Implementation and Evaluation of Espresso Stream Cipher in 65nm CMOS

RICHARD LOWENRUD
JACOB KIMBLAD
Implementation and Evaluation of Espresso Stream Cipher in 65nm CMOS

Richard Lowenrud; Jacob Kimblad

2016-11-27

Bachelor’s Thesis

Examiner
Elena Dubrova

Academic adviser
Sha Tao

TRITA-ICT-EX-2016:174

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Electronic and Embedded Systems
SE-164 40 Kista, Sweden
Abstract

With the upcoming 5G networks and expected growth of the Internet of Things (IoT), the demand for fast and reliable encryption algorithms will increase. As many systems might be time critical and run on internal power sources, the algorithms must be small, fast, energy efficient and have low latency. A new stream cipher called Espresso has been proposed to answer these demands, optimizing for several parameters unlike other stream ciphers such as Trivium and Grain. Espresso has previously been compared to the industry standard, Advanced Encryption Standard (AES), in a FPGA implementation and has shown promising results in terms of power usage but further testing needs to be done to gain knowledge about the cipher's characteristics. The purpose of this thesis is to implement and evaluate Espresso in 65nm CMOS technology and compare it to AES. Espresso is implemented in VHDL in several configurations, optimizing for size and latency. The implementations are then compared to AES in terms of area, throughput, energy efficiency and latency through simulation. This is done using the UMC 65nm CMOS library and Synopsys Design Vision. It is found that Espresso, implemented with 1 bit sequential loading of the key and IV, is 18.2x smaller, 3.2x faster, uses 9.4x less power and has 1.5x less latency than AES. When implemented with full parallel loading, Espresso still is 13.6x smaller, 3.2x faster, draws 7.1x less power while also having 3.2x lower latency than AES. Espressos energy efficiency can further be improved by applying low-power techniques although some techniques, like clock gating and power gating, have limited applicability due to of the nature of stream ciphers.

Keywords

Espresso, AES, IoT, Encryption, Power consumption, Low-power techniques
**Sammanfattning**

Med de kommande 5G nätverken och den förväntade tillväxten av Internet of Things (IoT) kommer efterfrågan på snabba och pålitliga krypteringsalgoritmer att öka. Eftersom många system kan vara tidskritiska och drivas av interna kraftkällor måste algoritmerna vara små, snabba, energieffektiva och ha låg latens. Ett nytt strömchiffer vid namn Espresso har föreslagits som ett svar på dessa krav och har optimiserats för flera parametrar till skillnad från andra strömchiffer såsom Trivium och Grain. Espresso har tidigare jämförts med branschstandarden, Advanced Encryption Standard (AES), i en FPGA implementation och visat lovande resultat för strömförbrukning men ytterligare tester måste utföras för att få kunskap om algoritmens egenskaper. Syftet med detta examensarbete är att implementera och utvärdera Espresso i 65nm CMOS teknologi och jämföra den med AES. Espresso implementeras i flera konfigurationer i VHDL som optimiserar för storlek och latens. Implementationerna jämförs sedan med AES i area, genomströmning, energieffektivitet och latens genom simulering. Detta görs med hjälp av UMC 65nm CMOS biblioteket och Synopsys Design Vision. Resultaten visar att Espresso implementerad med sekventiell laddning av nyckel och IV är 18.2x mindre, 3.2x snabbare, använder 9.4x mindre ström och har 1.5x mindre latens än AES. När Espresso implementeras med full parallel laddning är den fortfarande 13.6x mindre, 3.2x snabbare, drar 7.1x mindre ström men har samtidigt 3.2x lägre latens än AES. Espresso’s energieffektivitet kan förbättras ytterligare genom att applicera strömsparande tekniker, även om vissa tekniker såsom clock gating och power gating har begränsad användbarhet på grund av strömchiffers natur.

**Nyckelord**

Espresso, AES, IoT, Kryptering, Energiförbrukning, Strömsparande tekniker
Acknowledgments

We would like to thank our examiner, Professor Elena Dubrova, for suggesting this thesis and for answering all of our questions about the Espresso stream cipher. We would also like to thank her for taking the time to discuss our implementations of the algorithm and results with us. We would like to thank our supervisor, Postdoctoral Sha Tao, for guiding us through the process of writing our first thesis and for providing much of the material that was used in the thesis. We would also like to thank Professor Mark T Smith for granting us access to the KTH Mentorspace, where the majority of the work for this thesis was done.

Stockholm, June 2016
Richard Lowenrud & Jacob Kimblad
# Table of contents

Abstract ................................................................. 1
Keys words ........................................................................ 1
Sammanfattning .............................................................. iii
Nyckelord .......................................................................... iii
Acknowledgments ................................................................ v
Table of contents ........................................................... vii
List of Figures ................................................................. xi
List of Tables ....................................................................... xiii
List of acronyms and abbreviations ............................... xv

## 1 Introduction ......................................................... 1
### 1.1 Background ...................................................... 1
### 1.2 Problem definition ........................................... 1
### 1.3 Purpose .......................................................... 1
### 1.4 Goals ............................................................. 2
### 1.5 Research Methodology ..................................... 2
### 1.6 Delimitations .................................................. 2
### 1.7 Structure of the thesis ....................................... 3

## 2 Background ......................................................... 5
### 2.1 5G networks .................................................... 5
### 2.2 Internet of Things .............................................. 5
### 2.3 Block ciphers and Stream ciphers .................... 6
### 2.4 AES ............................................................... 6
### 2.5 Espresso ......................................................... 7
### 2.6 Low power techniques ..................................... 8
#### 2.6.1 Clock gating ................................................. 9
#### 2.6.2 Power gating ............................................... 9
#### 2.6.3 Gate level power optimizations .................. 9
#### 2.6.4 Multi-Threshold Logic ............................... 9
### 2.7 Related works ................................................ 9
#### 2.7.1 Espresso: A stream cipher for 5G wireless communication systems ................................................. 10
#### 2.7.2 Evaluating Power Consumption of IoT Ciphers in FPGA 10
### 2.8 Summary ......................................................... 11

## 3 Methodology .......................................................... 13
### 3.1 Research Process .............................................. 13
### 3.2 Research Paradigm .......................................... 14
### 3.3 Data Collection ................................................ 15
#### 3.3.1 Sampling method ........................................... 15
#### 3.3.2 Area estimation .......................................... 15
#### 3.3.3 Power estimation ..................................... 16
#### 3.3.4 Maximum clock rate ................................. 16
3.3.5 Throughput ................................................................. 16
3.3.6 Latency ........................................................................ 17

3.4 Applying Low-power Techniques ..................................... 17

3.5 Assessing reliability and validity of the data collected .......... 17
3.5.1 Reliability ..................................................................... 17
3.5.2 Validity ......................................................................... 18

3.6 Planned Data Analysis ....................................................... 18
3.6.1 Data Analysis Technique ................................................ 18
3.6.2 Software Tools .............................................................. 18

4 Implementation and evaluation of AES and Espresso .......... 19
4.1 Hardware design ............................................................ 19
4.2 Espresso ......................................................................... 19
4.3 AES .................................................................................. 20
4.4 Verification of the logic circuits .......................................... 20

4.5 Different implementations of Espresso and AES ............ 22
4.5.1 AES ............................................................................. 22
4.5.2 AES optimized ............................................................ 22
4.5.3 Espresso ....................................................................... 22
4.5.4 Espresso 1-bit sequential loading .................................. 22
4.5.5 Espresso 1-bit sequential loading optimized .................. 22
4.5.6 Espresso 2-bit parallel loading optimized ...................... 22
4.5.7 Espresso 256-bit parallel loading optimized ................... 23

4.6 Synthesis in Synopsys ....................................................... 23
4.6.1 Operating conditions and constraints .............................. 23
4.6.2 Mapping the design ...................................................... 24

4.7 Generating the results ...................................................... 25

4.8 Implementing and Evaluating Low-power Techniques .... 26

5 Analysis ............................................................................. 27
5.1 Major results ................................................................. 27
5.2 Reliability Analysis ........................................................ 27
5.3 Validity Analysis ............................................................ 28
5.4 Discussion ...................................................................... 28
5.4.1 AES ............................................................................. 28
5.4.2 AES optimized ............................................................ 28
5.4.3 Espresso ....................................................................... 28
5.4.4 Espresso 1-bit sequential loading .................................. 29
5.4.5 Espresso 1-bit sequential loading, optimized .................. 29
5.4.6 Espresso 2-bit parallel loading, optimized ...................... 29
5.4.7 Espresso 256-bit parallel loading, optimized ................... 29
5.4.8 Espresso optimization discussion .................................... 30
5.4.9 Low-power Technique Discussion ............................... 30
5.4.10 Loading phase ............................................................ 31
5.4.11 Initialization- and Encryption phase ............................. 31

6 Conclusions and Future work ............................................ 33
6.1 Conclusions ................................................................. 33
6.1.1 Espresso vs AES ........................................................ 33
6.1.2 Power saving ................................................................. 33
6.2 Limitations ...................................................................... 33
6.3 Future work .................................................................... 34
6.4 Reflections ...................................................................... 34
References ........................................................................... 35
Appendix A: Complete results ............................................ 37
Appendix B: Source code .................................................... 39
List of Figures

Figure 1: Output function of the Espresso stream cipher ......................... 8
Figure 2: Workflow for the implementation and evaluation of Espresso and AES ................................................................. 14
Figure 3: Loading logic for sequential loading of Espresso .................... 20
Figure 4: Compilation settings in Design Vision .................................... 24
Figure 5: Post-synthesis circuit needing to be recompiled ..................... 24
Figure 6: Optimizations made to Z6 .................................................... 25
Figure 7: Internal logic of MUX ......................................................... 30
Figure 8: Output function optimization .............................................. 30
Figure 9: Power gating example of MUX’es ......................................... 31
List of Tables

Table 1: Comparison of Espresso, Trivium and Grain stream ciphers .......... 10
Table 2: Cell sizes for UMC65nm library .......................................................... 15
Table 3: Major results table ........................................................................... 27
### List of acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog-To-Digital Converter</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>ASIC</td>
<td>Applied Specific Integrated Circuit</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher Block Chaining</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DUT</td>
<td>Device Under Test</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field programmable Gate Array</td>
</tr>
<tr>
<td>GE</td>
<td>Gate Equivalent</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IV</td>
<td>Initialization Vector</td>
</tr>
<tr>
<td>KS</td>
<td>Key Stream</td>
</tr>
<tr>
<td>MUX</td>
<td>Multiplexer</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NLFSR</td>
<td>Nonlinear Feedback Shift Register</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuit</td>
</tr>
</tbody>
</table>
1 Introduction

This chapter provides the general outlines for this thesis project by giving a short introduction to the area, defining the problem and giving the purpose and goals of the thesis. The research methodology is also presented along with the projects delimitations and is concluded with a description of the structure of the thesis.

1.1 Background

For each new generation of mobile communication networks, the specifications get updated to meet the evolving technological requirements, set by society and other technological advances. In the upcoming 5th generation networks, one of the major focuses is on the Internet of Things. With the growth of IoT, more devices will be connected at once, sending more data than ever before. Because of this, we are expected to see data rates increase by a factor of 1000 [1]. Another aspect of IoT is that the type of devices that are connected will vary more and bring with them new requirements and limiting factors such as internal, limited power sources, limited resources in terms of size and more. As the development of these technologies progresses, we will therefore not only see increased demands on data rates but also power efficiency while still requiring data to be sent securely. This makes it highly relevant to develop safe encryption technologies that are fast enough to handle the increased data rates while also being small and energy efficient.

Currently, one of the most widely used encryption algorithm for wireless communication is the industry standard, Advanced Encryption Standard (AES). As AES might not meet the evolving requirements of the new 5G networks, it is crucial to compare it to other ciphers in order to find out if better alternatives exist. Many attempts have been made to find more effective ciphers. These attempts often focus on only optimizing either size or speed, resulting in failure to meet the mentioned requirements for different 5G applications. A stream cipher called Espresso has been proposed as an alternative to today’s encryption methods, trying to optimize for several of the required parameters [2].

1.2 Problem definition

With the increasing demands on data rates, reliability and size, today’s encryption algorithms may not be good enough to keep up with the upcoming 5G networks. Even if they are, the requirements for energy efficiency will also be increasing as they might need to be implemented in mobile devices with limited, internal power sources. The new stream cipher Espresso have been proposed to solve many of these issues, however not much work has been done to see if Espresso can deliver. More in depth testing needs to be carried out in order to determine its viability as a new standard. For Espresso to be considered as a new standard, it is relevant to make fair comparisons against the Industry standard AES. This has been done to some extent [3], but need to be evaluated further.

1.3 Purpose

The purpose of the thesis is study characteristics of the stream cipher Espresso and check its viability as a new industry standard.
1.4 Goals

The goal of this thesis project is to implement and evaluate the Espresso stream cipher to make a fair comparison against the Industry standard AES. In particular, this work will address the following aspects:

- To implement the Espresso stream cipher in a standard-cell 65nm CMOS technology
- To evaluate its area, throughput, power consumption and latency and compare the results with the industry standard block cipher AES.
- To explore state-of-the-art low-power digital design-techniques in order to improve the energy efficiency of Espresso.

1.5 Research Methodology

This project will draw conclusions from positivism as a consequence of using a deductive research approach, supported by data gathered through quantitative methods. To fully understand how Espresso compares to AES, the algorithms will be implemented in VHDL and analyzed in order to collect concrete, empirical data. The algorithms will be simulated on gate level to measure the performance and characteristics of the algorithms in terms of area, throughput, power consumption and latency. By using these methods, concrete results and relevant data can be produced. With the gathered data, a fair comparison can be made and conclusions can be drawn for Espresso’s viability as a new industry standard.

In order to make a fair and truthful comparison between the algorithms, empirical data is needed. This type of data can only be obtained by conducting experiments and measurements on the algorithms. Other means of data collection would be insufficient and produce data based on subjective speculations. Therefore, a quantitative research approach is our only option, as other research methods won’t provide us with the necessary data.

1.6 Delimitations

The project will only focus on comparing the Espresso stream cipher with the Advanced Encryption Standard. No other encryptions algorithms will be compared in this thesis. AES can be implemented in many different ways, optimizing for different aspects such as area, throughput, energy efficiency and latency [4]. Only one version of AES will be used during the comparison.

The Espresso cipher is able to output a maximum of 4 bits per clock cycle for faster encryption/decryption. This is done through parallelization which, however, imposes an extra cost of logic gates, resulting in larger area. Parallelization will not be covered in the thesis.

To get Espresso running, the key and IV need to be loaded into the 256 bit shift register. This can be done sequentially, in parallel or a combination of the both. In this project, only 1 bit sequential loading, 2 bit parallel loading and full 256 bit parallel loading will be implemented. Although other combinations will not be implemented, they will still be discussed.

There exist many low-power digital design techniques and many of them might be able to improve the energy efficiency of the Espresso stream cipher [5]. Because of this, only Clock gating, Power gating, Gate level power optimization and Multi-threshold logic will be evaluated and no other low-power techniques.
1.7 Structure of the thesis

Chapter 2 presents relevant background information for the upcoming 5G networks, the two encryption algorithms compared in this thesis, previous work and the low-power digital design techniques that will be discussed later on in the thesis. Chapter 3 provides a description of the research methodology used in the thesis and how the project is planned to be carried out. This description includes planned data collection and data analysis together with planned reliability and validity analysis. Chapter 4 describes the actual work carried out during the thesis project and the different steps taken to ensure the validity of the gathered data. Chapter 5 presents the results of the work together with an analysis of the reliability and validity and is concluded with a discussion of the results. Chapter 6 presents the final conclusions and reflections together with what limited the thesis and what can be done in future work. Chapter 7 provides a list of references used in the thesis. Appendix A includes a complete table of the results obtained in the thesis. Appendix B includes the source code for the different implementations of the ciphers. Appendix C includes the reports generated from Design Vision.
2 Background

This chapter describes the fundamentals of two types of encryption algorithms, block cipher and stream cipher. Since the Espresso might be considered for use in the upcoming 5G network, background for 5G and IoT is given. The chapter also describes the encryption algorithms to be compared and the low-power techniques discussed in the thesis are described. The chapter also summarizes the related works of “Espresso: A stream cipher for 5g wireless communication systems” and “Evaluating power consumption of IoT ciphers in FPGA.

2.1 5G networks

With ever increasing demands on mobile data, the expectations and requirements for the next generation networks are huge. The 5G network is the upcoming standard for mobile communication, meant to replace the current 4G LTE network and is planned to be deployed by the year 2020. In 2012, the project Metis2020 was started by the European Commission as an integrated project under the Seventh Framework Programme for research and development. The project aimed to lay the foundation for 5G by defining the challenges of the future, giving a starting point for the development. Metis2020 was ended in 2015 after successfully reaching its goal and has since been followed by the Metis-II project, focusing on developing the overall 5G network design as well as regulatory and standardization bodies [6][7]. The requirements for 5G are summarized into the following points. The goal is to achieve these requirements without introducing extra costs.

- 1000x higher data volumes
- 10-100x more connected devices
- 10-100x typical end-user data rates
- 10x longer battery life for low-power devices
- 5x lower end-to-end latency

2.2 Internet of Things

When discussing the Internet of Things, IoT for short, the definition is somewhat diffuse and tends to differ. The definition is often biased and depends on what aspects are deemed to be the most relevant from the perspective of the proponent entity. When looking at the most basic definition derived from its name, the IoT is about the interconnection of things. It is a global network, concerned with the accessing of data and identification of connected devices, varying in type and complexity. It is the variation and number of devices that set the IoT apart from all previous networks. It’s estimated to consist of 28 billion devices connected to the internet by the year 2021 [8][9].

The various devices impose different requirements and limitations that must be taken into consideration when developing the protocols and standards to be used in the IoT. The amount of data that is going to be sent by different devices may vary from several Gigabit per second to just a few Kilobit per month. The devices may have different area constraints, having to be implemented within a limited die area. Certain devices are also going to be powered by limited internal power sources and may be installed in hard to reach places, making replacement of the batteries impractical and sometimes even impossible. It is therefore important that the IoT-devices are able to meet all of these requirements while still being able to send security-critical data.
2.3 Block ciphers and Stream ciphers

Encryption algorithms can be divided into several categories depending on their various characteristics. Encryption algorithms usually use a key to encrypt/decrypt the provided plaintext although the type of key used can differ. The key can be symmetric, meaning that the key used for encryption and decryption is the same and thus must be kept secret. The other alternative is that the key is asymmetric, meaning that different keys are used for encryption and decryption. When using different keys, only the key used for decryption is secret while the key used for encryption is not and is therefore public [10]. One problem with asymmetric keys is that because the encryption key is public, you can’t be sure of who encrypted the message and thus, additional systems must often be put into place.

Encryption algorithms using symmetric keys can further be divided into two main groups, block ciphers and stream ciphers, encrypting data in two very different ways. Although block and stream ciphers are very different, block ciphers can be implemented as stream ciphers and vice versa. Block ciphers operates on fixed size blocks of plaintext with a fixed transformation. This means that a specific block of plaintext and key always produces the same block of ciphertext. The exception is when the plaintext block is manipulated before being encrypted. These manipulations are referred to as modes of operations [11]. Because they operate on blocks of data, block ciphers are often easier to implement in software rather than hardware. Stream ciphers instead produce a pseudo-random stream of bits that is XORed with the plaintext to produce the ciphertext. Because of this, two identical blocks of plaintext encrypted after each other will produce two different blocks of ciphertext [12]. One advantage of stream ciphers is that they are easier to analyze mathematically compared to block ciphers. They have also been proven to be generally smaller, faster and easier to implement efficiently in silicon, making them suitable for hardware encryption [10].

2.4 AES

The Advanced Encryption Standard is an encryption standard established in 2001 by the National Institute of Standards and Technology (NIST) in the US to replace its predecessor, the Data Encryption Standard (DES). The AES is a 128-bit block cipher based on the Rijndael block cipher, meaning that it encrypts 128-bit blocks of plaintext to ciphertext all at once using a key of size 128, 192 or 256 bits. A longer key increases the security but also increases the time required to encrypt and decrypt data.

The algorithm consists of two major phases, the Key expansion phase and the Encryption phase. The Key expansion phase takes the provided cipher key and derivatives round keys from it that are later used in the transformation rounds of the plaintext to ciphertext. Each transformation round requires a separate round key and the number of rounds is determined by the length of the provided cipher key. This initial phase only needs to be performed once per cipher key and initiates the algorithm. The next phase is the encryption where the plaintext is encrypted by one initial round of AddRoundKey followed by 10, 12 or 14 rounds of encryption depending on the cipher key used. Each round performs the operations SubBytes, ShiftRows, MixColumns and AddRoundKey with the exception of the final round that does not include the MixColumn transformation [13].

To increase the security of the algorithm when encrypting more than one block of plaintext, the algorithm can be used in a Cipher Block Chaining (CBC) mode of operation. In this mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. By doing this, you avoid exposing information of the encrypted data or cipher key when sending similar blocks of
information. Several similar modes of operation exist to solve this specific problem, most of them using the result from previous rounds of encryptions to alter the current one [11].

2.5 Espresso

Espresso is a stream cipher which has been developed to accommodate the high demands of 5G wireless communications. Current stream ciphers often make a choice of either optimizing for speed or size. Espresso has instead been developed taking both of these parameters into consideration while still being secure against different types of attacks. The cipher consists of a 256-bit nonlinear feedback shift register and a 20-variable nonlinear output function. The NLFSR is in Galois configuration which makes it have short feedback functions and therefore short propagation delay. This results in the cipher being able to operate at a high clock rate and gives it a high maximum throughput. Because of the large number of feedback functions used, the cipher is limited to a maximum degree of parallelization of 4. To analyze the cipher’s security, the feedback function is transformed into an equivalent Fibonacci configuration, which is more easily analyzed. Hence the NLFSR combines the advantages of both the Galois configuration and Fibonacci configuration. The output function of the cipher is also pipelined, delaying it two clock cycles and increasing the latency. This is done to shorten the propagation delay even further and increase the throughput of the design. The cipher contains a total of 14 feedback functions which are configured as follows.

\[
g_{255}(x) = x_0 \oplus x_4 x_7 z_0 \\
g_{251}(x) = x_{252} \oplus x_{42} x_{83} \oplus x_8 \\
g_{247}(x) = x_{248} \oplus x_{44} x_{102} \oplus x_{40} \\
g_{243}(x) = x_{244} \oplus x_{43} x_{118} \oplus x_{103} \\
g_{239}(x) = x_{240} \oplus x_{46} x_{141} \oplus x_{117} \\
g_{235}(x) = x_{236} \oplus ((x_{67} x_{90})' + (x_{110} x_{137})')' \\
g_{231}(x) = x_{232} \oplus x_{50} x_{159} \oplus x_{189} \\
g_{217}(x) = x_{218} \oplus x_3 x_{32} \\
g_{213}(x) = x_{214} \oplus x_4 x_{45} \\
g_{209}(x) = x_{210} \oplus x_6 x_{64} \\
g_{205}(x) = x_{206} \oplus x_5 x_{80} \\
g_{201}(x) = x_{202} \oplus x_8 x_{103} \\
g_{197}(x) = x_{198} \oplus ((x_{29} x_{52})' + (x_{72} x_{90})')' \\
g_{193}(x) = x_{194} \oplus x_{12} x_{121}
\]

Where \(g_n(x)\) represents the next state of flip-flop \(y\) depending on current state of flip-flops \(x\). The XOR operand is represented by \(\oplus\), the NOT operand is represented by \(\prime\), the OR operand is represented by +, and the AND operator is represented by \(\cdot\). It’s also important to know that during the initialization phase of the cipher, functions \(g_{255}(x)\) and \(g_{217}(x)\) are modified by XOR’ing the existing functions with the output from the output function. The modified functions are shown below.

\[
g_{255}(x) = x_0 \oplus x_4 x_7 z(x) \\
g_{217}(x) = x_{218} \oplus x_3 x_{32} \oplus z(x)
\]

The output function of the cipher is configured as follows:

\[
z_1(x) = x_{80} \oplus x_{99} \oplus x_{137} \oplus x_{227} \\
z_2(x) = x_{222} \oplus x_{187} \oplus x_{243} x_{217} \\
z_3(x) = x_{247} x_{231} \oplus x_{213} x_{235} \\
z_4(x) = x_{255} x_{251} \oplus x_{181} x_{239}
\]
\[ z_5(x) = x_{174}x_{44} \oplus x_{164}x_{29} \]
\[ z_6(x) = x_{255}x_{247}x_{243}x_{213}x_{181}x_{174} \]
\[ z_7(x) = z_1(x) \oplus z_2(x) \oplus z_3(x) \oplus z_4(x) \]
\[ z_9(x) = z_5(x) \oplus z_6(x) \]
\[ z(x) = z_7(x) \oplus z_9(x) \]

The output function is also visualized in Figure 1 below.

To initialize Espresso, a 128-bit key, a 96-bit initialization value and a predetermined sequence of ones and a zero are loaded into the NLFSR. The loading can be done either sequentially, in parallel or a combination between the both. Which method is chosen depends on the requirements for the specific implementation of the cipher. Parallel loading reduces the number of clock cycles needed before the keystream can be produced but increases the amount of logic needed. The opposite is true for sequential loading. After the loading is complete, the circuit is clocked 256 times for the initialization phase. During this phase, the output is feed back into the register at two points as mentioned earlier. After the initialization phase, the circuit is clocked three more times before the keystream is produced due to the extra flip flops in the output function. The keystream from the cipher is then bitwise XORed with the plaintext to produce the ciphertext [2].

2.6 Low power techniques

Throughout the years, several methods and design techniques have been proposed to save energy in digital circuitry. These techniques effectively reduce the energy consumption without affecting the
overall performance of the circuit by adding very few or no additional gates. This subchapter goes through some of these techniques and describe how they work. Other techniques also exist but will not be discussed in this thesis.

2.6.1 Clock gating

The clock distribution network often accounts for a big part of the systems overall power usage. This makes intuitive sense, since there often are many clock buffers with high drive strength in this network and they have the highest toggle rate in the whole system. Clock gating therefore disables the clock for unused parts of the system, disabling the switching for both buffers and flip-flops. This can make quite large power savings for certain designs while others are unaffected because no parts of the design can be temporarily disabled [5].

2.6.2 Power gating

Power Gating is similar to Clock Gating but instead disable the supply voltage to unused parts of the logic. This is either done by adding header cells, disconnecting the supply voltage or with footer cells, disconnecting the ground. Power gating suffers from the same disadvantage as Clock gating, in that it can only be applied if parts of the circuit is at times unused [14].

2.6.3 Gate level power optimizations

There exist many ways to optimize the power consumption of circuits by re-routing or replacing logic depending on net activity and input impedance of the gates used. When using multiple input gates, there is often a significant difference between the input impedance of the gates pins. It is therefore beneficial to pair high activity nets with low impedance input pins and low activity nets with high impedance input pins. This way, the high impedance pins that require more power are used more seldom and result in lower power consumption. The same effect can be achieved by replacing gates in high activity nets with gates that have low input impedance. For example, a AND-gate followed by a NOR-gate can be replaced with a AND-OR-gate followed by an inverter. This makes the high activity net (the output of the AND-gate) internal to the cell and thus the AND-gate drives a much smaller capacitance, reducing the power consumption [5].

2.6.4 Multi-Threshold Logic

Sub-threshold leakage current in cells depend exponentially on the cells voltage-threshold while the delay of the cell has a much weaker dependence. It is therefore common for gate libraries, used for synthesis of circuits, to contain multiple version of each cell with different voltage-thresholds and to use these for energy saving purposes. The synthesis can then be done initially with the high performance, leaky cells. Later, optimizations can replace cells in non-critical paths with slower cells with higher thresholds that don’t have as high leakage. Because the replaced cells are not in the critical path, their lower speed won’t affect the overall performance of the system [5].

2.7 Related works

This subchapter gives an overview of the most relevant works. This is to give more context to what has been done before and what still needs to be done within this area.
2.7.1 Espresso: A stream cipher for 5G wireless communication systems

Elena Dubrova and Martin Hell have proposed a new encryption algorithm, called Espresso, which is described in “Espresso: A stream cipher for 5G wireless communication systems” [2]. Their paper goes into the details of circuit design and explains different design choices. Many custom design optimizations have been proposed to ensure a small area and low propagation delay. It is therefore necessary to study the work thoroughly so that the cipher can be implemented exactly as it was intended in order to give a precise analysis of its characteristics. The presented encryption algorithm is a 256-bit NLFSR in Galois configuration with a 20-variable nonlinear pipelined output function.

Many existing stream ciphers are vulnerable against attacks, such as the one used in GSM that was, because of its weakness, replaced with a block cipher. Because of this, the confidence in stream ciphers has been lowered. Therefore, being able to analyze the security of Espresso was one of the main concerns. To enable this analysis, Espressos Galois configured NLFSR was transformed into an equivalent NLFSR in the Fibonacci configuration. Since there are many cryptanalytic methods for the Fibonacci configuration, these can also be utilized to analyze Espresso. Because of the large propagation delay of the Fibonacci configuration, it is only used to analyze the security of the cipher. The effects of different attacks against the cipher have been explained as how effective they would be. These attacks include linear approximations, algebraic attacks, time-memory-data tradeoff attacks, chosen IV attacks and differential attacks as well as analyzing the probability of weak keys.

The cipher is compared in terms of area, throughput and latency to the existing stream ciphers Trivium and Grain for parallelization up to 4 bits per cycle. The result of the comparison for parallelization 1 is presented in the table below.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Area (GE)</th>
<th>Throughput (Gbit/s)</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espresso</td>
<td>1497</td>
<td>2.22</td>
<td>232</td>
</tr>
<tr>
<td>Grain</td>
<td>1446</td>
<td>1.30</td>
<td>296</td>
</tr>
<tr>
<td>Trivium</td>
<td>1513</td>
<td>1.86</td>
<td>663</td>
</tr>
</tbody>
</table>

Table 1: Comparison of Espresso, Trivium and Grain stream ciphers

2.7.2 Evaluating Power Consumption of IoT Ciphers in FPGA

The most relevant work to this thesis is “Evaluating Power Consumption of IoT Ciphers in FPGA” [3]. In this work, the Espresso stream cipher and AES block cipher were implemented on a FPGA and compared in terms of power consumption. The FPGA board used was a Xilinx ML605 evaluation board, which provide means to easily perform power measurements on implemented designs using an internal ADC or an external oscilloscope. To enable the testing, a surrounding framework was set up which monitored and handled the testing of the DUT by implementing a finite state machine. Data was generated in the framework and transferred into the DUT together with the encryption key. The framework also monitored the output from the DUT, verifying the encrypted data for correctness, making sure the encryption was successful. The framework was made dynamic so that both the stream cipher and block cipher could be tested within the same framework, making sure testing conditions were equal and fair for both ciphers.

To make the comparison between the ciphers, the tests were run in three different configurations. One with AES implemented on the FPGA, on with Espresso implemented and one with a dummy function without encryption. The last test was to determine a baseline for the power consumed by the
test framework. The baseline was then excluded from the total power readings of the other tests, leaving only the consumption from the ciphers. AES was not implemented in this report but instead an existing version of the block cipher was used. The implementation used was chosen on the basis of being lightweight and small, giving a fair comparison to the Espresso stream cipher. The test framework, including the ciphers, were instantiated 30 times each on the FPGA in each of the tests in order to create a higher power consumption allowing more accurate power readings. The results of the simulations showed that AES consumed 140 times more power than Espresso giving the stream cipher a energy saving of 99.28%. When measuring the power consumption on the FPGA, AES used 3 times more power than Espresso. These results were in line with the simulations despite the difference between the ciphers being smaller because of the extra logic required for stream cipher to be used in the same test framework as the block cipher.

2.8 Summary

Cryptography has long been an important part of telecommunication and many different encryption algorithms has been invented to allow secure communication. The algorithms encrypt plaintext into ciphertext by using keys and different methods of transformation and are divided into categories depending on the type of key and transformation that is used. Among these categories are block ciphers and stream ciphers, both using the same private key for encryption and decryption. The two types are commonly used in different applications, having both strengths and weaknesses, with stream ciphers generally being both smaller and faster.

In the future of telecommunication, the need for mobile data and the number of connected devices will increase significantly. As a result, new technologies emerge to answer the increasing demands. One of these technologies, currently in development, is the 5G networks. Together with the Internet of Things, 5G will require faster and more versatile encryption algorithms and it’s uncertain if current algorithms fulfill the requirements. A stream cipher called Espresso has recently been proposed as a viable option and has shown promising results when compared to both other stream ciphers and the industry standard block cipher, AES. The algorithm has been designed, taking both size and propagation delay into consideration. More thorough comparison and analysis still needs to be made to evaluate further its characteristics and viability.

Depending on how circuits are designed, many optimizations can be done by applying low-power techniques, increasing their energy efficiency. Such techniques can drastically decrease the power consumption, adding only a few or no extra gates to the design. With these optimizations, the circuits can become even more suitable for energy restricted implementations as in those expected in the 5G networks.
3 Methodology

The purpose of this chapter is to provide an overview of the methodology and methods used during this thesis. The research process used in the thesis is visualized and described along with the research paradigm, planned data collection methods, reliability and validity of the collected data and planned analysis of the data.

3.1 Research Process

The research process of this thesis is going to be carried out in several steps, utilizing several programs and methods along the way. The first step in the process is writing the VHDL code for Espresso. This can be done using any simple text editor but to ensure correctness of the code, a compiler is needed. For this step, Altera Quartus II is chosen. This step is not necessary for AES, since the code for the cipher is retrieved from an online source, and is therefore skipped in the workflow for AES. When the code has been written and compiled it will be both visualized as a gate network and simulated in ModelSim. This acts as means of verification, ensuring that the implementation produces the correct result and use the correct gates. This process is carried out until the circuits has been verified to behave as intended. However, since AES is already working when retrieved, it is not necessary to change any of the VHDL for it. When the code has been verified, the implementations are transferred to Synopsys Design Vision. Here the implementations are analyzed and compiled in order to replace all gates with standard 65nm CMOS-gates. The implementations are then once again verified visually to ensure that the design are not changed from how they are intended to be. When verified the circuits are optimized and analyzed to get information about the size, throughput, power consumption and latency. The whole workflow is visualized in the image below.
3.2 Research Paradigm

For this thesis project, positivism and a quantitative research approach are adopted. With the ontological belief that one objective reality exists that can be measured, experiments can be designed to obtain objective, empirical data. Because the truth is independent of the observer and instruments used for measurements, anyone will reach the same results if the experiments are conducted properly. This quantitative and experimental approach is also the only way to properly gather the data needed since other means of data collection might become distorted by subjective thoughts and reasoning. It is therefore important to design the experiments in a way so that they are not affected by outside factors in order to get accurate results. With the data collected through experiments and measurements, a conclusion can be drawn through deductive reasoning. Again, because of the independent nature of the measured reality, we can say with certainty what the results are. This differs from inductive reasoning based on qualitative research where results only can be express in probability. This is because the results can be biased and distorted by outside factors such as subjective reasoning [15].
3.3 Data Collection

This subchapter describes how the data is going to be collected. The programs used for this process are described and how different parts of the data collecting process works.

3.3.1 Sampling method

All the data of the ciphers characteristics are going to be collected using simulation in Design Vision from Synopsys, Inc. By using library files containing information about the 65nm CMOS gates [16], relevant and accurate data can be collected. This method is chosen for its ability to accurately and easily collect the data. If compared to making real measurements on the ciphers implemented in silicon, it is easy to see the benefits with simulation. Not only is it easier, cheaper and less time consuming but the design can also be changed and recompiled without having to go through the process of fabricating the cipher.

To ensure that the sample size is adequate and provide the best possible readings for the analysis, two steps are going to be taken. First, Design Vision is set to use high effort for area and power when compiling the design. Using this setting, Design Vision analyses the design several times over, trying to find what configuration of gates would use the least amount of area and power. The second step taken is running the compilation of the circuit multiple times. This is done until the compilation stops improving the design and makes sure that the evaluated circuits are as good as they could be.

3.3.2 Area estimation

Using the information about the gates found in the UMC 65nm library, Design Vision can calculate the area requirement for a circuit. When calculating the cell area, Design Vision adds the area of all the gates used to implement the design. The cell area is often used to compare different circuit’s area requirement and is measured in the relative unit GE. When using GE, all gates are expressed in terms of relative size to the smallest NAND-gate in the used library, which have the defined size of 1 GE. This way, circuits can be compared in terms of size independent of what technology they are implemented in.

The area of the circuit can also be measured in total area. This is the area required to implement the circuit, including signals between gates and the clock distribution network. Although the total area gives a more accurate reading of the actual area requirement of a circuit, it is not as suitable when comparing different designs. This is because a circuits can have wildly different area depending on how efficient the compiler manages to place and route the design. Only taking into account the area used by the cells of the design gives more reliable results.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Cell area(μm²)</th>
<th>65nm library cell used</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-input NAND</td>
<td>1,44</td>
<td>CKND2M2R</td>
</tr>
<tr>
<td>2-input NOR</td>
<td>1,44</td>
<td>NR2M1R</td>
</tr>
<tr>
<td>2-input OR</td>
<td>2,16</td>
<td>OR2M1R</td>
</tr>
<tr>
<td>2-input AND</td>
<td>2,16</td>
<td>AN2M1R</td>
</tr>
<tr>
<td>3-input AND</td>
<td>2,52</td>
<td>AN3M1R</td>
</tr>
<tr>
<td>2-input XOR</td>
<td>3,6</td>
<td>XOR2M2RA</td>
</tr>
<tr>
<td>Flip-Flop</td>
<td>6,84</td>
<td>DFQ2M2RA</td>
</tr>
</tbody>
</table>

Table 2: Cell sizes for UMC65nm library
3.3.3 Power estimation

Design Vision include the tool Power Compiler, which can be used for gate-level power analysis of circuits. Power Compiler uses the gate-level netlist, the technology library and the switching activity of the gates to calculate the power estimation. This is done both for the power consumed during net switching and the transistors leakage power. The switching activity can be annotated using post synthesis simulations using a testbench. If no switching activity is annotated, Power Compiler instead switches the primary inputs of the circuits with a predetermined probability. This probability is by default a 10% chance that an input will be high.

When the dynamic power and leakage power of the design has been evaluated, the power consumption per bit is calculated. This is to enable fair comparison between ciphers with different throughputs and also to see how the power consumption scales with the clock rate. The calculation is done with the formula below.

\[
\text{Total Power/Clock rate} = \frac{\text{Throughput}}{\text{Throughput}}
\]

3.3.4 Maximum clock rate

The maximum clock rate of a circuit is determined by its critical path. Signals need to pass from the input of a gate to the output by switching the internal transistors of the gate. Since the switching of a transistor is not done instantaneously, but takes a little bit of time, it induces a delay in the signal. If a signal then have to travel through a series of gates, this delay becomes more significant. If we switch the input of a gate hierarchy and instantaneously check the output, it will not be correct since it takes time for the signal to reach the output. The time it takes determines how fast we can switch the clock, giving the circuit its maximum clock speed.

Design vision includes a tool to calculate the critical path of circuits, making it easy to determine the maximum clock speed. Path slack reports the paths with the worst delay so that the user can either try to improve these or calculate the maximum clock speed. Calculating the maximum clock speed is done through specifying the circuits clock input using an arbitrary clock speed. Using the path slack tool, design vision can report how the timing of the critical path compares relative to the clock. If the time needed to pass through the critical path is less than the period of the clock, the slack will be met and the left over time will be reported. If the time needed to pass through the critical path is longer than the period of the clock, the slack will not be met and the time violation will be reported. The reported time is what is used to calculate the maximum clock speed. Since the clock speed is defined in nanoseconds the slack time can easily be added or subtracted to the specified clock inputs period time then reporting the critical path again for insurance of correct results. Since the period(T) is reciprocal to the frequency(f), the frequency can be calculated with formula below.

\[ f = \frac{1}{T} \]

3.3.5 Throughput

The throughput of a cipher is the number of bits the cipher can encrypt every second. This is determined both by how many bits the cipher can encrypt per clock cycle and how fast the cipher can be clocked. The number of bits encrypted per cycle is one for all non-parallelized stream ciphers. This
is because one bit of the keystream is generated every clock cycle that can be used for encryption. For block ciphers, the throughput is the number of bits encrypted per encryption round, divided by the total number of cycles it takes to load data to and from the cipher together with the cycles it takes to encrypt.

3.3.6 Latency

The latency of the ciphers is for our purpose the amount of time it takes from turning on the power of the circuits until it’s able to deliver the first bit of an encrypted message. This is done through calculating the number of clock cycles it takes to deliver the first part of the encrypted message and multiplying it by the period of a clock cycle. This means that the faster you can clock the cipher, the lower the latency becomes.

3.4 Applying Low-power Techniques

Implementing low-power techniques in designs is a great way to lower the cipher’s power consumption. These techniques have different levels of applicability on different designs. Some techniques, like clock gating and power gating, can be hardcoded into the design and implemented manually. This method, although it allows for great control, is error prone and can result in the circuit not working as intended. Therefore many modern ASIC compilers, including Design Vision, have built in support for one or both of these techniques and can apply them automatically during compilation. For this thesis, clock gating will be implemented automatically using Design Vision, as it is the only low-power technique directly supported. The circuit will also be analyzed visually to verify the implementation and to see if other low-power techniques, like power gating, might be viable.

3.5 Assessing reliability and validity of the data collected

This chapter describes what will be done to ensure that the reliability and validity of the research will be at an acceptable level. Reliability means how we ensure that no matter how many times the tests are redone, the same results will always be acquired. Validity on the other hand means that the results that are gained from our experiments actually are correct and reflects the actual situation.

3.5.1 Reliability

Test-retest reliability can be used by compiling and evaluating the circuits several times over. One option is to redo the experiments in this thesis several times over. This will be done to some extent, as the compilation is run several times over to try to improve the result and in order to find the best solution. This is performed both automatically and manually as described in chapter sampling Method. Another option would be to let an independent actor perform the same tests to compare the results with those gathered from this report.

Parallel forms reliability could be done by letting two different and independent programs run their simulations and evaluations in parallel. The data collected from the two programs would then be compared to each other to ensure that they are equal, which ensures a higher reliability. Parallel forms reliability also relates to another method called internal consistency reliability, which is a measure of how consistent the results are after conducting the experiments with different tools, all doing the same type of work. This could be done using different types of software to evaluate the area, power consumption, throughput and latency for the circuits.

Inter-rater reliability would require several judges, or a jury, to judge the correctness of the results from the experiments and give their independent reviews. These reviews of the correctness would
then be compared to each other in order to see if they differ or correspond. As different people may interpret the data and results different based on their level of knowledge within the subject, a constant interpretation throughout the independent reviews would favor a strong reliability of the data [17].

3.5.2 Validity

Construct validity ensures that the tool used for measuring actually measures the variables that are of interest. There exist different methods to ensure the construct validity, some of which are face validity, sampling validity and convergent validity, whom all are discussed below [17].

Face validity aims to let the stakeholders assess how viable the tests seem. Although this is not one of the best validity-tests, it is one of the most common methods. It also ensures beforehand that the obtained results will be accepted by the stakeholders as they know, and approved, the method used to obtain the results. Throughout the thesis, the stakeholders will be the examiner and supervisor. As the stakeholders are interested in the validity of the results, it is important to use face validity to discuss the used methods with the stakeholders. This ensures that at the end of the thesis, results can be delivered which to them are useful.

Sampling validity has limited applicability in this thesis as it mostly concerns sampling sizes. When deploying a survey, the results will vary different amounts based on what group of people were questioned. To get the exact results, all answers of the people whom the survey concerns needs to be recorded. For very general surveys, this focus group may include the entire population of the earth. Since it is practically impossible to construct a survey or experiment which includes this many people, researchers only asks a group of people. The size of this group compared to the focus group can play a crucial role in the validity of the results, where a bigger sample group obviously gives a better validity.

Convergent validity compares the results on the object with other results, to see if they are related, which is expected. This can be done through examining the theoretical results given in previous work, such as cell area and throughput with the simulated cell area of Espresso. We can also use this to compare the throughput of Espresso. The power consumption of Espresso and AES implemented in a FPGA can also be used for comparison.

3.6 Planned Data Analysis

This subchapter describes the technique of which the acquired data will be analyzed in order to draw conclusions. It is important that the data is handled appropriately to avoid creating errors during the analysis. The subchapter therefore also contains which software tools are used to facilitate the data analysis.

3.6.1 Data Analysis Technique

Since the data generated by the experiments in the research already are very concrete, the need for further analysis of the data will be limited. The gathered data will instead be inserted straight into a spreadsheet, ready for comparison by the reader.

3.6.2 Software Tools

To assemble the gathered data, a spreadsheet in Excel will be used to ensure good readability. No further software tools will be used for data analysis.
4 Implementation and evaluation of AES and Espresso

This chapter describes the work done during the thesis and the choices made when conducting the research. The hardware design is presented along with the reasoning behind some of the design choices. The verification process and synthesis of the circuit is also presented along with the methods used to produce the correct result.

4.1 Hardware design

The circuits compared in this thesis are implemented using the descriptive computer language VHDL. For Espresso, this code was written and compiled using Quartus 2 and can be found in Appendix B. For AES, which is a more complex algorithm, the code was instead retrieved from an online source [18]. This was done to avoid making mistakes when implementing the circuit and to save time.

4.2 Espresso

The description of the Espresso stream cipher does not include how to load the key and IV into the shift register. Therefore, this has to be implemented for the circuit to work. The loading of the circuit can be done either sequentially, in full parallel or a combination between both. Full parallel loading allows the whole key and IV to be loaded into the register with the latency of one clock cycle. This is done by loading each bit into its respective register simultaneously using one extra mux for every flip-flop in the register. This technique greatly reduces the latency to the cost of increasing the size of the circuit. When loading sequentially, all bits are loaded into the first flip-flop of the register, one bit per clock cycle, and then shifted along the register. Loading sequentially takes 256 clock cycles but greatly reduces the area requirement of the circuit compared to parallel loading. When combining sequential and parallel loading, \( n \) number of bits are loaded into the register at once, where \( n \) has base 2 and \( 1 < n < 256 \). This method requires less area than full parallel loading and can be done in less clock cycles than sequential loading. Which method is chosen depends on the requirements for that specific implementation of the algorithm.

Because the NLFSR gets updated every clock cycle, additional steps needs to be taken when using loading methods that take several clock cycles. The feedback of the shift register, caused by the feedback functions, needs to be temporarily disabled during loading. This is to avoid the already loaded values to be overwritten by the feedback functions during the loading phase. To do this, one AND-gate is added for each of the feedback functions. The AND-gates act as single input MUX’es with the LOAD_EN signal acting as enable, only allowing feedback when set to ‘1’. The LOAD_EN signal is therefore active low, allowing uninterrupted loading when set to ‘0’. Another effect of sequential loading is that the pipelined output function will hold values in its flip-flops after loading is complete. This is unwanted since it will make the cipher produce a different keystream during the encryption phase. This can be avoided by either disconnecting the output function during the loading phase or resetting the output function after loading is complete. For this thesis, the latter option was chosen to avoid the additional logic required to disconnect the output function’s many inputs.

The loading of values into the registers is done by adding MUX’es between the registers. By changing the select signal, external values can be shifted into the register instead of the value of the previous register. Because MUX’es are fairly large (2 AND-gates, 1 OR-gate and 1 NOT-gate) a simpler circuit can be used when area is a concern, which is the case when implementing the 1 bit sequential loading. Instead of the mux, just one AND-gate and one OR-gate can be used. This method is able to disconnect the feedback and priorities the LOAD signal using the LOAD_EN signal but not the other way around. It is therefore important to not send anything other than zeros on the LOAD signal when not loading. The simplified loading circuit can be seen below.
4.3 AES

The implementation of AES used in this thesis was chosen for being lightweight, small and for having been compared to Espresso in previous work [3]. The implementation consists of three modules that can be used individually or together. The modules are responsible for the Key expansion, Encryption and Decryption respectively. Because the modules can be used independently, the Decryption module has been left out. This is worth noting since block ciphers and stream ciphers work differently. Leaving out the decryption module makes it so that the implementation of AES can encrypt data but not decrypt data. When using a stream cipher like Espresso, the decryption is done by XORing the ciphertext with the keystream. Because of this, the encryption and decryption parts of the stream cipher are the same and one part can not be excluded from the design. The choice of leaving out the Decryption module was further motivated by how the modules worked together. If all modules would have been imported without making any changes, the result would be two independent and separate circuits. One circuit handling encryption and the other one handling decryption with both implementing their own Key expansion modules. To avoid this, a framework for the encryption algorithm would have to be coded to synchronize the modules.

For the loading of the Key and plaintext into AES, a combination between sequential and parallel loading is used. 8 bits are loaded every clock cycle, requiring 16 clock cycles to load the 128 bit Key and plaintext separately. Using a 128 bit key as done in this thesis, the Key expansion phase takes 133 clock cycles and the encryption takes 99 clock cycles. This would take longer time using a longer key since more rounds of encryption would be executed for each block of plaintext.

AES is in this thesis implemented in the Electronic Codebook mode of operation. Unlike most other modes of operation, the ECB mode does not use any type of feedback or operation to alter the plaintext or ciphertext, before or after the encryption is done. This avoids the need to implement additional logic and makes it the most lightweight and energy efficient implementation [19]. Although other modes of operation are more secure against attacks as previously explained, ECB provides a better baseline for the algorithms power consumption.

4.4 Verification of the logic circuits

To verify that the ciphers had been coded correctly in VHDL, two different methods were used in combination. The first method used was visually verifying the circuits in the built in RTL-viewer in Quartus, and later on, the schematic viewer in Design Vision. These are tools that draw the circuit and netlist in a graphic interface. This allows the individual gates in the circuit to be checked, making sure that the right gates are used when the compiler interprets the code. This can also be done to look at
the critical path of the circuits in order to find way to make it shorter. This is a suitable method of verification for smaller circuits such as the output and feedback functions of Espresso, but becomes increasingly difficult when looking at more complex circuits such as AES.

While a visual verification of a design can make sure the correct gates are used, it is often hard to fully understand the behavior of the circuit by only looking at it. To verify that the circuit display the correct behavior, running a simulation is a much better choice. After the ciphers had been compiled in Quartus, they were simulated in ModelSim. ModelSim is a simulation tool developed by Mentor Graphics. It is used for simulation of VHDL files and shows states of the inputs, outputs, signals and variables of the circuit during simulation. The circuit’s inputs needs to be set to desired values, the clock speed needs to be defined and the outputs needs to be checked for correctness. This can either be done by coding a test bench in VHDL which can, depending on the test bench complexity, do some or all of these things automatically during simulation. This can also be done directly in ModelSim by forcing the values of the inputs during simulation directly in the GUI. This technique is good for quick, small simulations, but not suitable for this application as the input for sequential loading needed to change many times during initialization. Manually doing this would take huge amounts of time for every simulation attempt make it more prone to errors. Therefore, writing a testbench was more suitable.

The testbench for AES acted as a controller, going through the different stages in the encryption. The testbench fed the encryption key to AES and waited for the Key expansion phase to finish. When the round keys has been produced, the plain text was fed into AES for encryption. When AES signaled that the encryption was complete, the testbench controlled that the encryption had been done properly. This was done by comparing the ciphertext to the known result when using the specific Key and data used. If the ciphertext was correct, new plain text was loaded into the cipher and the encryption cycle continued. The vectors used can be seen below.

- Key: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C
- Test vector: 6B C1 BE E2 2E 40 9F 96 E9 3D 7E 11 73 93 17 2A
- Ciphertext: 3A D7 7B B4 0D 7A 36 60 A8 9E CA F3 24 66 EF 97

As with the AES testbench, the testbench for Espresso simulated regular operating conditions for the circuit, making sure it worked properly. The testbench went through the loading, initialization and generation phases for Espresso while at the same time checking that the output was correct. Because Espresso is a stream cipher, the initialization and output differs from AES. During the loading, the IV and Key were loaded into the circuit for as many clock cycles as needed. The cipher was then initiated and the keystream was produced. Instead of controlling an encrypted message, the testbench controlled the keystream based on the expected output when using that specific pair of IV and Key. The test vectors used are as follows.

- Key: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
- IV: 00 01 02 03 04 05 06 07 08 09 0A 0B
- KS: E6 AF DE 2F AC B5 C8 9A AB E9 36 1B F8 13 9C 96 A6 3D E3 9F
Regarding the test vectors, it’s worth noticing that the given key stream is only valid for a version of the Espresso without a pipelined output function. This means that the output function, which is described to include flip-flops, needs to be implemented without flip-flops for the test vectors to be used properly. This removed the ability to test the actual implementation of the output function that was used when comparing the ciphers. Therefore, the real output function was instead verified graphically using the RTL-viewer as described earlier.

### 4.5 Different implementations of Espresso and AES

Because of the many ways of implementing the algorithms, several different versions were implemented in order to make a more thorough comparison. This subchapter goes through the different implementations of both AES and Espresso compared in the thesis.

#### 4.5.1 AES

This is the most basic implementation of AES, compiled without any form of optimization done to the circuit. Its primary role was to establish a baseline for the ciphers performance.

#### 4.5.2 AES optimized

AES was also compiled with optimization enabled. This let the compiler change what gates were used to find equivalent alternatives to the functions with better performance, resulting in a smaller, faster and more energy efficient version of the implemented circuit.

#### 4.5.3 Espresso

This version of Espresso is the most basic one, exactly following the description of the algorithm [2]. Because of this, it does not have the logic required for loading any values into its registers. Therefore it cannot be used in any real life application but was still used to establish a baseline for the ciphers size and energy efficiency.

#### 4.5.4 Espresso 1-bit sequential loading

This version of Espresso is equivalent to the basic version with the addition of the logic required for sequential loading. This version is the most basic one that can be used in real applications.

#### 4.5.5 Espresso 1-bit sequential loading optimized

Espresso with sequential loading that has been compiled with optimizations enabled. Several of these optimizations are implementation and library specific but some are general improvements and will be discussed later.

#### 4.5.6 Espresso 2-bit parallel loading optimized

Espresso with 2 bit parallel loading that has been compiled with optimizations enabled. This version loads 2 bits into the NLFSR at once, one bit at register 255 and the other one at register 127. This cuts the loading phase into half while only adding a very small amount of logic.
4.5.7 Espresso 256-bit parallel loading optimized

Espresso with full 256 bit parallel loading that has been compiled with optimizations enabled. This version loads into all the 256 registers at once, reducing the latency of the cipher as much as possible but adds a lot of logic.

4.6 Synthesis in Synopsys

After the circuits had been confirmed to work correctly in Modelsim they could be implemented in 65nm technology. In order to do this, the program Design Vision was used. Design Vision was ran on one of the ICT-schools shell servers through an SSH connection at the address atlantis.it.kth.se, the specs of the server was a HP Proliant DL385 G7 consisting of two AMD Opteron 6172 at 2,1GHz CPU’s for a total of 24 cores with 82GB of RAM and a 2TB HDD running the RHLE5 operating system. The VHDL files of the ciphers were imported into the program through a two-step process called analyze and elaborate. The first step analyzes the HDL files, converting their information into an HDL-independent format file which then was used during elaboration. The second step elaborates the design, making a technology-independent design with the file created during the analyze-process. After importing the design, the circuit was shown graphically through schematic view as previously mentioned.

4.6.1 Operating conditions and constraints

Before compilation could be done the operating conditions for the circuits need to be specified in Design Vision. The first thing was to set the target library which the compiler should use to tie the VHDL code to cells. This was done automatically using a .setup file. Through File -> Setup, the Search path could be confirmed to containing the files the compiler was told to use for Link library, Target library, Symbol library and Synthetic library.

Design Vision enables for Input delay and Output delay to be defined through Attributes -> Operating environment -> input delay/Output delay. This helps to define how a module would behave on a system level. These delays depend on the output characteristics of the proceeding module and the input of the following module and they define when signals reaches inputs relative to the clock. These conditions are used to include the paths between modules when estimating the critical path or path slack. Since the interest was the critical path within the circuits these delays were set to 0.

The physical wires used to connect cells within the design have different resistive and capacitive characteristics depending on the amount of wire. This induces a delay in the signals traveling through the wires. To enable Design Vision to calculate these delays a predefined wire model was used. This was set through Attributes -> Operating Environment -> Wire Load. The wire load used was QA from the u065gioll25mvr_25_tc library.

Physical circuits behave differently for a range of operating conditions such as supply voltage, temperature and process. The process specifies the deviations that may arise during the semiconductor fabrication process. These conditions can strongly affect the speed of the circuit. It is therefore necessary to specify these conditions to Design Vision before evaluating the circuits performance. This was done through Attributes -> Operating environment -> Operating conditions. The library was set to uk6gsclmlvbbbr_120c25_tc as well as the condition. Design Vision can be set to use these conditions either in worst case and best case scenarios or a single analysis. Single analysis was used, and the library file defined the supply voltage as 1.2V, the working temperature as 25°C and process as 1.00.
4.6.2 Mapping the design

Before compilation in Design Vision was done, a generic library called GTECH was used to represent all the cells of the design. Compiling the circuit tied the code together with the UMC 65nm library instead of GTECH, and therefore switched out the cells of the GTECH library for cells from the UMC 65nm library. For compilation to work, Design Vision needed three inputs, these were a RTL hardware description, design constraints and a standard cell library. Through synthesis, Design Vision could produce an output in the form of a gate level netlist file.

When compiling the circuit in Design Vision the user can set a number of options. Since it was crucial that Espresso was implemented the way described [2], it was necessary to make some changes to these options before compilation.

![Compilation settings in Design Vision](image)

The preceding options made sure that the target library UMC65 was mapped to the design without making changes to the circuit. The key note is the design rule option “Fix design rules only”.

How Design Vision interpreted the code differed from how it was intended, which lead to some parts of the design being faulty. This was checked graphically using the Schematic View in Design Vision and by doing so, it was found that the feedback functions $g_{255}(x)$ and $g_{217}(x)$ and the output function $z(6)$ were faulty.

![Post-synthesis circuit needing to be recompiled](image)
The above image is an illustration of the post-synthesis circuit and represents how the feedback functions $g_{255}(x)$ and $g_{217}(x)$ looked after compilation. The desired gates of the functions are two NAND gates and one NOR because of their small area and low propagation delay. In order to customize the functions to their desired equivalence they had to be optimized, gate by gate. To do this, Design Vision was specified to optimize only certain parts of the design. This was done by setting “DON’T TOUCH” to true for all the cells which the compiler should not touch and compiling the design using the design rule option “Fix design rules and optimize mapping”. The functions needed to be optimized one gate at a time, as optimizing the whole feedback functions would make the compiler use library specific gates and change the functions in undesirable ways. This was done by having “DON’T TOUCH” set to false for the gates which were to be merged into a single gate, then compiling the design, setting “DON’T TOUCH” to true for the new gate. This was repeated for all group of gates which were desired to be merged.

For the function $Z_{6}$ of the output function, six variables are combined through AND-gates. When imported into Design Vision this function was interpreted as consisting of six AND-gates, combining all the variables. It was however desired that these instead were switched out for one two-input and two three-input AND-gates. The same process as described in the above paragraph was applied to these gates to transform them into the desired circuit. The transformation of the gates is shown in the figure below.

For the function $Z$ of the output function, six variables are combined through AND-gates. When imported into Design Vision this function was interpreted as consisting of six AND-gates, combining all the variables. It was however desired that these instead were switched out for one two-input and two three-input AND-gates. The same process as described in the above paragraph was applied to these gates to transform them into the desired circuit. The transformation of the gates is shown in the figure below.

![Figure 6: Optimizations made to Z6](image)

The flip-flops of the output function can be reset by either RESET or $Z_{RESET}$. This is managed by using one extra AND-gate before the flip-flop’s reset inputs. Design Vision interpreted this as using one OR-gate and one NOT-gate for every flip-flop in the output function and two NOT-gates for the output function. This can instead be implemented by combining all of this logic into a single AND-gate for the whole output function. To do this, all of the redundant logic’s “DON’T TOUCH” was set to false and the circuit was recompiled in Design Vision to combine all the logic into a single AND-gate.

When the circuit was finished and working as intended, the compilation was ran again. This time the mapping options for **Area effort** and **Power effort** were both set to high while all the cells were set to “DON’T TOUCH”. By doing this, the compiler tried to find the best way to build the circuit without changing any of the cells in the design.

### 4.7 Generating the results

The results of the circuits after synthesis can be shown in Design Vision with the use of “reports”. Design Vision is able to generate many different reports, most of which can be found under Design in
the toolbar. The ones of most interest were generated through Design -> Report area, Design -> Report clocks, Design -> Report power and Timing -> Report timing path. For Report power, the option Analysis effort was set to high to get the most accurate value possible. This used more processing power and time to make the analysis. These reports had to be generated for all of the different design-implémentations. Report timing path, Report clocks, and Report power were also used every time an implementation was clocked with a new speed. Since the area of an implementation is unaffected by the clock speed, the same area report could be used for different clock speeds.

4.8 Implementing and Evaluating Low-power Techniques

When results for the ciphers had been generated, further efforts were made to improve the energy efficiency of Espresso. Among the previously discussed low-power techniques, the only one directly supported in IC Compiler is clock gating. Implementing this was done automatically by enabling it in the compiler options. When the design had been recompiled, additional reports were generated in order to compare the results, with and without clock gating. This was done for the sequential loading, 2 bit parallel loading and full parallel loading versions of Espresso. When examining the circuits post compilation, it was noticed that no changes had been made to the circuit. This was also confirmed by the reports that showed a power consumption equivalent with that of the non-clock gated versions of the designs. The ineffectiveness of applying clock gating here can be explained by the fact that all the computing blocks and shift registers of Espresso are active in every clock cycle. Therefore, no clock can be disabled for any parts of the design. Examining the design further, possible ways to implement both clock gating and power gating were discovered although they were not implemented.
5 Analysis

This chapter presents the major results of the thesis in the form of a table. The results are then evaluated for their reliability and validity. Under discussion the results are explained and discussed from different viewpoints of how they are interpreted.

5.1 Major results

The results gathered from the evaluation of the different implementations of the circuits are arranged in table 2. The different implementations can be clocked at different speeds. The table therefore show the characteristics of the implementations when clocked at the fastest possible speed and at 1GHz since this is of interest in the discussion. 1GHz was the maximum speed at which the AES optimized could be clocked, so this worked as a good baseline. Since the un-optimized version of AES could be clocked at a maximum of 666.7MHz this specific implementation was not clocked at 1GHz.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cell area (GE)</th>
<th>Energy/bit (pJ)</th>
<th>Throughput (bit/s)</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>30919.00</td>
<td>21.97</td>
<td>651'399'491</td>
<td>396</td>
</tr>
<tr>
<td>AES optimized 1GHz</td>
<td>28808.50</td>
<td>21.79</td>
<td>977'099'236</td>
<td>264</td>
</tr>
<tr>
<td>Espresso</td>
<td>1424.00</td>
<td>2.11</td>
<td>1'000'000'000'000</td>
<td>259</td>
</tr>
<tr>
<td>Espresso sequential loading 1GHz</td>
<td>1669.50</td>
<td>2.31</td>
<td>1'000'000'000'000</td>
<td>515</td>
</tr>
<tr>
<td>Espresso sequential loading 2.94GHz</td>
<td>1669.50</td>
<td>2.31</td>
<td>2'941'176'471</td>
<td>175.1</td>
</tr>
<tr>
<td>Espresso sequential loading optimized 1GHz</td>
<td>1582.75</td>
<td>2.32</td>
<td>1'000'000'000'000</td>
<td>515</td>
</tr>
<tr>
<td>Espresso sequential loading optimized 3.13GHz</td>
<td>1582.75</td>
<td>2.32</td>
<td>3'125'000'000'000</td>
<td>165</td>
</tr>
<tr>
<td>Espresso 2 bit parallel loading optimized 1GHz</td>
<td>1590.00</td>
<td>2.37</td>
<td>1'000'000'000'000</td>
<td>387</td>
</tr>
<tr>
<td>Espresso 2 bit parallel loading optimized 3.13GHz</td>
<td>1590.00</td>
<td>2.37</td>
<td>3'125'000'000'000</td>
<td>123.84</td>
</tr>
<tr>
<td>Espresso 256 bit parallel loading optimized 1GHz</td>
<td>2120.50</td>
<td>3.07</td>
<td>1'000'000'000'000</td>
<td>260</td>
</tr>
<tr>
<td>Espresso 256 bit parallel loading optimized 3,13GHz</td>
<td>2120.50</td>
<td>3.07</td>
<td>3'125'000'000'000</td>
<td>83.2</td>
</tr>
</tbody>
</table>

Table 3: Major results table

Because the clock gating was not able to be implemented, no results exist for how much power saving clock gating could provide. Although it was not able to be implemented, it could be applied with a few changes to the code. Because no other low-power technique were supported by Design Vision, these will instead be discussed together with possible clock gating implementations in section 5.4.

5.2 Reliability Analysis

To make sure that the same results is acquired independent of who uses this method, test-retest reliability was used. This was done to the extent that all the implementations were compiled in Design Vision several times. Even though different values were acquired for different compilation attempts of the same implementation, these differences were neglected as they were well below 1%. The designs
were however compiled until what deemed to be good and fair results of the characteristics were acquired.

Parallel-forms reliability was done through using different servers and computers to run the evaluation tool Design Vision on. Since several servers were available for use, it was possible to compile the implementations on different ones to ensure the same results were acquired. It was also possible to connect to the servers using SSH from different computers to ensure this did not affect the results.

5.3 Validity Analysis

Since Design Vision is a very well-known and trusted software, used throughout the industry for creating and evaluating IC’s, it would be more likely that an error in the results would be due to the user rather than the program. Comparing the implementation of Espresso without any loading shows us that the cell area of 1424GE compares fairly closely to the theoretical area of 1497GE calculated in [2] using typical 90nm CMOS values for area. The difference is due to differences in the technologies used for evaluation as we use the UMC65nm library and the earlier value was using a 90nm technology.

5.4 Discussion

This subchapter discusses the results of the thesis. The discussion includes the different versions of the compared algorithms and the optimizations done to Espresso. Low-power technique implementations are also presented along with explanations of advantages and disadvantages of the implementations.

5.4.1 AES

We use AES as the baseline for our comparisons of different implementations of Espresso. This implementation was done without letting Design Visions compiler make any changes to the circuit.

5.4.2 AES optimized

The optimized version of AES is considerably better with respect to several aspects compared to AES, this was to be expected as we give Design Vision freedom in making changes to the circuit in order to make it smaller, faster and more energy efficient. After optimization was done, the circuit became 6.8% smaller, 33% faster, used 0.8% less energy per bit and had 33% lower latency. The data gathered for this implementation of AES was used for comparison throughout this chapter. This version was able to be clocked at a maximum of 1GHz which was also the clock speed used as a baseline for comparison of the different ciphers ran at the same clock speed.

5.4.3 Espresso

This implementation is the smallest version of Espresso with a size of 1424GE cell area. This is explained by the absence of a loading function. The circuit could therefore not be used in an actual implementation, since there is no way to load the key and IV value. This implementation was made for comparing the theoretical characteristics of the circuit explained in [2] for validity of the results. The most interesting characteristic of this implementation is its area, as we then can compare how much extra area is used for different types of loading of the key and IV.
5.4.4 Espresso 1-bit sequential loading

This implementation uses sequential loading to load the key and IV into the shift register. The algorithm is mapped to the CMOS library exactly as described [2], with the addition of the sequential loading. Sequential loading requires a small amount of gates but takes a long time to load the key and IV, making this implementation suitable for applications with size-limits but low requirements of latency. The area is 1669.5GE which shows that the cost of the additional logic used to load the key and IV is 245.5GE. The power-per-bit for Espresso with sequential loading is calculated as 2.31pJ for both 1GHz and the maximum clock speed at 2.94GHz. Thus, using only about 11% of the power-per-bit as the standard, optimized AES implementation. This implementation could be clocked at about 2.94GHz before violating the critical path. This is 2.94x faster than the maximum clock speed of AES.

5.4.5 Espresso 1-bit sequential loading, optimized

This implementation uses the same concept as “Espresso sequential loading” but is implemented in the hardware with the use of different gates. The design is compiled in Design Vision without putting any constraints on Design Vision and thus allowing it to change the circuit. This enables Design Vision to map some of the cells to non-general cells which may be library specific to the UMC65nm library. These might be gates such as a 4-input XOR-gate with two inputs being inverted, etc. This is done to optimize the circuit in terms of area, throughput and power usage. The most interesting implementation to compare this to would be the Espresso with sequential loading that was not optimized by Design Vision. We can see that the optimized version saves about 5% of the total area and only about 0.01 pJ of energy-per-bit. What is interesting is the increase in the throughput as this version can be clocked 1.06x faster giving it a clock speed of 3.13GHz.

5.4.6 Espresso 2-bit parallel loading, optimized

This implementation focuses on reducing both the area and latency. This is done through the use of one extra MUX, loading the key and IV into two places at the shift register at the same time, effectively cutting the load time in half. Theoretically, this should decrease the latency about one fourth compared to sequential loading when run at the same clock rate. At 3.13GHz the latency is 123.84ns. The same circuit with sequential loading ran at 3.13GHz gives a latency of 165ns. This shows us we have a 24.9% latency time saved with the addition of a single MUX. This corresponds to the calculated theoretical savings. Comparing the cell area of 1590GE to 1-bit sequential loadings 1582.75GE, we have an increase of only 0.45% in area. Making this tradeoff can be very profitable depending on the application of the circuit.

5.4.7 Espresso 256-bit parallel loading, optimized

This implementation of Espresso focuses on having low latency. The cost of this is a higher area, as additional logic for every flip-flop in the shift-register is required to sequentially load all of the key and IV into the register at once. This enables the key and IV to be loaded into the register in just one clock cycle. For the clock rate of 3.13GHz, this implementation have a latency of 83.2ns. Compared to the 1-bit sequential loading clocked at 3.13GHz which has a latency of 165ns, this saves us 49.6% latency. The cost for this is mainly in the area. With the area of 2120.5GE, this equals an increase of 34%.The energy-per-bit is also increased from 2.31pJ to 3.07pJ. This is a result of the 256 extra muxes added to the cipher. Even when the cipher has been loaded, the data from the previous flip-flop has
to pass through one AND-gate and one OR-gate on its way to the next flip-flop. This results in switching of these gates when the signal changes from zero to one or one to zero, using extra energy.

Figure 7: Internal logic of MUX

5.4.8 Espresso optimization discussion

When analyzing the optimized version of the Espresso stream ciphers, it is noticed that many small changes were made. Most of these are implementation specific and cannot be a part of a general description of the algorithm. This kind of optimization is best left to the compiler, although the changes made should be manually controlled post-compilation. The other kind of optimization are those who are general improvements to the algorithm and can be part of its description. In this category, two optimizations are found. The first one is in the output function of Espresso. In the first part of the output function, the pairs of AND-gates connected to XORs can be replaced with NAND-gates. Because NAND-gates are smaller than AND-gates, this method makes the area of the output function smaller. Although this only saves a few GE, there are six gates that can be replaced and the optimization is not implementation specific.

Figure 8: Output function optimization

The other general optimization that can be done is in the circuit used for switching between the Initialization and the Keystream generation phase. In the description of the cipher, it is stated that the circuit is implemented two times, one time for register 255 and one time for register 217. If you instead take the signal from the same flip-flop and XOR this with both registers, you can use one AND-gate and one flip-flop instead of two of both.

5.4.9 Low-power Technique Discussion

Both clock gating and power gating save power by deactivating logic when it is not used. Because the large majority of the logic in stream ciphers is being used during the encryption phase, these techniques have limited applicability. When compiling Espresso with clock gating enabled, no changes were made. This is most likely a result of the compiler not wanting to change the design and could be resolved by making changes to how the code was written. When analyzed, several ways to
implement both clock gating and power gating was found. Therefore, these implementation suggestions will be presented and discussed. Some logic needed for the Initialization- and Encryption phase is not needed during loading. Likewise, some logic needed for loading is not needed for the Initialization- and Encryption phase. Therefore, the Loading phase will be discussed separately from the other phases.

In addition to clock gating and power gating, both Gate level power optimization and Multi-threshold logic could be utilized to achieve power saving in Espresso. By utilizing a library with multi-threshold logic, Espressos critical paths can use faster and more energy consuming gates while the rest of the circuit uses gates that are slower but have lower power consumption. Some gate level power optimizations could possibly be done as well but are much harder to analyze manually and are best left to the compiler to implement.

### 5.4.10 Loading phase

During the Loading phase, both the feedback functions and the output function are not needed. These together equals 22 2-input AND-gates, 2 3-input AND-gates, 4 2-input NAND-gates, 2 2-input NOR-gates, 34 XOR-gates and 10 flip-flops if Espresso is implemented exactly as it is described. If Espresso is implemented with sequential loading, the loading phase is 256 clock cycles. If clock gating were to be implemented, this would mean that 10 flip-flops could be inactive during 256 cycles. If instead power gating were to be implemented, the entirety of the logic listed could temporarily be powered down. This energy saving could become substantial if the cipher is reloaded often. When instead loading Espresso with several bits simultaneously, the effectiveness of the mentioned methods will decline. This is because with increasing number of bits being loaded at once, the number of clock cycles required to load the Key and IV declines. This means that the fewer bits that are loaded at once into the register, the bigger the relative energy saving can become.

### 5.4.11 Initialization- and Encryption phase

During the Initialization- and Encryption phase, the only logic not used is the parts used to load the Key and IV into the cipher. Therefore, unlike with the Loading phase, the relative energy saving possible for the Initialization- and Encryption phase increases with the number of bits loaded simultaneously into the cipher. Because only the MUX’es used to load the values can be partially disabled, clock gating, that targets flip-flops, cannot be applied. This makes power gating the only option for these phases. Additionally, parts of the MUX still has to be enabled to let values through for the shift register to work properly. By letting the load enabling signal be active low, power gating can be utilized to turn off one AND-gate and one NOT-gate for every mux when not loading. This is visualized in the image below where the disabled gates are shown as gray. LOAD EN is the load enabling signal, LOAD is where bits are loading into the register and SHIFT is connected to the previous register in the shift register.

![Power gating example of MUX’es](image-url)
6 Conclusions and Future work

This chapter summarizes our discussion of the results to give an overview of what has been learned from the experiments and thesis. It also discusses possible future works within the area to further investigate the characteristics and Espresso’s viability to become a standard within 5G.

6.1 Conclusions

In this subchapter the conclusion of the comparison between Espresso and AES is given as well as the insights gathered from evaluating possible power-saving techniques.

6.1.1 Espresso vs AES

The conclusion of the comparisons is that Espresso outperforms for every characteristic that was evaluated. The implementation of Espresso with sequential loading, optimized in 65nm CMOS, shows us that it uses 18.2x less area, has a 3.2x higher throughput, 9.4x less energy-per-bit and have 1.5x less latency than AES. For time-critical applications, an Espresso implementation using parallel loading of the Key and IV can be used. This enables a lower latency at the cost of a larger area. Comparing this implementation shows that Espresso uses 13.6x less area, has 3.2x higher throughput, 7.1x less energy-per-bit and 3.2x lower latency than AES. Comparing the implementations of sequential loading and parallel loading shows the trade-off in characteristics. Since both have the same throughput of 3.1GHz the trade-off will be in area and power consumption versus latency. For sequential loading, the area is 1.3x smaller, the power consumption is 1.3x less than for parallel loading while the latency is 2.1x lower for parallel loading.

From the gathered results the conclusion can be drawn that Espresso would be a good encryption algorithm for the upcoming 5G network. The results show that Espresso is a very promising solution to replace AES for many applications. This is however only true for the characteristics examined in this thesis and not for security measurements.

6.1.2 Power saving

Although Espresso already has a low power consumption, it can be further improved by applying low-power techniques. Some methods like clock gating and power gating have limited applicability because of the nature of stream cipher but can still be applied by careful analysis of the cipher. Additionally, other techniques can also be applied but requires support from both the library and compiler of the code.

6.2 Limitations

A few limiting factors were discovered when conducting the work for this thesis. First and foremost, the lack of relevant previous work made it difficult to start. The only previous work done was the work describing Espresso [2] and the work comparing Espresso to AES on a FPGA[3]. No comparison of different implementations of Espresso had previously been done and no low-power techniques had been explored for Espresso. Because of the time limitation of the thesis, many low-power techniques and loading implementations of Espresso had to be left out. The time limitation also removed the ability to do post-synthesis simulation of the circuits and evaluate the power consumption of the ciphers in more than one way. Another limiting factor was that only clock gating was supported by
Design Vision. Since clock gating couldn’t be applied to the circuit, no data could be gathered for the power saving possible through applying low-power techniques.

6.3 Future work

As described in the chapter delimitations, something that isn’t covered in this thesis is the comparison to other circuits than AES. For future work, it would be interesting to compare the same type of characteristics as in this report, but for other encryption algorithms, especially ones considered for use in the 5G network. Espresso has earlier been compared to Grain-128 and Trivium. However this was done only in terms of area, throughput and latency. Espresso’s characteristics were also calculated from the theoretical circuit. It could also be of interest to compare Espresso to different implementations of AES. The one used in this thesis was quite general, but there are different implementations that prioritize one or more of the characteristics examined in this thesis, such as area or speed.

One report described in the related work looked at the power consumption of AES and Espresso implemented on a FPGA. However, because of the nature of a FPGA, it is hard to affect how the actual circuit will be implemented on the chip. Different but equivalent functions may be used which could affect the characteristics, such as power usage. To instead fabricate the circuit on a silicon chip and run the encryptions as they would be implemented in a real scenario would get more exact results of the power consumption.

As described in section 2.3, block ciphers are generally easier to implement in software and stream ciphers are generally easier to implement in hardware. Although Espresso is designed to be implemented in hardware, it could be of interest to compare a software implementation of the algorithms as well. Which characteristics should be compared in the software implementation are left to be explored.

Because no low-power techniques were able to be implemented in this thesis, it would be relevant future work to implement and evaluate the discussed techniques and implementations. In addition, several other low-power techniques exist that could be examined to see if they can be applied to Espresso.

6.4 Reflections

The social benefits of this thesis relate closely to the IoT. With the low power consumption of Espresso, new IoT products may be able to be developed that were previously not possible. These products may provide social benefits by themselves and also contribute to the overall growth of the IoT. Existing and new products may also get prolonged lifetimes by the reduction of power consumption, reducing the need to replace batteries and hardware. This helps to reduce the environmental impacts of the technologies by lowering the amount of electronic waste products and overall power consumption.
References


### Appendix A: Complete results

<table>
<thead>
<tr>
<th>Model</th>
<th>Total area (um²)</th>
<th>Total area (GE)</th>
<th>Cell area (um²)</th>
<th>Cell area (GE)</th>
<th>Dynamic power (uW)</th>
<th>Leakage power (uW)</th>
<th>Total power (uW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES optimized 1GHz</td>
<td>44 052.24</td>
<td>30 591.83</td>
<td>41 484.24</td>
<td>28 808.50</td>
<td>21 288.70</td>
<td>3 329</td>
<td>21 292.01</td>
</tr>
<tr>
<td>AES</td>
<td>47 171.36</td>
<td>32 757.89</td>
<td>44 523.36</td>
<td>30 919.00</td>
<td>14 308.20</td>
<td>4 865</td>
<td>14 312.05</td>
</tr>
<tr>
<td>Espresso</td>
<td>2 306.56</td>
<td>1 601.78</td>
<td>2 050.56</td>
<td>1 424.00</td>
<td>2 107.40</td>
<td>330</td>
<td>2 107.58</td>
</tr>
<tr>
<td>Espresso 1-bit sequential 1GHz</td>
<td>2 911.08</td>
<td>2 021.58</td>
<td>2 404.08</td>
<td>1 669.50</td>
<td>2 305.10</td>
<td>346</td>
<td>2 305.27</td>
</tr>
<tr>
<td>Espresso 1-bit sequential 2.9GHz</td>
<td>2 911.08</td>
<td>2 021.58</td>
<td>2 404.08</td>
<td>1 669.50</td>
<td>6 779.30</td>
<td>346</td>
<td>2 779.47</td>
</tr>
<tr>
<td>Espresso 1-bit sequential optimized 1GHz</td>
<td>2 781.16</td>
<td>1 931.36</td>
<td>2 279.16</td>
<td>1 582.75</td>
<td>2 316.90</td>
<td>330</td>
<td>2 317.05</td>
</tr>
<tr>
<td>Espresso 1-bit sequential optimized 3.13GHz</td>
<td>2 781.16</td>
<td>1 931.36</td>
<td>2 279.16</td>
<td>1 582.75</td>
<td>7 245.30</td>
<td>330</td>
<td>7 245.45</td>
</tr>
<tr>
<td>Espresso 2-bit parallel optimized 1GHz</td>
<td>2 792.6</td>
<td>1 939.31</td>
<td>2 289.6</td>
<td>1 590.00</td>
<td>2 366.5</td>
<td>332</td>
<td>2 366.65</td>
</tr>
<tr>
<td>Espresso 2-bit parallel optimized 3.13GHz</td>
<td>2 792.6</td>
<td>1 939.31</td>
<td>2 289.6</td>
<td>1 590.00</td>
<td>7 395.3</td>
<td>332</td>
<td>7 395.45</td>
</tr>
<tr>
<td>Espresso full parallel optimized 1GHz</td>
<td>3825.52</td>
<td>2 656.61</td>
<td>3053.52</td>
<td>2 120.50</td>
<td>3072.3</td>
<td>596</td>
<td>3 072.51</td>
</tr>
<tr>
<td>Espresso full parallel optimized 3.13GHz</td>
<td>3825.52</td>
<td>2 656.61</td>
<td>3053.52</td>
<td>2 120.50</td>
<td>9600.8</td>
<td>596</td>
<td>9 601.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Throughput (bits/s)</th>
<th>bits/clockcycle</th>
<th>Clockcycles/s</th>
<th>Energy per bit (pJ)</th>
<th>Latency (ns)</th>
<th>clockcycles x clockspeed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES optimized 1GHz</td>
<td>977 099 236</td>
<td>0.977099236</td>
<td>1 000 000 000</td>
<td>21.79</td>
<td>264</td>
<td>(16+133+16+99)cycles x 1ns</td>
</tr>
<tr>
<td>AES</td>
<td>651 399 491</td>
<td>0.977099236</td>
<td>666 666 667</td>
<td>21.97</td>
<td>396</td>
<td>(16+133+16+99)cycles x 1.5ns</td>
</tr>
<tr>
<td>Espresso</td>
<td>1 000 000 000</td>
<td>1</td>
<td>1 000 000 000</td>
<td>2.11</td>
<td>259</td>
<td>(256+3)cycles x 1ns</td>
</tr>
<tr>
<td>Espresso 1-bit sequential 1GHz</td>
<td>1 000 000 000</td>
<td>1</td>
<td>1 000 000 000</td>
<td>2.31</td>
<td>515</td>
<td>(256+256+3)cycles x 1ns</td>
</tr>
<tr>
<td>Espresso 1-bit sequential 2.9GHz</td>
<td>2 941 176 471</td>
<td>1</td>
<td>2 941 176 471</td>
<td>2.31</td>
<td>175.1</td>
<td>(256+256+3)cycles x 340ps</td>
</tr>
<tr>
<td>Espresso 1-bit sequential optimized 1GHz</td>
<td>1 000 000 000</td>
<td>1</td>
<td>1 000 000 000</td>
<td>2.32</td>
<td>515</td>
<td>(256+256+3)cycles x 1ns</td>
</tr>
<tr>
<td>Espresso 1-bit sequential</td>
<td>3 125 000 000</td>
<td>1</td>
<td>3 125 000 000</td>
<td>2.32</td>
<td>165</td>
<td>(256+256+3)cycles x 320ps</td>
</tr>
<tr>
<td>Optimized</td>
<td>3.13GHz</td>
<td>1GHz</td>
<td>2,37</td>
<td>83,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>---------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Espresso 2-bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parallel optimized</td>
<td>1 000 000 000</td>
<td>1 000 000 000</td>
<td>387</td>
<td>(128+256+3) cycles x 1 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Espresso 2-bit</td>
<td>3 125 000 000</td>
<td>3 125 000 000</td>
<td>123,84</td>
<td>(128+256+3) cycles x 320 ps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parallel optimized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Espresso full</td>
<td>1 000 000 000</td>
<td>1 000 000 000</td>
<td>260</td>
<td>(1+256+3) cycles x 1 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parallel optimized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13GHz</td>
<td>3 125 000 000</td>
<td>3 125 000 000</td>
<td>83,2</td>
<td>(1+256+3) cycles x 320 ps</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B: Source code

---------------------------------------
--Program: Espresso stream cipher
--Author: Richard Lowenrud & Jacob Kimblad
--Version: 1.0
--Updated: 2016-05-03
--Comment: The Espresso stream cipher implemented in hardware using VHDL,
--the cipher is implemented with one bit sequential loading through
--flip-flop 255 (the foremost left flip flop in the NLFSR). The NLFSR
--also makes use of a asynchronous reset signal.

library IEEE;
USE IEEE.std_logic_1164.all;

ENTITY Espresso IS
   PORT( LOAD, LOAD_EN, INIT, CLK, RESET, ZRESET : IN std_logic;
        Zout : OUT std_logic);
END Espresso;

ARCHITECTURE Behavioral OF Espresso IS

SIGNAL shift: std_logic_vector(255 DOWNTO 0) := (OTHERS => '0');
SIGNAL z: std_logic_vector(8 DOWNTO 0) := (OTHERS => '0');
BEGIN
   PROCESS(CLK, RESET, ZRESET)
   BEGIN

      IF(RESET = '1') THEN
         shift(255 DOWNTO 0) <= (OTHERS => '0');
         z(0) <= (OTHERS => '0');
      ELSIF(RESET = '1') THEN
         z(0) <= (OTHERS => '0');
      ELSIF(Clk'EVENT AND Clk='1') THEN
      -- Linear registers
      shift(255 DOWNTO 0) <= shift(255 DOWNTO 1);
      shift(255) <= (shift(0) XOR LOAD_EN AND ((shift(41) AND shift(70)) XOR shift(0))) OR LOAD;
      shift(251) <= shift(252) XOR LOAD_EN AND ((shift(41) AND shift(55)) XOR shift(1));
      shift(247) <= shift(248) XOR LOAD_EN AND ((shift(41) AND shift(102)) XOR shift(2));
      shift(243) <= shift(244) XOR LOAD_EN AND ((shift(41) AND shift(118)) XOR shift(3));
      shift(239) <= shift(240) XOR LOAD_EN AND ((shift(41) AND shift(141)) XOR shift(4));
      shift(235) <= shift(236) XOR LOAD_EN AND (((shift(67) NAND shift(90)) XOR shift(5)) AND shift(117));
      shift(231) <= shift(232) XOR LOAD_EN AND ((shift(50) AND shift(159)) XOR shift(6));
      shift(217) <= shift(218) XOR LOAD_EN AND ((shift(5) AND shift(12)) XOR shift(7));
      shift(213) <= shift(214) XOR LOAD_EN AND (((shift(4) AND shift(145))) XOR shift(8));
      shift(209) <= shift(210) XOR LOAD_EN AND (((shift(6) AND shift(146))) XOR shift(9));
      shift(205) <= shift(206) XOR LOAD_EN AND (((shift(5) AND shift(147))) XOR shift(10));
      shift(191) <= shift(198) XOR LOAD_EN AND (((shift(59) NAND shift(52)) XOR shift(11)));
      shift(187) <= shift(198) XOR LOAD_EN AND ((shift(12) AND shift(121)))\n      -- Output function
      z(0) <= (shift(10) XOR shift(99)) XOR (shift(137) XOR shift(227));
      z(1) <= (shift(122) XOR shift(187)) XOR (shift(245) AND shift(217));
      z(2) <= (shift(247) AND shift(231)) XOR (shift(215) AND shift(235));
      z(3) <= (shift(255) AND shift(251)) XOR (shift(186) AND shift(239));
      z(4) <= (shift(174) AND shift(44)) XOR (shift(304) AND shift(23));
      z(5) <= (shift(255) AND shift(24)) AND shift(243)) AND (shift(23) AND shift(174));
      z(6) <= (z(0) XOR z(1)) XOR (z(2) XOR z(3));
      z(7) <= (z(4) XOR z(5));
      z(8) <= (z(6) XOR z(7)) AND INIT;
      END IF;
   END PROCESS;

END ARCHITECTURE;
--Asynchronous output bit
Zout <= (z(6) XOR z(7));

END Behavioral;