaller
No moving parts	Might have moving parts (for example a switch)
Can withstand harsh environments, for example corrosive chemicals and high temperatures.	Less compatible with harsh environments.

To overcome the problem of crosstalk in the RF (radio frequency) environment a dual frequency protocol is implemented in a passive tag system. Readers transmit at 125 kHz and receive at a carrier frequency of 6.8 MHz. The signal transmitted by a

passive tag, even though the power level is very low, can be extracted from the surrounding spectrum by careful filtering.

1.3 TAG identification numbers (IDs)

ID numbers are used to distinguish between different tags. Classification of tags is based on the ability of the tag to handle rewrite procedures. The most common classifications are:

- RO (read-only)
- WORM (write once, read many) and,
- RW (read-write)

Both passive and active tags can be found to support any of these three rewrite classes. [5]

1.3.1 RO (read-only)

RO tags are programmable only once. The ID is usually burned onto the tag IC at the time of factorization. This method places a limitation on the customization of a tag's ID, however ensures high ID security. RO tags are referred to as factory programmable. [5] Tags used in this project are passive RO tags.

1.3.2 WORM (write once, read many)

Unlike RO tags, WORM tags are user programmable. ID security of WORM tags is compromised by allowing this versatility. WORM tags are referred to as field programmable. [5]

1.3.3 RW (read-write)

The ID's of RW tags may be rewritten any number of times varying from 10,000 through 100,000. The ability to change the ID of a tag greatly extends the freedom of application given to the user. These tags may also be rewritten by either themselves or the reader. This ability however, poses a security risk. RW tags are referred to as field programmable or reprogrammable. [5]

1.4 Readers

RFID readers, as mentioned previously, are devices build to read tag IDs in its field of reception. Readers are also known as interrogators. The physical size of the reader is dependent on the type of tag it reads. The circuitry required to generate the energizing radio waves, to activate passive tags, can be large. Contrary to the former statement, active tag readers are normally only receivers. Certain readers have the capability to write to the tags as well. This ability provides the reader with the capability of creating or destroying tags. [5]

As discussed in sections 1.2.1 and 1.2.2 passive tag readers transmit an energizing signal in order for the tag to operate. This energizing signal is not continuously transmitted. Readers only transmit at certain time frames, referred to as the duty cycle of the reader. These duty cycles are controlled by international laws. [5]

Figure 1.3 depicts a typical duty cycle as transmission is undertaken by a reader.

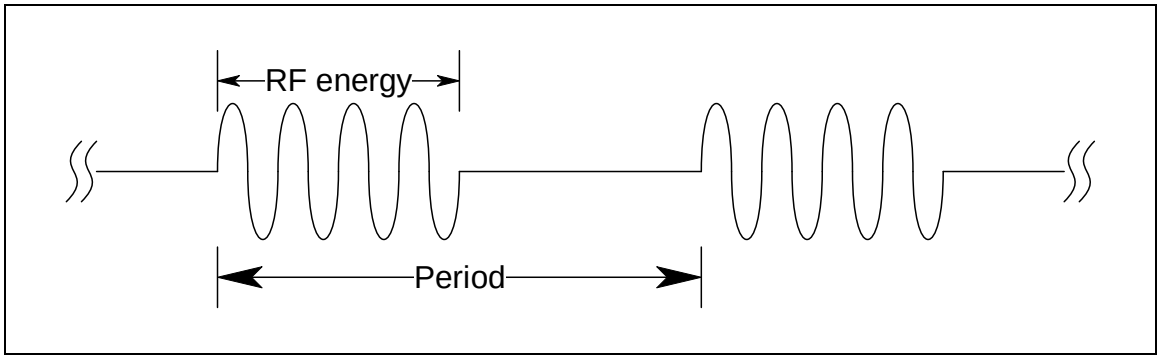


Figure 1.3 Reader's duty cycle

Readers can operate in two different modes namely:

- autonomous and,
- interactive [5]

Autonomous mode readers constantly read all tags within the limits of its reading zone.

Interactive mode readers receive a read command from the application software. This read command may be induced by a user operating the application. A tag is read once and added to a tag list. A tag list contains the following information: [5]

- an unique tag identifier,
- reading time,
- number of times a specific tag is read and,
- reader's name

Fixed readers as well as handheld readers are commercially available. Fixed readers are stationary and usually mounted on walls. These readers are less expensive than their handheld counterparts however, they provide less mobility.

Typical communication protocols utilized in the industry to interface with RFID middleware are: [5]

- serial communication and,
- network communication

Serial communication may either be RS-232 or RS-485. Network communication may be either a wired or wireless network. [5]

1.5 Advantages of RFIDs

Certain advantages are offered by RFID systems versus current adopted identifications technologies such as: bar-codes or magnetic strips typically used on credit cards. The following few critical issues address and summarise the characteristics of RFID systems: [2][5]

- *Alignment is not necessary*: It is not requirement for the reader and tag to be in line of sight of each other. This is an advantage of an RFID system reading tags by means of RF. This characteristic saves time by improving the processing of large quantities of items in a shorter time. [2][5]

- *High inventory speed*: A large number of items can be read simultaneously. [2][5]
- *Variety of forms factors*: A variety of readers and tags exist. The latter differ in size, packaging material and read range. These characteristics make RFID a versatile identification system that can be implemented in almost any imaginable environment. [2][5]
- *Item-level tracking*: The length of the RFID number makes it possible to uniquely identify billions of items. [2][5]
- *Rewritable:* Not all RFID tags bear this characteristic. The latter provides a reusable functionality for tags although posing a security risk. [2][5]
- *Contactless:* An RFID tag can be read without any physical contact between the reader and tag. [2][5]
- *Rugged:* RFID tags can, to a fair extent, sustain rough operational and environmental conditions. [2][5]

1.6 System Composition for this Research

In order to comply with the objectives of this research it seems necessary to supply a definition of the different research elements namely:

a. Algorithm simulation tool
b. Middleware
c. Revised algorithm namely “Algorithm III”
d. Implementation of “Algorithm III”

Designers of RFID detection algorithms for implementation in VHDL are supplied with certain tools of ease during software development. VHDL in general is known to originate from VHSIC (very high speed integrated circuit) and HDL (hardware description language)) and is the language which will be used for development of the revised algorithm, before implementation on the FPGA (field programmable gate array). One such tool is the algorithm simulation tool which provides the developer with a visual feedback and testing environment of the algorithm being developed whilst coding in a C++ environment. The latter poses the advantage of quickly implementing major software changes, quick compilation and visual feedback

resulting in more information concerning the performance of the algorithm being developed.

As stated previously, the middleware, simply put, provides the designer with visual feedback as to which tags was detected by a reader once the algorithm is implemented in VHDL.

During the course of this research an improved algorithm may have to be developed in order to address certain issues inherently designed into previous written software called “Algorithm II”. These issues form part of the purpose of this research and will be addresses in the following section. For the rest of this dissertation, this improved algorithm will be referred to as “Algorithm III”.

After development of “Algorithm III” in the simulation environment it will be possible to convert the code from C++ to VHDL, load the VHDL binaries onto the FPGA and compare the output of the latter with that of the simulation. This should give an indication of the accuracy of the VHDL implementation and proof the success thereof.

1.7 Problem Statement

The following two issues, defined as part of the tasks to be completed during this research, are key problems:

- Previously a simulation version of the digital detection algorithm (Algorithm II) was implemented in C++.and a practical version of the digital detection algorithm was implemented in VHDL on a FPGA (see Appendix G, phases VI and V). However, when using the same input data, there is a significant difference in effectiveness between the two implementations, with the C++ simulation version obtaining better detection of tags. The first objective of this project is to find the problem with the VHDL implementation and eliminate it or at least minimize it.
- Secondly, the digital detection algorithm (Algorithm II) is very sensitive for data rate deviations in the communication from the tag to the reader. The second

objective of this project is to develop a new version of the detection algorithm (Algorithm III) that does not exhibit the sensitivity of Algorithm II for data rate deviations.

1.8 Purpose of this Research

- To convert “Algorithm II” from the C++ environment to VHDL with greater accuracy in order to improve on the effectiveness of the VHDL implementation of “Algorithm II”.
- To increase the effectiveness of “Algorithm II” by desensitizing this algorithm with regard to data rate deviations in the *tag to reader* communication.

Due to the level of complexity and interaction between different software components in this system it might be necessary to re-develop “Algorithm II” in total starting in the simulation environment (C++), later converting to VHDL in order to properly address all of the above mentioned issues.

1.9 Issues to be addressed

In order to achieve the objectives of this project the following issues need to be addressed:

- *Algorithm simulation software:* There are a number of problems and limitations with regard to the current simulation software. They need to be corrected before the algorithm can be improved and implemented on the FPGA.
- *Ghost (absent) tags:* In order to receive an accurate indication of the detection capabilities of an algorithm implemented in VHDL, ghost (absent) tags need to be eliminated. Ghost tags are tags of which ID’s are being displayed by the middleware but physically they are not present in the field of the reader. Ghost tags can also be classified as invalid ID’s being presented by middleware.
- *The sensitivity of “Algorithm II” with regard to data rate variation:* The algorithm is currently very sensitive to data rate variation. The reason for this must be found and corrected.

- *The VHDL implementation of an algorithm*: Previous studies conducted found that the VHDL implementation of “Algorithm II” resulted in a less accurate output than the C++ simulation implementation. This conversion issue needs to be addressed.

1.10 Dissertation Layout

The core focus of this study is the implementation of an enhanced digital detection algorithm (Algorithm III) on a FPGA.

In order to study this phenomenon thoroughly and to address the core of the problem, various components need be examined. The following paragraphs will provide the reader with a global layout of this dissertation as well as the order in which the different issues at hand will be addressed.

Chapter 2 is an outline of the history regarding this project. This is a summary of the previous projects undertaken, the different objectives, results and conclusions.

Chapter 3 will concentrate mainly on the necessary theoretical knowledge in order to thoroughly conduct the present study. The current algorithm, “Algorithm II” is also discussed in detail; in order for the reader to better understand this dissertation and the outcomes thereof.

Before the current VHDL implementation of Algorithm II can be studied, the baseline C++ implementation will be examined. The C++ implementation is available in a program called “GrafiekE” (included on the CD). The simulation program however experiences a number of errors and limitations. ***Chapter 4*** discusses these errors and limitations as well as the new simulation program.

The RFID middleware functions as an indicator of the detection capability of the VHDL implementation of “Algorithm II”. The current RFID middleware displays ghost (absent) tags. ***Chapter 5*** focuses on finding the origin of these ghost tags and also provides a feasible solution.

As previously discussed, “Algorithm II” currently exhibits a great sensitivity to tag data rate deviation. ***Chapter 6*** examines this phenomenon and provides a possible solution.

The core of this project, the VHDL implementation of the digital detection algorithm, is discussed in ***Chapter 7***. This improved algorithm, “Algorithm III“ is implemented and will be tested with the aid of the RFID middleware.

Chapter 8 is concerned with testing and the test results. Herein the different tests are defined and the results given.

A conclusion as well as possible recommendations for future research concerning this project is discussed in ***Chapter 9***.

1.11 Conclusion

The reader should now have a basic understanding of an RFID system in the broad and the advantages of the technology as discussed. The technology, however not without fault, is very versatile and has even more versatile application areas. Thus from previous studies it seems there is a need and demand to improve RFID systems to be more effective in a variety of environments.

Chapter 2

2 Project history

2.1 Introduction

In order to facilitate the reader of this dissertation in understanding a certain amount of concepts dealt with during the research as well as certain technical aspects of the system a brief history of the build up to this research will be given here.

With the understanding that the entire history of this project encapsulates the development of a digital RFID reader to possibly replace now old analogue readers it is important to understand that the history of the project basically consists out of *five different phases*. This research will conclude the sixth phase.

It is also important to keep in mind that, the research conducted during this phase (phase VI) of the greater project was a separate study on its own entirely and conducted by a separate person than the individual whom conducted the research in phase V of the project as was the four phases prior to that. In other words, each one of the six phases comprised of a master degree dissertation research study in its own right.

The following five paragraphs very briefly summarises the objectives and outcomes of the previous five phases as conducted during the greater project.

- ***Phase I*** mainly dealt with a feasibility study. Subsequent to this feasibility study was the development of an algorithm (then developed in a language of choice) in order to proof the functionality of the feasibility outcomes. This algorithm was referred to as “Algorithm I”. Secondly hardware had to be identified on which the digital algorithm (Algorithm I) could have been implemented. For reasons stated by the researcher two TMS’s from Texas

Instruments were recommended, specifically the TMS320C6711C-200 or TMS320C6711B-100 (see Appendix G for details).

- ***Phase II*** was concerned with the implementation of the latter algorithm on some sort of processor. The choice as to which processor should have been used was not clear however, at that stage a TMS (digital signal processor supplied by Texas Instruments) was chosen amongst the following two options: a FPGA or a TMS

- ***Phase III*** Mainly, the objectives of this phase of the project history addressed the refinement of "Algorithm I" in order to compromise between system specifications and current available hardware. These refinements typically included change of filter orders implemented as well as ADC (analogue-to-digital converter) considerations. (As the main idea behind the entire project is to develop a digital RFID reader in order to harness the advantages of digital signal processing techniques over those of analogue techniques, ADC considerations become critical.)

- ***Phase IV*** of the project history dealt with the comparison between the older "Algorithm I" and the newly developed "Algorithm II" Both algorithms were subjected to similar environments regarding signal-to-noise ratios, ADC specifications, input signal voltage ranges, analogue filters as well as input pulse voltage ranges. After completing this study it was concluded that a 12-bit ADC better suits the performance of "Algorithm II" rather than an 8-bit ADC for obvious reasons.

- ***Phase V*** mainly dealt with the optimization of the coding of "Algorithm II" for the TMS by ridding the code of unnecessary operations, calculations and extra overhead. The final stage of this phase required the first implementation of the detection algorithm (Algorithm II) on a FPGA. The reason for moving to a FPGA is simple, a FPGA is a parallel processor compared to the sequential TMS. This change meant much more processing power was available at lower processor clock cycles.

For an elaborated discussion, including examples, results as well as documentation from the different history phases please refer to appendix G.

3 • Chapter

3 Literature study

3.1 Introduction

Chapter 1 covered the basic components of an RFID system and provided the reader with an overview of the broader operation of an RFID system. The possible variations and combinations of tags and readers in an RFID system are almost unlimited. This project focuses on medium range RFID readers combined with passive tags, especially the medium range RFID readers used by iPico. The digital detection algorithm being developed during the period of this research can be implemented on any RFID system using the same protocols as does the readers supplied by iPico.

The main focus of chapter 2 fell upon the components and history thereof being used and implemented in the RFID system relevant to this project.

Basic theories, mechanisms and assembly of the relevant components will be discussed during the course of this chapter. As the project is contingent upon previous projects (Refer to Appendix G for a short description of the previous projects), a brief description of “Algorithm II”, the algorithm used to further improve upon during the course of this particular study, will also be given. “Algorithm II” is also known as the PPD (peak pulse detection) algorithm. [11][16]

The literature and background given as this chapter progresses will be presented as part of two categories.

1. The tags being used and their specifications and,
2. The RFID readers and their specifications.

The section on the readers will be divided into two categories namely the analogue readers and the forthcoming digital readers.

3.2 Passive Tags

3.2.1 Tag Specifications

In this project, iPico's passive credit card type tags are being used. These tags implement iPico's dual frequency technology. Refer back to chapter 1 for a description of the implementation of dual frequencies when using passive tags in a RFID system. The tags used implement iPico's iP-X multi read anti-collision protocol. The basis of operation of the anti-collision protocol is random transmission. Any number of tags will transmit their IDs at random times when energized by the reader's RF field. This protocol allows the system to read multiple tags simultaneously. Put more practically, the transmission of the random IDs occur at such speed that reading of different tags seems simultaneous. Without any doubt the number of collisions between transmissions increases as the number of tags increase, resulting in more time elapsing whilst reading more tags. Also, the time necessary to read a certain number of tags do not increase linearly as the number of tags increase. [3][10]



Figure 3.1 iPico's passive credit card tag. [3]

The low frequency used to wake (energize) the tag can penetrate light (not dense) mediums better than high frequencies, for example UHF (ultra - high frequency). The high frequency being used to transmit the tag ID allows high data rates. This high data rate together with the anti - collision protocol enables the system to achieve high tag read rates. [3][10]

The most important specifications of iPico's passive dual frequency tags are listed in Table 3.1.

Table 3.1 iPico's tag specifications [3]

Description	Value	Unit
Transmitting frequency	6.8	MHz
Receiving frequency	125	kHz
Factory programmed ID length	64	Bits
Transmission rate	128	kbits/sec
Temperature range	-40 to 85	°C
IP rating (see Appendix A)	IP68	-

3.2.2 RF Transmission Encoding

Encoding of transmission data in this project refers to the data transmitted by the tag. The signals transmitted by the reader contain no data and the tag only use this transmitted wave as a source of energy.

Line codes are used by the receiver to extract clock-synchronization information from the incoming data stream. Line codes are a digital encoding scheme which represents the digital signal to be transmitted, by an amplitude and time discrete signal, which is optimally tuned for the specific properties of the physical channel. One such line code scheme is the Manchester encoding scheme. [6][7]

Clock period information can only be extracted effectively when the incoming data stream contains random digital ones and zeros. Digital zeros are usually represented by a negative voltage the latter is a result of the construction of most RF receiver hardware. However very little or mostly incorrect information can be derived if long strings of one’s or zero’s follow each other. This results in the receiver becoming biased meaning that the reference voltage used for comparing the incoming data stream against, shifts mainly towards the positive or the negative side. [6][7]

To overcome this problem any one of a number of encoding schemes may be implemented. Encoding schemes are technologies developed many years ago and are still in use today. One of the techniques used, is the Manchester encoding scheme. In Manchester encoding, a logic one (or positive voltage value) is represented by a one-to-zero transition, all in the duration of a single clock cycle.

Where the first half of the bit-clock is one and the second half of the bit-clock is zero. A logic zero (or negative voltage value) is represented in the exact opposite way. [6][7][8]

Manchester encoding thus ensures that, at least one transition will always occur inside a single bit-clock. Although the problem of repeating ones and zeros for clock-synchronization is solved, the solution comes at a price. Data rate is always half the clock rate, or described more simply, twice the bandwidth is necessary to transfer the same amount of Manchester encoded information then is necessary to transfer information not encoded by this scheme. [6][7]

By making use of Manchester encoding, the longest modulation period is restricted to one full clock period or a single bit. This long modulation period – relative to RFID systems, especially when using passive tags - requires large amounts of energy. Passive tags must usually be as small and compact as possible. To comply with this criteria the passive tags cannot contain large capacitors, meaning they must be energy efficient. As mentioned in chapter 1, passive tags are depended on a reader for energizing the onboard circuitry. [6][7][9]

The latter results prove that Manchester coding is not suitable for use in passive tag systems. However, the problem can be overcome by reducing the modulation period to a quarter of a bit period. This encoding scheme is known as Glitch encoding. [9][17]

In Glitch encoding a logic one is represented by a digital one in the first quarter of the bit period, followed by zeros in the remaining three quarters. A logic zero is represented by a digital one in the third quarter of the bit period and zeros in the rest of the quarters. This coding scheme also eliminates the clock-synchronization problem resulting from repeated ones and zeros with the added benefit of reducing the energy needed for conducting the modulation. [9][17]

Figure 3.2 illustrates the two coding schemes. Benefits and disadvantages of the coding schemes discussed can be seen fairly clear in the figure below.

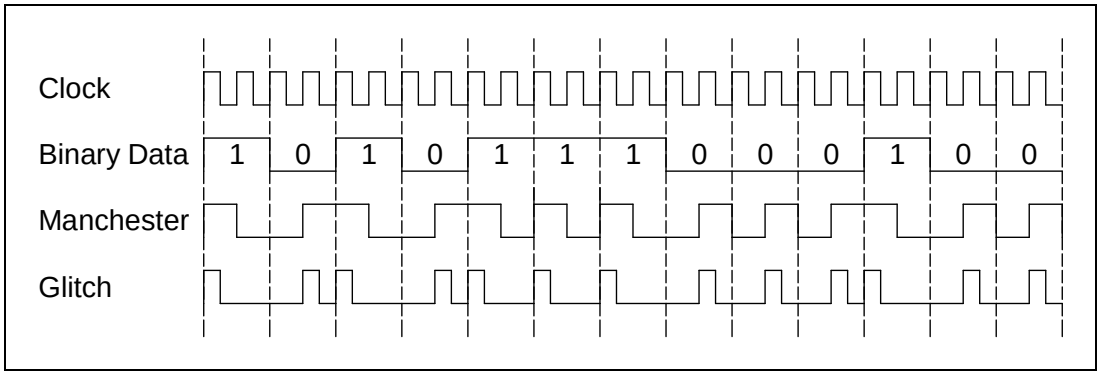


Figure 3.2 Data encoding techniques [9]

3.2.3 Tag ID Packets

An ID packet transmitted by a RFID tag comprises of four parts. The ID packet is Glitch encoded and the composition of the packet is specific to iPico's passive tags, other manufacturer ID packet compositions may vary. Figure 3.3 provides the composition of iPico's tag ID packet.

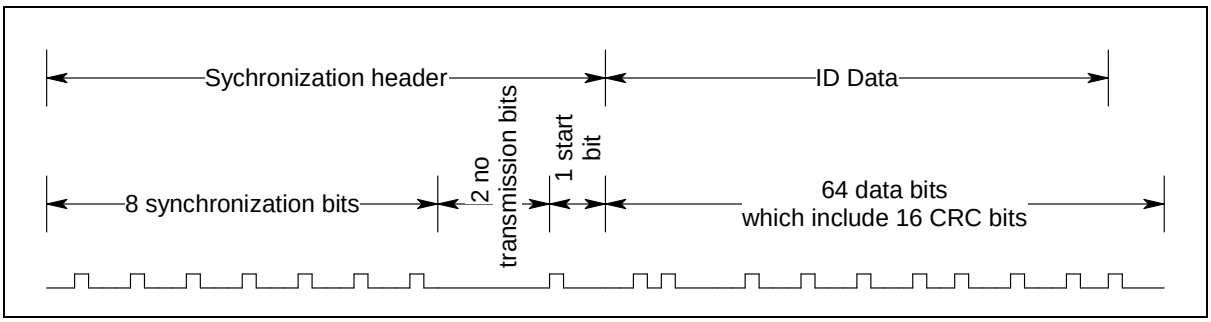


Figure 3.3 Basic structure of ID packed [3]

The first part included in the ID packet is taken up by the synchronization bits. Synchronization bits are eight logic zero bits. The functionality of the synchronization bits is to give an indication to the reader that a possible ID might be present and that a tag is transmitting.

The second part of the ID packet is 2 *no-transmission* bits; thus neither a logic one nor a logic zero is transmitted in this time frame. The third part comprises of a start bit. The latter is simply a logic one. The first 3 parts of the ID frame is used by a reader to verify that the data received next will be ID data.

The fourth part is composed of two augmented parts. The latter comprises of a sixty-four data bit part serving a dual purpose. The entire 64 bits are the actual ID being transmitted however, the last 16 bits are also used to run a CRC (cyclic redundancy check) on the ID data sent. [3][11]

3.2.4 CRC (cyclic redundancy check)

Two methods can be used to determine whether a message (in this case the ID data packet) received was correct.

1. *Error detection:* This method enables the user to determine whether a possible error has occurred during transmission and provides the user with options to request for re-transmission of the incorrect data packet. [12][13][14][18]
2. *Error correction:* In error correction the user is able to determine whether an error occurred and can correct it. Error correction methods is only able to correct a certain number of bit errors in a data packet. [12][13][14][18

The tags in this project (iPico's passive tags) make use of error detection. A more elaborated discussion follows.

Error detection makes use of a checksum which is appended to the original message, as shown in Figure 3.4 below. In order to be certain that a packet is received correctly, a robust checksum algorithm is implemented. The main function of checksum algorithms is to distribute the set of valid bit sequences randomly and evenly across the set of possible bit sequences. [14][18]

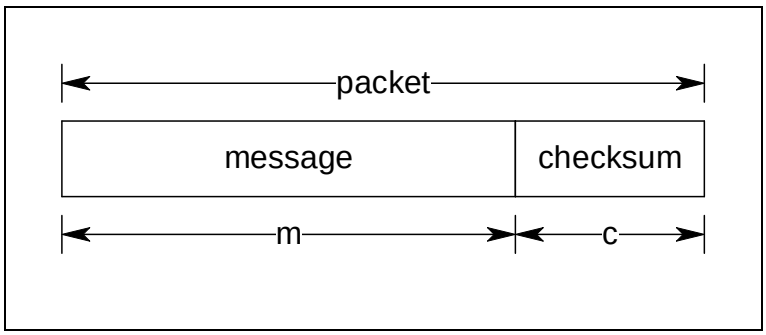


Figure 3.4 A packet of information [13][14]

A commonly used error detection method in binary streams is CRC. The tags used in this project make use of the latter error detection method. CRC uses the properties of division for its checksum. CRC uses the properties of division for its checksum. The appealing property of division is that the remainder of the division process changes rapidly when only a small number of bits in a data packet are changed. Thus, a bit sequence can be validated by its unique CRC. [12][13][14][18]

The basic method for calculating a CRC checksum is as follows: [13][14][18]

- Message bits are appended with *c* 0 bits; this augmented message is the dividend, using modulo-2 binary division. (See the next paragraph for a description of modulo-2 binary division.)
- A predetermined c + 1 bit binary sequence, called the generator polynomial, is referred to as the divisor.
- The checksum is the c-bit remainder that results from the division operation.

Basic operation of modulo-2 binary division is as follows; (Starting at the most significant bit in the dividend and apply the following) [13][14][18]

- If the most significant bit of the remainder is a one, the divisor is said to have divided into the dividend.
  - Set the appropriate bit in the quotient to a one.
  - XOR the remainder with the divisor and store the results in the remainder.
- Else if the most significant bit is a zero, the divisor *cannot* divide into the dividend.
  - Set the appropriate bit in the quotient to a zero.
  - XOR the remainder with zeros
- Left-shift the remainder, shifting the next bit of the dividend.

Figure 3.5 on the next page is a pictorial summary of the latter description.

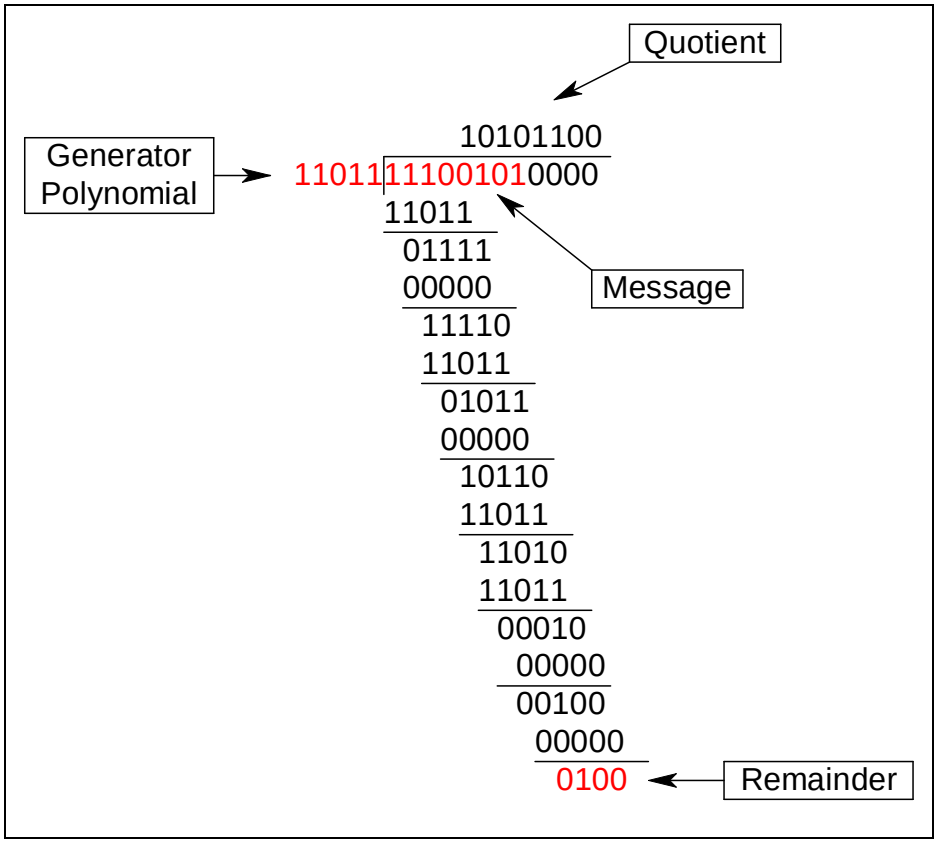


Figure 3.5 Example of modulo-2 Binary division [13][14][18]

The robustness of a checksum is determined by the length of the generator polynomial as well as the specific generator polynomial. The length of the generator polynomial refers to the number of bits the polynomial has, for example CRC -16 has 16 bits. The specific generator polynomial refers to the composition of the polynomial, for example the first two CRC generator polynomials both have 16 bits but there composition differs. [13][14][18]

The probability of detecting a random error is $1-2^{-c}$, where c refers to the number of bits of the CRC. Standard lengths and strong generator polynomials adopted internationally do exist, refer to table 3.2. [13][14][18]

Table 3.2 on the next page provides a few examples of CRC generator polynomials, but CRC-16 is important for this project.

Table 3.2 A few international standard CRC generator polynomials [13][14]

	CRC - CCITT	CRC - 16	CRC - 32
Checksum Width	16 bits	16 bits	32 bits
Generator polynomial	10001000000100001	11000000000000101	10000010011000001000111011011011

3.3 Reader

The reader as mentioned in the introduction of this chapter focuses on the analogue reader (iPico's reader) and the baseline digital detection algorithm (Algorithm II), in accordance with the appropriate hardware necessary for this project. The reason for the in-depth examination of the analogue detection techniques is because the receiver and carrier wave suppression is currently handled by the analogue reader, as supplied by iPico. The digital hardware being developed during this project contains no receiver components and thus utilizes the analogue reader for this short fall.

3.3.1 Analogue Reader Hardware and Operation

The analogue RFID reader, which the digital RFID reader is meant to replace, is an IP-X dual frequency medium range reader from iPico. The analogue reader implements iPico's anti-collision algorithm, which enables the reader, at a rate of 30 tags/sec, to successfully read up to 120 tags simultaneously. Due to the high receiving frequency the analogue reader read range is extended further than other low frequency RF systems, at the same time it still posing the low frequency advantages for its transmit link. [3]

Analogue readers can be implemented as a single unit, known as the master configuration or as a dual unit, known as the master and slave configuration. Master units transmit at 125 kHz, where as the slave unit transmits at 126 kHz and both receive at a frequency equal to 6.8 MHz. A master unit control the slave units bound to it. The master unit sends the collected tag data as well as the tag data collected by the slave to the RFID middleware. The reader read range, increases from 1 m to 1.8 m, when the master – slave configurations is implemented. [3]

Figure 3.6 is a flow diagram representing the detection method used by iPico's analogue reader. The depiction also illustrates the basic principles of the detection method.

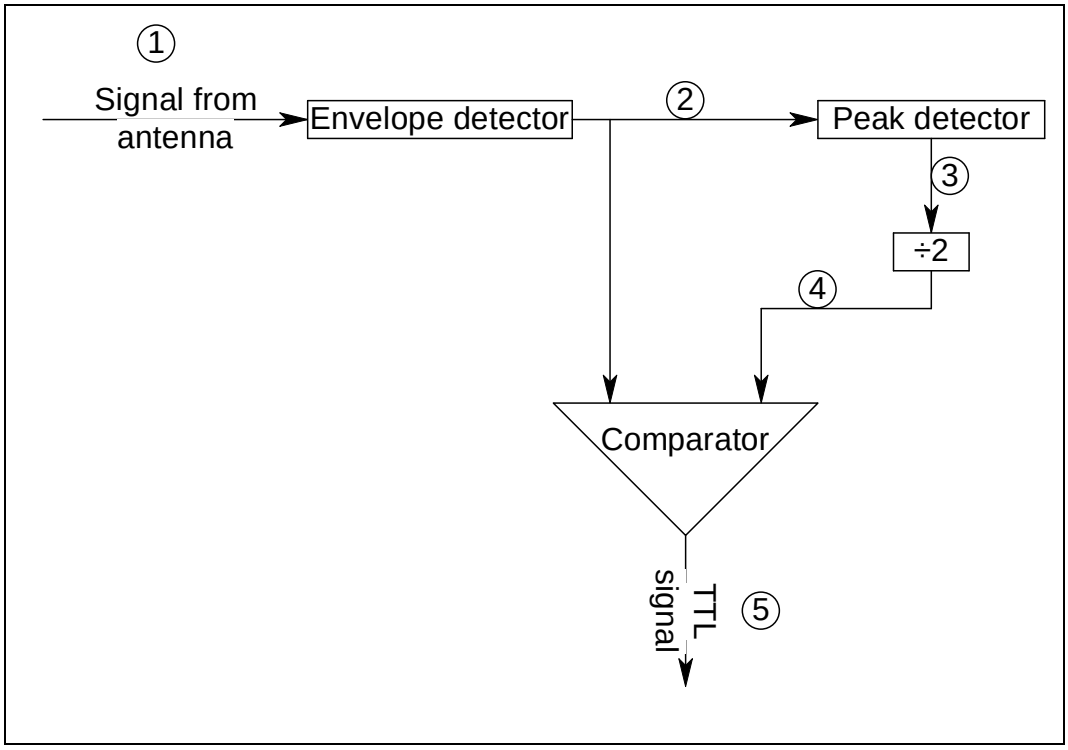


Figure 3.6 Analogue reader detection flow diagram [16]

Figure 3.7 illustrates the metamorphoses undergone by the analogue signal as the components from the flow diagram in Figure 3.6 transforms the analogue signal. The blue signal represents the signal attained from the antenna (1), it is the 6.8 MHz signal transmitted from the tag. During this first stage the signal is enveloped. A basic envelope detector comprises of a rectifier and a low pass filter respectively. The rectifier rids the signal attained from the antenna of the ever so present negative part after which a low pass filter is implemented to filter out any unwanted high frequencies. [7] The red signal (2) represents the output of this enveloping stage. [16]

The latter signal is then split into two separate signals, one for the peak detector and the second for the comparator signal. During the peak detector stage the peaks of the signal in (2) are detected. A basic peak detector circuit is a diode in series with a parallel resistor and capacitor. The function of this diode is to rectify the signal, however the signal currently attained from the system is already rectified at some

stage. Thus, the output of a peak detector is depended on the time constant generated by the parallel resistor and capacitor combination. [8] The output of the peak detector stage is illustrated by the green signal depicted in Figure 3.7 (3). [16]

During the last stage of manipulation, signal (2) and signal (4) levels are compared. When signal (2) levels are larger than signal (4) levels a logic one is obtained as output from the manipulation process as depicted in Figure 3.6. When the opposite is true, a logic zero is obtained. These latter ones and zeros are a result of a comparator making use of TTL (transistor-transistor logic) as depicted in Figure 3.6. TTL signals are represented by the black signal (5), in Figure 3.7. [16]

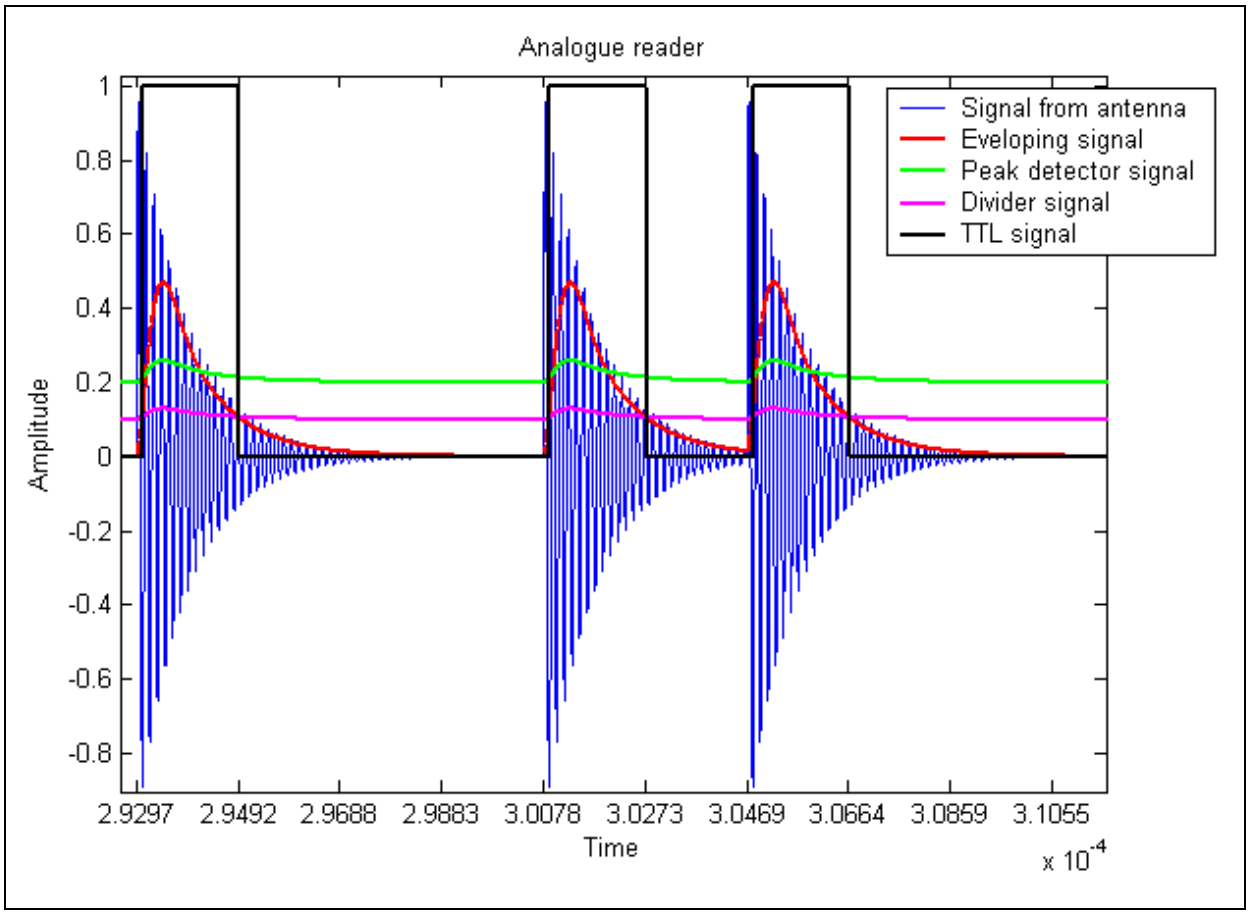


Figure 3.7 Analogue reader detection, signals [16]

Figure 3.8 is an illustration of a medium range RFID reader as supplied by iPico.



Figure 3.8 Medium range reader.[3]

Important specifications of the medium range readers are listed in table 3.3, followed by the reading capabilities of the reader.

Table 3.3 Medium range RFID reader's specifications [3]

Description	Value	Unit
Transmit frequency for master unit	125	kHz
Transmit frequency for slave unit	126	kHz
Receive frequency	6.8	MHz
Read range in master setup	1	m
Read range in master and slave setup	1.8	m
Anti-collision protocol reading saturation point (simultaneously)	120	tags
Moving tags readability: at 8m/sec	10	tags
: at 4m/sec	20	tags
Multi read rate: 20 tags in beam	90	tags/sec
: 100 tags in beam	55	tags/sec
Temperature range	-10 to 70	°C
IP rating (see Appendix A)	IP40	-

The following graphical illustrations as depicted in figure 3.9, respectively convey the expected reading capabilities of the reader as obtained from documentation. The specific test setups are not specified in the relevant documentation.

From figure 3.9 (a) it can be seen that the number of tags detected per second decrease as the number of tags increase in the reader's read range. This is due to the saturation of the anti-collision protocol implemented.

Figure 3.9 (b) illustrates the deprivation in detection capability of the analogue reader to reader tags moving as different speeds as the number of tags in the read range is increase.

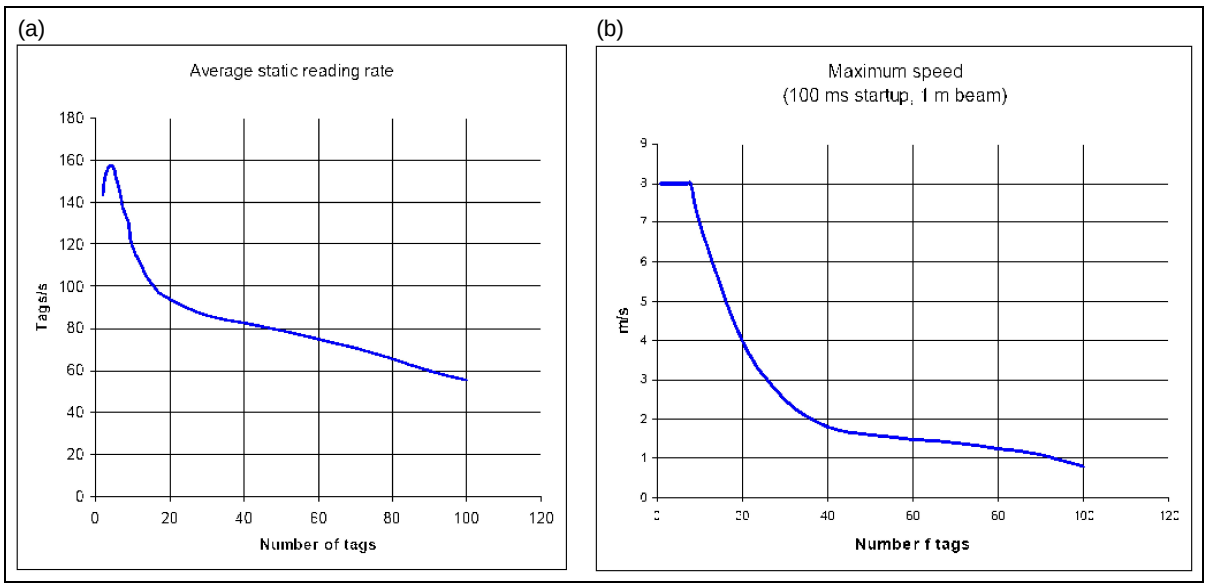


Figure 3.9 Analogue reader's reading capability [3]

3.3.2 Digital Reader Hardware and Operation

The development of the digital reader hardware was not part of a specific previous project. It was rather a collaboration between researchers (Mr Christo Vorster and Mr. Gerdhard Oosthuizen), designers (Stellenbosch University) and the knowledgebase obtained from previous projects as described in appendix G. Thus for the sake of clarity only a brief elaboration will be given. The digital hardware comprises of three main components:

- a data unit,
- a processing unit and,
- a communication unit.

The data unit is composed of a signal modification unit and a signal conversion unit.

The signal modification unit is connected to the analogue reader after the enveloping circuitry of the analogue reader as depicted in figure 3.6. The signal modification unit comprises of two stages. The first stage is a buffer and the second stage an amplifier. The purpose of the signal modification unit is to isolate the analogue

readers' circuitry from the circuitry of the digital reader and to manipulate the analogue signal so as to place it inside the quantification range of the ADC. [30]

The second unit, housed inside the data unit known as the signal conversion unit is a 12 bit ADC from Texas Instruments (THS1206). The sampling rate of this unit is 3.072 MHz and is interfaced parallel to the processing unit. The input voltage range is between 1.5 V and 3.5 V. From the tests performed during phase IV and results thereof it was decided to use a 12 bit ADC. (Refer to chapter 2 and appendix G for an in depth description of the previous phases of the project as well as tests performed during these phases.)

Before venturing further into the detail of the processing and communication units, it seems appropriate to give a short overview of the ADC process in itself in order to establish a solid understanding of the sampling theorem, Nyquist theorem and conversion techniques. The latter is important because this basic knowledge was used to decide the oversampling frequency for the tag data, as will be discussed in the next paragraph.

One modulation technique used to convert from an analogue signal to a digital signal is PCM (pulse code modulation). An analogue signal's voltage range is divided into different levels (or equal levels for linear PCM); each level represents a predefined binary value. The analogue signal is then sampled and the digital or binary value representing a level closest to its amplitude is assigned. The number of levels is depended on the number of binary bits used to represent a level. [7][8]

$$= 2 \quad - 1$$

Where N = the number of levels

m = the number of bits per sample

The process of conversion from an analogue signal to a digital signal is called quantization. An analogue signal has infinite voltage levels, thus an error is made when analogue signals is quantize. This is known as quantization error. [7][8]

The Nyquist rate is the minimum sampling rate which must be implemented in order to accurately, without aliasing errors, convert an analogue signal into its digital equivalent. This rate is taken as twice the maximum frequency present in the analogue signal to be digitized. Determining the Nyquist rate from the rate of transmission generated by the tag (128kbits/s) combined with the effect of the Glitch encoding on the bit representation a minimum sampling rate of 1.024MHz is obtained, see figure 3.10. [27]

$$1.024\ \text{MHz} = 2(\text{Nyquist requirement}) \times 4(\text{Glitch encoding}) \times 128\ \text{kHz}(\text{From } 128\ \text{kbits/s tag transmission rate})$$

Figure 3.10 Calculation of Nyquist sampling frequency

To allow for oversampling, the sample rate was increased to 3.072 MHz. The latter provides the algorithm (embedded in the processing unit) with six samples per quarter bit. [27]

The processing unit is the Cyclone (EP1CT144C6) FPGA from Altera. “Algorithm II” was found to be very computational intensive as per the testing under Phase V of the greater project. (Refer to appendix G) For this reason and possible others a FPGA is used. The parallel processing abilities of a FPGA makes this type of processing unit very attractive. Other digital signal processors for example the TMS range from Texas Instruments, which is a sequential processor, was found not to be able of executing the mentioned necessary algorithm between samples. [11]

Last but not the least the communication unit comprises of two RS-232 communication channels. The function of the communication unit is mainly to transfer the detected RFIDs from the reader and manipulation processes to the middleware. The middleware in this case is a programme which displays the IDs as reference for the user or developer.

3.3.3 Digital Reader Algorithm

Next, the digital detection algorithm “Algorithm II”, will be conveyed because it was developed during previous stages of this project. As a precondition for reading this

section, because this development is not part of this dissertation, the reader is advised to refer to the reference if specific validation for constants and principles used in the algorithm is desired by the reader. This section only provides an overview of the functioning of the algorithm, to provide the reader with insight to certain aspects of the dissertation. Furthermore the constant values in the algorithm are depended on the hardware used, especially the ADC. [16] Remember that the latter algorithm was already an upgrade of a previous version developed even earlier called “Algorithm I”. (Refer to appendix G for more detail on these algorithms.) The detection algorithm is composed of five sections:

- the filter section,
- a threshold section,
- a synchronization section,
- a data detection section and,
- a CRC section

The latter four sections can be referred to as the four stages of detecting a valid tag ID. These stages are dependent on each other and are activated in the same sequence, once the specific stage’s criteria are met.

- *stage 1,* pre-detection stage. Searching for the start of an ID packet.
- *stage 2,* synchronization stage. Determines whether the synchronization header (see figure 3.3) of the ID packet is present.
- *stage 3,* ID extraction stage. Once a valid synchronization header was detected the ID data section of the ID packet can be extracted.
- *stage 4,* validation stage. Verify the CRC extracted and there by confirming the validity of the ID detected.

The filter section is a five point averaging filter. This section stores the previous four samples gathered and adds the next (newest/latest) sample to these previous four samples. The initial values of the four stored samples are zero. Values obtained are then used in the rest of the algorithm (the four stages) as the new sample values.

The objective of this section is to rid the samples of the ever so present white noise or more generally known as *common mode*. [11][16]

The algorithm utilizes a threshold value, to minimize the number of false activation of the synchronization stage. If a value (start of possible ID packet) above this threshold value should be detected the synchronization stage is activated. [11][16]

The threshold value is calculated by adding a buffer value (200 quantization levels) to the maximum value detected amongst the previous hundred samples. The threshold value changes with every hundred samples, see figure 3.11. The threshold value is zero for the first hundred samples. The search for an ID packet only starts after the first hundred samples. After activation of the second stage the threshold section is inhibited and resumes only after a valid ID is detected or an error occurs in one of the following three stages. [11][16]

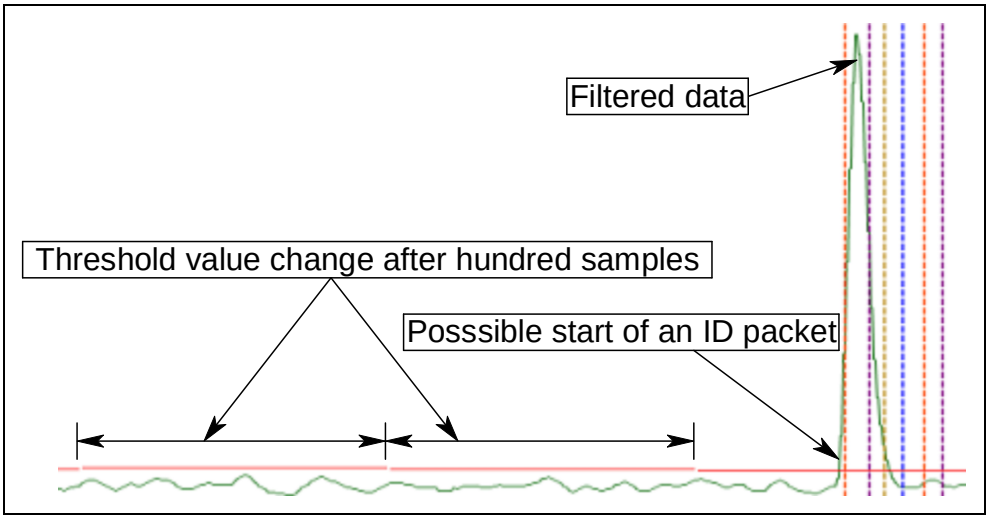


Figure 3.11 Threshold change and start detection

During the second stage the algorithm determines whether seven synchronization bits, seven logic zeros, are present. However the synchronization header is composed of eight logic zeros. The algorithm uses the first synchronization bit only as a start bit for the ID packet. Refer back to the section covering the tag ID packets for an explanation of the latter. The algorithm allows one undefined bit amongst these seven synchronization bits. An undefined bit is defined as neither a logic zero nor a logic one. [11][16]

The seven synchronization bits are detected in the following manner. Energy in the first quarter, logic one position, is compared to energy in the third quarter, logical zero position. Remember that the definitions of the logic bits are a result of the tag's ID packet which uses Glitch encoding scheme. If the energy in the third quarter is greater than the energy in the first quarter, it is assumed that a logic zero was detected. The same holds true for the reversal of this scenario. [11][16] Figure 3.12 illustrates this detection method.

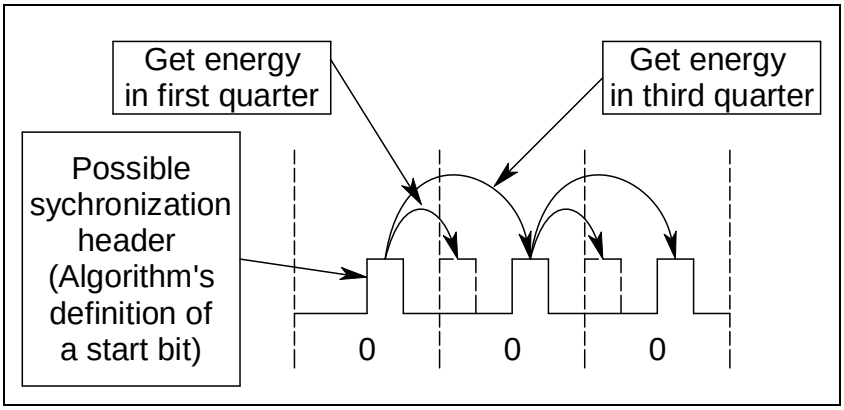


Figure 3.12 Start detection and synchronization bit detection [11][16

After the synchronization bits were detected, the algorithm checks for 2 no-transmission bits. (Refer to the section on the tag ID packets for an explanation of the composition of the ID packets.) During this stage, it is expected that neither a logic one nor a logic zero should be detected. The algorithm searches whether logic bits are present during these 2 no-transmission cycles.

The manner in which this is done is simply by comparing the sample value to a certain predetermined threshold value and determining if the sample value is higher or lower than the threshold. The latter threshold should not be mistaken for the threshold referred to earlier, used to detect a possible start of transmission. The latter threshold is determined by the energy in the synchronization bits and is defined in the algorithm as *synchronization weight*. The equation below is used to calculate the synchronization weight.

$$\text{synchronizationweight} = \frac{\sum_0^{\text{sync bits}} 1^{st}\text{ quarter energy}}{\text{sync bits}} + \text{scaling factor} \times \left(\frac{\sum_0^{\text{sync bits}} 3^{rd}\text{ quarter energy} - \sum_0^{\text{sync bits}} 1^{st}\text{ quarter energy}}{\text{sync bits}} \right)$$

Figure 3.13 Synchronization weight equation [16]

Where sync bits referred to the number of valid synchronization bits detected. This number can either be six or seven, depending whether an undefined bit was detected.

The algorithm allows nine samples to be above the threshold to compensate for noise in the environment. The reason nine samples are allowed is because of the definition of noise in this research. The noise characteristics were specified by iPico and are intended to simulate noise generated by a switch mode power supply. If these conditions are fulfilled, the algorithm determines whether the start bit (ID data frame's start bit, see figure 3.3) is present. The start bit is defined as a logic one. [11][16]

The second stage is now inhibited and the third stage is started. The tag's ID is extracted from the filtered sample data. The ID bits are determined by the same method as the synchronization bits. [11][16]

Figure 3.14 illustrates the detection of logical ones and zeros in the ID data frame of the ID packet.

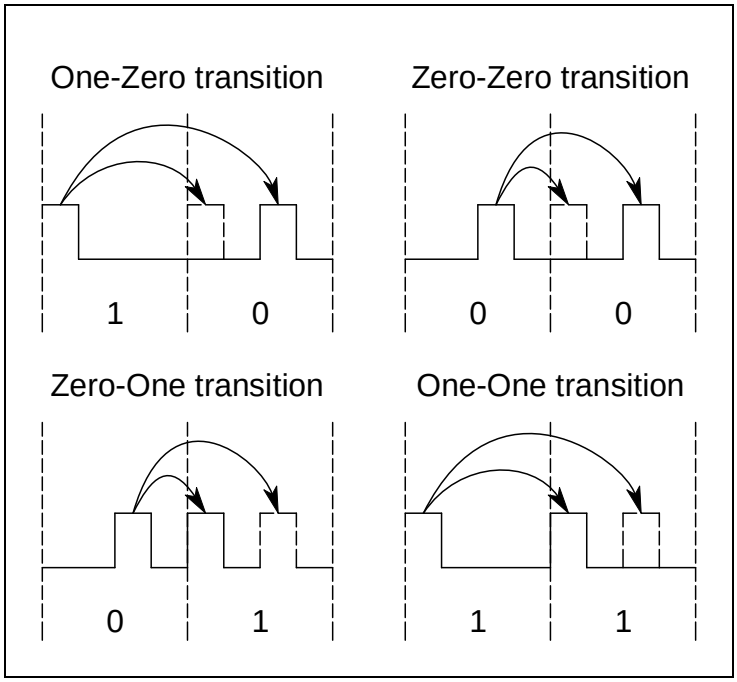


Figure 3.14 Detection of data bits [11][16]

The last stage is activated after 64 bits were detected. Although the CRC is an *error detection method,* a method is implemented to correct certain invalid IDs. The principles implemented by the algorithm determining the presence of a logic one or a logic zero, allows for indecisive conditions (usually occurs when defined noise is present in the bit). When these conditions occur the bit is set to an undefined bit. The algorithm allows for three undefined bits amongst the ID bits. The values of these undefined bits are determined by the CRC section. [11][16]

The invalid ID correction mentioned in the previous paragraph, only applies to these undefined bits. When no undefined bits are present in the detected ID bits and the CRC of the ID bits is invalid, the ID is discarded. However when undefined bits are present the CRC is calculated first for a logic zero and second for a logic one in the undefined bit position. If only one of the CRC validations is correct, the ID with the valid CRC is presumed as a valid ID detected. The number of IDs which need to be validated is depended on the number of undefined bits and increases exponentially, see equation below. [11][16]

$$\text{number of possible valid IDs} = 2^{\text{number of undefined bits}}$$

Figure 3.15 Number of possible valid IDs

Chapter 4

4 Simulation tools

4.1 Introduction

During the development of “Algorithm II”, Mr. C. Vorster developed a piece of software (named “GrafiekE”) to serve as a simulation environment for the latter algorithm, to aid in the design process. This simulation program was written in Visual Studio 6.0, on a C++ platform. ”GrafiekE provided him with the ability to test the algorithm without implementing the algorithm on the applicable hardware. [16]

The first section of chapter 4 serves only as informational background. In this section the origin of the data used in ”GrafiekE” is provided and the relevancy of the data for this project is discussed.

Furthermore a brief elaboration of the simulation program (“GrafiekE”) will be given. The elaboration will include the functionalities, limitations and errors of ”GrafiekE”. This will aid the reader in understanding certain decisions made in the development of the new simulation program.

The development of the new simulation environment (named “Simulator”) will be discussed briefly later in this chapter. A more comprehensive discussion and detail design of ”Simulator” is provided in appendix B. The simulation environment had to be improved in order to address the issues experienced with “GrafiekE” and to better aid the designer of the algorithm during the design process.

4.2 Simulation Data

Algorithm II was first implemented on a TMS (Refer back to chapter 2 and appendix G.). The TMS provided the designers/researchers with the ability to store ADC samples and transmit it to the PC. This ADC data was used in the simulation

program (”GrafiekE”), instead of theoretical generated data, contrary to previous studies.

The test data was obtained by making use of a 12-bit ADC from Texas Instruments (TH1206) sampling at 3.072MHz. The same ADC IC was used in this project.

The sampled data was stored in a .csv (comma separated values) file format. The number of samples captured in a data set were very near to 500 000 samples. Two types of sampled data were considered; data including noise and data excluding noise. The latter noise referred to is the iPico defined noise (simulating a switch mode power supply) and do not refer to noise in the setup environment.

Furthermore the data sets also contained an important aspect of a RFID system. The signal strength the reader receives from the tag decreases exponentially as the distance between the two increases. The equation below displays the relationship between the transmitter and receiver distance and the path loss (degrading signal strength). [7][8][19]

$$\overline{PL}(d)\alpha\left(\frac{d}{d_0}\right)^n \text{ [19]}$$

Where : [19]

- PL is the path loss,
- d is the distance between the transmitter and receiver,
- n is the path loss exponent, which indicates the rate at which the path loss increases as d is increased. Under normal circumstances the path loss exponent n is environment specific where RF signals is concerned and,
- d_0 is the free space close – in reference distance.

The sample data sets were generated by using the following setup, see figure 4.1. The data sets have the following characteristics:

- Specific distance between reader and tag, with iPico specified noise, or
- Specific distance between reader and tag, without adding iPico specified noise.

The distance between the reader and tag was increased in increments of 10 cm starting at 10 cm. The last data set was recorded with a distance of 130 cm between the reader and tag.

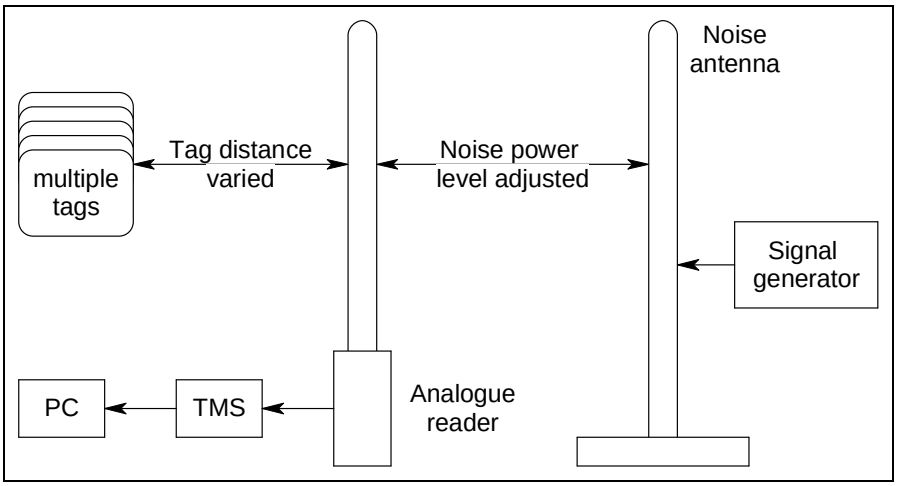


Figure 4.1 Setup to obtained sampled data

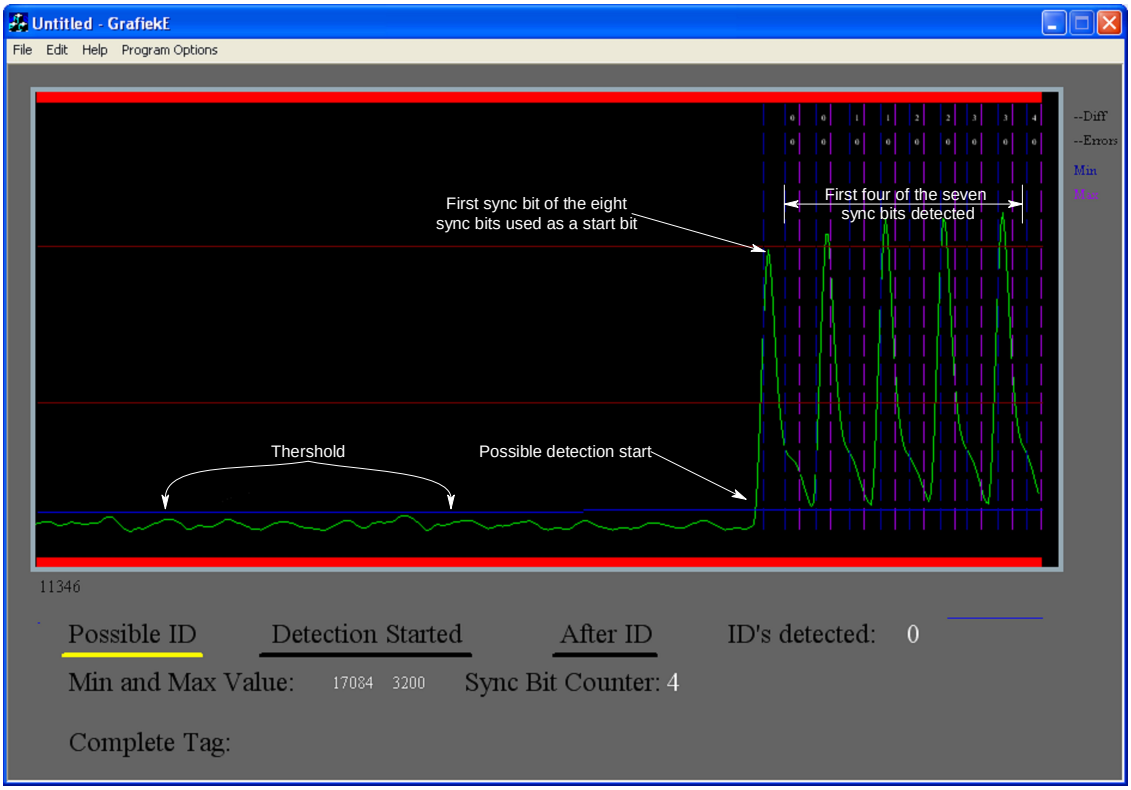
4.3 GrafiekE

4.3.1 Program's functionalities

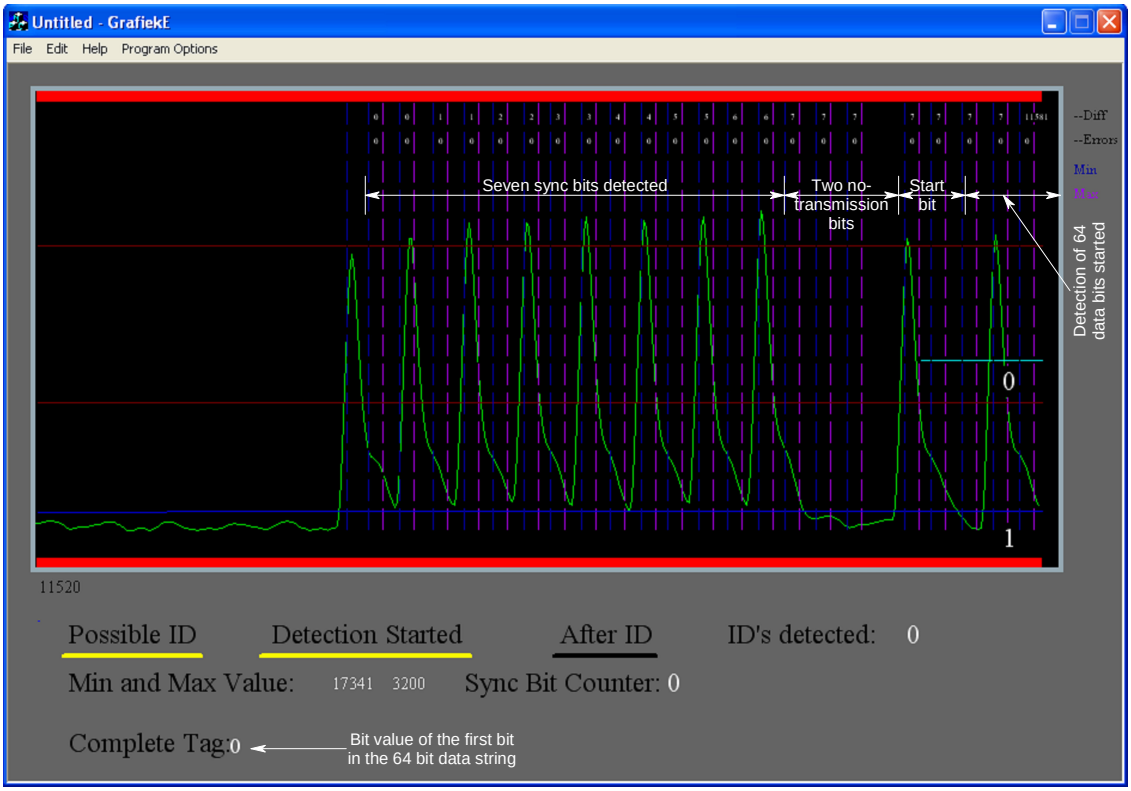
This program's functionalities will be discussed from an operational point of view. Before any of the program's functionalities can be used, sample data must be loaded and a starting point in a specific data set must be specified. The program is capable of interpreting two file formats, .csv files and .txt files. A further requirement has been placed on the input files, each line in the file should only represent one data sample. The reason for allowing the user to specify a starting point is to provide the user with the ability to skip a certain amount of samples if needed.

The simulation environment provides the user with the ability to view the output of the algorithm for a specific data set, 420 samples at a time (one program frame). The significance of the 420 samples is unknown as the program was developed in a previous study and no documentation is available. The simulation environment was developed in such a way that the user can view the algorithm's output in repetitive steps, by calling the run function. This allows the user to examine and interpret the algorithm's output on the following 420 samples. Figure 4.2 shows three screenshots of "GrafiekE", to visually illustrate the simulation program.

(a)


Untitled - GrafiekE
File Edit Help Program Options
--Diff
--Error
Min
Max
First sync bit of the eight sync bits used as a start bit
First four of the seven sync bits detected
Thershold
Possible detection start
11346
Possible ID
Detection Started
After ID
ID's detected: 0
Min and Max Value: 17084 3200
Sync Bit Counter: 4
Complete Tag:

(b)


Untitled - GrafiekE
File Edit Help Program Options
--Diff
--Error
Min
Max
Seven sync bits detected
Two no-transmission bits
Start bit
Detection of 64 data bits started
0
1
11520
Possible ID
Detection Started
After ID
ID's detected: 0
Min and Max Value: 17341 3200
Sync Bit Counter: 0
Complete Tag: 0
Bit value of the first bit in the 64 bit data string

(c)

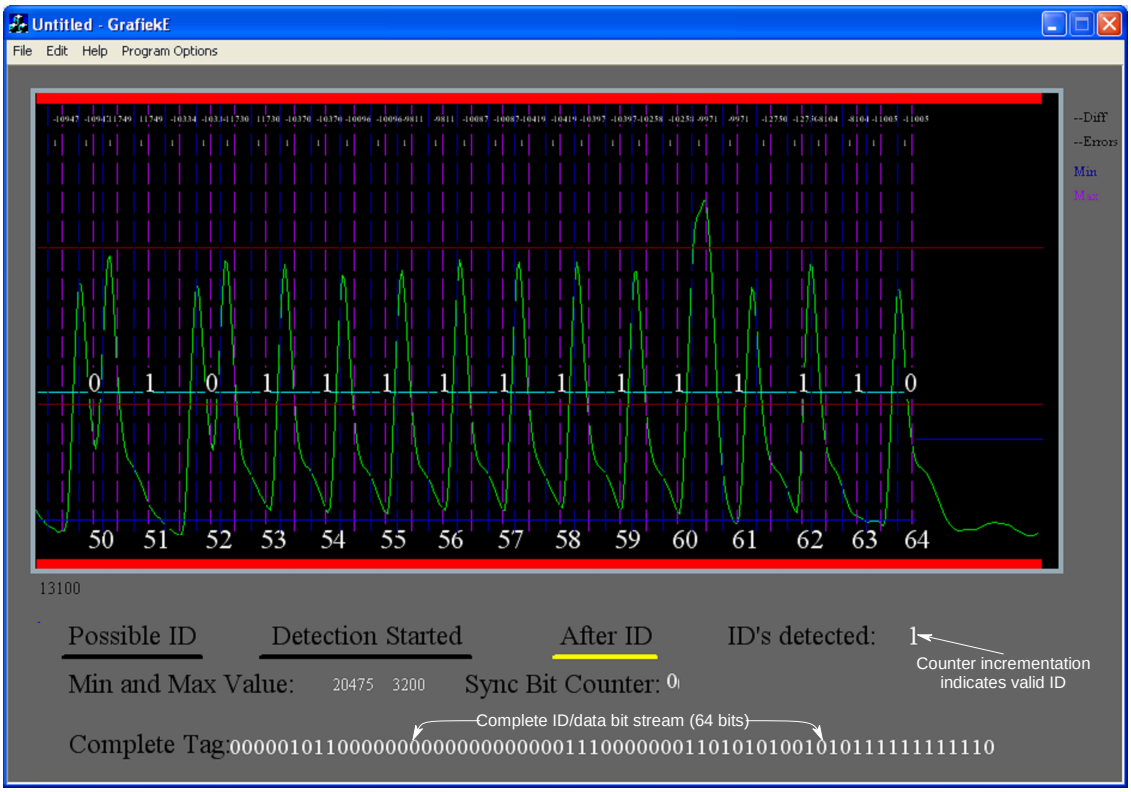


Figure 4.2 Detection sequence: (a) Start of detection sequence, (b) Synchronization stage complete and data detection stage started and (c) End of detection sequence, valid ID detected

The program displays the filtered data (green signal) and the algorithm output. The output is represented by a number of signals. Starting at figure 4.2 (a) the different signals will be explained.

The horizontal blue line represents the detection threshold which, when exceeded, indicates that a possible ID detection could have started. In figure 4.2 (a) the filtered data exceeds the blue line and the synchronization detection stage is activated. The simulator indicates this by making a yellow line visible below the “Possible ID” label. Furthermore the individual quarter bit lines of the sync bits are displayed. These lines are represented by the vertical blue and purple dash lines. The number of valid synchronization bits detected is displayed next to the “Sync Bit Counter” label. See section 3.3.3 for detail on algorithm’s stages.

Figure 4.2 (b) illustrates the display state of the program after seven valid synchronization bits, 2 no-transmission bits and the start bits were detected (Consult

chapter 3 for discussion on Algorithm II). When comparing the bottom half of screenshot (a) and (b), the “Detection Started” label is underlined by a yellow line, which indicates that the data bits (ID) detection started. The first bit value is also visible in the screenshot. Furthermore the magenta horizontal line represents the ID threshold value. This threshold value forms part of the decision process to determine the value of a specific bit (see chapter 3 for details). Each bit inside the 64 ID bit-string representing the ID value as well as the position of the specific bit in the ID bit-string is indicated on the screen, by the two white values in the bit. The top value represents the logic value of the bit and the bottom value the number of the bit inside the 64 bit sequence.

Figure 4.2 (c) represents the end of the ID packet. The “After ID” label is underlined with a yellow line, which indicates that the detection process is complete. The user will know whether it was a valid ID, if the “ID detected” label’s count value is incremented. The binary string of the 64 bit ID is displayed next to the “Complete Tag” label.

The minimum and maximum values amongst the 420 samples displayed at a time are displayed below the “Min and Max Value” label. On the right hand side of the screen two labels can be found “--Diff” and “--Errors”. In line with these values are numbers in every bit. The “--Diff” label represents the energy difference between the third quarter and the first quarter. The “--Errors” label represents the number of undefined bits detected thus far in the specific detection sequence.

The simulation environment provides the user with the ability to change the algorithm’s detection constants (refer back to chapter 3 for the definition of the different constants and their corresponding functionalities in the Algorithm II). Figure 4.3 is a screenshot of the constant adjustment window.

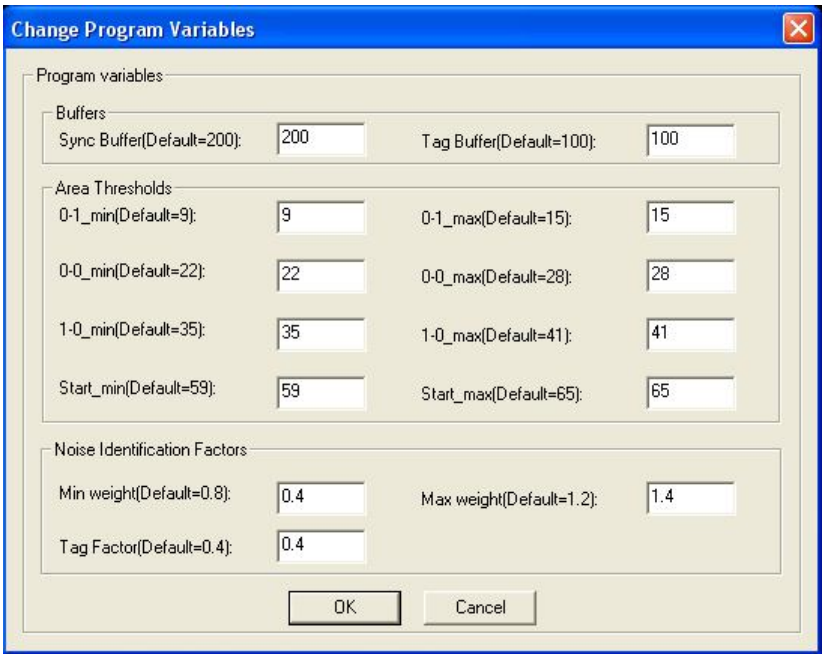


Figure 4.3 GrafiekE parameter change function

4.3.2 Errors present in "GrafiekE"

The purpose of the simulation program is to aid the designer during the developmental phase of the algorithm. In order for the simulation tool to be useful it must be error free. However, the simulation program, "GrafiekE" exhibits a number of errors. Some of the errors are semantics and others affect the algorithm directly.

The graphical user interface of "GrafiekE" turns white when the minimize function is used. There are also a number of functionalities/menu selections provided by the simulation environment which is not implemented. Examples of the latter are the save and edit commands.

As mentioned previously, the program displays 420 samples at a time. The vertical scale and amplitude values of the consecutive subsets are displayed with different ranges, by the program. Combing the latter with the fact that the vertical axis is also not indexed, makes the signals displayed very elusive and difficult to interpret

The binary format of the ID displayed next to the "Complete Tag" label is not cleared before a new detection sequence starts. This error can easily be corrected and falls in the semantics category.

Given that the simulation program was originally designed to be used by its creator; preventative/foolproof measurements were not implemented. An example of this is the algorithm “run” functionality which is available even before any data was loaded, which when selected corresponds to a program error and the simulation program closes.

The quarter bit lines displayed do not distinguish between the four quarters (Consult chapter 3 on the encoding scheme used in this project). Only alternating blue and purple vertical dash lines are displayed. The readability of the different quarters is reduced by this fact.

The errors in the following paragraphs directly affect the algorithm. The program throws an exception when some of the data sets are used. This exception results in the program closing. During this stage of development the origin of this error was unknown. But it is safe to assume that if it originates in the algorithm itself it will also affect the embedded implementation of the algorithm and its performance.

The detection threshold displayed by the simulation program, the horizontal blue line, is sometimes not displayed at all. There were two possible reasons for this, either a display error or an algorithm calculation error which resulted in displaying the threshold outside of the display range.

Of the 420 samples, the minimum and maximum values are not always displayed correctly. This adds to the vertical axis problem mentioned previously. The minimum value always persists. Another issue to be dealt with is the fact that the maximum value sometimes exceeds the maximum range. A 12-bit ADC was used to obtain the data sets. Thus a maximum digital value of 4095 per sample is possible. The algorithm uses a five point filter. So the maximum range should never exceed 20475, which it does.

4.3.3 Limitations

Although the simulation program “GrafiekE” is a helpful tool it comes at a price. The program executes the algorithm on 420 samples at a time. Furthermore the output of

the algorithm on the data is displayed with an animated function. The animation function refers to the simulation's "run" function discussed previously. This function animates the output on the current filtered samples displayed. These two factors make this a time consuming process, because each data set consist of ± 500 000 samples.

The program does not have a "scroll" functionality. Thus if the exercise of stepping through the ± 500 000 samples, 420 with each mouse click is complete, the exercise must be repeated to view any previous sections. If the desired position in the set is known the user can use the sample point functionality. This functionality provides the user with the ability to skip to certain parts in the data set. However the algorithm is depended on previous data, for example for its threshold calculations, thus by advancing or going back in a specific data set, the software will not always provide an accurate output.

The detected ID value is displayed in binary format. The ID as mentioned previously is 64 bits long, thus it is difficult to compare 2 ID values with each other in this format. A more readable format is hexadecimal. In binary format the ID is a 64 bit number and in hexadecimal it is a 16 bit number. Comparison between the simulation and embedded system will also be easier, because the middleware already displays the IDs in hexadecimal format.(Refer to chapter 1 for the definition of middleware and see next chapter for an example of middleware for this project.

As mentioned previously RF signal strengths (the power of the transmitted RF signal) decrease exponentially when the distance between the tag and the reader increase. Thus the signals originating from tags further from the reader will be weaker than those closer to the reader. The simulation program provides the user with only a linear vertical axis scale and no zoom functionality. The readability of the weak signals captured from tags further from the reader tends to be difficult.

The effectiveness of the algorithm can be tested by the number of tags it detects out of the possible tags contained in a data set. A more realistic indication of the algorithm's detection capabilities can be obtained if the tests were repeated on numerous data sets. This can be used to see whether a change in the algorithm

increases or decreases its detection capabilities. The simulator “GrafiekE” makes this a time consuming process, because of its display method of the detected IDs.

4.3.4 Conclusion

The simulation program is a useful tool for the designer, because it gives a visual indication of the algorithm’s output on a data set. Other information can also be extracted. The latter include the detection capabilities of the algorithm. However the program has a number of errors and limitations as discussed earlier. This tool can be used more effectively and be complementary to the design process if these errors and limitations are eliminated.

4.4 New Simulation Software (Simulator)

4.4.1 Introduction

The name of the new simulation program is “Simulator”. The development of the new simulation program aimed to eliminate the errors and limitations of the old simulation program “GrafiekE”. The new simulation program could then better aid the designer to debug the algorithm and make improvements.

A secondary objective of the new simulation program, which was more relevant to this project, was to separate the algorithm from the display code. This later proved to be helpful with the implementation of the algorithm in VHDL and made it easier to implement improvements made to the algorithm (See section 4.4.2 for reasons).

The design process of “Simulator” comprised of three parts. Firstly the algorithm was extracted from the “GrafiekE” simulation program’s code. Secondly the graphical side of the simulation program’s functionalities were addressed. Lastly the interface method used and the problems between the algorithm’s code and the display code were addressed.

4.4.2 Algorithm extraction

Phase V, (refer to chapter 2 and Appendix G) showed that the algorithm was very computational intensive. [11] For this reason the algorithm was written in C++. C++ is

known to be optimised for mathematical calculations and thus the algorithm was kept in the C++ compiler language.

As mentioned previously the algorithm's (Algorithm II) C++ code was embedded in "GrafiekE"'s display code. The code of "GrafiekE" was furthermore not divided into different functional specific sub-sets. The code's readability was reduced by this fact and the identification of the section of code, which represented the algorithm, was difficult to be distinguished.

As mentioned in chapter 3, Algorithm II can be divided into five parts. Each part performs a different task. Thus by implementing each one of the algorithm's five parts in a different function, this made optimization of a single aspect of the algorithm easier. This also proved to be helpful when the algorithm was implemented in VHDL, refer to chapter 7. See figure 4.5 for the categorization of the parts.

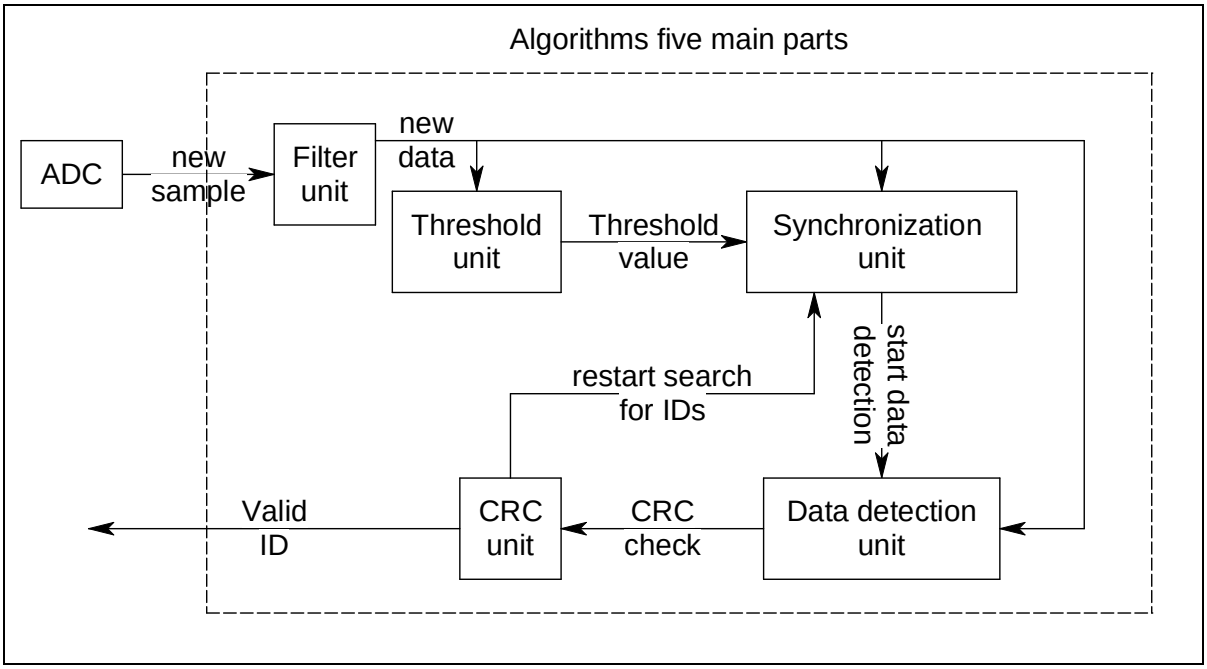


Figure 4.4 Algorithm's five main parts

Description of each unit's functionality (see chapter 3 for details):

- *Filter unit:* House the five point average filter.

- *Threshold unit:* Determines the threshold for every hundred samples (see chapter 3 and reference [16] for constant details).
- *Synchronization unit:* Detects whether possible IDs are present by detecting the eight synchronization bits, the 2 no – transmission bits and the start bit (refer to chapter 3 for ID packed structure).
- *Data detection unit:* Detects the ID data after a valid synchronization frame was detected by the synchronization unit.
- *CRC unit:* Check whether a valid ID was detected by the data detection unit.

4.4.3 User interface

The graphical part or user interface of the simulation program was written in VB (visual basic). The reason for the choice of program language was because of VB's simplicity of handling string variables and graphical displaying functionalities. The first version of "Simulator"'s user interface was written in Microsoft Visual Studio 2003. The program was later ported to Microsoft Visual Studio 2005, to keep the software up to date. Figure 4.6 is a screenshot of "Simulator".

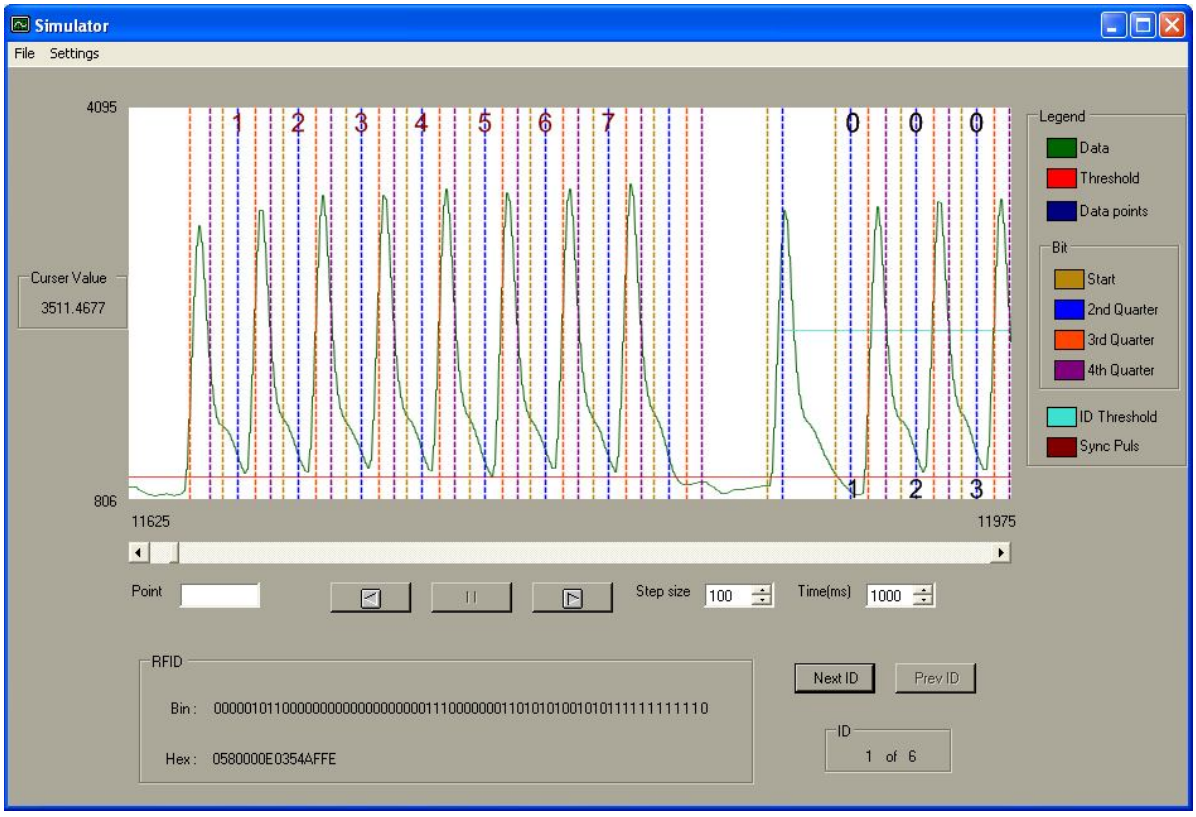


Figure 4.5 Simulator

“Simulator” still uses the same format of data files as the previous simulator, namely .csv files and .txt files. A major difference in “Simulator” compared to “GrafiekE” is that the algorithm is executed on the complete data set loaded and the output is stored. The relevant data is extracted from the variables, which contain the algorithm’s output on the data set, and displayed by the program, as the user uses the different functionalities of the program.

Simulator displays 350 samples together with the algorithms output per visible frame. The value of 350 was chosen because of the development PC display screen’s size and to achieve an easy readable window layout.

The user has a variety of functions available to analyze the algorithm’s output.

- A scroll function is available with a variable step size, to easily examine the output of the algorithm.
- The user can also enter a point in the data set to jump to, just like the previous simulator, without affecting the output of the algorithm.
- A video function is also available, with a play function in both directions and a pause function. The speed of the play function can be varied and the step size of each screen is the same as the scroll function’s step size. The video function enables the user to automatically scroll through the data.

The vertical axis range as well as the horizontal axis range is displayed. The vertical axis displays the 12 – bit ADC amplitude values and the horizontal axis displays the sample number (the horizontal axis can also be interpreted as a time axis, because each sample corresponds to a certain time elapsed, the inverse of the sample frequency). The user is able to read any amplitude (vertical axis) value of a signal displayed by moving the mouse cursor to the desired location. The amplitude value is displayed on the left-hand side of the display screen (See figure 4.5, “Cursor Value” groupbox for an example)

“Simulator” displays the total number of possible IDs detected as well as the total number of valid IDs in a data set, as displayed in figure 4.5 in the “ID” groupbox. With the “Next” and “Previous ID” function buttons the user can skip between possible ID’s

detected, in other words, jump between possible ID's. The values of the ID's detected are displayed in binary and hexadecimal format. The user can also request a summary report, which will show how many times a certain/single ID was detected in the data set.

"Simulator" provides the user with three different vertical axis scales. The default is a linear scale. Logarithmic and decibels scales are also available. The latter two scales are not completely correct. The range of the data is from 0 to 4095, because it is a 12 – bit ADC. But logarithmic value of zero is undefined. The program handles this exception by keeping the value zero.

The default scale range of the vertical axis is set to the minimum and maximum values of the data set loaded. The user is allowed the freedom to change these values if necessary. An example of where this function may be useful is when the tag data is very small and noise is present in the system. It is almost represents a typical zoom function.

"GrafiekE" provided the user with the functionality to change the constants of the algorithm, this functionality has also been carried over to "Simulator". The difference is that after the constants are changed the algorithm is re-executed on the complete data set. This enables the user to immediately evaluate the effect the changes had on the detection capability of the algorithm with the new algorithm constants.

4.4.4 Interface

As mentioned in the previous section "Simulator" was written in two different languages. The reason was to utilize each language's strong points for the specific section implemented in the simulator software. In order to be able to interface the two sections, the algorithm part was compiled to a DLL (dynamic link library).

The DLL was first implemented with all the display variables declared in the DLL. The only data send to the DLL was the loaded data set. The VB part of the simulator could then access the stored data through functions in the DLL. The end result of this method was a sluggish and memory intensive simulation program.

The method was later revised and re-designed. The variables were now declared in the VB section and only pointers to the memory locations were passed to the DLL. However this method presented a problem. Different languages declare variables' memory locations differently. This does not present a problem when a one dimensional array is declared, but for multidimensional arrays it does.

Through testing it was found that VB stores multidimensional arrays row-wise and C++ does the same declaration but column-wise, for integer arrays. To overcome the problem the memory location of the first element of the array is sent to the DLL and received by a pointer. The value of the specific memory location is then found for indexing starting at zero, with the following formula (see figure 4.7):

(,) = (∗ ( ) +)

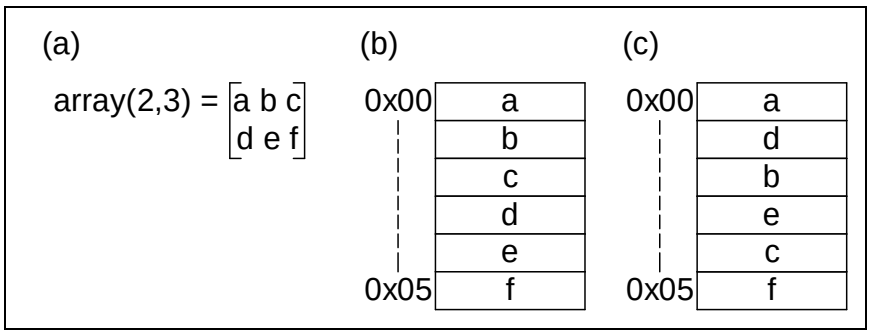


Figure 4.6 (a) array declared, (b) row-wise in memory and (c) column-wise in memory

Not all of the variables are the same in different languages. VB does not contain "Character" variables like C++, but it has "String" variables. A C++ "String" variable is a structure and differs immensely from a VB "String" variable. To overcome the problem of sending characters between the two languages, integer variables were used. The character's integer value is stored in the variables. The value in the memory location is the same, it is just a matter of interpretation.

4.4.5 Conclusion

The end result is that the "Simulation" program contains none of the previous simulator ("GrafiekE") errors and limitations. It can now be utilized as a valuable tool in the design process of the final detection algorithm ("Algorithm II"). The fact that the

simulation program was now error free provided a baseline algorithm. After the algorithm was implemented in VHDL, the detection capabilities of the VHDL and C++ implementation could be compared.

5 • Chapter

5 Middleware (Ghost tags)

5.1 Introduction

Chapter 5 will examine the origin of the ghost tags (Tag IDs displayed but not physically present) in the digital RFID reader system. The reasons why the ghost tags are present in the system and the solutions to these problems will also be discussed. But first an overview of the applicable components, which have not been discussed in previous chapters, will be given. Refer back to chapter 1 for a definition of middleware.

The different middleware programs are:

- MTTY, [11]
- MTTY2 and
- Serial Port Monitor.

and will be discussed here.

RFID middleware in this project is represented by a GUI (graphical user interface). The user interface connects to the hardware (digital RFID reader) via a RS232 communication protocol. The middleware in this project is used to display the tag's IDs detected by the RFID reader.

A first attempt at RFID middleware, namely "MTTY" (Mutli-threaded TTY) was undertaken in a previous project by Mr. G Oosthuizen. This middleware will be discussed first after which the improvements made on this middleware will be discussed, which was conducted during this project. [11].

The second attempt at middleware will be referred to as "MTTY2" and will be discussed accordingly. Finally the limitations and errors of the two middleware

programs and the development of the new middleware program “Serial Port Monitor” (Developed during the course of this project) will be discussed.

5.2 Multi-threaded TTY (MTTY)

This middleware implements the complete RS232 protocol. The port settings can be changed via five “ComboBoxes” namely;

- “Port”
- “Baud”
- “Parity”
- “Data Bits”
- “Stop Bits”

Figure 5.1 illustrates how this RFID middleware displays the tags detected from the digital RFID reader (FPGA, previous implementation, refer to chapter 2, appendix G and [11]).

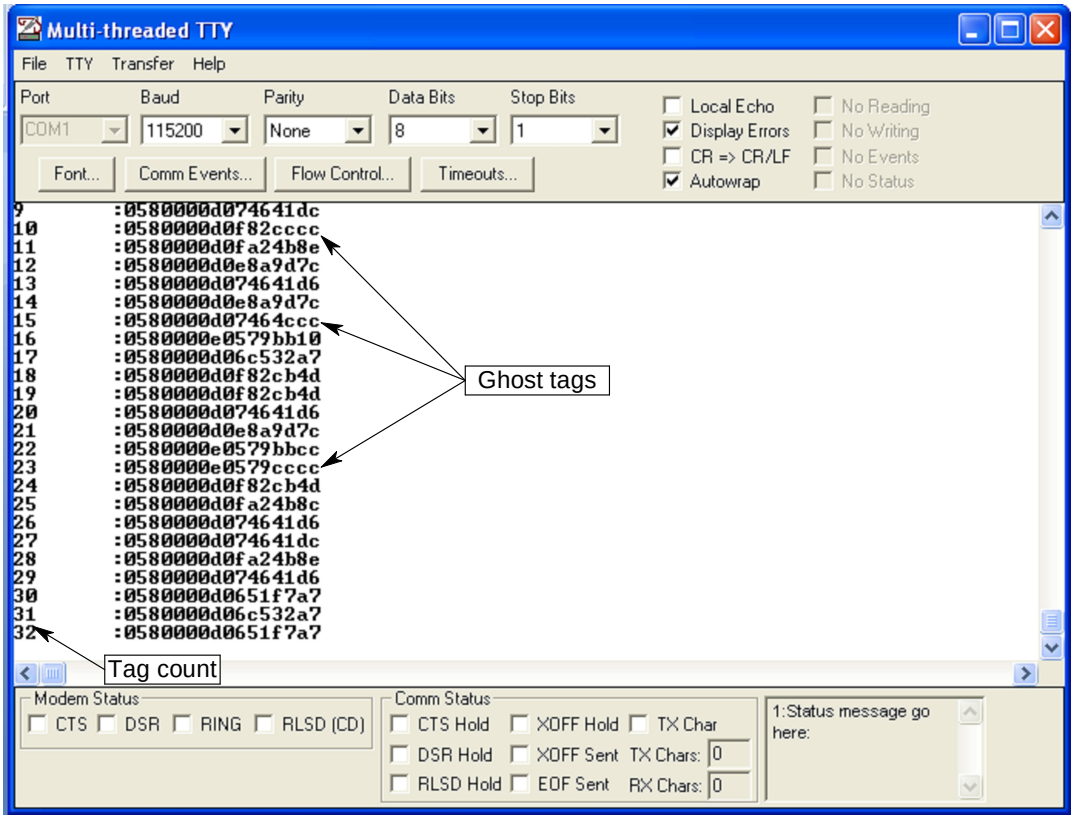


Figure 5.1 Multi-threaded TTY, version 1

The RS232 communication setup used to obtain figure 5.1 was, [11]:

- Baud rate: 115200 bits/sec
- Number of bits: 8 bits
- Parity: None
- Number of stop bits: one
- Number of bytes per tag packet: ten bytes.

Furthermore the algorithm implemented on the digital hardware (FPGA) at this stage where Algorithm II. The reason for this is to remove all the unknowns from the supporting systems before the implementation of Algorithm III, see chapter 6.

The ID tag packet is composed of a header byte, which indicates the start of a tag packet and the data section which contains the 64 bit ID. The ID is split up into eight bytes. The ID packet also contains an end byte. The composition was specified in Phase V. [11] Figure 5.2 illustrates the ID tag packet composition.

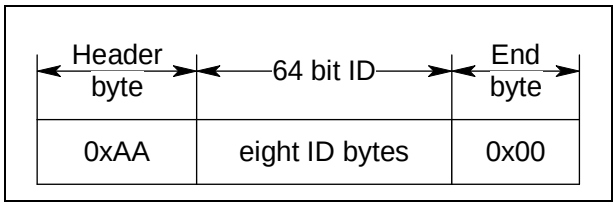


Figure 5.2 ID tag packet composition to RFID middleware

The program displays the detected tags by adding a new line, after receiving a complete ID tag packet, on the display screen. The new line contains the total number of tags detected and the ID received from the RFID reader.

Figure 5.1 displays a number of ghost tags detected during a test. A ghost tag in the system is defined as a tag ID which is invalid or not present but is received from the RFID reader. The ghost tags can in most cases be identified as IDs with a number of consecutive “c” at the end of the ID. The numerous ghost tags detected result in an invalid total number of tags detected.

5.3 Multi-threaded TTY version 2 (MTTY2)

The objective of the second version of “MTTY” was to provide a more accurate count of the number of tags detected. Figure 5.3 is an illustration of the program. The same RS232 communication setup was used as in the previous setup, with the same digital RFID reader.

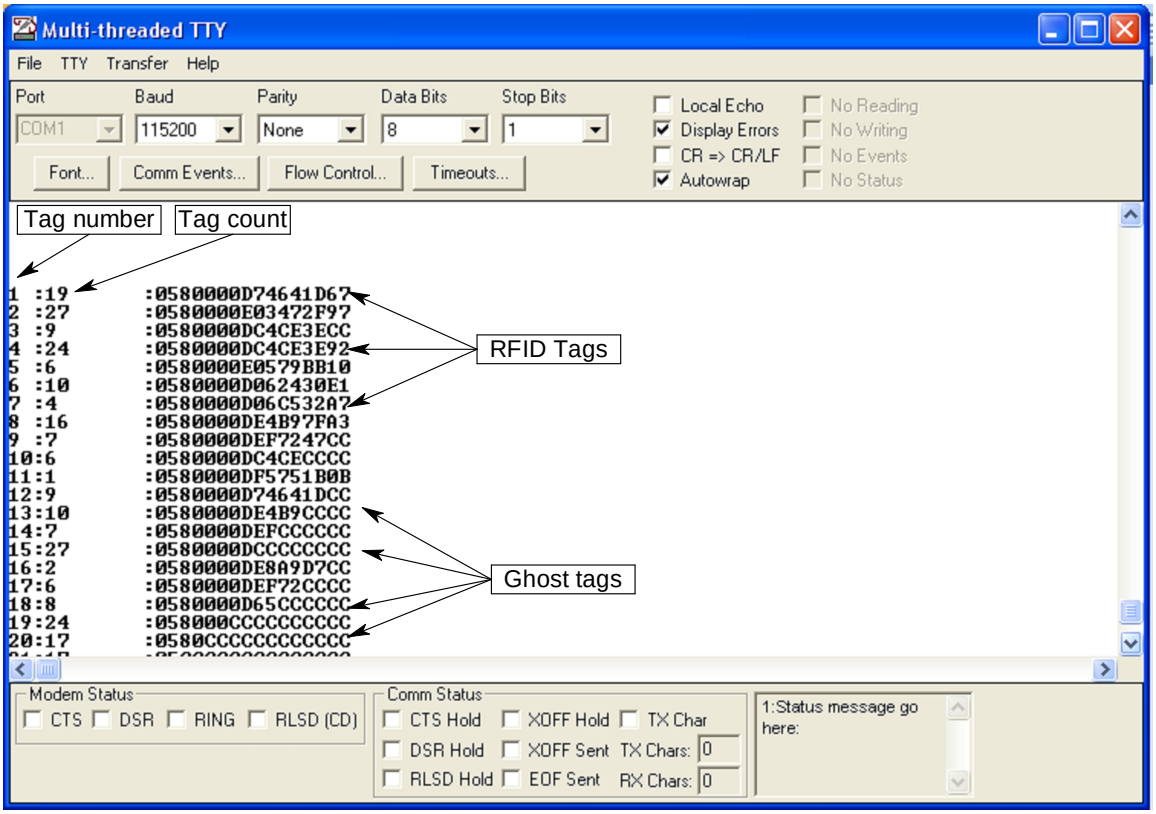


Figure 5.3 Multi-threaded TTY, version 2

The difference between “MTTY” and “MTTY2” is that instead of adding the new detected ID in a new line, the first 32 IDs detected are displayed and the number of times they were detected is updated. The number of IDs displayed, namely 32 was chosen because the middleware is used in a research environment and the maximum number of tags used during any tests was 18. The first column in the display screen represents the number of tags. The second column represents the number of times a tag was detected. The final column represents the ID number of the specific tag detected.

The user can now add all the valid tags counts together to get an indication of the reader's detections capability. The problem as to the origin of the ghost tags was still not solved. In fact, "MTTY2" added a problem; if valid tags were not part of the first 32 tags detected, the tags would not be displayed. This new problem persists due to the numerous ghost tags detected by the system.

5.4 Ghost tags

To identify the origin of the ghost tags three tests were conducted.

- The first was to establish whether the ghost tags originate in the CRC. This was done by programming a constant valid ID into the firmware. Thus no matter what ID was detected the valid ID would have been sent to the CRC. **The test output persisted in displaying ghost tags in the RFID middleware. Therefore it was safe to assume that the ghost tags did not originate in the CRC.**

- Second test :

    - Second test part one. For both sections the firmware transmitted a constant 8 bit value; namely 0xFF. The 8 bit value represents the data section of the ID packet, which is transmitted from the firmware. Thus instead of dividing a 64 bit ID into eight parts, a constant 8 bit value was used. The test objective was to determine whether an error in the transmission could result in ghost tag detection. In the first section, the RFID middleware was used to display the transmitted value. **The detected results still showed ghost tags.**

    - Second test part two. During this stage the RFID middleware was replaced by a serial port monitor program. The object of the second section was to determine whether the ghost tags might originate in the middleware. If the IDs transmitted was received correct by the serial port monitor program, the ghost tags originate in the RFID middleware. **No ghost tags were detected.**

- In the final test a constant ID value was sent with different delays between the IDs, the test used the “MTTY2” middleware. **The results of the test showed that as the delay between the IDs sent decreased the number of ghost tags detected increased.** Therefore it seems as if the origin of the ghost tags lies in the update-module of the display screen. The receive function does not run in a separate thread from the function which updates the display screen. Closer examination revealed that when the middleware returns from the function updating the display, the serial port receiver is out of sync with the serial data stream sent from the reader.

Figure 5.4 is the results of the above mentioned tests.

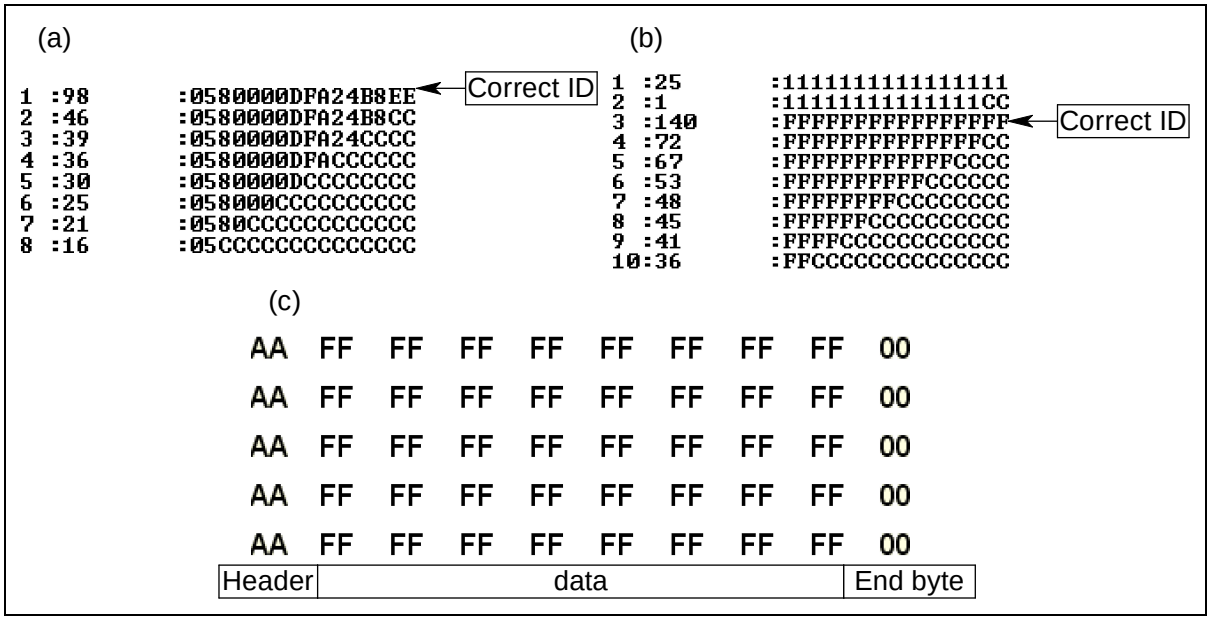


Figure 5.4 (a) Test 1 results, (b) Test 2, section 1 results and (c) Test 2, section 2 results

The above mentioned tests determined that the ghost tags originated in the RFID middleware. Based upon these outcomes it was decided to develop a new program to eliminate the ghost tag problem. Additional disadvantages of the older middleware (“MTTY” and “MTTY2”) included that the software was developed in Visual Studio C++ version 6. One could also argue that a cleaner approach to the solution of the problem will simply be to re-develop the middleware instead of wasting additional time on debugging issues.

5.5 Serial Port Monitor – the new middleware

"Serial Port Monitor" was written in Microsoft Visual Studio 2005 and the language chosen was VB. The program is composed of two forms and a serial port component. The program does not implement the whole RS232 protocol like the "MTTY" programs, rather only the necessary components were implemented. The details of the components implemented are discussed below. Figure 5.5 is a screenshot of the program.

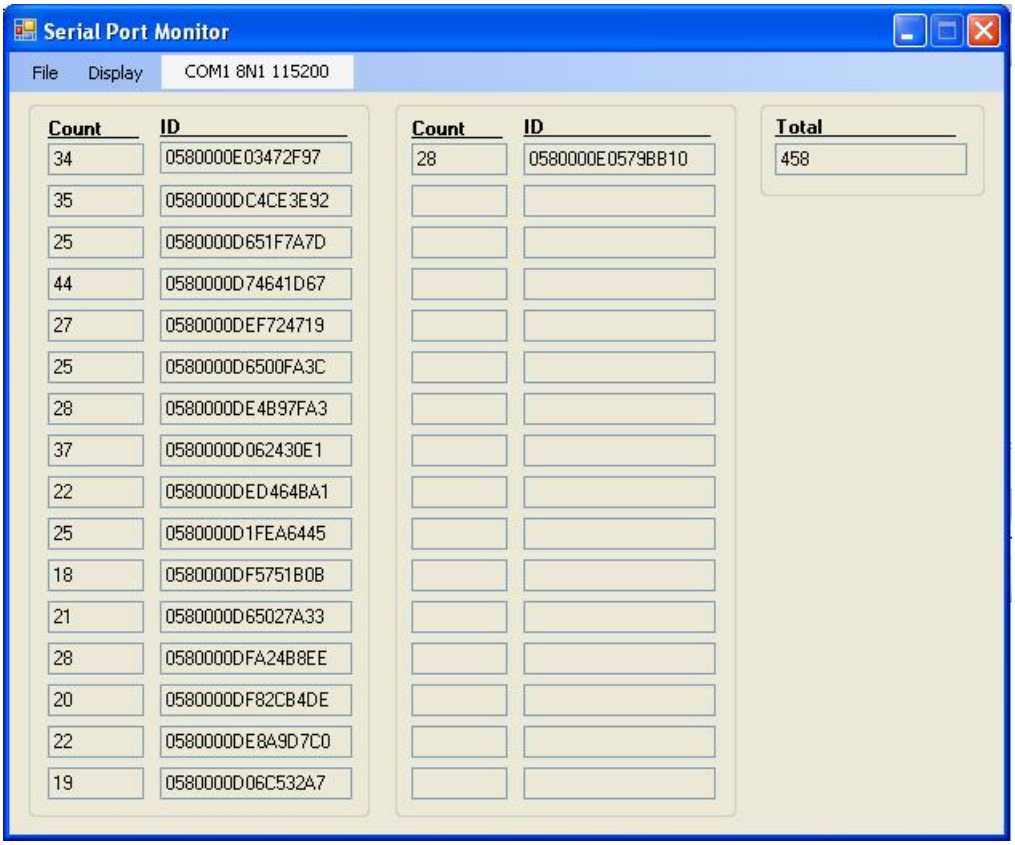


Figure 5.5 Serial Port Monitor, main form

5.5.1 Main window

The program opens in a main window, as displayed in above figure 5.5. The default settings for the serial port are set up as the window loads. The RS232 communication settings were kept the same as those of the "MTTY" programs. There are five components of the RS232 communication protocol which can be changed by the user and four which remains constant; namely

User changeable parameters

- Port name, default on load is COM1.
- Baud rate, default on load is 115200.
- Parity, default on load is none.
- Number of data bits, default on load is 8
- Stop bits, default on load is 1.

Constant parameters

- Handshake is set to none
- Read buffer size is set to 10240 bytes
- Receive bytes threshold is set to 10 (ID packet size)
- Encoding, also use for decoding is set to Unicode charters of 8 bits each.

The default user and constant settings are the settings currently implemented for this project. The receive buffer threshold is set to ten, because of the ID consisting of eight bytes combined with one header and end bytes respectively. Refer to Figure 5.2 and 5.4 (c) for a pictorial representation.

The following functionalities are included in the program. Allows the user to connect or disconnect form the specified serial port. Provision for feedback to the user is also made namely; reporting whether the serial port is connected, whether the port is available and if the port settings are correctly set up. Current connection status is also displayed. The display of the program can be cleared. The latter will result in a restart of all the ID counters.

5.5.2 Settings

The “Settings” window can only be accessed if the serial port is disconnected. Figure 5.6 is a screenshot of this window. The five user changeable settings for the serial port can be modified in this window. These settings are then used to setup the port when the connect function is called.

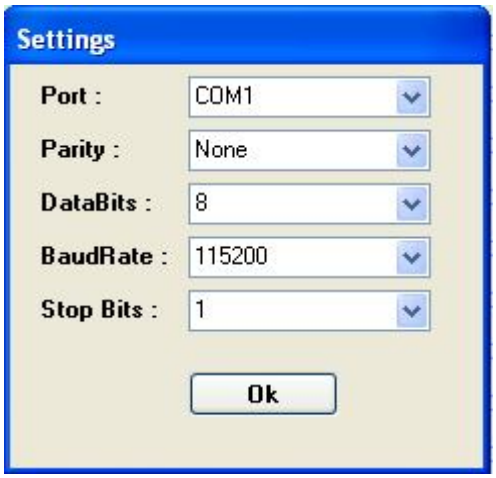


Figure 5.6 Serial Port Monitor, Settings form

5.5.3 Threads

The program runs in two threads. The main program runs on a thread from the start of the program. The serial port component, which is a build-in class of Visual Studio 2005, runs on a separate thread. It handles all the serial port functions and resulting events.

To keep the main window responsive, the serial port component's receive data event is used to detect data in the receive buffer. As mentioned earlier the buffer's threshold is set to ten. Therefore an event is raised when ten or more bytes are in the receive buffer. The ID is then extracted from these ten bytes. A string, representing the hexadecimal value of the ID is passed to the screen update function as a parameter. Because the two functions run in separated threads, an "invoke" method has to be called to pass the string between the two threads. The "invoke" method is used to make a safe cross thread call. The invoke method determines whether the thread can be interrupted (is in a state to handle function calls from other threads), without crashing the program.

5.5.4 Test

The program ("Serial Port Monitor") was tested to determine whether ghost tags are still detected. The results of the tests showed that no ghost tags were detected, irrespective of the number of tags in the reader's range. The number of different tags that can be displayed is limited to 32, as in "MTTY2".

The problem of the ghost tags in the system was successfully solved. The middleware could now be used as a validation tool, with the added benefit that it will now also proof to be more accurate. The new middleware now also played a key role in validating the accuracy of the algorithm's implementations in VHDL.

6 • Chapter

6 Frequency sensitivity

6.1 Introduction

Data frequency in this project refers to the tag's baud rate. The communication link established between the tag and reader, for testing in this project was 128 kbits/s. The baud rate in most systems should be fixed and detection methods should be sensitive to slight variations in the baud rate. An example is RS232 which can be setup for different baud rates.

However, the RFID system tested in this project made use of passive tags. These tags were made as small and inexpensive as possible, thus, their clock generator in most cases comprised of a resistor and capacitor combination. As a result of component tolerances, basic electronic background explains that the clock frequency will drift or result in slight variations from the frequency it was designed for. The bits in the data stream are also known to float. The latter statement refers to bits in the data stream of which the bit *period* is correct, but overlaps with the previous or next bit *position*.

Chapter 6 addresses data frequency deviations by discussing the following topics.

- Why is the sensitivity of Algorithm II an issue? In essence, why does it need to be addressed?
- How Algorithm II overcomes the bit floating phenomena. Addresses the question of why Algorithm II is sensitive for data frequency deviation if it implements the technique to be discussed here to overcome bit floating.
- A possible solution to the sensitivity issues of Algorithm II, referred to as Algorithm III.

6.2 Bit detection method

As mentioned, the data rate for the communication link from the tag to the reader was set up for 128 kbits/s data transfer. The analogue signal received from the reader is sampled at 3.072 MHz, refer to chapter 2 and appendix G for more detail on the sampling frequency. With these specifications in mind theoretically each bit should have 24 samples. The bits are glitch encoded (see chapter 3) which means there are six samples per quarter bit.

Algorithm II uses the number of samples to determine the quarter bit positions, see figure 3.12. This orientation method is impeded by the following two phenomena:

- The bit floating which is present in the bit stream.
- The phase shift introduced by the five point filter in the data bit stream. Figure 6.1 illustrates this effect.

Figure 6.1 (a) shows two logic zero bits followed by a logic one bit. The dots in the figure represent the samples. Figure 6.1 (b) is the output of the five point filter with (a) as input. The phase shift due to filter can be seen in figure 6.1 (c).

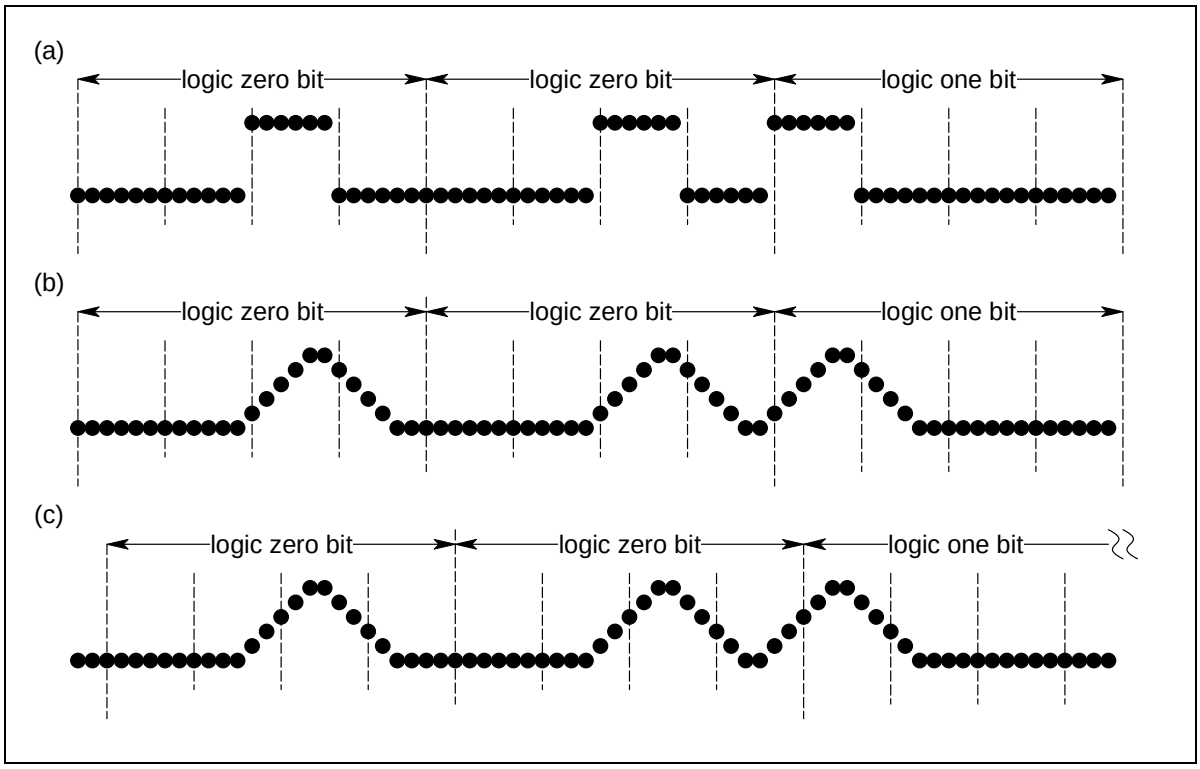


Figure 6.1 Data samples for 128 kbits/s data tempo and 3.072 MHz sampling frequency, (a) Theoretical bits, (b) Filtered bits and (c) Phase shift due to filtering

Algorithm II compensates for these effects with an automatic adjust function. The automatic adjust technique works on a simple principle. Before the signal is manipulated by any filtering, the distance from the centre of a quarter bit to the end of the quarter bit is three samples long. After filtering, the previous statement can be redefined as, “the distance between the *peak value in the first or third quarter bit to the end of the specific quarter bit* is three samples long”. Figure 6.1 (c) illustrates the latter statement visually.

Algorithm II makes great use of the latter fact to adjust its orientation in the next bit. The *sample count* value of the peak value in a quarter bit is stored for referencing purposes.

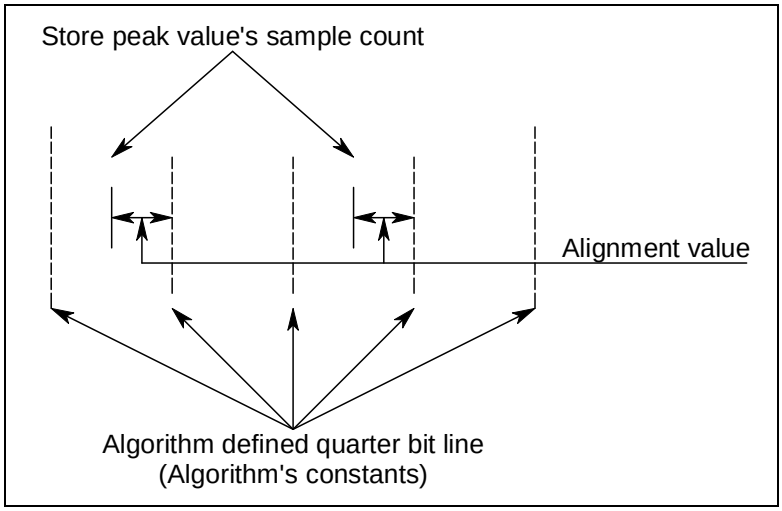


Figure 6.2 Alignment calculations

Figure 6.2 illustrates the calculation of the alignment value. Depending on whether the latter is determined to be a logic one or zero, the peak value *sample count* position of the respective bit value is subtracted from the *sample count* of the end of the specific quarter line (if logic one end of first quarter or if logic zero end of third quarter). Refer back to chapter 3 for the definition of a logic bit. If the alignment value reported is three then the algorithm is aligned correctly (peak value centred).

The constant values (see chapter 3 for detail, or figure 6.2 for a reminder) define the consecutive quarter bit position between bits. The constants are defined to include the three sample differences. For example the distance in samples from the end of

the third quarter to the beginning of the next bit is 6 samples, but the constant value is defined as nine. Thus if the value is less than three the peak value is closer to the end of the quarter bit. The sample counter is either set to two, one or zero. The opposite is also true, if the difference is bigger than three the peak value is nearer to the start of the quarter and the logic simply reverses. Figure 6.3 illustrates the automatic adjustment function visually.

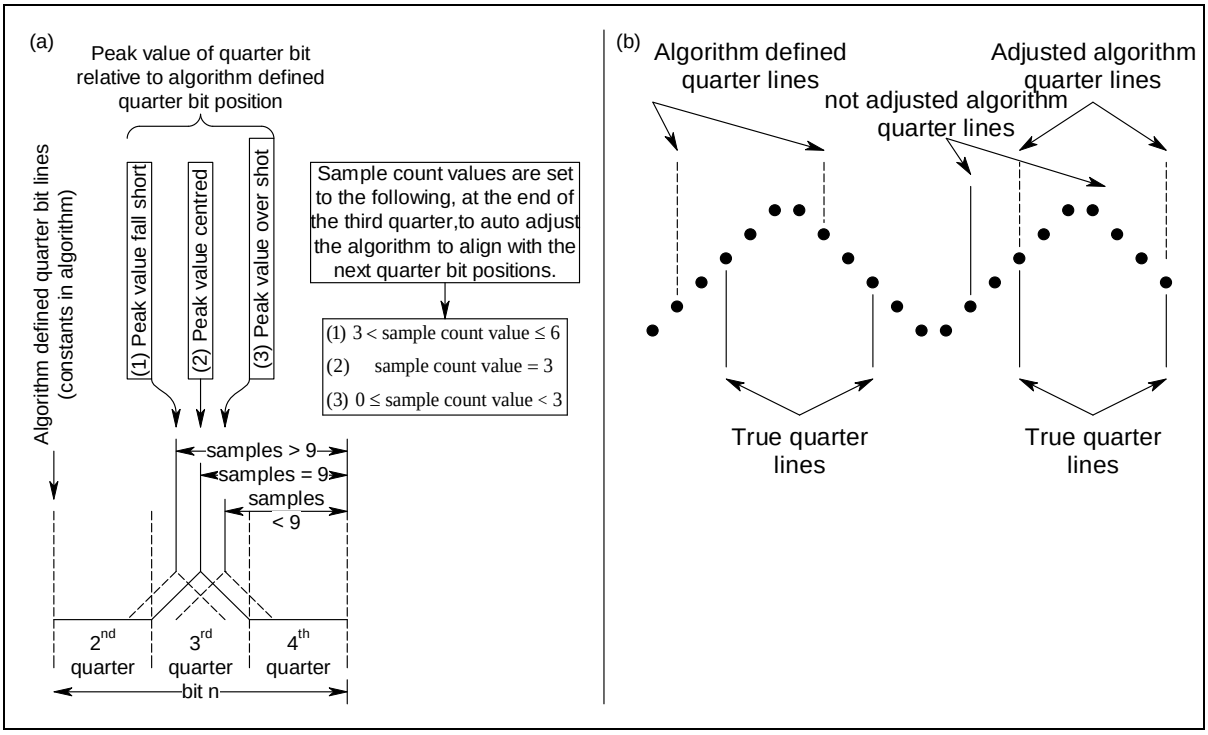


Figure 6.3 Auto adjust technique (a) Calculation of *sample count* value (b) Effect of auto adjust

Even though Algorithm II implements this automated adjusting technique, it still was found to be sensitive for frequency deviation. Mathematics confirm the latter, for a ± 1 sample per bit variation, a 5 kbits/s tolerance around 128 kbits/s was present. A 1 sample deviation is well within the range of the auto adjusting technique and can be corrected. Thus theoretically a detection range from 123 kHz through 133 kHz should be possible.

6.3 Algorithm improvements

The previous paragraph shed some light on why the sensitivity of the algorithm was seen to be a concern. For these reasons Algorithm II had to be examined to find and identify the problems relating to the discussed issues.

Algorithm II uses the first of the eight synchronization bits as a start bit for the synchronization detection stage. The method (technique) used to align the algorithm's defined quarter bit lines (sample counter start position and start value) via this start bit was crucial for the auto adjustment technique to function properly. The latter method thus greatly affected the detection capability of the algorithm. Examination of the algorithm's effectiveness in this start area was conducted with the new simulation tool ("Simulator") and it was found to be inadequate.

Algorithm II's code was examined to track down the source of the inadequacy. The algorithm determines a possible sync sequence by testing whether a sample value is above the threshold or not (refer to chapter 3 for more details on threshold). There after the *sample count* is set to -2 and peak detection is conducted when the sample counter value is between 1 and 5 (this will be the first sync bit's peak value). Thus the algorithm compensates for the filter by searching for the peak value within the last five samples amongst eight samples after a possible detection has occurred. The *sample count* continues and detects the first quarter of the next sync bit, between samples 10 to 14. The third quarter is detected between samples 19 to 23 (See figure 6.3 (a)).

The core of the detection method used, in Algorithm II, is the assumption that a constant distance between the first and second synchronization bits will be present. Thus the auto adjustment technique is only implemented after the detection of the second synchronization bit.

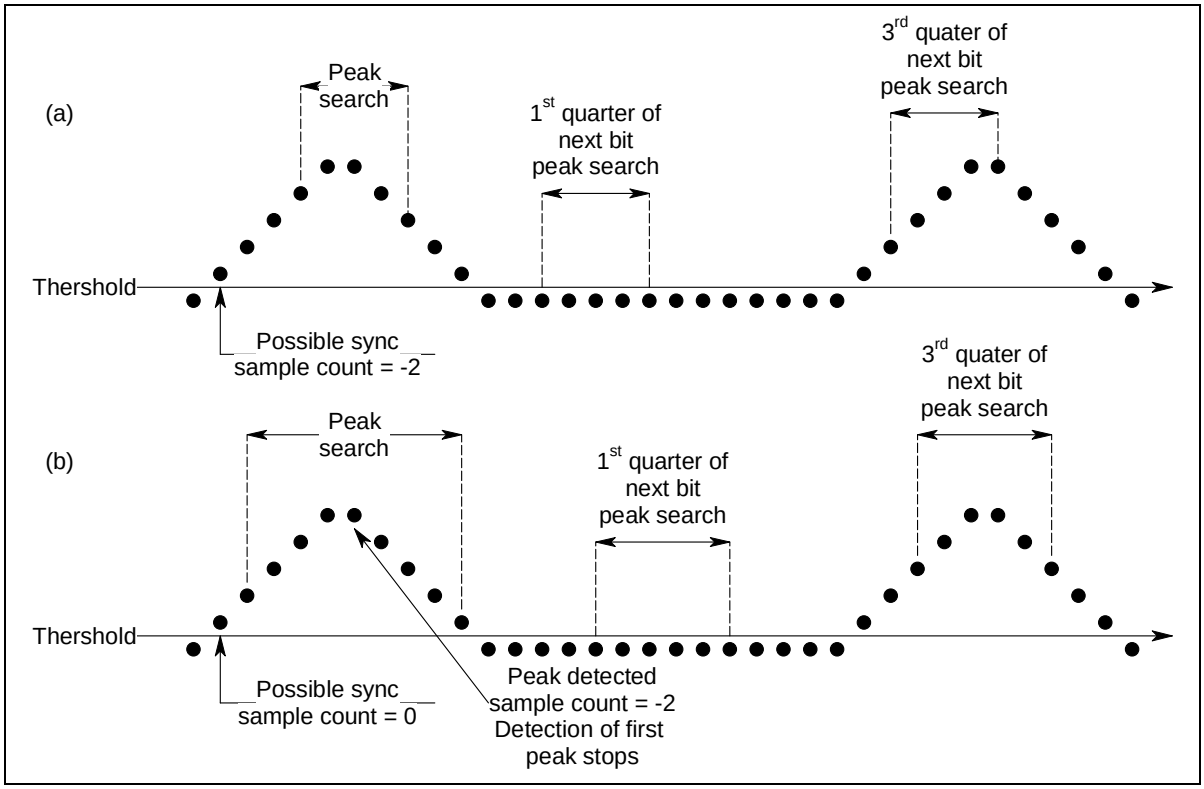


Figure 6.4 Synchronization start methods, (a) Old implementations, (b) New implementation

The new detection technique harnesses the auto adjust technique's principles. Once a possible ID is detected, a sample value above threshold, the sample counter is set to zero. The significance of the value zero is, to let search of the peak value begin at the next sample. Thus it is an arbitrary value which must be one value less than that of the start of the first peak search *sample count*.

The new detection technique searches for the first peak value in a larger sample range than previously. The search range in *sample count* value is 1 to 9. The upper search limit was calculated by taking into account the effect of the five point averaging filter.

Once the peak value is found the algorithm sets the *sample count* to -2 thus implementing the auto adjustment technique (See figure 6.3 (b)). Thus no fixed distance is assumed which makes the algorithm more versatile. The new method improved the algorithm's detection capability with some margin.

The reason why the new method sets the *sample count* to -2 at the peak, instead of 0, is because the detection parameters were redefined. The peak value in the quarter bit is -3 in *sample counts*. The distance between the quarter bits are thus defined as six samples instead of nine. By understanding the latter it should be clear that both are the same.

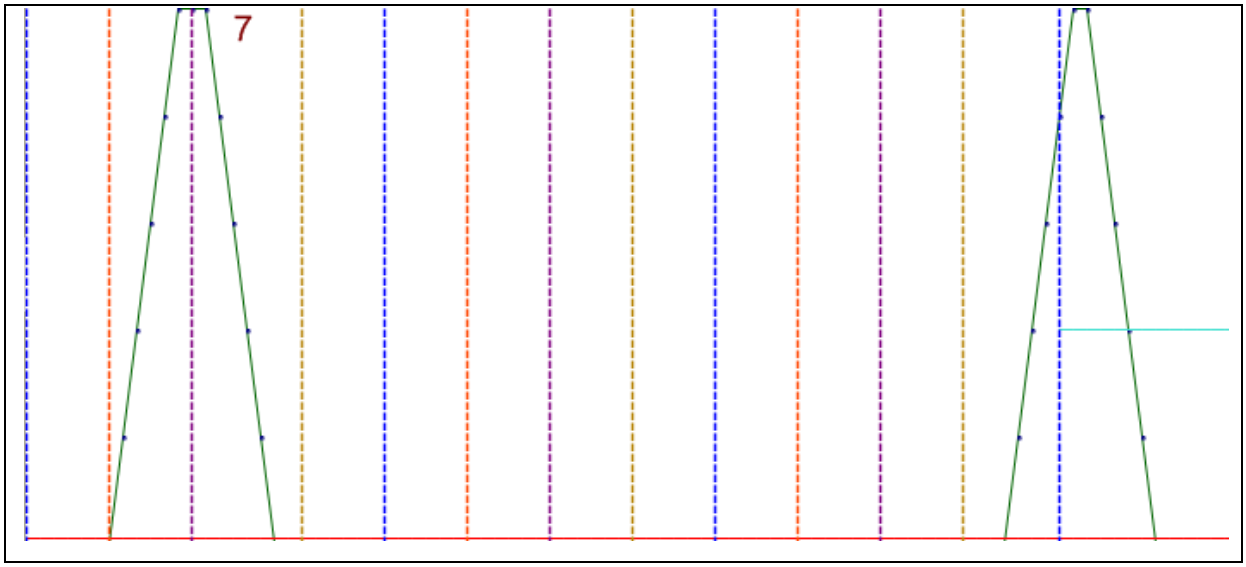


Figure 6.5 Last synchronization bit to start bit problem

A second pitfall for the auto adjustment technique was identified, which contributed to Algorithm II's sensitivity for data tempo deviation. Figure 6.5 is illustrative of the problem. It occurs between the last synchronization bit and the logical one start bit. Between the last synchronization bit and the start bit, two 'no-transmission' bits are present. The auto adjustment technique cannot be used here because there are no bits for two bit periods to align the algorithm. Algorithm II overcame the problem by defining special constants for the no-transmission part of the ID packet. The constants were defined for a 128 kbits/s data tempo and for consecutive bits. Thus a deviation of ± 5 kbits/s translates to a ± 1 sample deviation. But in the no-transmission section it translates to a ± 2 sample deviation.

Figure 6.4 is an illustration of a 119 kbits/s theoretical ID packet, with the last sync bit, 2 no-transmission bits and the start bit visible. The ID packet was generated with a Matlab® function. For more detail specifications and information see Appendix C.

From figure 6.5 the auto adjustment function's attempt to align the algorithm can be seen. But due to the low data tempo the start bit was almost missed. Algorithm II determines whether the start bit is valid by implementing a method testing whether the peak value is greater than a certain factor of the first sync bit's peak value. If this method classifies the start bit as invalid the algorithm would start a new search for an ID packet.

Algorithm II uses an ID threshold in the data bits to determine if the bits are valid before determining whether it is a logical one or zero bit [16]. This ID threshold is calculated by using the peak values of the sync bits. Thus a sample or peak value above the threshold can be seen as a valid glitch bit. Instead of using a factor to define a valid start bit, the ID threshold is used. The new method would classify the start bit as depicted in figure 6.5 as valid.

Figure 6.6 illustrates the effect of a ± 5 kbits/s data tempo deviation. The effect of the ± 1 sample per bit is clearly visible over the period of the 8 sync bits. Without the auto adjustment technique the algorithm would definitely not be able to compensate for data tempo deviations.

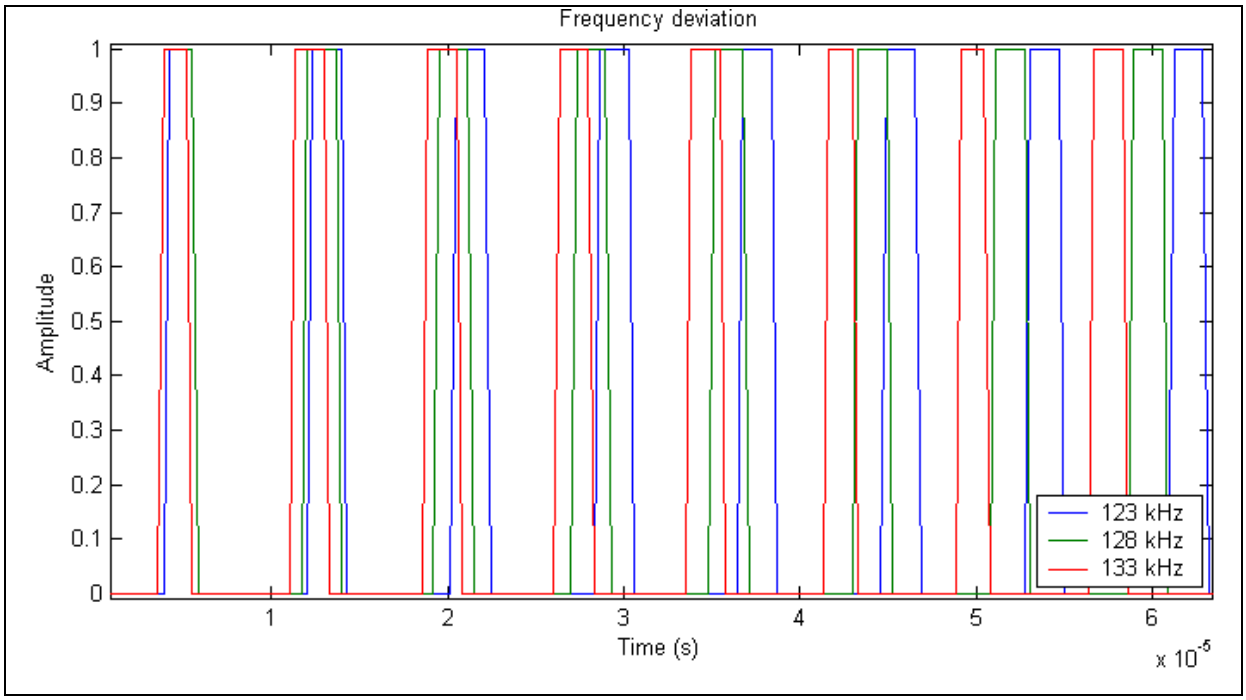


Figure 6.6 Affect of frequency deviation on synchronization bits

6.4 Test

To get a clear indication of whether these improvements would in fact make Algorithm II less sensitive for frequency deviation the following test was conducted. With a Matlab® function (see appendix C) data sets were generated with 200 IDs per data set. All the IDs in a data set had the same data tempo. The range of data tempos examined varied from 112 kbits/s to 141 kbits/s. The test was repeated four times and the average taken as final result. The result of the test is given in figure 6.7.

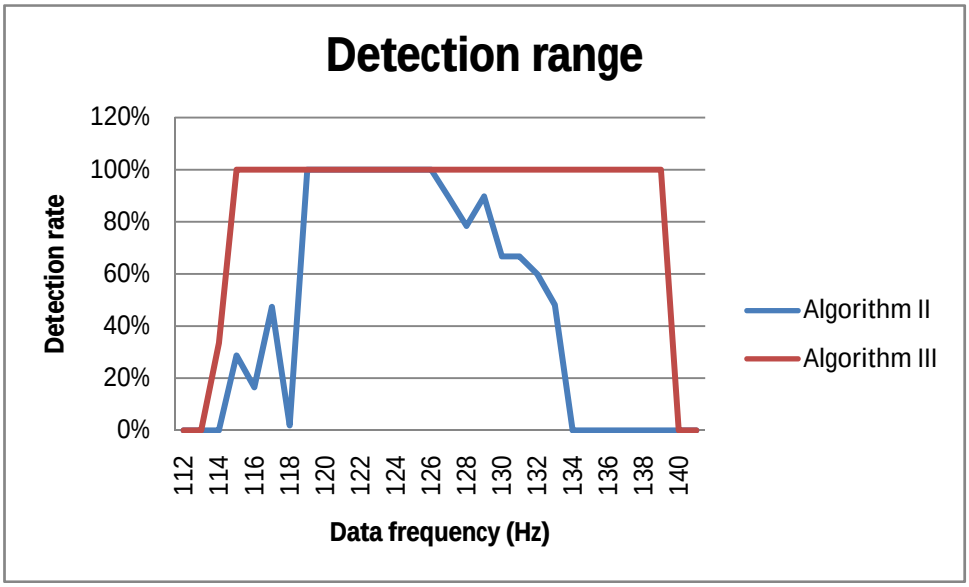


Figure 6.7 Algorithm's ability to handle frequency deviation

6.5 Conclusion

The results indicated that the detection capabilities of Algorithm II were centred around 122.5 kbits/s. Algorithm II had a tolerance of 3.5 kbits/s on the centre data tempo, for hundred percent detection rates. This translates to a detection range of 7 kbits/s.

The Algorithm III centre detection frequency is 127 kbits/s, which is nearer to the specified data tempo of 128 kbits/s. The Algorithm III has a tolerance of 12 kbits/s around the centre data tempo. This provides a 24 kbits/s detection range. The cut–off frequency of the algorithm is clearer.

Although the data used was theoretical data, it clearly indicated that the mentioned improvements made the algorithm less sensitive for data tempo deviations. The real test will come when the new algorithm will be tested on real data.

7 • Chapter

7 Algorithm III's VHDL implementation

7.1 Introduction

The support software for the RFID reader has now been designed and implemented. The issue surrounding the ghost tags have been addressed and Algorithm II's sensitivity for variation in the data tempo reduced. The final stage before Algorithm III could be tested on the digital hardware was to port the Algorithm from C++ to VHDL, in order to implement it on the FPGA, driving the hardware. The ported VHDL should of course yield the same result as its C++ equivalent in order to satisfy the last requirement of the project.

Before wandering off into the porting of the software, the basics of VHDL will be discussed. This will provide readers not familiar with the language with insight into certain implementation decisions.

Thereafter the implementation of Algorithm III in VHDL will be discussed. The support *entities* of the algorithm will also be discussed. These *entities* are the algorithm's interfaces to the outside world.

The following chapter will focus on the accuracy of this VHDL implementation and Algorithm III's detection rate, compared to the analogue reader's.

7.2 VHDL basics

Programming languages, for example C and C++ are generally compiled to a sequential executing sequence, in other words one operation at a time. Contrary to this sequential paradigm, the VHDL programming language is a parallel execution process. VHDL, as its name implies is descriptive of hardware or a digital circuit. The functional components of such a digital circuit operate concurrently. [6]

The building blocks for these circuits (VHDL descriptive circuits) are *entities*. These *entities* are self-contained and independent. An *entity* does not contain any information of another *entity*. Furthermore VHDL describes the connectivity or connection wires between these *entities*.. [6]

Figure 7.1 (a) illustrates the syntax for an *entity* declaration in VHDL. The specific *entity* describes the inputs and outputs of certain *components* of a digital circuit. [6], [29]

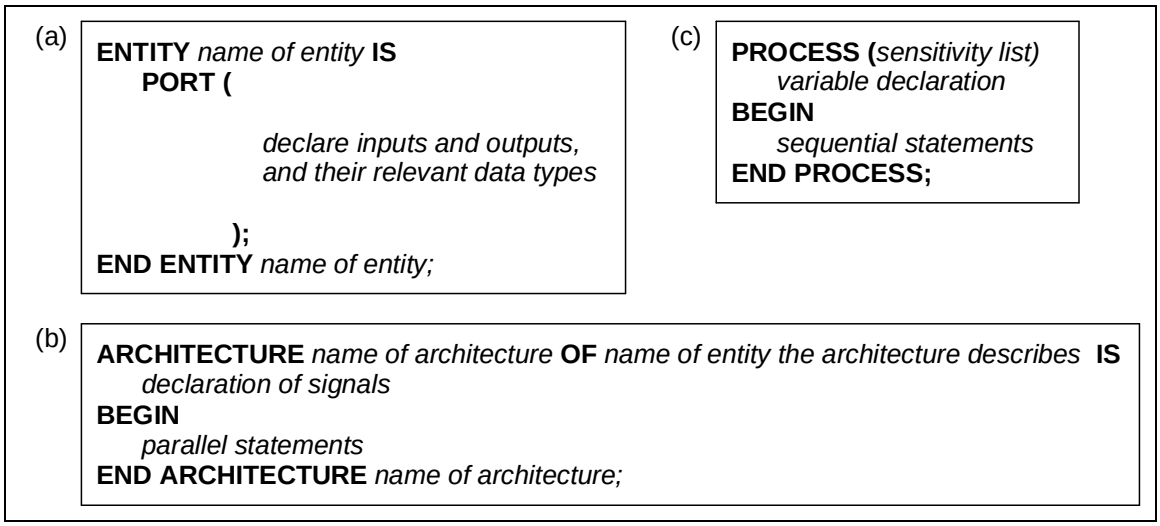


Figure 7.1 VHDL syntax's basic components [6],[29]

The internal circuitry of an *entity* is described in the *architecture* statement of an *entity*. Statements in the *architecture* are executed in parallel, thus the sequence of these statements are irrelevant. Figure 7.1 (b) illustrates the syntax for an *architecture* of an *entity*. [6], [29]

VHDL provides a means for sequential processing *inside* the *architecture* of an *entity*. Sequential processes are described inside a *process* statement. Figure 7.1 (c) illustrates the syntax for a *process* statement. [6], [29]

The *sensitivity list* (see figure 7.1 (c)) of a *process* is known as the activation *signals* for the *process*. If their values change, the specific *process* is executed. Although statements in a *process* are executed sequentially, different *processes* in an *architecture* are executed in parallel. [6], [29]

VHDL provides two means for data storage. The first is known as *signals*. To put it simply, *signals* are nothing else than connection wires. The second type of data storage is known as *variables. Variables* are references to memory locations and can only be used inside a *process*. These two types of data storage exhibit different behaviour inside a *process*. The values of *variables* change immediately after a new value has been assigned. Contrary to the latter, the value of *signals* change at the end of a *process* when the entire *process* has executed. [6], [29]

VHDL has a number of data types. In this project the following data types have been used: [6], [29]

- *STD_LOGIC:* a standard logical bit, thus it can either be logic zero or logic one.
- *STD_LOGIC_VECTOR:* a one – dimensional array of standard logical bits. Its length is described by a clause, for example *(15 DOWNTO 0)* is a 16 bit array.
- *INTEGER:* a decimal data type. The size of the object is defined by a clause, for example *RANGE -15 TO 200.*
- *CONSTANT:* a constant value.
- *TYPE:* used to define custom data types. This is useful for a finite state machine, example *TYPE new data IS (IDLE, DO_SOMETHING_1 , DO_SOMTHING_2)*

7.3 VHDL implementation of algorithm

In order to implement Algorithm III in VHDL, the software was divided into five functional components (See figure 7.2).

- *Filter entity*: This unit houses the five point filter.
- *Limit entity*: This unit determines the algorithm's activation threshold.
- *Sync detect entity*: This unit detects the 8 synchronization bits, 2 no – transmission bits and the start bit. (See chapter 3 for details of the ID packet)
- *ID detect entity*: This unit detects the 64 bit ID.
- *CRC entity*: This unit performs the CRC check on the detected ID.

Figure 7.2 provides an oversight of all the VHDL *entities*, their connectivity and interfaces to external hardware. Detailed discussions of each *entity* will follow in the next few sections.

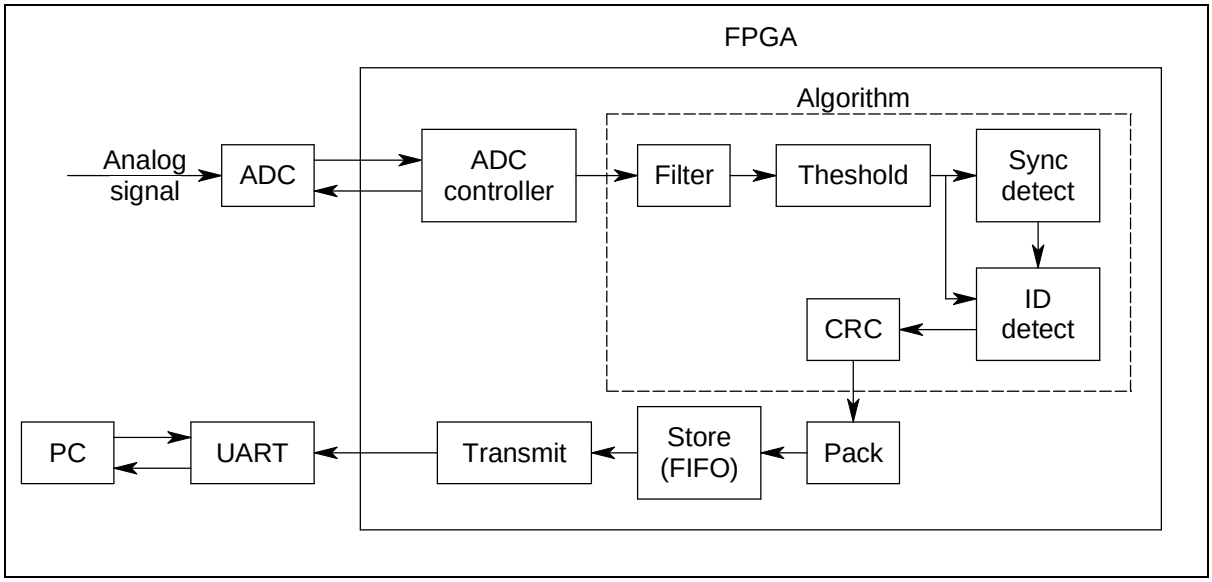


Figure 7.2 VHDL, algorithm entities, supporting entities and external hardware

As mentioned previously, VHDL's *entities* execute in parallel. *Processes* in or across *entities* can be executed synchronously or asynchronously. In order for *processes* to be executed synchronously, a synchronisation clock has to be generated and added to the *sensitivity list* of the *process*.

Have a look at figure 7.2 for example. All the indicated *entities* in this figure will execute in parallel. The system in this project has been designed as a synchronous system. This improves synchronization between *entities* and accommodates the time dependent functions of the algorithm and its support systems.

7.3.1 Filter entity

Figure 7.3 (a) depicts the in and outputs of the *filter entity*. The *system clock signal* is an external clock which synchronises the FPGA's *processes*. The clock frequency is 18.432 MHz. This frequency was found to be sufficient to execute the algorithm, taking into account the parallel processing capabilities of the FPGA.

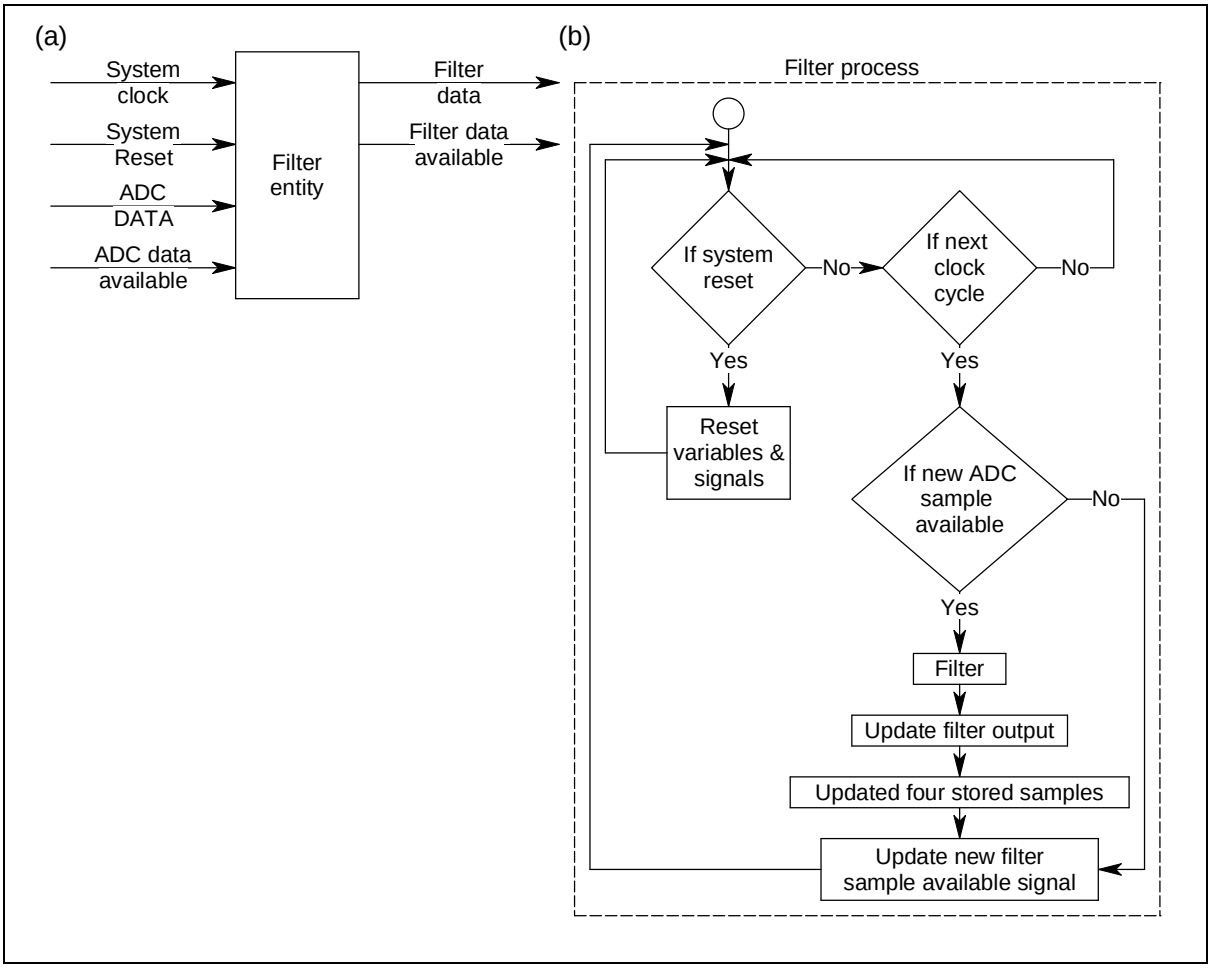


Figure 7.3 Filter entity basic design

The *system reset signal* as indicated in figure 7.3 is an external hardware reset. It enables the user to manually reset the RFID digital reader. The *ADC data signal* is the 12 – bit parallel data interface to the ADC unit (see support *entities* section). The *ADC data available signal* is generated in the *ADC entity* and indicates when a new sample is available.

The *filter data signal* is representative of the output of the five point filter. This data *signal* will be used by the rest of the algorithm's *entities*. The *filter data available signal* indicates when data on the filter data bus (output of *filter entity*) is valid.

The *architecture* of the *filter entity* comprises of one *process*. Figure 7.3 (b) is an illustration of the design. The *process* has three *signals* in its *sensitivity list*, *system clock, system reset and ADC data available,* these *signals* triggers the execution of the *process*.

There are four *signals* which store the previous four sample values respectively in the *entity*. The latter objects are declared as *signals* so that they only change their values on the next clock cycle. This eases the coding behind the filter. The *entity also has one variable* in its *process.* This *variable* is used to perform a storage function between sequential statements, temporary store a calculated value. It is declared as a *variable* because its value must change immediately upon new assignment thereof.

7.3.2 Limit entity

Figure 7.4 (a) illustrates the inputs and outputs of the *limit entity*. The first four inputs were already discussed in the previous section. The *Possible ID* input is generated in the *sync detect entity*. This *signal* indicates whether the algorithm is in a possible ID detections mode. The *signal* stops the activation threshold calculations, because the threshold value must not change during an ID detection. The reason for this is, if the threshold calculations continue during ID detection, the threshold value after the ID detection would be a very large value. The possibility then exists that a possible next ID would be missed. The threshold is also used in the synchronization detection *process*, to determine if a valid synchronization bit is detected (for more detail refer to [16]).

The *Activation threshold* is the output after threshold manipulation was performed and is used by the *sync detect entity*. The *Start search signal* indicates to the *sync detect entity* when detection of a possible ID may begin. This forms part of the initialization process, because a threshold value is first available after reception of the first hundred samples.

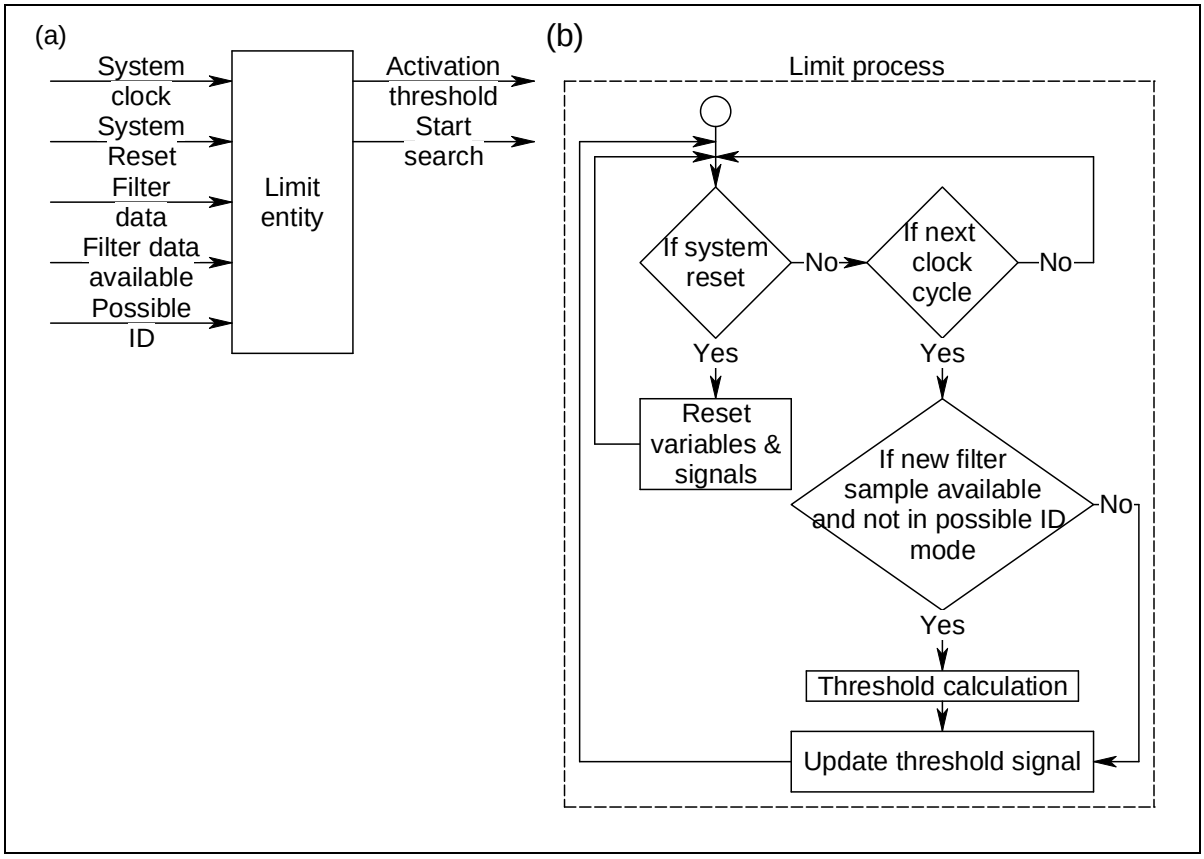


Figure 7.4 Limit entity basic design

The *limit entity* contains one *process* in its *architecture*. Figure 7.4 (b) illustrates the basic design of this *process*. The *process* has three signals in its *sensitivity list*, *system clock, system reset and ADC data available.* These signals trigger the execution of the latter *process*.

All the data types are defined as *variables* in this *entity*. *Variables* were use to ease the calculations of the threshold value. The calculations are based on a sequential process, thus the variable values should change immediately after assignment.

7.3.3 Sync detect entity

Figure 7.5 (a) illustrates the in and outputs of the *sync detect entity*. All the inputs except *Algorithm reset* were already discussed. This input is an internal (FPGA) reset signal. The signal is generated in the *entity RFID detector's process*. (see support *entities*) The *signa*l is used to reset the *sync detect* and *ID detect entities*. The reset occurs when an error occurred in either one of the two latter *entities* or after the 64

bits of a possible ID was detected and its CRC is being validated. Thus detection of a new ID is possible whilst the previously detected ID is being validated.

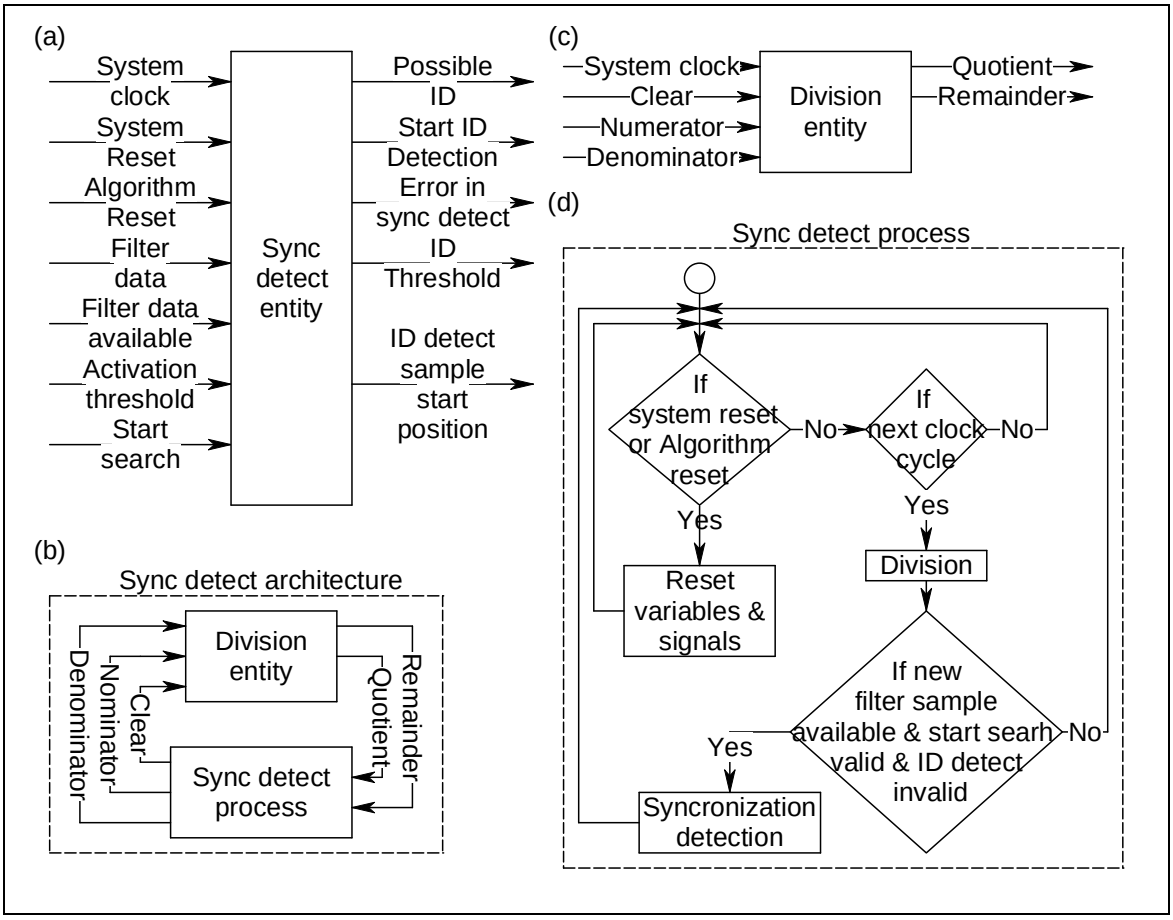


Figure 7.5 Sync detect entity basic design

The first output namely *Possible ID* was already discussed. The *Start ID Detection signal* indicates when the *ID detect entity* should start with the ID detection sequence. This *signal* is generated when a valid synchronization sequence was detected.

The *Error in sync detect* output serves as an indication of an error occurring during the synchronization detection and is also an input to the *RFID detector entity*. This *signal* is used to reset the algorithm when an error occurred in the detection processes and forms part of any one of three possible sources which can reset the algorithm internally.

The *ID Threshold* value is used in the ID detection section of the algorithm. This code resides in the *ID detect entity* and is used to determine whether a specific bit is valid. More (precisely) importantly it is used to validate the energy level of a bit. The calculation of the value is problematic in VHDL. The equation (see reference [16] for equation details) has a divider part and standard IEEE libraries' division statement takes too long. (The calculations do not fit between ADC samples). To fit the calculation of the ID Threshold value between ADC samples and get an accurate value, one of Altera's *entities* namely their "division entity" has been implemented. This *entity* will be treated as a black box as an in depth description thereof falls outside the scope of this dissertation. Internal signals connect the *division entity* to the *sync detect process* as illustrated in figure 7.5 (b).

Figure 7.5 (b) simply shows the common linkage *signals* between the *sync detect process* and the *division entity*. The *system clock* and *system reset* signals are also connected to the *division entity*. Figure 7.5 (c) illustrates all the inputs and outputs of the *division entity*. The *system clock* input is connected to the system clock and the *clear* input is connected to the system reset. The latter was necessary in order to synchronise the *division entity* with the rest of the *entities*.

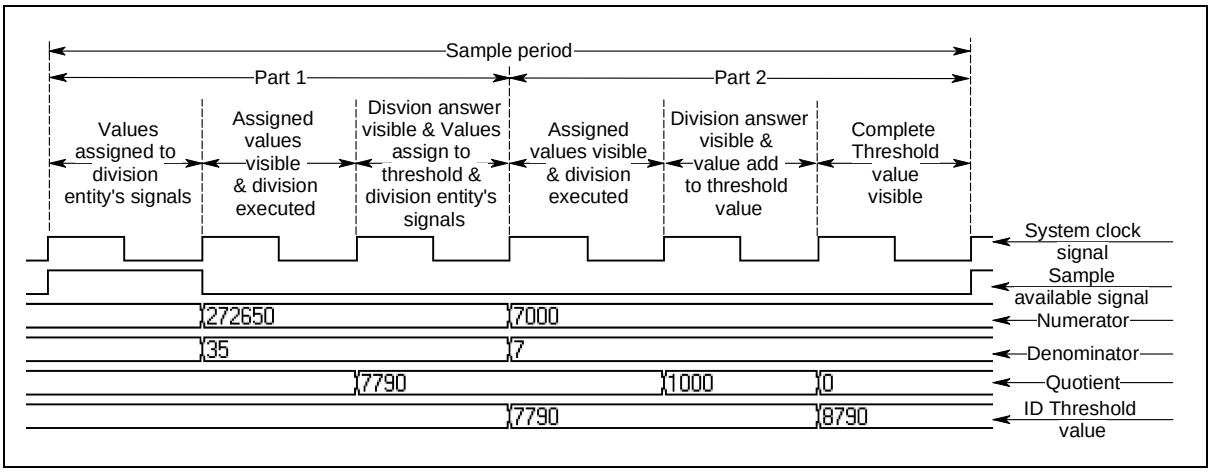


Figure 7.6 Division entities' signals

The equation for the ID Threshold requires two division operations and the second division is dependent on the answer of the first division. Here a trade-off between the usage of speed or logic gates had to be made. Instead of using two *division entities* for the two division operations, one operation was implemented. This resulted in the

yielding of an answer after only one clock cycle. By solving the problem in this manner a reduction in the amount of logical gates used was also achieved. Only six clock cycles are available between different samples, which means that speed is of the utmost importance. One also needs to take into account that whether speed is important or not, accuracy also needs to be taken into account. Figure 7.6 indicates the operation performed in each clock cycle

Algorithm III has also implemented a value known as a noise limit. This value is used to distinguish whether a peak value encountered in a quarter bit originated from a tag or common mode noise. The previously mentioned *division entity* is also used here for calculation purposes. This value is calculated by multiplying the peak value of the first synchronization bit by a factor of value 1.4 (see reference [16] for more detail). The fraction value representative of 1.4 is $\frac{7}{5}$.

The *ID detect sample start position* indicates to the ID detect function where the *sample count* should start. This defines the transition from the start bit to the first bit of the 64 bit ID.

All the data types in the *sync detect entity* are declared either as *variables* or *constants*. Like the previous *entity* this code resembles a sequential process and the assigned values must change immediately upon assignment thereof. The constant values are used to define the quarter parameters (see chapter 3 & 4). The only exception to this rule is the *signals* which interconnect the *process* and *division entities* (see section on basics of VHDL, for detail). Here, *signals* must be used, because as described earlier in this chapter, *variables* are confined to a *process.* The only “global variables” in an *entity’s architecture* are *signals*, *signals* on the other hand are confined to an *entity*.

7.3.4 ID detect entity

Figure 7.7 (a) illustrates the in and outputs of the *ID detect entity*. All the inputs of this *entity* were already discussed as part of the previously discussed *entities* in the afore-mentioned sub-sections. The *ID* output is a 64 bit value which, is a possible ID and must still be verified by the *CRC entity*.

The *Error in ID detect* is the second input to the *RFID detector process*, which resets the algorithm's *entities* when an error occurs in the detection process. The *Number of uncertain bits* output, provides the *CRC entity* with the number of undefined bits detected. As define by the designer (Defined in Algorithm II's design [16]) of the algorithm, the number of uncertain bits will be limited to three.

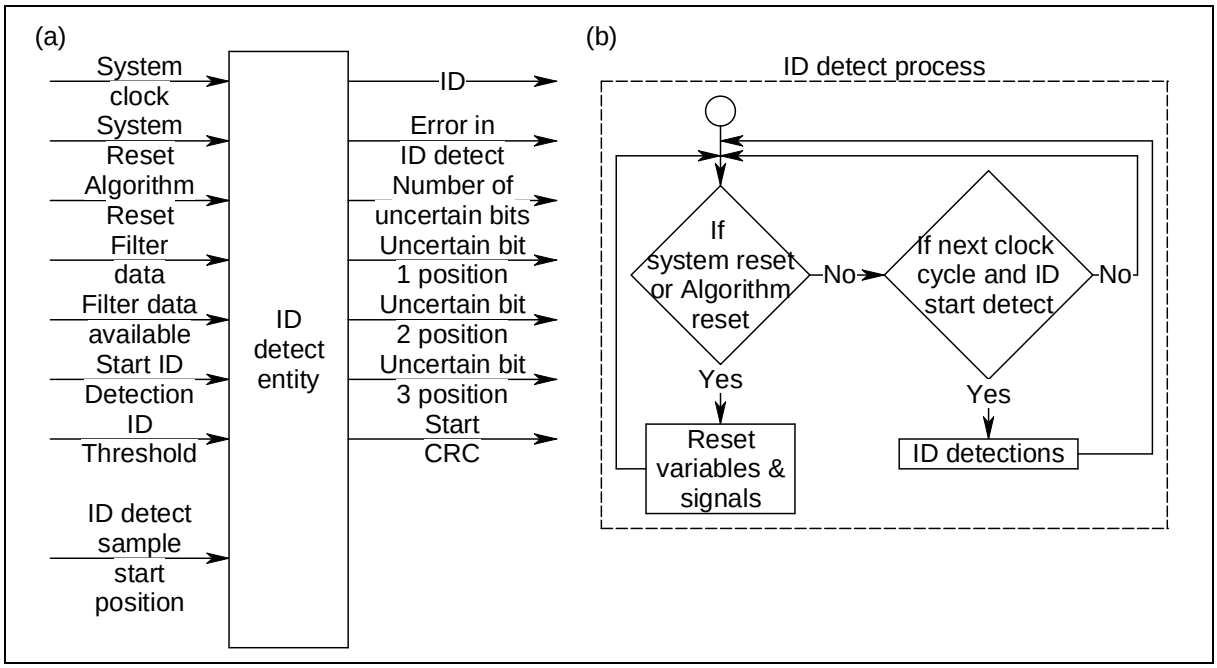


Figure 7.7 ID detect entity basic design

The next three outputs, *Uncertain bit 1, 2 and 3 position* indicates where the undefined bits are located in the yet to be verified 64 bit ID. The *Start CRC* output activates the *CRC entity* and verification of the detected ID's will be performed.

Just as in the *sync detect entity*, data types in the *ID detect entity* are declared as *variables* and *constants*. No internal *signals* are present in this *entity*. The reason for the latter is, one *process* and no *components* are present.

7.3.5 CRC entity

Figure 7.8 (a) illustrates the inputs and outputs of the *CRC entity*. All the inputs to the *entity* were already discussed in the previous sub-sections.

The *Valid ID* output is the validated ID, which would be used by the *pack entity* (See supporting *entities* for detail on *pack entity*).

The *Reset Algorithm* output is the third source for the *RFID detector process* which resets the algorithm when an error occurred during the algorithm performance. After the *CRC entity* latched the ID from the *ID detect entity*, the algorithm is reset. Thus whilst the CRC of the detected ID is being validated, the algorithm can start searching for the next ID.

The *Start ID packing* output indicates for the *pack entity*, that the 64 bit ID is valid and can be send to the *store entity* (See support *entities*, for *store entity's* details).

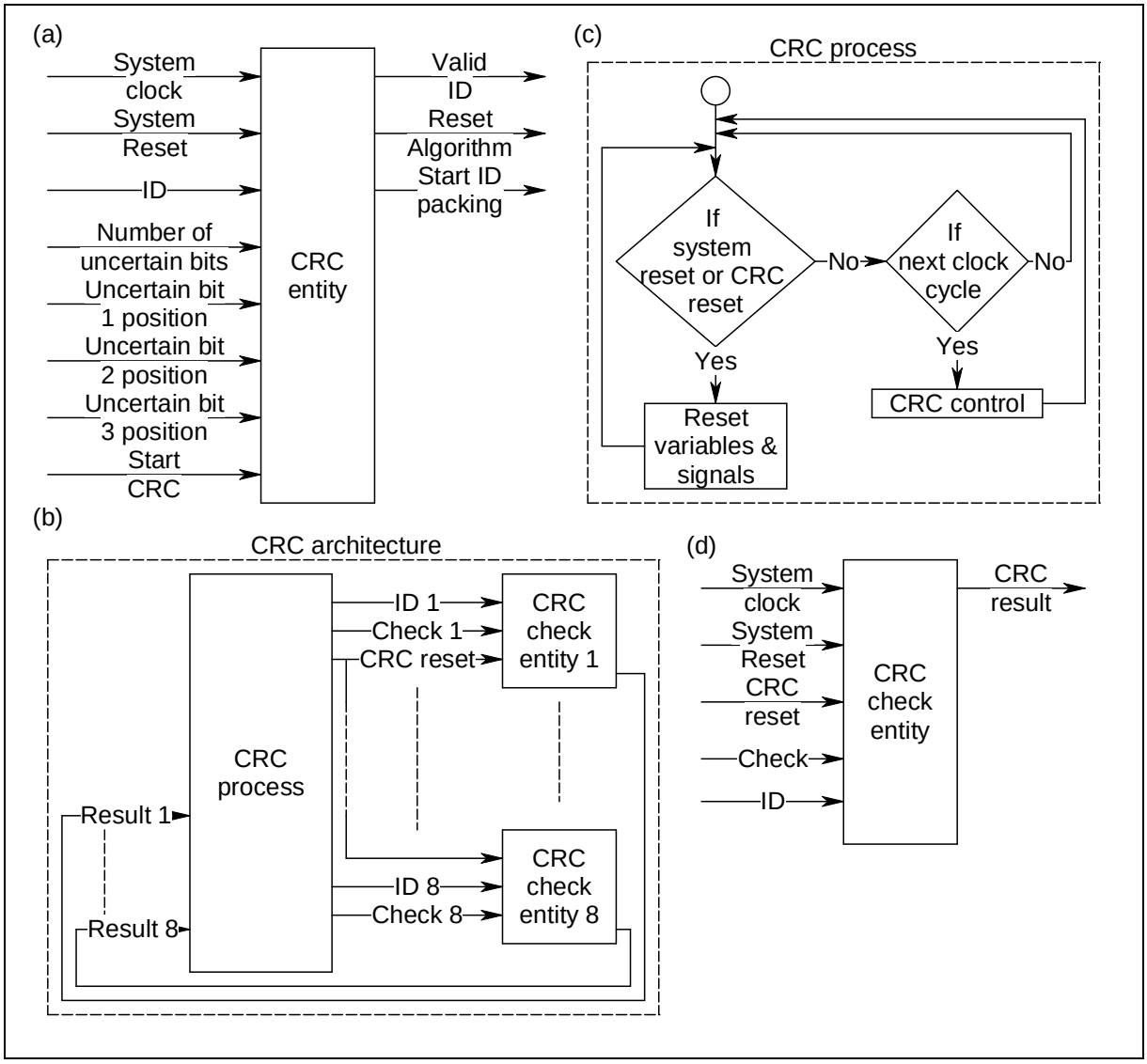


Figure 7.8 CRC entity basic design

The *CRC entity* is composed of a *control process* and eight *CRC check entities*. The *CRC entity* does not perform the CRC check in itself, but rather controls the process of checking the CRC (see figure 7.8 (c) and figure 7.9 (b)). The *CRC check entity* tests the CRC to see whether it is valid. There are eight *CRC check entities* to cover all the possible combination, for three uncertain bits.

Depending on the number of uncertain bits, a number of variations of the detected ID's CRCs should be tested. For example if there are three uncertain bits and because the value of a bit can be either a logic one or zero, eight different combinations must be tested.(see figure 7.8 (b)).

These *CRC check entities* provide an answer in one clock cycle (see figure 7.9 (a)). The *CRC control process* determines which of the IDs are valid and sets the *Start ID packing signal*. If more than one valid ID is present the CRC is reset, yielding the same result as an invalid CRC. In Appendix D, figure D.2 a *signal* diagram illustrates the complete CRC check sequence for a detected ID with three uncertain bits.

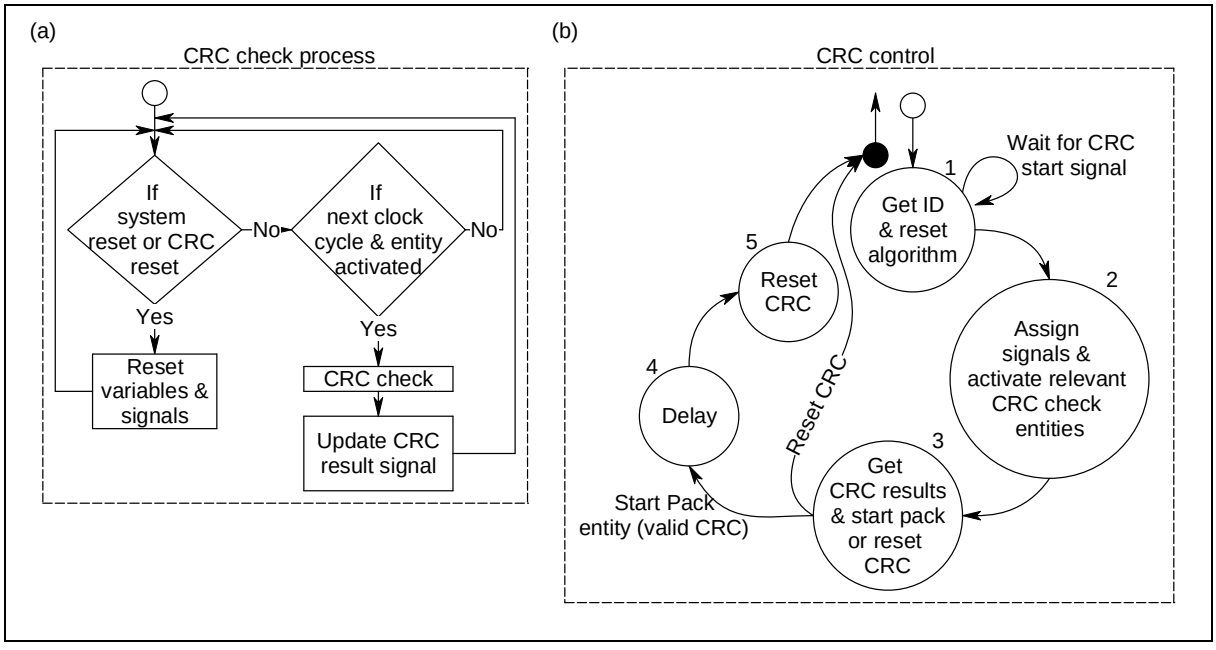


Figure 7.9 CRC check entity basic process design

The CRC entity's data types are almost all *signals*. The reason for this being, the values generated in the *CRC process* must also connect to the *CRC check entities*

and vice versa. There are a number of *variables* in the *CRC entity* which stores the uncertain bit information as these are lost once the algorithm is reset.

The *CRC check* function contains only *variables* in the one defined *process*. The reason for this is that assigned values must be available for use for the next statement.

7.4 Support entities

The support *entities* are not an integral part of the algorithm, rather as the group name implies, supports the algorithm. They function more independently than the algorithm's *entitie*s. Five support *entities* are present:

- *RFID detector entity:* This entity houses all other *entities*, both support and algorithm *entities*. Its function is to connect the *entities* to the external hardware (hardware outside the FPGA).
- *ADC entity:* Controls the external ADC.
- *Pack entity:* Divided the 64 – bit IDs into bytes.
- *Store (FIFO) entity:* Stores the bytes received from the *pack entity*.
- *RFID UART:* Transmit the bytes in the *store entity* to the RFID middleware.

7.4.1 RFID detector entity

Figure 7.10 (a) illustrates the in and the outputs of the *RFID detector entity*. This *entity* in essence describes the FPGA. All the other *entities* are *components* of this *entity* (see figure D.1).

The *External ADC DATA* is the 12 – bit data bus which, is connected to the external ADC. The bus is implemented both as an in and an output. It is used to setup the ADC and to retrieve the new sample values. The *External ADC DATA available* signal provides an indication to the *ADC entity* telling it when the ADC's new sample value is available.

The *ADC clock* is generated by the ADC entity and determines the sampling frequency (see next sub-section). The *ADC read* and *ADC write* signals are the control lines of the ADC entity to read and write to the ADC.

Figure 7.10 (b) describes the only *process* contained in the *RFID detector entity*. The function of this *process* is to generate the internal algorithm reset *signal*. This *signal* is generated when an error *signal* is received from one of the previously mentioned three possible error (error in, *sync detect* or *ID detect* or *CRC entity*) sources. The *process* is housed inside this *entity* because it is the only *entity* which has access to all three error signals. (See previous discussions) Thus the *entity* enables a centralized function for the generation of the algorithm reset *signal*.

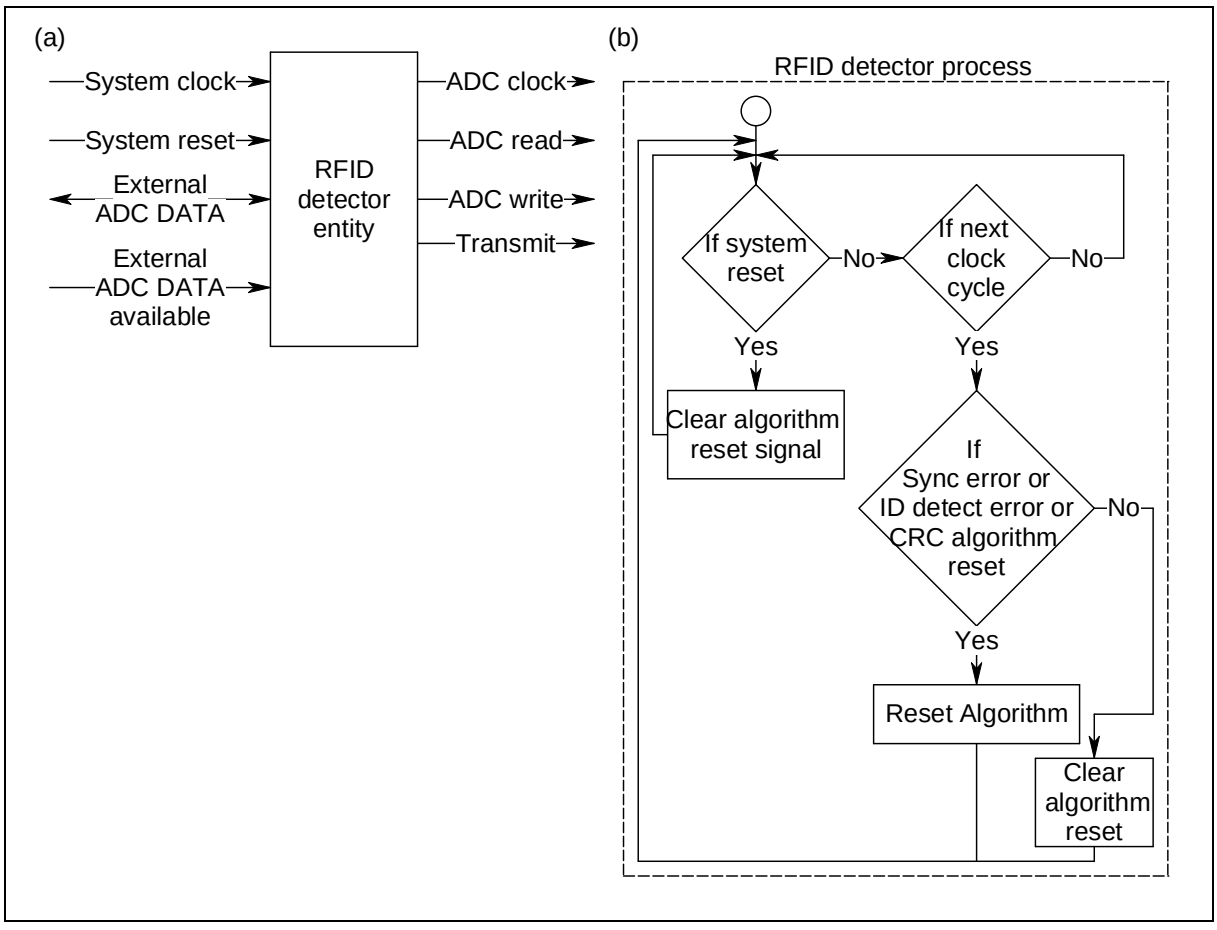


Figure 7.10 RFID detector entity basic design

7.4.2 ADC entity

Figure 7.11 (a) illustrates the in and outputs of the *ADC entity*. All the input signals were already discussed in the previous sub-sections. The sampling clock *signal* is assigned to the *ADC clock* signal and is use by the external ADC.

The *ADC entity's architecture* contains two *processes*. One is used to generate the sampling clock, *clock process,* and the other, *control process,* to control the external ADC. Figure 7.11 (c) is representative of the basic design of the *clock process* and (d) the *control process's* design.

The *control process* controls the setup of the external ADC after a *system reset*, read new sample values from the external ADC and starts the clock process. The *clock process* is started by the *ADC control process* via the *start signal*, see figure 7.11 (b). The sampling clock is a 3.072 MHz (see reference [11],[16] and Appendix G) square wave with a 50 % duty cycle.

The *control process* has three states namely *ADC reset*, *ADC setup* and *ADC sample*. The *ADC read* and *ADC write signals* are generated in the *ADC control process*. The *ADC read* signal is used to read a new sample from the ADC unit, when the *ADC entity* is in the ADC sample state. The *ADC write* is used by the *ADC reset* and *ADC setup* states to reset and setup the ADC unit respectively.

The setup of the ADC, to which the previous paragraphs refer to, are the two control register values of the ADC unit which must be set. They determine the behaviour of the ADC unit (see CD enclosed for the TH1206's datasheet, for more technical detail about the control registry).

The *ADC data* and *ADC data available* signals are generated in the *ADC sample* state and is the internal equivalent of the external ADC unit's signal. After the *ADC entity* obtained a new sample value from the ADC unit the *ADC data* and *ADC data available signal* values are set. These *signals* are use by the *filter entity* to indicate when a new sample value is available.

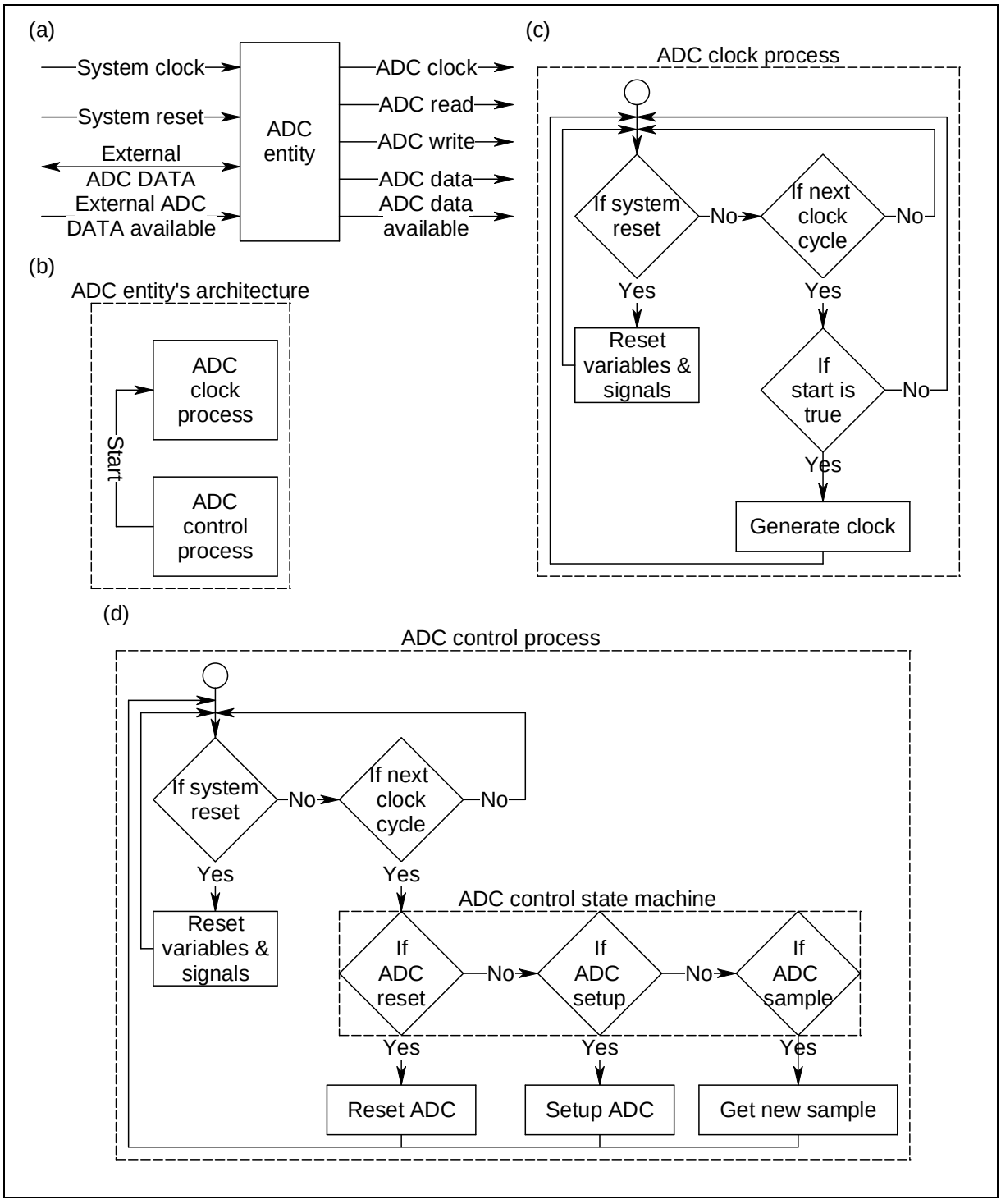


Figure 7.11 ADC entity basic design

Each of the *ADC control process* states has four sub states. The different sequences of the *ADC reset* and *ADC setup* states are:

1. Assign values to ADC external data, for first register write.
2. Write values to register.
3. Assign values to ADC external data, for second register write.
4. Write values to register.

The different sequences of the *ADC sample states* are:

1. Start ADC clock process.
2. Wait for new sample and send read signal to external ADC unit.
3. Get new sample and assigned value to internal ADC data.
4. Clear external ADC data port and return to state 2.

7.4.3 Pack entity

Figure 7.12 (a) depicts the in and outputs of the *pack entity*. The first four inputs were already discussed in the previous sub-sections. The *FIFO full* signal is generated in the *store entity* (refer to section 7.4.4). This *signal* indicates to the *pack entity* whether the FIFO is full or not. This *signal* along with *Number of bytes in FIFO* is used by the *pack process* to determine whether it will store the valid ID or discard it.

If the *pack process* is not able to store a complete ID in the FIFO it will be discarded. This prevents the possibility of a valid ID currently in the FIFO being overwritten. In such a case the questionable ID is rather discarded and the valid ID forwarded than forwarding a possible ghost or invalid ID. The next sub-section will handle the number of IDs that can be stored and the reason why they are stored.

The *FIFO write* along with the *FIFO data signals* enable the *pack entity* to write data to the *store entity*. The 64 – bit ID is divided into 8 bytes and sent to the *store entity*. The ID is divided into bytes to make transmission easier and to centralize functionality. Thus the *RFID UART entity* (refer section 7.4.5) only transmits an ID.

Figure 7.12 (b) illustrates the basic flow of the *pack process*. The *packing process* is activated by the *start signal* obtained from the *CRC entity*.

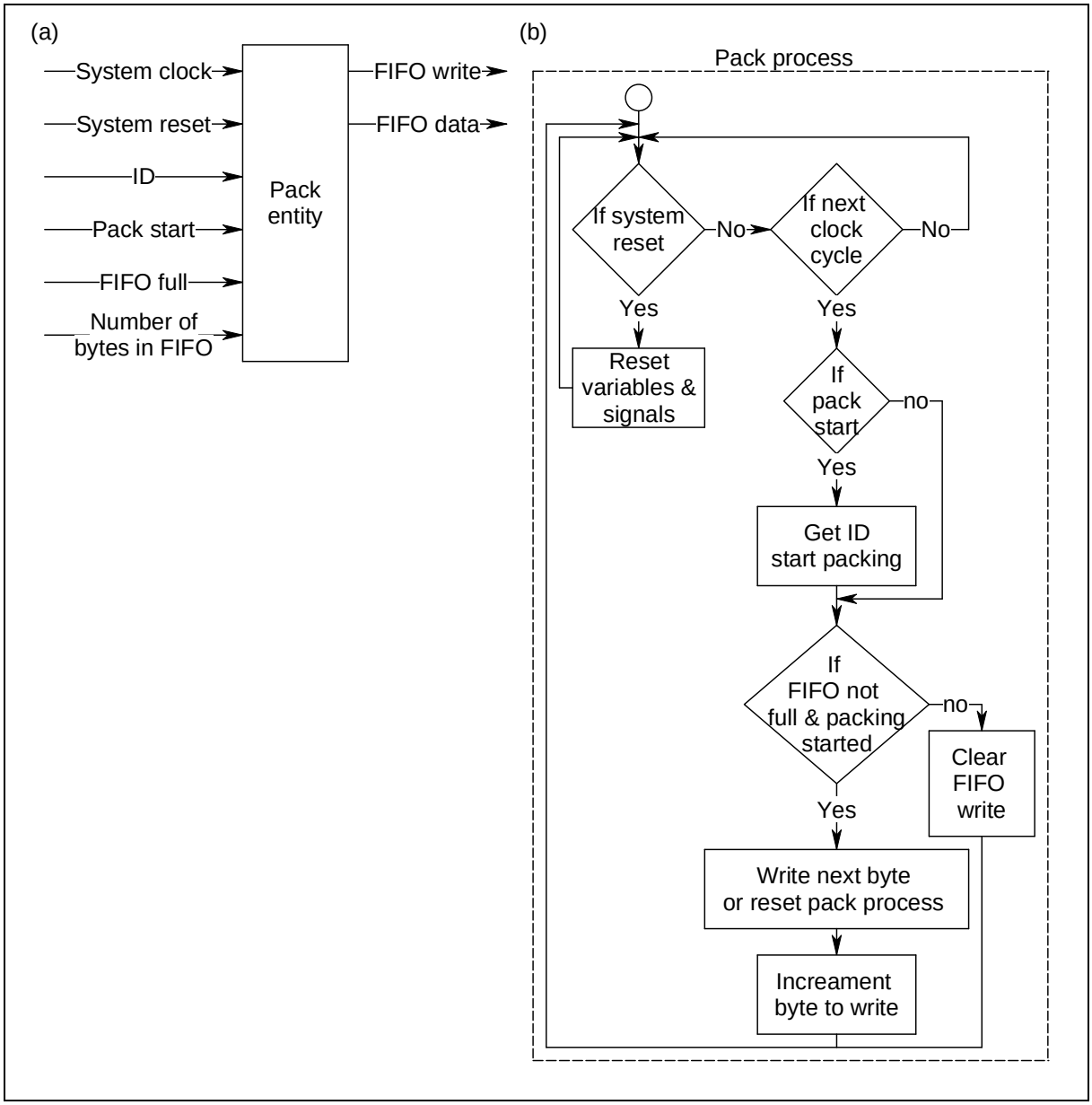


Figure 7.12 Pack entity basic design

7.4.4 Store (FIFO) entity

Figure 7.13 (a) illustrates the basic design of the *store entity*. The *store entity* is a FIFO stack and is used to store valid IDs while the *RFID UART* is busy. The reason why the IDs are stored is because the transmit rate is to slow to transmit an ID packet before the arrival of the next ID, this constraint is imposed by the hardware. The *store entity* will be treated as a black box, because it is one of Altera's plug–in *entities*.

The FIFO parameters can be changed. The width of the FIFO is set to 8 bits. This is due to the fact that the *RFID UART* transmits 8 bits at a time. The number of bytes in the FIFO stack is 8192. Memory available on the FPGA used in this project constrains the latter setting.

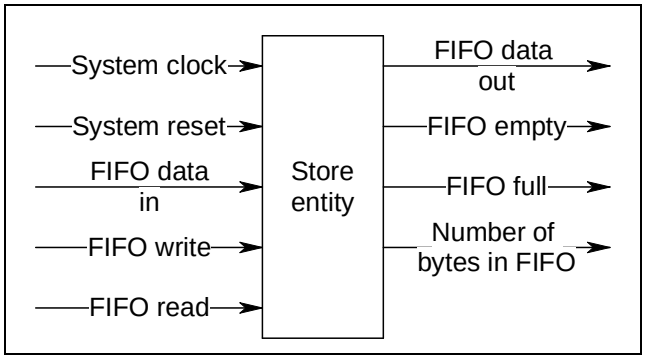


Figure 7.13 Store entity basic design

Thus, for a 64–bit ID, consumes 8 bytes of the FIFO. A simple mathematical calculation shows that a total of 1024 IDs can be stored. The analogue RFID reader system is compared to the digital RFID reader system in time frames of 30 s (refer to Appendix G). In order to determine whether the storage capacity and transmission rate of the digital reader is sufficient for the 30 s time frame, without losing any detected IDs, the following study was conducted:

The study conducted to determine the capacity (ID throughput) the combination of the FIFO and UART can accommodate consist of two scenarios. The first scenario was completely time based calculations. The following assumptions were made for the first scenario:

- A valid ID is detected in the time frame allowed for a complete ID, back to back.
- An ID packet is transmitted in the time frame it takes to transmit the bytes with a baud rate of 115200.

A M file (included on the CD)was used to implement the above mentions criteria and figure 7.14 contains the results obtained.

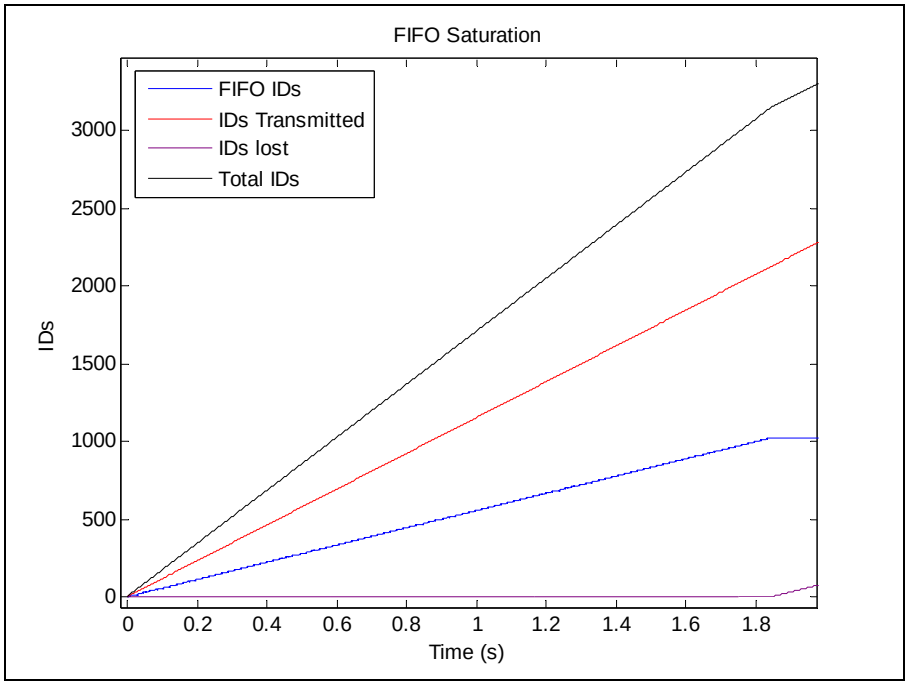


Figure 7.14 FIFO saturation

The assumptions made in the generation of the graph thus provide a theoretical maximum time during which the digital reader's FIFO and UART would be able to keep up with the detection of IDs without saturating the FIFO and as a result discarding detected IDs. From figure 7.14 this maximum is about 1.8 s. If the time frame is extended to 30 s a total of about 35000 IDs are detected which excludes the number of IDs lost.

The large amount of IDs detectable in the first scenario is not representative of a true system. The second scenario made use of statistical principles. In this scenario the IDs were not assumed to be back to back, but rather probability to be present in a given time frame.

If the statistical approached is taken the total number of possible IDs in a 30 s time frame with the analogue system as reference, a total of almost 7000 IDs can be read. This maximum number of IDs is obtained from the number of IDs the analogue system can read in 30s, with 18 tags 20 cm from the reader. This is a more representative scenario of the capacity the FIFO and UART would encounter.

In order to simulate this, a random generator's output was used to determine the probability of an ID existence in the current time frame of a complete ID. The total number of IDs in the 30s time frame is reduced to about 25000 IDs. This is still more than the total number of IDs the analogue system can detect in the same 30 s time frame. With the statistical approach no IDs were lost and the system could handle the number of IDs.

A more thorough study must be conducted to determine whether the hardware is sufficient for the digital reader, especially if the system would be commercialized, but this falls outside the scope of this thesis. The system is currently mostly constrained by the hardware. In order for this problem to be resolved a hardware revision must be done or a more elegant transfer protocol must be designed for the system to meet specified requirements.

7.4.5 RFID UART entity

Figure 7.15 (a) illustrates the in and outputs of the *RFID UART entity*. The *FIFO empty* and *Number of bytes in FIFO* is used to determine whether an ID is in the FIFO. The *FIFO data* input is used to retrieve the ID in bytes, to transmit it.

The *FIFO read* output is used to retrieve the data from the FIFO, by starting a FIFO read sequence. The *Transmit* output is the serial data out string of the UART.

The latter *entity* is composed of two *processes*. The *UART baud rate generator process,* which generates the serial data out clock and the *UART transmit process*, which handles the retrieval and transmission of the ID packet. (See figure 7.15 (b))

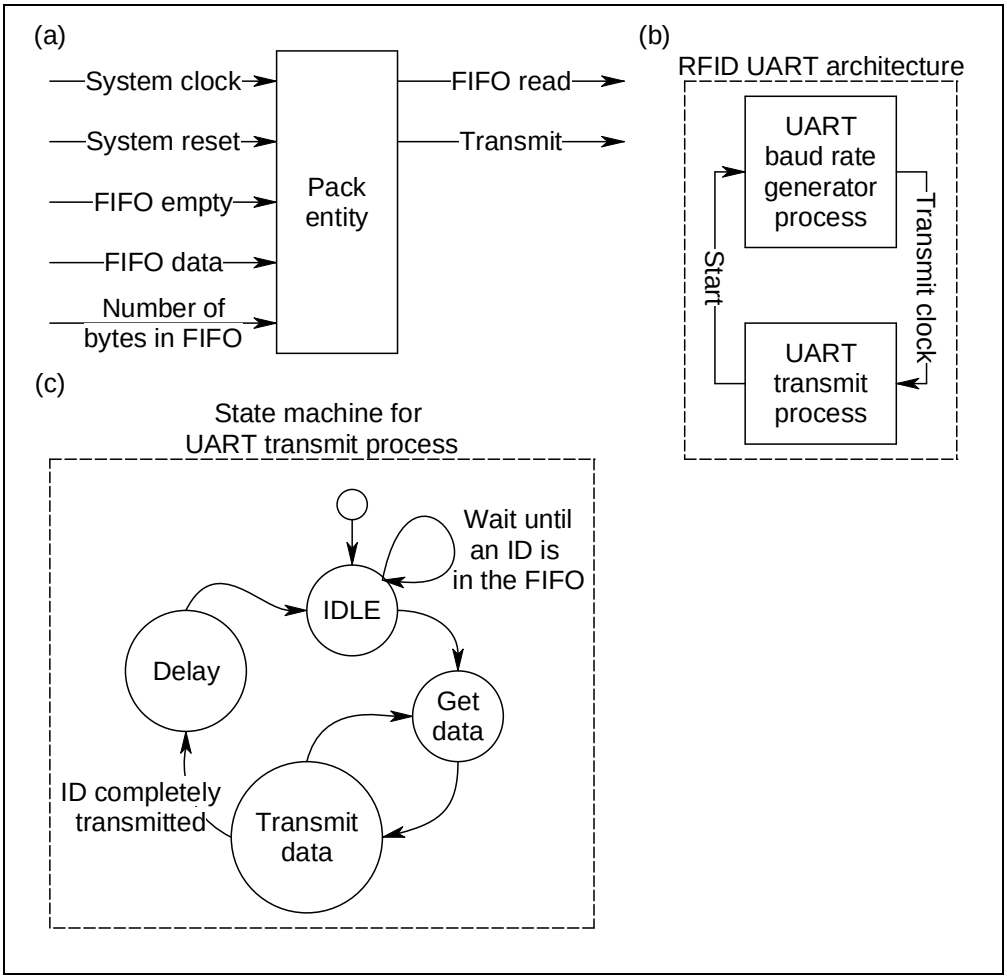


Figure 7.15 RFID UART entity basic design

Figure 7.15 (c) illustrates the state machine for the *UART transmit process*. The *IDLE* state waits until a complete ID is present in the FIFO and only then starts the sequence of transmitting an ID packet. The *Get data* state has 10 states namely:

- State 0: send header byte of ID packet to *Transmit data.*
- State 1 to 8: get data byte(s) from FIFO and send data byte to *Transmit data.*
- State 9: send the *end byte* of ID packet to *Transmit data.*

The *Transmit data* state uses the clock generated in the *UART baud rate generator process* to send the 10 bits over the serial port. The 10 bits is composed of the start bit, 8 data bits and a stop bit.

The *Delay* state is used after a complete ID was transmitted. This state delays the transmission of IDs for two *system clock cycles* in order to allow the RFID middleware time to process the ID send.

7.5 Conclusion

At this stage the algorithm has been implemented on the FPGA.

The protocol implemented for handling the IDs after detection and verification must be revised. The brief study described earlier in this chapter highlighted only some shortfalls in the current protocol and hardware. As this is not the core of the study and there is hardware issues coupled to the implementation, it should be researched in another study.

The accuracy of the implementation as well as the effect the improvements have made on the algorithm's detection rate must be tested. The next chapter will deal with these issues and answer further questions arising from the implementation.

8 • Chapter

8 Test

8.1 Introduction

The accuracy and efficiency of the VHDL implementation of Algorithm III will be examined during the course of this chapter. Furthermore the improvements made to the algorithm will be examined, to test whether the detection capabilities of the algorithm have truly improved or not.

8.2 C++ vs VHDL

As stated previously, the core objective of this project is to implement an enhanced RFID detection algorithm in VHDL of which the performance equals its C++ counterpart. The only way to accurately compare the two implementations is to use the same data as input and compare the detection capabilities of the two different implementations.

The evaluation was conducted in two phases. The first phase tested the VHDL implementation's capability to handle frequency deviations in tag data against the C++ implementations. This phase was conducted with the aid of simulation software.

The second phase tested the VHDL detection capabilities against the C++ detection capabilities. In order for this to be possible the sample data used by the VHDL must be captured, but the FPGA currently has insufficient memory for this task. An add on board with SRAM memory was designed and used to expand the FPGA's memory. (Consult Appendix E for detail design of this solution).

In both phases the tests were repeated four times and the averages used to generate the resulting graphs. The tests were repeated to obtain a statistical average.

8.2.1 Phase one - Method

The theoretical data used in chapter 6 to test the enhanced algorithm's (Algorithm III) ability to handle data frequency deviations was used to test the VHDL's detection ability. The test was conducted in ModelSim – Altera 6.1g, which is a VHDL simulation tool. A testbench entity (A self - contained entity, no inputs or outputs [6]. This type of entity is only used for simulation purposes.) was used to simulate the external input signals of the FPGA.

The C++ implementation of Algorithm III, was tested in the new simulation software (Simulator, refer to chapter 4). And the results compared to the VHDL's results from ModelSim.

8.2.2 Phase one - Results

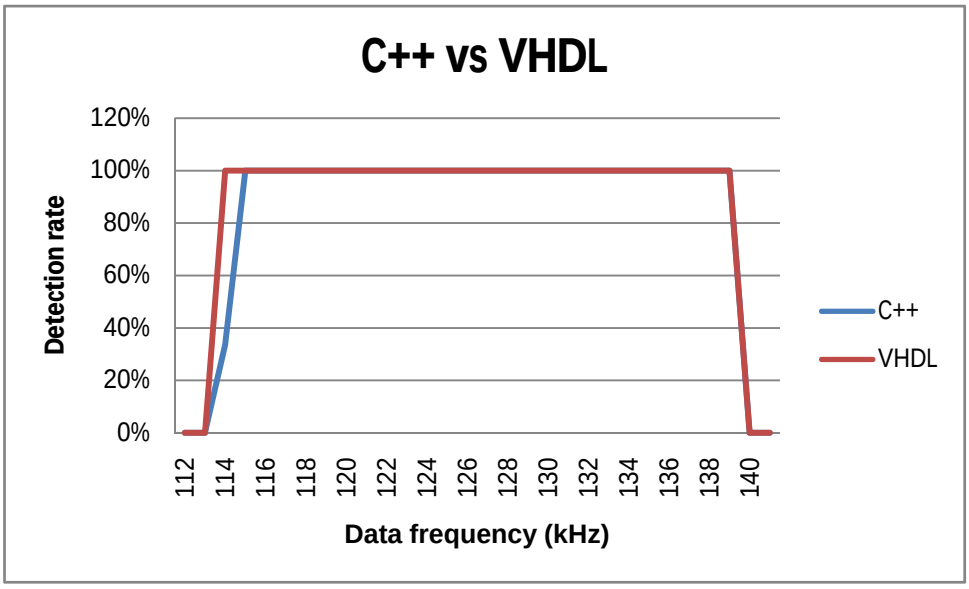


Figure 8.1 The frequency abilities of the different implementation of the algorithm

8.2.3 Phase one - Conclusion

Figure 8.1 illustrates that the VHDL implementation detection range is the same as its C++ counterpart. There is only one difference in the detection capability and it occurs at 114 kHz. The VHDL has a 100 % detection rate, whiles the C++ has a 34 % detection rate at the mentioned frequency.

8.2.4 Phase two - Method

The second phase consisted of two tests. In the first test no noise, as defined by iPico, was added to the system, but do not account for noise in the environment. In the second test noise was added. The noise was specified by iPico to simulate a switch mode power supply. The noise is a 2.2 kHz square wave with a 50 % duty cycle.

The noise was generated by a signal generator and transmitted via antenna, to interfere with the tag's signal, received by the analogue reader. Figure 8.2 illustrates the noise signal after it was sampled by the ADC. Figure C.2 (b) illustrates a random tag's synchronization bits. Notice that figures 8.2 and C.2 (b) have a resemblance. This resemblance indicates that it is possible for a noise signal to be mistaken for a tag bit.

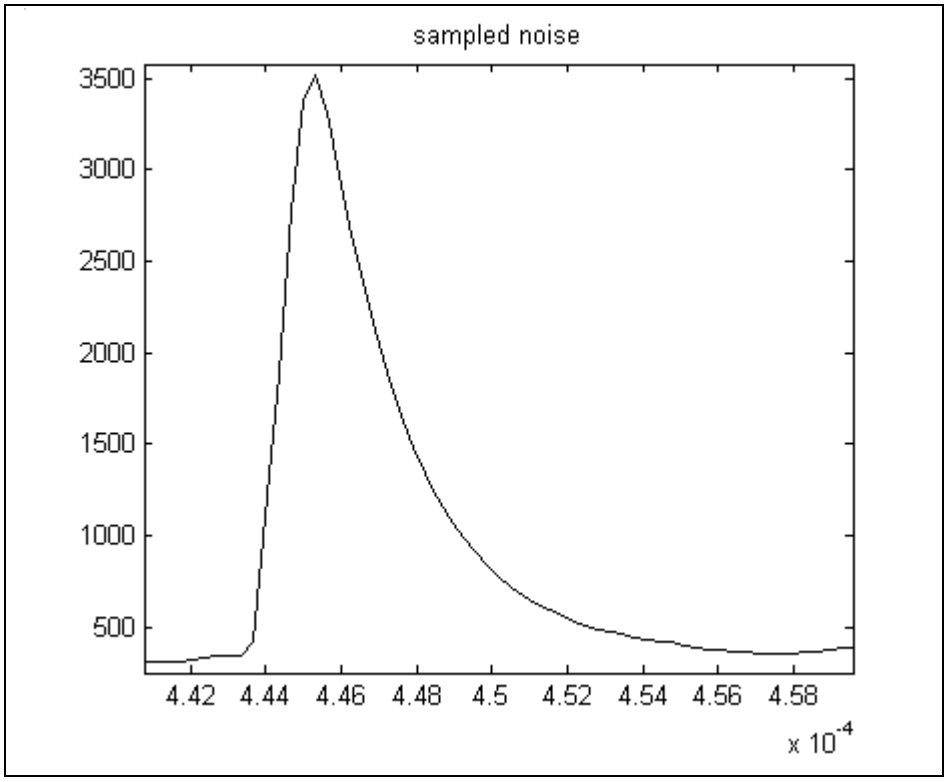


Figure 8.2 Sampled noise signal

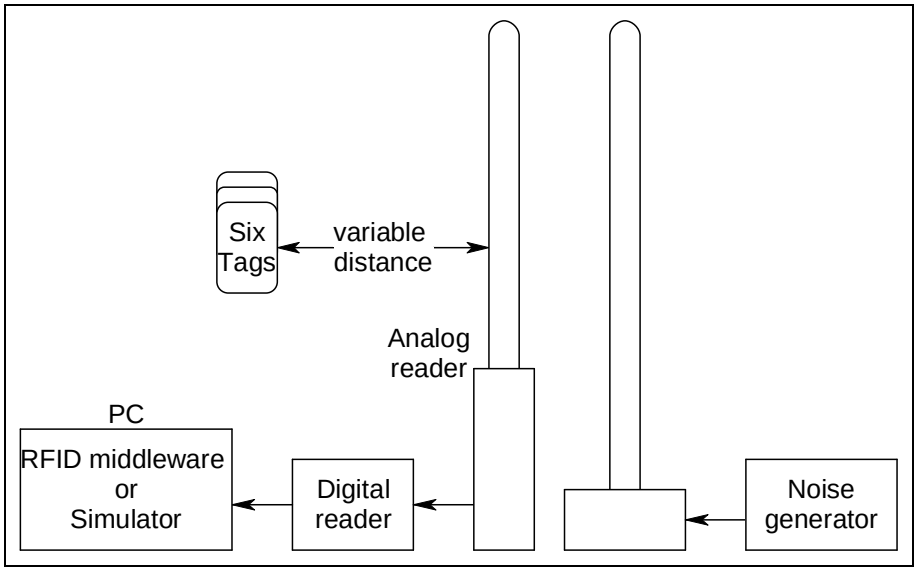


Figure 8.3 FPGA setup for testing

Figure 8.3 illustrates the test setup. The different components are:

- The analogue reader, used to wake-up the tags and read the tag's response.
- The passive RFID tags.
- The digital reader, with Algorithm III implemented on the FPGA.
- A noise generator.
- RFID middleware to obtain the detection capabilities of the digital algorithm implemented in VHDL.
- The simulator to evaluate the C++ implementation of the algorithm, on the same data as the VHDL implementation.
- The software which reads the SRAM, to the PC and stores this data in a .cvs file format.

For the no-noise setup the noise generator and its antenna are removed. Six tags are place in the analogue reader's range at different distances, starting at 10 cm with 10 cm increments respectively. For consistence, six tags were used, the same as in phase V (refer to appendix G). The digital reader receives, the tags' response, from the analogue reader after the carrier wave is suppressed and is an analogue signal. This signal is signal (2) in figure 3.6.

Testing was conducted in two stages, for this phase. In the first stage the digital reader, SRAM expansion board, analogue reader, tags and the RFID middleware were used. The purpose of the first stage is to obtain sample data for the second stage and test the digital reader's detection capabilities.

In the second stage the digital and analogue reader are inhibited. The stored data on the SRAM is retrieved and stored in a .csv file format. Thereafter the .csv file is used in the simulation software to test the C++ implementation's detection capability. The length of the data stored is roughly 0.7 s (see Appendix E for details).

8.2.5 Phase two - Results

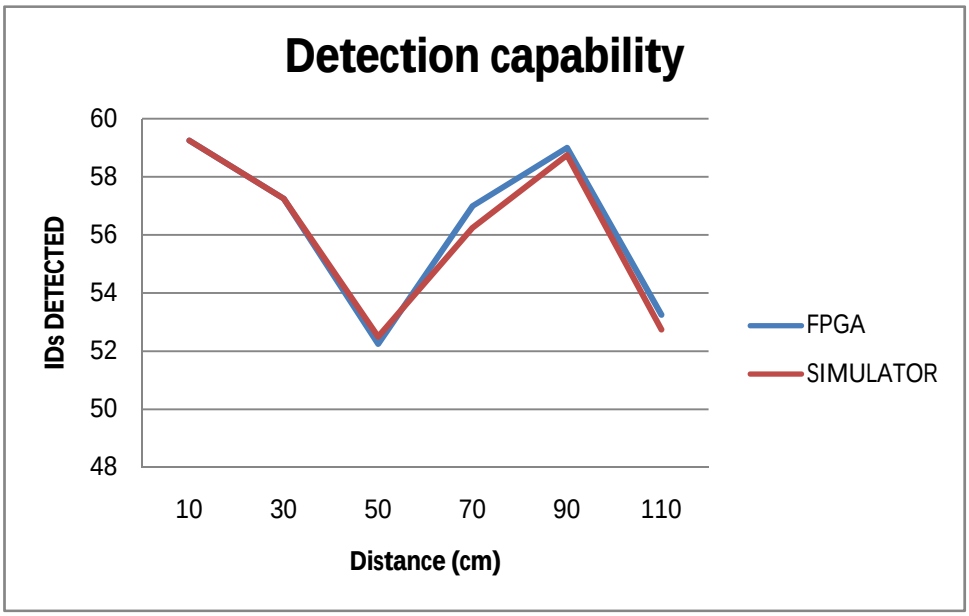


Figure 8.4 Detection capabilities of C++ vs VHDL, noise excluded

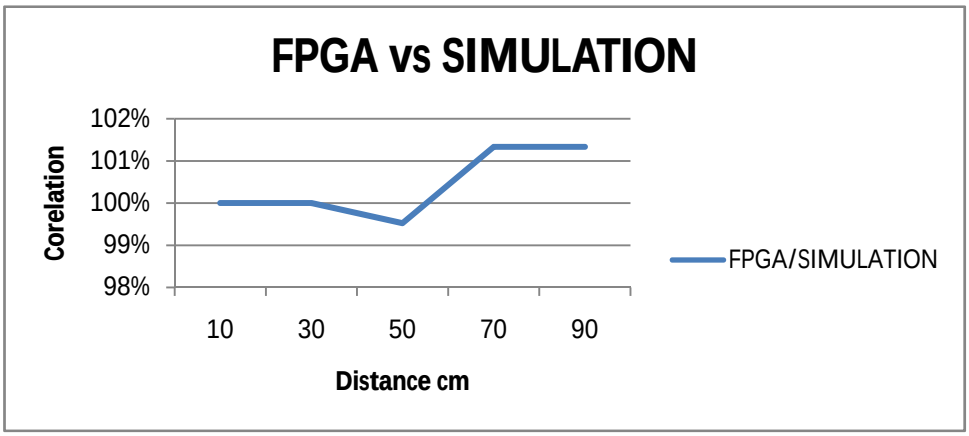


Figure 8.5 VHDL correlation to C++ detection capability, noise excluded

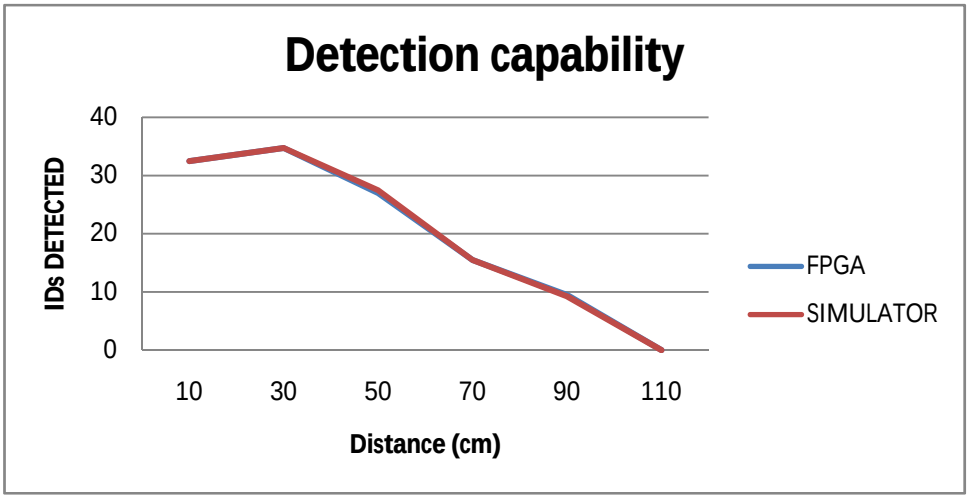


Figure 8.6 Detection capabilities of C++ vs VHDL, with noise

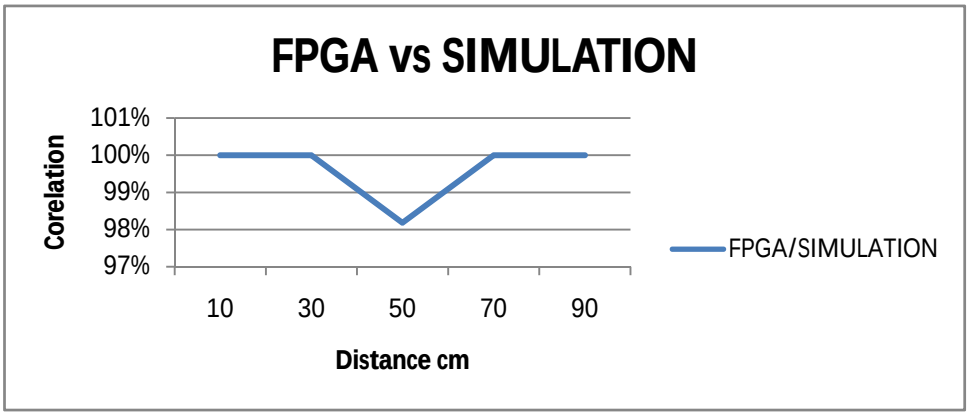


Figure 8.7 VHDL correlation to C++ detection capability, with noise

8.2.6 Phase two - Conclusion

The results indicate that the two implementations correlated relatively accurately. This correlation is independent of whether noise was added to the system or not. The data is available on the CD included in the back cover of this dissertation.

8.3 C++ detection capability

The detection capability of the C++ implementation of Algorithm II was compared to that of Algorithm III. This test aims to illustrate two effects,

- The performance of the algorithm after the code was cleaned up.
- The effect of the improvements made.

8.3.1 Method

The data used in the previous test was used as input for the simulation software. The C++ version of Algorithm II from the previous simulation software (GrafiekE) was used after it was converted to a DLL for the new simulation software (Simulator).

The test results for algorithm II were generated in the first version of the new simulation tool. The only difference between the final version of the new simulation software, except for the improved C++ algorithm (Algorithm III), and the first was a few new functions. Also the data is now only reference in the DLL. This will in no way hamper the performance of the algorithm.

8.3.2 Results

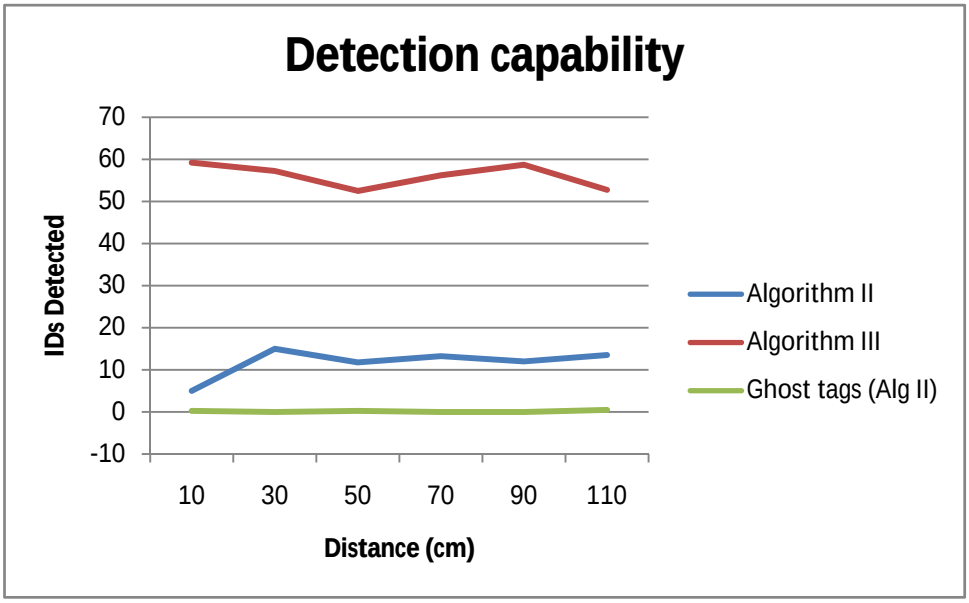


Figure 8.8 C++ algorithms' comparison, noise excluded

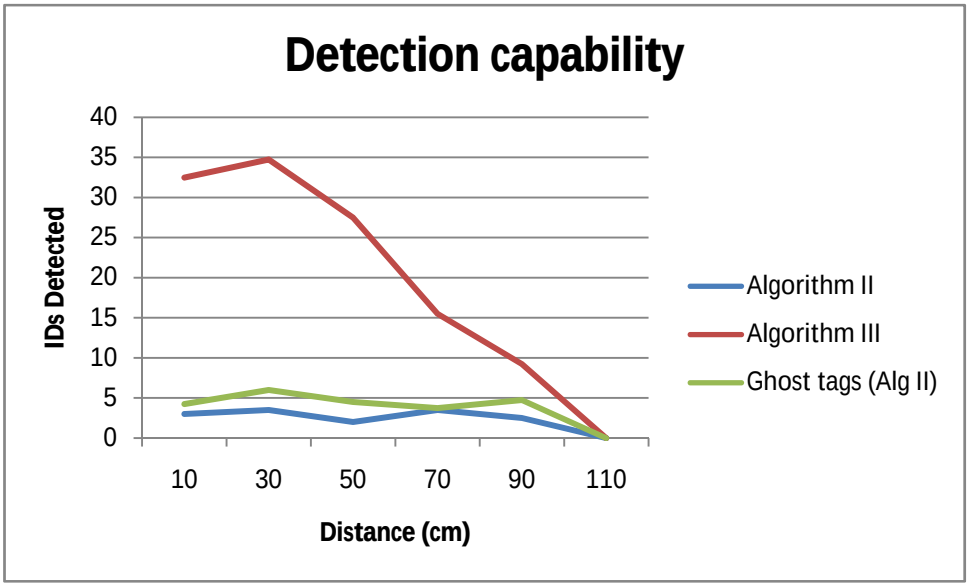


Figure 8.9 C++ algorithms' comparison, with noise

8.3.3 Conclusion

First of all the ghost tags in figures 8.8 and 8.9 do not have the same structure as that of the ghost tags detected in the RFID middleware described in chapter 5. These ghost tags are due to the incorrect implementation of the CRC function, which was corrected during the code clean up. The “Algorithm II” line illustrates the valid number of IDs detected and excluded the ghost tags detected. The figures lack the ghost tags detected in the new C++ algorithm, because none was detected.

Figures 8.8 and 8.9 illustrate the improvement in performance of the C++ implementation of the algorithm. This is due to the code clean up and the improvements made to the algorithm.

8.4 Trio test

The trio test compares the detection capabilities of three RFID readers, namely

- The analogue RFID reader from iPico
- The digital RFID reader used in phase V (the compiled version on the CD was used[11])
- The digital RFID reader of this project.

8.4.1 Method

Figure 8.10 illustrates the test setup. Just as in the previous section the tests were conducted with and without noise (noise generator and antenna removed). The distances between the tags and reader were also varied. All three readers ran concurrently.

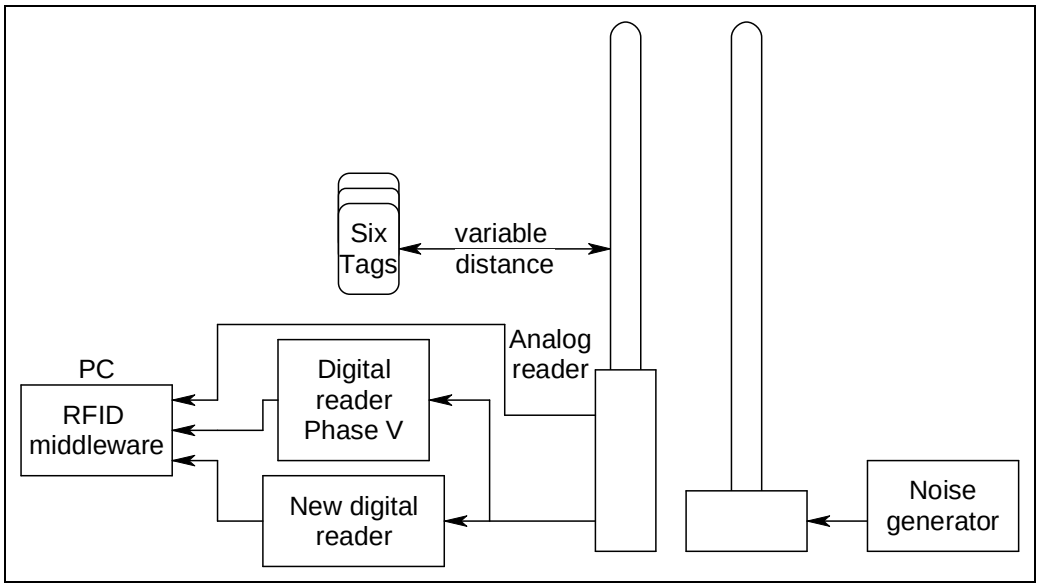


Figure 8.10 Trio test setup

8.4.2 Results

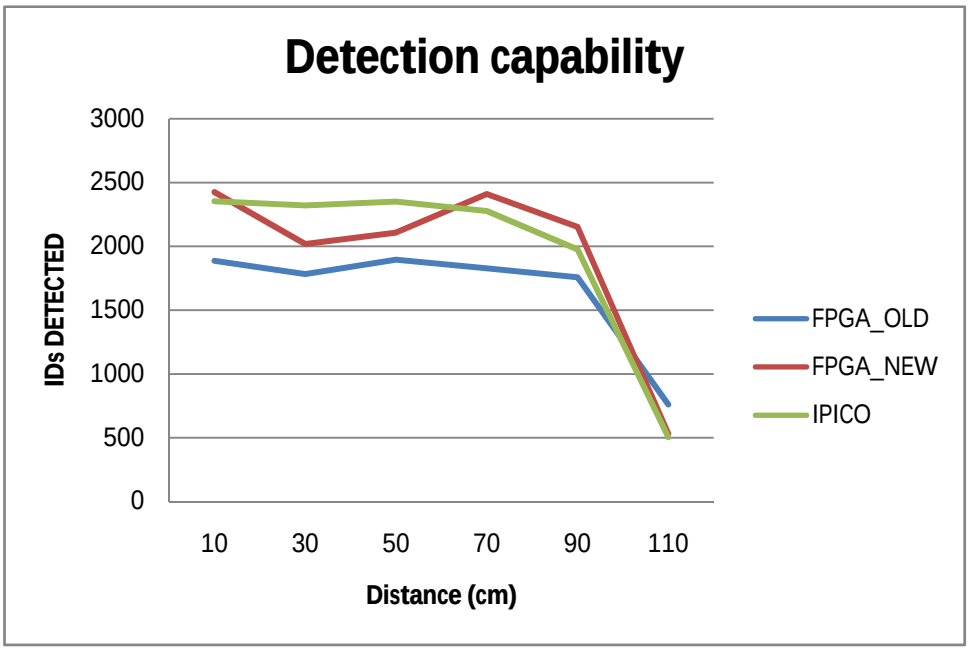


Figure 8.11 Detection capabilities of trio test, noise excluded

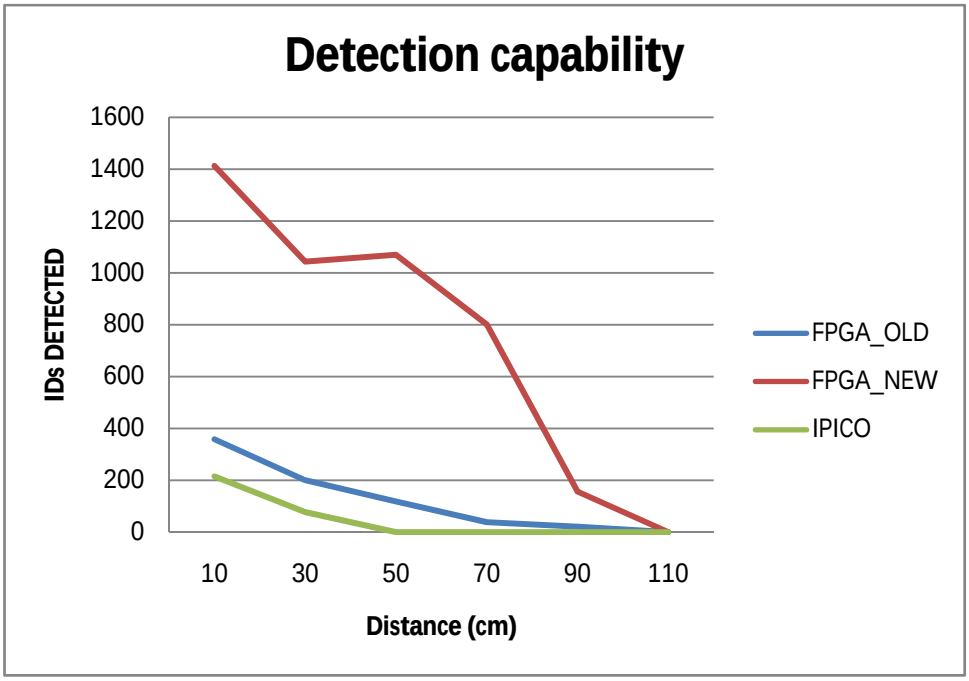


Figure 8.12 Detection capabilities of trio test, with noise

8.4.3 Conclusion

From figure 8.11 it can be seen that Algorithm III has a better detection capability than phase V's algorithm (Algorithm II's predecessor). Algorithm III and iPico's reader detection capabilities cross each other over the detection range, no one is can be classified as better than the other.

On average over the detection range with the analogue reader as base line the phase V algorithm has 84% and Algorithm III 91 % detection capability correlation respectively. Thus an improvement in the detection performance of the digital algorithm can be seen in a noiseless system.

From figure 8.12 it is clear that Algorithm III out performance both the analogue reader and phase V's reader. On average over the detection range with the analogue reader as base line the phase V algorithm has 251 % and Algorithm III 1533 % detection capability correlation respectively. The analogue reader cannot compete with the digital readers in the noisy environment. Furthermore the digital detection capabilities of Algorithm III in a noisy environment are superior to Algorithm II's detection capabilities.

9 • Chapter

9 Final words

9.1 Conclusions

It is difficult to relate the origin of the improvements in the detection capabilities of the digital algorithm to any one specific factor. This is due to the fact that the algorithm implemented on the FPGA in phase V is not the same as the algorithm used in the previous simulation tool. However, improvements to the noise detection functionalities were made to the phase V algorithm. The algorithm in the previous simulation tool (GrafiekE) forms the basis of this project and improvements were made to that.

The frequency and noise detection improvements possibly are the two most important contributing factors to the enhanced detection capabilities of the new algorithm. The elimination of the ghost tags from the system is not seen as an important contributing factor. This is due to the fact that the same RFID middleware is used to display the detected IDs for the digital readers. Besides, referring back to chapter 5 tests confirmed that the ghost tags did not originate in the algorithm, rather it was an RFID middleware originated problem.

9.2 Future recommendations

Although the detection capabilities of the digital RFID detection algorithm, especially in a noisy environment, outperform the analogue reader, it can only be seen as relative. This performance is only for a specific noise “type”. That being said, there are a number of aspects of the current digital RFID reader which needs attention, even for this one specific noise.

9.2.1 Analogue signal

This project is still in its design phase so it is understandable that the digital reader does not have its own enveloping circuit. This in itself also poses a problem, because it is not known how iPico's enveloping circuit works and the digital reader's performance can be hampered by that. The output of this circuit, an analogue signal is the input to the digital system (designed and tested during the course of this study) and the quality of this signal is dependent on the analogue reader's circuitry.

Furthermore, from basic electronic knowledge, the circuit preceding the ADC is critical to the performance of the algorithm in itself, because any changes made to the signal, directly affects the algorithm. The preceding circuit currently has an RC high pass filter, followed by a buffer and an amplification stage. The amplification stage has to be set up manually.

A more thorough investigation needs to be conducted to design an analogue circuit to precede the ADC. This circuit has three aspects which need to be addressed.

1. Is the analogue reader's enveloping circuit adequate for the digital reader's needs? And does this fit the environment where the system will be used?
2. What effect does the current filter have on the analogue signal and what is its effect on the digital reader? Furthermore is the current filter adequate for the specific application?
3. The manual amplification stage needs to be replaced by an automatic gain control stage and the output must comply with the signal level limits of the ADC.

Aspect three is critical since if the signal level is out of the range of the ADC, invalid data is send to the algorithm. This data might be chopped, clipped or simply muted. Aspects two and three hold in themselves a great effect on the performance of the digital detection algorithm. The basic determination the algorithm uses to know where it currently is within in a bit is based on a theoretical number of samples. Thus if the analogue signal is disfigured by the filter and amplification stages these sample numbers will inherently vary. An underlying floating ground is currently detected at the output of the preceding stages to the ADC.

9.2.2 Digital filter

One of the effects the digital filter has on the sampled data was briefly mentioned in chapter 6 and illustrated in figure 6.1. The digital filter effects the number of samples per bit. Because it is just an averaging filter, of which the function is to filter out white noise (common mode), the following need to be examined.

- Is there a better filter for this task?
- Can the analogue filter not be used to limit the noise before it is sampled, by using a band pass filter for example?

9.2.3 Threshold

The threshold is used to determine the activation of the detection algorithm. The threshold is currently a constant value larger than the average value of the peaks of the previous one hundred samples. Because an RF signal decreases exponentially, the signals received from the tags are very small if the distance between the tag and reader grows large. This corresponds to less sample values above the threshold value. As mentioned the algorithm's functionality is based on the number of samples, it is then easy to see that the detection capability of the algorithm would decrease independent of the noise floor's level as the distance between tag and reader increases. The effectiveness of the threshold needs to be determined and if possible a more versatile threshold calculation method needs to be derived. The detection capability of the algorithm was tested for the tags at a specific distance from the reader. The algorithm's detection capability needs to be tested with tags at different distances simultaneously to see how effective the threshold will be.

9.2.4 ID threshold

The current level of the ID threshold needs to be revised. The algorithm detects some unmerited undefined bits. This usually occurs at a logic zero to logic one transition, when the logic one's energy level is higher than the rest of the bits. Figure 9.1 illustrates the problem. From the figure it is clear that the undefined bit's energy level is higher than the rest of the logic one and logic zero bits.

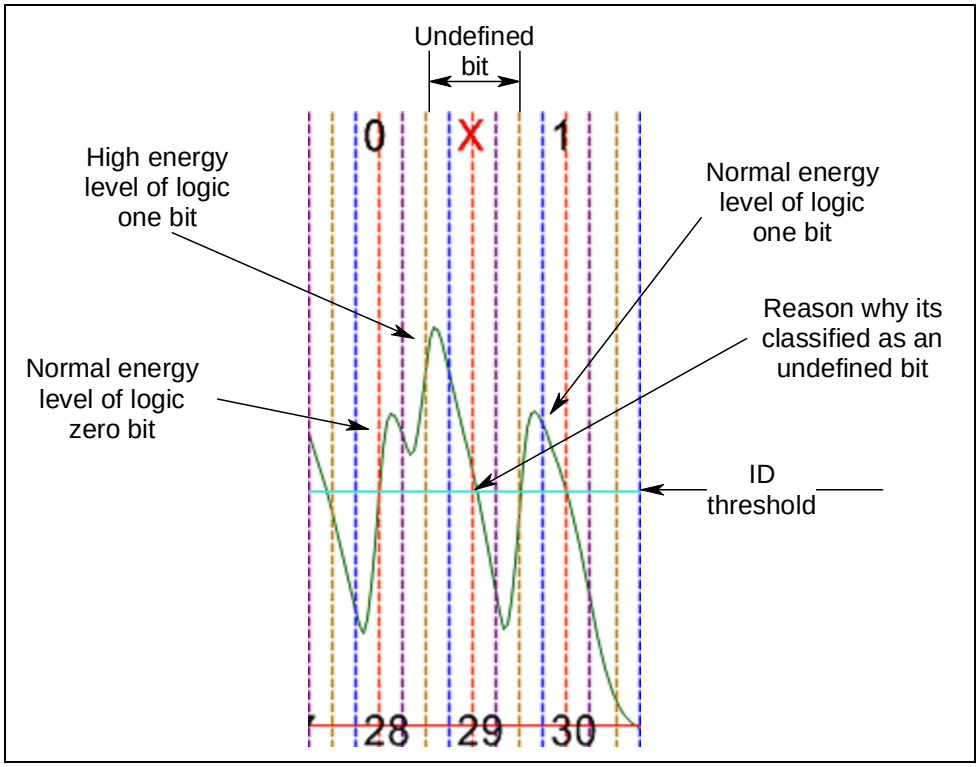


Figure 9.1 ID threshold error

The problem can be argued in three different manners,

- Is the extra energy of the logic one generated in the tag?
- Is the enveloping circuit to slow to accommodate the fast transition?
- Is this effect generated by the averaging filter?

Thus before an adjustment is made to the ID threshold calculations, the source of the problem should be determined and then only can appropriate action be taken.

9.2.5 Algorithm

Although the algorithm's detection capability is high, the implementation of the essence of the algorithm must be revised. The algorithm compares the energy or peak value of the first quarter with the third quarter's energy or peak value; this forms the essence of the algorithm. These quarters are defined by a number of samples, the part which needs to be revised. As mentioned in the previous sub–section this poses a number of problems for the algorithm. Any deformities in the tags signal are problematic for the algorithm. A more elegant implementation of the basis of the

algorithm, which is not based on the number of samples, could improve its detection capabilities.

9.2.6 ID storage

In chapter 7 a theoretical test was done to determine the adequacy of the transmission protocol of the detected IDs between the RFID reader and the RFID middleware. The transmission rate of the detected IDs as required by iPico, needs to be set. Then the current protocol needs to be examined to determine if it meets the requirements. If not, for example a protocol which sends the amount of times a specific ID was detected for a certain time frame can be used. This saves storages space and transmission time, because the complete ID is not transmitted or stored every time.

As stated numerous times throughout the duration of this dissertation, the core objective of the project was to implement an enhanced digital RFID detection algorithm in VHDL. This objective was successfully achieved. The results of the comparison tests in chapter 8 confirms the latter statement, with an almost 100 % correlation over the detection range.

The new simulations tool can also add more value to the design process of the algorithm. This is due to the fact that improvements in the detection capability achieved in the C++ algorithm would reflect in the VHDL implementation, provided a correct translation is made. This outcome of the project confirms that the C++ algorithm can be converted accurately to an equivalent VHDL implementation. With the improved functionalities of the simulation tool the effect of the changes to the C++ algorithm can be seen in a more time effective manner and with less effort.

Reference

[1] Glover, B. & Bhatt H. 2006. RFID Essentials. O'REILLY : Cambridge

[2] Finkenzeller, K. 2003. RFID Handbook. 2nd ed. John Wiley & Sons Ltd : England

[3] Document, "iPico Medium Range Dual Frequency Reader" and "iPico RO Passive Dual Frequency Tags", http://www.ipico.com (accessed on 2006/09/15), Include on the CD

[4] Griffiths, D.J. 1999. Introduction to electrodynamics. 3rd ed. Prentice Hall : New Jersey

[5] Lahiri, S. 2006. RFID Sourcebook. IBM Press : New York

[6] Chu, P.P. 2006. RTL hardware design using VHDL: Coding for efficiency, portability and scalability. John Wiley & Sons, Inc. : New Jersey.

[7] Blake, R. 2002. Electronic communication systems. 2nd ed. Delmar Thomson Learning : Canada

[8] Young, P.H. 2004. Electronic communication techniques. 5th ed. Pearson Prentic Hall : New Jersey

[9] EM Microelectronic – Marin SA. 2002. Multi frequency contactless identification device: Anti-collision compatible with BTG's supertag category protocols www.emmicroelectronic.com/webfiles/Products/RFID/ds/EM4022_DS.pdf (accessed on 2007/07/09)

[10] Document, iP-X3 DF Tag Specification v4. Include on the CD

[11] Oosthuizen, G. 2005. Executing computation intensive algorithms on digital hardware. Thesis, North-West University, Potchefstroom Campus.

[12] Bruen, A.A. & Forcinito, M.A. 2005. Cryptography, information theory, and error-correction: A handbook for the 21st century. John Wiley & Sons, Inc : New Jersey.

[13] Barr, M. 1999. Strength in numbers. Miller Freeman, Inc. www.netrino.com/Connecting/1999-12/index.php (accessed on 2007/07/11)

[14] Rumsey, F. & Watkinson, J. 2004. Digital interface handbook. 3rd ed. Focal Press (Elsevier) : Boston

[15] IP Ratings: IP explanation and ratings. 2006. Ventcroft Ltd. www.protectingpeople.co.uk/fire_tech/ip_explained.htm (accessed on 2007/07/18)

[16] Vorster, C. North-West University, Potchefstroom Campus. (The development of “Algorithm II” was part of his doctoral thesis, which was not completed at the time)

[17] Tijhuis, A.B.C. 2004. Digital detection system for RF-identification. Thesis, North-West University Potchestroom Campus.

[18] Bleazard, G.B. 1982. Handbook of data communications. The National Computing Centre Limited: Oxford, England

[19] Gibson, J.D. 1996. The mobile communications handbook. CRC Press, Inc. pp355-369

[20] Microsoft Visual Studio 2003 professional edition. Help files. (Used during the development of the PC software)

[21] Microsoft Visual Studio 2005 professional edition. Help files. (Used during the development of the PC software)

[22] Blum, C. Serial Communication Using RS-232. Barleywood : United Kingdom www.z80.info/1656.htm (accessed on 2008/02/13),

[23] Document, “Phase I”, Include on the CD.

[24] Document, “Phase II”, Include on the CD.

[25] Document, “Phase III”, Include on the CD.

[26] Document, “Phase IV”, Include on the CD.

[27] Ifeachor, E.C. & Jervis, B.W. 2002. Digital Signal Processing: A Pratical Approach., 2nd ed. Prentice Hall : New York.

[28] IP – Ingress protection ratings. www.EngineeringToolBox.com (Accessed on 2008/03/10)

[29] Zwolinski, M. 2000. Digital System Design with VHDL. 2nd ed. Pearson Prentice Hall : London.

[30] Documents in folder, “Digital Hardware”, Included on CD

Appendix A.

A.1. Introduction

IP (Ingress Protection) ratings are an international classification system developed by European Committee for Electro Technical Standardization, described in IEC/EN 60529 specification. The IP ratings specify the environmental protection an enclosure provides. [15], [28]

The IP rating is usually composed of two numbers, but sometimes three. The different numbers describe the following protection: [15], [28]

1. Protection from solid objects or materials.
2. Protection from liquids.
3. Protection against mechanical impact (commonly omitted).

A.2. The First IP number

Table A.1 Protection against solid objects [15], [28]

Number	Protection
0	No protection
1	Protected against solid objects up to 50mm, e.g. accidental touch by hands.
2	Protected against solid objects up 12mm, e.g. fingers.
3	Protected against solid objects over 2.5mm (tools and wires).
4	Protected against solid objects over 1mm (tolls, wire and small wires).
5	Protected against dust limited ingress (no harmful deposit).
6	Totally protected against dust.

A.3. The second IP number

Table A.2 Protection against liquids [15], [28]

Number	Protection
0	No protection.
1	Protection against vertically falling drops of water e.g. condensation.
2	Protection against direct sprays of water up to 15° from the vertical.

3	Protection against direct sprays of water up to 60° from the vertical.
4	Protection against water sprayed from all directions, limited ingress permitted.
5	Protection against low pressure jets of water from all directions, limited ingress.
6	Protection against low pressure jets of water, e.g. for use on ship decks – limited ingress permitted.
7	Protected against the effect of immersion between 15cm and 1m.
8	Protects against long periods of immersion under pressure.

A.4. The third IP number

Table A.3 Protection against mechanical impacts [15], [28]

Number	Protection
0	No protection.
1	Protects against impact of 0.225 joule (e.g. 150g weight falling from 15cm height).
2	Protects against impact of 0.375 joule (e.g. 250g weight falling from 15cm height).
3	Protects against impact of 0.5 joule (e.g. 250g weight falling from 20cm height).
4	Protects against impact of 2.0 joule (e.g. 500g weight falling from 40cm height).
5	Protects against impact of 6.0 joule (e.g. 1.5kg weight falling from 40cm height).
6	Protects against impact of 20 joule (e.g. 5kg weight falling from 40cm height).

A.5. Example

An IP rating of IP 23, the 2 describes the level of protection from solid objects and 3 describe the level of protection from liquids. [15], [28]

Appendix B.

B.1. Introduction

A detail description of the design of the new simulation software (Simulator) will be discussed. The design of the different modules is discussed and a comprehensive discussion of the design of the DLL (Algorithm in C++ implementation). This will help future programmers to understand the structure of the code, in order to make improvements or correct errors which are detected. No known errors currently exist.

B.2. Graphical user interface

The GUI was written in VB with Visual Studio 2005. The GUI is composed of four forms, with their classes, and two functional classes

- Forms
    1. FormMainWin: Main display form.
    2. Settings: Set graphical display preferences.
    3. Constants: Set algorithm's parameters (constants in algorithm)
    4. Tag_Results: Display a summation of the different tags detected and the number of times they were detected.
- Classes
    1. DrawAlg: The algorithm's feedback is drawn by this class.
    2. DrawData: The filtered data is drawn by this class.

B.2.1. FormMainWin

A complete flow diagram of the design for this class is provided on the CD (Not display here due to size). This form is the main control and display form, all functionalities are access from this from. The form provides the user with a number of functions. The various functions will be discussed in the course of this appendix. To gain access to these functions a data set must first be loaded. The only function available before data is loaded is the load function.

The program allows the user to load only csv and txt files. Furthermore the format of these file must have the following format, one sample per line and complies with 12–

bit ADC values. The compliance of the data in a set is tested, must be 12-bit ADC values, thus values between 0 and 4095.

The number of samples was first limited to 500 000 samples, because this was the length of the previous data sets. To conduct the comparison test (C++ vs VHDL implementations), the limit was increase to 2 097 151 (see Appendix E).

Figure B.1 provides a visual illustration of FormMainWin's form.

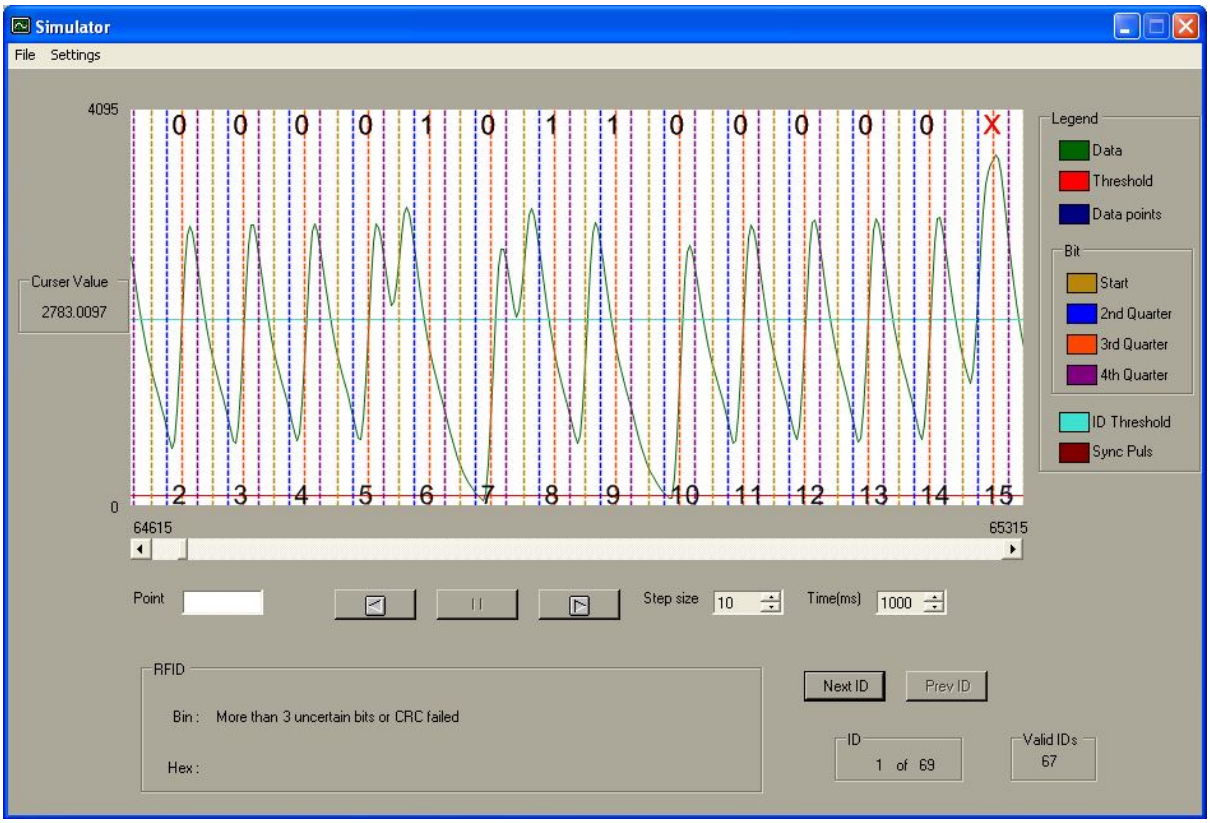


Figure B.1 FormMainWin form

The maximum and minimum values found in the data set are displayed on the y–axis. These limits can be set by the user after the data was loaded, with the settings function (see Settings form). The current position in the data set is displayed on the x-axis.

The user is able to read any y value of the graph by moving the mouse cursor to the desired position. The pixel index of the y-axis has an error factor (number of pixels limit resolution), thus the cursor value displayed may differ from the actual value. (An

upgrade of the simulator may read the values from the arrays instead of using the pixel index value, for a more accurate value.) A legend is provided to indicate what the different coloured lines represent.

There are currently four ways to step through the response data. A critical point, the data displayed is not the raw data, but the average filter's output scaled. The first is the scroll functions, currently implemented by the scroll bar. The large scroll steps can be set on the form in the step size box.

The second is the data position function. This allows the user to go to a certain position in the data array. The position entered will appear at the left side of the graph. The program prevents the user to enter invalid data.

The third function is an auto scroll function. The user directs the program to automatically scroll through the data. The step size of the frames complies with the step size of the manual scroll function (first function). The tempo the program steps through the data can be set, in the time box. This function also has a pause function, if the user wants to examine the data more carefully.

The number of possible IDs as well as the number of valid IDs is display. The possible IDs in this case refer to all the valid synchronization sequence detected. The possible detections which exited in the synchronization sequence is displayed, but not included in the possible IDs count. For a summary of the IDs detected see IDs function (Tag_Results form).

When a valid ID is displayed its ID value is displayed in binary and hexadecimal format. This value is the valid CRC value. The undefined bits are indicated on the graph with a red X, at the specific bit.

The graph displays a number of information.

- The number of the current sync bit.
- The number of the ID bit with its logic value.
- The bits' quarter lines, as define by the algorithm.

- The algorithm activation threshold.
- The ID threshold.
- The filtered data.

B.2.2. Settings

A complete flow diagram of the design for this class is provided on the CD (Not display here due to size). The settings form allows the user to set a number of display options.

Figure B.2 is an example of the Settings form.

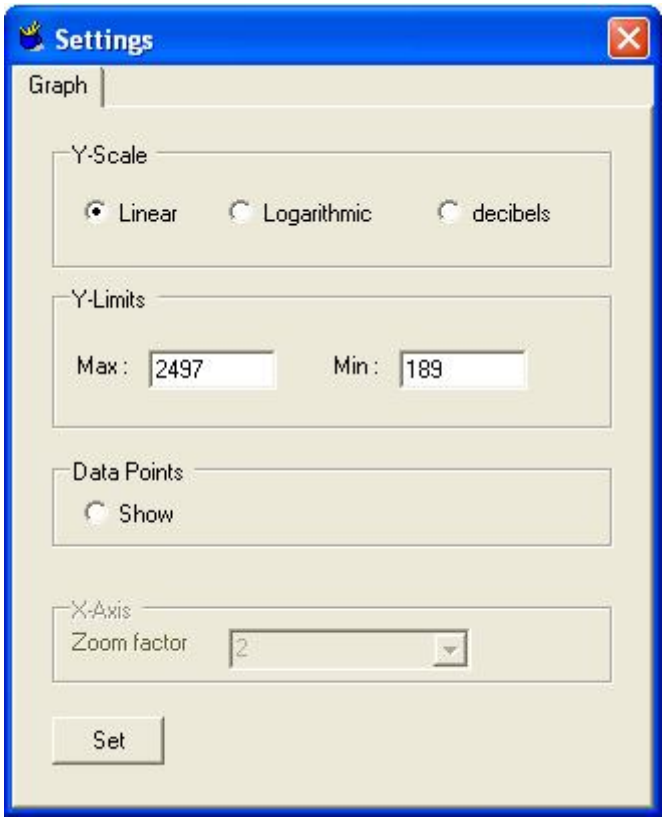


Figure B.2 Settings form

There are three scales the user can choose from. The default is a linear scale. Then there is a logarithmical and decibels scale. The y-axis scale's maximum and minimum values can also be set on this form.

The user has the option to display the data points. This can be useful when debugging the algorithm.

B.2.3. Constants

A complete flow diagram of the design for this class is provided on the CD (Not display here due to size). The Constants form enables the user to change the parameters of the algorithm. When the parameters are changed the algorithm is called again with the new parameters. The effects of the changes can thus be seen immediately on the whole data set.

Figure B.3 illustrates the Constants form.

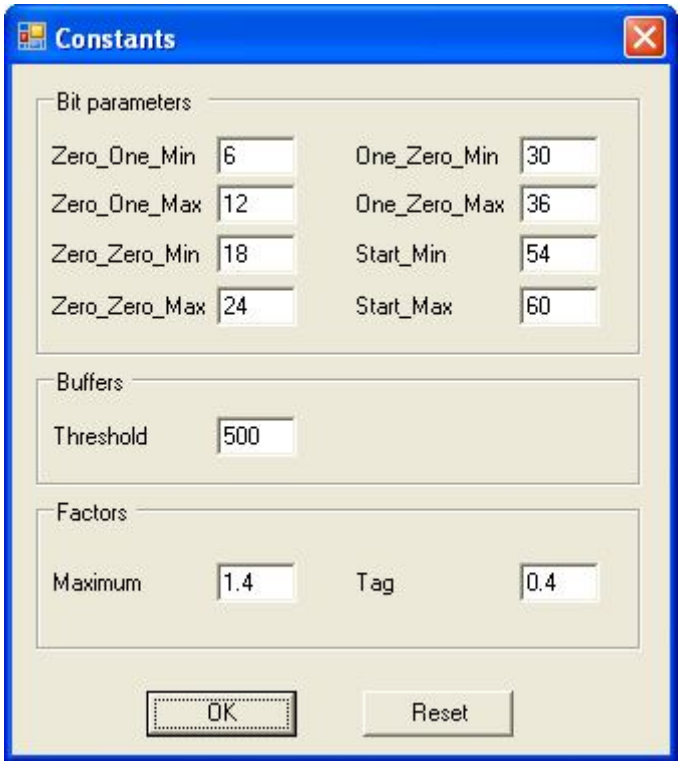


Figure B.3 Constants form

The parameters the user can change are:

- The algorithm defined quarter lines.
- The algorithm defined distance from the last sync bit to the start bit's first quarter.
- The constant value added during the threshold calculation.
- The noise limit factor during synchronization detection.
- The factor use in the calculations for the ID threshold.

The default values of the algorithm can be retrieved by using the reset function. These defaults values are the current determined optimum values.

B.2.4. Tag_results

This form displays the different ID values detected in the data set and the number of times they occurred. The information are retrieved from the two dimensional Tag_ID_val array (see DrawAlg). It provides the user with a summation of the IDs detected in the data set.

Figure B.4 illustrates the design of the class and figure B.5 is an illustration of the form.

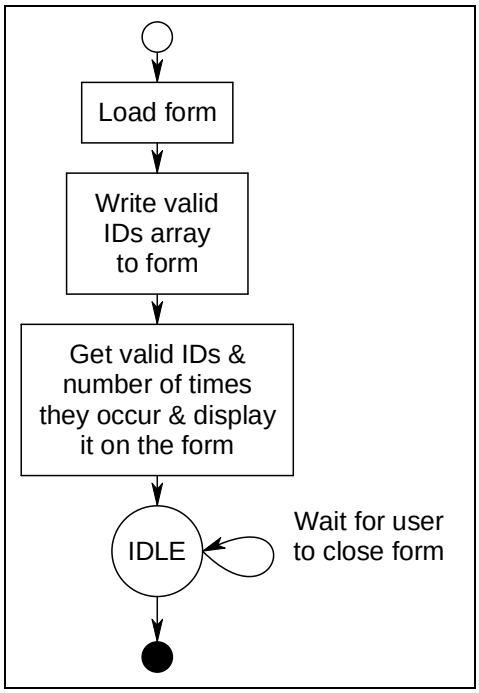


Figure B.4 Tag_results design

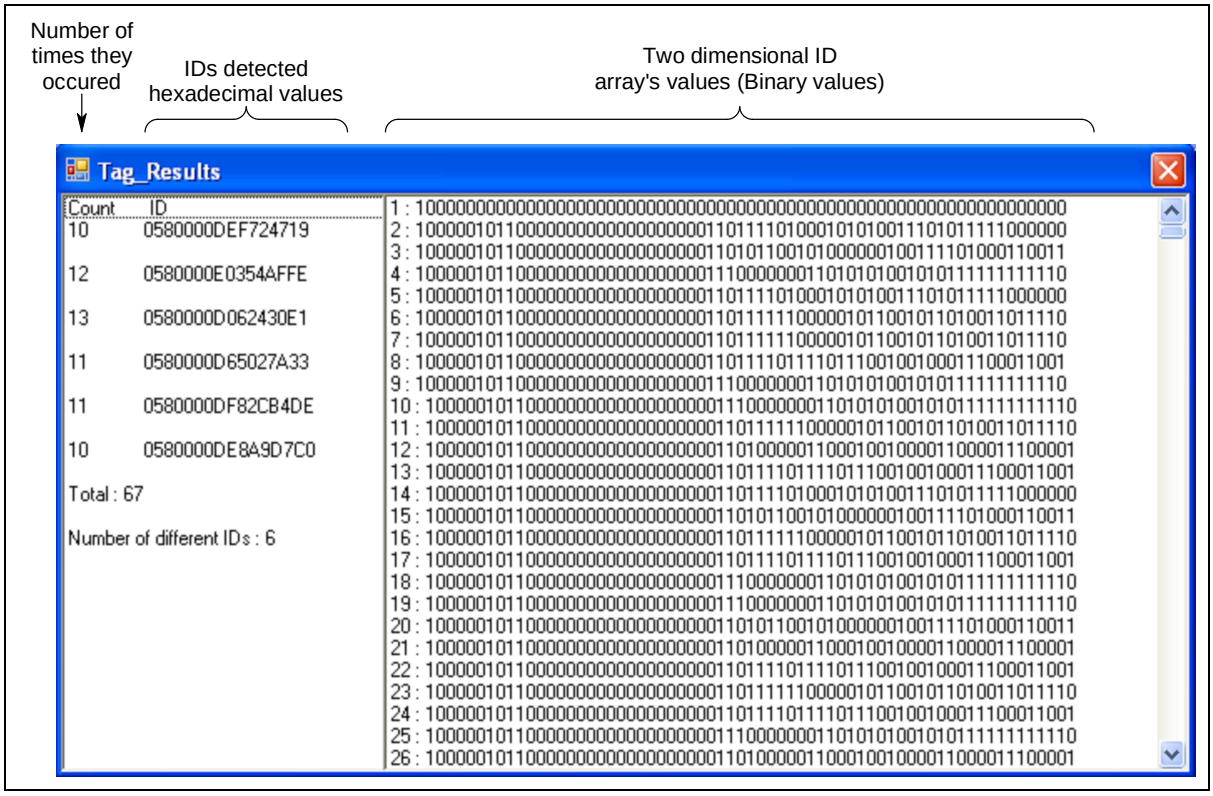


Figure B.5 Tag_Results form's layout

B.2.5. DrawData

All the aspects and functions of the sampled data are housed in this class. The class design is discussed below.

Fields:

- Data – this array stores the sample data and the filtered data after the algorithm is called.
- Data_Reset – this array stores the sample data and is used to reset Data array when the user changes the algorithms parameters.
- dataPen – define the style of the data points on the graph.
- graphPen – define the style of the filtered data on the graph.
- Pixel – This array stores the pixel values of the filtered data for the current pixel index value.
- show_data – this variable indicates whether the user selected to display the data points.
- User_H_Limit – this variable indicates if the user defined maximum value or maximum value of the sample data will be use as upper limit of the graph.

- User_L_Limit – this variable indicates if the user defined minimum value or minimum value of the sample data will be use as the lower limit of the graph.
- valHeight – stores the height of the graph.
- valIndex – stores the pixel index value
- valLength – stores the number of samples imported.
- valMax – stores the current maximum value (user defined or sample data).
- valMIn – stores the current minimum value (user defined or sample data).
- valScale – stores the current selected scale, default linear.
- valWidth – stores the width of the graph.

Properties:

- Height – height of graph
- Index – the value a pixel in the graph's y-axis represents.
- Length – number of samples in data set.
- Max – Maximum value of graph.
- Min – Minimum value of graph.
- Scale – scale of graph.
- Width – width of graph.

Methods:

- CalculatePixels – convert the sample values to pixel values and update the pixel array. This method also calculates the pixel index value which is used by DrawAlg.
- DrawGraph – draw the current frame's filtered data.
- WriteText – write text on the graph, parameters specify size and location.

B.2.6. DrawAlg

All the aspects and functions of the algorithm's response are housed in this class. The class design is discussed below.

Fields:

- Bit_val – this array is updated by the algorithm and stores the quarter bit lines.
- fourqPen – define the style of the fourth quarter line on the graph.

- ID_Bin – this array is updated by the algorithm and stores the ID's binary value of a specific bit and the synchronization bit count value for the specific bit.
- ID_locate_val – this array is updated by the algorithm and provides location information for the simulator where possible IDs were detected.
- secndqPen – define the style of the second quarter line on the graph.
- startbitPen – define the style of the start line of a bit on the graph.
- syncpen – define the text style of the synchronization count values on the graph.
- syncweight_val – this array is updated by the algorithm and stores the ID threshold values.
- Tag_ID_val – this two dimensional array is updated by the algorithm and stores the binary values of the valid IDs detected. Where the columns represent the number of the ID detected and the rows its binary value.
- Tagcount_val – this array is updated by the algorithm and stores the current number of the ID bit value detected, range 1 to 64.
- thirdqPen – define the style of the third quarter line on the graph.
- Thres_ID_Pix – this array stores the pixel values of the ID threshold.
- Thres_IDPen – define the style of the ID threshold on the graph.
- Thres_Pix – this array stores the pixel value of the algorithm activation threshold.
- Thres_val – this array is update by the algorithm and stores the algorithm activation threshold values.
- thresPen – define the style of the algorithm activation threshold on the graph.

Methods:

- CalculatePixels_Thres – convert the algorithm activation threshold values to pixel values and update Thres_Pix array.
- CalculatePixels_Thres_ID – convert the ID threshold values to pixel values and update Thres_ID_Pix array.
- Draw_Bit – draw the bit lines for current frame.
- Draw_Thres – draw the algorithm activation threshold for current frame.
- Draw_Thres_ID – draw the ID threshold for current frame.

- Hex_con – convert the current valid ID displayed, binary value to hexadecimal value.
- Write_ID_count – writes all the text information on the graph.
- Write_Tag – write the current valid ID's binary and hexadecimal value on the main form.
- Writetext – any other information which must be written on the graph calls this method.

B.3. DLL (C++ algorithm)

The DLL houses the C++ RFID digital detection algorithm. The DLL can only be accessed through one function, namely "Algorithm". This function receives memory location of the different arrays declared in the VB part of the simulator. These memory locations are assigned to pointers in the DLL. This function controls the algorithm and calls the internal function as required.

As mentioned previously the algorithm was divided into different functional parts. Each of these functional parts is a function in the DLL.

- Filter – house the averaging filter of the algorithm. It takes no parameters and returns the next filter output.
- Limit – house the algorithm activation threshold calculations. It takes the filter output value as parameter and returns nothing.
- Sync_detect – house the synchronization bits, 2 no-transmission bits and start bit detection method. It takes filter output as input and returns nothing.
- Data_detect – house the ID detection method. It takes the filter output as input and returns nothing.
- CRC – house the CRC function which controls the CRC checks for the different undefined bit's possible combinations. Its takes no parameters and returns nothing.
- CRCcheck – implements the CRC algorithm to determine if the CRC is valid. It takes no parameters and return a true (valid CRC) or false (invalid CRC).
- set_graph – this function updates most of the arrays use for display purposes in the VB part. This correlates as the response of the algorithm. It takes filter output as input and returns nothing.

Appendix C.

C.1. Introduction

To use the new simulation tool in the algorithm design prosses, data was needed to test the algorithm with. Furthermore the parameters of the IDs in the data set must be changeable. A Matlab® M – file was used to generate theoretical IDs, which parameters could be changed. The M – files is included in the CD.

The data set generator is composed of three parts.

- Sample
- Tag ID
- Hexadecimal conversion

C.2. Sample

The sample function is called from the command line with the specified parameters of the IDs and data set. The following parameters must be given:

- *Ftag* – The frequency of the tag.
- *Fsam* – The sampling frequency.
- *ID* – The hexadecimal value of the IDs the user wants in the data set.
- *Aid* – The amplitude of the IDs.
- *N* – The number of IDs the user wants in the data set.
- *Name* – Specifies the name of the csv file for the data set.
- *D* – The distance between the IDs.
- *SN* – The noise floor in the data set.

The ID parameter must be a string array with one ID per row. The IDs' values must be in hexadecimal. The IDs specified in the array will be used randomly in the data set to satisfy the number of IDs the user indicated. The M – file displays the number of times a specific ID is used in the data set in the command window. The M – file is written for 64 bit IDs only.

Aid parameter allows the user to specify the amplitude of the IDs. The user can use this parameter if a specific number of ADC bits are used. For example the hardware for this project is a 12 – bit ADC, so 4095 is used in the data sets for the project.

As mentioned the N parameter specifies the number of IDs in the set where as the ID parameter specify what the IDs' values must be. The D parameter specify the distance, in number of samples, the IDs must be apart.

The sample M – file first generate an array of the specified IDs by calling the Tag ID M – file. The IDs is converted into sample format in the Tag ID file. Then a random generator selects an ID and places it in the data set. An array is update to indicate which ID was used. This process is repeated until the number of IDs specified is met.

The algorithm then writes the data set to a csv file. The csv file's name is specified by the Name parameter. Just be before the program exits the specified IDs and the number of times it was used is written to the command window (see figure C.1).

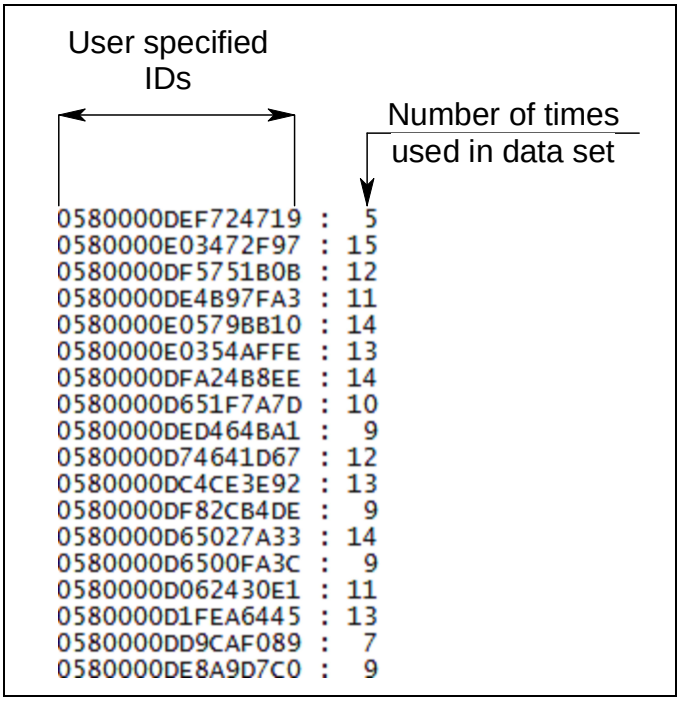


Figure C.1 Number of times ID is used in data set

C.3. Tag ID

This M – file converts the hexadecimal ID value to sample data, for the specific tag frequency and sampling frequency. The hexadecimal ID value is first converted to a binary string value by the Hexadecimal conversion M – file.

The principle on which the function works is time based. An array is declared, which length is equal to the number of samples an ID would have, with the IDs frequency and sampling frequency in mind. Each sample thus represents 1/sampling frequency time increment. The array is initialized with the specified noise floor.

The function steps through the array in increments of 1/sampling frequency. The binary value of the specific bit is obtained from the binary string of the ID. Thus depending whether it is a logic one or logic zero, the appropriated quarter bit samples are set to the specified amplitude. The quarter is determined by the tag frequency and a test is conducted to determine whether the time value is inside the quarter or outside.

C.4. Hexadecimal conversion

This M – file is a switch statement which steps through the 16 hex values of the ID and converts them to binary values. The output of the function is a 64 – bit binary string.

C.5. Data

As mentioned in the introduction this data generator, generates theoretical data. Figure C.2 (a) is an example of the generator's synchronization bits of an ID and (b) is an example of real synchronization bits as sampled by the ADC.

Although the theoretical data does not look like the real data, it enables the user to get an indication of the algorithm's detection capability. The main reason for the generator is to generate ID set with different data frequencies. This method presented an easy and quick way to vary the data frequency.

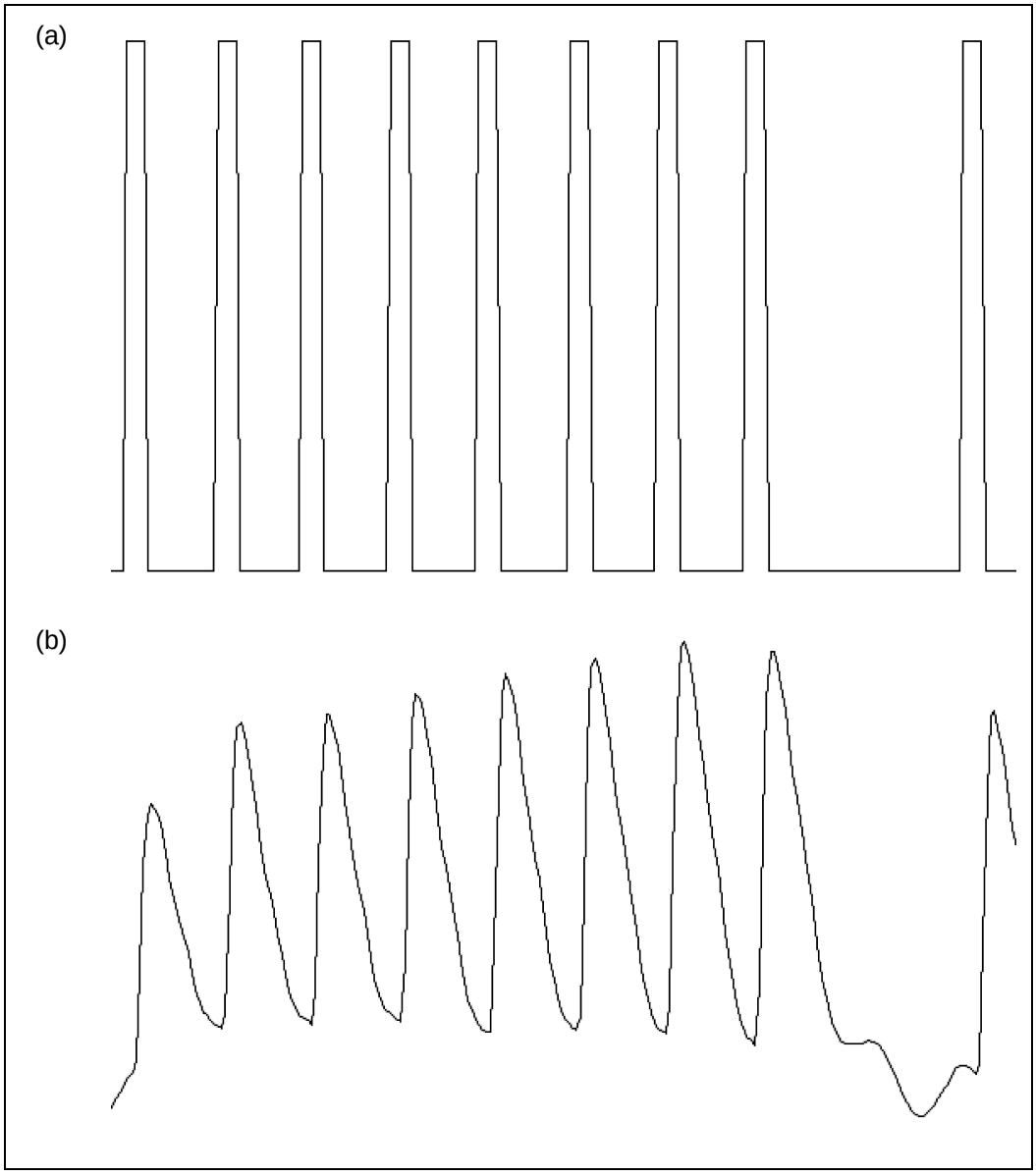


Figure C.2 Generated data, (b) Sampled data

Appendix D.

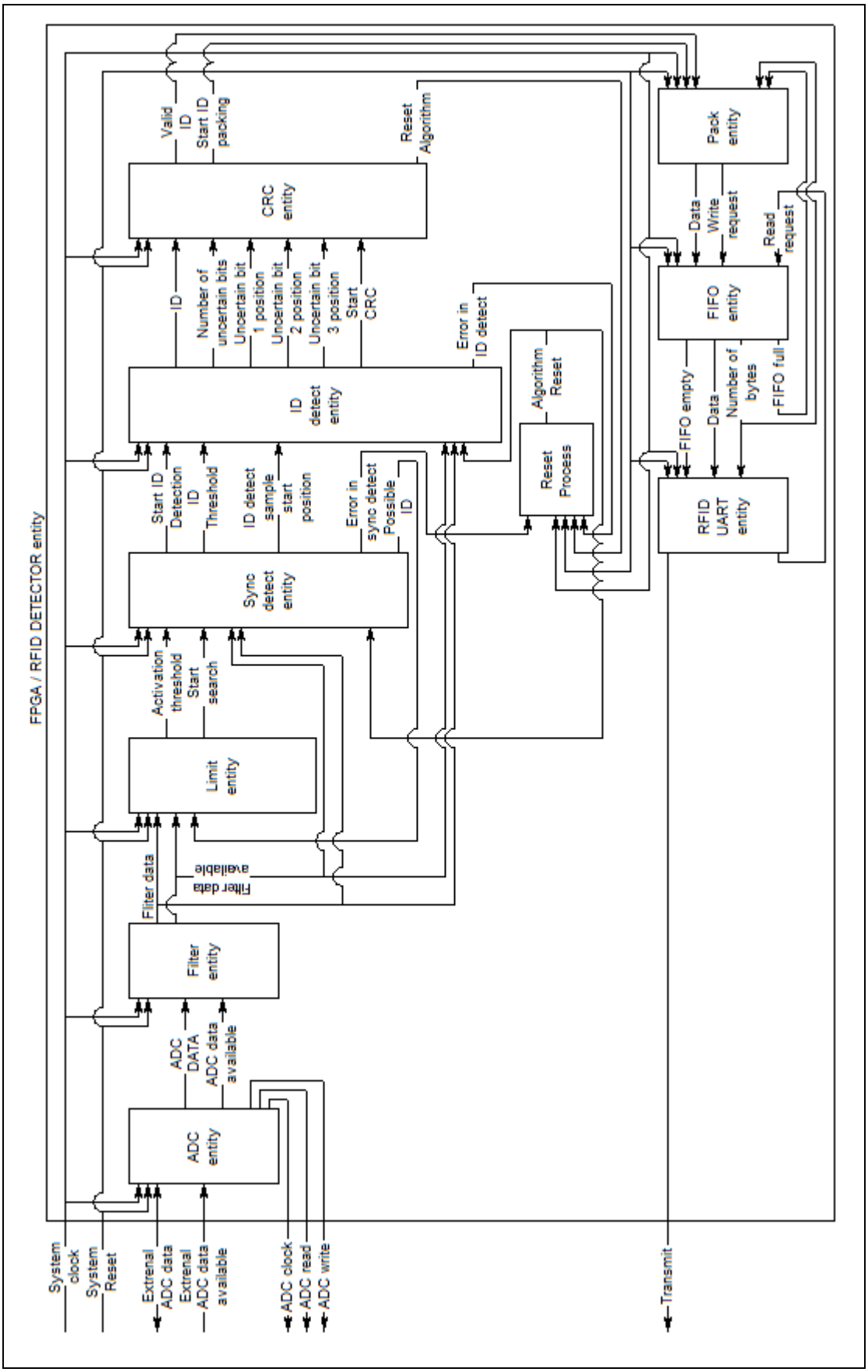


Figure D.1 Entities' connectivity

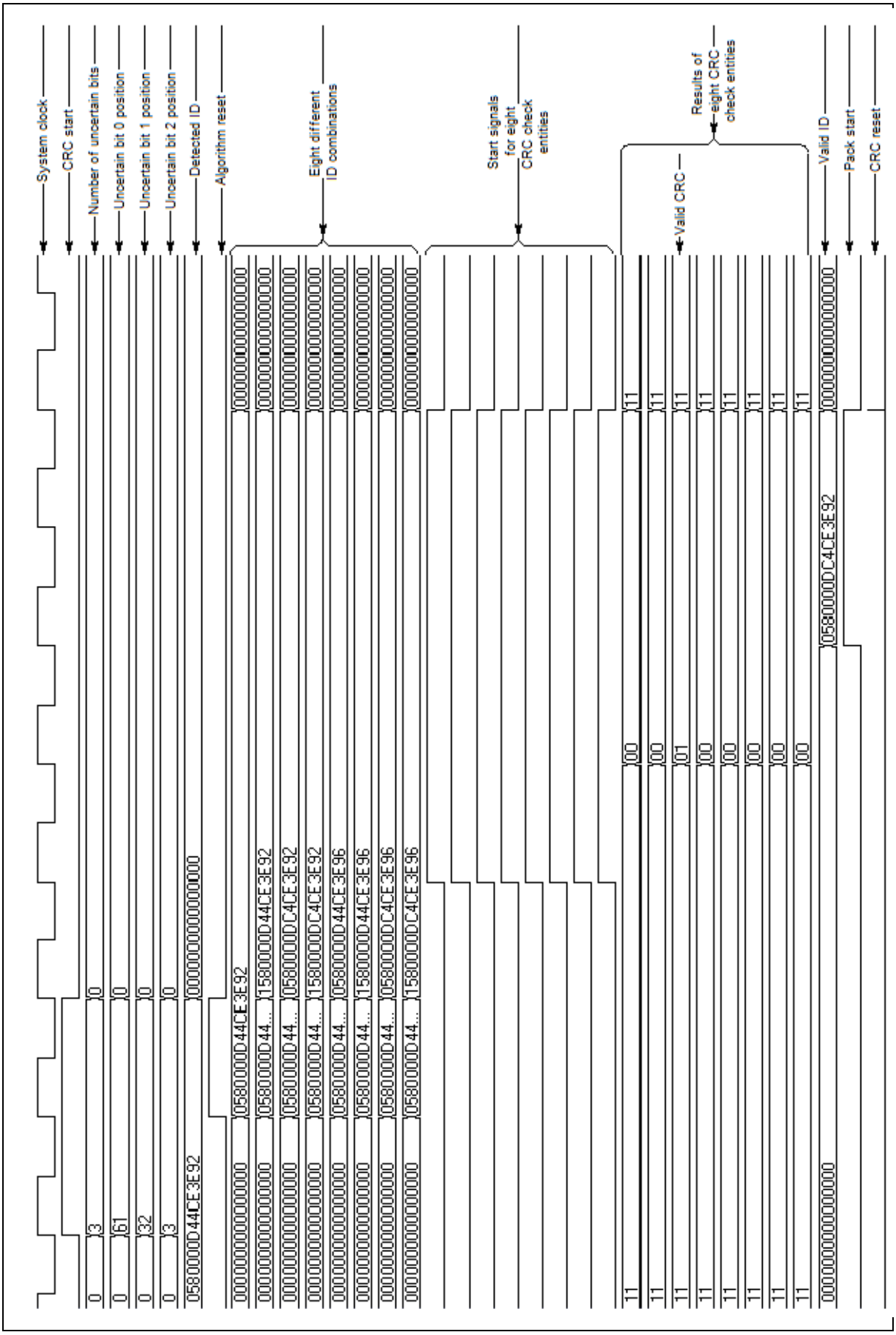


Figure D.2 CRC validating signals

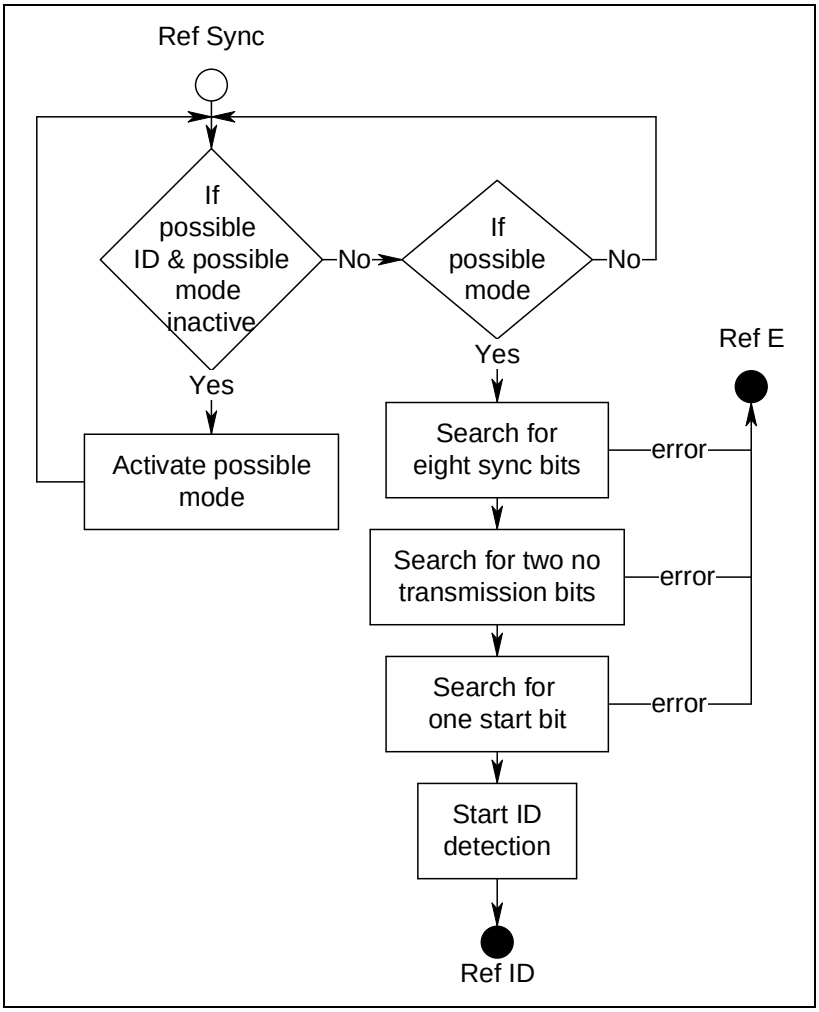


Figure D.3 Synchronization bits detection's structure

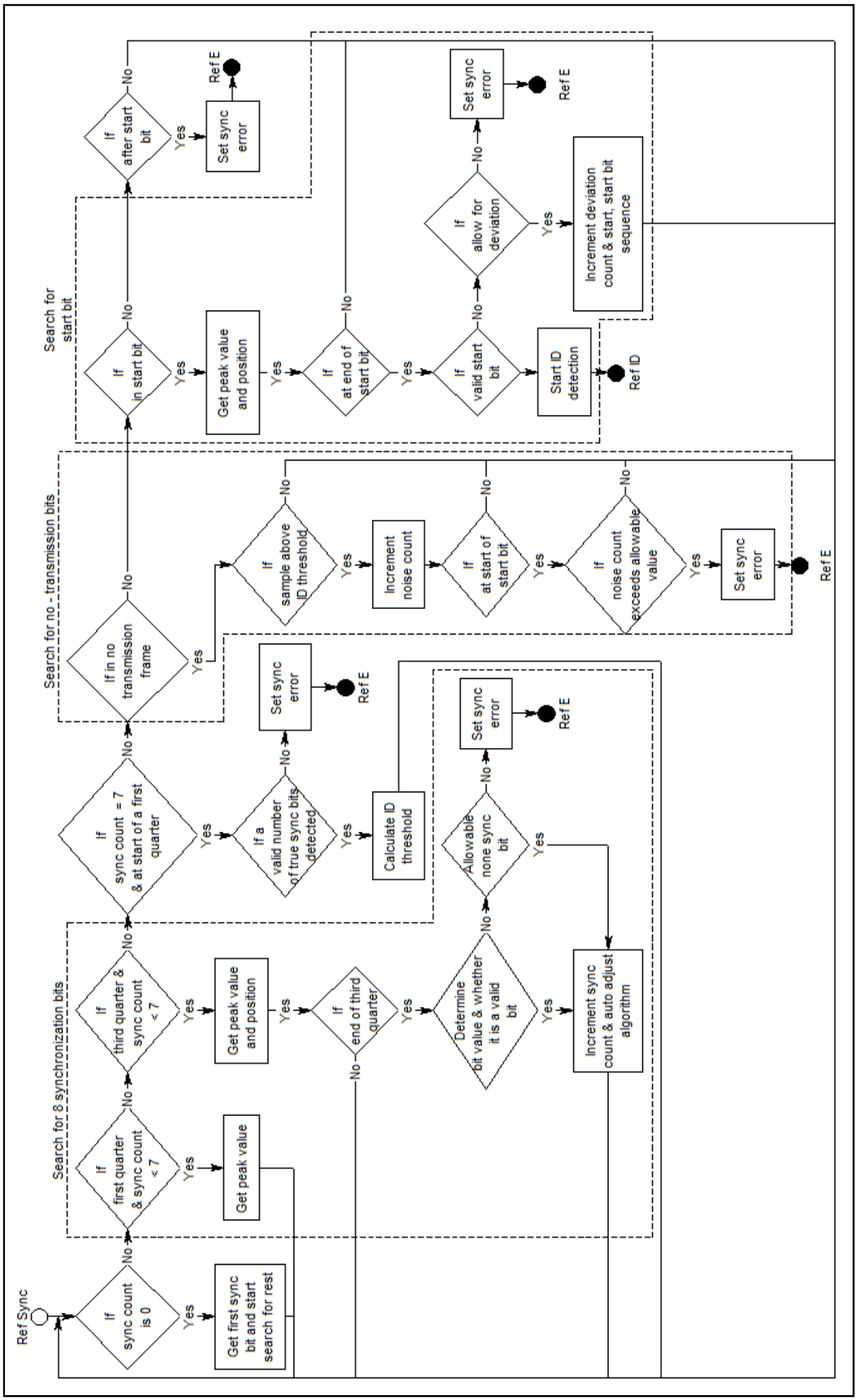


Figure D.4 Detail of sync bits detections

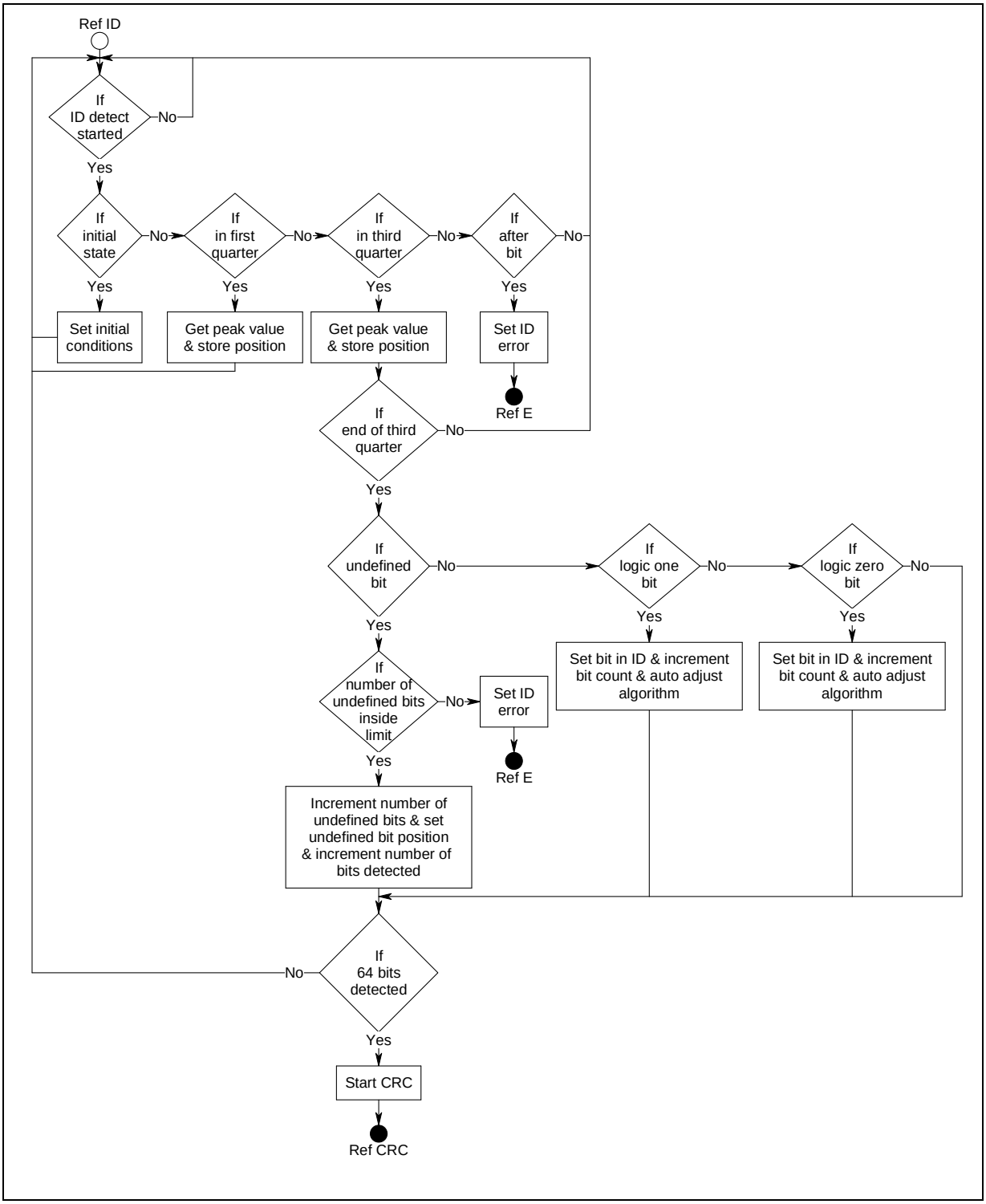


Figure D.5 ID bits detections structure

Appendix E.

E.1. Introduction

To accurately compare the performance of the C++ implementation against the VHDL implementation of the algorithm, the same data must be used in the different implementations. The design of the hardware, firmware and software necessary for this task will be discussed in this chapter.

The algorithm's functionality is based on a continuous data stream. The section of the algorithm which binds it to this statement is the algorithm's activation threshold. Thus to compare the two implementation, with the algorithm in optimum conditions, a continuous data stream is necessary. ID data can also be lost if a continues data stream is not used.

There are basically two options in order to obtain a continuous data stream. Firstly store a certain time frame's data on the hardware or secondly continuously send the data to an external storage source. The continuous transmission of the data was examined first. The only serial protocol capable theoretically to handle the transmission rate is USB 2.0.

This approach was abandoned due to the cost, time to implement and steep learning curve of the protocol. The second possibility was explored and the storage medium, decided on was SRAM (static random access memory). The reason for this is because the data can be written to the media in parallel and at high speeds (at a frequency must greater than clock cycle).

E.2. Hardware

The current FPGA board was design for debugging purposes. Thus additional hardware can be add and still interface with the FPGA. The external memory could easily be added, with an add-on board. The new board is composed of three units (see figure E.1), namely

- Storage unit or SRAM
- Communication
- PSU (power supply unit)

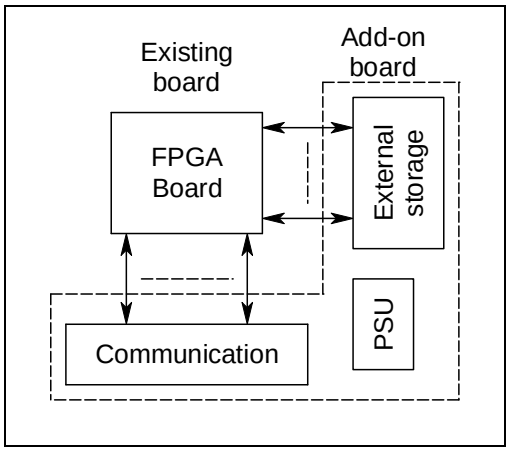


Figure E.1 Add-on board's components

The storage unit is composed of two SRAM ICs. The SRAM ICs used is the CY7C1061AV33 (datasheet included on CD) form Cypress Semiconductors Corporation. It was decided to use this SRAM due to the following characteristics of the IC:

- Data bus 16 bits wide, no need to divide the 12-bit ADC value into bytes.
- Storage capacity, 16 Mbit (1M x 16).
- Fast access time, 10 ns.
- Availability, due to time constrain.
- Uncomplicated read and write cycles, simplifies firmware design.
- 3.3 V operating voltage compatible with FPGA.
- Easy memory expansion, two chip select inputs.

The external storage unit can store 2097151 (2 M) samples. At a sampling frequency of 3.072 MHz, a continuous time frame of 0.68s is captured.

The number of samples which must be transfer is quite large. The current communication available on the FPGA board is RS232. The PC's RS232 port can

accommodate a board rate up 115200 bits/s. To transfer 4 Mbytes to the PC will take about 6 minutes. Thus it will be a time consuming exercise.

There are a number of USB/UART bridges ICs on the market. These devices' maximum board rate is 921600 bits/s. This reduces the transfer rate to about 45s, 8 times faster. Furthermore the manufactures provide royalty-free software, which eliminates USB driver development. This is either a virtual serial port or DLL functions. The DLL's functions communicates with device, thus the software design is simplified.

For the reasons mentioned above the FT232R USB UART IC from Future Technology Devices International Ltd. (FTDI^{TM} chip) is used. Also the availability of the IC was a factor and the operation voltage of 3.3V (3.3V regulator inside the IC). The FPGA digital outputs and inputs operate at 3.3V

The power supply unit uses the LM1085 – 33 voltage regulator from National Semiconductors. The board was originally intended to have four of the SRAM IC, thus the power supply unit's current capability can accommodate four SRAM ICs. To save time and overcome routing difficulties they were reduced to two. A multilayer board can be design if more memory is required in the future to overcome the routing difficulties.

One SRAM IC uses a maximum of 300mA. Thus 600mA total, but originally the total current requirements were 1.2A. For this reason the LM1085 was chosen. It can supply up to 3A. Because of the original design the LM1117 regulator was discarded, because it can deliver only 800mA. The USB/UART IC power is supplied by the USB bus.

The schematic design and layout of the add-on board is available on the CD. Figure E.2 illustrates the add-on board and figure E.3 the existing FPGA board.

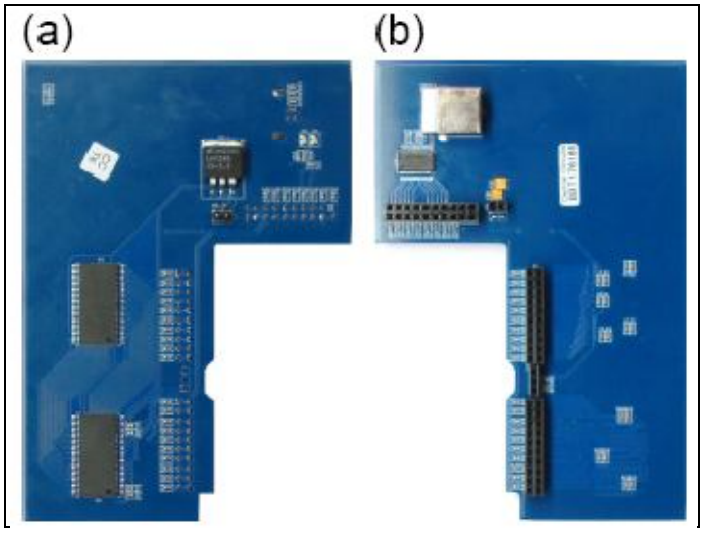


Figure E.2 Add-on board (a) Top and (b) Bottom.

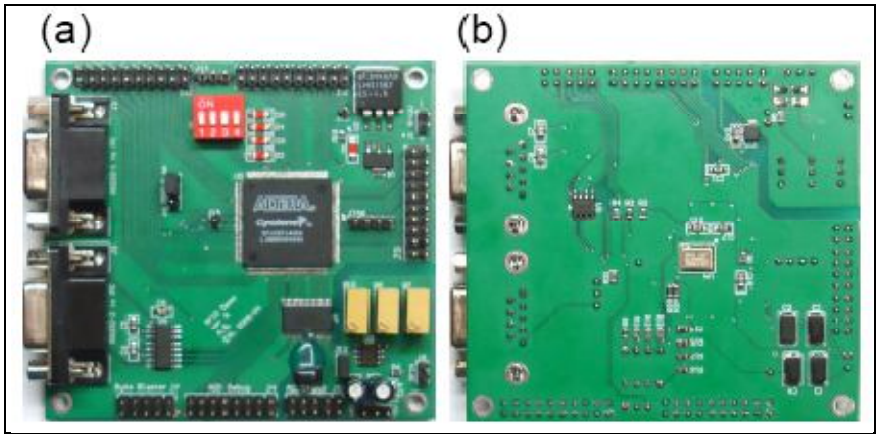


Figure E.3 Existing FPGA board, (a) Top and (b) Bottom.

E.3. Firmware

This section will discuss the firmware design necessary for storing and transmitting (to PC) the sample data and the PC control software in the next section. The SRAM firmware is incorporated in the algorithm's code by making use of a *component*. The component handles all the task of the SRAM and the new communication unit. The component does not interfere with the algorithm. The only interfering signal is the ADC stop signal, which will stop the algorithm's detection process (new sample data are stopped), but not the transmission of valid detected IDs in the FIFO stack.

Figure E.4 illustrates the path the sampled data travels.

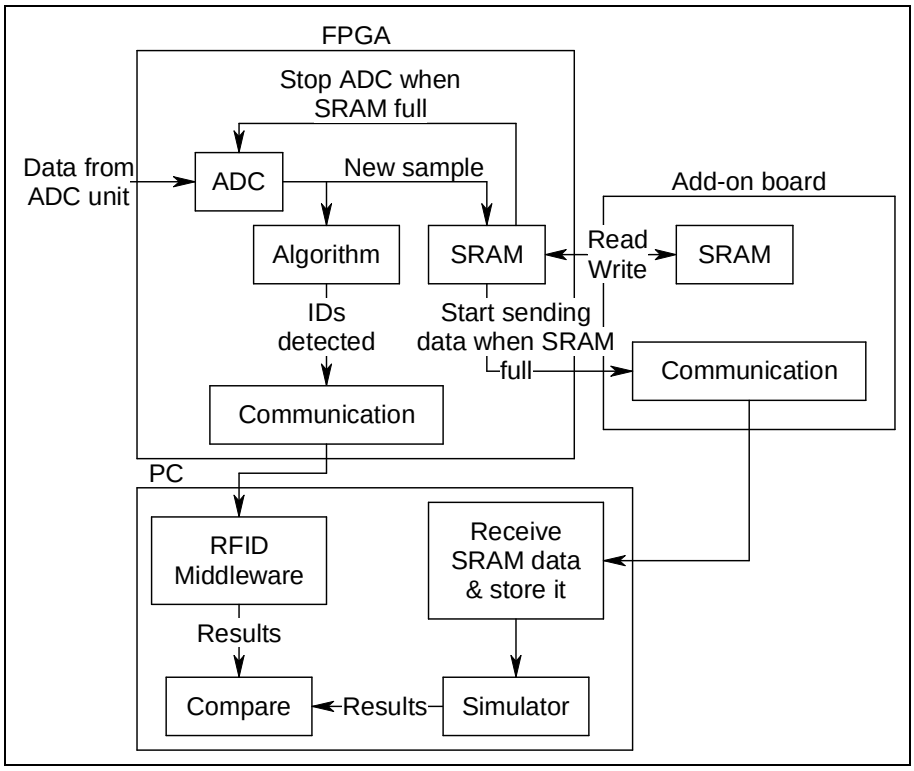


Figure E.4 Data flow diagram.

The add-on board is controlled from the FPGA. The *entity* housing the control code is the SRAM Interface *entity.* Figure E.5 illustrates the inputs and outputs of the SRAM Interface *entity*.

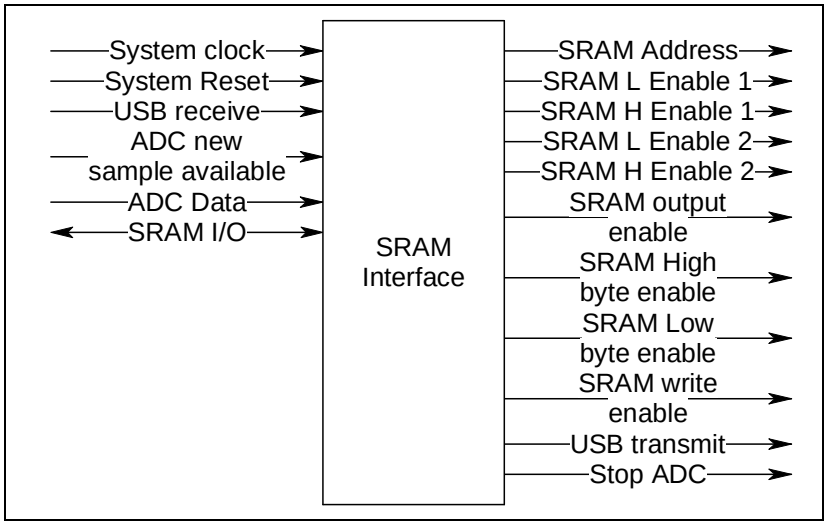


Figure E.5 SRAM entity's inputs and outputs

The *System clock* and *System Reset* inputs are the same signals as described in chapter 7. The *USB receive* input comes from the USB UART bridge chip and is the RS232 receive pin. The *ADC new sample available* and *ADC Data* inputs comes from the ADC entity, see chapter 7. The *SRAM I/O* is the data bus for the SRAM and is both an input and output.

The *SRAM Address* output is the address bus of the SRAM. The *SRAM L Enable* and *SRAM H Enable* outputs is the chip select signals. There are two per SRAM IC. These signals are used to expand the SRAM storage capability.

The *SRAM output enable* is a read enable signal, but its internal function in the IC is to set the I/O pins as output pins. The *SRAM High byte enable* and *SRAM Low byte enable* indicates for the SRAM when the high and low byte is valid. These signals provide the option of using a byte or 2 bytes interface with the SRAM. In this project the 2 bytes interface is used, in order to transfer 16 bits simultaneously.

The *SRAM Write enable* output is use to put the SRAM in a write state. The *USB transmit* output is the transmit pin for the USB UART bridge. The *Stop ADC* output is used to stop the ADC entity, which in turn stops the algorithm from detecting. This signal does not stop the transmitting of valid IDs already detected. This signal can only be reset, with *System Reset* signal, which is generated externally by the user. This prevents corruptions of the stored samples and the algorithm will not start detecting again, whiles the user is busy on the PC.

Figure E.6 illustrates the architecture of the SRAM Interface *entity*. From the figure the three components which are used to control the add-on board's hardware can be seen. The Controller *entity* directs the data to the appropriate memory locations through the SRAM Read Write entity and generates the *Stop ADC* signal when the SRAM is full. Its other function is to control the sending of the data to the PC. It retrieves the data from the SRAM through the SRAM Read Write entity and divide the 16 bits into two 8 bits for transmission.

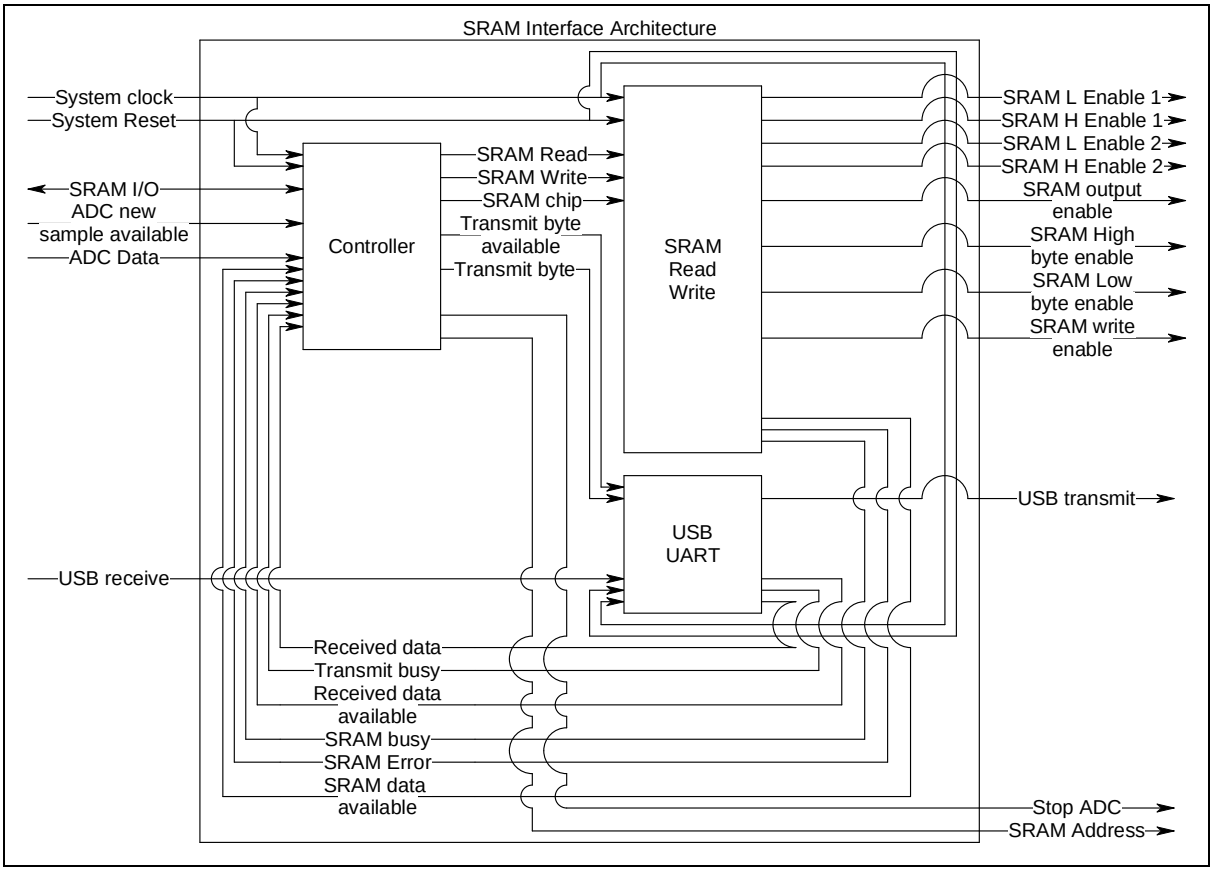


Figure E.6 Components of SRAM Interface and their connectivity

Figure E.7 (a) illustrates the Control State Machine of the Control *entity*. From the figure it can be seen that the state machine has a none return path, when in the Stop state. The state machine will stay in this state until the user resets the FPGA externally.

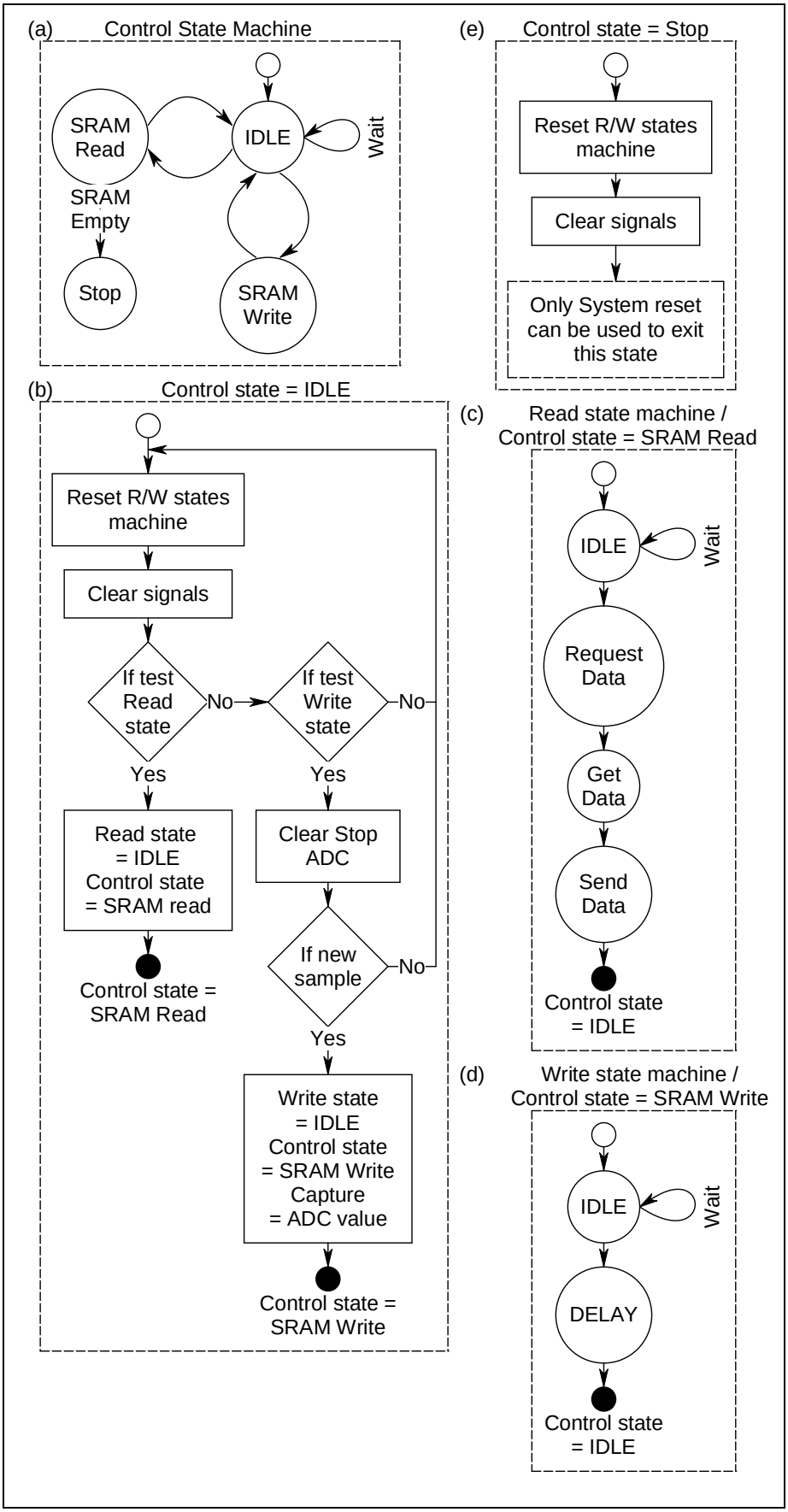


Figure E.7 (a) Control entity's Control State Machine, (b) Control IDLE state, (c) Control read state, (d) Control write state and (e) Control stop state.

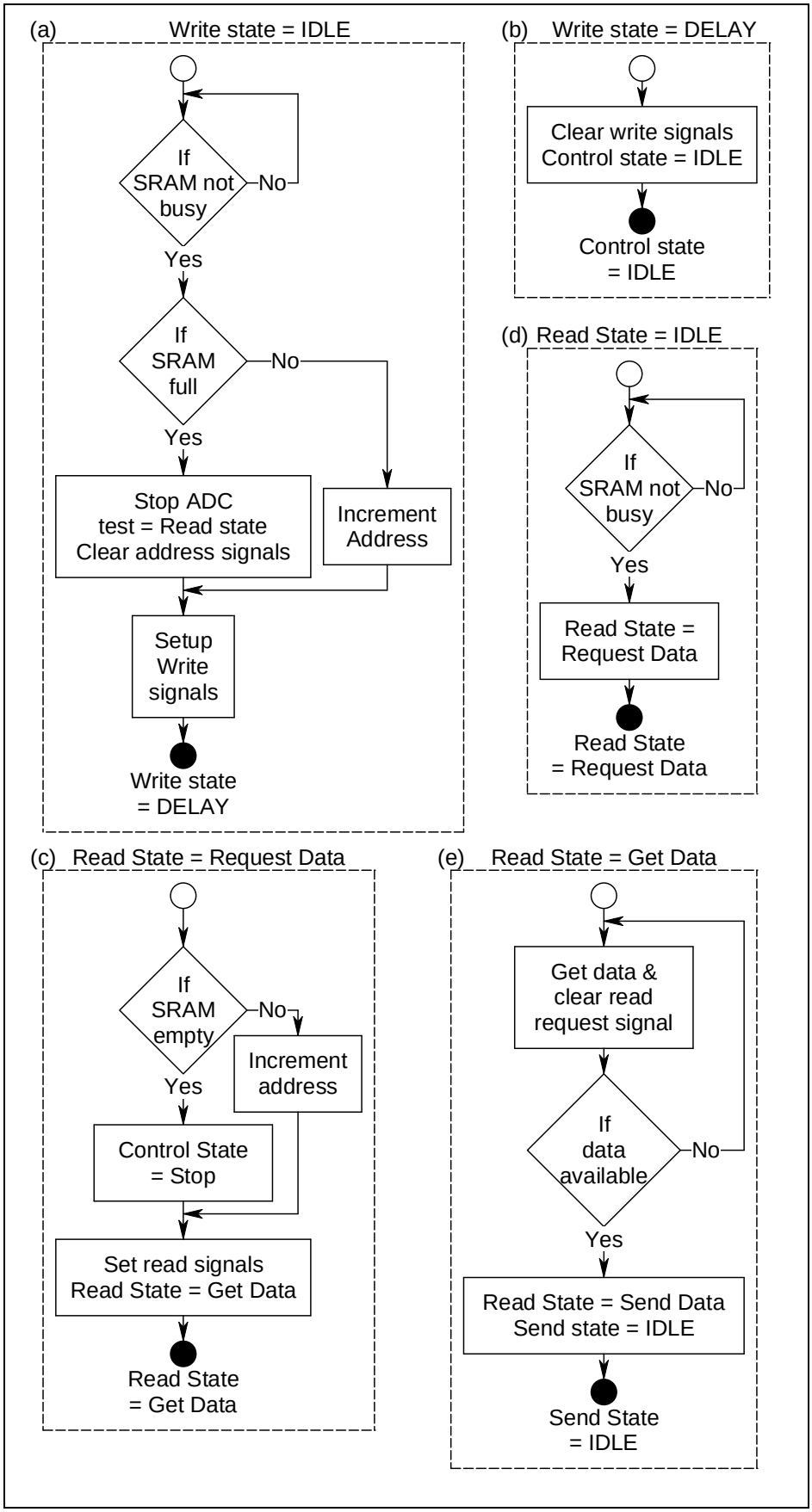


Figure E.8 (a) Write IDLE state, (b) Write DELAY state, (c) Read Request data, (d) Read IDLE state and (e) Read Get Data state.

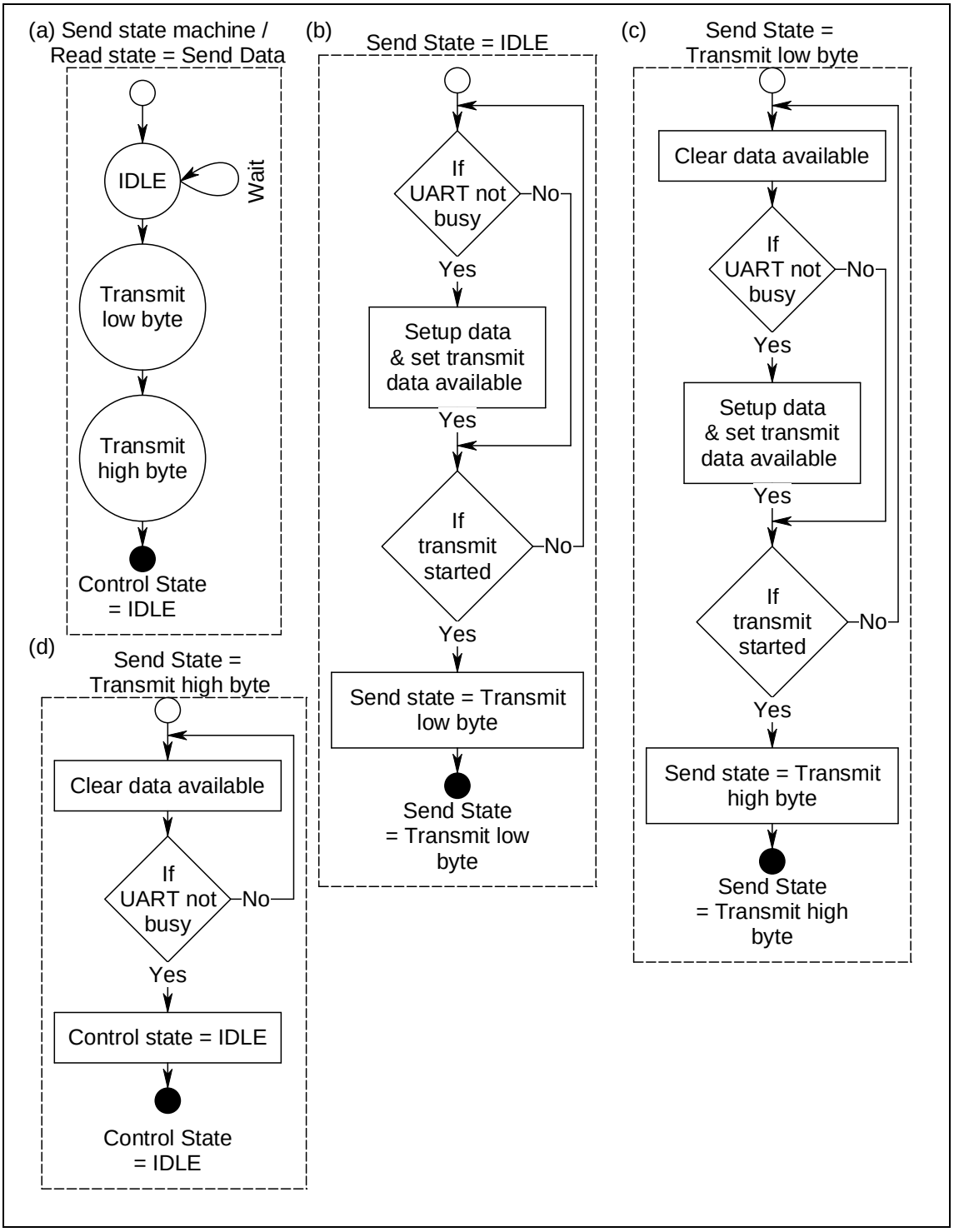


Figure E.9 (a) Read Send Data state or Send state machine, (b) Send IDLE state, (c) Send Transmit low byte state and (d) Send Transmit high byte state.

E.4. Software

The software refer to in this section is the PC GUI which receives the data stored in the SRAM via USB/UART bridge and stores it to be used by the simulation software. Figure E.10 illustrates the basic flow of the GUI.

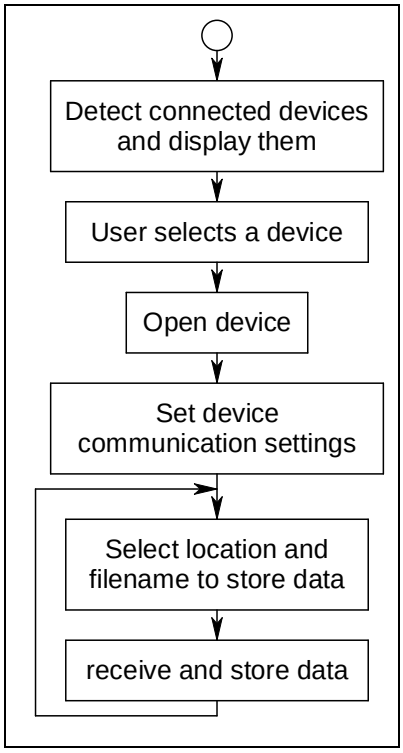


Figure E.10 Basic flow of program

Figure E.11 is a screenshot of the GUI. The program is based on the functionalities provided by the DLL (ftd2xx.dll) for the USB/UART bridge. The DLL is provided by FTDI™ as royalty free software.

The device display area only displays the USB device supported by the DLL and accompanied driver. The user can select the appropriate device, for the add-on board. Currently the device displayed is not connected to the USB connection event. Thus the list is not updated automatically when a USB device is connected or disconnect. The program however provides the user with the functionality to update the list.

When the appropriate devices is selected the open button will be active, and connection will be established. Once a device is open the user has the functionality to close it and the “Port Settings” will be enabled.

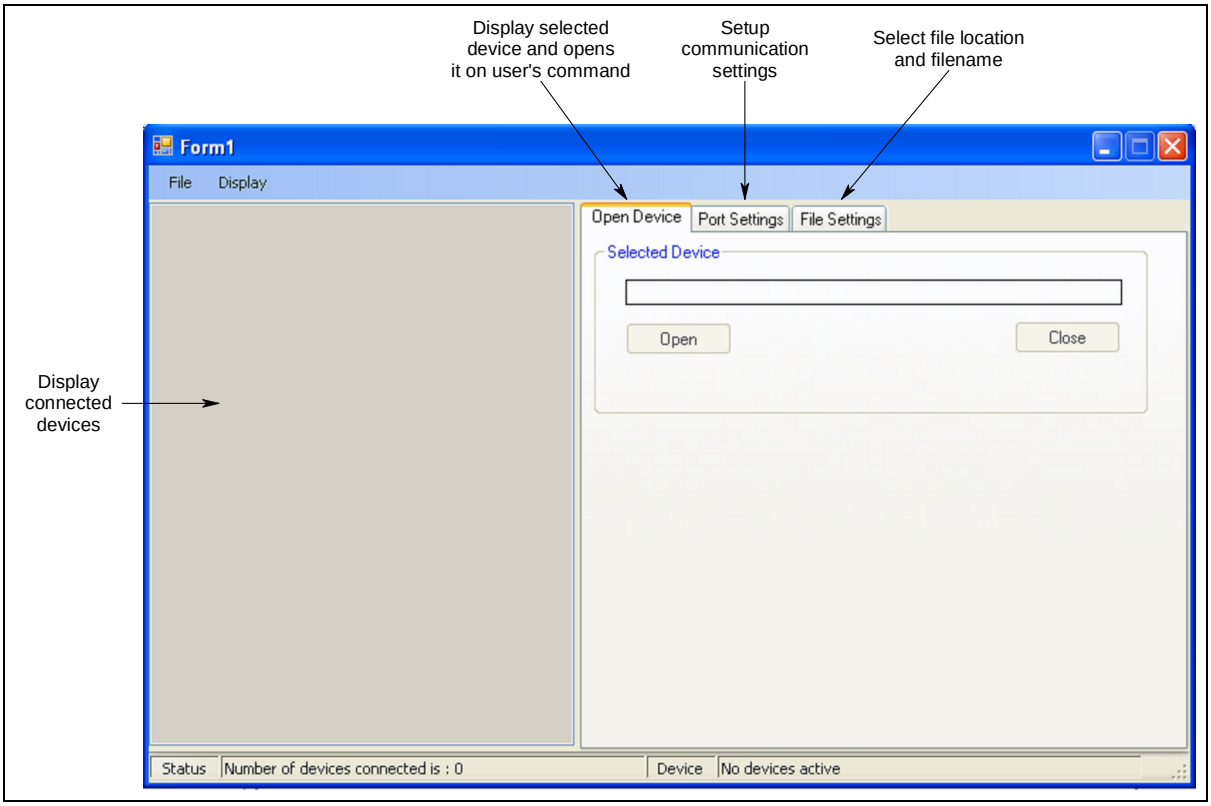


Figure E.11 User interface to the add-on board

The “Port Settings” consist of the basic UART settings, namely

- Baud rate (Chip provide the user with the ability to communicate at non-standard baud rates, see datasheet.).
- Number of bits
- Stop bits.
- Parity

Once the “Port Settings” is specified the “File Settings” tab will be activated. This allows the user to select the destination the file must be saved to and the file name. Because the simulation program uses csv or txt files, these files default to csv file format.

If the user wants to save a new file, only the “File Settings” have to be changed, otherwise the previous file will be overwritten (the program asks the user before a file is overwritten).

Appendix F.

F.1. Faraday's law

An electric field is generated by a changing magnetic field. Faraday's law describes the characteristics of magnetic fields. The effect of the electric field generated by Faraday's law depends on the material properties of the surrounding area [2]

- *Vacuum*: the electric field strength E gives rise to an electromagnetic rotational field. Periodic changes in magnetic flux (high frequency current in an antenna coil) generate an electromagnetic field that propagates itself into the distance. [2]

- *Open conductor loop*: an open circuit voltage builds up across the ends of an almost closed loop, which is called an induced voltage. The voltage corresponds with the line integral of the field strength E that is generated along the path of the conductor loop in space. [2]

- *Metal surface*: an electric field strength E is also induced in the metal surface. The electric field cause free charge carries to flow in the direction of the electric field strength. Eddy currents (currents flowing in circles) are created. Eddy currents works against the exciting magnetic flux (Lenz's law), which may reduce the magnetic flux in the area of the metal surface. The effect describe is undesirable in inductive coupled RFID system.[2]

Figure F.1 depicts Faraday's law in the different materials mentioned.

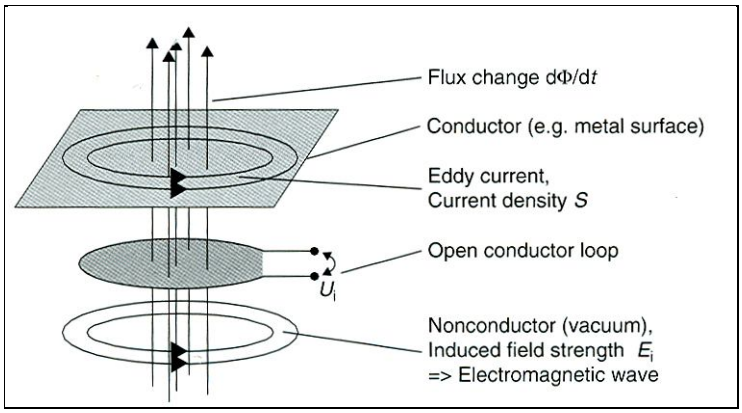


Figure F.1 Induced electric field strength E in different materials. From top to bottom: metal surface, conductor loop and vacuum [2]

Formula for Faraday's law: $= \oint \quad = -\frac{\Psi(\ )}{}$ [2][4]

Formula for conductor loops with N windings: $= -\frac{\Psi(\ )}{}$ [2]

RFID inductive system operates on Faraday's law. Figure F.2 depicts the basic mechanism of a RFID inductive system. In figure F.2, L_1 would represent the reader in the RFID system. The conductor loop L_2 represents the tag antenna. The resistance R_2 is the tag's antenna's resistance and R_L represents the current consumption of the tag's internal circuit. [2]

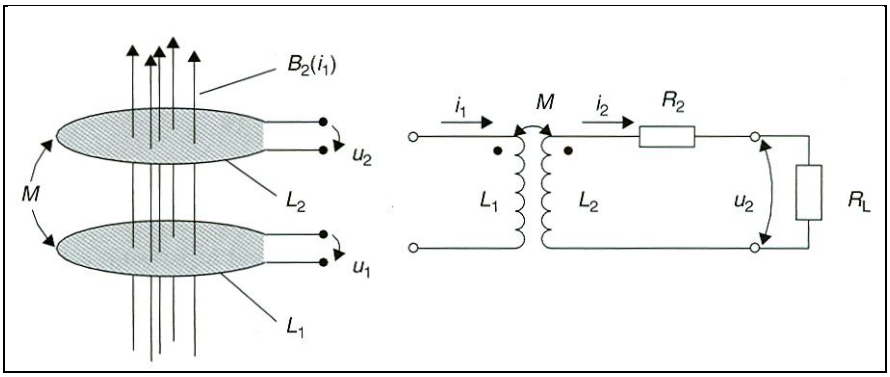


Figure F.2 Left, magnetically coupled conductor loops; right, equivalent circuit diagram for magnetically coupled conductor loops [2]

A time variant current $i_1(t)$ in the conduction loop L_1 generates a time variant magnetic flux $\frac{d\phi(i_1)}{dt}$. As Faraday's law states, a voltage is induced in the conduction loop L_1 and L_2 when magnetic flux flows through them. Two types of inductance can be identified namely: [2]

- Self-inductance: the flux change generated by the current change $\frac{di_n}{dt}$ induce a voltage u_n in the same conductor circuit. [2]
- Mutual inductance: the flux change generated by the current change $\frac{di_n}{dt}$ induces a voltage in the adjacent conductor circuit L_m. Both circuits are coupled by mutual inductance. [2]

Appendix G.

G.1. Introduction

As previously discussed in Chapter 2 of this dissertation dealing with the history of the greater project, the reader should keep in mind that the research continued during this phase (phase VI) of the project. This phase is a continuation of previous research conducted during phases I through V. The essence of the previous five phases will be discussed in this appendix.

G.2. Phase I

Currently iPico, the company in conjunction with whom this research is being conducted are using analogue signal processing techniques in their RFID readers. But analogue signal processing techniques have certain constraints. Digital signal processing provides certain advantages which cannot be achieved by analogue signal processing techniques. The first objective of phase I of the research was to conduct a feasibility study in order to see whether digital signal processing techniques can provide any advantages over the now older analogue processing techniques. The simulations for this feasibility study were conducted in Matlab®. The second objective was to determine whether the digital techniques identified in objective one can be implemented on available hardware technology. [23]

G.2.1. Tests using high frequency tags

The first tags used to generate an input for the simulation during this feasibility study were high frequency tags of iPico. These tags contain a 64-bit identification number (ID). The signal received by the reader was a signal with the following characteristics; 50 µV pulses superimposed on a 0.7 V, 1 kHz carrier signal. The assumption was made that the tags transmitted their IDs at 256 kbits/s and that these IDs were Glitch encoded (see Chapter 3). [23]

In order to test the newly studied digital techniques an algorithm was developed. Due to the specification of the input signal, especially the 50 µV-peak pulses which contained the data, it was decided to make use of a 16 bit ADC. This provided a

lowest bit resolution of 10.7 µV opposed to a 14 bit ADC which would have provided a lowest bit resolution of 42.7 µV. [23]

The researcher used a 100 kHz window sync high pass filter to rid the incoming signal of the carrier. Here after the output of the filter was used as the input to a Matched filter. Bit information of the 50 µV pulses were obtained by inserting a theoretical copy of the 50 µV pulses, which was not superimposed on the carrier signal, into the Matched. Using a copy of the 50 µV pulses, values were calculated by drawing a comparison to the input signal values. A result of the Matched filter's output was an improved S/N (signal to noise) ratio. [23]

Following the Matched filter, the output of the previous stage was tested against a "Bit threshold". A "Bit threshold" determines whether a signal value is low or high by comparing an input signal against a reference. The "Bit threshold's" ones and zeros only determined whether an ID was present in the original signal or not. It did not supply any information with regard to the data in the original signal sample. [23]

The output generated by the "Bit threshold" was stored in an array for one complete bit period, because bits were glitch encoded. After the bit period the output values were compared to determine whether a logic zero or logic one was present in the previous bit period. [23]

Next the effectiveness of the algorithm was analysed. During the first test, five IDs were generated as an input signal. In the second test 10 IDs were used. In both these tests 100 % detection rates were obtained. [23]

To conduct a more realistic test, a statistical examination was done (third test). During this test, ten different input signals with 1000 IDs in each were generated. For each input signal the IDs were different and generated randomly. Table G.1 on the next page shows the results obtained. [23]

Table G.1 Phase I, high frequency tags, third test results [23]

Signal	Detection rate
1	98 %
2	97 %
3	98 %
4	98 %
5	97 %
6	97 %
7	97 %
8	98 %
9	98 %
10	98 %

The first three tests were conducted under ideal circumstances. (No Guassian noise in the sample environment.) During the fourth test, the same ten input signals used in the third test were used as sample data. However, white Guassian noise was added to these signals. Table G.2 is a summary of the results obtained during this test. [23]

Table G.2 Phase I, high frequency tags, fourth test results [23]

S/N / Signal	Inf	15 dB	10 dB	8 dB	6 dB	4 dB
1	98 %	96 %	92 %	89 %	84 %	74 %
2	97 %	95 %	95 %	93 %	87 %	76 %
3	98 %	96 %	91 %	88 %	83 %	74 %
4	98 %	98 %	96 %	93 %	86 %	75 %
5	97 %	96 %	94 %	91 %	86 %	74 %
6	97 %	97 %	96 %	93 %	89 %	79 %
7	97 %	95 %	90 %	85 %	77 %	66 %
8	98 %	98 %	95 %	93 %	88 %	77 %
9	98 %	95 %	90 %	85 %	78 %	65 %
10	98 %	97 %	94 %	91 %	87 %	75 %

The researcher concluded from the fourth test that the algorithm (Algorithm I) performed quite well, even for environments where the S/N was as low as 4 dB.

G.2.2. Tests using low frequency tags

iPico's low frequency tags contain 64-bit identification numbers. From the specifications of these tags it is known that the frequency of the sync pulses (pulses used by the reader to detect the presence of an ID) is 7 MHz. It was assumed that the IDs were transmitted at 256 kbits/s and that Glitch encoding was used. (Refer back to the first section in appendix G) [23]

Due to the frequency of the sync pulses as well as to allow for oversampling, a sample rate of 21 MHz was decided on. It was decided to use an automatic gain control to regulate the incoming pulses, resulting in 500 mV pulses being used as input to the ADC. Given the sheer differential voltage of these pulses it was decided to use an 8-bit ADC, resulting in a lowest bit resolution of 3.91mV. [23]

The previous four test methodologies used for the high frequency tags were also used for the low frequency tags, but with the new tag specifications for the input signal. During the third test, five IDs were used instead of the ten used during the previous third test conducted with the high frequency tags. The results of the first three tests were clear, a 100 % detection rate in all three cases. [23]

Conducting the fourth test, five IDs were used instead of the ten used when conducting the test with the high frequency tags. Table G.3 depicts the results of the fourth test. [23]

Table G.3 Phase I, low frequency tags, fourth test results [23]

S/N / Signal	Inf	15 dB	10 dB	8 dB	6 dB	4 dB
1	100 %	99 %	97 %	92 %	80 %	61 %
2	100 %	99 %	97 %	92 %	80 %	61 %
3	100 %	99 %	97 %	92 %	80 %	61 %
4	100 %	99 %	97 %	92 %	80 %	61 %
5	100 %	99 %	97 %	92 %	80 %	61 %

The researcher reached the same conclusion for the fourth test, as concluded for the high frequency tags fourth test.

G.2.3. Recommendations

Based on the results obtained by the simulation it was deemed possible to implement a digital RFID reader. Two methods of implementing a digital algorithm were identified: [23]

- Making use of a digital signal processor or,
- Implementing the algorithm on a FPGA.

Analog Devices and Texas Instruments are two of the most popular branches in digital signal processing playing field. During this phase of the research a choice as to which processing unit were going to be used had to be made. The author states that it was decided to concentrate on the Texas Instrument processor, because the processor was known to him. At that time the best digital signal processor from Texas instruments was the TMS320C6000 series, with both fixed and floating point processors available. Due to the fact that an FIR (finite impulse response) filter was used in the algorithm, it was decided to use the floating point processor. Two processors were recommended, the TMS320C6711C-200 or TMS320C6711B-100. [23]

However, the digital detection algorithm could also have been implemented on a FPGA. An advantage of a FPGA would have been that the FIR (finite impulse response) filter could have been implemented to execute the FIR filter in parallel. [23]

G.2.4. Conclusion

Theoretically it was possible to implement a digital RFID reader, as could be seen from the simulation results. The author stated that although ideas usually seem possible for implementation theoretically, unforeseen issues quite often place constraints upon the practical solutions. It was suggested that in order to evaluate the simulation results obtained during the phase of the research, one single digital RFID reader should be constructed and implemented. [23]

It was also stated that the following issues should have received attention before implementing a digital RFID reader practically: [23]

- The digital detection algorithm (Algorithm I) functioned fairly well, but it should have been optimized to obtained the best results.
- It should be decided whether to use a digital signal processor or a FPGA during implementation.
- A decision should be made about the sampling rate and the most appropriate ADC for the application should be selected from different available supplier companies which develop ADCs.

G.3. Phase II

The objective of phase II was to implement the algorithm, as derived in Phase I, on some sort of digital processor in order to demonstrate the correct functioning of a digital RFID reader. During the first attempt it was decided to implement the algorithm on a digital signal processor. From the then top of the range Texas Instrument's floating point processors, the TMS320C6713 processor was chosen. [24]

The developed algorithm was ported to the TMS320C6713 digital signal processor system. After implementation, the first test made use of data similar to the previous simulation data as input. The correct IDs were obtained as output from the digital signal processor. [24]

To conduct the second test it was planned to make use of real data, however interface problems between the TMS and ADC occurred. Instead data was sampled by a digital oscilloscope and used as input to the system. The system was able to correctly identify all IDs. [24]

A second problem occurred with the implementation of the digital RFID reader. It was impossible to implement the two FIR filters as well as the Match filter on the same hardware. The execution time for the FIR and Matched filters overstepped the time constraints placed on the system (200 ns) by a factor 50 to 100 times. [24]

Two alternative solutions were explored. As a first solution the suppliers of FPGAs were contacted. The suppliers were confident that the two FIR filters could be

implemented on a FPGA and executed in less than 200 ns. The reason was that the filters could be executed in parallel. A literature study indicated that the FIR filter should be implemented by a bit serial technique. Due to a lack of manpower and time this solution was not explored further in Phase II. [24]

Solution two focused on the development of a new detection algorithm since the first algorithm spent most of the available execution time on the execution of the FIR and Matched filters. The new algorithm called “Algorithm II” did not make use of FIR and Matched filters at all. [24]

The fundamental principles of “Algorithm II” were based on the following. The latter algorithm, “Algorithm II” firstly detected whether an ID was present in the incoming signal or not. When the presence of an ID was detected, energy in the first quarter of the bit was compared to energy in the third quarter of the bit. The energy difference between the two quarters determined whether a, logic one or zero was detected. (Definition of glitch encoding adhere to this principle of Algorithm II) [24]

The results of the latter test executed on “Algorithm II” were positive. “Algorithm II”, executed, in the region of 400 ns. Although still not the desired 200 ns it was a significant improvement over the previous execution time obtained making use of the TMS. The only other hardware change made using “Algorithm II” was the second order Butterworth high pass input filter. The latter was replaced by a fourth order Butterworth high pass filter. [24]

G.3.1. Recommendations

Studying the results obtained during Phase I and II, the following method depicted in Figure G.1 was proposed for final implementation of the digital detection algorithm. [24]

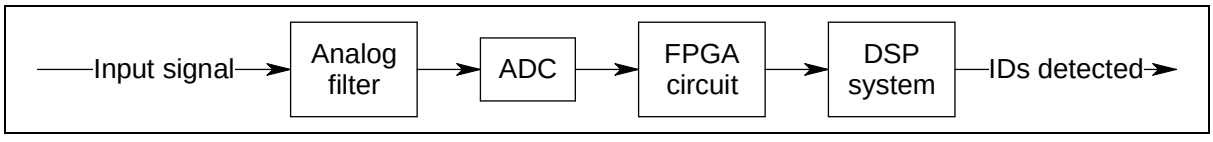


Figure G.1 Phase II, proposed implementation

For the sake of clarity the different blocks depicted in figure G.1 will now be discussed: [24]

- *Analogue filter:* This filter was used to separate the ID information from the carrier wave information locked up in the input signal. The carrier was suppressed. A 12-bit ADC could be used instead of a 16-bit ADC, when an analogue filter was implemented before the ADC. To minimize distortion on the input signal, the order of the filter have been made as low as possible. The researcher recommended a second order Butterworth high pass filter was implemented for use in “Algorithm I” and a fourth order Butterworth filter for use in “Algorithm II”.
- *ADC:* If this analogue filter was implemented a 12-bit ADC could be used. Refer to the previous paragraph.
- *FPGA circuit:* All the digital filters necessary for the digital RFID reader would be implemented on a FPGA. A FPGA could also house any other time consuming operations.
- *DSP system:* All additional digital signal processing could be implemented on this unit. A good example would be the CRC checking and communication with other critical system elements.

  Proposed *s*teps before implementation of the latter idea as depicted in figure G.1: [24]

- A decision had to be made between the implementation of “Algorithm I” or “Algorithm II”.
    - All digital filters should have been implemented on a FPGA. If “Algorithm I” would be implemented the FIR and Match filters should be implemented on a FPGA. When “Algorithm II” would be implemented a secondary FIR high pass filter could be implemented to reduce the specification of the analogue filter. However, additional topics should be investigated before implementation:

- A method of implementation for a FIR filter on a FPGA circuit. According to previous publications a bit serial approach should have been used.
- The development of an interface between the ADC and FPGA circuit.
- The development of an interface between the FPGA circuit and DSP system.
- Whether only the FIR and Matched filters should have been implemented on a FPGA circuit, or if the digital signal processing functions should also have been implemented on a FPGA circuit.

- The implementation of the algorithm on the digital signal processing system should have been optimized.
- Since many of the functionalities of different routines have been based on empirical knowledge regarding the system, theory regarding the functioning of the two algorithms should have been investigated.
- The digital system should have been tested in an environment where the analogue reader was exposed to interferences in order to give an indication whether the digital reader solved the analogue readers' problems.

G.3.2. Conclusion

The proposed implementation most probably would have lead to a feasible digital RFID reader. The investigation undertaken in this phase exposed certain issues which, if explored further, might have led to an improvement of the design. These issues were: [24]

- Would a Kalman filter not increase the effectiveness of the digital reader in a noisy environment? Kalman filters are known for their ability to function in a noisy environment.
- The use of neural networks may be explored in order to detect IDs.
- The use of Viterbi schemes may provide advantages in correcting incorrect detected bits. It should be noted however that, the tags currently used for the purpose of this research, do not support a Viterbi scheme.

G.4. Phase III

The problems identified in phase II inspired phase III. Phase III of the research mainly concentrated on investigating the developed algorithm in order to determine whether it could be simplified. If the algorithm could be simplified, hardware requirements could be reduced, which would lead to a less expensive implementation. [25]

For reference a quick overview of the digital RFID reader will be given.

“Algorithm I”, made use of a second order Butterworth filter to suppress the carrier wave. A 12-bit ADC was used for the latter conversion. In addition a 2 x 200 point FIR filter was used to completely remove the carrier wave from the input signal. The range of the input pulses in this analysis was 50 μV peak to peak. [25]

The following aspects were explored in Phase III, to see how these aspects affected the detection capabilities of “Algorithm I”. [25]

1. The order of the analogue Butterworth filter.
2. The voltage range of the input signal.
3. The bit-width of the ADC.
4. The number of points to be used in the FIR filter given the use of an 8-bit wide AD converter.
5. The number of points to be used in the FIR filter given the use of a 12-bit wide AD converter.
6. Re-evaluations of the order of the analogue Butterworth filter.
7. The voltage range of the pulses superimposed on the carrier wave.

G.4.1. Aspect 1: the order of the analogue Butterworth Filter

The main aspect concentrated upon during this review was the detection capability of “Algorithm I” using a second order Butterworth filter compared to a fourth order Butterworth filter. By increasing the order of the analogue filter, it might have been possible to decrease the order of the FIR filter. [25]

A first test conducted made use of a second order analogue Butterworth filter, 12-bit ADC, 2 x 200 point FIR filters and a ± 10 mV signal range. The simulation software created during Phase I was used with the five different input signals, each containing a 1000 IDs using different noise conditions. Table G.4 displays the detection rates of test 1. [25]

Table G.4 Phase III, Aspect 1's detection rates of test 1 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	98 %	92 %
2	100 %	99 %	99 %	99 %	98 %	92 %
3	100 %	99 %	99 %	99 %	98 %	92 %
4	100 %	99 %	99 %	99 %	98 %	92 %
5	100 %	99 %	99 %	99 %	98 %	92 %

The second test made use of a fourth order analogue Butterworth filter, 12-bit ADC, 2 x 200 point FIR filters and a ± 10 mV signal range. Table G.5 contains the detection rates obtained from test 2.

Table G.5 Phase III, Aspect 1's detection rates of test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	98 %	90 %
2	100 %	99 %	99 %	99 %	98 %	90 %
3	100 %	99 %	99 %	99 %	98 %	90 %
4	100 %	99 %	99 %	99 %	98 %	91 %
5	100 %	99 %	99 %	99 %	98 %	90 %

From the test results a conclusion was derived that by increasing the analogue Butterworth filter to a fourth order filter, had no significant implication on the detection capabilities of “Algorithm I”. [25]

G.4.2. Aspect 2: the voltage range of the input signal

When a fourth order, analogue Butterworth filter was used, the input signal (carrier wave) at the ADC displayed an amplitude of 110 µV, although the previous test

accepted an amplitude of 10 mV. The second analysis explored the effect of a ± 1 mV input signal at the 12-bit ADC. [25]

The first test concerning the second aspect was the same test as test 2 of aspect 1. Refer to the previous aspects, test 2 results. In the second test a fourth order analogue filter, 12-bit ADC, 2 x 200 point FIR filters as well as a ± 1 mV signal range was used. Table G.6 provides the detection rates for test 2. [25]

Table G.6 Phase III, Aspect 2's detection rates for test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	98 %	92 %
2	100 %	99 %	99 %	99 %	98 %	92 %
3	100 %	99 %	99 %	99 %	98 %	92 %
4	100 %	99 %	99 %	99 %	98 %	90 %
5	100 %	99 %	99 %	99 %	98 %	90 %

In conclusion, the input signal to the ADC with a smaller range, had little effect on the detection rate of “Algorithm I”.

G.4.3. Aspect 3: the bit-width of the ADC

The effect on the detection capabilities of “Algorithm I” utilizing a 12-bit ADC compared to an 8-bit ADC will be tested. If a lower bit resolution could be used, it would lower the cost of implementing ”Algorithm I”. [25]

Aspect 3’s first test and aspect 2’s second test had the same test parameters. Refer to aspect 2 for the results. In the second test, a fourth order analogue Butterworth filter, an 8-bit ADC, 2 x 200 point FIR filters and a ± 1 mV signal range was used as input. Table G.7 summarise the detection rates obtained for test 2 [25]

Table G.7 Phase III, Aspect 3's detection rates of test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	97 %	87 %
2	100 %	99 %	99 %	99 %	97 %	87 %
3	100 %	99 %	99 %	99 %	97 %	87 %
4	100 %	99 %	99 %	99 %	96 %	86 %
5	100 %	99 %	99 %	99 %	97 %	87 %

The conclusion was made that by implementing an 8-bit ADC, the detection capabilities of “Algorithm I” were reduced by a very small margin.

G.4.4. Aspect 4: the number of points to be used in the FIR filter given the use of an 8-bit wide AD converter

Aspect 4 explores the effect the length of the FIR filter had on the detection capabilities of “Algorithm I”. In this case a fourth order analogue Butterworth filter with an 8-bit ADC was implemented. [25]

The first test was the same as the second test performed under aspect 3. Refer to aspect 3 for the results. In the second test a fourth order analogue Butterworth filter, 8-bit ADC, 2 x 100 point FIR filters and a ± 1 mV signal range was implemented. Table G.8 indicates the detection rates of test 2

Table G.8 Phase III, Aspect 4's detection rates for test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	96 %	82 %
2	100 %	99 %	99 %	99 %	96 %	79 %
3	100 %	99 %	99 %	99 %	97 %	82 %
4	100 %	99 %	99 %	99 %	96 %	80 %
5	100 %	99 %	99 %	99 %	96 %	80 %

For the purpose of the third test a fourth order analogue Butterworth filter, 8-bit ADC, 2 x 40 point FIR filters and a ± 1 mV signal range was used. Table G.9 shows the detection rates of test 3. [25]

Table G.9 Phase III, Aspect 4's detection rates for test 3 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	96 %	82 %
2	100 %	99 %	99 %	99 %	96 %	79 %
3	100 %	99 %	99 %	99 %	97 %	82 %
4	100 %	99 %	99 %	99 %	96 %	80 %
5	100 %	99 %	99 %	99 %	96 %	80 %

For the purpose of the fourth test a fourth order analogue Butterworth filter, 8-bit ADC, 2 x 30 point FIR filters and a ± 1 mV signal range was used. Table G.10 provides the detection rates obtained for test 4. [25]

Table G.10 Phase III, Aspect 4's detection rates for test 4 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	97 %	86 %	58 %	37 %
2	100 %	99 %	97 %	85 %	57 %	40 %
3	100 %	99 %	97 %	85 %	59 %	39 %
4	100 %	99 %	98 %	86 %	59 %	40 %
5	100 %	99 %	98 %	88 %	59 %	40 %

In the fifth test a fourth order analogue Butterworth filter, 8-bit ADC, 2 x 20 point FIR filters and a ± 1 mV signal range were implemented. In this case it was impossible to detect any correct IDs. [25]

In conclusion the following issues must be taken in account when deciding on the length of FIR filter which would have been used. [25]

- The length of the FIR filter was depended on the S/N ratio of the environment wherein “Algorithm I” would be implemented. For example, when a S/N ratio of 4 to 6 dB was present a higher order FIR filter should have been implemented than when a 10 to 15 dB S/N ratio was present.

- The minimum length of a FIR filter for “Algorithm I” was 30 points. No IDs were detected using FIR filters with less than 30 points.

G.4.5. Aspect 5: The number of points to be used in the FIR filter given the use of a 12-bit wide AD converter.

In order to confirm the results obtained in aspect 4, the test concerning the number of points necessary for the FIR filter was repeated with a 12-bit ADC instead of an 8-bit ADC. [25]

Aspect 5’s first test conducted the same test as aspect 3’s first test. See aspect 3’s test 1 for the mentioned test results. In the second test a fourth order analogue Butterworth filter, 12-bit ADC, 2 x 100 point FIR filters and a ± 1 mV signal range was used. Table G.11 shows the detection rates of test 2. [25]

Table G.11 Phase III, Aspect 5's detection rates for test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	98 %	86 %
2	100 %	99 %	99 %	99 %	98 %	84 %
3	100 %	99 %	99 %	99 %	98 %	87 %
4	100 %	99 %	99 %	99 %	97 %	85 %
5	100 %	99 %	99 %	99 %	97 %	86 %

In the third test a fourth order analogue Butterworth filter, 12-bit ADC, 2 x 40 point FIR filters and a ± 1 mV signal range was implemented. Table G.12 summarises the detection rates of test 3. [25]

Table G.12 Phase III, Aspect 5's detection rates for test 3 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	92 %	75 %
2	100 %	99 %	99 %	99 %	91 %	74 %
3	100 %	99 %	99 %	99 %	91 %	74 %
4	100 %	99 %	99 %	99 %	91 %	73 %
5	100 %	99 %	99 %	99 %	91 %	73 %

In the fourth test a fourth order analogue Butterworth filter, 12-bit ADC, 2 x 30 point FIR filters and a ± 1 mV signal range was used. Table G.13 indicates the detection rates obtained in test 4. [25]

Table G.13 Phase III, Aspect 5's detection rates for test 4 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	92 %	75 %
2	100 %	99 %	99 %	99 %	91 %	74 %
3	100 %	99 %	99 %	99 %	91 %	74 %
4	100 %	99 %	99 %	99 %	91 %	73 %
5	100 %	99 %	99 %	99 %	91 %	73 %

In the fifth test, a fourth order analogue Butterworth filter, 12-bit ADC, 2 x 20 point FIR filters and a ± 1 mV signal range was used. No correct IDs were detected. [25]

From the mentioned test results it could be concluded that the degradation in detection capabilities of “Algorithm I” was similar to the results obtained in the 8-bit ADC tests.

G.4.6. Aspect 6: Re-evaluations of the order of the analogue Butterworth filter

Aspect 6 investigated the effect, an increase in the order of the fourth order analogue Butterworth filter would have on the detection capabilities of “Algorithm I”. [25]

Test 1 was the same as test 4 used in aspect 4. Refer to aspect 4, test 4 for the results. In the second test conducted a fifth order analogue Butterworth filter, 8-bit ADC, 2 x 30 point FIR filters and a ± 1 mV signal range was used. Table G.14 summarises the detection rates obtained in test 2. [25]

Table G.14 Phase III, Aspect 6's detection rates for test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	98 %	88 %	62 %	42 %
2	100 %	99 %	98 %	88 %	62 %	41 %
3	100 %	99 %	98 %	88 %	64 %	42 %
4	100 %	99 %	98 %	88 %	63 %	42 %
5	100 %	99 %	99 %	88 %	62 %	42 %

The conclusion reached: the results indicated a small improvement in the detection capabilities of “Algorithm I” for a S/N in excess of 4 dB; it was therefore not recommended to increase the order of the analogue filter. [25]

G.4.7. Aspect 7: The voltage range of the pulses superimposed on the carrier wave

This last topic explored the effect the range of the input pulses had on the detection capabilities of “Algorithm I”. Tests were conducted in two phases. The first phase implemented a 2 x 30 point FIR filters and the second a 2 x 200 point FIR filters. In both cases it was assumed that the ADC could handle a ± 1 mV input signal and limited to this range. Also a fourth order analogue Butterworth filter and an 8-bit ADC were used. [25]

Phase 1 test 1, 50 µV input pulse ranges and 2 x 30 point FIR filters were used. Table G.15 displays the detection rates for this first test. [25]

Table G.15 Phase III, Aspect 7's phase 1 detection rates for test 1 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	97 %	86 %	58 %	37 %
2	100 %	99 %	97 %	85 %	57 %	40 %
3	100 %	99 %	97 %	85 %	59 %	39 %
4	100 %	99 %	98 %	86 %	59 %	40 %
5	100 %	99 %	98 %	84 %	59 %	40 %

Phase 1 test 2, 500 µV input pulse ranges and 2 x 30 point FIR filters were used. Table G.16 summarises the detection rates for test 2. [25]

Table G.16 Phase III, Aspect 7's phase 1 detection rates for test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	95 %	68 %	44 %
2	100 %	99 %	99 %	95 %	68 %	47 %
3	100 %	99 %	99 %	95 %	68 %	46 %
4	100 %	99 %	99 %	95 %	67 %	44 %
5	100 %	99 %	99 %	94 %	68 %	45 %

Phase 1, third test, 5 mV input pulse ranges and 2 x 30 point FIR filters were used. Table G.17 contains the detection rates for test 3. [25]

Table G.17 Phase III, Aspect 7's phase 1 detection rates for test 3 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	78 %	49 %	32 %
2	100 %	99 %	98 %	77 %	46 %	31 %
3	100 %	99 %	98 %	77 %	47 %	32 %
4	100 %	99 %	98 %	76 %	48 %	32 %
5	100 %	99 %	98 %	76 %	46 %	32 %

Phase 1 test four, 50 mV input pulse ranges and 2 x 30 point FIR filters. Table G.18 summarises the detection rates for test 4. [25]

Table G.18 Phase III, Aspect 7's phase 1 detection rates for test 4 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	90 %	61 %	37 %
2	100 %	99 %	99 %	90 %	59 %	41 %
3	100 %	99 %	99 %	89 %	59 %	41 %
4	100 %	99 %	99 %	90 %	58 %	39 %
5	100 %	99 %	99 %	90 %	59 %	40 %

Phase 2, test 1, in this case 50 µV input pulse ranges and 2 x 200 point FIR filters were used. Table G.19 provides the detection rates obtained for test 1. [25]

Table G.19 Phase III, Aspect 7's phase 2 detection rates for test 1 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	97 %	87 %
2	100 %	99 %	99 %	99 %	97 %	87 %
3	100 %	99 %	99 %	99 %	97 %	87 %
4	100 %	99 %	99 %	99 %	96 %	86 %
5	100 %	99 %	99 %	99 %	97 %	87 %

Phase 2, test two, 500 µV input pulse ranges and 2 x 200 point FIR filters were used. Table G.20 displays the detection rates for test 2. [25]

Table G.20 Phase III, Aspect 7's phase 2 detection rates for test 2 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	98 %	92 %
2	100 %	99 %	99 %	99 %	98 %	91 %
3	100 %	99 %	99 %	99 %	98 %	91 %
4	100 %	99 %	99 %	99 %	98 %	90 %
5	100 %	99 %	99 %	99 %	98 %	91 %

Phase 2, test three, 5 mV input pulse ranges and 2 x 200 point FIR filters were used. Table G.21 summarises the detection rates for test 3. [25]

Table G.21 Phase III, Aspect 7's phase 2 detection rates for test 3 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	99 %	90 %
2	100 %	99 %	99 %	99 %	98 %	88 %
3	100 %	99 %	99 %	99 %	98 %	90 %
4	100 %	99 %	99 %	99 %	98 %	89 %
5	100 %	99 %	99 %	99 %	98 %	89 %

Phase 2, fourth test, 50 mV input pulse ranges and 2 x 200 point FIR filters were used. Table G.22 indicates the detection rates for test 4.

Table G.22 Phase III, Aspect 7's phase 2 detection rates for test 4 [25]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	99 %	98 %	93 %
2	100 %	99 %	99 %	99 %	98 %	92 %
3	100 %	99 %	99 %	99 %	98 %	93 %
4	100 %	99 %	99 %	99 %	99 %	93 %
5	100 %	99 %	99 %	99 %	99 %	93 %

The conclusion reached: the results indicated that, by increasing the input pulse range to the limits of the ADC did not decrease the detection capabilities of “Algorithm I” [25].

G.4.8. Conclusion

From afore mentioned topics investigated and discussed in the previous few paragraphs, the following conclusions had been drawn: [25]

- For the correct functioning of “Algorithm I” the following minimum hardware requirements had been imposed: a fourth order analogue Butterworth filter, an 8-bit ADC and 2 x 30 point FIR filters.
- The bit resolution of the ADC was depended on the input signal range. An automatic gain control might have been implemented in order to limit the input signal ranges.
- The FIR filter length was depended on the S/N ratio requirements imposed upon the system.

G.5. Phase IV

The main objective of Phase IV was the comparison of the detection capabilities of “Algorithm I” with those of “Algorithm II” under similar S/N ratio conditions. Both algorithms made use of a fourth order analogue Butterworth filter (to suppress the carrier wave), 8-bit ADC, input signal ranges of ± 1 mV and 50 µV input pulse range. [26]

The input signal for the tests conducted during Phase IV differs from the previous Phases. In the first three Phases noise were added to the input signal *after* the analogue filter and the first FIR filter manipulated the input signal, thus after the carrier wave was removed from the input signal. Under Phase IV, the noise was added to the input signal before manipulation of the signal by the filters. [26]

The S/N ratio was calculated as follows. First the energy in the pulse information of the input signal was calculated, followed by the generation of a zero mean, unity variance Gauss noise signal. The Gaussian noise was then scaled by a constant factor and added to the input signal in order to obtain the wanted S/N ratio. [26]

The following topics were explored during the investigation conducted under Phase IV: [26]

1. “Algorithm I”.
2. “Algorithm II”.
3. “Algorithm II” with a high pass and low pass filter.
4. “Algorithm II” with a high pass filter, low pass filter and a 12-bit ADC.

G.5.1. “Algorithm I”

In all of the tests conducted under topic 1 a fourth order analogue Butterworth filter, an 8-bit ADC and a ± 1 mV input signal range was used. The variables in the tests were the lengths of the FIR and Match filters and the S/N ratios. [26]

Test 1 used 2 x 200 FIR filters. Table G.23 summarises the detection rates obtained in test 1. [26]

Table G.23 Phase IV, Topic 1's detection rates for test 1 [26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	100 %	100 %	97 %	61 %	26 %
2	100 %	100 %	100 %	98 %	62 %	27 %
3	100 %	100 %	100 %	97 %	61 %	26 %
4	100 %	100 %	100 %	97 %	64 %	26 %
5	100 %	100 %	100 %	98 %	62 %	24 %

Test 2 used 2 x 100 FIR filters. Table G.24 shows the detection rates obtained in test 2. [26]

Table G.24 IV, Topic 1's detection rates for test 2 [26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	97 %	56 %	24 %
2	100 %	99 %	99 %	97 %	56 %	25 %
3	100 %	99 %	99 %	97 %	57 %	23 %
4	100 %	99 %	99 %	97 %	58 %	23 %
5	100 %	99 %	99 %	97 %	57 %	23 %

Test 3 implemented 2 x 40 FIR filters. Table G.25 summarises the detection rates obtained in test 3. [26]

Table G.25 Phase IV, Topic 1's detection rates for test 3 [26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	92 %	47 %	16 %
2	100 %	99 %	99 %	92 %	45 %	16 %
3	100 %	99 %	99 %	92 %	44 %	15 %
4	100 %	99 %	99 %	92 %	46 %	17 %
5	100 %	99 %	99 %	93 %	43 %	15 %

Test 4 implemented 2 x 30 FIR filters. Table G.26 provides the detection rates obtained in test 4. [26]

Table G.26 Phase IV, Topic 1's detection rates for test 4 [26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	100 %	99 %	99 %	92 %	47 %	16 %
2	100 %	99 %	99 %	92 %	45 %	16 %
3	100 %	99 %	99 %	92 %	44 %	15 %
4	100 %	99 %	99 %	92 %	46 %	17 %
5	100 %	99 %	99 %	93 %	43 %	15 %

G.5.2. Algorithm II

"Algorithm II" only needed one test to measure its performance against phase IV's criteria. The latter is simpler because the algorithm does not contain any FIR or Match filters. In this test a fourth order analogue Butterworth filter, an 8-bit ADC and a ± 1 mV input signal range was implemented. Table G.27 is a summary of the detection rates obtained in the latter test. [26]

Table G.27 Phase IV, Topic 2's detection rates for the test [26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	99.9 %	95.7 %	96.1 %	95.9 %	93.2 %	90.6 %
2	99.9 %	95.7 %	96.1 %	95.9 %	93.2 %	90.6 %
3	99.9 %	95.6 %	96.1 %	95.9 %	93.1 %	90.8 %
4	99.9 %	95.7 %	96.1 %	95.9 %	93.2 %	90.6 %
5	99.9 %	95.7 %	96 %	95.8 %	93.2 %	90.7 %

G.5.3. Conclusion for Topic 1 and Topic 2

The tests indicated that "Algorithm II" does perform better than "Algorithm I" under S/N ratios below 10 dB. However, "Algorithm I" performs slightly better than "Algorithm II" for S/N ratios exceeding 10 dB. [26]

When the algorithms were implemented on a TMS, "Algorithm I" exceeds the time constrain by a factor of 40 compared to the factor of 6 obtained by "Algorithm II". It must also be said that, neither one of these algorithms were optimized for implementation on a TMS. [26]

G.5.4. Algorithm II with a high pass and low pass filter

A 10 kHz high pass filter as well as a 2 MHz low pass filter was implemented during the examination of topic 3. The latter was done with noise reduction of the system in mind. This also aided in the avoidance of aliasing issues. In this test both fourth order analogue Butterworth high and low pass filters were used. An 8-bit ADC and a ± 1 mV signal input range was used as inputs. Table G.28 shows the detection rates obtained in the test. [26]

Table G.28 Phase IV, Topic 3's detection rates for the test [26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	99.9 %	95.7 %	96.5 %	95.5 %	93.6 %	91.3%
2	99.9 %	95.7 %	96.6 %	95.6 %	93.7 %	91.3 %
3	99.9 %	95.6 %	96.6 %	95.6 %	93.7 %	91.2 %
4	99.9 %	95.6 %	96.6 %	95.6 %	93.4 %	91.2 %
5	99.9 %	95.5 %	96.5 %	95.6 %	93.5 %	91.1 %

It could be seen that the detection capabilities of “Algorithm II“ were not affected by the low pass filter. Thus, in order to reduce noise and avoid aliasing problems, both filters might have been implemented in a practical solution.

G.5.5. Algorithm II with a High Pass Filter, Low Pass Filter and a 12-bit ADC

Topic 4 investigated the effect a 12-bit ADC compare to an 8-bit ADC would have had on the detection capabilities of “Algorithm II” both with analogue filters implemented as discussed in Topic 3. In test 1, a 12-bit ADC was used. Table G.29 displays the detection rates obtained in the test 1. [26]

Table G.29 Phase IV, Topic 4's detection rates for test 1[26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	98.9 %	97.6 %	97.5 %	97.1 %	95.3 %	93.5%
2	98.5 %	98.2 %	97.3 %	97 %	95.2 %	93.3 %
3	99 %	97.8 %	97.2 %	97 %	95.3 %	93.3 %
4	98.6 %	97.8 %	97.4 %	97.1 %	95.3 %	93.3 %
5	98.5 %	98 %	97.2 %	97 %	95.2 %	93.3 %

In test 2 an 8-bit ADC was used. Table G.30 summarises the detection rates obtained in the test 2. [26]

Table G.30 IV, Topic 4's detection rates for test 2 [26]

S/N / Signal	Inf	20 dB	15 dB	10 dB	6 dB	4 dB
1	97.6 %	98 %	98.1 %	96.7 %	95.6 %	92.8 %
2	97.8 %	98.2 %	98.3 %	96.8 %	95.6 %	92.7 %
3	98.1 %	98.3 %	98.1 %	96.8 %	95.6 %	92.7 %
4	97.9 %	98.3 %	98.3 %	96.8 %	95.6 %	92.8 %
5	97.7 %	98 %	98 %	96.7 %	95.6 %	92.6 %

The conclusion reached, with no uncertainty, “Algorithm II” detection capabilities were found to be better when implementing a 12-bit ADC rather than an 8-bit ADC. [26]

G.6. Phase V

G.6.1. Overview

In the previous section it was mentioned that neither of the two algorithms being tested were optimized for implementation on a TMS. Due to the results obtained in Phase IV, it was decided to investigate the possibility of optimizing “Algorithm II”. [11]

Optimization in this context refers to the removal of unnecessary overheads in the code for the specific hardware used or stated differently to optimally use the resources of the specific hardware used. See reference 11 for examples.

In the previous Phases the TMS320C6713 floating point digital signal processor was used. The mentioned processor cannot execute “Algorithm II” in real time, the operating speed of the processor is too slow. The latter causes the ADC buffer to overflow and data was lost. [11]

Also, the ADC values obtained were not scaled and were fixed-point values. The algorithm’s structure only used fixed-point arithmetic for calculation purposes. This led to a system review and the TMS was replaced by the TMS320C6416 fixed-point digital signal processor. [11]

The following issues were explored: [11]

1. Optimization of, ”Algorithm II” and the utilization of the hardware specific features, for the TMS.
2. Acceleration of the digital system’s hardware by implementing “Algorithm II” on a FPGA (field programmable gate array).

G.6.2. Test

To investigate the effects of the above mentioned issues on the detection capabilities of “Algorithm II”, a certain amount of tests were conducted. All tests were repeated ten times for the purpose of statistical accuracy. Three tests were conducted namely: [11]

- *Tag range test;* In this test a tag was moved through the antenna field of reception at different distances from the antenna for a period of 15 s starting at the absolute maximum range (see figure G.2). The baseline used for comparison during this test was the detection rate obtained by the older analogue system. The test has shown how the two digital systems performed against the analogue system. [11]

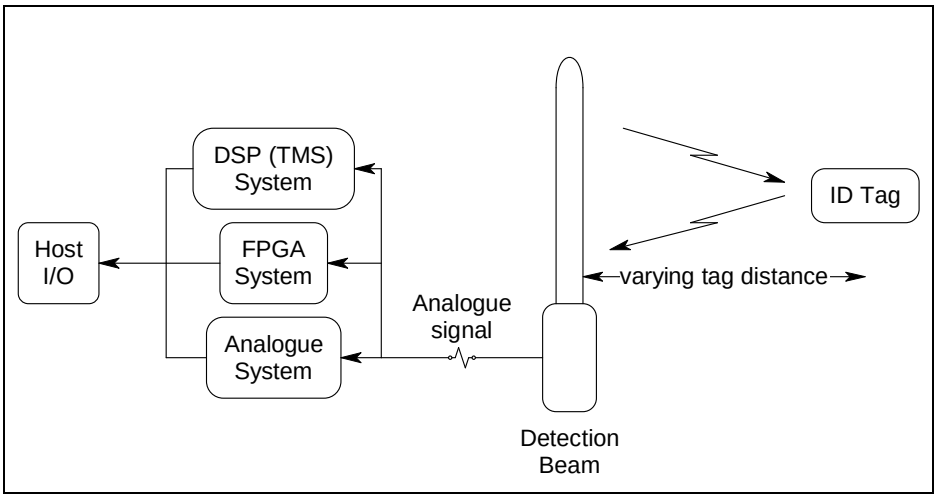


Figure G.2 Phase V, Tag range test setup [11]

- *Number of tags detected;* Six tags were held in the antenna field of reception for a period of 15 s (see figure G.3). This test showed the performance of the different systems when multiple tags were present and thus displayed the

systems' ability to operate under strenuous time consuming operational conditions. [11]

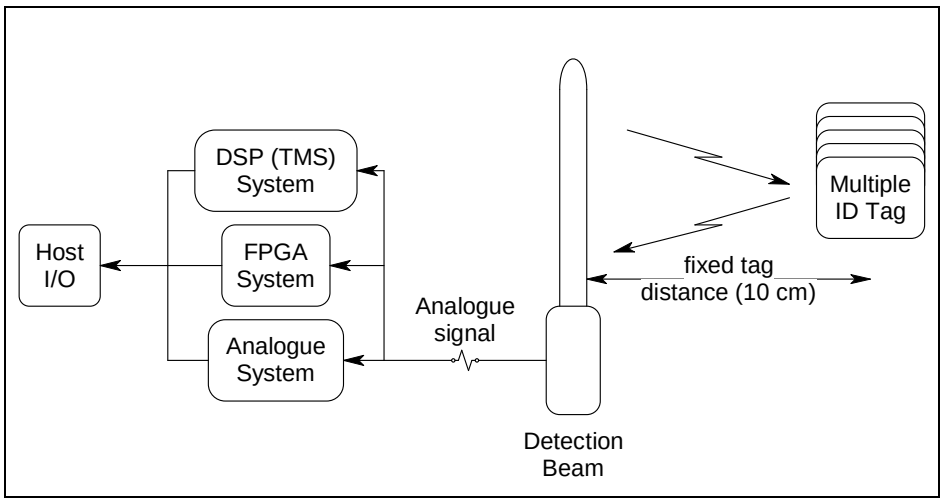


Figure G.3 Phase V, Total tags detected test setup [11]

- *Cycle count test and number of instructions;* In order to compare the optimised DSP (TMS) system to the original DSP (TMS) system, the CPU (central processing unit) cycle counts of the two systems were compared. The cycle count was measured while 15 different tags, at the maximum detection range, were being held in the antenna field of reception. The total number of instructions used for the pulse detection was also compared. [11]

  Tests were done at different optimisation stages in the algorithm to illustrate that the algorithm is in fact being optimised. Each set of tests were done for a duration of 30 s. [11]

  Four different versions of the software were used to conduct this test.

  - Version one (V1) is the un-optimised DSP system and forms the baseline for these tests.
  - Version two (V2) included the sign optimisation (refers to corrections of the averaging filter code, which had invalidated the signal, which coarse unnecessary testing in the code) for the averaging filter, and tag data storage method optimisation, which included CRC function optimisation.

- Version three (V3) included the if-else-if statement (refer to the optimization technique used for nested if statements, see reference 11 for details) optimisation and all of the V2 optimisations.
- Version four (V4) included all of the optimizations brought on under V3 as well as a more refined if-else-if statement optimisation and EDMA (enhanced direct memory access) hardware peripheral optimisation. [11]

G.6.3. Results

The average results for the "Tag range test" are shown in table G.31. The results show that the FPGA does indeed keep up with the performance of the analogue system however, the DSP systems amongst themselves cannot compete. [11]

Table G.31 Phase V, Tag range test results [11]

Range	Analogue (Baseline)	FPGA	Optimised DSP	Un-optimised DSP
10 cm	215.2	212.6	175	164
25 cm	215.2	211.6	173.8	163
50 cm	217.2	214	163.8	154
75 cm	210	204	96	93.2
100 cm	154.6	192.8	83.2	80.6
125 cm	87.2	94.2	5.2	4

The results for the "*Number of tags detected"* test are summarised in table G.32. The same conclusions as those mentioned for the "Tag range test" apply here. A possible reason for both the DSP systems performing poorly was that data may have been lost during the buffer swapping. (The DSP system utilize two buffer while ADC data were being read into one of the buffers, the digital detection algorithm had read ADC data out of the other buffer. This technique was implemented to compensate for the fact that the DSP could not execute the digital algorithm in real time). [11]

Table G.32 Phase V, Total detected tags test results [11]

Test run	1	2	3	4	5
Analogue	1201	1184	1182	1206	1219
FPGA	1180	1169	1166	1178	1209
Optimised DSP	876	872	873	875	863
UN-optimised DSP	835	836	830	837	835

The results for the "*Cycle count test and number of instructions*" test are shown in table G.33. The results show that the optimised DSP system was less computational intensive than the un-optimised DSP system. [11]

Table G.33 Phase V, Criteria 3 test results [11]

	V1		V2	
	Algorithm	**CRC function**	**Algorithm**	**CRC function**
Instruction count	4862432	6032	3229992	896
Cycle time (µs)	8194.25	10.56	5386.48	1.57
	V3		**V4**	
	Algorithm	**CRC function**	**Algorithm**	**CRC function**
Instruction count	2475240	896	2260464	424
Cycle time (µs)	4180.59	1.47	3766.52	1.57

G.6.4. Conclusion

The latter investigation showed that the FPGA was a viable alternative for the older analogue system. Although the digital system performed on par with the analogue system, the following issues surfaces as reasons for the development of a FPGA system. [11]

- The FPGA digital system is cheaper.
- The FPGA digital system is more scalable.
- The FPGA digital system can still be optimised for better performance.
- The FPGA digital system performs better under noisy conditions than the analogue system.
- The digital realm provides a sufficient platform for many algorithms not available in an analogue system.

The FPGA system could still be improved upon. The simple implementation of a FIR filter optimised the updated "Algorithm II" as well as optimised the VHDL code. [11]