Flexible and Compact Multi-Algorithm Cryptographic Engine

João Carlos Cabrita Resende

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors:  Prof. Ricardo Jorge Fernandes Chaves
              Prof. Leonel Augusto Pires Seabra de Sousa

Examination Committee
  Chairperson:  Prof. Nuno Cavaco Gomes Horta
  Supervisor:   Prof. Ricardo Jorge Fernandes Chaves
  Members of the Committee:  Prof. Fernando Manuel Duarte Gonçalves

October 2014
Acknowledgments

By concluding this presented thesis and achieving my Master Degree, I am also closing a full chapter of my life. If fate allows it, I will continue finding new sciences to learn but at a very different pace than the student life I have grown accustomed to this point. For this, I would like to acknowledge my sincere appreciations to everyone that stood along side me through this years of highs and lows.

Thank you to my family, those still here and the few ones already gone. They always seemed to be proud of me no matter how small or grand a feat I would achieved.

Thank you to all the friends I had the honour and joy of knowing through the years, even those from which I've parted ways long ago but never forgot. Even if few, any excitement in my life exists because of them.

I would like to specially thank Prof. Ricardo Chaves for being my guide in this final project. His dedication not only help me conclude this thesis, but also motivated me to search new interests and pursue them in the future with the best of my capabilities.

Finally, but far from least, my greatest, deepest gratitude, for which I'll be forever in debt, goes to my mother Helena and my father Carlos. No one else has given me so much, and everything that I have is, one way or another, thanks to them. May this thesis be a shy symbol of their efforts to successfully put me trough all my studies, and may it also mark my own future efforts to give to them everything that they might need, has they have given to me.
Resumo

A presente tese tem como objectivo o desenvolvimento de um circuito multi-cRIPTOGÁFICO para o qual foram considerados o standard de encriptação AES e a cifra de baixo recurso CLEFIA. Ambas são cifras simétricas para blocos de 128 bits e partilham de algumas técnicas de encriptação como substituição de bytes e matrizes de difusão, que podem ser implementadas por consulta de tabelas do tipo TBox. As estruturas dedicadas e mais eficientes do Estado de Arte apenas permitem o processamento de um único algoritmo criptográfico. A estrutura de dados compacta aqui apresentada é uma arquitectura “pipeline”, com 32 bits de largura, de ronda única recursiva (“rolled round”), e os resultados comprovam que implementações eficientes de dupla-cifra são possíveis graças a uma partilha cuidada de componentes dedicados de FPGA, como BRAMs de 36Kb e “shift registers”. Com agendamento adequado, ritmos de processamento de 800 Mbps e 1 Gbps são alcançados para AES e CLEFIA, respectivamente. Estas métricas são alcançadas com um custo de 119 SLICEs e 3 BRAMs numa Xilinx Virtex 5 FPGA, funcionando numa frequência máxima de 337 MHz, resultando numa eficiência de 6.71 Mbps/SLICE. Em comparação com o Estado de Arte, a eficiência de AES está entre 12% melhor e 18% mais pesada que as melhores propostas dedicadas para AES, enquanto que o CLEFIA impõe um custo maior de 16% em relação ao Estado de Arte dedicado.

Palavras-chave: AES, CLEFIA, FPGA, Compacto, TBox, BRAM
Abstract

The current thesis has the objective of developing a multi-cryptographic circuit considering the AES encryption standard and CLEFIA lightweight cipher. Both are symmetric 128-bit block ciphers and share encrypting techniques such as byte substitution and diffusion matrices, which can be implemented through TBox-type lookup tables. The most efficient and dedicated structures in the State of the Art only allow for the processing of a single encryption algorithm. Herein, a compact, 32-bit wide, rolled round architecture is proposed. The obtained results suggest that efficient dual-cipher designs can be achieved through careful resource sharing and an adequate usage of FPGA dedicated components, such as 36Kb BRAMs and addressable shift registers. With adequate scheduling, throughputs up to 800 Mbps and 1 Gbps, for AES and CLEFIA ciphers respectively, can be reached. These metrics are achieved at a cost of 119 SLICEs and 3 BRAMs on a Xilinx Virtex 5 FPGA, operating at a maximum frequency of 337 MHz, resulting in an efficiency metric of 6.71 Throughput/SLICE. In comparison with the related State of the Art, AES efficiency is between 12% better and 18% more costly than the best standalone AES proposals can be achieved, while CLEFIA imposes a higher cost of 16% regarding the best standalone CLEFIA in the State of the Art.

Keywords: AES, CLEFIA, FPGA, Compact, TBox, BRAM
# Contents

Acknowledgments ................................................................. iii
Resumo ........................................................................ v
Abstract ........................................................................ vii
List of Tables ................................................................ xi
List of Figures ................................................................ xiii
List of Acronyms ................................................................. xiv

1 Introduction ................................................................. 1
  1.1 Proposal and Methodology ........................................ 1
  1.2 Document organization ........................................... 2

2 Considered Ciphers .................................................. 3
  2.1 The Advanced Encryption Standard .......................... 3
    2.1.1 “Sub_Bytes” Substitution Table (“SBox”) ........ 4
    2.1.2 “Shift_Rows” Operation .................................. 5
    2.1.3 “Mix_Columns” Operation ............................... 5
    2.1.4 “Add_Round_Key” Operation ......................... 6
    2.1.5 Key Scheduler ............................................. 6
  2.2 CLEFIA ................................................................. 8
    2.2.1 The F-functions ............................................ 9
    2.2.2 S-function’s Substitution Tables (“SBoxes”) .... 10
    2.2.3 Feistel Word Swap ...................................... 11
    2.2.4 Key Scheduler ........................................... 11

3 State of the Art .......................................................... 13
  3.1 Introduction to FPGAs ............................................. 13
    3.1.1 FPGAs and block ciphers ............................. 15
  3.2 State of the Art in AES ........................................... 15
    3.2.1 Chodowiec and Gaj [2003] Shift Register ........ 15
    3.2.2 Rouvroy et al. [2004] TBoxes ......................... 17
    3.2.3 Chaves et al. [2006] Last Round ...................... 18
    3.2.4 Drimer et al. [2010] DSPs implementation ....... 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.5</td>
<td>Bulens et al. [2008] LUT-based SBoxes</td>
<td>20</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Liu et al. [2013] Unrolled and El Maraghy et al. [2013] Rolled Pipelines</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>State of the Art in CLEFIA</td>
<td>21</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Kryjak and Gorgon [2009] Unrolled Pipeline</td>
<td>21</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Proença and Chaves [2011] Compact Circuit</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>State of the Art Conclusions</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Designing the Dual-cipher Data Path</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Using the “Shift Register” for CLEFIA Feistel Word Swap and last round</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>TBoxes into BRAMs</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Critical Path Reduction</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Conclusions of the data path general design</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>Implementation details and improvements</td>
<td>31</td>
</tr>
<tr>
<td>5.1</td>
<td>Critical Path and Propagation Delays</td>
<td>31</td>
</tr>
<tr>
<td>5.2</td>
<td>Output Multiplexer Removal</td>
<td>33</td>
</tr>
<tr>
<td>5.3</td>
<td>Final Data Path and Details</td>
<td>33</td>
</tr>
<tr>
<td>5.4</td>
<td>Pipeline Scheduling</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>Control Unit and Final Circuit</td>
<td>38</td>
</tr>
<tr>
<td>5.6</td>
<td>Dismissed Alternatives</td>
<td>41</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusions on the Dual-Cipher implementation</td>
<td>44</td>
</tr>
<tr>
<td>6</td>
<td>Result Analysis</td>
<td>45</td>
</tr>
<tr>
<td>6.1</td>
<td>Dual-cipher Performance Analysis</td>
<td>45</td>
</tr>
<tr>
<td>6.2</td>
<td>State of the Art Comparison</td>
<td>46</td>
</tr>
<tr>
<td>6.3</td>
<td>Result Analysis Conclusion</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions</td>
<td>51</td>
</tr>
<tr>
<td>7.1</td>
<td>Future Work</td>
<td>52</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>A</td>
<td>FPGA Components Schematics</td>
<td>55</td>
</tr>
<tr>
<td>B</td>
<td>Detailed CLEFIA Pipeline Scheduling</td>
<td>58</td>
</tr>
<tr>
<td>C</td>
<td>Detailed AES Pipeline Scheduling</td>
<td>60</td>
</tr>
<tr>
<td>D</td>
<td>Detailed Co-processor Signal Mapping</td>
<td>62</td>
</tr>
<tr>
<td>E</td>
<td>Alternative modified Mix_Columns</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Tables

2.1 State Matrix distribution .................................................. 4
2.2 AES Key Scheduler features ................................................ 7
2.3 CLEFIA Key Scheduler features .......................................... 11
3.1 Chodowiec and Gaj [2003] Shift Register Schedule .................. 16
5.1 Performance Study of FPGA logic levels ................................. 32
5.2 CLEFIA Encryption Pipeline Scheduling ............................... 37
5.3 AES Pipeline Scheduling .................................................... 37
6.1 Resource usage of the Dual-Cipher Co-processor ..................... 45
6.2 Performance Study of the Dual-Cipher Co-processor ................. 46
6.3 State of the Art Performance Comparison .............................. 48
B.1 Shift Register Scheduling in CLEFIA Encryption ..................... 58
B.2 CLEFIA Encryption Pipeline Scheduling ............................... 59
C.1 Shift Register Scheduling in AES ........................................ 60
C.2 AES Pipeline Scheduling .................................................... 61
D.1 Co-processor Signal Mapping ............................................. 62
D.2 Co-processor Signal Mapping ............................................. 63
# List of Figures

2.1 The AES Data Path .................................................. 4  
2.2 Key Scheduler for 128-bit AES Key ................................ 7  
2.3 The CLEFIA Data Path for a 128-bit Key ......................... 8  
2.4 CLEFIA F-Functions ............................................... 9  
3.1 CLB SLICE column distribution .................................. 14  
3.2 Chodowiec and Gaj [2003] AES Data Path .......................... 15  
3.3 SRL16 LUT mode .................................................. 16  
3.4 Rouvroy et al. [2004] AES Architecture ........................... 17  
3.5 Chaves et al. [2006] AES Data Path .............................. 18  
3.6 Drimer et al. [2010] AES Data Path .............................. 19  
3.7 Bulens et al. [2008] AES Architecture ........................... 20  
3.8 Proença and Chaves [2011] CLEFIA Data Path .................... 22  
4.1 Core Architecture for the AES/CLEFIA Data Path ................ 26  
4.2 Output circuit for the AES/CLEFIA Data Path .................... 26  
4.3 Improved Feistel Word Swap ..................................... 28  
4.4 Separated Critical Path .......................................... 29  
5.1 Data Path of the Dual-Cipher architecture ....................... 32  
5.2 Detailed View of the Improved Output Stage ..................... 33  
5.3 Detailed View of the Forwarding and Memory Stage ............. 34  
5.4 The Dual-Cipher Co-Processor .................................. 38  
5.5 Flowchart of State Machine for Control Unit .................... 39  
5.6 Keys Storage input logic ....................................... 40  
5.7 Shift Rows through temporary registers ........................ 41  
5.8 Modified Mix Columns during AES Decryption ................... 43  
A.1 Virtex 5 SLICEL schematic ..................................... 55  
A.2 Virtex 5 SLICEM schematic ..................................... 56  
A.3 Virtex 5 BRAM schematic ....................................... 57  
E.1 Modified Mix Columns during AES Encryption ................... 64
E.2 Modified Mix_Columns during AES Decryption ........................................ 65
E.3 Modified Mix_Columns during CLEFIA F0-function .................................. 65
E.4 Modified Mix_Columns during CLEFIA F1-function .................................. 66
## List of Acronyms

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard.</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit.</td>
</tr>
<tr>
<td>BRAM</td>
<td>Block of Random Access Memory.</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher-Block Chaining.</td>
</tr>
<tr>
<td>CFB</td>
<td>Cipher Feedback.</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block.</td>
</tr>
<tr>
<td>CRYPTREC</td>
<td>Japanese Cryptographic Research and Evaluation Committees.</td>
</tr>
<tr>
<td>CU</td>
<td>Control Unit.</td>
</tr>
<tr>
<td>DPath</td>
<td>Data Path.</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor.</td>
</tr>
<tr>
<td>ECB</td>
<td>Eletronic CodeBook.</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standards.</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array.</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission.</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization.</td>
</tr>
<tr>
<td>KStore</td>
<td>Keys Storage.</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-Up Table.</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology.</td>
</tr>
<tr>
<td>PCBC</td>
<td>Propagating Cipher-Block Chaining.</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>
<tr>
<td>XOR</td>
<td>eXclusive OR.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Cryptography is the science that studies and develops forms for secret communications. History is filled with cases where private information needed to be concealed from strangers, all the way from Ancient Greece, Renaissance Italy, both World Wars to today’s digital world. With the rapid evolution of digital communications and its expansion into everyday electronics, ensuring the safety and privacy of information has become a permanent concern for industrial and academic works, where the usage of encryption systems has become one of the most widespread requirements.

In 2001, the U.S. National Institute of Standards and Technology (NIST) published FIPS-197, a document that certified the Rijndael algorithm as the Advanced Encryption Standard (AES) and the replacement for the older Data Encryption Standard (DES).

Rijndael was selected for AES, among several competing candidates, due to its balanced efficiency in the areas of security, complexity and speed, but many other alternative ciphers exist with their own dedicated purposes. One of this alternative algorithms is the 2007’s cipher CLEFIA, proposed by the SONY corporation in Shirai et al. [2007]. Originally designed for Digital Rights Management (DRM) purposes, this algorithm has become notable for its lightweight efficiency, being declared an International Standard in ISO/IEC 29192-2 in 2012, and more recently a Candidate Recommended Cipher by the Japanese CRYPTREC in their 2013 revision.

In a constantly connected World, electronic devices would greatly benefit if they could perform multiple ciphers for varying needs, but different ciphers require different resources and implementing them in the same device is often costly or inefficient.

1.1 Proposal and Methodology

The main goal of this work is to show that efficient resource re-utilization can be used to combine different ciphers into dedicated cryptographic engines. To achieve this, State of the Art compact structures for AES (Chodowiec and Gaj [2003], Rouvroy et al. [2004], Chaves et al. [2006], Bulens et al. [2008]) and CLEFIA (Kryjak and Gorgon [2009], Proença and Chaves [2011]) algorithms will be considered due to some of their similarities.
Both AES and CLEFIA are symmetrical block ciphers sharing ciphering techniques such as diffusion matrices and byte substitution (or SBoxes). The major differences reside on the fact that CLEFIA is based on a Feistel structure, while the AES algorithm is based on a “Shift Rows” operation, so minimizing their impact in a merged architecture is an inadvertent challenge.

In order to provide a proof of concept, Field Programmable Gate Arrays (FPGA) technology is considered, given their increasing deployment in embedded systems, adaptability, and ease of prototyping. The obtained experimental results suggest that by adequately reusing the BRAMs in the TBox implementation, with a LUT based addressable shift register, and an adequate logic mapping into LUTs, an efficient mix AES/CLEFIA FPGA implementation can be accomplished. At a cost of 119 Slices and 3 BRAMs, throughputs between 800 Mbps and 1 Gbps can be achieved, with efficiency metrics identical to the related compact, single algorithm, State of the Art.

1.2 Document organization

In Chapter 2, the considered ciphers, namely the AES and CLEFIA algorithms, are introduced. The existing State of the Art is reviewed in Chapter 3, with a particular focus on FPGA implementation. Chapter 4 depicts the main data path of the solution herein proposed, while Chapter 5 discusses the implementation details and the remaining dual-cipher components. The result analysis and performance comparison with the State of the Art is presented in Chapter 6. Finally, Chapter 7 concludes this document with some final remarks and future work suggestions.
Chapter 2

Considered Ciphers

Several encryption algorithms have been developed throughout time, with different security requirements.

Symmetric ciphers encrypt the information through the use of a secret key, and can only be decrypted with that same key, unlike asymmetric ciphers where encryption and decryption keys are different.

Block ciphers encrypt fixed size data, which requires the information to be divided into fixed size blocks. Most of the modern block ciphers are iterative. In other words, during the encryption process the data suffers the same transformation throughout a predetermined number of rounds.

For this thesis, the encryption standard AES was selected for the proposed multi-cryptographic engine, as it is widely employed through numerous applications. As an alternative cipher, the CLEFIA algorithm was selected due to its lightweight features and fair security, which makes it an excellent candidate for proof of concept of multi-cryptography architectures.

Either AES or CLEFIA are symmetric block ciphers that operate with data blocks of 128 bits (16 bytes), and 128, 192 or 256 bits sized keys. In the case of symmetric block ciphers it is possible to use different keys to cipher each block, as long as the respective key is used to decipher each one.

In order to familiarize the reader with the specifications of AES and CLEFIA algorithms, the following presents the main operations in both algorithms.

2.1 The Advanced Encryption Standard

The Advance Encryption Standard, or AES, is a symmetric block cipher made official by the NIST in 2001. It is a subgenre of the Rijndael algorithm, developed by John Daemen and Vincent Rijmen, with the difference that AES is purposed for 128-bit blocks, while Rijndael is configured to accept additional block sizes (192 and 256 bits).

In the beginning of the algorithm, the 128 bits of original text are distributed in a 4x4 matrix, byte wise and columnwise (as depicted in Table 2.1), followed by a process of 10, 12, or 14 rounds of transformations (as depicted in Figure 2.1), depending if the original key is 128, 192, or 256 bits long. The resulting matrix at the beginning/end of each round is called a State, in which the first and last States correspond
to the original and ciphered text.

<table>
<thead>
<tr>
<th></th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₀₀</td>
<td>a₀₁</td>
<td>a₀₂</td>
<td>a₀₃</td>
<td></td>
</tr>
<tr>
<td>a₁₀</td>
<td>a₁₁</td>
<td>a₁₂</td>
<td>a₁₃</td>
<td></td>
</tr>
<tr>
<td>a₂₀</td>
<td>a₂₁</td>
<td>a₂₂</td>
<td>a₂₃</td>
<td></td>
</tr>
<tr>
<td>a₃₀</td>
<td>a₃₁</td>
<td>a₃₂</td>
<td>a₃₃</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: State Matrix distribution

![Figure 2.1: The AES Data Path](image)

The outcome of each operation is only dependable of their input State, except for Add_Round_Key that depends on the State and the respective Round Key. As can be noted, the final round is slightly different as it does not execute the Mix_Columns transformation.

### 2.1.1 “Sub_Bytes” Substitution Table (“SBox”)

The byte substitution transformation, Sub_Bytes, is a non-linear function that operates in each byte of the State matrix independently. It replaces each byte by another one in accordance to an indexed table. This table is created by firstly taking the multiplicative inverse of the input byte, in the finite field GF($2^8$)*, and secondly by applying the following affine transformation in GF(2):

*AES GF($2^8$) irreducible polynomial is: $x^8 + x^4 + x^3 + x + 1$, or ‘0x11B’ in hexadecimal representation
The possible results of the SubBytes operation define 16x16 addressable table named SBox.

While decrypting AES, the non-linear function transformation of SubBytes needs to be reversed, InvSubBytes, but just like the encryption counter-part, all inverted solutions can be mapped on an equally sized InvSBox.

2.1.2 “ShiftRows” Operation

The ShiftRows operation consists of left round shifting the 2°, 3°, and 4° line of the State matrix by one, two and three positions.

Inverting the ShiftRows, InvShiftRows, simply consists of reverting the shift process by right round shifting the 2°, 3°, and 4° line of the State matrix by one, two and three positions.

2.1.3 “MixColumns” Operation

The MixColumns transformation is a linear matrix multiplication within a finite field GF(2^8). Consider each column of the State matrix a polynomial over GF(2^8), and multiply all of them by the following one:

\[ c(x) = 03x^3 + 01x^2 + 01x + 02 \]  \hspace{1cm} (2.2)

with the resulting matrix:

\[
\begin{bmatrix}
    b_{0i} \\
    b_{1i} \\
    b_{2i} \\
    b_{3i}
\end{bmatrix} = \begin{bmatrix}
    02 & 03 & 01 & 01 \\
    01 & 02 & 03 & 01 \\
    01 & 01 & 02 & 03 \\
    03 & 01 & 01 & 02
\end{bmatrix} \begin{bmatrix}
    a_{0i} \\
    a_{1i} \\
    a_{2i} \\
    a_{3i}
\end{bmatrix}
\]  \hspace{1cm} (2.3)

The inverse of the MixColumns operation, InvMixColumns, is very similar and only changes the multiplicative polynomial to:

\[ d(x) = 0Bx^3 + 0Dx^2 + 09x + 0E \]  \hspace{1cm} (2.4)

\[
\begin{bmatrix}
    b_{0i} \\
    b_{1i} \\
    b_{2i} \\
    b_{3i}
\end{bmatrix} = \begin{bmatrix}
    0E & 0B & 0D & 09 \\
    09 & 0E & 0B & 0D \\
    0D & 09 & 0E & 0B \\
    0B & 0D & 09 & 0E
\end{bmatrix} \begin{bmatrix}
    a_{0i} \\
    a_{1i} \\
    a_{2i} \\
    a_{3i}
\end{bmatrix}
\]  \hspace{1cm} (2.5)
If Sub_Bytes and Mix_Columns are sequential they can be combined into a TBox. Unlike the SBox 1-to-1 byte, the TBox is a 1-to-4 bytes, since it represents a column-wise partial matrix multiplication.

\[
\begin{bmatrix}
    b_{0i} \\
    b_{1i} \\
    b_{2i} \\
    b_{3i}
\end{bmatrix} = \begin{bmatrix}
    02 & 03 & 01 & 01 \\
    01 & a_{0i} \oplus 02 & a_{1i} \oplus 03 & a_{2i} \oplus 01 \\
    01 & 01 & 01 & 03 \\
    03 & 01 & 02 & 02
\end{bmatrix} \begin{bmatrix}
    a_{0i} \\
    a_{1i} \\
    a_{2i} \\
    a_{3i}
\end{bmatrix}
\]

\[
\iff \begin{bmatrix}
    b_{0i} \\
    b_{1i} \\
    b_{2i} \\
    b_{3i}
\end{bmatrix} = T_0\{a_{0i}\} \oplus T_1\{a_{1i}\} \oplus T_2\{a_{2i}\} \oplus T_3\{a_{3i}\}
\]

(2.6)

\[
\begin{bmatrix}
    b_{0i} \\
    b_{1i} \\
    b_{2i} \\
    b_{3i}
\end{bmatrix} = \begin{bmatrix}
    0E & 0B & 0D & 09 \\
    09 & a_{0i} \oplus 0E & a_{1i} \oplus 0B & a_{2i} \oplus 0D \\
    0D & 0B & a_{1i} \oplus 0E & a_{2i} \oplus 0B \\
    0B & 0D & 09 & 0E
\end{bmatrix} \begin{bmatrix}
    a_{0i} \\
    a_{1i} \\
    a_{2i} \\
    a_{3i}
\end{bmatrix}
\]

\[
\iff \begin{bmatrix}
    b_{0i} \\
    b_{1i} \\
    b_{2i} \\
    b_{3i}
\end{bmatrix} = IT_0\{a_{0i}\} \oplus IT_1\{a_{1i}\} \oplus IT_2\{a_{2i}\} \oplus IT_3\{a_{3i}\}
\]

(2.7)

Note that simply using TBoxes does not complete the Mix_Columns operation, since the 16 resulting bytes (4 per column) still need to be XORed into 4 bytes (next State column).

### 2.1.4 “Add_Round_Key” Operation

The Add_Round_Key operation is the simplest one, as it is only composed of bitwise XOR between the current State and the corresponding Round Key.

Each Round Key is 128 bits in length with a total of Nr+1 128-bit values, where Nr is the number of rounds in the cipher. The Round Keys are created by expanding the original cipher key through key scheduling.

Because Add_Round_Key is simply composed of a XOR operator, it is its own inverse.

### 2.1.5 Key Scheduler

As described, the AES algorithm requires a 128-bit Round Key at the end of each round, making them a total of (Nr+1) Round Keys, where Nr is the number of rounds, the additional key is used before the first round. The Round Keys are samples extracted from an Expanded Key that derives from the original Cipher Key. The Expanded Key is a linear array of 4-byte words, W[i], and has a total number of 4x(Nr+1) words, where each word constitutes 32 bits of the Round Key, as depicted in Table 2.2.
The Key Schedule for expanding the key always starts by affixing the Cipher Key as the first 4, 6, or 8 words of the Expanded key. For a 128-bit key, four sets of 32-bit keys are calculated through 10 rounds of key scheduling, as depicted in the example of Figure 2.2.

Figure 2.2: Example of a Key Scheduler for 128-bit AES Key (i from 1 to 10)

The Sub_Byte operation is the same as the one described in the encryption process, while the RotByte is a simple byte-wise left round shift of a 4-Byte word (RotByte(a,b,c,d) = (b,c,d,a)). The RConst parameter is a 4-Byte word where only the first byte varies between a fixed set of constant values:

- \( \text{RConst}[i] = (rconst[i], 0x00, 0x00, 0x00) \)
- \( rconst[0] = 0x01; \)
- \( rconst[0<i<=7] = 0x02 \times rconst[i-1] \leq 0x80; \)

For a 192-bit key, the key scheduling round works with 6 arrays of 32-bit words, with the 6th one suffering the same extra transformation process. For a 256-bit key, the key scheduling round works with 8 arrays of 32-bit words, with the 8th one suffering the same extra transformation process, and the 4th one suffering an added Sub_Byte transformation.

When performing an AES decryption, the order in which the Round Keys are supplied needs to be inverted (from last to first), and each one transformed through the Inv_Mix_Columns operation, as depicted at the top of Figure 2.2.
The need to transform the Round Keys between encryption and decryption reflects the fact that the AES has an asymmetric cipher structure, or in other words, the plain text is not obtained from the cipher text through the exact inverted order of transformations used in encryption. It is possible to change the standard so that a symmetrical structure is achieved, but this causes the sequence of operations to be different from encryption to decryption and is rarely considered.

The Key Schedule can be executed in parallel to the encryption process (on-the-fly) with only the previous and current Round Keys stored. The Inv Key Schedule does not have this advantage, since the entire Expanded Key is required before the inverting process can occur (Inv_Round_Key). This has led to a preference in the State of the Art to implement the Key Schedule and Inv Key Schedule before the ciphering process (pre-computed), keeping all (Inv) Round Keys stored.

2.2 CLEFIA

The CLEFIA algorithm is a 128-bit symmetric block cipher with a 4-branched Feistel network structure, and was introduced by SONY corporation in 2007. The initial 128-bit plain text (16 bytes) is arranged into an array of four 32-bit words. An initial Whitening process is performed in the second and fourth word. After that, all 128-bits are submitted to 18, 22 or 26 rounds of transformations, depending on the Cipher Key size of 128, 192 or 256 bits, as depicted in Figure 2.3.

![Figure 2.3: The CLEFIA Data Path for a 128-bit Key](image)

The name State will be used to define the 128-bit result at the beginning/end of each round, or four 32-bit words. This nomenclature is not used in the original CLEFIA definition, but it will occur in this work.
for the purpose of drawing similarities with AES, since each column of the AES State matrix is also a
32-bit word.

The input variables RK[i] represent a 32-bit Round Key, and each round requires two of them. All
CLEFIA rounds perform the same operations except for the last one, where the word swap needs to be
bypassed or cancelled (see further) before the last Whitening Keys are added.

2.2.1 The F-functions

In CLEFIA two types of non-linear functions exist, F-function_0 and F-function_1 (F0 and F1 for short).
Both require two 32-bit inputs (32-bit X and 32-bit RK) and output another 32 bits (Y). Each round
executes both functions but in different sides of the Feistel structure, which means that their execution
and sequence is independent of one another within the same round.

Each F-function (depicted in Figure 2.4) starts by XORing both its inputs into a 4-byte result, which
then replaces each byte by another through an alternating non-linear function (S0 and S1). After the sub-
stitutions are performed a new 4-byte word is attained and then multiplied, in GF(2^8), by a corresponding
Diffusion Matrix (M0 or M1).†

Figure 2.4: CLEFIA F-Functions

The equivalent expressions can be written has follows:

F0: \( r = M_0.S(x \oplus RK) \)

\[
\begin{bmatrix}
  r_0 \\
  r_1 \\
  r_2 \\
  r_3
\end{bmatrix} =
\begin{bmatrix}
  01 & 02 & 04 & 06 \\
  02 & 01 & 06 & 04 \\
  04 & 06 & 01 & 02 \\
  06 & 04 & 02 & 01
\end{bmatrix}
\begin{bmatrix}
  S_0(x \oplus k_0) \\
  S_1(x \oplus k_1) \\
  S_0(x \oplus k_2) \\
  S_1(x \oplus k_3)
\end{bmatrix}
\]

(2.8)

\[
M_0 \text{ polynomial : } m_0(x) = 06x^3 + 04x^2 + 02x + 01
\]

(2.9)

†CLEFIA GF(2^8) irreducible polynomial is: \( x^8 + x^4 + x^3 + x^2 + 1 \), or ‘0x11D’ in hexadecimal representation
\( F_1: r = M_1.S(x \oplus RK) \)

\[
\begin{bmatrix}
  r_0 \\
  r_1 \\
  r_2 \\
  r_3
\end{bmatrix} =
\begin{bmatrix}
  01 & 08 & 02 & 0A \\
  08 & 01 & 0A & 02 \\
  02 & 0A & 01 & 08 \\
  0A & 02 & 08 & 01
\end{bmatrix}
\begin{bmatrix}
  S_1(x_0 \oplus k_0) \\
  S_0(x_1 \oplus k_1) \\
  S_1(x_2 \oplus k_2) \\
  S_0(x_3 \oplus k_3)
\end{bmatrix}
\]

(2.10)

\[ M_1 \text{ polynomial} : m_1(x) = 0Ax^3 + 02x^2 + 08x + 01 \]

(2.11)

### 2.2.2 S-function’s Substitution Tables (“SBoxes”)

The substitution functions, S0 and S1, can be considered as two 1-to-1-byte SBox lookup tables or implemented through a more fine grained definition, just like AES SubBytes operation but slightly more complex.

The S0 function is defined by first separating the input byte into two 4-bit variables, and substituting each using smaller lookup tables of 16 outcomes, SS0 and SS1. Both results are then XOR with the GF(2⁴) multiplication of the other one by 0x2, before being once again replaced by one of two SS-Boxes, SS2 and SS3 (Shirai et al. [2007] or ISO/IEC [2012]).

The S1 function is less straight forward than the S0, because it involves two affine transformations and an inverse one, in GF(2⁸):

\[
y = S_1(x), y, x \in \{0, 1\}^8
\]

\[
y = \begin{cases} 
  g(f(x)^{-1}) & \text{if } f(x) \neq 0 \\
  g(0) & \text{if } f(x) = 0 
\end{cases}
\]

(2.12)

\[
f(x) \equiv \begin{bmatrix}
  000011000 \\
  010100001 \\
  000000001 \\
  000000110 \\
  011001011 \\
  010111000 \\
  011000000 \\
  100000001
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5 \\
  x_6 \\
  x_7
\end{bmatrix}
\]

(2.13)

\[
\begin{bmatrix}
  000011010 \\
  010000001 \\
  010111000 \\
  001000000 \\
  001100000 \\
  010010000 \\
  010000100 \\
  010000100
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5 \\
  x_6 \\
  x_7
\end{bmatrix}
\]

(2.14)
Because of their non-linear complexity, the S-funtions are often implemented as two distinct 16x16 SBoxes, or when adjoined with their respective diffusion matrices four 16x16x4 bytes TBoxes.

2.2.3 Feistel Word Swap

Since CLEFIA follows the design of an adapted 4-branched Feistel structure, every output of the F-function needs to be XOR with the corresponded parallel State word before the final resulting 128 bits are left-round shifted 32 positions (recall Figure 2.3). For decryption, the only structural difference in the data path is that, at end of each round, the 128 bits need to be right-round shifted by 32 bits.

2.2.4 Key Scheduler

The CLEFIA Key Scheduler is used to expand the original cipher key into an Expanded Key. All the required Round Keys for the algorithm are sampled from this Expanded key.

The CLEFIA Key Scheduler is particularly complex, as it takes more processing time to create the Round Keys than the time needed to cipher the plain text itself. Because of this, the process can not be implemented on-the-fly and must rather be previously computed.

If a 128-bit Cipher Key is used, the Key Scheduler starts by submitting the Cipher Key to 12 rounds of the same Feistel round-structure as the main data path. During the plain text ciphering, the F-functions are fed with Round Keys, but during the Key Scheduler they are fed with previously computed 32-bit constants.

After the 12 rounds are concluded four 32-bit words are obtained, and the Key Scheduler enters into a different iterative process of 9 rounds (with more constants). It is from this 9 rounds that a total of 36 Round Keys of 32 bits are extracted for the ciphering of the plain text.

Summarizing, the Key Scheduler requires 60 32-bit constants, previously calculated, and 21 iterative rounds for a 128-bit cipher Key. For the 192 and 256-bit cipher keys the Key Scheduler becomes even more complex with up to 92 constants and an 8-branched Feistel network, instead of the previous 4-branch (Shirai et al. [2007] or ISO/IEC [2012]).

<table>
<thead>
<tr>
<th>Cipher Key (bits)</th>
<th>Ciphering Rounds</th>
<th>Key Scheduler Rounds</th>
<th>Constants (32b)</th>
<th>Round Keys (32b)</th>
<th>Whitening Keys (32b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>12</td>
<td>12+9</td>
<td>60</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td>192</td>
<td>14</td>
<td>10+10</td>
<td>84</td>
<td>44</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>10+12</td>
<td>92</td>
<td>52</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2.3: CLEFIA Key Scheduler features
Chapter 3

State of the Art

Ever since AES was selected as the encryption standard in 2001, it has been subjected to deep scrutiny by the security community, considering several implementations targeting FPGA and ASIC technologies. As for CLEFIA, its relative novelty results in less referable implementations namely Sugawara et al. [2008] and Akishita and Hiwatari [2012] for ASIC and Kryjak and Gorgon [2009] and Proença and Chaves [2011] on FPGAs. Herein FPGA technology is considered given its flexibility and ease to prototype.

The following chapter starts by introducing some of the features of the FPGA technology, and the most common techniques used to implement block ciphers in this environment. Further, it is presented a State of the Art study for the most relevant works in AES implementations, followed by an equivalent study for CLEFIA.

3.1 Introduction to FPGAs

A Field-Programmable Gate Array (FPGA) consists of a post-manufactured, multi-configurable, integrated circuit, in contrast to the static Application-Specific Integrated Circuit (ASIC).

FPGAs are known for their high flexibility and non-scalable cost. They possess programmable logic blocks that are capable of performing any combinatorial logic function (such as AND, OR, XOR, etc.), but also possess reconfigurable interconnection paths that allows combinations of logic blocks to implement any arbitrary circuit.

In the past, FPGA technology had the disadvantage of being less efficient in area resources, power consumption and frequency performance then equivalent designs in ASIC. However, over the years the differentiation gap has been significantly reduced, with serious improvements in power, speed, cost, and re-configuration options (such as partial on-the-fly re-configuration). Dedicated ASIC circuits can still benefit from better overall efficiency levels, but the effort required in designing them and shorter life-span has given FPGAs an increasing weight in the digital market, such as embedded systems or prototyping.

For this work, the Xilinx Virtex board family will be considered, but other products exist such as
Altera's Stratix or Microsemi's ProASIC that can be targeted.

A Xilinx Virtex 5 is composed of several Configurable Logic Blocks (CLBs) and each one connects to the general routing matrix (the signal paths). Each CLB contains 2 SLICEs, arranged column-wise within the same fast carry chain, as depicted in Figure 3.1.

![Figure 3.1: CLB SLICE column distribution, as presented in Xilinx's Virtex 5 User Guide](image)

Inside every SLICE 4 configurable lookup tables (LUTs) exist, 4 memory components (Flip-Flops, FF) and extra selective logic, the latter not directly configurable by a designer.

The LUT is the most basic logic component of an FPGA. It works as an addressable lookup table where its contents simulate a logic gate operation. In older Virtex Series a SLICE only contained 2 LUTs that worked has 4-by-1-bit lookup table (and 1 FF), but with the increasing circuit complexity they started suffering from bottleneck issues, and Xilinx decided to improve LUTs to accept 6 logic inputs, starting with Virtex 5 and upward, while also increasing the number of LUTs and FFs to 4 within a single SLICE.

The most common SLICEs in an FPGA is the SLICEL, that performs mostly according the description so far. In lesser number, there exists a different type of SLICE named SLICEM. This SLICE contains the same elements as the previous one but offers extra configuration options, such as configuring its four LUTs to function as distributed addressable RAM, or a 16-bit deep addressable shift register (SRL16 mode). The last one in particular has a prominent role in compact State of the Art AES implementations.

A more coarse grain component in FPGAs is the configurable dual-port RAM block (BRAM) that is approximately the size of 5 CLBs (Drimer et al. [2010]). This memory block is one of the best features in FPGA technology since no equally efficient solution exists in ASIC. Older FPGAs only provided BRAMs of 4Kb or 18Kb of storage, where the most recent ones have a maximum of 36Kb. They have synchronous writing but optional synchronous/asynchronous reading, and can be configured with different input/output bit-widths.

Detailed schematics of SLICEL, SLICEM and BRAM can be found in Appendix A, as they appear in the XILINX Virtex 5 User Guide.
3.1.1 FPGAs and block ciphers

When considering block cipher implementations on dedicated architectures, the improvements in the State of the Art are mainly based on round folding, with unrolled or rolled structures; on the type of components used for the typical substitution operation, with dedicated logic or lookup tables; and with operation rescheduling (Rouvroy et al. [2004], Proença and Chaves [2011]).

One of the most direct ways to obtain a trade off between area and throughput is with round folding/expansion. In unrolled structures the round computation is expanded and pipelined. With this, multiple rounds of the algorithm can be executed independently and in parallel. These approaches are known for imposing higher area demands but, on the other hand, allowing for higher throughputs and working frequencies in particular cases (such as in ECB mode). When ciphering in feedback modes (such as CBC) with dependencies between blocks, the throughput improvements cannot be achieved. When folding the round computation, each round is performed in one or more clock cycles, resulting in more compact structures. Consequently, lower throughputs are typically achieved.

Another major implementation differentiation in the State of the Art is in the substitution operation. These vary from a fine grained implementation of the byte substitution (Logic based), to more coarse grained ones using equivalent lookup tables (Memory based). Logic based structures implement the byte substitution operations and matrix multiplications by hard-wiring their actual mathematical definition through logic components. They are cipher-specific and can be the most area efficient but usually the slowest to complete. Typically in FPGAs, the most efficient implementations to perform the byte substitution are lookup table based (Rouvroy et al. [2004]). These use multiple FPGA LUTs, or even BRAMs, to implement an equivalent lookup table for byte substitution, and in some cases, part of additional operations such as the matrix multiplication of AES, creating a so called TBox. Memory based approaches can lead to faster circuits at the cost of memory blocks.

3.2 State of the Art in AES

3.2.1 Chodowiec and Gaj [2003] Shift Register

One of the first compact implementations of AES in FPGAs is proposed by Chodowiec and Gaj [2003]. The authors consider the use, for the SubBytes operation, two embedded dual-port 512x8 bit...
BRAMs of a Xilinx Spartan II for a memory based SBox (and InvSBox) and then a logic design for the matrix multiplication \((c(x), a(x))\), as depicted in Figure 3.2.

However the most remarkable feature (and one that is herein considered) is the execution of AES ShiftRows operation through the use of the dedicated multi-addressable 16-bit deep shift registers of Xilinx’s boards. Some LUTs of Xilinx boards possess a special mode of operation, SRL16, where each acts as a 1-bit wide 16-bit deep addressable shift register, as depicted in Figure 3.3. Eight LUTs are require to store a byte in each clock cycle, and 32 LUTs are needed for a 32-bit datapath.

![Figure 3.3: SRL16 LUT mode](image)

If the entire State is stored within the Shift Register (also known as Folded Register) they can extract any 16 bytes of a State in an order adequate of the ShiftRows operation (as depicted in Table 3.1). Due to the Shift Register behaviour, for every clock cycle passed, the State data advances within the Shift Register positions, leaving the initial 32 bits ready for overwriting, which is then used to start writing the first column of the next State. Because a 16-bit deep Shift Register can be configured from a SLICEM LUT, this technique allows for a fair reduction in logic resources (only 32 LUTs) when implementing the ShiftRows and InvShiftRows operations within the same circuit.

Chodowiec and Gaj [2003] also use integrated asymmetric decryption with AddRoundKey and InvMixColumns executed in this order, bypassing the latter in the last round. For InvMixColumns Chodowiec and Gaj [2003] apply the second power of the decryption polynomial over the encryption one:

\[
c(x) \ast d(x) = 01 \Leftrightarrow c(x) \ast d^2(x) = d(x)
\]

\[
d^2(x) = 04 \cdot x^2 + 05
\]

<table>
<thead>
<tr>
<th>Shift Register Occupation</th>
<th>A_{12} A_{13} A_{14} A_{15}</th>
<th>B_0 B_1 B_2 B_3</th>
<th>B_4 B_5 B_6 B_7</th>
<th>B_8 B_9 B_{10} B_{11}</th>
<th>B_{12} B_{13} B_{14} B_{15}</th>
<th>C_0 C_1 C_2 C_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X X X X X X X X X X X X X X</td>
<td>A_8 A_9 A_{10} A_{11}</td>
<td>B_0 B_1 B_2 B_3</td>
<td>B_4 B_5 B_6 B_7</td>
<td>B_8 B_9 B_{10} B_{11}</td>
<td>B_{12} B_{13} B_{14} B_{15}</td>
<td>C_0 C_1 C_2 C_3</td>
</tr>
<tr>
<td>X X X X X X X X X X X X X X</td>
<td>A_8 A_9 A_{10} A_{11}</td>
<td>B_0 B_1 B_2 B_3</td>
<td>B_4 B_5 B_6 B_7</td>
<td>B_8 B_9 B_{10} B_{11}</td>
<td>B_{12} B_{13} B_{14} B_{15}</td>
<td>C_0 C_1 C_2 C_3</td>
</tr>
<tr>
<td>X X X X X X X X X X X X X X</td>
<td>A_8 A_9 A_{10} A_{11}</td>
<td>B_0 B_1 B_2 B_3</td>
<td>B_4 B_5 B_6 B_7</td>
<td>B_8 B_9 B_{10} B_{11}</td>
<td>B_{12} B_{13} B_{14} B_{15}</td>
<td>C_0 C_1 C_2 C_3</td>
</tr>
</tbody>
</table>

Table 3.1: Chodowiec and Gaj [2003] Shift Register Schedule for encryption. In a 4-cycle round, a single byte needs to be forwarded each fourth round.
3.2.2 Rouvroy et al. [2004] TBoxes

The use of shift registers for the Shift_Rows operation is also considered in Rouvroy et al. [2004] for a Xilinx Spartan 3. The more recent technology brought larger BRAMs of 18Kb, allowing the authors to improve the memory occupation efficiency by using two of this dual-port BRAMs, in a 256x32 bit configuration, to implement the necessary TBoxes and make the circuit process 1/4 round per cycle, as depicted in Figure 3.4(a).

Recalling Section 2.1, the AES last round computation does not perform the (Inv)Mix_Columns operation, but this one is inherent to a TBox implementation. Two coefficients of the Mix_Columns polynomial are unitary (’0x01’), therefore equivalent to a simple Sub_Bytes transformation. Rouvroy et al. [2004] use this detail to bypass the Mix_Columns in the last round. However, the Inv_Mix_Columns polynomial does not possess any unitary values, so the authors decide to map an extra Inv_SBox into the BRAMs, in order to only perform the Inv_Sub_Bytes operation at the last decrypting round.

The AES TBox technique allowed for Rouvroy et al. [2004] work to be one of the most compact architectures in State of the Art, allowing embedded key schedule (Figure 3.4(b) ) and ciphering/deciphering modes, but the extra required selective logic for the last round, depicted in Figure 3.4(c) (after the 2 RAM blocks), made the critical path its biggest disadvantage, affecting the overall performance.

![Figure 3.4: Rouvroy et al. [2004] AES Architecture](image-url)
3.2.3 Chaves et al. [2006] Last Round

Another work using the TBox implementation is Chaves et al. [2006] for Xilinx Virtex II-Pro (with 18Kb BRAMs). Their work is slightly more extensive than simple AES implementation due to their focus on fully programmable co-processors, but their main 128-bit data path still presents some interesting details.

The 128-bit data path (depicted in Figure 3.5) allows to address the entire State matrix in parallel, so the Shift_Rows operation can be performed through hard-wired connection (or multiplexed path when both enc/dec modes are implemented), instead of using a shift register. A total of eight 18Kb BRAMs are used for TBox storage. Two BRAMs for each 32-bit word of the State. After this, the byte shift operation is hard-wired in order to align the (Inv)Mix_Columns coefficient before a 32-bit XOR tree to add the Round Keys (depicted at the bottom of Figure 3.5). However, there is a difference between Chaves et al. [2006] and Rouvroy et al. [2004] relatively to the TBoxes mappings, since Chaves et al. [2006] does not map a single InvSBox into the BRAMs.

To solve the last AES round, Chaves et al. [2006] notices that by summing, in GF(2^8), all the four coefficients of a (Inv)Mix_Columns column, one obtains the unitary value of the input byte, either in encryption (3.3) or decryption (3.4). They implement this technique through four parallel 8-bit XOR trees.

\[
01 \cdot a_i = 03 \cdot a_i \oplus 01 \cdot a_i \oplus 01 \cdot a_i \oplus 02 \cdot a_i; \quad (3.3)
\]

\[
01 \cdot a_i = 0B \cdot a_i \oplus 0D \cdot a_i \oplus 09 \cdot a_i \oplus 0E \cdot a_i; \quad (3.4)
\]

Chaves et al. [2006] work also considers the use of off-chip Key Scheduling, in order to improve resource usage efficiency, as suggested by Sklavos and Koufopavlou [2002]. Two extra BRAMs are then used as local memory storage for the Round Keys.

Figure 3.5: Chaves et al. [2006] AES Data Path
3.2.4 Drimer et al. [2010] DSPs implementation

In 2008, Drimer et al. proposes an AES implementation, for Xilinx FPGAs, using Digital Signal Processing blocks (DSPs). This work would be revised and extended in Drimer et al. [2010].

DSP blocks are components specialized in arithmetic operations. They were originally introduced in Virtex 4 as 18bx18b integer multipliers, but have been progressively updated to also allow for 48 bit decimal sums and bitwise logic.

Drimer et al. goal was to create an AES architecture where the critical path is contained to a single FPGA component, allowing them to reach higher working frequencies, namely 550MHz on the Virtex 5 family. The authors use 2 BRAMs (with 36Kbits) to store encrypting and decrypting TBoxes, performing the following XOR operation (and Round Key addition) through a 4 cascaded DSP blocks (Figure 3.6). The Shift Rows operation is performed through custom design shift registers (not SRL16 LUT mode).

Given the DSPs usage, Drimer et al. are capable to significantly decreased the number of used LUTs. However, the resulting AES architecture is composed of 8 pipeline stages, becoming adequate for ciphering two block streams in non-feedback mode.
3.2.5 Bulens et al. [2008] LUT-based SBoxes

During the design of a new AES architecture, if memory based lookup tables are defined several options can be considered. The majority of the State of the Art considers BRAM-based TBoxes architectures, while Chodowiec and Gaj [2003] describe BRAM-based SBoxes. Bulens et al. [2008] proposes a different approach by using LUT-based SBoxes on a Virtex 5 with logic based MixColumns. In order to do this they take advantage of new technology feature, the F7MUX and F8MUX.

As described in Section 3.1, the most elementary logic unit of an FPGA is the LUT, where the upward component is the SLICE which contains four 6-input LUTs since Virtex 5. Among the SLICE extra configuration logic there exists two F7MUXs and one F8MUX. As the names imply this components are output multiplexers that allow for the combination of two or four 6-input LUTs within a SLICE to create logic functions of 7 or 8 inputs, with minor or no performance downgrade. With this feature, the entire SLICE can be used to create a 256x1bit lookup table, while 8 of them can be used to create a 256x8bit lookup table, being the latter the exact same size of an AES SBox. For Bulens et al. [2008] AES architecture with a 128-bit data path, a total of 16 SBoxes, 128 SLICEs and 512 LUTs are needed for the SubBytes operation.

The advantage of using LUTs as local memory tables, instead of dedicated BRAMs, is the fact that the former can perform their operation much faster, requiring between none to one clock cycle, while BRAMs are slower and force the need of one to two clock cycles. The disadvantage is that designing the equivalent memory capacity (and features) of a BRAM into LUTs is severely inefficient in area resources. This leads to a compromise from Bulens et al. [2008] where their implementations can only be design exclusively for encryption or decryption, requiring double resources for both modes.

Bulens et al. [2008] also extend the merging of a 128-bit Key Scheduling presented by Rouvroy et al.
Thanks to their larger data path width, they improve it by using a XOR mesh, rather than the iterative form of the latter.

### 3.2.6 Liu et al. [2013] Unrolled and El Maraghy et al. [2013] Rolled Pipelines

One of the fastest AES implementations in the State of the Art, for FPGA, is the unrolled 128-bit architecture proposed by Liu et al. [2013]. The authors used LUT-based SBoxes and logic MixColumns, but decided to separate the former ones into a single pipeline stage, arguing that the remaining operations combined were less resource demanding than lookup tables. It must be noted that ShiftRows is performed by hard-wired path, making the structure encoding-specific. This means that it either does encryption or decryption. The authors also present a modified Key Scheduler with two pipeline stages, parallel to the data path, for the expansion of a 128-bit cipher key. Several Virtex boards were tested, from the 4° to the 7° series, but the 64.1 and 66.1 Gbps of throughput in a Virtex 6 and 7, respectively, are two of the highest throughputs in the literature.

In contrast to Liu et al. [2013] unrolled round implementation, there is the rolled 128-bit architecture proposed by El Maraghy et al. [2013]. In this structure, every operation of AES is kept between sub-round pipeline registers (4 stages), with SubBytes performed by 8 dual-port, 256x8b configured, BRAMs of 36kb, and logic-based MixColumns. The authors also present one dedicated Key Scheduler, on-the-fly and encoding specific. For this, two extra dual-port BRAMs with four extra SBoxes are required, making a total 10 BRAMs (20 SBoxes) used in the entire architecture.

### 3.3 State of the Art in CLEFIA

#### 3.3.1 Kryjak and Gorgon [2009] Unrolled Pipeline

One of the first implementations of the CLEFIA algorithm in FPGA technology was proposed by Kryjak and Gorgon [2009], for Virtex 2-Pro, Virtex 4, and Virtex 5 devices. In their work, the authors present an unrolled 128-bit data path structure, for 128-bit Cipher Keys, while also considering four alternative approaches to the S-functions namely: logically implemented, SBox, TBox, and hybrid combination of them. The latter approach implements the S0 function by logic, while the S1 function is implemented by SBox lookup table, due to the GF($2^8$) inversion being more demanding in logic resources. All approaches use LUT based configurations, never resorting to BRAMs.

Tested on older technology, Kryjak and Gorgon [2009] show that no S-function implementation prevails over another, but in the more recent Virtex 5 the results suggest that the mixed approach is more efficient in resources and performance, with the TBox approach as the second best option.

Further details of this work include an independent dedicated Key Scheduler, for an initial expansion of 128-bit keys, and the lack of mid-round registers.
3.3.2 Proença and Chaves [2011] Compact Circuit

A compact CLEFIA implementation on FPGA is proposed by Proença and Chaves [2011]. Two rolled architectures are considered, namely with a 128 and a 32 bit wide data paths; while it also introduces the BRAM-based TBox strategy for CLEFIA.

The 128-bit architecture follows closely the CLEFIA round definition, with all branches of the Feistel network being processed in parallel. A total of four BRAMs (18Kb or 36Kb) are required for the 256x32b data storage.

The 32-bit architecture, depicted in Figure 3.5, is a more interesting approach, as it folds a single round to half, but still keeping the performance. This is achieved, since the 1º and 3º words of a round “State” are always being alternately fed to the two, 512x32b configured, BRAM-based TBoxes, while a parallel set of registers keeps the CLEFIA Word Swap properly scheduled at the end of each round. For future reference, this more parallel structure is named “CLEFIA Swap Module”.

Figure 3.8: Proença and Chaves [2011] CLEFIA Data Path
3.4 State of the Art Conclusions

The presented State of the Art research provided an insight of the most common FPGA implementation approaches. Unrolled round structures are more suitable for faster ciphering implementations, however they are only efficient in non-feedback modes where input blocks are independent from each other. Rolled round structures are more area-efficient in feedback modes, but require better tuning to achieve high throughputs.

Relating to block ciphers, several methods can be considered to perform the non-linear byte transformations, as hard-wired logic or addressable lookup tables: SBox and TBox. In FPGA technologies, a common practice is to use the dedicated memory components BRAM to implement Look Up tables, as well as specific implemented shift registers for AES ShiftRows operation. The next chapter describes the proposed dual-cipher design, considering the techniques here described.
Chapter 4

Designing the Dual-cipher Data Path

As stated before, the goal of this work is to design a compact AES and Clefia implementation on FPGA. The first step towards this is to obtained a functional design that combines the best State of the Art compact architectures from both ciphers. This model considers the use of a folded structure, computing each round in multiple iterations, with a data path of 32 bits and a TBox approach. Since the initial step for AES is to XOR the plain text with the first input keys, a 32-bit XOR is considered at the input stage, as depicted in point 1 of Figure 4.1. For CLEFIA, the first and third words of plain text do not need to be XOR, but we will assume for now that the input keys can be set to the zero value, externally.

For the AES ShiftRows operation, a byte addressable 32-bit wide and 16-bit deep Shift Register is considered in point 2 of Figure 4.1, given that this solution is the most compact one in State of the Art when using LUTs in the SRL16 LUT mode on Xilinx FPGA’s. This component is used to temporarily store and address the State. For CLEFIA, the Shift Register works as a simple redundant register buffer, since the forwarding stage 3 will be responsible for feeding the mid-round words. More efficiency improvements will be applied as discussed ahead.

To allow for a more efficient data path, a forwarding multiplexer is used to select the data fed to the TBoxes, as detailed in 3 of Figure 4.1. This block is also used to add Round Keys for CLEFIA, or zero values in AES.

The main computation is performed in 4, where the TBox lookup tables compute the byte substitutions and the coefficient multiplications. The remaining part of the matrices multiplication, the necessary GF($2^8$) additions, are performed by the 4-to-1 32-bit XOR tree in 5. Still in 5 the CLEFIA Swapped word may, or may not, be added depending on which cipher is being used. When not used the input of the MUX is set to zero. After the XOR tree in 5, the 32-bit data is fed back to 1 where AES Round Keys (or zeros) are now being added, terminating a ciphering round.

In parallel to the TBoxes path, there is the CLEFIA Swap Module X, introduced by Proença and Chaves [2011] for the purpose of maintaining a proper schedule for CLEFIA Feistel Swap operation.

Finally, a different XOR tree stage, depicted in Figure 4.2, is used to add all coefficient multiplications of the AES MixColumns, as described in (3.3) and (3.4) of Section 3.2.3, and add the last Round Key. The CLEFIA computation uses this stage to add the last Whitening Keys.
Figure 4.1: Core Architecture for the AES/CLEFIA Data Path

Figure 4.2: Output circuit for the AES/CLEFIA Data Path
4.1 Using the “Shift Register” for CLEFIA Feistel Word Swap and last round

As stated before, using a 32-bit wide 16-bit deep addressable Shift Register is a very compact and efficient solution while processing the AES algorithm, as it allows to temporarily store a State matrix and perform the Shift Rows operation altogether. However, when performing the CLEFIA cipher, the Shift Register is highly underused. It serves no data feeding purposes, as that function is already assured by the forwarding multiplexed path 3.

Because two pipeline stages were maintained within the TBoxes, as the original CLEFIA compact State of the Art, and the fact that the Shift Register still stores the State words, independently of the cipher, we can use it to simply supply the Feistel swapped words for addition at the end of the XOR tree. By doing so, it is possible to replace the CLEFIA Swap Module completely, reducing its five 32-bit register array into a single 32-bit Shift Register.

With this new implementation, the input data block to be de/encrypted using CLEFIA, needs to be supplied through the same input port. The second and fourth words need to be Whitened. This can be achieved in the same way as for AES, by supplying the input data block through the input port but Whitening every 32-bit words of the State.

As for the last round of CLEFIA, the last four resulting words do not need to be swapped. These words can be easily extracted, in any order, from the Shift Register, so a direct connection between the latter and the output circuit is performed. One consequence from this new design is an extra clock cycles. Because the Shift Register now becomes responsible for performing the Feistel Word Swap and supply the final State in the correct order, it needs an extra 6 cycles to conclude without data hazards: 2 extra cycles to conclude the final round, and the last 4 to eject the properly ordered State words (a more detailed discussion on this is presented in Section 5.4).

4.2 TBoxes into BRAMs

AES and CLEFIA require SBox operations followed by a matrix multiplication, which can be partially compressed into a TBox operation if a memory based implementation is adopted. In the AES algorithm the same SBox substitution is performed for all input Bytes. Following this, each Byte output is multiplied in GF(2^8) by the coefficients {3,1,1,2} for encryption and {B,D,9,E} for decryption. The result, four sets of 32 bits, are then shifted and added by a 32-bit XOR tree resulting in a 32-bit result. Given this, the minimum amount of memory required for the AES TBox (encryption and decryption) is (Chaves et al. [2006]):

\[ 2^{\text{modes}} \times 256_{\text{cases}} \times 32_{\text{bits}} = 16Kb \]  

(4.1)

The CLEFIA algorithm does not require a different computation for encryption and decryption, since it is based on a Feistel network. However, as described in Section 2.2, four different TBoxes are required in each round. This is due to the two different SBoxes (S0 and S1) and the two different MDS matrices.
Given this, the minimum amount of memory that the CLEFIA’s TBoxes require is:

\[ 2^{SBoxes} \times 2^{MDS} \times 256_{\text{combinations}} \times 32_{\text{bits}} = 32Kb \]  

(4.2)

4.3 Critical Path Reduction

The structure presented in Figure 4.3 (and Figure 4.2) depicts a functional dual-cipher circuit disregarding performance. This feature is determined (among other things) by the number of logic elements that exists in the data path of a single pipeline stage, where the longest path is known as the critical path. Both the Shift Register and TBoxes delimited a pipeline stage due to their inherent input registers, with the latter having an extra register at the output.

With this structure, the critical path is foreseen to reside between the 128-bit output of the TBoxes and the 32-bit input of the same component. In this path, a total amount of eleven different variables can be counted throughout three logic levels. By sacrificing the ability to directly forward the input plain text, from the input to the TBox in, the critical path can be reduced to only two logic levels. This
way the forward values are extracted directly after the XOR tree, but the capability of adding the AES Round Key to the forwarded value needs to be replicated, as depicted in Figure 4.4.

With this new structure, the path between the BRAMs output and Shift Register input has approximately the same critical path as the feedback circuit between the BRAMs output and the input. The parallel output circuit (Figure 4.2) has a smaller 7-to-1 logic, therefore it should have the path with the least impact in the frequency performance.

![Separated Critical Path](image)

**Figure 4.4: Separated Critical Path**

### 4.4 Conclusions of the data path general design

This chapter introduces the core structure for the proposed dual-cipher data path. The most predominant details of the 32-bit round structure is the compact use of an addressable Shift Register to temporarily store the mid-round State, and a memory based TBox implementation. The design was also improved by using the Shift Register to perform the AES ShiftRows operation and the CLEFIA Feistel Word Swap operation, while special caution was taken on defining the forwarding data path toward minimizing the critical path. In the next chapter, the physical implementation of this circuit is addressed along with the possible and proposed hardware improvements.
Chapter 5

Implementation details and improvements

This chapter presented the implementation detail and FPGA optimizations of the dual-cipher AES/CLEFIA structure proposed in the previous chapter.

For this implementation a Xilinx Virtex 5 FPGA is considered as the target technology. Implementation details for the Xilinx virtex 6 technology are also herein considered.

5.1 Critical Path and Propagation Delays

Implementing a circuit on an FPGA requires the configuration of LUTs but also of the data flow routing. According to the Xilinx documentation the propagation time through a single LUT is independent of the number of inputs that are being used (maximum 6, LUT6). However, the performed analysis, presented in Table 5.1, suggests variations in the maximum clock period if the LUT inputs are underused (used as LUT5 or LUT4). This means that, although a single LUT has an independent local performance, the routing of a different amount of signals has an impact on the global delay. The routing of less input signals into a LUT should, theoretically, be equal or faster than routing a larger number of signals, but in practice this strongly depends either on the software synthesis tool, or a designer’s experience in manually routing the logic components. Still, if a software synthesis tool is assumed, a performance enhancement should be more likely achieved if simpler logic levels are used. Additionally, considering the use of LUTs with less inputs allows for easier optimizations on older technology (such as the Xilinx Virtex 4).

For the proposed dual-cipher structure, the critical path remains the same even with only two logic levels, due to the implemented routing. Still, it is possible to make this data path use only LUT4s, further improving its efficiency. For this, the XOR tree after the BRAMs is isolated, and the CLEFIA word adding is passed onto the next logic levels. The feedback path to the Shift Register becomes selective between adding Round Keys or the CLEFIA swapped word, while the forwarding path is slightly modified in the XOR component, as depicted in Figure 5.1.
<table>
<thead>
<tr>
<th>Logic Levels</th>
<th>xc6vlx240t-3</th>
<th>xc5vlx30t-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT4</td>
<td>0.714</td>
<td>0.838</td>
</tr>
<tr>
<td>LUT5</td>
<td>0.768</td>
<td>0.963</td>
</tr>
<tr>
<td>LUT6</td>
<td>0.806</td>
<td>1.039</td>
</tr>
<tr>
<td>LUT6-LUT6</td>
<td>1.223</td>
<td>1.289</td>
</tr>
<tr>
<td>LUT5-LUT6</td>
<td>0.996</td>
<td>1.128</td>
</tr>
<tr>
<td>LUT4-LUT6</td>
<td>1.075</td>
<td>1.074</td>
</tr>
<tr>
<td>RAMB36</td>
<td>1.519</td>
<td>1.545</td>
</tr>
</tbody>
</table>

Table 5.1: Logic level impact on the Clock period (ns) for Virtex6 and Virtex5 boards

Figure 5.1: Data Path of the Dual-Cipher architecture
5.2 Output Multiplexer Removal

As described in the previous chapter, the output circuitry, depicted in Figure 4.2, is a 7-to-1 32-bit function. This function does not fit on a LUT6, requiring a second set of LUTs just to implement the multiplexer. In order to improve the resource requirement, the existing multiplexer can be replaced by a two-entry XOR operator, as depicted in Figure 5.2. In order to maintain proper functionality, the TBoxes need to output zero values during the last cycles of CLEFIA. Similarly, zero values need to be outputted by the Shift Register during the last round of AES. The first case is easily done since the TBoxes outputs are registered and their values can be reset by the control logic. The second case is more complex since the Shift Register has asynchronous reading and its output is directly dependent of its contents. In Section 5.4, the pipeline scheduling and data path is discussed, presenting the solution for this problem.

![Diagram of improved output stage]

Figure 5.2: Detailed View of the Improved Output Stage

5.3 Final Data Path and Details

At this point, the final data path for the proposed dual-cipher architecture is already defined as depicted in Figure 5.1 and 5.2. These figures will now be addressed and referred to in order to explain the fine-grained details of the proposed design.

Block ① is composed of 32 LUT6s and is responsible for adding the AES Round Keys (KeyA input) or the CLEFIA swapped words. The Shift Register (block ②) is implemented by combining 4 groups of 8 LUTs (4 bytes wide) in SRL16 configuration mode. The Shift Register has the purpose of temporally storing the AES State, perform CLEFIA Feistel Word Swap or CLEFIA exit word supply.

Block ③ is shown in more detail in Figure 5.3 and depicts a selection operation that performs one of four possible logic operations:
\[ TB_{\text{in}} = SRL16 ; \]
\[ TB_{\text{in}} = SRL16 \oplus KeyB ; \]
\[ TB_{\text{in}} = FWD \oplus KeyB ; \]
\[ TB_{\text{in}} = SRL16 \oplus FWD \oplus KeyB. \]

Rather than using two selection bits to select between four operations, several input values can be set to 0 instead, thus requiring a single selection bit and allowing for the use of 32 LUT4s.

Block 3 is mainly responsible for determining whether the data supplied to the TBoxes is forwarded or not. During CLEFIA, this block adds to the 32-bit forwarded word (FWD) the Round Key (KeyB input) and the Feistel swapped word (from SRL16). During AES, most data comes directly from the Shift Register (SRL16), with the exception of the last State byte that needs to be forwarded (FWD \oplus KeyB) at the beginning of each round.

The remaining additions of the TBox substitution are performed in 5 with a XOR tree composed of 32 LUT4s, while the output circuit (block 6 and Figure 5.2) performs the last AES round and CLEFIA last Whitening Key addition with 32 LUT6s.

Figure 5.3: Detailed View of the Forwarding and Memory Stage
**BRAMs mapping**

Having the more recent Virtex families from Xilinx as the target technology, 36Kb BRAMs are considered for the TBoxes implementation. Considering a structure with a 32-bit data path, two dual-port BRAMs are required. In each input port of each BRAM, the first 8 bits of address are used by the respective byte input. The 9th address bit of the BRAM is used to differentiate between encryption/decryption mode when computing AES or between a F0/F1 TBox when computing the CLEFIA algorithm. A final 10th bit is used to select which algorithm is being considered, addressing different memory locations with different TBox mappings. Given the 32 bit output word of each BRAM a total of 32kb per BRAM are required.

A particular care needs to be taken regarding the mapping of the BRAMs outputs. As described in equations (2.3), (2.5), (2.8) and (2.10) each algorithm shifts the column coefficients in a particular way. Nevertheless, this shifting can coincide if the columns are properly paired, by noticing that the 3° column of every presented matrix suffers the same shifting from the 1° column, as the 4° column from the 2° one.

By using this characteristic when defining the connection and content of the BRAMs, the shifting path can still be hard-wired without the use of selection logic, as depicted in Figure 5.3.

### 5.4 Pipeline Scheduling

In the data path defined, in the previous section, four pipeline stages exist: the input/feedback stage that defines the Shift Register input; the forwarding stage that defines the BRAMs input; the actual BRAMs stage and; the last round output circuitry.

Given the algorithms on the proposed structure, the Pipeline Schedule for CLEFIA and AES is defined according to Tables 5.2 and 5.3, considering 128-bit keys (192 and 256-bit keys are perfectly compatible, requiring only more cipher rounds). In Table 5.3, depicting the AES scheduling, the ‘TBOX Process’ stage (inside the BRAMs) is not mentioned, simply because it does not change with the operating mode (Enc/Dec) during the process, as oppose to the F-function in CLEFIA. The ‘TBOX Output’ represents the result obtained after the common XOR tree (Figure 5.1) and belongs to the same pipeline stage as ‘SRL16 Input’ and ‘TBOX Input’. The Shift Register address scheduling is presented in Appendix B and C.

Each cipher round starts when the first 32-bit word of a respective State is addressed, or forwarded, into the BRAMs input. When placed into the BRAMs input, each 32-bit word takes 3 cycles to be processed and is stored in the Shift Register.

In CLEFIA only two words require TBox processing, meaning that each round takes 2 clock cycles to be computed. In AES, in order for the entire State (four words) to be processed by the TBoxes, and the new created State to be stored back into the Shift Register, 6 clock cycles are required (Table 5.3). With data forwarding, each round can start one cycle before the storage is complete (reduced to 5 cycles) but an inherent dead cycle is still required between rounds.
When starting a new ciphering process, 6 initial clock cycles are required to extract and store one input data block, after the Whitening Key addition (XOR), into the Shift Register (the 2 initial cycles are used for initiating the addresses of the keys memory storage). AES can only start after the entire data block has been stored and capable of being addressed by the Shift Register, but as long as the 1° and 3° CLEFIA words (P0 and P2) can be addressed sequentially, its 1° Round can start two cycles earlier.

In CLEFIA, the Shift Register only addresses the two initial words for the TBox BRAMs directly during the first Round. From the 2° to the 18° Round (128-bit Key) all BRAM inputs are fed through the data forwarding while adding the Shift Register Feistel swapped word.

In AES, the Shift Register addresses all 16 bytes (0 to 15) of a State into the BRAMs in its first round. In the remaining rounds, only the 16° byte (byte 15) is forwarded in the first cycle of an encryption round (in decryption, the forwarded byte is 13, or the 14° of a State). In this forwarding cycles, the Shift Register addresses an 8-bit zero value to be XORed with the forwarded byte and Round Key byte (recall the forwarding operations presented in Section 5.3). This zero value is always stored into the Shift Register by resetting the BRAMs output registers every dead cycle.

As already mentioned, the same TBox is used for the entire AES cipher, once pre-configured for encryption or decryption. In CLEFIA, however, each round executes two different TBoxes, so the BRAMs need to be addressed alternately for the different F-functions. The alternation is depicted in Table 5.2 for encryption. For decryption, due to the different Feistel Word Swap, the sequence becomes: F0, F1, F1, F0, ...

At the end of the 18° CLEFIA Round (or 22° or 26° for 192 or 256-bit cipher keys) the pipeline needs to be flushed, and the last processed words stored, before continuing. After this, the Shift Register bypasses the last Feistel Word Swap (Unswap Round in Table 5.2), by orderly addressing the final words to the Output Stage to add the Whitening Keys.

The AES conclusion (last round and text output) can be performed within its final round (10°, 12° or 14° according to the key size) by taking 8 clock cycles. In this round (10° of Table 5.3) the first 8 bytes of the State are sent to be processed by the BRAMs, but then the process is stalled 2 cycles to allow the direct output (and Whitening) of the first 64 bits of the final processed result (recall the output stage depicted in Figure 5.2). After that, the remaining 8 bytes of the State are then processed and outputted in the same way.

During the last cycles of AES final round or CLEFIA Unswap Round, the ‘SRL16 Input’ stage is unused. Because of this, a new input data block can be stored into the Shift Register before the processing of the last data block has concluded. This allows to reduce the overhead of clock cycles needed to input a new data block when processing sequential streams of data.
Table 5.2: CLEFIA Encryption Pipeline Scheduling for a 128-bit key, where TBOX Output depicts the results of the F-functions

<table>
<thead>
<tr>
<th>Stage</th>
<th>TBOX Input</th>
<th>TBOX Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>P0 + W K0 = S0</td>
<td>X</td>
</tr>
<tr>
<td>1° Round</td>
<td>P0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>P1 + W K1 = S1</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>P2 + W K2 = S2</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>P3 + W K3 = S3</td>
<td>X</td>
</tr>
<tr>
<td>2° Round</td>
<td>a1 + P1 + W K0 = A1</td>
<td>A1 + RK2</td>
</tr>
<tr>
<td></td>
<td>c1 + P2 + W K1 = C1</td>
<td>C1 + RK3</td>
</tr>
<tr>
<td>3° Round</td>
<td>a2 + C1 = A2</td>
<td>A2 + RK4</td>
</tr>
<tr>
<td></td>
<td>c2 + A1 = C2</td>
<td>C2 + RK5</td>
</tr>
<tr>
<td>4° Round</td>
<td>a3 + C2 = A3</td>
<td>A3 + RK6</td>
</tr>
<tr>
<td></td>
<td>c3 + A2 = C3</td>
<td>C3 + RK7</td>
</tr>
<tr>
<td>5° Round</td>
<td>a4 + C3 = A4</td>
<td>A4 + RK8</td>
</tr>
<tr>
<td></td>
<td>c4 + A3 = C4</td>
<td>C4 + RK9</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18° Round</td>
<td>a17 + C16 = A17</td>
<td>A17 + RK34</td>
</tr>
<tr>
<td></td>
<td>c17 + A16 = C17</td>
<td>C17 + RK35</td>
</tr>
<tr>
<td>Flush Round</td>
<td>a18 + C17 = A18</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>c18 + A17 = C18</td>
<td>X</td>
</tr>
<tr>
<td>Unswap Round</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>New 1° Round</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.3: AES Pipeline Scheduling for a 128-bit key. \(R_i^m\) are the \((i + 1)^{th}\) 4 Bytes of the \(m^{th}\) round MixColumn's result. \(FR_i^{10}\) are the \((i + 1)^{th}\) 4 Bytes of the SubBytes result in the last round. \(RK_i\) are the \(i + 1^{th}\) 4 Bytes the of the \(I + 1^{th}\) round Key

<table>
<thead>
<tr>
<th>Stage</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1° Round</td>
<td>(R_0 + RK_0 = S0)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_1 + RK_1 = S1)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_2 + RK_2 = S2)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_3 + RK_3 = S3)</td>
<td>X</td>
</tr>
<tr>
<td>2° Round</td>
<td>(R_0^0 + RK_0^0 = S0^0)</td>
<td>(R_0^0)</td>
</tr>
<tr>
<td></td>
<td>(R_1^0 + RK_1^0 = S1^0)</td>
<td>(R_1^0)</td>
</tr>
<tr>
<td></td>
<td>(R_2^0 + RK_2^0 = S2^0)</td>
<td>(R_2^0)</td>
</tr>
<tr>
<td></td>
<td>(R_3^0 + RK_3^0 = S3^0)</td>
<td>(R_3^0)</td>
</tr>
<tr>
<td>...</td>
<td>(R_0^{10} + RK_0^{10} = S0^{10})</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_1^{10} + RK_1^{10} = S1^{10})</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_2^{10} + RK_2^{10} = S2^{10})</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_3^{10} + RK_3^{10} = S3^{10})</td>
<td>X</td>
</tr>
<tr>
<td>New 1° Round</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_0^{11} + RK_0^{11} = S0^{11})</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_1^{11} + RK_1^{11} = S1^{11})</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_2^{11} + RK_2^{11} = S2^{11})</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>(R_3^{11} + RK_3^{11} = S3^{11})</td>
<td>X</td>
</tr>
</tbody>
</table>

37
5.5 Control Unit and Final Circuit

With the data path concluded, all that remains to complete the AES/CLEFIA dual-cipher architecture is a Control Unit (CU) and the local memory for the Round Keys storage, where the overall circuit can be seen in Figure 5.4. This section will mainly address the external interface signals. For a full description of all inter-connections, Appendix D can be consulted.

The Control Unit was designed as a state machine with 5 common states and 1 CLEFIA exclusive state. It progresses through the flowchart presented in Figure 5.5 using two counters: one for the number of rounds performed, a second for the number of clock cycles passed within a state or round.

Before processing a plain/cipher text, the CU needs to be reset ('Reset_UC') to its idle state, defining the encryption algorithm, mode, and key size. Once the Control Unit its signalled to start execution ('BLK_Ready') the keys must have been loaded to the local memory.

The first functional state of the CU is the preparative/extraction state, where two initial cycles are used to initiate the keys address. After that, four cycles are used to extract, from a 32-bit input port, four plain/cipher text words for encryption/decryption.

Figure 5.4: The Dual-Cipher Co-Processor
The Ini_Round state performs the 5 clock cycles of the first AES round, or the first 3 rounds (6 cycles) of CLEFIA. The reason why the initial round is separate from the remaining ones is due to the fact that the Shift Register addressing is still not periodic.

Once the first rounds are completed, the CU starts processing the remaining mid-rounds sequentially (Loop_Round/in_loop state), with 5 cycles per AES round, or 4 cycles per two CLEFIA rounds. At the end of each round, the respective counter is incremented and evaluated. When the final round is reached the CU interrupts the recursive processing and changes to a final round state depending on the cipher.

If CLEFIA is being used, the final round is processed by CLEFIA_PreFinal state. Four clock cycles are used to process this round and clear the pipeline stages.

![Flowchart of State Machine for Control Unit](image)

Figure 5.5: Flowchart of State Machine for Control Unit

The “Final” state is reached directly for processing AES last round, or just after CLEFIA_PreFinal has been concluded. For CLEFIA, the Final state is only used to output the final words and add to them the last two Whitening Keys, all within 4 clock cycles. For AES, the Final state is more complex and takes 8 clock cycles to process the last AES round while outputting directly its results. Every time a new
completely processed word is stored within the output register, the signal ‘Word.Done’ is active. For each block, this signal is activated a total of four times.

**Keys storage and memory loading**

Since only two key inputs are required for the data path, the best component to be used as local storage for the off-chip, pre-computed, Round Keys is a third dual-port 36Kb BRAM with 256x32b configuration, as depicted in Figure 5.6. However, the largest amount of 32-bit Keys required is 56, either for an AES or CLEFIA. This means that even if four equally sized memory sections are reserved within the BRAM for different cipher types and modes, the total amount of memory required is:

\[ 4 \times 56 \times 32 \text{bits} = 7 \text{Kb} \]

\[(5.1)\]

For this small amount of memory, only 6 bits are required to address the needed keys. In order to load all required keys, a single entry port of the BRAM is used to input each 32-bit key. The loading process of the keys is controlled through external interface.

![Figure 5.6: Keys Storage input logic, with External signals, Control Unit signals and signals to the Data Path](image)

40
5.6 Dismissed Alternatives

During the design process for the dual-cipher co-processor, many alternative implementations were considered, but later discarded to give place to the more efficient choices. However, two of this solutions took unorthodox methods to solve their respective problems: a circuit that implements the AES Shift_Row operation after the Mix_Columns, and another that extracts direct unitary values from AES Decryption TBox. A small insight into this solutions are herein provided.

Shift_Rows with mid registers and out-of-order inputs

One early approach to implement the dual-cipher architecture had a distinct solution for the AES Shift_Rows operation. With this approach there is no need for the Shift Register and the XOR tree, after the BRAM TBoxes. In each round, the four columns of a non-shifted State are directly transform by the TBoxes, but the outputs needs to be processed by a set of temporary 32-bit registers with a XOR operator at their inputs. This technique makes use of the column-wise TBox depicted in equation (2.6), and as the State

![Diagram of Shift_Rows through temporary registers](image.png)

Figure 5.7: Shift_Rows through temporary registers
columns are provided to the TBoxes, the temporary registers shift positions while accumulating the old values with the recently processed ones. Between feeding the first State column to the BRAMs and fully concluding a round, 7 clock cycles are required (no forwarding is possible).

During encryption, the shifting of the registers can be performed in a right-round manner, while left-rounded in decryption.

This solution has the interesting feature of the Shift_Rows being progressively performed after the Mix_Columns. With this approach several issues raise. One of them is that it becomes hardly adaptable for the CLEFIA cipher, while another problem is that it creates a 128-bit wide pipeline stage, better suited for larger data paths and not for 32-bit compact ones.

Last Round through Modified Mix_Columns

One recurrent problem of implementing AES with TBoxes is how to perform its last round, specially during decryption. Chapter 3 presents several State of the Art solutions, including the four parallel XOR trees of Chaves et al. [2006], the one chosen for this work final design. Still, one other viable solution was previously considered.

As shown in Section 2.1, it is possible to bypass the encrypting Mix_Columns operation, inherent to a TBox, by directly extracting its unitary coefficient result (equation 2.2), but such coefficient does not exist in decryption (equation 2.4). For the Dual-Cipher project, mapping an extra InvSBox is not viable since the BRAMs capacity has been fully used, and using Rouvroy et al. [2004] multiplexer solution would create an unbalanced 12-input function (2 LUTs) in parallel with the established LUT4s of the XOR tree.

In order to extract a direct value from the SBox substitution, in both modes, a possible solution is to decompose the Galois Field ($2^8$) multiplication and extract a different set of coefficients from the TBoxes, in which one is always unitary. For the remaining data path, the matrix multiplication process is concluded by an extra layer of XOR operators before the already existing XOR tree.

The new coefficients are described in the following equations, while the equivalent circuit for AES decryption is presented in Figure 5.8.

\[
\begin{bmatrix}
02 \times SB(x) \\
01 \times SB(x) \\
01 \times SB(x) \\
03 \times SB(x)
\end{bmatrix} = \begin{bmatrix}
00.SB(x) \oplus 02.SB(x) \oplus 00.SB(x) \\
00.SB(x) \oplus 01.SB(x) \\
00.SB(x) \oplus 01.SB(x) \\
02.SB(x) \oplus 01.SB(x)
\end{bmatrix}
\]

\[
\begin{bmatrix}
0E \times ISB(x) \\
09 \times ISB(x) \\
0D \times ISB(x) \\
0B \times ISB(x)
\end{bmatrix} = \begin{bmatrix}
0C.ISB(x) \oplus 0A.ISB(x) \oplus 08.ISB(x) \\
08.ISB(x) \oplus 01.ISB(x) \\
0C.ISB(x) \oplus 01.ISB(x) \\
0A.ISB(x) \oplus 01.ISB(x)
\end{bmatrix}
\]

As can be seen, one bytes of each BRAM outputs always represent the unitary coefficient, and can be directly extracted for the last round of both encryption and decryption. All resulting circuits using this
approach are depicted in Appendix E.

\[
\begin{bmatrix}
01.S_0(x) \\
02.S_0(x) \\
04.S_0(x) \\
06.S_0(x)
\end{bmatrix} =
\begin{bmatrix}
05.S_0(x) + 07.S_0(x) + 03.S_0(x) \\
03.S_0(x) + 01.S_0(x) \\
05.S_0(x) + 01.S_0(x) \\
07.S_0(x) + 07.S_0(x)
\end{bmatrix}
\tag{5.4}
\]

\[
\begin{bmatrix}
01.S_1(x) \\
08.S_1(x) \\
02.S_1(x) \\
0A.S_1(x)
\end{bmatrix} =
\begin{bmatrix}
03.S_1(x) + 0B.S_1(x) + 09.S_1(x) \\
09.S_1(x) + 01.S_1(x) \\
03.S_1(x) + 01.S_1(x) \\
0B.S_1(x) + 01.S_1(x)
\end{bmatrix}
\tag{5.5}
\]

It is possible to remove the output circuitry altogether by feed-backing the last round result to the Shift Register (a MUX at the end of the XOR tree), and extract the Ciphered Text directly from the latter. This rearranges the stalling cycles of AES last round that exist in the pipeline schedule (Table 5.3), but also removes the ability of immediately streaming a second block.

Requiring an extra logic level in the critical path, and the need of 32 extra LUTs to maintain a partial streaming ability, made Chaves et al. [2006] solution prevail over the modified Mix_Columns implementation.

![Figure 5.8: Modified Mix_Columns design during AES Decryption](image)

Figure 5.8: Modified Mix_Columns design during AES Decryption
5.7 Conclusions on the Dual-Cipher implementation

In this chapter, the process of implementing the dual-cipher processor is detailed. The general design of the data path is adapted in order to properly fit Xilinx Virtex 5 FPGA technology, while minimizing the resource usage. Herein the 5 pipeline stages are properly identified and scheduled, with an average of 2 clock cycles per round in CLEFIA and 5 clock cycles per round in AES, where the dead cycle of the latter is its greatest shortcoming.

To control the dual-cipher processor, a 6-state machine was created as a Control Unit, along with the addition of an extra dual-port BRAM (and respective interface logic) to store and provide the off-chip pre-computed Round Keys.

Two implementation alternatives are also described in this chapter for future reference, being dismissed on the final dual-cipher structure given their limitations.

In the next chapter the performance values of the dual-cipher processor are analysed and compared with the most competing State of the Art.
Chapter 6

Result Analysis

In this chapter, experimental results for the proposed structure and the related state of the art are presented and compared on several Xilinx FPGAs technologies. The gather results were obtained using the Xilinx ISE Design Suite (v14.5) with the design described using VHDL. The values presented for the proposed design were obtained after Place&Route with software default parameters, namely Synthesis Normal Speed Optimization Effort, and High Optimization Effort in Mapping (Area targeted) and Place&Route with no extra effort. The proposed structure was also tested and its outputs validated using Xilinx ISE Design Suite embedded simulator ISim (v14.5).

6.1 Dual-cipher Performance Analysis

Table 6.1 depicts the resource requirements for the Control Unit and Data Path when separately implemented, as well for the entire dual-cipher co-processor. The xc5v and xc6v labels represent the Xilinx Virtex 5 and 6 FPGA board families, while the remaining nomenclature states the board series and speed grade. This work was design with the Virtex 5 family in mind. Nevertheless implementation results were also obtained for the Virtex 6 family.

<table>
<thead>
<tr>
<th>Device</th>
<th>SLICEs</th>
<th>LUTs</th>
<th>FFs</th>
<th>RAMB36s</th>
<th>Freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Unit</td>
<td>xc5vlx30-3</td>
<td>52</td>
<td>109</td>
<td>33</td>
<td>447,4</td>
</tr>
<tr>
<td></td>
<td>xc6vlx240t-3</td>
<td>40</td>
<td>83</td>
<td>30</td>
<td>556,8</td>
</tr>
<tr>
<td>Data Path</td>
<td>xc5vlx30-3</td>
<td>67</td>
<td>160</td>
<td>32</td>
<td>321,6</td>
</tr>
<tr>
<td></td>
<td>xc6vlx240t-3</td>
<td>93</td>
<td>253</td>
<td>32</td>
<td>359,1</td>
</tr>
<tr>
<td>Co-processor</td>
<td>xc5vlx30-3</td>
<td>119</td>
<td>296</td>
<td>61</td>
<td>337,3</td>
</tr>
<tr>
<td></td>
<td>xc6vlx240t-3</td>
<td>115</td>
<td>329</td>
<td>61</td>
<td>332,1</td>
</tr>
</tbody>
</table>

Table 6.1: Resource usage of the Dual-Cipher Co-processor

As expected, the Control Unit can be designed to require less logic components than the Data Path. Nevertheless the need to control two different ciphers still imposes a significant cost, having a cost not much smaller than the datapath, having an important influence on the circuit size.

The entire co-processor includes the Control Unit, Data Path, and some small extra logic needed for
the Key Storage BRAM interface. Still, combining the entire circuit does not necessarily mean the sum of the parts, since the synthesis tool takes some configuration liberties, namely in Slice/LUT distribution.

Regarding the achieved throughput, the values presented in Table 6.2 define the average throughput at which the circuit processes input blocks. As illustrated, the proposed structure allows for a ciphering throughput of 1Gbps for the CLEFIA algorithm and near 800Mbps for the AES algorithm. These results are achieved at a cost of 119 SLICEs and 3 BRAMs on a Virtex 5 device.

The AES encryption mode results in a lower throughput, in regard to the CLEFIA computation, given the empty 5th cycle per round, as discussed in the Chapter 5. The throughput values are presented regarding the use of a single data stream in a feedback mode, considered as the most used and secure encryption mode. This implies that the computation of data block \( i \) can only be started after the full computation of data block \( i - 1 \) is completed.

If independent data blocks are considered, the proposed structure can be used to cipher two data blocks simultaneously, by changing the control unit, and fully using the 16 positions available in the SRL16 Shift Register, instead of the current 8. This allows to eliminate the 5th empty cycle in the AES computation, which results in a throughput of about 1Gbps.

One can question the performance difference between Virtex 5 and Virtex 6. The extrapolated circuit seems to cause a variation in the synthesis tool during the standard optimization process. Although requiring additional logic for the Data Path, the combined co-processor actually manages to require a similar amount of SLICEs. This is most likely due to added functionalities between board families, but without a deeper comprehension of the technologies it is hard to determine the actual causes.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Device</th>
<th>Key Size (bits)</th>
<th>Init Cycles</th>
<th>Cycles/Block</th>
<th>Throughput (Gbps)</th>
<th>Efficiency (Mbps/SLICE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>xc5vlx30-3</td>
<td>128</td>
<td>6</td>
<td>54</td>
<td>0.799</td>
<td>6.72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>6</td>
<td>64</td>
<td>0.675</td>
<td>5.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td></td>
<td>74</td>
<td>0.583</td>
<td>4.90</td>
</tr>
<tr>
<td></td>
<td>xc6vlx240t-3</td>
<td>128</td>
<td>6</td>
<td>54</td>
<td>0.787</td>
<td>6.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>6</td>
<td>64</td>
<td>0.664</td>
<td>5.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td></td>
<td>74</td>
<td>0.574</td>
<td>5.00</td>
</tr>
<tr>
<td>CLEFIA</td>
<td>xc5vlx30-3</td>
<td>128</td>
<td>4</td>
<td>43</td>
<td>1.004</td>
<td>8.44</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>4</td>
<td>51</td>
<td>0.846</td>
<td>7.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td></td>
<td>59</td>
<td>0.732</td>
<td>6.15</td>
</tr>
<tr>
<td></td>
<td>xc6vlx240t-3</td>
<td>128</td>
<td>4</td>
<td>43</td>
<td>0.989</td>
<td>8.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>4</td>
<td>51</td>
<td>0.834</td>
<td>7.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td></td>
<td>59</td>
<td>0.721</td>
<td>6.27</td>
</tr>
</tbody>
</table>

Table 6.2: Performance Study of the Dual-Cipher Co-processor

### 6.2 State of the Art Comparison

The obtained results for the proposed structure and those present in the state of the art are depicted in Table 6.3. A few particularities have to be considered when analysing these results.

In unrolled structures, higher throughputs can be achieved, but only if multiple blocks are processed simultaneously. When considering a single data stream in a feedback mode, such as CBC, these struc-
tures cannot be efficiently used due to the data dependency between blocks. In Table 6.3 the throughput values are depicted as presented by their authors, while between brackets are the throughput values considering a single data stream in feedback mode.

Efficiency wise, we consider the use of the Throughput per Slice metric. This metric can be contested as a biased measurement, since it does not take into account other FPGA modules such as BRAMs or DSPs. However, given the difficulty in extracting equivalent values, this is herein used as the efficiency comparison metric. Between brackets are the efficiency metrics considering the throughput in feedback mode.

Regarding the key expansion, two main approaches are used. Either computing them locally with dedicated logic, or computing them off chip and then storing them in local memory. Although being possible to share some resources with the datapath to perform the key expansion computation, dedicated logic is still required Sklavos and Koufopavlou [2002]. The off chip computation of the key expansion and loading to an auxiliary BRAM typically yields in more compact and efficient designs.

In regard to the related AES state of the art, the unrolled dual-pipelined round architecture proposed in Liu et al. [2013] allows for a throughput above 60Gbps in ECB mode. However, when considering feedback modes the maximum throughput is of 3.2Gbps with a area cost of 3121 Slices resulting in a Throughput per Slice efficiency of 1.03. The 128-bit folded datapath structure, single cycle per round, presented in Chaves et al. [2006] achieves a throughput of 2.4Gbps on a Virtex 5 device with an efficiency of 5.96 at a cost of a higher BRAM usage.

The more compact DSP-structure proposed in Drimer et al. [2010] allows for a throughput up to 1.76Gbps and achieved efficiency of 8.22, requiring 107 SLICEs and 4 DSPs. However, if feedback modes are used, the maximum throughput lowers to 880Mbps. Without the use of DSPs, 212 Slices are needed, resulting in an efficiency drop to 4.15 when considering feedback modes.

Considering the CLEFIA state of the art, the dedicated unrolled structure Kryjak and Gorgon [2009] allows for a throughput of 21Gbps in non-feedback modes. In feedback modes a maximum throughput of 1.2Gbps can be achieved. With a area cost of 2479 Slices, an efficiency of 0.48 is achieved for feedback modes. The structure proposed in Proença and Chaves [2011] allows for a throughput of 1.7Gbps with a cost of 170 Slices. The more compact structure, also proposed in Proença and Chaves [2011], allows for a throughput of 1.3Gbps at a cost of 86 Slices, resulting in efficiency of 15.13.

Overall, the proposed structure allows for a throughput between 800Mbps and 1Gbps for AES and CLEFIA algorithms respectively, at a cost of 119 Slices and 3 BRAMs, with a feedback mode efficiency identical to those in the related, single algorithm, state of the art. Given AES State of the Art, it can be concluded that the proposed structure here presented allows for the highest AES efficiency of 6.71, while still allowing for the CLEFIA computation. With a maximum CLEFIA throughput of 1Gbps, the proposed structure has an efficiency of 8.43, 45% lower that the best CLEFIA state of the art. These lower results are due to the additional logic used for the data forwarding and the different final key addition structure.
## Table 6.3: State of the Art Performance Comparison

<table>
<thead>
<tr>
<th>Device</th>
<th>Structure</th>
<th>Resources</th>
<th>freq (MHz)</th>
<th>Throughput (Gbit.s⁻¹)</th>
<th>Efficiency (Mbit.s⁻¹/S)</th>
<th>AES-Enc</th>
<th>AES-Enc+Dec</th>
<th>AES-Dec</th>
<th>AES-Dec+Enc</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELION Tiny</td>
<td>AES</td>
<td>SBox</td>
<td>65</td>
<td>15.13</td>
<td>12.88</td>
<td>0.84</td>
<td>5.42</td>
<td>7.36</td>
<td>0.95</td>
</tr>
<tr>
<td>HELION Standard</td>
<td>AES</td>
<td>SBox</td>
<td>92</td>
<td>15.63</td>
<td>13.26</td>
<td>0.88</td>
<td>5.89</td>
<td>8.17</td>
<td>1.01</td>
</tr>
<tr>
<td>Crush</td>
<td>AES</td>
<td>SBox</td>
<td>142</td>
<td>16.13</td>
<td>13.91</td>
<td>0.92</td>
<td>6.38</td>
<td>8.56</td>
<td>1.02</td>
</tr>
<tr>
<td>Drimer et al.</td>
<td>AES</td>
<td>TBox</td>
<td>347,5</td>
<td>16.45</td>
<td>15.12</td>
<td>1.02</td>
<td>7.40</td>
<td>9.47</td>
<td>1.09</td>
</tr>
<tr>
<td>El Maraghy et al.</td>
<td>AES</td>
<td>SBox</td>
<td>86, 128</td>
<td>16.29</td>
<td>14.81</td>
<td>1.05</td>
<td>8.25</td>
<td>10.29</td>
<td>1.16</td>
</tr>
<tr>
<td>Liu et al.</td>
<td>AES</td>
<td>TBox</td>
<td>312, 347</td>
<td>16.12</td>
<td>14.81</td>
<td>1.06</td>
<td>8.25</td>
<td>10.29</td>
<td>1.16</td>
</tr>
<tr>
<td>Proenc ¸a and Chaves [2011]</td>
<td>AES</td>
<td>TBox</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>Chaves et al.</td>
<td>AES</td>
<td>TBox</td>
<td>230, 257</td>
<td>2.42</td>
<td>1.32</td>
<td>1.02</td>
<td>1.70</td>
<td>2.49</td>
<td>1.32</td>
</tr>
<tr>
<td>Drimer et al.</td>
<td>AES</td>
<td>TBox</td>
<td>347, 366</td>
<td>2.40</td>
<td>1.20</td>
<td>1.02</td>
<td>1.70</td>
<td>2.49</td>
<td>1.32</td>
</tr>
<tr>
<td>Liu et al.</td>
<td>AES</td>
<td>TBox</td>
<td>312, 230</td>
<td>2.49</td>
<td>1.40</td>
<td>1.02</td>
<td>1.70</td>
<td>2.49</td>
<td>1.32</td>
</tr>
<tr>
<td>Proenc ¸a and Chaves [2011]</td>
<td>AES</td>
<td>TBox</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>El Maraghy et al.</td>
<td>AES</td>
<td>SBox</td>
<td>86, 128</td>
<td>2.40</td>
<td>1.20</td>
<td>1.02</td>
<td>1.70</td>
<td>2.49</td>
<td>1.32</td>
</tr>
<tr>
<td>Liu et al.</td>
<td>AES</td>
<td>TBox</td>
<td>312, 230</td>
<td>2.49</td>
<td>1.40</td>
<td>1.02</td>
<td>1.70</td>
<td>2.49</td>
<td>1.32</td>
</tr>
<tr>
<td>Proenc ¸a and Chaves [2011]</td>
<td>AES</td>
<td>TBox</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
</tbody>
</table>
6.3 Result Analysis Conclusion

In this chapter, the performance and efficiency results of the proposed dual-cipher co-processor were analysed and compared to the State of the Art. Two devices were tested upon, namely a Xilinx Virtex 5 and Virtex 6. The proposed structure requires 119 SLICEs and 3 BRAMs, and can operate with a working frequency of up to 337 MHz.

State of the Art implementations were compared with this thesis work. Unrolled round architectures are the best suited for non-feedback modes (throughputs of up to 64Gbps). In feedback modes however, this large structures become very inefficient in resource usage (Mbps/SLICE), being the compact rolled round architectures preferred for this case scenarios.

While performing the AES algorithm, the proposed compact rolled round dual-cipher architecture can reach a throughput of 800 Mbps and a feedback mode efficiency of 6.71 Mbps/SLICE, the highest value in the AES State of the Art.

During the CLEFIA algorithm the proposed co-processor has an efficiency value of 8.44 Mbps/SLICE (also in feedback mode), 45% lower than the best CLEFIA State of the Art, but its throughput of 1Gbps keeps the proposed solution competitive with the other implementations.
Chapter 7

Conclusions

The thesis work here presented, proposes a novel compact structure for a AES/CLEFIA dual-cipher co-processor, while providing proof that efficient multi-cryptographic engines are viable.

By studying the AES and the CLEFIA encryption algorithms, similar ciphering techniques were found, such as diffusion matrices, byte substitutions, and iterative rounds. The major differences reside on the fact that CLEFIA is based on a Feistel Structure, while the AES algorithm is based on a Shift.Rows operation, so minimizing their impact in a merged architecture is an inadvertent challenge.

State of the Art research has provided an insight on the most common implementation approaches. Unrolled round structures are more suitable for faster non-feedback ciphering, while rolled round structures are more area-efficient in feedback modes. Several methods can be found for performing the non-linear byte transformations of block ciphers, as hard-wired logic or addressable lookup tables: SBox and TBox. In FPGA technologies, there is a common practice of using the dedicated memory components (BRAM) for tables storage, as well as specific implemented shift registers for AES Shift.Rows operation.

The dual-cipher co-processor herein proposed was carefully designed for the Xilinx Virtex 5 technology and respective components, considering a 32-bit compact Data Path in rolled round format, with BRAM-based TBoxes and temporary storage within an addressable Shift Register. To improve the efficiency of the proposed structure, the encryption Round Key Scheduling are pre-computed by external means and stored in a dedicated BRAM.

The resulting structure herein proposed requires a total of 119 FPGA SLICEs, on a Virtex 5. With a maximum clock frequency of 337 MHz, the AES cipher can be used in feedback mode allowing a throughput of 800 Mbps and the best efficiency value in State of the Art of 6.71 Throughput/SLICE. For the CLEFIA ciphering, the dual-cipher structure allows for an efficiency of 8.43, 45% less than the best State of the Art alternative, but still capable of achieving 1 Gbps of overall throughput.
7.1 Future Work

Among the State of the Art, the proposed dual-cipher Co-processor is the first compact dedicated circuit to implement AES and CLEFIA through resource sharing and without the need for structural reconfiguration. This opens several research interests for future work.

One of the first extra implementations that can be considered is the addition of Key Schedulers for these ciphers. As discussed, off-chip Key Scheduling tends to be more efficient in resource usage, but some applications, such as field embedded systems, require dedicated on-chip schedulers. For the dual-cipher co-processor, it should be easy to merge a AES Key scheduler, since an extra BRAM is already being used for storage, a common component in Rouvroy et al. [2004] and Bulens et al. [2008] schedulers. Expanding a cipher key for CLEFIA is more complex, since the respective algorithm changes structure depending on the size of the original Key, but if a size of 128 bits is considered the key scheduler can actually use the already implemented Data Path for some of the most demanding computations.

The proposed dual-cipher circuit was design to easily allow ciphering in either feedback or non-feedback modes, but it implements none in particular. This is due to the fact that there are many security modes for different purposes (ECB, CBC, PCBC, CFB, etc...). Further studies in how to implement each one, or several at the same time, should provide a wide range of applicabilities in existing products.

Two ciphers is a relatively small number for a multi-cryptographic engine, so merging other similar ciphers should increase its functional value. One strong candidate for this is the Twofish algorithm, one of the other competitors for AES. Twofish also comprises SBoxes and diffusion matrices, with another adapted Feistel structure. Its two main differences reside on the fact that the SBoxes are directly dependent of the cipher key, and it also requires two decimal 32-bit full adders.

With every upgrade in FPGA technology, new and better configuration options are made available. Such as the SRL16 LUT mode of Virtex 5, more mature techniques may be available for Virtex 6 and 7. Exploring new compact ways to implement the dual-cipher co-processor in different technologies, such as ASIC, may also be considered.
References


Appendix A

FPGA Components Schematics

Figure A.1: Xilinx Virtex 5 SLICEL schematic, as presented in Xilinx’s Virtex-5 FPGA User Guide
Figure A.2: Xilinx Virtex 5 SLICEM schematic, as presented in Xilinx's Virtex-5 FPGA User Guide
Figure A.3: Xilinx Virtex 5 36Kb dual-port BRAM schematic, as presented in Xilinx’s Virtex-5 FPGA User Guide
### Appendix B

#### Detailed CLEFIA Pipeline Scheduling

<table>
<thead>
<tr>
<th>Shift Register Positions</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

| Init | P0 | P0 | X  | X  | X  | X  | X  | X  |
|      | P1+WK0 | P2 | P1+WK0 | P0 | X  | X  | X  | X  |

| 1° Round | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  | X  | X  |
| A1       | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  | X  | X  |

| 2° Round | C1 | A1 | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  | X  |
| A2       | C1 | A1 | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  | X  |

| 3° Round | C2 | A2 | C1 | A1 | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  |
| A3       | C2 | A2 | C1 | A1 | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  |

| 4° Round | C3 | A3 | C2 | A2 | C1 | A1 | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  |
| A4       | C3 | A3 | C2 | A2 | C1 | A1 | P3+WK1 | P2 | P1+WK0 | P0 | X  | X  |

| 5° Round | C16 | A16 | C15 | A15 | C14 | A14 | C13 | A13 |
|Flush Round | C17 | A17 | C16 | A16 | C15 | A15 | C14 | A14 |
|Unswap Round | C18 | A18 | C17 | A17 | C16 | A16 | C15 | A15 |

| 18° Round | P1+WK0' | P0' | X  | X  | X  | X  | X  | X  |
| New 1° Round | P2 | P1+WK0' | P0' | X  | X  | X  | X  | X  |

|                | C18 | A18 | C17 | A17 | C16 | A16 | C15 | A15 |
|                | A18 | C18 | A17 | C17 | A16 | A16 | C15 | A15 |

|                | X  | X  | X  | X  | X  | X  | X  | X  |
|                | X  | X  | X  | X  | X  | X  | X  | X  |

Table B.1: Shift Register occupation address outputs through a 128-bit key CLEFIA encryption
Table B.2: CLEFIA Encryption Pipeline Scheduling for a 128-bit key, where $a'_i$ and $c'_i$ are the results of the F-functions.
## Appendix C

### Detailed AES Pipeline Scheduling

<table>
<thead>
<tr>
<th>Shift Register Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>P0+Wk0</td>
</tr>
<tr>
<td>P1+Wk1</td>
</tr>
<tr>
<td>P2+Wk2</td>
</tr>
</tbody>
</table>

### 1° Round

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>P3+Wk3</td>
<td>P2+Wk2</td>
<td>P1+Wk1</td>
<td>P0+Wk0</td>
<td>X</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>X</td>
<td>P3+Wk3</td>
<td>P2+Wk2</td>
<td>P1+Wk1</td>
<td>P0+Wk0</td>
<td>X</td>
</tr>
</tbody>
</table>

### Dead Cycle

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>A1</td>
<td>0</td>
<td>X</td>
<td>P3+Wk3</td>
<td>P2+Wk2</td>
<td>P1+Wk1</td>
<td>P0+Wk0</td>
</tr>
</tbody>
</table>

### 2° Round

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>C1</td>
<td>B1</td>
<td>A1</td>
<td>0</td>
<td>X</td>
<td>P3+Wk3</td>
<td>P2+Wk2</td>
</tr>
<tr>
<td>A2</td>
<td>0</td>
<td>D1</td>
<td>C1</td>
<td>B1</td>
<td>A1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

### Dead Cycle

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B2</td>
<td>A2</td>
<td>0</td>
<td>D1</td>
<td>C1</td>
<td>B1</td>
<td>A1</td>
<td>0</td>
</tr>
</tbody>
</table>

...                              ...                              ...

### 10° Round

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C9</td>
<td>B9</td>
<td>A9</td>
<td>0</td>
<td>D8</td>
<td>C8</td>
<td>B8</td>
<td>A8</td>
</tr>
<tr>
<td>D9</td>
<td>C9</td>
<td>B9</td>
<td>A9</td>
<td>0</td>
<td>D8</td>
<td>C8</td>
<td>B8</td>
</tr>
<tr>
<td>0</td>
<td>D9</td>
<td>C9</td>
<td>B9</td>
<td>A9</td>
<td>0</td>
<td>D8</td>
<td>C8</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>D9</td>
<td>C9</td>
<td>B9</td>
<td>A9</td>
<td>0</td>
<td>D8</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>0</td>
<td>D9</td>
<td>C9</td>
<td>B9</td>
<td>A9</td>
<td>0</td>
</tr>
<tr>
<td>P0+Wk0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>D9</td>
<td>C9</td>
<td>B9</td>
<td>A9</td>
</tr>
<tr>
<td>P1+Wk1</td>
<td>P0+Wk0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>D9</td>
<td>C9</td>
<td>B9</td>
</tr>
<tr>
<td>P2+Wk2</td>
<td>P1+Wk1</td>
<td>P0+Wk0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>D9</td>
<td>C9</td>
</tr>
</tbody>
</table>

### New 1°Round

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>P3+Wk3</td>
<td>P2+Wk2</td>
<td>P1+Wk1</td>
<td>P0+Wk0</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>

Table C.1: Shift Register occupation through AES (Enc/Dec) and a 128-bit key
Table C.2: AES Pipeline Scheduling for a 128-bit key. $R^{in}_i$ are the $(i+1)^{th}$ 4 Bytes of the $i^{th}$ round MixColumn’s result. $FR^{0i}$ are the $(i+1)^{th}$ 4 Bytes of the Sub_Bytes result in the last round. $RKI_i$ are the $i+1^{th}$ 4 Bytes the of the $I + 1^{th}$ Round Key.

<table>
<thead>
<tr>
<th></th>
<th>SRL16 Input</th>
<th>TBOX Input</th>
<th>TBOX Exit</th>
<th>Exit Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Init</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$P_0 + WK_0 = ST^0_0$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_1 + WK_1 = ST^0_1$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$P_2 + WK_2 = ST^0_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$P_3 + WK_3 = ST^0_3$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>1° Round</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_1^0 + RK_0 = ST^1_0$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_0^1 + RK_0 = ST^0_0$</td>
<td></td>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$R_1^2 + RK_0 = ST^0_2$</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_0^2 + RK_0 = ST^0_0$</td>
<td></td>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$R_1^3 + RK_0 = ST^0_3$</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dead Cycle</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_2^1 + RK_1 = ST^1_2$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_2^2 + RK_1 = ST^1_2$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_2^3 + RK_1 = ST^1_2$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2° Round</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_3^0 + RK_0 = ST^1_0$</td>
<td>$ST^0[0; 5; 10; 15]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_3^1 + RK_1 = ST^1_1$</td>
<td>$ST^0[1; 6; 11]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_3^2 + RK_1 = ST^1_2$</td>
<td>$ST^0[4; 9; 14; 3]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_3^3 + RK_1 = ST^1_3$</td>
<td>$ST^0[15]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dead Cycle</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_4^0 + RK_0 = ST^2_0$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_4^1 + RK_1 = ST^2_1$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_4^2 + RK_1 = ST^2_2$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_4^3 + RK_1 = ST^2_3$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>10° Round</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_5^0 + RK_0 = ST^1_0$</td>
<td>$ST^0[0; 5; 10; 15]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_5^1 + RK_1 = ST^1_1$</td>
<td>$ST^0[1; 6; 11]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_5^2 + RK_1 = ST^1_2$</td>
<td>$ST^0[4; 9; 14; 3]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_5^3 + RK_1 = ST^1_3$</td>
<td>$ST^0[15]$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>New 1° Round</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>$ST^0[0; 5; 10; 15]$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$ST^0[4; 9; 14; 3]$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Appendix D

### Detailed Co-processor Signal Mapping

<table>
<thead>
<tr>
<th>Signal</th>
<th>Size (bits)</th>
<th>From</th>
<th>To</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset UC</td>
<td>1</td>
<td>Off-chip</td>
<td>UC</td>
<td>Hard-reset</td>
</tr>
<tr>
<td>BLK_Ready</td>
<td>1</td>
<td>Off-Chip</td>
<td>UC</td>
<td>Starting signal</td>
</tr>
<tr>
<td>192b_Key</td>
<td>1</td>
<td>Off-Chip</td>
<td>UC</td>
<td>Indicates a 192-bit cipher key (priority over 128-bit predefined value)</td>
</tr>
<tr>
<td>256b_Key</td>
<td>1</td>
<td>Off-Chip</td>
<td>UC</td>
<td>Indicates a 256-bit cipher key (priority over 192-bit input value)</td>
</tr>
<tr>
<td>AES_CLEFIA_Sel</td>
<td>1</td>
<td>Off-Chip</td>
<td>UC/KStore/DPath</td>
<td>Cipher selector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DPath</td>
<td>Bit selector for forwarding the 1^st and 3^rd highest ordered bytes of a 32-bit Word</td>
</tr>
<tr>
<td>Enc_Dec_Sel</td>
<td>1</td>
<td>Off-Chip</td>
<td>UC/KStore</td>
<td>Cipher Mode selector</td>
</tr>
<tr>
<td>Off_Chip_ADDRRA</td>
<td>6</td>
<td>Off-Chip</td>
<td>KStore</td>
<td>Input Port for loading the pre-computed Round Keys</td>
</tr>
<tr>
<td>Expanded_Keys</td>
<td>32</td>
<td>Off-Chip</td>
<td>KStore</td>
<td>Input Port for loading the pre-computed Round Keys</td>
</tr>
<tr>
<td>WEn</td>
<td>1</td>
<td>Off-Chip</td>
<td>KStore</td>
<td>External write enabler for Key loading</td>
</tr>
<tr>
<td>Plaintext</td>
<td>32</td>
<td>Off-Chip</td>
<td>DPath</td>
<td>Input Port for data block 32-bit Words</td>
</tr>
<tr>
<td>Word_Done</td>
<td>1</td>
<td>UC</td>
<td>Off-chip</td>
<td>Signals the availability of 32-bits of completed cipher text (encryption) or plain text (decryption)</td>
</tr>
<tr>
<td>Running?</td>
<td>1</td>
<td>UC</td>
<td>Off-Chip/KStore</td>
<td>Indicative signal for processor execution (idle/not idle)</td>
</tr>
<tr>
<td>ADDRRA</td>
<td>6</td>
<td>UC</td>
<td>KStore</td>
<td>Control address for KeyA supply</td>
</tr>
<tr>
<td>KeyA_reset</td>
<td>1</td>
<td>UC</td>
<td>KStore</td>
<td>Signal for KeyA output register reset</td>
</tr>
<tr>
<td>ADDRBB</td>
<td>6</td>
<td>UC</td>
<td>KStore</td>
<td>Control address for KeyB supply</td>
</tr>
<tr>
<td>KeyB_reset</td>
<td>1</td>
<td>UC</td>
<td>KStore</td>
<td>Signal for KeyB output register reset</td>
</tr>
<tr>
<td>Input_Sel</td>
<td>1</td>
<td>UC</td>
<td>DPath</td>
<td>Bit selector for Word input, or round feedback</td>
</tr>
<tr>
<td>Input_Adder</td>
<td>1</td>
<td>UC</td>
<td>DPath</td>
<td>Bit selector for adding KeyA or CLEFIA Swapped Word</td>
</tr>
<tr>
<td>FWD_Byte1</td>
<td>1</td>
<td>UC</td>
<td>DPath</td>
<td>Bit selector for forwarding the 2^nd highest ordered byte of a 32-bit Word</td>
</tr>
<tr>
<td>FWD_Byte3</td>
<td>1</td>
<td>UC</td>
<td>DPath</td>
<td>Bit selector for forwarding the 4^th highest ordered byte of a 32-bit Word</td>
</tr>
<tr>
<td>Minor_tbox</td>
<td>1</td>
<td>UC</td>
<td>DPath</td>
<td>Bit selector between AES Enc/Dec or CLEFIA F0/F1 TBoxes</td>
</tr>
</tbody>
</table>

Table D.1: Co-processor Signal Mapping
<table>
<thead>
<tr>
<th>Signal</th>
<th>Size (bits)</th>
<th>From</th>
<th>To</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tboxreset</td>
<td>1</td>
<td>UC</td>
<td>DPath</td>
<td>Reset signal for output register of TBox BRAMs</td>
</tr>
<tr>
<td>addr0</td>
<td>3</td>
<td>UC</td>
<td>DPath</td>
<td>Shift Register Address for least significant byte</td>
</tr>
<tr>
<td>addr1</td>
<td>3</td>
<td>UC</td>
<td>DPath</td>
<td>Shift Register Address for 2° least significant byte</td>
</tr>
<tr>
<td>addr2</td>
<td>3</td>
<td>UC</td>
<td>DPath</td>
<td>Shift Register Address for 2° highest significant byte</td>
</tr>
<tr>
<td>addr3</td>
<td>3</td>
<td>UC</td>
<td>DPath</td>
<td>Shift Register Address for highest significant byte</td>
</tr>
<tr>
<td>CipherText</td>
<td>32</td>
<td>DPath</td>
<td>Off-Chip</td>
<td>Output port for 32-bit (de)ciphered Words</td>
</tr>
</tbody>
</table>

Table D.2: Co-processor Signal Mapping
Appendix E

Alternative modified Mix_Columns

Figure E.1: Modified Mix_Columns design during AES Encryption
Figure E.2: Modified Mix_Columns design during AES Decryption

Figure E.3: Modified Mix_Columns design during CLEFIA F0-function
Figure E.4: Modified Mix_Columns design during CLEFIA F1-function