Implementation and Evaluation of a Phelix Encryption/Decryption IP-Block

RICKARD NORSTRÖM

Master's Degree Project
Stockholm, Sweden 2010-03-09

TRITA-ICT-EX-2010:50

Abstract

In our modern day life communication is a highly integrated part and we keep finding new ways to share information with other people. Not all information we share with our frinds, co-workers or different corporations is something we want the whole world to be able to see. This makes encryption more and more a thing that everyone must use, and most of us living in the developed part of the world uses encryption in out everyday work. This thesis is the result of a project to implement the Phelix stream cipher algorithm on an Altera Cyclone II FPGA circuit.

The finished circuit is compared with other Phelix implementations on both hardware and software. It is also compared to the AES block cipher on hardware and software for exapmple with regards to key setup, speed and size.

The design is realised using Verilog 2001 and is simulated using Modelsim and Quartus II to verify the results on the indended hardware. The project is completed with support from InformAsic AB in Gothenburg

Comparison with other results shows that the simulated results of this design can be competitive with other hardware implementations of the algorithm as well as implementations of AES.

Sammanfattning

I vårt dagliga liv är kommunikation en stor del och vi kommer hela tiden på nya sätt att kommunicera med andra människor. All information som vi på olika sätt meddelar våra vänner, arbetskollegor eller olika företag vill vi inte att hela världen skall ta del av. Detta gör att kryptering av informationen blir vanligare och något som i stort sett varje människa i den utvecklade delen av världen använder varje dag. Den här rapporten är resultatet av ett examensarbete med att implementera strömkrypteringsalgoritmen Phelix på en Altera Cyclone II FPGA. Den resulterande kretsen är jämförd med andra Pheliximplementationer på hårdvara och mjukvara samt blockkryptot AES realiserad i hårdvara och mjukvara bl.a. med hänseende till nyckelbyte, hastighet och storlek.

Arbetet genomfördes med kodning i Verilog 2001 som simulerades i Modelsim och Quartus II för att verifiera resultaten på den tänkta hårdvaran. Projektet genomfördes i samarbete med InformAsic AB i Göteborg.

Jämförelser med tidigare nämnda implementationer visar att designen kan vara konkurrenskraftig med andra hårdvaruimplementationer av Phelix men även med implementationer av AES.

Acknowledgement

I would like to thank my supervisor Björn Olsson at InformAsic and Joachim Strömbergson, also at InformAsic for the opportunity to do this thesis and for the support I have been given during the project. I would also like to thank my examiner at ICT Ingo Sander for his patience and David Kjellberg at Arrow Nordic for the help with Altera software

List of Figures

1.1 Example of the Caesar cipher 2
1.2 A two round Feistel network 4

2.1 Example of how the Electronic Code Book works . . . 11
2.2 Example of how the Cipher Block Chaining works . . . 12
2.3 Example of how the Cipher Feedback works 12
2.4 Example of how the Counter mode works 13
2.5 Idea of the Vernam cipher 14
2.6 Basic layout of a stream cipher 14
2.7 A complete block of Phelix from the original paper . . . 17
2.8 The idea behind a S-P network 22
2.9 A 16 byte block input in the AES cipher 23
2.10 The Add round key step 24
2.11 The Substitute bytes step 25
2.12 The Shift row step 26

3.1 The original idea of the top hierarchy 28
3.2 The top hierarchy of the design 29
3.3 Inside of the circuit 30
3.4 The Calculation block 32

4.1 The states of the FSM 34
4.2 Steps in the key mix state 36
4.3 Register placement to divide the block function 40

List of Figures

List of Tables

2.1 The Block function H 16
2.2 Steps of operation during encryption in AES 21
2.3 S-box LUT for encryption in AES 25
2.4 Steps of operation during decryption in AES 27

5.1 Usage of LE in the design 43
5.2 Speed and throughput with different models 43

6.1 Comparison between different implementations 46

List of Tables

Glossary

A

AES Advanced Encryption Standard.

C

CBC Cipher Block Chaining.

CFB Cipher Feedback.

CTR Counter mode.

D

DES Data Encryption Standard.

F

FIFO First In First Out buffer, in software basically a queue.

FPGA Field Programmable Gate Array.

FSM Finite State Machine.

I

IBM International Business Machines.

IV Initialization Vector.

L

LE Logic Elements.

LUT Look Up Table.

M

MAC Message Authentication Code.

N

NBS National Bureau of Standards.

Nonce Number used once.

Contents

Glossary **xiii**

1 Introduction **1**
1.1 Background 4
1.1.1 eSTREAM 5
1.1.2 NESSIE 6
1.1.3 ECRYPT 6
1.1.4 Cyclone II 6
1.1.5 NIOS II 7
1.1.6 InformAsic AB 7
1.2 Related Work 8
1.3 Problem formulation 9
1.4 Outline of the report 9

2 Theory **10**
2.1 Block cipher 10
2.2 Stream Cipher 13
2.2.1 MAC 15
2.3 Phelix Algorithm 15
2.3.1 The Phelix Block 16
2.3.2 Key Words X 18
2.3.3 Key Mixing 18
2.3.4 Initialisation 19
2.3.5 Encryption 19
2.3.6 Decryption 20
2.3.7 MAC 20

2.4 Advanced Encryption Standard 20
2.4.1 Encryption 23
2.4.2 Decryption 26

3 Method **28**
3.1 Top hierarchy 28
3.2 Control signals 29
3.3 Controller 30
3.4 Calculation Block 31

4 Implementation **33**
4.1 Controller 33
4.2 FSM and states 33
4.2.1 Idle state 35
4.2.2 Key In state 35
4.2.3 Key Mix state 35
4.2.4 Init state 36
4.2.5 Encryption 37
4.2.6 Decryption 37
4.2.7 MAC 38
4.2.8 Done Wait 38
4.2.9 Nonce In 38
4.3 Calculation Block 38
4.3.1 FIFO 41
4.3.2 Rotation 41
4.4 Reset 41

5 Results **42**
5.1 Software and Simulation 42
5.2 Speed and Area 43

6 Discussion **45**

7 Conclusion and Future Work **48**
7.1 Conclusion 48
7.2 Future Work 50

Chapter 1

Introduction

Encryption is a part of modern life that we most of the time do not need to worry about. It is present when we surf on the Internet, pay our bills or talking with friends and associates on our cell phones.

The art of keeping messages secret can be divided into two different parts. One is making it hard to get hold of the message which is called steganography. The other is making it hard to read the message which is called cryptography.

To hide the messages from your enemies has been done for a long time. According to The Code Book[22] by Simon Singh, a man called Demartos used writing tablets to pass a warning to the Greeks that Xerxes was going to attack by removing the wax from the tablet and carving into the wood that the tablet was made of. When he the reapplied the wax, the message was hidden. Even in modern time hiding the messages has been a used method. An example is microdots that are a scaled down version of a message that is placed somewhere to be passed on to the receiver. The Germans used microdots during Word War 2 placing them over dots in texts to hide them. Some of these were recovered and the message could be read by the allies, at least if it was written in plain text.

Cryptography on the other hand is done to make it hard, or preferably impossible, for an enemy to read the message sent. The oldest military used cryptographic invention is known as the scytale. The scytale

is a wooden stick which you wind a strip of some material around and write your message along the stick. When you then unwind the strip you get letters that are meaningless to a beholder. The strip is then carried to the receiver who has a stick with the same diameter as the sender and when the strip is wound onto this stick the message is again readable.

Another example is the Caesar cipher named after Julius Caesar. Caesar is known to have used ciphers when communicating with people around him. These were recorded by a man called Valerius Probus, unfortunately these records have been lost and the only remaining record is of the Caesar cipher made by Suetonius in his eight books of the Lives of the Twelve Caesars. The Caesar cipher substitutes the character that is about to be encoded with a character three steps ahead in the alphabet as seen in figure1.1

Plaintext	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Ciphertext	**d**	**e**	**f**	**g**	**h**	**i**	**j**

Figure 1.1: Example of the Caesar cipher

To use certain devices to encrypt messages, if the scytale is not counted as one, there are records of devices since the 15th century using a code wheel. It is not completely established if the code wheel was ever used, but the idea is dated to the 15th century Vatican. A later example is the Jefferson disk which is a number of disks with letters written along the circumference of the disks and with a hole in the middle to place them on an axle. Encryption is made by placing the disks in a certain order and then arranging the disks so that one row spells out the intended message. Another row is then chosen as the key. Then you send the order of the disks as placed on the axle and the key phrase. The receiver places the disks in the given order and then rotates them so that the key phrase appears, the receiver then looks through the rows of the disks to find the message. This method of course has the limitation of message length since you only have a certain number of disks. It also has the limitation that there has to be identical disks at the locations of senders and receivers making the risk of them ending up in the had of the enemy

larger.

With the invention of electronic machines, encryption and decryption was made more convenient since you no longer had to do it by hand. One of the more known machines is the German Enigma from the Second World War. It was built using several wheels that interlocked and this way gave, what was originally an easy substitution cipher, a more complex way of substituting the letters. This was done by setting the wheels starting positions in a certain way according to the instructions given for the given message. Although ingenious in design it was broken by the allied forces even though changes were made to it during the entire war.

You can of course combine cryptography and steganography to make it even harder for the enemy. The strip used on the scytel could be of leather and used as a belt by the courier and the text on the microdot could be encrypted. All steps to make it harder for anyone but the intended receiver to read the message increases the security.

The first major civilian standard for encryption was made during the early 1970'ies on a request from NBS in USA which ended up in a block cipher called Data Encryption Standard, DES.

The DES was a version of an algorithm called Lucifer created by Horst Feistel employed by IBM that entered, and later won, the competition for DES. The method used in the DES was later known as a "Feistel Network" and has been used in many ciphers since 1973. A Feistel Network as seen in figure 1.2 is made up of the F-function, that can be implemented in different ways, and a series of XORs. The Feistel cipher splits the input plaintext into two halves of equal size during encryption or decryption, using the F-function on half the data and letting the other half through. There are also Feistel ciphers where the halves are not of equal size, these are referred to as "unbalanced" Feistel ciphers.

The network seen in figure 1.2 is made up of two rounds, the one used in DES has 16 rounds, making it more secure by scrambling the input data more. As the years passed DES was considered unsafe and changes were made to make it more secure. This resulted in variants such as triple DES, which as it sounds was DES used three times on the

data that was to be encrypted.

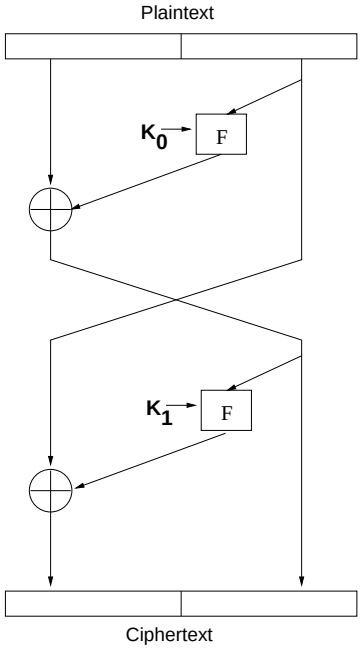

Figure 1.2: A two round Feistel network

The DES was replaced with the AES [7] in the year 2002 after another competition to select the "best" block cipher. The winning cipher of the AES competition is called Rijndael made by the Belgians Vincent Rijmen and Joan Daemen and is based on a design principle called Substitution permutation network. AES will be explained further in chapter 2.4.

1.1 Background

This thesis is, as previously mentioned concentrated around a cipher algorithm called Phelix[23], one of the original candidates in the eSTREAM project[4] for both hardware and software focus. Phelix is constructed by Whiting et all as a development of a previous algorithm by some of the same people that created Phelix, called Helix[14].

Helix and Phelix are stream ciphers as opposed to DES and AES that are block Ciphers. The main structure of Helix is identical to Phelix, although there have been some changes. The keystream output has been moved to increase the plain text diffusion and a feedback FIFO has been introduced.

1.1.1 eSTREAM

The eSTREAM[4] project is not a competition to determine a standard as was the case with DES and AES, it is a project with focus to get attention to promising stream ciphers.

The project was initiated after another project, NESSIE[5], failed in finding any viable stream ciphers that they could encourage using. The project eSTREAM was initiated by ECRYPT[2], the Network of Excellence in Cryptology, a network funded by the European Commission. The eSTREAM project had two different focus groups, one for software with high throughput and one for hardware with restricted resources. The project was active between November 2004 and May 2008, hosting 5 workshops and having a final portfolio of 7 algorithms, 4 for software and 3 for hardware.

The final portfolio of eSTREAM is not to be considered a standard, but rather a group of ciphers that has no apparent weakness and will be a functional solution to use.

In the process of this thesis work, Phelix was archived as a cipher that was not interesting enough to be put forward as one of the portfolio ones. This was due to a proposed weakness to the algorithm that was posted by Wu and Preneel[24]. Although Whiting et all answered that the attack was done by using the same key and nonce combination even though the paper they submitted clearly pointed out that each combination of key and nonce should be used only once. The attack should not be valid for this reason.

The response from eSTREAM was that in the paper submitted to the project[23], it was stated in the definition of Phelix that even though each key-nonce combination should be used only once, they claimed that it should not be a problem with attacks even if the key-nonce combination

was used for several encryptions.

1.1.2 NESSIE

The NESSIE project was a European project intended to contribute to the final parts of the NIST AES project aswell as investigating primitives to present as a portfolio of block ciphers, stream ciphers, hash functions, MAC algorithms, digital signature schemes and public-key encryption schemes. The project had an open call for cryptographic primitives to evaluate and scrutinise to make the final portfolio of algorithms that are useful. The project took little over three years to complete leaving a gap in the stream cipher part of the portfolio. This was the reason that eSTREAM[4] project was initiated. There were no stream ciphers entered that were good enough to propose in a portfolio.

1.1.3 ECRYPT

ECRYPT[2], European Network of Excellence for Cryptology, and subsequently ECRYPT II[3], is a network meant to "intensify the collaboration of European researchers in information security, and more in particular in cryptology and digital watermarking". It was active between September 2004 to June 2008 and held a number of conferences and workshops intended to bring European cryptologists together and share their work between them aswell as running narrower projects as eSTREAM. After September 2008 ECRYPT II continued the works of ECRYPT.

1.1.4 Cyclone II

The Cyclone II FPGA is developed and manufactured by Altera[1]. Altera has been in business since 1983 and provides FPGAs, CPLDs and "Hardcopy ASICs". The FPGA series that are available from Altera is the Cyclone series, Arria series and the Stratix series FPGAs. The Cyclone is the low-end product mostly suited for smaller designs that are cost sensitive. The Cyclone was introduced in 2002.

The Cyclone II FPGA is a development from the previous generation Cyclone FPGA and has been improved in a few ways. The technology used has been altered from 130nm to 90nm making it possible to increase the number of Logical Elements (LE), containing a lookup table (LUT) and some logic, on the device. The Cyclone II comes in some different variants with numbers of LE ranging from 4608 to 68416 as a contrast to the previous version Cyclone that consisted of models with 2910 to 20060 LEs.

Together with Altera's design suite Quartus II the designer is able to construct powerful devices to be used in various systems or applications. The FPGA is an electronic device that can be programmed after construction, unlike the ASIC, to suit the different needs that the designer has. It can also be reprogrammed after the initial programming making it suitable to tasks where upgrades are common or regular.

1.1.5 NIOS II

The NIOS II softcore processor is a product from Altera[1] designed to be implemented in an FPGA. The NIOS II comes in three different variants, fast economy and standard with the smallest one possible to implement with less than 700 LEs on the FPGA device. The three variants have a common instruction set architecture making it easy to change between them if the demands should change. As it is a soft core processor it is possible for the developer to make changes to the structure of the system making it to suit the needs with respect to I/O circuitry and embedded peripherals for example.

1.1.6 InformAsic AB

InformAsic is a company started in 2001, with the headquarter in Gothenburg. InformAsic specialise in design of cost effective electronic devices adapted to the needs of the customer. The main areas of expertise are Wireless Communication, High Speed Design and Infotainment realised on ASIC, FPGA or Embedded Processors.

1.2 Related Work

There are not many implementations of Phelix presented in detail. Most of the implementations found was made during the course of the eSTREAM project, which makes sense considering that Phelix was presented to this project and it was put aside before the final portfolio was presented.

Looking at hardware implementations of AES is a different matter. There are many presented with differing results. Looking at the paper presented by Chang et all [12] we can see that it is a very high performance algorithm that is possible to make an efficient device with little hardware costs. They succeeded in producing an impressive throughput using a small hardware footprint. At the same time it is a good example of a paper not saying much about the conditions of the experiment. Chang et all present their data saying that the have a throughput of 552 Mbps on a Xilinx Spartan 3 FPGA using 158 slices and 3 RAM blocks. They don't say anything about the setup delays and which mode of operation they used.

In the paper published by Fu et all [15] there is more information regarding the mode of operation (counter mode) and how the circuit was created. This implementation is not directly comparable with the implementation presented in this thesis since they chose to use the high-end FPGA series Virtex from Xilinx. The results are still impressive with a throughput of 1.49 Gbps in the general implementation and up to 27 Gbps with a pipelined version reaching clock frequencies of 212 MHz. The hardware footprint of 17800 slices is not usable in all uses though.

The reports filed with eSTREAM [4], for example [11] and [16], containing implementations of Phelix on hardware all state that Phelix is an algorithm that is easy to implement in hardware, which it should be since it was a hardware focus algorithm, and that the only choice that has to be made is how the key stream generator should be implemented, half block or full block. The report filed by Schaumont and Verbauwhede [20] was one of the pointers of how the implementation presented in this thesis should look in general. Even though this implementation is a hardware-software codesign, the idea to have a separate block to handle the states

of the helix shaped permutations of the internal state words.

1.3 Problem formulation

The purpose of this thesis project is to create an encryption/decryption device that uses the Phelix algorithm and that can be used alongside another device for example a NIOS II core to see if the algorithm is useful. The target technology for the design is an Altera Cyclone II FPGA device and the code was to be written in Verilog 2001[6].

The circuit was designed to have a 32 bit data-in bus, a 32 bit data-out bus, some control signals and a synchronous reset beside the clock signal. The synchronous reset was later changed to be an asynchronous reset to be able to reset the circuit without being held up by the clock.

After the design and coding process the results were to be compared with AES on both hardware and software, and also the software results presented by Whiting et all of how the Phelix performed in a software environment on different types of computer hardware.

This thesis project was performed with the support of InformAsic. The focus of the thesis is to investigate how to implement the Phelix stream cipher algorithm on an FPGA as an encryption device.

1.4 Outline of the report

This thesis is divided into six chapters where this is the introduction chapter which is intended to give a rough introduction to the thesis project but also to the differences between cipher types and to the target hardware.

The following chapters will contain an overview of the algorithm used (chapter 2), how the work was structured (chapter 3) followed by the implementation chapter (chapter 4) where the structure and coding of the device will be presented.

After the implementation chapter follows three chapters containing results, discussion and conclusions of the project.

Chapter 2

Theory

The Phelix algorithm is described by the authors (Whiting et all [23]) as a high speed stream cipher with built in MAC functionality (explained in section 2.2.1. It is constructed to work with a 256 bit key and a 128 bit "nonce", which can be seen as an IV for the algorithm. The key is in general seen as a secret while the nonce very well could be public knowledge. The important thing is to never encrypt two different messages using the same key-nonce pair. To do so could compromise the encrypted message although the authors claim that not even then is it likely that the key can be recovered, however, Wu and Preneel entered a report to eSTREAM in 2006 [24] claiming to have recovered the key using multiple attacks with the same nonce.

This chapter gives a basic view of the algorithm mainly focusing on how to set up and run it in the mode that was chosen for this circuit. For those who is interested in the full paper on Phelix it is available at Bruce Schneier's homepage [23]. It will also cover the AES algorithm and give a basic introduction of how block ciphers and stream ciphers work.

2.1 Block cipher

Block ciphers are a group of ciphers that work on blocks of data of a certain size using a secret key along with the algorithm to encrypt or decrypt the data. If the data size is larger than the block size of the cipher, another block is encoded. The block ciphers can work in different

"modes" and in its most basic mode of operation the block cipher is a computerised version of the early day code books or translation tables and is called the Electronic Code Book 2.1.

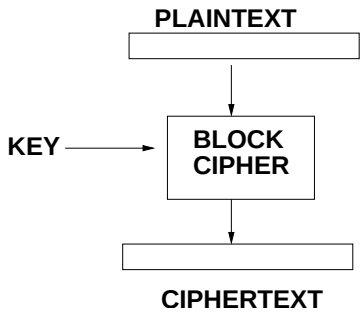

Figure 2.1: Example of how the Electronic Code Book works

An insecurity of block ciphers using the Electronic Code Book mode is that if the same message was encrypted twice it would result in the same ciphertext. That is a vulnerability that has been addressed by using for example Initialisation Vectors (IV) that contain random data to make the encryption a bit less easy to predict. The IV is used together with another mode of operation from the electronic code book, to make the encryption more secure. These modes use feedback or propagation of either ciphertext or plaintext or both into the next block that is to be encrypted.

One example is the Cipher Block Chaining or CBC which can be seen in figure 2.2. With this mode the ciphertext of the first block encrypted is set as the initialisation vector for the second block giving a more scrambled output than the Electronic Code Book mode.

With some of these modes the block cipher works in many ways like a stream cipher, further explained in section 2.2, making it hard to distinguish between the two types. An example of that is the Cipher Feedback mode or CFB. Here the ciphertext is propagated to the next block as an initialisation vector again. However the plaintext is introduced first after the block cipher has been applied to only the initialisation vector producing what could be described as the keystream of a stream cipher.

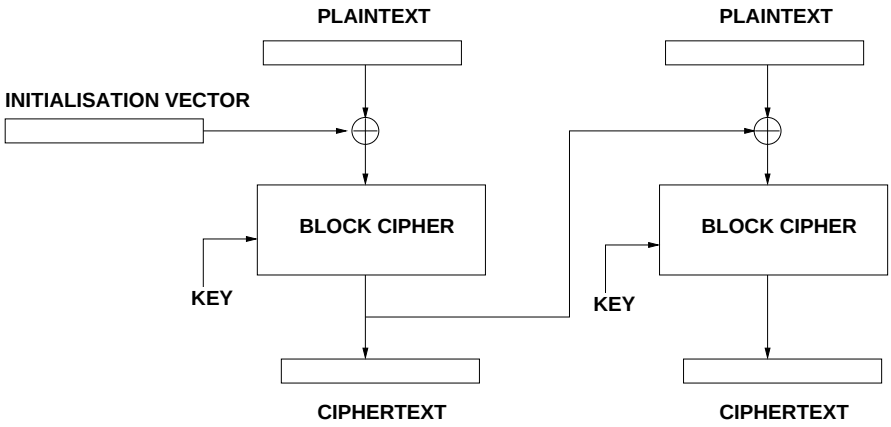

Figure 2.2: Example of how the Cipher Block Chaining works

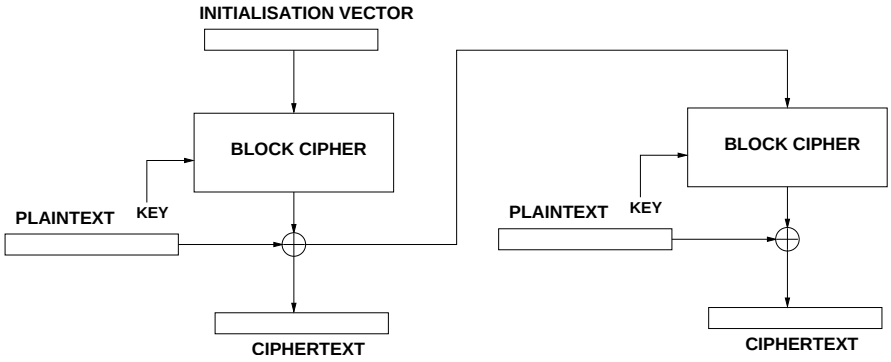

Figure 2.3: Example of how the Cipher Feedback works

The encryption of a CFB-mode block cipher is seen in figure 2.3.

Another way to turn a block cipher into a stream cipher is the Counter mode or CTR. This mode uses the IV called a Nonce together with a counter as input to the cipher module producing a keystream that is used to encrypt the plaintext. This mode of operation turns the block cipher into a synchronised stream cipher. An example of how a block cipher in CTR-mode looks like can be seen in figure 2.4.

One thing that makes the counter mode and cipher feedback mode even more like a stream cipher is the properties of how they encrypt and decrypt messages using only an encryption block to create the keystream

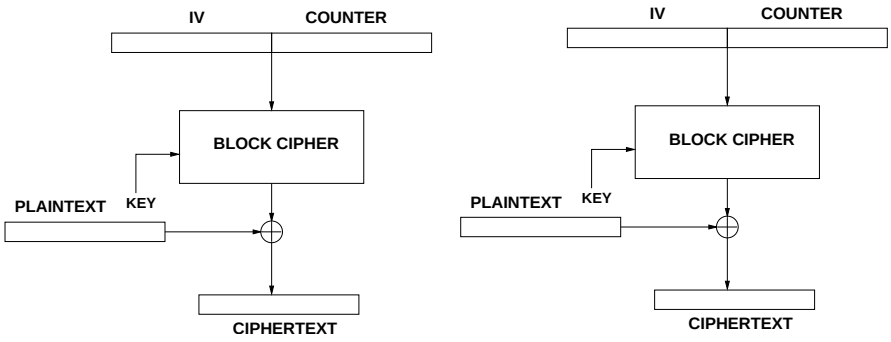

Figure 2.4: Example of how the Counter mode works

while the other modes have a decryption block to decrypt the cipher text. This can make the implementations of these modes smaller than other modes if the algorithm used has different blocks for encryption and decryption.

2.2 Stream Cipher

Stream ciphers work somewhat differently from block ciphers in that way that they encrypt data using "new" keys, called keystreams, for each data block. The datablock is also smaller than for a block cipher, often being a bit or a byte.

There is a similarity between stream ciphers and the "one time pad", a cipher method that is unbreakable if used correctly where you use a new key for each time you encrypt something. The one time pad, or Vernam cipher, consists of two pads that the sender and reciever has one each and for each message sent one or more pages from the pad is used to encrypt or decrypt. These pages are then destroyed if the system is used correctly. The keys consist of random characters that is used to encrypt one letter each by adding the letter number of the character to be encrypted with the letter number of the key in that way that if the plaintext character "a" is to be encrypted with the key character "f" you add 1 and 6, getting 7 which is the number for the letter "g". To decrypt you reverse the process and subtract 6 from 7 getting 1. The idea is

illustrated in figure 2.5.

Plaintext: F I ND T HE GRA I L
Key: AHFJ V XD HRJKB
Ciphertext: GQTN MC I OGKTN

Figure 2.5: Idea of the Vernam cipher

To encrypt a 1000 character long message you obviously need 1000 characters of key material making this a cumbersome way to encrypt message, although it is very secure since the only two that know the key for this message are the sender and the reciever. The problem with this system is that it is not so easy to create completely random keys and that the keys need to be distributed to sender and reciever aswell as keeping the creation process secret.

Stream ciphers work with a similar idea creating "unique" data blocks of key-stream that is combined with the plaintext to create the ciphertext. The problem is to make unique keystreams and while the keystreams can look unique to an observer, when you start examining them there are often some pattern in these. This is why they are called "pseudo random". There are often a maximum length of plaintext that a stream cipher algorithm can encrypt. One reason is that the algorithm starts repeating itself after a certain amount of generated keystream making the ciphertext less secure.

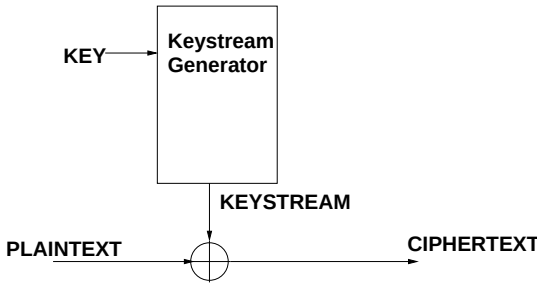

Figure 2.6: Basic layout of a stream cipher

These ciphers can be divided into two subtypes called "synchronous

stream ciphers" and "self synchronising stream ciphers". The former type have a keystream generator that works independently from the provided plaintext and produces keystrems that are added to the data provided. The self synchronising stream ciphers introduce some of the plaintext into the keystream generator and it becomes part of the keystream. An advantage of synchronous stream ciphers is that it can generate keystream independently from the data to be encrypted or decrypted and used afterwards.

2.2.1 MAC

MAC, or Message Authentication Code, is a way to ensure that the message recieved is authentic. The MAC is constructed using the same key as the message encrypted and the reciever uses the same key to make sure the message is authentic. This is different from a digital signature that uses PGP encryption.

2.3 Phelix Algorithm

Phelix is considered to have a design strength of 128 bits, although it uses a key of up to 256 bits. Shorter keys can be used however, this requires the key to be padded to 32 bytes. Phelix works with 32 bit blocks of Plaintext, or Ciphertext, with the least-significant-byte-first convention during the encryption/decryption process. It is specified to have 8-bit input and output in the definition paper, although the circuit described in this paper is designed to use 32 bit input and output.

The algorithm is specified to operate on plaintext (or ciphertext) up to 2^{64} bytes long (18 exabyte). Since the algorithm is based on 32-bit words only parts of the last words might be used. The remaining bytes are zero padded and the amount of data in the last word is part of the MAC, which will be explained later.

To make the algorithm a running unit, there are five major parts to consider. These are *key mixing*, *initialisation*, *encryption*, *decryption* and *computing the MAC*. Encryption and decryption works very similar but

Function $H(w_0, w_1, w_2, w_3, w_4, K_0, K_1)$
Begin

$$
\begin{aligned}
w_0 &:= w_0 \boxplus (w_3 \oplus K_0); & w_3 &:= w_3 \lll 15;\\
w_1 &:= w_1 \boxplus w_4; & w_4 &:= w_4 \lll 25;\\
w_2 &:= w_2 \oplus w_0; & w_0 &:= w_0 \lll 9;\\
w_3 &:= w_3 \oplus w_1; & w_1 &:= w_1 \lll 10;\\
w_4 &:= w_4 \boxplus w_2; & w_2 &:= w_2 \lll 17;
\end{aligned}
$$

$$
\begin{aligned}
w_0 &:= w_0 \oplus (w_3 \boxplus K_1); & w_3 &:= w_3 \lll 30;\\
w_1 &:= w_1 \oplus w_4; & w_4 &:= w_4 \lll 13;\\
w_2 &:= w_2 \boxplus w_0; & w_0 &:= w_0 \lll 20;\\
w_3 &:= w_3 \boxplus w_1; & w_1 &:= w_1 \lll 11;\\
w_4 &:= w_4 \oplus w_2; & w_2 &:= w_2 \lll 5;
\end{aligned}
$$

Return $(w_0, w_1, w_2, w_3, w_4)$;
End.

Table 2.1: The Block function H

contains a few differences as will be shown later.

2.3.1 The Phelix Block

Phelix is built around a block function illustrated in figure 2.7. It is structured around five 32-bit working words Z_0^i to Z_4^i and four 32-bit feedback words Z_4^{i-4} to Z_4^{i-1}. These words are being modified using exclusive or, addition modulo 2^{32} and rotate. In this report $\oplus$ represents exclusive or, $\boxplus$ represents addition modulo 2^{32} and $\lll$ represent rotation.

In each block three more 32-bit words are introduced, two key words $X_{i,0}$ and $X_{i,1}$ alongside the plaintext word P_i. The block is divided into two identical half block functions described below.

To create the keystream used to encrypt or decrypt the data being issued, a 32-bit word w_4 (or Y_4^i in figure 2.7) which is generated after the first half block has finished. The keystream S_i is the sum modulus 2^{32} of Y_4^i and Z_4^{i-4}.

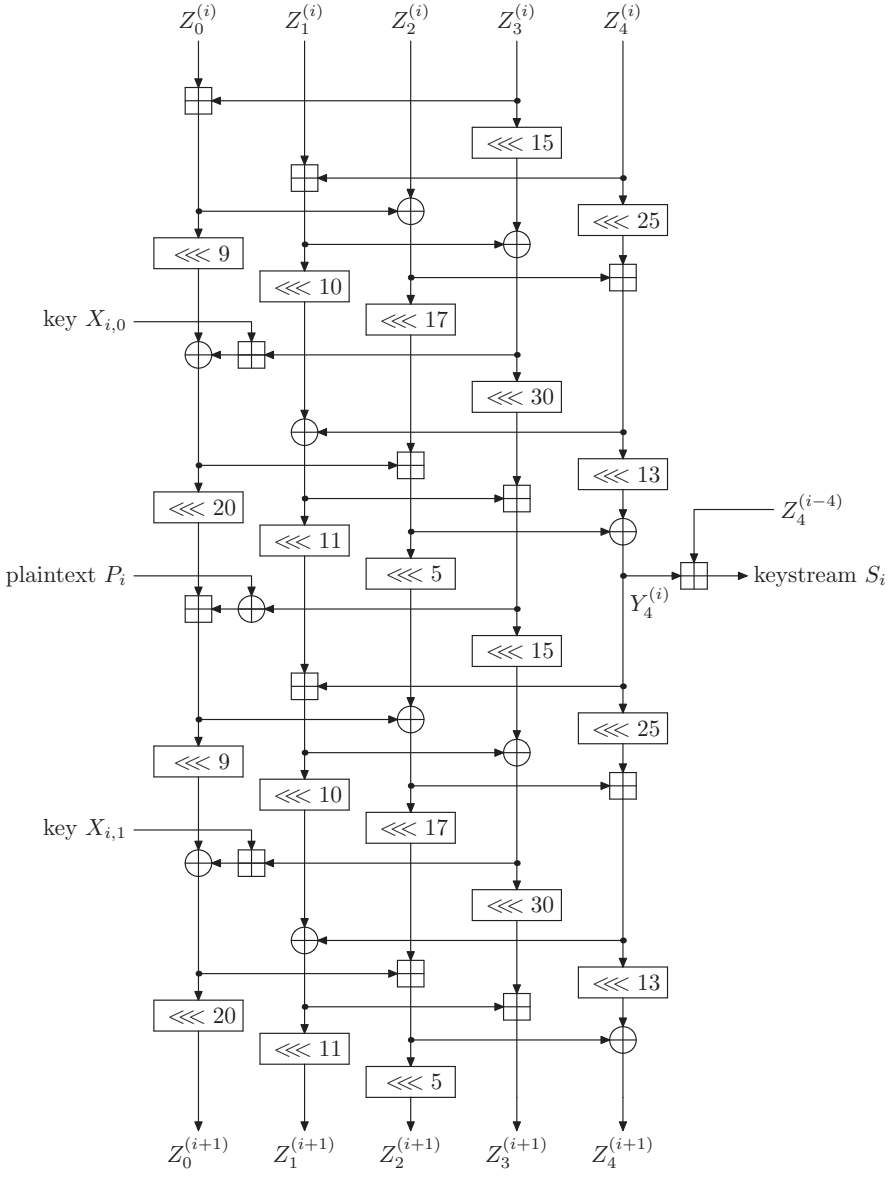

Figure 2.7: A complete block of Phelix from the original paper

With the definition of the function H in table 2.1, the block function

can be described

$$(Y_0^i, Y_1^i, Y_2^i, Y_3^i, Y_4^i) = H(Z_0^i, Z_1^i, Z_2^i, Z_3^i, Z_4^i, \quad 0, X_{i,0}) \tag{2.1}$$

$$(Z_0^{i+1}, Z_1^{i+1}, Z_2^{i+1}, Z_3^{i+1}, Z_4^{i+1} = H(Y_0^i, Y_1^i, Y_2^i, Y_3^i, Y_4^i, \quad P_i, X_{i,1}) \tag{2.2}$$

2.3.2 Key Words X

The key words $X_{i,0}$ and $X_{i,1}$ are calculated throughout the encryption using equations 2.4. The key words K_i are the 8 working key words derived from the raw key explained in section 2.3.3 while the nonce words N_0 trough N_7 are the four nonce words provided by the user expanded to eight words with the relation described in equation 2.3

$$N_k = (k \mod 4) - N_{k-4}(mod 2^{32}) \text{ for k = 4,...,7} \tag{2.3}$$

$$\begin{aligned} X_{i,0} &= K_{i \bmod 8} \\ X_{i,1} &= K_{(i+4) \bmod 8} + N_{i \bmod 8} + X_i^{'} + i + 8 \\ X_i^{'} &= \begin{cases} \lfloor (i+8)/2^{31} \rfloor & \text{if } i \bmod 4 = 3 \\ 4\dot{l}(U) & \text{if } i \bmod 4 = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2.4}$$

These equations provide us with the key words needed for each block during the operations. In equation 2.4, $l(U)$ is the length of the raw key provided by the user, all through this paper the raw key is of length 256 bit giving us a $l(U)$ of 32.

2.3.3 Key Mixing

The initialisation of Phelix starts with mixing the key. Beginning by converting the 32 raw key bytes to 8 words $K_{32}, \ldots, K_{39}$. To mix the raw keys into the working keys the block function is used. We start by setting the state word w_4 (i.e Z_4^i) to 96. The remaining four state words Z_0^i

through Z_3^i is given the values of the raw keys using equation 2.5

$$(K_{4i}, \ldots, K_{4i+3} = R(K_{4i+4}, \ldots, K_{4i+7}) \oplus (K_{4i+8}, \ldots, L_{4i+11}) \quad (2.5)$$

where $i = 7, \ldots, 0$, and R is defined being a full round of the block function with the key words as starting state words and the resulting state words $Z_0^{(i+1)}$ through $Z_3^{(i+1)}$ being the resulting words. The words K_0 through K_7 are given after the recursive mixing have run 8 rounds and these are the key words needed for the encryption/decryption operation.

2.3.4 Initialisation

After the key mixing the working words need to be set up for operation. This is done by setting the working words as follows

$$\begin{aligned} Z_j^{-8} &= K_{j+3} \oplus N_j \quad \text{for } j = \quad 0, \ldots, \quad 3 \\ Z_4^{-8} &= K_7 \end{aligned}$$

After this is done, the block is run eight turns giving us eight words of key stream that is discarded. For these eight blocks the plaintext word is set to 0. The feedback words Z_4^{i-4} for i $-8, \ldots, -5$ are set to 0 to start the initialisation of the feedback. These eight rounds mixes the working words as well as filling the feedback FIFO.

2.3.5 Encryption

Encryption utilises the block function shown in section 2.3.1, the keys described in section 2.3.2 and the plaintext word provided by the user. Ciphertext is generated with the keystream word S_i and the plaintext word P_i with the relation $C_i = S_i \oplus P_i$. Since Phelix works on 32-bit words but takes in data in byte size parts, the last word might only be partially filled with data. Since the decryption relies on xor as well, the final word of decrypted data will be all zero if the input data is it, giving the zero padding in the plaintext encrypted resulting in output zero padding after decrypting the ciphertext, which is exactly what is expected of a decrypted ciphertext.

2.3.6 Decryption

Decryption of ciphertext is very similar to encryption. The reverse relation $P_i = C_i \oplus S_i$ gives us the plaintext. However, since the plaintext is needed in the block function to generate the correct working words, and in turn the keystream, the plaintext must be generated as soon as the keystream is provided and then propagated into the block function as plaintext.

2.3.7 MAC

Phelix has a built in MAC generation that will generate a tag for each encoded message as well as for each decoded message. This is useful to determine whether the received message is authentic or not. The downside is that it is only possible to determine if it is an authentic message after the entire message has been deciphered. Other ciphers have a MAC generation as an option meaning that it will need to do a setup to generate a MAC, this is an advantage to Phelix if a MAC is needed since it is automatically generated.

After the last block is encrypted, or decrypted, the working word Z_0^k is xored with the value 0x912d94f1. The block function is then run eight turns to set up the working words for MAC generation in the same manner as the working words are set up for the encryption or decryption during the initialisation phase. For these turns the plaintextword is set to $l(P)$ mod 4, in other words the number of data bytes in the last block encrypted or decrypted. The keystream for these rounds is discarded, as was the keystream from the initialisation rounds.

After these executions of the block, four more rounds of the block function is applied using the same plaintext word, $l(P)$ mod 4, and the keystream words generated by these final blocks are the MAC.

2.4 Advanced Encryption Standard

The Advanced Encryption Standard is a slightly modified version of the cipher algorithm "Rijndael" [13] submitted by Vincent Rijmen and Joan

Step	Action	Description
Initialisation round	Add round key	Adds key to block
Rounds of encryption	Substitute bytes Shift rows Mix columns Add round key	Substitution using LUT Rotates rows of block Each column is multiplied with a polynomial
Final Round	Substitute bytes Shift rows Add round key	

Table 2.2: Steps of operation during encryption in AES

Daemen to the competition to find a new block cipher algorithm standard replacing the DES.

The AES works with key lengths of 128 bits, 192 bits or 256 bits and uses different computing lengths depending on the key size. The algorithm runs on blocks of data 128 bits long, or 16 bytes, and does 10 rounds of encryption to each block with the 128 bit key, 12 rounds with the 192 bit key or 14 rounds of encryption with the 256 bit key. Each block goes through the same cycle which can be described with the steps seen in table 2.4. The "Initialisation" step and the "Final round" is only performed once on each block while the "Rounds of encryption" is performed 9, 11 or 13 times depending on the key size.

AES is what is called a Substitution-Permutation cipher as compared to the DES that is a Feistel Network cipher. The difference is that AES uses Substitution boxes (S-boxes) and Permutation boxes (P-boxes) to scramble the data. How the S-P network in AES works will be explained here, a simple S-P-network however is seen in figure 2.8.

For the first round of a 128-bit key AES encryption, the key consists of the input key divided into four 32-bit words, after this the key expansion routine starts producing new round keys until the full ten rounds of encryption is done, using 44 key words altogether. Each byte of the state has its own byte of key which it is being Xored with. This key expansion is put in a key scheduler.

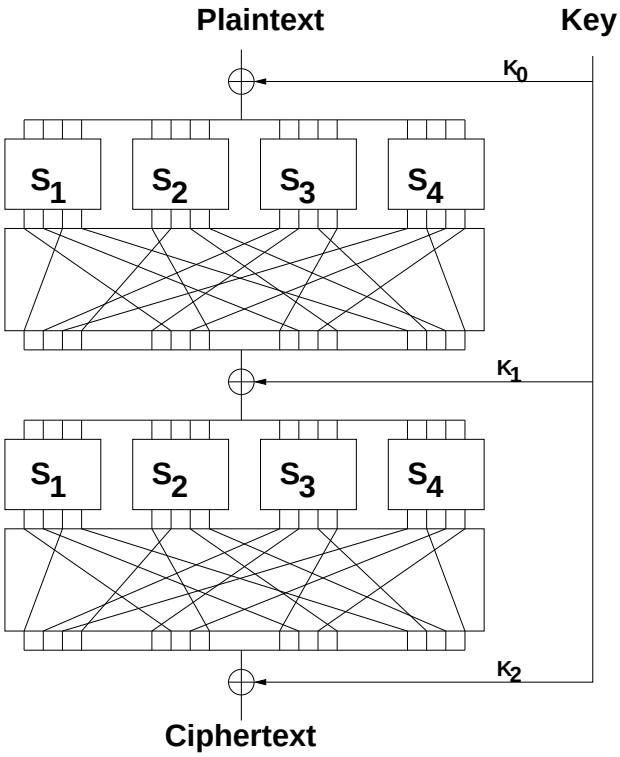

Figure 2.8: The idea behind a S-P network

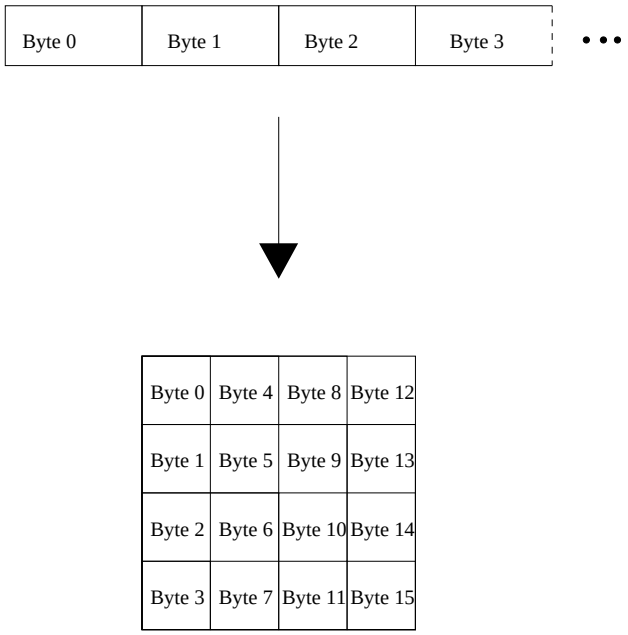

Figure 2.9: A 16 byte block input in the AES cipher

2.4.1 Encryption

The AES goes through a number of operations on each block that is encrypted. During these rounds there are four operations that is applied to each block. The block size in AES is set to be 128 bits, although the original algorithm Rijndael can support different block sizes. These blocks are divided into four columns of consisting of four byte size part giving the block a 4x4 matrix seen in figure 2.9

This matrix will be modified during encryption and is called "the state" to which different operations are done.

Add round key

The first operation is to add the key for the round. The key for the round is expanded from the key given by the user using a key expansion algorithm. This is slightly different depending on the key size, but

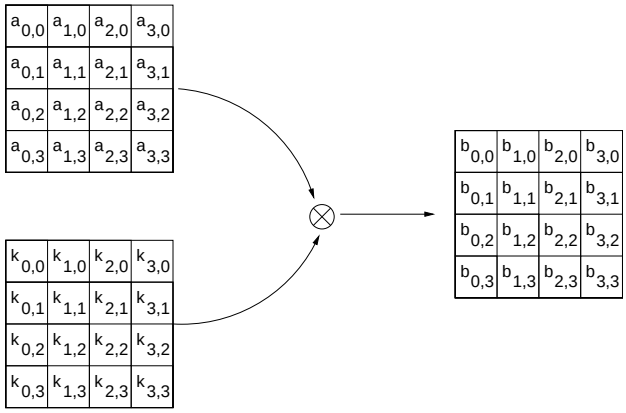

Figure 2.10: The Add round key step

for key size 128 it consists of Xor of key[n] and key[n-4] with an extra transformation of keys that are a multiple of 4. This transformation uses rotation of the previous keyword, the substitution LUT of the encryption, which will be explained later and Xor with the constant word of the round Rcon. The Add round key step is illustrated in figure 2.10.

Substitute bytes

The substitute bytes step is preformed using a LUT which is derived from the multiplicative inverse over GF(2^8) resulting in the matrix in table 2.4.1. This is used by taking each byte in the state in hexadecimal form and inputting the first character in the row field and the second character in the column field. If, say byte $a_{(2,1)}$ in figure 2.10, would be "e3", then using the matrix 2.4.1 would make the resulting byte $b_{(2,1)}$ after the s-box "11". This is done to all 16 bytes of the state.

Shift rows

The Shift-rows-step is a simple rotation of rows two through four of the state. The first row is not rotated, the second is rotated one byte step to the left. The shift rows is illustrated in figure 2.12

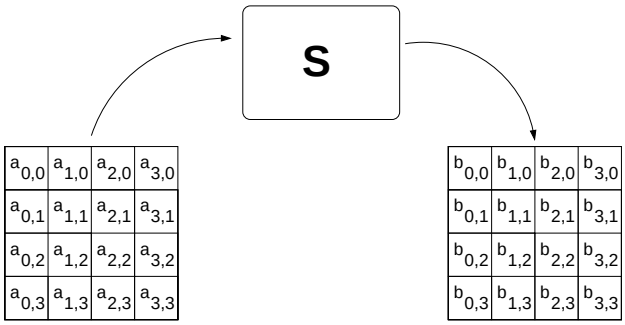

Figure 2.11: The Substitute bytes step

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 2.3: S-box LUT for encryption in AES

Figure 2.12: The Shift row step

Mix columns

The final operation is the mix columns operation. This step multiplies each column with a fixed polynomial c(x) modulo $x^4 + 1$. This can be viewed as a matrix multiplication described in equation 2.6, where the modulo calculation already has been taken care of.

$$\begin{bmatrix} b_{(2,0)} \\ b_{(2,1)} \\ b_{(2,2)} \\ b_{(2,3)} \end{bmatrix} = \begin{bmatrix} a_{(2,0)} \\ a_{(2,1)} \\ a_{(2,2)} \\ a_{(2,3)} \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \tag{2.6}$$

2.4.2 Decryption

To decrypt data encrypted by AES, a series of operations similar to the encryption steps are done. These operations can be seen in table 2.4.2. The operations are similar to the encryption operations and will not be covered in any depth in this thesis. The LUT used during decryption is another LUT than during encryption, the polynomial is a different one and the rotation is to the left instead of right. The only thing that is the same is the add key operation since it is just an Xor making the reverse operation the same as the encrypting Xor.

Step	Action	Description
Initialisation round	Add round key	Adds key to block
Rounds of decryption	Invshift rows Inv substitute bytes Add round key Inv mix columns	Rotate back rows of state Substitute bytes using LUT Each column is multiplied with a polynomial
Final Round	Substitute bytes Shift rows Add round key	

Table 2.4: Steps of operation during decryption in AES

Chapter 3

Method

This chapter will cover the ideas of the framework of the device and how the different tasks of encoding and decoding is put into separate blocks in the device.

3.1 Top hierarchy

The project was aimed at producing a device capable of encryption and decryption using the Phelix algorithm. The device had a target technology of a FPGA circuit from Altera, the Cyclone II. The connections specified was a 32 bit input bus, a 32 bit output bus, a clock input, some controlling signals and a synchronous reset, roughly sketched in figure 3.1

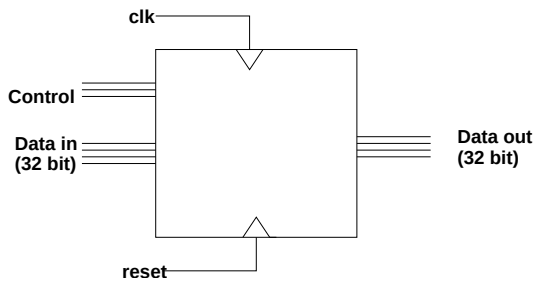

Figure 3.1: The original idea of the top hierarchy

The circuit was designed with a top-down approach by defining what parts was needed to make the circuit deliver ciphertext on the outgoing bus. The synchronous reset was changed early on to be an asynchronous reset to be able to reset the circuit at any time without having to wait for a clock pulse. At high clock frequencies the latency would not be very long, but if something went wrong for any reason, the asynchronous reset might be able to take care of the problem.

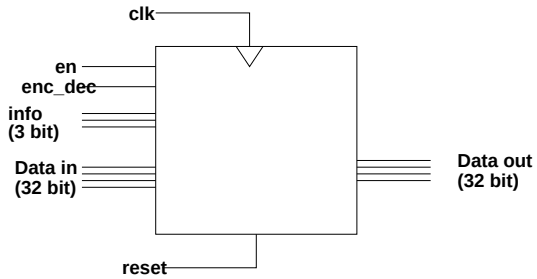

Figure 3.2: The top hierarchy of the design

The entire device was chosen not to have any external hand shaking signals and that the device attached to it would have to be able to deliver the correct data at some given times. The data delivery is simple, after enabling the device there are twelve clocks of key input, after that the device sets up for working. The exact setup process will be described later.

As seen in figure 3.2 the control signal bus of figure 3.1 has been broken down to contain an enable signal, an encryption/decryption signal and an info signal bus with the width of 3 bits. On top of this the clock signal and the asynchronous negative reset besides the 32 bit input and output buses.

3.2 Control signals

The control signals are there to make the co-processor being able to know what the circuit attached to it wants it to do.

Beginning with the enable signal, which is obviously there to turn the

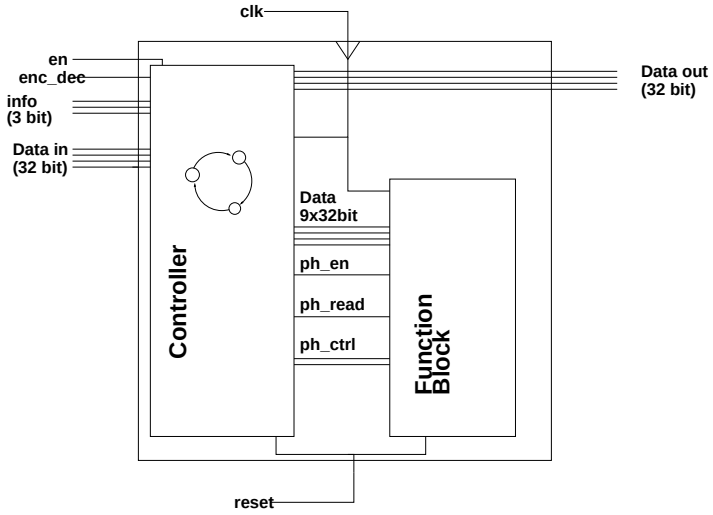

Figure 3.3: Inside of the circuit

encryption circuit on and off. However, as seen in chapter 4 it also gives some input to how the circuit handles some of the parts of the work. This was chosen to keep the controlling signals to a minimum.

The reset signal resets the circuit and sets it to an internal waitstate. This is used when the circuit is turned on to be certain that it is in the correct operating position, it is also used to change the key used.

The encryption/decryption signal tells the circuit to encrypt data if low and decrypt it if it is high. This gives the state machine instructions which path to choose after the initialisation.

Last there is the three-bit information bus which is used to deliver the amount of bytes in the last plaintext word, or ciphertext word. This information is used during the MAC generation.

3.3 Controller

To be able to control the circuit, a control mechanism was needed. The controller block holds a Finite State Machine (FSM) of Mealy type. The general structure of the inside of the circuit can be seen in figure 3.3

The controller is connected to the control signals of the top hierar-

chy to get the input of the attached unit. The controller is also connected to the main calculation part of the circuit. This was done to separate the controls from the crunching of numbers. The only thing left in the controller was the key generation and storage of working keys and nonce.

The connections from the controller to the calculation block part are nine 32 bit buses sending state words to and from the calculation block during operations. There are also control signals to set different modes of operation. These control signals are the one bit signals ph_en and ph_read. There is also the 2-bit signal ph_ctrl.

The en-signal (e.g. enable signal) turns the calculation block on to execute the task it has been given. The tasks are controlled by the ph_ctrl which sets the calculation block to different modes. The control signal also sets the function of the ph_read signal which can have different uses for different modes of operation. This will be explained in chapter 4

3.4 Calculation Block

The calculation block is the container of the Phelix algorithm. Besides the calculation parts, this block holds the FIFO for the feedback words. This calculation block takes care of all calculations where the block function described in chapter 2 and is switched between the small differences using the control signals from the controller.

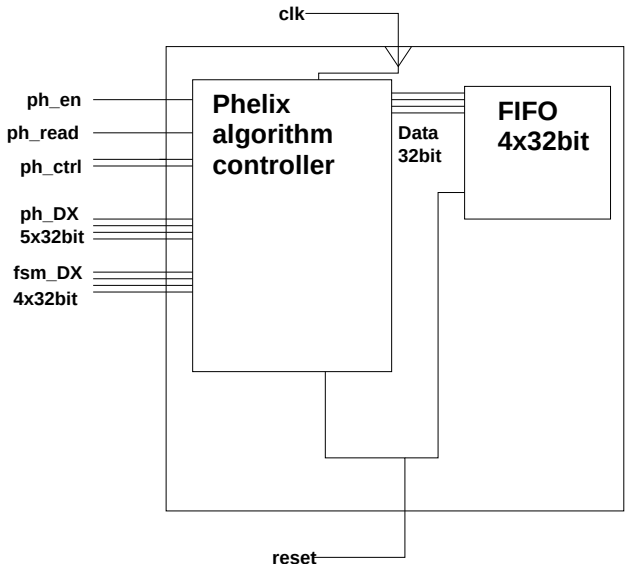

Figure 3.4: The Calculation block

Chapter 4

Implementation

The implementation of the ideas presented in chapter 3 will be described here. They will be divided into the different parts of the design seen in the figures describing the interior of the circuit.

4.1 Controller

The controller part of the circuit is the brain. In the controller resides the Mealy Finite State Machine that is controlling what happens at all times. The states that the FSM can take is described in figure 4.1 and will be described in detail later.

Alongside the FSM there are a series of registers that hold vital data during operations. These data are the working key of the algorithm and the nonce for the data being encrypted or decrypted at the moment. The nonce is discarded after each encryption or decryption however the key is left in the registers since it has been modified during the initialisation part of the encryption so to save some time during key changes only the nonce can be chosen to be changed. If the key needs to be changed as well, the device is reset and the entire process is restarted.

4.2 FSM and states

The FSM was made with nine states to control the flow of operations. These are *key in*, *key mix*, *init*, *encrypt*, *decrypt*, *mac*, *done wait*, *nonce in* and

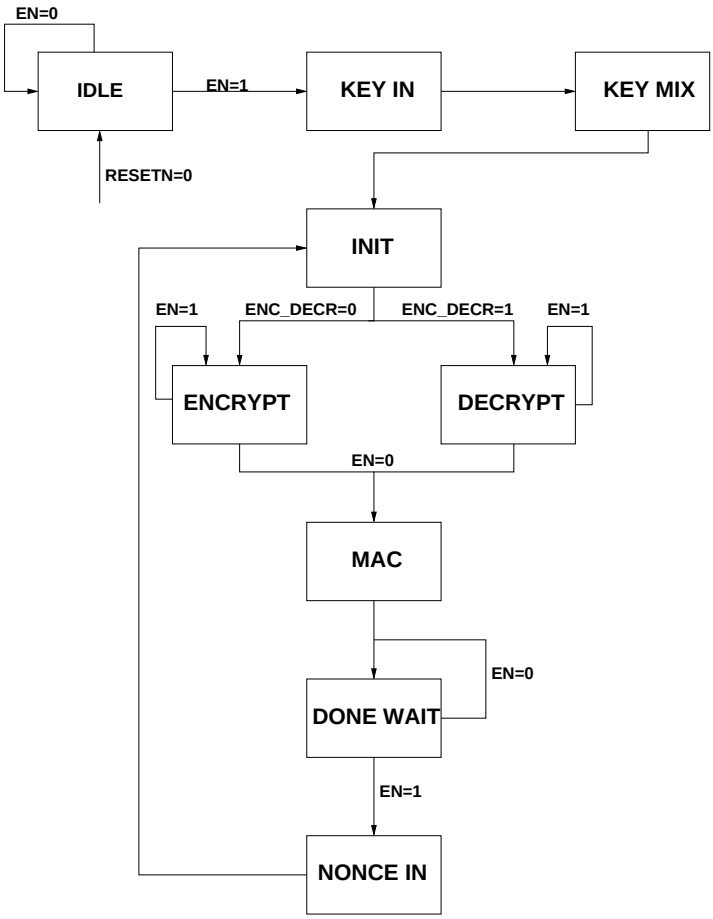

Figure 4.1: The states of the FSM

finally *idle*.

The states are as can be seen fairly coherent with the subsections of chapter 2. Each state has a quite narrow part of the algorithm to handle, all being synchronous except for the reset as mentioned earlier.

To be able to control the actions of the circuit there are also a number of counters present keeping track of how many words have been encrypted or decrypted and how many rounds of mixing and has been completed.

4.2.1 Idle state

This state is one of two waiting states. This is where the FSM ends up after a reset or if the circuit is turned off (en=0) in all states except encryption, decryption, mac and done wait. In the idle state all operations are halted.

4.2.2 Key In state

The key in state is the first to activate when the circuit is turned on. During this state the circuit will take data from the data in bus and transfer them to different data storage registers. This state is always 12 clock cycles long and it starts by taking in keys 0 through 7 and continues with nonce 0 through 3. After these 12 cycles the FSM automatically goes to the key mixing state.

4.2.3 Key Mix state

During the key mix state the eight raw keys are being mixed using the calculation block. The keys are sent in groups of four and then being mixed using the function described in section 2.3.1 where the device is programmed to set the fifth working word to 96 (0x60). The key mix state is designed with an internal stepper of eight steps, each step taking one clock cycle to complete and the first step is only run when the state is first addressed. This stepper is run eight times to complete the key mixing.

The steps of the key mix state can be seen in figure 4.2

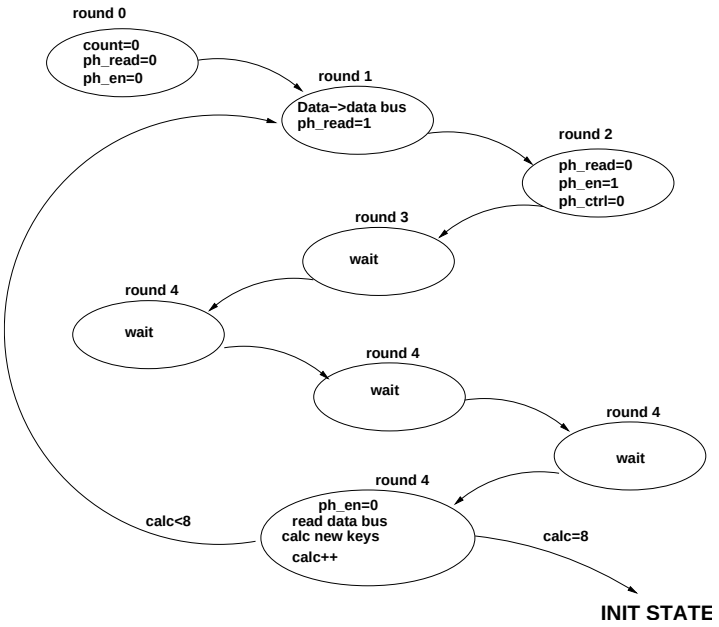

Figure 4.2: Steps in the key mix state

4.2.4 Init state

During the init state the working words are prepared to start encryption or decryption. In the same manner as in the key mix state the FSM steps through the steps of the state to ensure that the calculation block is ready to be used. The init state also has eight steps of which four are used during the init process, the other four are setup steps for the FSM and one exit step needed to do a clean exit from the state.

The four steps that are iterated during the eight rounds of initialisation are similar to the encryption/decryption steps except that there is no use of plaintext and the keystream is not used as explained in section 2.3.4. The key words $X_{(0,i)} and X_{(1,i)}$ for these rounds are supplied using block numbers $-8 \ldots -1$. During the first round of initialisation the nonce is expanded from four to eight words using the relation described in section 2.3.2 earlier.

After the eight rounds of initialisation the state machine switches state to encryption or decryption according to the input of the attached

device on the enc_decr signal and also sends the signal 0x01 to the data output to signal that the circuit is ready to receive data. This method was chosen to reduce the number of signal wires into and out from the circuit.

4.2.5 Encryption

Encryption is continuing the work from the initialisation state. The encryption state has five steps that it goes through during operation. The first four are the steps used to encrypt data and are iterated while the en-signal is high. As soon as it goes low, the machine goes to the MAC calculation state.

During the encryption phase data is received and sent every four clock cycles as it is ready from the calculation block.

The steps of encryption is as follows

1. At step one, data is sent to the function block. This contains the plaintext along with the two key words for the block being computed. The key words for each block being processed differs as seen in equation 2.4.

2. Wait

3. Wait

4. At step four the data is sent out on the data out bus and parts of the next rounds keywords are calculated. If the circuit is turned off, i.e. the encryption is over since the data has run out, some preparation for the MAC calculation is done here.

5. The fifth step is a transition step for the next state, the MAC calculation step. The function block is set to start the MAC calculation

4.2.6 Decryption

Decryption works in almost the exact same way as encryption with one exception. The FSM tells the calculation block that the data sent is ciphertext instead of plaintext by altering the ph_ctrl signal. The calcula-

tion block applies the key stream to the ciphertext before including it in the Phelix block.

4.2.7 MAC

After the last word of plaintext or ciphertext has been computed the next phase starts with eight rounds of MAC initialisation. The altering of working word 0 has already been taken care of during the last parts of the encryption or decryption states so there is only twelve rounds of MAC left, the last four rounds give data to present to the device connected to the circuit.

4.2.8 Done Wait

This is a state that was put into the design to be able to retain the working key and only provide a new nonce to encrypt more data. The advantage is that it will save some time in the initialisation phase of the circuit. When the enable signal is turned high the FMS enters the nonce in state and starts taking in a new nonce

4.2.9 Nonce In

This state takes in a new nonce as stated earlier. It is five clock cycles long to be able to take in four new nonce words as well as expand them into eight words to be able to start the initialisation of the circuit. When the five clock cycles have completed, the FSM turns to the initialisation state again.

4.3 Calculation Block

The calculation block contains the working parts to calculate the keystream. It also contains a FIFO for the feedback words. The function block has five data input buses of 32 bits, four data output buses of 32 bits, one control bus of 2 bits, one enable signal and a read signal.

The Block function (algorithm) described in section 2.3.1 has been divided further so it now contains four parts. This is done to reduce the

gate depth of the design with the intention to make it possible to run at higher clock frequencies. The effect this have on the design is that it takes four clock cycles to complete work on one word of key stream giving the block a throughput of one byte per clock cycle when encrypting. The placement of the "pipeline stages" can be seen in figure 4.3. The design is not pipelined in the normal sense since it needs the output of one turn of the block to start the next.

When the enable signal is high, the function block does work with the algorithm with slight modifications depending on the control signal provided.

0. Key Mix mode. The algorithm only does one turn and then waits for new input. When the read signal is high, the function block takes four data words from the data in and set the working words Z_{00} through Z_{30} to the data received. The working word Z_{40} is set to 96. After the four clock cycles that one turn needs, the four working words $Z_{00}^{(i+1)}$ through $Z_{30}^{(i+1)}$ are sent back to the FSM.

1. Initialisation mode. The algorithm works continuously and takes data from the input buses. The FIFO is being used. When the read signal is high, all five data in buses are used to set working words.

2. Encryption. The algorithm works continuously and takes data from the input buses. Cipher data is sent out. When the read signal is high, the working word Z_{00} is xored with the value $0x912d94f1$ to begin initialising the MAC. This mode is also used to calculate the MAC

3. Decryption. The algorithm works continuously and takes data from the input buses. Plaintext data is sent out. The cipher data is first decoded before it is used in the algorithm. When the read signal is high, the working word Z_{00} is xored with the value $0x912d94f1$ to begin initialising the MAC.

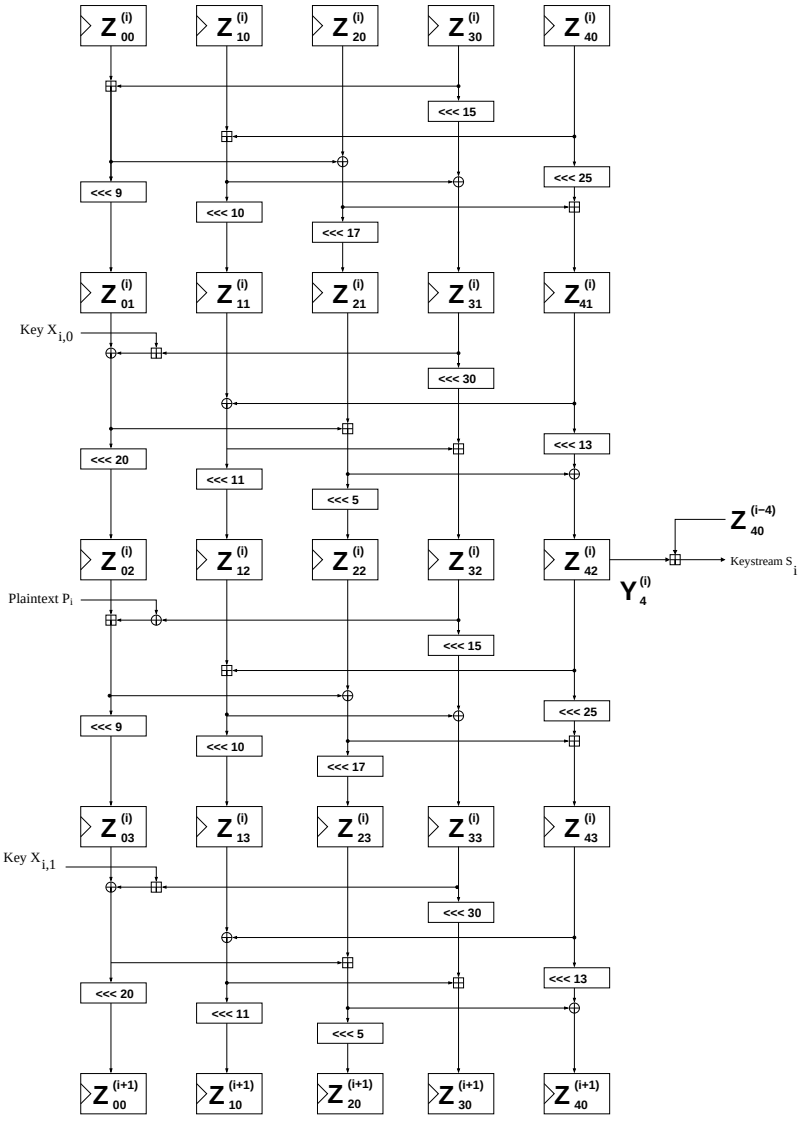

Figure 4.3: Register placement to divide the block function

4.3.1 FIFO

For every full turn of the block the state word Z_{40}^{i+1} (as defined in figure 4.3) is inserted in the FIFO. During operation, starting with the initialisation, a word Z_{40}^{i-4} is extracted from the FIFO to be added with the state word Z_{42}^{i} to create the keystream S_i. Since the FIFO delivers a word before it takes in a new word it is enough to have four 32-bit registers in the FIFO to reduce space.

4.3.2 Rotation

During operations there are a number of rotations carried out. To make the coding easier and to make the code more understandable, a function to rotate a 32 bit word a specific number of steps was included in the code. This is very simple. It takes in 32 bits and a 4 bit control signal and depending on the control signal it rotates the 32 bits provided and returns the result. For example, the code to make the working word W_{43}^{i} looks as follows

```
 W_43 = (W_22^(W_02+(W_32^(ph_D2^(Z_2+W_42)))))+
+(rott(W_42,1));
```

4.4 Reset

At any time during operation of the circuit it can be reset without having to wait for any clock flank. The asynchronous active low reset turns all registers of the circuit to zero as well as all signals turning everything off. When the reset signal is high again, the FSM goes to the idle state and waits for the enable signal to go high.

Chapter 5

Results

The purpose of this master thesis project was to make an implementation of the Phelix algorithm on a Cyclone II FPGA board and to compare it to software implementation of Phelix as well as hardware and software implementations of AES.

5.1 Software and Simulation

To make the simulation of the operation of the circuit, Modelsim SE 6.5 was used. With the software all operations was verified to make sure that all parts of the design communicated as was planned. The simulation was verified at 200 MHz however these results is only theoretical since no mapping to hardware was performed during these simulations.

The software used to do planning of the FPGA on hardware was Quartus II SJ web edition 9.1 from Altera. This software can be used to make the entire design from coding to verification and programming of the FPGA. During this thesis project it was only used to estimate the design on the target hardware which was chosen to be the Cyclone II EP2C5T144C6. This was chosen because the design needed 71 pins to connect the signals and that the design needed 4344 Logic Elements (LE) to fit the design on the FPGA when using the "optimise for area" option during synthesis. To make it a crypto core to a NIOS II, a larger circuit is needed, although the core fits on this circuit.

Part	Total LE	Register LE
FSM	2656	820
Calculation Block	1692	908
Total	4344	1728

Table 5.1: Usage of LE in the design

Model	Frequency	Throughput
Slow	80MHz	648 Mbps
Fast	183MHz	1464 Mbps

Table 5.2: Speed and throughput with different models

5.2 Speed and Area

The results from synthesis and timing analysis gives that the circuit would use 4344 LE on an Altera device. The unit LE is the smallest logic unit on the board and is the representation of how big a design is using Altera's devices. This area was derived using optimisation for area in the design tool. This was used because the speed advantage to use "balanced" or "speed" optimisation was very low and when using the area optimisation it was possible to use a smaller device.

Of the 4344 LE in the design, 1728 are used for registers. How the LE are used in different parts of the device is seen in table 5.2.

The speed of the design is also estimated using Quartus II 9.1 SJ Web edition. The speed results from this estimation is that the circuit can run at 81 MHz in the worst case scenario which is high temperature, slow silicon and low voltage. Using the best case scenario with fast silicon, low temperature and high voltage the circuit can run at 183 MHz. The results of these clocks frequencies can be seen in table 5.2

The simulation shows that the transfer of one of the keys from the FSM to the calculation block is where most time is lost. Considering that the other four ports from the FSM to the calculation block have less transfer time this is probably a placement or routing problem that can be made better. The throughput in table 5.2 is calculated with one byte per cycle which is the average output after the key has been set up

for operation given that the encryption or decryption takes 4 cycles per 32 bit word.

The total setup time for the device is 121 clock cycles. This is the cycle count to input 12 key words and do the key mixing using the Feistel network described in section 2.3.3. After the input key has been initialised to the working key once it does not have to be initialised again since it is sufficient to just change the nonce to keep the security in the cipher. The nonce change is done in 5 cycles.

Chapter 6

Discussion

The results just presented were to be compared with Phelix in software as well as AES on hardware and in software. To find results was not the only problem, one problem is to find a reasonable way of comparing them. There has been development of both hardware and software since AES first saw the morning light and the comparison of design in hardware is very much depending on the hardware used. The data presented in the papers published have not always all the data needed to make fair, or even any, comparisons. Mostly the data missing are setup times and what is implemented in the hardware. Several implementations of AES only show the device made for encrypting and since there are differences between encryption and decryption in AES that might make an unfair comparison.

The implementations of Phelix that are presented are all from the reports submitted to eSTREAM In table 6 the Phelix SW results are on a Pentium M taken from Whiting et all [23]. The Phelix HW results are on a Altera Cyclone device taken from Good et all [8] (where I used the full-round 160 bit design since I think it is the one most similar to this design). The AES SW results are on a Pentium 4 f12 processor taken from Bernstein et all [9]. The AES HW[8] results are on a Xilinx Spartan II, again from Good et all. Aditionally I have AES hw performance from Chang et all[12] aswell as from Grabowski and Youssef[18] since I referred to them in section 1.2. Both of these used a Virtex FPGA from Xilinx. I tried to use as few sources as possible to remove as many

Algorithm	HW/SW	Key Setup	Throughput	Speed	Size	Frequency
This thesis worst case	HW	121 cycles	648 Mb/s	1cpb	4344 LE	80MHz
This thesis best case	HW	121 cycles	1464 Mb/s	1cpb	4344 LE	183MHz
Phelix [23]	SW	$<$ 600 cycles	900Mb/s	15 cpb		1700MHz
Phelix [8]	HW		1440 Mb/s		1772 LE	
AES [9]	SW		1085 Mb/s	14 cpb		1900MHz
AES [8]	HW		2.32 Mb/s		500 LE	
AES [12]	HW		876 Mb/s		312 LE	306MHz
AES [18]	HW		480 Mb/s		14904 LE	56,3MHz

Table 6.1: Comparison between different implementations

sources of errors or unfairness as possible. There might be other comparisons done that state other figures than these but they give a pointer on performance and sizes. The speed is on a "large" data packet ($>$ 1024 bytes).

As mentioned, the papers presented does not always present all data such as ket setup. For Phelix HW, Gaj et all states that the key setup is 44 cycles in [17] and Gaj and Chodowiec states in [16] that the key setup for AES on hardware is close to zero. This can very well be true using the key expansion algorithm mentioned in section 2.4. Schneier et all claims that the key setup on AES on software is around 800 cycles in [21].

Looking at the size differences to the other implementations it should be mentioned that the smallest one, the AES [12], has three blocks of RAM used besides the LEs (slices) used. Comparing the three AES hardware implementations, we see that the last one uses a lot more logic space. That implementation has the possibility to use five different modes of operation. Compared to another implementation made by Lázaro et all [19] that not only uses the counter mode of operation but

also has a MAC feature it is about as large which would mean that if feedback modes are introduced to the AES hardware implementations they lose the size advantage that is obvious otherwise.

To compare Virtex and Spartan FGPAs to the Cyclone II that was my target platform is not completely fair. They use a different architecture and the comparison that was made is by the number of LUTs used.

With these observation as a background, the implementation presented in this thesis is a usable design and given that the Cyclone II comes in larger versions, it would be possible to include this device on a larger FPGA with more functions or to use it as a standalone device connected to another core. If the actual throughput is only a little higher than the worst case scenario, the design is able to work on data fed by a gigabit network. The algorithm also provides MAC functionality with little extra computing making the security a little better.

Chapter 7

Conclusion and Future Work

This thesis project was preformed to evaluate the implementation of the Phelix algorithm on a Cyclone II FPGA in comparison with software implementations of Phelix and AES.

7.1 Conclusion

The implementation of Phelix presented in this thesis shows some promise to be a usable device for some applications. Since the algorithm should not be used without a MAC comparison it will not be a good choice to use in real time data transfers such as telephones and similar equipment. However to transfer data which does not have to be decoded on the fly, it might be a usable algorithm and to use it on a cheap FPGA could have advantages to small organisations that do not have a very large budget for security. The MAC generation is on the other hand done using a very small amount of time compared to other algorithms, such as AES for example, which need to use more computing time to complete the MAC tag after the encryption is done. The size of the design is in comparison with other designs a little more than twice as large using an Altera Cyclone device. One possible reason for this is that it uses a quite large amount of registers in this design. The values these registers hold could either be calculated or perhaps put in the device's RAM banks.

Security

To compare Phelix to AES security wise is another story. Phelix has 128 bit key security which is the lowest key security provided by AES which has the possibility to use 192 bit keys and 256 bit keys. The key expansion algorithm of AES in 256-bit mode has been questioned and it was the key expansion that Biryukov et all [10] used in their attack.

There has been no successful attacks on Phelix more than the one by Wu and Preenel [24] that I have been able to find. This attack is a bit doubtful since it demands that the sender uses the same key-nonce-pair more than once. Even though the people behind Phelix (Whiting et all) say that it should not matter if the key-nonce pair is used more than once they also clearly points out that each such pair should never be reused. The lack of more presented attacks on Phelix could be a result of the removal of the algorithm from the sSTREAM project.

In July 2009 there was a paper submitted by Biryukov et all [10] claiming to have broken a 10 round 256-bit AES using related key-attack with 2^{70} time which Bruce Schneier (one of the authors of Phelix and a crypto analyst) states is "almost practical". This attack is not done on the standardised AES-256 which uses 14 rounds to encrypt a block. The attack also requires that the "attacker" has access to plaintext encrypted with multiple keys that are related. The attack uses a vulnerability in the key scheduler of the 256 bit version that is slightly different to the one used in the 128-bit version making the smaller key version actually more secure.

Looking only at security, it would seem that AES draws the longer straw. However, to make a fast encryption circuit of AES, one has to use a fast device since each round of encryption uses four different operations which cannot be done in parallel giving each block encrypted by AES-128 at least 40 cycles of operations, in other words 2.5 cycles per byte.

7.2 Future Work

Future work around this implementation could be concentrated around finding the critical paths in the design and investigate if the gate depth between registers can be decreased to make the register to register time go down. Another approach could be to find ways of making the routing more ideal to reduce timing issues on pure data transports.

Another thing to investigate could be what can be done with the keystream generator to decrease the size. As it is done now the full round is implemented in code and the compiler does the placement. If the code is modified to narrow the freedom of the compiler there might be space so save.

The device would need to be tested on real hardware to see how the timing estimates of Quartus II matches the real world device working in a environment where it might be used.

A fourth thing to look at could be area usage of the device and if reducing the amount of LE of the Cyclone II perhaps could improve the routing and even make it faster.

Bibliography

[1] Altera corporation. web page, last visited 2009-07-12. URL `http://www.altera.com/`.

[2] ECRYPT, European Network of Excellence for Cryptology. European project IST-2002-507932, available at http://www.ecrypt.eu.org/ecrypt1/, last visited 2010-02-04.

[3] ECRYPT II, European Network of Excellence for Cryptology II. European project ICT-2007-216676, available at http://www.ecrypt.eu.org/, last visited 2010-02-04.

[4] eSTREAM. web page, last visited 2009-08-02. URL `http://www.ecrypt.eu.org/stream/`.

[5] NESSIE, New European Schemes for Signatures, Integrity, and Encryption. European project IST-1999-12324, available at https://www.cosic.esat.kuleuven.be/nessie/, last visted 2010-02-04.

[6] Ieee standard verilog hardware description language. *IEEE Std 1364-2001*, pp. 0_1–856, 2001.

[7] Specification for the advanced encryption standard (aes). Federal Information Processing Standards Publication 197, 2001. URL `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[8] *"Review of stream cipher candidates from a low resource hardware perspective", presented at Symmetric Key Encryption Workshop, Aarhus, Denmark*, 2005 may. Available as eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016 `http://www.ecrypt.eu.org/stream`.

[9] Daniel J. Bernstein and Peter Schwabe. New aes software speed records. webpage, last visited 2009-08-07. URL `cr.yp.to/aes-speed/aesspeed-20080926.pdf`.

[10] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich and Adi Shamir. Key recovery attacks of practical complexity on aes variants with up to 10 rounds. Cryptology ePrint Archive, Report 2009/374, 2009. `http://eprint.iacr.org/`.

[11] Philippe Bulens, Kassem Kalach, Francois-Xavier Standaert and Jean Jacques Quisquater. Fpga implementations of estream phase-2 focus candidates with hardware profile. Available as eSTREAM, ECRYPT Stream Cipher Project, Report 2007 `http://www.ecrypt.eu.org/stream`.

[12] Chi-Jeng Chang, Chi-Wu Huang, Kuo-Huang Chang, Yi-Cheng Chen and Chung-Cheng Hsieh. *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, chapter High throughput 32-bit AES implementation, pp. 1806–1809. ISBN 978-1-4244-2341-5. Digital Object Identifier: 10.1109/APCCAS.2008.4746393.

[13] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002. ISBN 3-540-42580-2.

[14] Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey and Stefan Lucks. Helix, fast encryption and authentication in a single cryptographic primitive. web page, last visited 2009-07-22. URL `http://www.schneier.com/paper-helix.html`.

[15] Yongzhi Fu, Lin Hao, Xuejie Zhang and Rujin Yang. *Embedded Software and Systems, 2005. Second International Conference on*, chapter Design of an extremely high performance counter mode AES reconfigurable processor. 2005. ISBN 0-7695-2512-1. Digital Object Identifier: 10.1109/ICESS.2005.43.

[16] Kris Gaj and Pawel Chodowiec. Hardware performance of the aes finalists - survey and analysis of results. web page, last visited 2009-08-07. URL `ece.gmu.edu/crypto/AES_survey.pdf`.

[17] Kris Gaj, Gabriel Southern and Ramakrishna Bachimanch. Comparison of hardware performance of selected phase ii estream candidates. Available as eSTREAM, ECRYPT Stream Cipher Project, Report 2007/026 `http://www.ecrypt.eu.org/stream`.

[18] J.S. Grabowski and A. Youssef. *Microelectronics, 2007. ICM 2007. Internatonal Conference on*, chapter An FPGA implementation of AES with support for counter and feedback modes, pp. 39–42. 2007. ISBN 978-1-4244-1846-6. Digital Object Identifier: 10.1109/ICM.2007.4497657.

[19] Jesús Lázaro, Armando Astarloa, Unai Bidarte, Jaime Jiménez and Aitzol Zuloaga. *Reconfigurable Computing: Architectures, Tools and Applications*, volume Volume 5453/2009 of *Lecture Notes in Computer Science*, chapter AES-Galois Counter Mode Encryption/Decryption FPGA Core for Industrial and Residential Gigabit Ethernet Communications, pp. 312–317. Springer Berlin/Heidelberg, 2009 March. ISBN 978-9-642-00640-1. DOI: 10.1007/978-3-642-00641-8_34.

[20] Patrick Schaumont and Ingrid Verbauwhede. Hardware/software codesign for stream ciphers. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/016, 2007. Available at `http://www.ecrypt.eu.org/stream`, last visited 2010-02-08.

[21] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall and N. Ferguson. Performance comparison of the aes submissions. web page, last visited 2009-08-07. URL `http://www.schneier.com/paper-aes-performance.html`.

[22] Simon Singh. *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography*. Doubleday, New York, NY, USA, 1999. ISBN 0385495315.

[23] Doug Whiting, Bruce Schneier, Stefan Lucks and Frédéric Muller. Phelix, fast encryption and authentication in a single cryptographic primitive. web page, last visited 2009-07-22. URL `http://www.schneier.com/paper-phelix.html`.

[24] Hongjun Wu and Bart Preneel. *Fast Software Encryption*, volume Volume 4593/2007 of *Lecture Notes in Computer Science*, chapter Differential-Linear Attacks Against the Stream Cipher Phelix, pp. 87–100. Springer Berlin/Heidelberg, 2007 August. ISBN 978-3-540-74617-1.