Optimization Algorithms for Ultra-Constrained Applications and Lightweight Cryptography Circuits

Jia Jun Tay

Faculty of Engineering, Computing and Science
Swinburne University of Technology Sarawak Campus
Kuching, Malaysia

Submitted for the degree of Doctor of Philosophy

2019
To my beloved family and friends.
Abstract

Emerging trends in modern computing severely increase the workload demanded from edge devices. Due to expansive deployment of these devices, cost efficiency becomes a major consideration when designing these circuits. Conventional logic optimization algorithms used for logic synthesis in computer-aided design tools are designed to be able to solve for economical circuits efficiently with no restriction to the number of variables and outputs. However, it is known that they do not necessarily provide the global minimum solution. This thesis explores opportunities for heuristics that are not necessarily practical for many-variable minimization problems but are capable of deriving solutions of better quality in metrics attractive to applications in ultra-constrained environments. Simultaneously, security concerns are also a popular topic of discussion in the era of ubiquitous computing. To provide sufficient strength in security for low end devices, introductions of new lightweight cryptographic primitives are frequently observed over the last decade to address a variety of hardware constraints. State-of-the-art hardware optimization techniques on these ciphers are mostly based on varying degree of architecture serialization. While effective in area and power reductions, serial designs come at the cost of severalfold increase in latency which is a major detriment in real-time applications.

Logic optimization based on optimal multiplicative complexity is a relatively new concept in logic design for low gate count implementation. The premise is that given a target function to compute, a circuit implementation requiring the minimal number of AND gates gives close to optimal solution in terms of circuit size. In this thesis, the Boyar-Peralta algorithm is studied extensively as the original logic optimization algorithm based on this heuristic. Due to involvement of a randomized selection procedure in the algorithm, the solutions produced can be unpredictable and the consistency and reliability of the algorithm can be called into question. From this study, enhancements to the Boyar-Peralta algorithm are proposed to mitigate this problem and also improve the algorithm in other aspects. Specifically, the key contributions of the enhanced algorithm include improved average quality of results and reduced variation.

Ideally, an alternative approach to the heuristic without any association to randomness
is desirable to completely eliminate the concerns on consistency. It is noticed that a significant number of practical logic optimization problems involve functions with lower bounded multiplicative complexity. This observation inspired the derivation of a deterministic approach to achieve optimal AND-count based on the principles of Reed-Muller decomposition that is applicable only to the optimization of lower bounded functions. From that knowledge, a novel tree search algorithm is proposed as the deterministic counterpart to the enhanced Boyar-Peralta algorithm. In addition to eliminating the consistency issues plaguing the original algorithm, experimental results showed significant improvement in computation time along with comparable (if not better) quality of results. Application of the tree search algorithm on the optimization of the AES S-Box also resulted in the smallest hardware implementation of the function compared to existing works.

This thesis studies the design characteristics of seven popular lightweight block ciphers to investigate new methodologies for area and/or power reduction aside from serialization in order to preserve good latency. These optimization methodologies aim to reduce hardware resources required for common cryptographic transformations such as non-linear substitution, finite field multiplication, key scheduling and round constant generation. Each proposed methodology is evaluated on applicable ciphers to observe improvements or the lack thereof. In case of the latter, sufficient explanations are provided to clarify the causes of incompatibility with the particular cipher. A final recommendation is proposed for each cipher covering methodologies that showed positive impact in area and/or power reductions. The results are verified through hardware synthesis on ASIC. Additional commentaries are provided on the differences with serial designs through comparisons with state-of-the-art implementations to facilitate the discussion on properties of block ciphers that prevent optimal serialization.
Acknowledgements

First of all, I wish to thank the supervisory team for their guidance and encouragement over the course of this undertaking. These include my principle supervisor Professor M. L. Dennis Wong and co-supervisors Dr. Ming Ming Wong, Professor Cishen Zhang and Ismat Hijazin. Their generosity in the sharing of expertise, knowledge and experience have been invaluable to the progress of this study. Constructive feedback provided in many aspects of this research improves the quality of the work done be it in the form of experimental results or academic publications. I have learnt a lot from them as a research student and am honoured the be part of the team.

It would be remiss of me to overlook the contribution from the university. A most sincere gratitude is directed to Swinburne University of Technology Sarawak Campus for offering my PhD studentship. Specifically, the Melbourne-Sarawak Research Collaboration Scheme has financially supported this research in the form of tuition fee waiver and monthly stipend. The same funding has also covered the expenses to attend several conferences as well as a visit to the main campus in Hawthorn, Victoria for a wonderful learning experience.

Special thanks are given to the research team led by Dr. Fakhrul Zaman Rokhani from Universiti Putra Malaysia (UPM). Their hospitality during my one-month attachment at UPM has been invaluable to the completion of the experiments required for the duration. The team has also been extremely helpful in tutoring and guiding me in the use of multiple computer-aided design tools that I am not familiar with.

In addition, I wish to express my deepest gratitude to my beloved family, whose support has been the greatest motivation that keeps me going. That goes double for my parents who have been extremely patient and tolerant of my various shortcomings for almost three decades.

Finally, I want to thank my friends and colleagues for being such wonderful human beings and for all the unforgettable memories. Special thanks go to Dr. Nguan Soon Chong, Dr. Wei Jing Wong, Zhi Hao Chang, Bih Fei Jong, Wenlong Jing, Nicholas Ching Yun Bong and Abdulkadir Lawan.
Declaration

I hereby declare that, to the best of my knowledge, this thesis contains no material that has been accepted, in whole or in part, for the award of any other academic degree or diploma. In addition, any material previously published or written by another person is fully acknowledged in accordance with the standard referencing practices.

JIA JUN TAY
2019
Contents

1 Introduction .......................... 1
   1.1 Challenges in Ubiquitous Computing ........................................... 1
   1.2 Logic Optimization for Economical Circuits .................................. 2
   1.3 Cryptography in Constrained Environments .................................... 4
   1.4 Research Objectives and Contributions ........................................ 6
   1.5 Thesis Outline ................................................................. 8

2 LMC Heuristic for Logic Optimization .................................................. 9
   2.1 Background ................................................................. 9
   2.2 Preliminaries .............................................................. 12
      2.2.1 Nomenclature ......................................................... 12
      2.2.2 Logic Basis (AND, XOR, NOT) ....................................... 12
      2.2.3 Multiplicative Complexity ............................................ 13
   2.3 LMC Heuristic .............................................................. 14
   2.4 Boyar-Peralta Two-Step Algorithm ............................................ 15
      2.4.1 Step 1: AND-Minimization ............................................ 16
      2.4.2 Step 2: XOR-Minimization ............................................ 16
   2.5 AND-Minimization in Multiple-Output Problems ............................... 20
   2.6 Summary ................................................................. 22

3 Enhanced Boyar-Peralta Algorithm for LMC Logic Optimization .............. 25
   3.1 Introduction .............................................................. 25
   3.2 Problem Statement and Motivation ............................................ 25
   3.3 Proposed Enhancements ...................................................... 27
3.3.1 Reduction of Algorithm Overhead in AND-Minimization . . . . . . 27
3.3.2 Sample Size Limitation in AND-Minimization . . . . . . . . . . . . 29
3.3.3 Solving Sequence for Multiple-Output Problem . . . . . . . . . . . . 31
3.3.4 Inclusion of Non-Linear Circuit in XOR-Minimization . . . . . . . 33
3.3.5 Circuit Depth Criterion for XOR-Minimization . . . . . . . . . . . . 34
3.3.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
3.4 Evaluation of Proposed Algorithm . . . . . . . . . . . . . . . . . . . . . 37
3.4.1 Single-Output Problem . . . . . . . . . . . . . . . . . . . . . . . . . . 39
3.4.2 Multiple-Output Problem . . . . . . . . . . . . . . . . . . . . . . . . 43
3.4.3 Circuit Depth . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.4.4 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
3.5 MCNC Benchmarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
3.6 Application: Stochastic Random Number Generator . . . . . . . . . . . 49

4 Deterministic AND-Minimization through Reed-Muller Decomposition 52
4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.2 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.2.1 Modulo-2 Arithmetic . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.2.2 Reed-Muller Expression . . . . . . . . . . . . . . . . . . . . . . . . . 53
4.3 Decomposition of PPRM Expressions . . . . . . . . . . . . . . . . . . . . 55
4.4 Tree Search Algorithm for Lower Bounded Problems . . . . . . . . . . . 57
4.4.1 Regarding Interchangeable Literals . . . . . . . . . . . . . . . . . . . 60
4.5 Product Sharing for Multiple-Output Problem . . . . . . . . . . . . . . . 62
4.5.1 Regarding Leaves of Higher Depth . . . . . . . . . . . . . . . . . . . 65
4.6 Performance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
4.6.1 Time Complexity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
4.6.2 Computation Time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
4.6.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
4.7 Quality of Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
4.7.1 Substitution Circuits . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.7.2 Majority Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
List of Figures

1.1 Paradigm shift in modern computing. ........................................ 1
1.2 Transforming the description of a function (a) into an equivalent circuit (b) is the main goal of logic design. ......................... 3
1.3 Cryptographic schemes can be applied alongside a secret key for data encryption. This process is reversible with either the same key (symmetric encryption) or a different key (asymmetric encryption). ............... 4
1.4 The four key security objectives describe the roles of cryptography in modern computing [1]. ................................................. 5
2.1 Both circuits (a) and (b) compute the same function but require different amount of logic gates. ........................................ 15
2.2 Boyar-Peralta two-step algorithm for LMC logic optimization. ......... 17
2.3 Boyar-Peralta AND-minimization step using an iterative randomized selection process. ................................................. 18
2.4 Boyar-Peralta XOR-minimization step using the SLP approach. ........ 21
2.5 Boyar-Peralta AND-minimization step for multiple-output non-linear problems. The randomized selection process with AND and XOR rounds are illustrated as a single step for brevity. ........................................ 23
3.1 A circuit consisting of an upper linear component, a middle non-linear component and a bottom linear component. ................. 33
3.2 Two linear circuits (a) and (b) computing the same functions $z_1, z_2, ..., z_4$. .................................................. 35
3.3 Distributions of optimized results for $f_1$: (a) Circuit size and (b) no. of operations. .................................................. 39
3.4 Distributions of optimized results for $f_2$: (a) Circuit size and (b) no. of operations. .................................................. 40
3.5 Distributions of optimized results for $f_3$: (a) Circuit size and (b) no. of operations.

3.6 Distributions of optimized results for $F$: (a) Circuit size and (b) no. of operations.

3.7 Distribution of number of AND gates required in optimized results for $F$.

3.8 Distributions of results by proposed algorithm with opposing solving sequence: (a) Circuit size, (b) no. of operations and (c) no. of AND gates.

3.9 19-gate implementation of the SBoNG substitution circuit. The 4-bit inputs are $X = \{x_1, x_2, x_3, x_4\}$ and the 4-bit outputs are $Y = \{y_1, y_2, y_3, y_4\}$.

4.1 Tree diagram for an $n = 3, d = 3$ function.

4.2 Tree diagram with product sharing.

4.3 Size-15 depth-8 implementation of Canright’s $GF(2^4)$ inversion circuit $F_{inv}$.

4.4 Size-15 implementation of PRESENT S-Box $F_{PRESENT}$.

4.5 Size-13 implementation of PRESENT S-Box $F_{PRESENT}$ after gate replacement.

4.6 Size-12 implementation of the majority function with $n = 5$.

4.7 The AES S-Box as a three-part circuit: top linear component $U$, middle non-linear component $M$ and bottom linear component $B$.

5.1 mCrypton encryption process.

5.2 PRESENT encryption process.

5.3 Piccolo encryption process. Note that each instance of round key $rk$ is unique as generated through key scheduling.

5.4 LED-128 encryption process.

5.5 PRINCE encryption process. Note that the first and last key additions include both round keys and the respective whitening key.

5.6 SIMON-64/128 encryption process. The 64-bit input block is split into two 32-bit blocks $x_1$ (most significant bits) and $x_0$ (least significant bits) in the Feistel network.

5.7 Midori-64 encryption process. Key additions in the middle 15 rounds include a sparse round constant per round.

6.1 Finite field multiplication circuits for (a) $\times 2 \mod p$ and (b) $\times 3 \mod p$. 

xi
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Single finite field multiplication circuit for $\times 2 \mod p$ and $\times 3 \mod p$ computations.</td>
</tr>
<tr>
<td>6.3</td>
<td>Round-based architecture for PRINCE cipher.</td>
</tr>
<tr>
<td>6.4</td>
<td>Round-based architecture for PRINCE cipher with added demultiplexers.</td>
</tr>
<tr>
<td>6.5</td>
<td>LFSR configuration for the round constants in SIMON-64/128 encryption.</td>
</tr>
<tr>
<td>6.6</td>
<td>Round key implementation for Midori cipher with a 64-bit multiplexer.</td>
</tr>
<tr>
<td>6.7</td>
<td>Reduced round key implementation for Midori cipher.</td>
</tr>
<tr>
<td>6.8</td>
<td>Circuit area for the different configurations of ciphers. (a) Silterra 180nm. (b) Silterra 130nm.</td>
</tr>
<tr>
<td>A.1</td>
<td>Top linear component $U$ of the proposed AES S-Box. 8-bit inputs are $x_0, x_1, ..., x_7$. 22-bit outputs are $x_0, U_0, U_1, ..., U_{22}$ excluding $U_4$ and $U_{11}$.</td>
</tr>
<tr>
<td>A.2</td>
<td>Middle non-linear component of the proposed AES S-Box. 22-bit inputs are $x_0, U_0, U_1, ..., U_{22}$ excluding $U_4$ and $U_{11}$. 18-bit outputs are $N_0, N_1, ..., N_{17}$.</td>
</tr>
<tr>
<td>A.3</td>
<td>Bottom linear component of the proposed AES S-Box. 18-bit inputs are $N_0, N_1, ..., N_{17}$. 8-bit outputs are $y_0, y_1, ..., y_7$.</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Summary of results and t-tests for single-output problems.</td>
<td>42</td>
</tr>
<tr>
<td>3.2</td>
<td>Summary of results for multiple-output problem $F$.</td>
<td>46</td>
</tr>
<tr>
<td>3.3</td>
<td>t-tests for multiple-output problem $F$.</td>
<td>46</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary of results for linear optimization problems $M_1, M_2$ and $M_3$.</td>
<td>47</td>
</tr>
<tr>
<td>3.5</td>
<td>Optimization results for 11 MCNC benchmark functions.</td>
<td>48</td>
</tr>
<tr>
<td>3.6</td>
<td>Comparison of circuit size between the original and proposed SBoNG substitution circuits.</td>
<td>51</td>
</tr>
<tr>
<td>3.7</td>
<td>FPGA implementation results for both 8-bit SBoNG RNSs.</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison of computation time.</td>
<td>68</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of logic gate count on $F_{inv}$ and $F_{PRESENT}$.</td>
<td>71</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of circuit complexities between the proposed AES S-Box and existing works</td>
<td>75</td>
</tr>
<tr>
<td>5.1</td>
<td>Summary of the seven chosen lightweight block ciphers.</td>
<td>82</td>
</tr>
<tr>
<td>6.1</td>
<td>Properties of S-Boxes under study. $f_1, f_2, ..., f_4$ are the four functions of a 4-bit S-Box in ascending order of bit significance.</td>
<td>88</td>
</tr>
<tr>
<td>6.2</td>
<td>Optimization goals addressed by individual methodology.</td>
<td>98</td>
</tr>
<tr>
<td>6.3</td>
<td>Applicability of proposed methodologies on targeted lightweight block ciphers.</td>
<td>98</td>
</tr>
<tr>
<td>6.4</td>
<td>Summary of relevant metrics.</td>
<td>100</td>
</tr>
<tr>
<td>6.5</td>
<td>Comparison of applicable ciphers before and after low multiplicative complexity S-Box optimization.</td>
<td>102</td>
</tr>
<tr>
<td>6.6</td>
<td>Comparison of Piccolo and LED ciphers before and after circuit sharing in finite field multiplication.</td>
<td>103</td>
</tr>
<tr>
<td>Table</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.7</td>
<td>Comparison of PRINCE cipher before and after circuit gating.</td>
<td>104</td>
</tr>
<tr>
<td>6.8</td>
<td>Comparison of applicable ciphers using LFSR and combination logic for</td>
<td></td>
</tr>
<tr>
<td></td>
<td>round constant generation.</td>
<td>105</td>
</tr>
<tr>
<td>6.9</td>
<td>Comparison of Midori and LED ciphers before and after round key circuit reduction</td>
<td>106</td>
</tr>
<tr>
<td>6.10</td>
<td>Proposed implementations for the seven lightweight ciphers of interest</td>
<td>107</td>
</tr>
<tr>
<td>6.11</td>
<td>Area and performance results for implementations on Silterra 180nm process.</td>
<td>108</td>
</tr>
<tr>
<td>6.12</td>
<td>Area and performance results for implementations on Silterra 130nm process.</td>
<td>109</td>
</tr>
<tr>
<td>6.13</td>
<td>Power and energy consumption results for implementations on Silterra 180nm process.</td>
<td>111</td>
</tr>
<tr>
<td>6.14</td>
<td>Power and energy consumption results for implementations on Silterra 130nm process.</td>
<td>112</td>
</tr>
<tr>
<td>6.15</td>
<td>Comparison between different architectures for PRESENT cipher.</td>
<td>116</td>
</tr>
</tbody>
</table>
## Commonly Used Acronyms

<table>
<thead>
<tr>
<th>Acronym</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>CAD</td>
<td>Computer-aided design</td>
</tr>
<tr>
<td>CFA</td>
<td>Composite field arithmetic</td>
</tr>
<tr>
<td>ESOP</td>
<td>Exclusive-OR sum of products</td>
</tr>
<tr>
<td>FN</td>
<td>Feistel network</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-programmable gate array</td>
</tr>
<tr>
<td>FPRM</td>
<td>Fixed Polarity Reed-Muller</td>
</tr>
<tr>
<td>GE</td>
<td>Gate equivalent</td>
</tr>
<tr>
<td>GF</td>
<td>Galois field</td>
</tr>
<tr>
<td>GFN</td>
<td>General Feistel network</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>LFSR</td>
<td>Linear-feedback shift register</td>
</tr>
<tr>
<td>LMC</td>
<td>Low multiplicative complexity</td>
</tr>
<tr>
<td>LUT</td>
<td>Lookup table</td>
</tr>
<tr>
<td>MPRM</td>
<td>Mixed Polarity Reed-Muller</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>PLA</td>
<td>Programmable logic array</td>
</tr>
<tr>
<td>PPRM</td>
<td>Positive Polarity Reed-Muller</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio-frequency identification</td>
</tr>
<tr>
<td>RNS</td>
<td>Random number source</td>
</tr>
<tr>
<td>S-Box</td>
<td>Substitution-box</td>
</tr>
<tr>
<td>SC</td>
<td>Stochastic computing</td>
</tr>
<tr>
<td>SLP</td>
<td>Shortest linear path</td>
</tr>
<tr>
<td>SN</td>
<td>Stochastic number</td>
</tr>
<tr>
<td>SNG</td>
<td>Stochastic number generator</td>
</tr>
<tr>
<td>SOP</td>
<td>Sum of products</td>
</tr>
<tr>
<td>SPN</td>
<td>Substitution-permutation network</td>
</tr>
<tr>
<td>TSA</td>
<td>Tree search algorithm</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very-large-scale integration</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Challenges in Ubiquitous Computing

The era of modern computing began in 1936 with the introduction of the universal Turing machine by Alan Turing [2]. Since then, technology advancements have progressed the computing paradigm through several phases as depicted in Figure 1.1 [3]. The contemporary concept resulting from the paradigm shift is the idea of ubiquitous computing. At its core, ubiquitous computing is a vision proposed by Mark Weiser [4] that interconnects everyday devices through the embedding of computing power. Depending on the aspects emphasized or objects connected, ubiquitous computing may also be referred to as pervasive computing [5], physical computing [6] or the Internet of Things (IoT) [7]. The promise of the concept is to make technologies disappear into the background and allow users to benefit from them without focus of attention. On the consumer level, ubiquitous computing is poised to bring quality of life improvements by enabling applications such as automated smart house [8,9] and personal health care [10,11]. At the same time, the mass deployment of ubiquitous computing devices also promotes several benefits to businesses such as increased productivity in factories [12], reduced logistic costs [13], and enabling location-based services among others [14].

![Figure 1.1: Paradigm shift in modern computing.](image)

There exist several challenges to the universal adoption of ubiquitous computing, with two of them being the chief motivations behind the work done in this research. Firstly, by definition of the concept, ubiquitous computing devices have to be pervasive and deployed in large volume. This in turn implies severe cost constraints on the technology
used. For instance, design of hardware implementations such as application-specific integrated circuits (ASICs) demands cost functions derived from metrics such as circuit area and power consumption to be minimal. Radio-frequency identification (RFID) is widely regarded as the enabler for ubiquitous computing by virtue of enabling object tracking in the form of uniquely identifiable electronic product codes (EPCs). Low cost RFID tags are among the most pervasive devices and are heavily constrained in computing resources. Juels and Weis [15] reported that a typical low cost RFID tag has only approximately 1000 to 10000 gates available for computations. The authors noted how the amount of computing resources available is weak and faces similar limitations as the working memory of most human beings. While abundant computing power can be made available in the course of time on account of Moore’s Law, the pace of advancement in transistor density has slowed down recently [16]. Therefore, many foresee an increase in the demand for lightweight (hence cheaper) design or optimization techniques to enable complex computations in limited hardware.

The second challenge is the inherent security risks that stem from the pervasive nature of ubiquitous computing. Such devices often carry sensitive information especially in applications related to military, finance, automotive, or health care. In addition, they are commonly deployed in hostile environments where adversaries have easy access to the devices [17]. As a result, privacy and security concerns are seen as the prominent obstacles to the success of ubiquitous computing [18]. This scenario is further aggravated by the limited computing resources available to such devices as per the reasoning described in the previous paragraph. Consequently, conventional cryptographic solutions are too expensive to implement in these devices [19]. This has prompted research interests in designing new lightweight primitives and hardware optimization efforts to meet the stringent requirements in area and power.

1.2 Logic Optimization for Economical Circuits

The foundation of modern computing devices is the two-valued binary logic system. To perform calculations or process information, logic circuits rely on basic building blocks known as logic gates to process input signals into the desired outputs. Logic design (a.k.a. logic synthesis) is then the exercise of interconnecting these basic logic building blocks to perform the desired function [20]. While functional correctness is the main goal of this exercise, it has become increasingly important to produce logic circuit that fulfill several design parameters as to be beneficial (or mandatory) to the targeted applications. Therefore, a process to find equivalent implementations of a logic circuit that fulfill specific constraints becomes necessary as part of logic design. This process is referred to as logic optimization or minimization. The three main dimensions of logic optimization
are:

- **Area.** A measure of physical circuit size or gate count consumed by a circuit.
- **Speed.** A general measure of how fast a circuit can be clocked at.
- **Power.** A measure of the power consumption required by a circuit in terms of dynamic power and static power.

As proven by Buchfuhrer and Umans [21][22], logic minimization is a \(P^{2}\)-complete problem. This means that it takes polynomial time to determine the optimal implementation using minimal number of gates given a circuit minimization problem. Therefore, to achieve economical circuits for specific applications, designers often rely on a variety of heuristics [23]. These heuristics become the foundation for various logic optimization algorithms, allowing good results to be discovered in practical computation time. However, it is important to reiterate that these results are often sub-optimal, i.e. they are not the smallest circuit to compute the same function.

In the early stages, logic optimization involves methods such as the Karnaugh map [24] and the Quine-McCluskey algorithm (a.k.a. the tabular method) [25] which are discussed in almost every logic design textbook [20][26][27]. Particularly, the former is designed to be intuitive and suitable for manual derivation while the latter enables automation of logic optimization through the use of computers. However, following the rapid advancement in technology, circuits have become too large and complex for these algorithms to optimize efficiently. As a result, the industry saw the introduction of the Espresso logic minimization heuristic in 1984 [28]. Espresso-based algorithms produce results that are closely approximated to the global minimum. More importantly, the algorithm is also more efficient than prior optimization techniques by several orders of magnitude in computation time and memory usage, hence capable of solving logic minimization problems with up to tens of input/output variables. Accordingly, the
Espresso heuristic is incorporated into modern logic synthesis tools for both ASIC and field-programmable gate array (FPGA) designs to minimize the desired circuits.

While there are no major quandaries surrounding the capabilities of the Espresso heuristic, there exists potential for new heuristics that may outperform the former in quality of results under specific circumstances. The Espresso heuristic, first and foremost, is designed with algorithmic efficiency in mind so as to solve for large-scale minimization problems in practical time and resources. However, many modern circuits compute functions that can be decomposed naturally into smaller components or even repeated use of the same small components. Boyar et al. [29] illustrated some examples such as arithmetic units built from full adders, matrix multipliers built from multiplications with smaller submatrices and cryptographic functions with repeated use of linear and non-linear transformations. Therefore, it is reasonable to speculate on logic optimization heuristics that may not be feasible for large-scale problems but enable closer approximations to the optimal result for small components. Many applications (e.g. RFID in food packaging [30]) face obstacles related to cost constraints (due to large-scale deployment of pervasive devices) that hinder universal adoption. Ultra-constrained environments in ubiquitous computing also prevent the use of off-the-shelf components that have lower non-recurring engineering (NRE) costs [31]. Advancements in the field of logic optimization can be worthwhile in this regard as the reduction in circuit area ultimately leads to reduction in the manufacturing cost of said devices (especially in large volume).

1.3 Cryptography in Constrained Environments

Cryptography is the art of obscuring a piece of information such that third party adversaries are unable to make sense of the information without knowledge of a secret key. The history of cryptography can be traced back to the year 1900 BC where classical cryptography existed in the form of transposition and substitution ciphers. One such example is the Caesar cipher, where each letter in a message is replaced by the letter three positions later in the alphabet sequence. Classical cryptography ciphers were invented mainly to ensure secrecy in communication, especially for military related purposes.

![Figure 1.3: Cryptographic schemes can be applied alongside a secret key for data encryption. This process is reversible with either the same key (symmetric encryption) or a different key (asymmetric encryption).](image-url)
Modern cryptography has evolved to encompass significant roles in a variety of applications such as secure computation, identity authentication, integrity checking and much more. There are four key security objectives that are critical to the concept of modern cryptography [1]: confidentiality, integrity, authentication, and non-repudiation. To fulfill services in line with said objectives, modern cryptographic primitives are designed with assumptions on computational hardness. In general, it is impractical to implement cryptographic solutions that satisfy unconditional security; rather, a cryptographic scheme is deemed computationally secure if it is sufficiently difficult to be broken by an adversary, i.e. the time or cost required to break the algorithms severely outweigh the potential profit to be gained by the adversary.

### Figure 1.4: The four key security objectives describe the roles of cryptography in modern computing [1].

“Secure Cyberspace” is listed as one of the grand challenges for engineering in the 21st century by the National Academy of Engineering [32]. The trends of ubiquitous computing and IoT promote interconnections of small inexpensive devices to form robust networks capable of decision making and actuation without human input. However, ensuring secured communications between the enormous number of connected devices is challenging as many of such devices are not equipped with sufficient hardware resources for conventional security measures [19]. More importantly, the security strength of an IoT network is only as strong as its weakest link. A successful tampering of an edge sensor node in the system can have severe ramifications as it can influence the decision making and actions of other devices. To meet the harsh constraints in area and power on low end devices, researchers are embracing the opportunity to design new lightweight cryptographic primitives that are better suited for this nature of applications. Hardware optimization of these primitives is also an active research field to design economical circuits that compute the complex transformations involved in said primitives.

---

1Unconditionally secure implies a system that is resistant to any cryptanalytic attack given infinite computing power and resources available to the adversary. Computational security, on the other hand, assumes limited computing power and resources on the adversary (as they are in practice).
Chapter 1: Introduction

1.4 Research Objectives and Contributions

Following the emerging trends in modern computing, devices operating in the lower end of the spectrum are tasked with increasingly complex computation workload. This has drawn interests from researchers to explore new logic optimization techniques to yield further improvements to circuit area to cope with hardware limitations in constrained environments and achieve reduction in deployment costs. Low multiplicative complexity (LMC) heuristic (a.k.a. Boyar-Peralta heuristic) based on the proposal by Boyar et al. [29] is a novel concept for low gate count logic optimization. The heuristic showed promising potential when it successfully derived a smaller $GF(2^4)$ AES S-Box [33], a circuit thought to be well optimized over the last decade. Nevertheless, the original Boyar-Peralta algorithm for LMC logic optimization has several undesirable characteristics [34]. Chief among them is the reliance on a randomized selection procedure as a core part of the algorithm which naturally results in skepticism towards the reliability and consistency of the algorithm.

At the same time, security and privacy challenges are also major obstacles to the advancement of ubiquitous computing [35]. To cope with the increased demand on security measures for low end devices, a variety of lightweight block ciphers have been proposed over the last decade [36]. These cryptographic algorithms are specially designed to have specific advantages in hardware requirement, power/energy consumption or other metrics that are more relevant in ultra-constrained environments. Since then, hardware optimization of these ciphers has been largely concentrated on architecture serialization [37,38]. In [39], it was reported that latency exceeding 50 cycles can render a cipher unworkable in RFID related applications. This sentiment was then referenced in National Institute of Standards and Technology’s (NIST) latest report on lightweight cryptography [40]. This raises concern on state-of-the-art implementations of the lightweight ciphers as serialization typically increases the latency of a cipher by severalfold (often over 100 cycles).

One of the main objectives of this thesis is to study the principles behind the LMC heuristic as a promising new approach in logic optimization. Of particular interests are the methodologies to mitigate the side effects caused by randomness in the Boyar-Peralta algorithm to improve overall consistency in regards to quality of results and computation time. Given the potential of the heuristic in low gate count logic optimization, contribution towards an enhanced optimization algorithm improves the ease of use in practical problems. Simultaneously, the potential of a deterministic approach in LMC logic optimization is also of significant research value as it completely eliminates any concern surrounding the aforementioned consistency issues. Therefore, this study investigates a novel approach based on algebraic manipulation that can be applied to a subset of logic minimization problems to achieve LMC implementations without element of ran-
domness. A strong focus is given to ensure that the deterministic approach is capable of at least competitive (if not better) quality of results compared to the Boyar-Peralta algorithm.

This thesis also aims to explore hardware optimization methodologies for lightweight block ciphers. A total of seven primitives are studied for optimization: mCrypton [41], PRESENT [42], Piccolo [43], LED [44], PRINCE [45], SIMON [46], and Midori [47]. In conjunction with their role in lightweight applications, optimization efforts are focused on expensive cryptographic transformations that are common among block ciphers with the goal of area and/or power reductions. Due to intricate design differences between the ciphers of interest, the same methodology may not derive the same benefit to all applicable ciphers and thus have to be verified through careful evaluation of the hardware synthesis results. The intention of this study is to propose hardware efficient implementations for the selected lightweight ciphers without excessive latency costs observed in state-of-the-art serial architectures.

The major contributions of this thesis are declared as follows:

1. An improved algorithm for LMC logic optimization is presented. The algorithm is an enhanced version of the Boyar-Peralta algorithm [34] with a series of refinements and new proposals involving (a) algorithm overhead reduction, (b) restriction to sample space expansion, (c) guidelines on solving sequence, (d) XOR-minimization of non-linear circuit and (e) new circuit depth criterion. The proposed algorithm demonstrated improvements in average quality of results and reduced variation compared to the original, with increased number of solutions that closely approximate the best-case found. The aforementioned improvements are crucial for consistency and reliability in a randomized algorithm. More importantly, the enhancements enabled significant improvement in the ability to derive solutions of optimal multiplicative complexity in a multiple-output problem which is critical in regards to the premise of the LMC heuristic.

2. A use case of the enhanced Boyar-Peralta algorithm is demonstrated through the optimization of a stochastic number generator (SNG). Specifically, the enhanced algorithm is applied to optimize the non-linear substitution circuit used in the pseudo-random number generator SBoNG proposed in [48]. The non-linear substitution circuit, whose area increases linearly with the length of stochastic number (SN) desired, is the most expensive component of SBoNG. The implementation generated by the enhanced Boyar-Peralta algorithm is significantly smaller than the original implementation in [48] that was generated using computer-aided design (CAD) tools.

3. A novel tree search algorithm (TSA) is proposed to achieve LMC implementations
Chapter 1: Introduction

for a subset of logic minimization problems with lower bounded multiplicative complexity. Optimal implementations (in terms of multiplicative complexity) of these functions can be derived through decomposition and manipulation of the Fixed Polarity Reed-Muller (FPRM) expressions that describe said functions. The design philosophy behind the proposed TSA is to leverage this property to eliminate the need for randomness in the logic optimization process. The algorithm is proven to be competitive, with quality of results surpassing circuits optimized using the same heuristic reported in existing works.

4. A compact implementation of the AES S-Box is proposed based on the combination of tower field architecture and application of the proposed TSA on the $GF(2^4)$ multiplicative inversion circuit. The proposed design showed improvements in both circuit size and circuit depth in comparison to the current state-of-the-art and represents the smallest area for the function at the time of this writing.

5. Area and power optimized implementations of seven lightweight block ciphers are proposed. Cryptographic transformations involved in the targeted ciphers are reviewed and hardware optimization is devised for the necessary functions. These include the substitution circuit, finite field multiplication circuit, key scheduling mechanism, and round constant circuit. The proposed methodologies do not affect circuit latency, allowing each implementation to meet the 50-cycle requirement in [39]. All designs are synthesized on ASIC using Silterra’s 180nm and 130nm technologies to validate the outcome of the proposals. Hardware performances of the proposed implementations contribute to NIST’s current effort to standardize a lightweight cryptographic primitive [40].

1.5 Thesis Outline

This section provides an overview on the structure of the rest of the chapters in this thesis. Chapter 2 provides a review of important heuristics in logic design as well as the introduction to the LMC heuristic and the original Boyar-Peralta algorithm. In Chapter 3 concerns associated with the original Boyar-Peralta algorithm are outlined alongside five proposed enhancements to address the expressed concerns. Chapter 4 presents a novel deterministic algorithm to achieve optimal multiplicative complexity for the Boyar-Peralta heuristic. Chapter 5 introduces lightweight block ciphers and existing works on the hardware optimization of these circuits. This is followed by the proposed methodologies for hardware optimization of round-based lightweight block ciphers in Chapter 6. To conclude the thesis, Chapter 7 summarizes the contents of this thesis and discusses potential future works related to this study.
Chapter 2

LMC Heuristic for Logic Optimization

2.1 Background

To better understand its role in logic optimization, it is necessary to precede the introduction of the LMC heuristic with discussion on the inherent qualities of a “good” logic optimization heuristic. Complications surrounding the problem of large-scale logic optimization can be referenced in [49]. Firstly, given the large number of inputs and outputs (upwards of 50), it is understandable that deriving the global minimum solution through an exhaustive procedure is no longer sensible. While logic optimization heuristics provide the avenues to derive good solutions in practical time, it is not possible to determine whether the solutions are optimal due to the inability to precisely identify the global minimum solution. Consequently, a “good” heuristic is justified by its ability to give good solutions through extensive trials on any arbitrary logic optimization problems and is accepted with the assumption that it will give good solutions on other similar problems.

When comparing two logic optimization heuristics, time-space complexity [50] or CPU time [51] analysis on the algorithms provide the means to quantify the efficiency of each algorithm. The quality of results, however, need to be proven by comparing the solutions obtained through applications of said algorithms on arbitrary logic optimization problem or a subset of it with specific properties. If superior quality of results is observed, a heuristic is then accepted to be better than the other on the same subset of problems where the comparison is demonstrated.

The Espresso logic minimization heuristic [28] is one of the most efficient and flexible heuristic for logic optimization. The algorithm is incorporated as a standard combinational logic optimization procedure in many modern logic synthesis tools including publicly available logic synthesis systems such as SIS by University of California, Berke-
Chapter 2: LMC Heuristic for Logic Optimization

ley 52 and BOLD by University of Colorado, Boulder 53. Fundamentally, the Espresso algorithm is designed for two-level logic minimization not unlike the Quine-McCluskey algorithm 25. Hence, the goal of the algorithm is to minimize the number of product terms in a sum of products (SOP) expression which corresponds directly to the area cost of two-level logic arrays such as programmable logic array (PLA).

However, outside of two-level PLAs, multilevel implementations generally give lower area costs compared to their two-level equivalent 54. This is attributed to the increased potential of reusing intermediate signals and the degree of freedom allowed in the solution space 55. For these reasons, multilevel logic (a.k.a. random logic) optimization is more relevant in the field of lightweight applications where circuit area is a major concern in contrast to clock speed. The distinguishing factor for a good heuristic in the space of multilevel logic optimization is the extent to which said heuristic can exploit the freedom allowed in the design problem. However, the increased freedom also means that multilevel logic optimization is significantly more complex than two-level logic optimization.

In 55, it is highlighted that the approach to multilevel logic optimization can be generalized into two categories: (a) rule-based or local-transformation methods and (b) algorithmic approaches. Rule-based methods operate by identifying specific patterns in the arrangement of logic gates in the circuit and replacing them with equivalent alternatives (usually better in desired metrics). Regardless, the capabilities of rule-based methods are often limited to circuits constructed using specific logic gate types only. At the same time, they are considered to be local in nature as they do not offer a global perspective on the circuit. Example of rule-based methods include LSS 56, SOCRATES 57 and LORES 58. On the other hand, algorithmic approaches rely on technology-independent algorithms to manipulate the target functions. These algorithms can involve algebraic operations such as decomposition, factorization, extraction, resubstitution, and elimination. Subsequently, a technology-mapping step is performed to map the manipulated functions into the set of gates available in the target technology. Examples include MIS 54 and BOLD 59.

All of the multilevel logic optimization algorithms mentioned above are relatively old, having been introduced in the 1980s. Regardless, the principles and philosophy behind the algorithms remain relevant even today. Multilevel optimization algorithms used in modern commercialized CAD tools are typically confidential and proprietary programs not subjected to public scrutiny 49. Therefore, it is difficult to review the heuristics involved. However, it is at least possible to compare the quality of results based on the solutions derived from said CAD tools using identical logic optimization problem.

Up to this point, the aforementioned logic optimization techniques largely operate within the logic basis (AND, OR, NOT), i.e. they construct a circuit using only AND, OR and NOT gates. This set of logic gates are chosen because they are functionally complete, i.e.
sufficient to realize any arbitrary functions. It is possible to observe the use of other logic gate types in some algorithms where certain gates are replaced post optimization (e.g. rule-based methods). In 1990, Sasao and Besslich \cite{60} made an important deduction that PLAs implemented using the logic basis (AND, XOR, NOT) require fewer product terms on average than conventional PLAs. The implication of this observation is that logic optimization using the logic basis (AND, XOR, NOT) has the potential to consume lower gate count than their equivalents over the logic basis (AND, OR, NOT) in two-level logic implementations. Although the same deduction does not apply directly for multilevel logic optimization, Koda and Sasao \cite{61} highlighted that logic design using XOR gates can benefit arithmetic circuits and error-correcting circuits in terms of gate count reduction.

Unlike logic optimization over (AND, OR, NOT), both two-level and multilevel optimization techniques over (AND, XOR, NOT) have not reached the same maturity level and remain an active field of research today. The difference in logic basis means that most of the logic optimization techniques designed for the former cannot apply to the latter (at least not directly). Two-level optimization has been focused on determining the best polarity for each input variables in order to minimize the number of product terms in ESOP. On the other hand, multilevel optimization generally focuses on restructuring two-level logic networks using methods such as Reed-Muller restructuring, Functional Decision Diagram (FDD), Davio expansion, and rule-based optimization, all of which demonstrated improvements in quality of results and/or computation time against multilevel optimization algorithms that operate over the logic basis (AND, OR, NOT).

More recently, logic optimization over (AND, XOR, NOT) saw interesting development with the introduction of a new heuristic by Boyar et al. in \cite{29}. The proposed heuristic is based on the concept of LMC which the authors have studied intensively in \cite{70,71}. The authors then devised a search algorithm for LMC-based combinational logic optimization as patented in \cite{34}. The heuristic gained significant traction when it successfully reduced the circuit size of a CFA-optimized AES S-Box in \cite{33}. This achievement is impressive considering the AES S-Box is a complex circuit well studied and optimized over the last two decades \cite{72,77}. The same sentiment is echoed by Courtois et al. in \cite{78} where a low gate count implementation of the lightweight PRESENT S-Box is found to agree with the LMC heuristic. At the same time, the algorithm is also flexible in execution as it can be tweaked to include considerations for other important parameters such as circuit depth \cite{79}.

The evident ability to derive good solutions for complex substitution circuits makes

\footnote{On average, exclusive-OR sum of products (ESOP) requires 11% less number of product terms than conventional SOP. \cite{61}.}
LMC heuristic a prime candidate for study as a promising heuristic for low gate count logic optimization. It is also viable in achieving further area reduction when applied on components in a circuit that has already been optimized through other means \[29\]. Before elaborating on the heuristic itself and the associated logic optimization algorithm, the nomenclature and the preliminary knowledge necessary to facilitate further discussions are first defined as follows.

### 2.2 Preliminaries

#### 2.2.1 Nomenclature

The relationship between the input(s) and output(s) of a function is often described with a logic expression. The following definitions are established for key terms that are important in identifying different elements in these expressions. Most of the definitions used in this thesis follow their respective definitions in \[54\].

**Definition 1.** A variable is a symbol representing a single input for a function. For example, \(x_1, x_2, ..., x_n\) represent the set of variables for an \(n\)-input function.

**Definition 2.** A literal refers to a variable or its negation. For example, \(x_1\) and \(\overline{x_1}\) are literals.

**Definition 3.** A cube is the conjunction of one or more literal(s). For example, given a function \(f = x_1 x_2 x_3 \oplus x_1 x_2 \oplus x_1\), its cubes are \(x_1 x_2 x_3\), \(x_1 x_2\) and \(x_1\).

**Definition 4.** An expression is the exclusive disjunction or XOR of one or more cube(s). For example, \(f = x_1 x_2 x_3 \oplus x_1 x_2 \oplus x_1\) is an expression.

**Definition 5.** The degree of an expression is an integer \(d\) which implies the number of literals present in the cube of an expression that has the highest number of literals. For example, the function \(f = x_1 x_2 x_3 \oplus x_1 x_2 \oplus x_1\) has a degree of \(d = 3\).

**Definition 6.** A truth vector is the vector representation for a truth table of a function such that \(f = [f(0), f(1), ..., f(2^n - 1)]^T\). For example, the truth vector for the function \(f = x_1 x_2\) is written as \(f = [0, 0, 0, 1]^T\).

#### 2.2.2 Logic Basis (AND, XOR, NOT)

A logic basis describes a list of logic gates available to a logic optimization algorithm for the construction of a circuit to perform a desired function. In order to do so, it is paramount for a logic basis to be functionally complete, i.e. sufficient to construct

\[\text{This definition is slightly different from \[54\] where it implies disjunction of cubes instead of exclusive disjunction. This is due to the difference in logic basis.}\]
any arbitrary function using only the logic operators available in the basis. The most popular functionally complete logic basis is (AND, OR, NOT). This logic basis is featured in classic logic minimization techniques such as Karnaugh mapping [24] and the Quine-McCluskey (and similar) method [25,80]. Many modern optimization algorithms rely on this logic basis as well to first satisfy the functionality of a circuit before attempting substitution using a variety of other logic gates for further savings.

Logic basis (AND, XOR, NOT) is an alternative to the logic basis (AND, OR, NOT). The functional completeness of the logic basis can be easily proven with relation to the logic basis (AND, OR, NOT). This is because the function of an OR gate can be substituted using AND and XOR gates as demonstrated in (2.2.1).

\[
\text{OR}(a, b) = \text{XOR}(\text{AND}(a, b), a, b)
\]  

(2.2.1)

However, when designing logic minimization algorithm using the logic basis (AND, XOR, NOT), the set of logic operators are often shortened to just (AND, XOR). This is because NOT gates are only required when constructing negative functions, i.e. \( f(0) = 1 \). In this case, one can optimize the positive function \( f' \) using just AND and XOR operators and terminate the circuit with a NOT gate to achieve the desired transformation.

In addition, the following definitions are made in regards to the linearity of an XOR-AND circuit:

**Definition 7.** A circuit within the logic basis (AND, XOR) is said to be **non-linear** if it contains AND gates.

**Definition 8.** A circuit within the logic basis (AND, XOR) is said to be **linear** if it contains only XOR gates.

### 2.2.3 Multiplicative Complexity

There are several complexity measures to classify a circuit under evaluation. Given a Boolean function \( f \), Sipser [81] outlined two major notions of circuit complexity namely:

- Circuit-size complexity: Minimal size of any circuit computing the function \( f \).
- Circuit-depth complexity: Minimal depth of any circuit computing the function \( f \).

In the same vein, multiplicative complexity is a measure of the non-linearity of a circuit [82]. A detailed definition for multiplicative complexity is given as follows:
Definition 9. The multiplicative complexity $c_{\Lambda}(f)$ of a function $f$ is the minimal number of multiplication (AND gates) required to realize the function over the logic basis (AND, XOR, NOT). The same notation can be used on a set of functions of the same $n$-input variables to describe the multiplicative complexity of a multiple-output circuit as a whole, i.e. $c_{\Lambda}(f_0, f_1, ..., f_i)$ where $f_0, f_1, ..., f_i \in \langle x_1, x_2, ..., x_n \rangle$.

Determining the complexity measures of an unrestricted function is known to be a difficult problem in complexity theory and multiplicative complexity is no exception. However, several works over the years have allowed some guidelines to be drawn regarding the topic. In particular, works in [83] and [84] have established Lemmas 1 and 2 respectively. The first gives the lower bound for the multiplicative complexity of a function, whereas the latter gives the upper bound for functions with up to five variables.

Lemma 1. Given a function $f$ of degree $d$, the multiplicative complexity is at least $d - 1$, i.e. $c_{\Lambda}(f) \geq d - 1$.

Lemma 2. Given an $n$-variable function $f$, its multiplicative complexity is at most $n - 1$ as long as $n \leq 5$, i.e. $c_{\Lambda}(f) \leq n - 1$ given $n \leq 5$.

In regards to the LMC heuristic, both the lower and upper bound rules serve as useful guidelines to estimate the range of possible multiplicative complexity for a function.

2.3 LMC Heuristic

The premise of a logic optimization algorithm based on multiplicative complexity is heavily based on the LMC heuristic (a.k.a. the Boyar-Peralta heuristic) introduced in [29]. The key proposition is as follows:

Proposition 1. Given a function $f$ to be implemented over the logic basis (AND, XOR, NOT), the circuit implementations with the minimal number of AND gates are gate-efficient.

In essence, the LMC heuristic suggests optimizing the gate count of a combinational circuit by prioritizing the reduction of AND gates to the bare minimum. Figure 2.1 shows two XOR-AND circuits that compute the function $f = x_1x_2 \oplus x_1x_3$. Circuit (b) is able to compute the function with less gate count by minimizing the number of AND gates. Note that the symbols $\otimes$ and $\oplus$ represent an AND gate and an XOR gate respectively.

Technically, the heuristic is suggested based on extensive observation and experimentation but lacks substantial theoretical proofs (example in Figure 2.1 is not a proof of concept). However, there are some rationales that support the proposition as an interesting foundation for logic optimization.
In logic synthesis, decomposition of Boolean functions is known to produce area-efficient implementations [85]. However, the optimization process has enormous computational complexity and is impractical for automation. The principle of this approach however, is similar to the LMC heuristic. This is because decomposition of expressions reduces the number of multiplications required and multiplication over the Galois Field $GF(2)$ is synonymous with AND gates (more on this in Chapter 4).

In addition, it is noted that a circuit constructed using minimal number of AND gates naturally retains a large number of purely linear sections. This property is particularly desirable as purely linear circuits can be optimized efficiently using algorithms designed to solve shortest linear path (SLP) problems. For instance, Paar [86] has a greedy algorithm for this purpose that inspired a number of variations.

All in all, the heuristic showed potential and to put it into practice, the authors devised a two-step algorithm to leverage the heuristic for combinational logic optimization [34].

### 2.4 Boyar-Peralta Two-Step Algorithm

It was briefly mentioned in the previous section that circuits constructed with low AND-count tend to have larger sections of purely linear components. It turns out that this property is intuitive for logic optimization through a two-step approach. The idea is simple: (a) identify non-linear sections of a circuit and perform AND-minimization on them to minimize the number of AND gates, then (b) perform XOR-minimization to optimize the remaining linear sections of the circuit. Specifically, the Boyar-Peralta two-
Chapter 2: LMC Heuristic for Logic Optimization

The step algorithm in [34] is designed as an iterative algorithm to facilitate the aforementioned procedure. This means that the algorithm is to be used repeatedly to optimize a circuit, with each iteration set to solve for better solution than the previous iterations. Once a satisfactory result is found or the algorithm fails to make further improvement to the previous results, the process terminates. The iterative nature of the algorithm is due to the non-deterministic or randomized search algorithm used in the AND-minimization step. In other words, the Boyar-Peralta algorithm will produce a different result for each execution. The overall flow of the two-step algorithm can be summarized in Figure 2.2.

2.4.1 Step 1: AND-Minimization

The AND-minimization step is also known as the non-linear step as it focuses on optimizing the non-linear portion(s) of a circuit. The objective of the AND-minimization step is to solve for an XOR-AND circuit that realizes the desired function using the minimal number of AND gates possible. It serves as the key step in the Boyar-Peralta algorithm that fulfills the premise of the LMC heuristic. In doing so, the cost of XOR gates is temporarily ignored in this step.

The Boyar-Peralta approach to AND-minimization is essentially a randomized search algorithm. Given an $n$-variable problem, an initial sample space is formed with the aforementioned $n$ variables. The algorithm then alternates between performing a number of XOR and AND operations using randomly selected pairs of elements from the sample space. Each new signal resulting from the operation is added to the sample space if the number of AND gates required thus far does not exceed the known multiplicative complexity of the target function. This process of applying XOR and AND operations on random pairs of elements is repeated until a resulting signal computes the desired function. To prevent excessive expansion to the sample space, a threshold can be set so that the algorithm restarts from initialization once the number of elements in the sample space exceeds said threshold. Figure 2.3 shows a summarized flow of the Boyar-Peralta AND-minimization step.

2.4.2 Step 2: XOR-Minimization

The second step is designed to optimize the purely linear portions of the circuit. Therefore, it is also referred to as the linear step. This step is tasked solely to maximize XOR-sharing within the linear circuits for further area reduction. It is a necessary procedure in the two-step algorithm because the randomized nature of AND-minimization step often results in inefficient linear connections.

---

This process of applying XOR and AND operations on random pairs of elements is repeated until a resulting signal computes the desired function. To prevent excessive expansion to the sample space, a threshold can be set so that the algorithm restarts from initialization once the number of elements in the sample space exceeds said threshold. Figure 2.3 shows a summarized flow of the Boyar-Peralta AND-minimization step.

---

3If the exact multiplicative complexity is unknown, the upper bound is assumed as per Lemma 2 in the first iteration. Subsequent iterations will tighten the upper bound whenever a solution with lower AND-count than the upper bound is discovered.
Figure 2.2: Boyar-Peralta two-step algorithm for LMC logic optimization.

Unlike the AND-minimization step, the algorithm used for XOR-minimization is deterministic and does not involve randomized selection of elements. From the resultant
Identify a non-linear function to optimize

Form a sample space $S$ with the input signals

Randomly select one or more pair(s) of signals from $S$ and XOR each pair

Include eligible results into the sample space $S$

Randomly select one or more pair(s) of signals from $S$ and AND each pair

Include eligible results into the sample space $S$

Repeat until target function is found

Repeat for alternative solution(s)

Return best solution

Figure 2.3: Boyar-Peralta AND-minimization step using an iterative randomized selection process.

circuit derived by the AND-minimization step, purely linear portions of the overall circuit are identified for optimization. For the purpose of illustration, the inputs of a linear circuit are referred to as $w_1, w_2, ..., w_n$ and the outputs from the same circuit as $z_1, z_2, ..., z_m$. Equations in (2.4.1) gives the expressions for a linear circuit with $n = 5$ inputs and $m = 6$ outputs for optimization with reference to [34].
The expressions from (2.4.1) can also be represented in the form of a matrix $M$ shown in (2.4.2), with each row of the matrix representing an expression from (2.4.1). For instance, the first row of matrix $M$ corresponds to the expression $z_1$. The first three columns of the row are marked as ‘1’ ($w_1, w_2, w_3$ are present in $z_1$) while the fourth and fifth columns are marked as ‘0’ ($w_4, w_5$ are absent in $z_1$). The same applies to subsequent rows on matrix $M$ with correspondence to expressions $z_2, z_3, \ldots, z_6$ in that order.

$$
M = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

Once the expressions for the linear circuit are established, a set $S$ is formed with all input signals $w_1, w_2, \ldots, w_n$. For the example above, the initial set would be $\{w_1, w_2, \ldots, w_5\} \in S$. The distance vector $D$ is also computed for matrix $M$. By definition, the distance vector is a measure of the number of XORs required to compute each expression in matrix $M$ using signals available in set $S$. This is calculated for each individual expression. Using the same example above, the distance vector for matrix $M$ would be $D = [2 \ 2 \ 3 \ 2 \ 2 \ 3]$ which implies expressions $z_1$ and $z_2$ each requires two XOR gates, $z_3$ requires three XOR gates and so on. The term magnitude is used to refer to the scalar sum of all elements in the distance vector.

The core function of the XOR-minimization algorithm is to identify a pair of signals $s_i$ and $s_j$ in set $S$ such that adding the signal resulting from the XOR between $s_i$ and $s_j$ into set $S$ would reduce the magnitude of the distance vector $D$ by the maximum amount. This process can be computationally intensive as the algorithm needs to check each combination of signals in set $S$ to identify the best pairing. In addition, as set $S$ grows
larger with each iteration, so does the complexity of the algorithm. Boyar et al. showed that the XOR-minimization process is an NP-hard problem in \cite{29}. Regardless, the main advantage of this approach is that it allows for XOR cancellation to be considered which results in lower gate count compared to greedy algorithms \cite{86}.

This process is repeated until the magnitude of the distance vector $D$ is reduced to zero. Since each iteration involves performing an XOR transformation on the optimal pair of signals, the hardware cost is thus one XOR gate per iteration. As such, the fact that the algorithm aims for maximum reduction on the magnitude of distance vector $D$ in each iteration means that it maximizes the "value" of each XOR gate. In other words, the XOR-minimization step gives the shortest path (hence minimal number of XOR gates) to implement the linear circuit. The Boyar-Peralta XOR-minimization step is summarized in Figure 2.4.

2.5 AND-Minimization in Multiple-Output Problems

A multiple-output problem refers to the instance where a circuit computes more than one outputs that are functions of the same set of inputs. Discussion on this topic is relevant for practical applications as most circuit optimization problems in real life are multiple-output problems. While this scenario does not impact the XOR-minimization step (the algorithm is inherently designed to solve multiple expressions at once), the AND-minimization step has some unique properties for this purpose that are not discussed previously.

The AND-minimization step as portrayed in Figure 2.3 shows the flow of the algorithm to solve for a single target function. While the algorithm can be applied to solve for each function independently in a multiple-output problem, the results obtained from this approach is hardly optimal due to the lack of circuit sharing between functions. Consequently, it only achieves optimal multiplicative complexity for each individual function at best but not the multiple-output problem as a whole.

Consider a set of functions $\{f_1, f_2, ..., f_n\} \in F$, it is natural to expect the multiplicative complexity of the problem to be at most the sum of the multiplicative complexity of each individual function, i.e. $c_\Lambda(F) \leq \sum_{i=1}^{n} c_\Lambda(f_i)$. This upper bound is easily achievable by solving each function individually using the algorithm outlined in Figure 2.3. However, it is possible to achieve a smaller number of AND gates than the upper bound due to product sharing between functions, thus enabling the potential for further area reduction in accordance with the LMC heuristic.

Product sharing is achieved when the output signal from an AND gate (a product) is shared between two or more functions in a multiple-output problem. The benefit of
Identify expressions for the linear circuit to be optimized

Form a set \( S \) with the input signals

Determine the distance vector \( D \) for the expressions

Determine the pair of signals in \( S \) to be XORed for a maximum reduction in the magnitude of \( D \)

Include the XORed result into \( S \) and update the values of \( D \)

Repeat until the magnitude of \( D \) equals zero

Return solution

Figure 2.4: Boyar-Peralta XOR-minimization step using the SLP approach.

Product sharing is obvious: for each product shared between two functions, the number of AND gates required is reduced by one from the upper bound. A simple way to attempt product sharing is through the use of a matching algorithm. Since there usually exist more than one minimal-AND solutions for each function, the single-target AND-minimization algorithm can be executed multiple times on each function to obtain a set of optimal solutions per function. A matching algorithm is then applied post optimization to analyze the products required in each discovered solution and returns the best solution set with the largest overlap between the products required between functions.

Regardless, this approach is not ideal in practice. Attempting product sharing post logic optimization means that the products themselves are not involved in the actual optimization process. Without knowledge of the available products, the AND-minimization
algorithm is unable to maximize their usage in the construction of the circuit. Hence, the resulting multiple-output circuits are rarely optimal in terms of multiplicative complexity. The Boyar-Peralta solution to product sharing between functions in multiple-output problems is to allow the AND-minimization algorithm to “consider” the use of signals required in previous function(s) when solving for the subsequent function(s). The approach is described as follows:

2. Store intermediate signals required for the previous output for possible use in AND-minimization for the next output.
3. Iterate until all outputs are successfully computed.

By accumulating the intermediate signals computed when solving each individual function, the algorithm can include them in the initial sample space when solving for the subsequent functions. This has significant benefits in a multiple-output problem as the accumulated signals are essentially free to use in subsequent functions, since they are already required as parts of the previously solved functions. More importantly, these signals include all the previously established products. This means that the AND-minimization algorithm has the potential to use these products in the randomized selection process when building the circuit for subsequent functions. This fulfills the goal of product sharing in the AND-minimization algorithm in a much more elegant fashion as the products are directly involved in optimization process rather than being addressed as an afterthought post optimization. It also helps that the process of “carrying over” used signals is much less computationally taxing compared to the matching algorithm approach.

## 2.6 Summary

The LMC heuristic provides a novel perspective to combinational logic design. Its main contribution to logic optimization is the ability to achieve further gate reduction for non-linear substitution circuits as demonstrated in [33][78]. The associated logic optimization algorithm suggests a two-step approach to minimize AND-count which includes: (a) an iterative randomized selection algorithm in the AND-minimization step and (b) a SLP algorithm in the XOR-minimization step.

Compared to modern logic optimization algorithm, the Boyar-Peralta two-step algorithm is particularly unsuitable to be applied on functions with large number of inputs and/or outputs. This is mainly due to the exponential growth in sample size that such a problem would induce on the AND-minimization step and the NP-hard nature of the
Chapter 2: LMC Heuristic for Logic Optimization

Identify a sequence of non-linear functions to optimize

Form a sample space $S$ with the input signals

Select a target function to be optimized

Randomly select one or more pair(s) of signals from $S$ and AND/XOR each pair

Include eligible results into the sample space $S$

Repeat until target function is found

Retain intermediate signals in $S$ and select the next target function (if available)

Return solution

Figure 2.5: Boyar-Peralta AND-minimization step for multiple-output non-linear problems. The randomized selection process with AND and XOR rounds are illustrated as a single step for brevity.

XOR-minimization step. Conversely, the LMC logic optimization approach is useful when isolating a smaller portion of a complex circuit for optimization. Modern computations often rely on complex circuits that are made up of many smaller components responsible for different functions. There are also a variety of functions that naturally decompose into repeated use of smaller components [29]. Examples include: (a) Arithmetic functions that can be built using multiple full adders, (b) matrix multiplications that can be broken up into smaller submatrices and (c) cryptographic functions which rely on multiple iterations of linear and non-linear transformations. These properties provide opportunities for the LMC heuristic to be applied in a variety of practical prob-
problems despite its limitations. In fact, it is encouraged to apply the LMC heuristic in combination with readily available logic synthesis systems such as SIS [52] in which the Espresso heuristic can first be used to generate an efficient circuit for the target function followed up by the application of the LMC heuristic on smaller partitions of the circuit for further area reduction.

When reviewing the Boyar-Peralta algorithm, it is important to highlight the flexibility of the algorithm (mainly in the AND-minimization step). While reliance on randomness in the selection process has significant ramifications on the consistency of the algorithm (more on this in the next chapter), it allows the freedom to append additional selection criteria to artificially reject specific signals so as to indirectly constrict the possible solution set. For instance, signals with excessive circuit depth can be rejected in favor of low depth implementations when propagation delay is a concern [79,87]. However, it is often necessary to loosen the AND-count restriction in these scenarios, therefore trading optimality in multiplicative complexity for speed. In this sense, the Boyar-Peralta algorithm is capable of balancing the trade-off in area and speed according to the design constraints imposed by the application environment.

Last but not least, the use of LMC heuristic is semi-reliant on the knowledge of the multiplicative complexity of the target function. This is helpful in allowing the AND-minimization step to accurately discard signals exceeding the known multiplicative complexity. Regardless, it is possible to leverage the Boyar-Peralta algorithm without said knowledge. In this case, the initial iteration of the Boyar-Peralta algorithm can begin with an assumption on the multiplicative complexity for each function using the upper bound rule as given by Lemma 2. This ensures that the algorithm is always able to formulate solutions for each function while at the same time allowing solutions of lower AND-count than the upper bound to be discovered if available. In such instance, the signal discard criterion in the AND-minimization step will be updated to reflect the proven lower AND-count in subsequent iterations for better results. However, this does imply that later iterations of the Boyar-Peralta algorithm will generally produce solutions with better gate counts as they converge toward the optimal multiplicative complexity.

Ultimately, as the first algorithm to make use of the LMC heuristic in logic optimization, the Boyar-Peralta algorithm is functionally sufficient for the role. Nevertheless, given its current description as illustrated in [34], there is room for improvement to be explored for the algorithm. In the next chapter, the undesirable aspects of the Boyar-Peralta algorithm are discussed to formulate strategies to eliminate or at least mitigate the issues.

---

4This property is relevant when performing statistical analysis on the result distribution of the Boyar-Peralta algorithm.
Enhanced Boyar-Peralta Algorithm for LMC Logic Optimization

3.1 Introduction

In the previous chapter, the LMC heuristic is introduced as a new perspective to low gate count logic optimization. The fundamental principle of the heuristic revolves around achieving area reduction through discovering implementations that require the least amount of AND gates possible for the target function. To this end, the Boyar-Peralta two-step algorithm proposed by the same authors is also presented in detail as a means to solve for low AND-count circuits. It serves as a solid reference on how to approach logic minimization problems using the heuristic.

The Boyar-Peralta algorithm boosts several desirable properties including its applicability on any arbitrary function and flexibility in execution. However, it also comes with significant drawbacks and room for improvement. In this chapter, notable flaws in the original two-step algorithm are discussed to understand the consequences on the performance of the algorithm. This is followed by novel propositions to address the outlined problems for an enhanced LMC logic optimization algorithm while retaining the key benefits of the original algorithm.

3.2 Problem Statement and Motivation

The key problem with the Boyar-Peralta algorithm is its inconsistency. In fact, most of the concerns regarding the algorithm originate from this issue. Naturally, the source of this problem can be directly attributed to the reliance on randomness in the AND-minimization step. The problem statements are as follows:

- Inability to ascertain the quality of solution. For most logic minimization problems,
there exist more than one solutions with optimal multiplicative complexity. Among
these solutions, some may be superior than others in terms of gate count. Ideally,
the algorithm would derive all optimal solutions and a post optimization selection
algorithm can easily fetch the best case from the available candidates. However,
as the Boyar-Peralta algorithm is random in nature, it is difficult to ensure all
the possible solutions are accounted for. This problem is further aggravated in
multiple-output problems where the quality of solutions for prior functions heavily
influences the possible solutions for subsequent functions due to the product sharing
feature.

- **Large number of iterations required.** In conjunction with the previous problem
  statement, a large solution set is needed to get a sense of the range of distribution
  for the quality of results. Without the ability to compare against the alternatives, a
  single execution of the algorithm provides little insight to the value of the solution.
  Consequently, the “true” computation time of the algorithm is significantly higher
  than just the time required for a single execution of the algorithm.

- **Non-convergent nature of the algorithm.** Given the need for multiple iterations
  of the algorithm, its non-convergent nature is a significant detriment. When op-
  timizing a function of known multiplicative complexity, the algorithm does not
  guarantee that each iteration would improve upon the quality of result from prior
  iterations. Therefore, it is difficult to determine the appropriate terminating con-
  ditions for the algorithm as the solutions do not converge towards the best case.
  The general strategy to obtain good solutions from the algorithm is just “the more
  iterations, the better”.

- **Potential worst-case scenario.** As yet another side effect of the randomized selec-
  tion process, it is inevitable that the algorithm may occasionally perform worse
  than exhaustive search in computation time. Although rarely, the algorithm can
  sometimes fail to discover a solution for the target function over an extended pe-
  riod of time. This scenario is more likely to occur as the complexity of the problem
  increases either in the number of input variables or output functions. Reinitial-
  ization of the sample space can act as a “safety mechanism” to some extent but
  there is no assurance that the same will not recur after. It is also an interesting
  research question to determine the optimal sample size for reinitialization so as to
  be beneficial to the performance of the algorithm.

It is difficult to eliminate all of the aforementioned problems without a major overhaul
to the AND-minimization step. Regardless, there are still many aspects of the algorithm

1 The term “optimal” in this context is used to refer to solutions that have minimal AND-count (as
given by the multiplicative complexity) rather than optimality in gate count or area.
that can be enhanced for a variety of benefits. The goal is to retain the positive qualities of the original two-step algorithm while proposing enhancements that contribute to the motivations as outlined below:

- Improve the average quality of results.
- Reduce variation in the quality of results.
- Shorten the average computation time of the algorithm.

3.3 Proposed Enhancements

In this section, a number of enhancements/modifications to the original Boyar-Peralta algorithm are proposed in line with the aforementioned motivations. Both the AND-minimization step and the XOR-minimization step are subjected to the enhancements proposed. The rationale behind each proposal is supported with theoretical proofs and/or examples when appropriate.

3.3.1 Reduction of Algorithm Overhead in AND-Minimization

The term algorithm overhead is defined as the extra computational time and memory space required by processes that are not necessarily the core of the algorithm but are nonetheless vital to ensure the proper function of the algorithm. For example, in the randomized selection process of the AND-minimization step, it is necessary for the algorithm to “remember” the source\(^2\) of each signal added to the sample space along with the operation that generated it (XOR or AND). This procedure, while not technically a main function in the AND-minimization step, is mandatory to reconstruct the optimized circuit once the target function is discovered. Processes such as these operate in the background and add a varying degree of computational time and memory requirement to the algorithm.

The first proposed enhancement is targeted at one such background process, aptly named the product tracking procedure. The term product is used to infer a signal that is an output from an AND gate. The purpose of product tracking is to record the number of AND gates required for every signal generated in the randomized selection process of the AND-minimization step. It is vital in determining whether a signal generated exceeds the predetermined AND-count limit and influences the decision whether to include the signal into the sample space. Given \(c_A(t)\) as the number of AND gates required to compute signal \(t\) and \(\{s_1, s_2\}\) as the sources of \(t\), the general consensus for the value of

\(^2\)If a signal \(t\) is a function of \(s_1\) and \(s_2\), then \(s_1\) and \(s_2\) are said to be the sources of \(t\).
Chapter 3: Enhanced Boyar-Peralta Algorithm for LMC Logic Optimization

Let \( c_A(t) \) if the operation is XOR would be as follows:

\[
c_A(t) = c_A(s_1) + c_A(s_2) \tag{3.3.1}
\]

Whereas if the operation is AND:

\[
c_A(t) = c_A(s_1) + c_A(s_2) + 1 \tag{3.3.2}
\]

Tracking the number of AND gates in the above manner is not computationally intensive. However, the formulas (3.3.1) and (3.3.2) are true only if the products required for \( s_1 \) and \( s_2 \) are mutually exclusive. Consider an example where \( s_1 = x_1 x_2 \) and \( s_2 = x_3 x_4 \). Given \( t = s_1 \oplus s_2 \), then it is true that \( c_A(t) = c_A(s_1) + c_A(s_2) = 2 \) since the products in \( s_1 \) and \( s_2 \) are mutually exclusive. However, this is not always the case.

Consider a second example where \( s_1 = x_1 (x_2 x_3 + x_4) \) and \( s_2 = x_4 (x_2 x_3 + x_1) \). If an XOR operation is performed on the two signals to generate \( t \), the same calculation would give \( c_A(t) = c_A(s_1) + c_A(s_2) = 4 \) which is not true. This is due to the existence of the product \( x_2 x_3 \) which is shared by both \( s_1 \) and \( s_2 \). A circuit computing \( t = s_1 \oplus s_2 \) would only require \( c_A(t) = 3 \) AND gates. Thus, if formulas (3.3.1) and (3.3.2) are used for product tracking, the algorithm may end up with an inaccurate interpretation on the AND-count of the signals generated. The false rejection of such signals may render the algorithm incapable of discovering specific solutions that are indeed optimal in terms of multiplicative complexity. As a result, the actual approach to product tracking demands a more computationally intensive solution.

It is paramount to store information on not just the number of AND gates required but also the actual products for each signal in the sample space. To begin, the input signals \( x_1, x_2, \ldots, x_n \) are not associated with any product, i.e. they are tagged with an empty set \( \emptyset \). For each signal generated through the XOR operation, it will carry over the product sets from the selected pair of signals. An extra step is then taken to remove duplicate products from the new set. The number of products in this new set will now accurately reflect the number of AND gates required to compute the new signal. As for AND operation, the same procedure is applied with the exception that the new signal itself is also added to the product set (since it is an output from an AND gate). To give an example, a signal \( t = x_4 + x_1 (x_2 + x_3 (x_1 + x_4)) \) would have a product set consisting of \( p_1 = x_3 (x_1 + x_4) \) and \( p_2 = x_1 (x_2 + x_3 (x_1 + x_4)) \).

Up to this point, the purpose of the discussion is to highlight the complexity involved in the product tracking procedure required to accurately represent the multiplicative complexity of each generated signal. In addition to computation time, the process affects the space complexity of the algorithm as it demands memory resources to keep track of.
the growing number of product sets. In response, a proposition is made to enable the algorithm to forgo product tracking entirely.

**Proposition 2.** Let \( m \) be the number of signals generated from AND operations per iteration of the AND-minimization step and \( i \) be the number iterations before the AND-minimization step restarts from initialization. If \( m = 1 \), all signals generated will not require number of multiplications exceeding \( i \).

**Proof.** For an AND-minimization step that restarts from initialization after \( i \) number of iterations, the maximum number of AND operations performed in a full run is given by \( m \times i \). Given that \( m = 1 \), it follows that the maximum number of products generated is \( i \). Assuming the worst-case scenario where all products are essential components of a signal, the signal will only require at most \( i \) number of AND operations.

Proposition 3 can then be easily derived from Proposition 2.

**Proposition 3.** If \( m = 1 \) and \( i = c_A(f) \), all signals generated will not require number of AND gates exceeding \( c_A(f) \).

The implication that can be drawn from Proposition 3 is that by setting \( m = 1 \) and \( i = c_A(f) \), it is possible to prevent the AND-minimization step from generating any signal that exceeds the intended AND-count limit. Thus, this allows the algorithm to completely eliminate the need for product tracking and significantly reduce the computation time and space complexity. In addition, this proposition derives further benefit when applied in combination with the next.

### 3.3.2 Sample Size Limitation in AND-Minimization

In essence, the AND-minimization step can be viewed as a randomized search algorithm in an expanding sample space. More precisely, the probability of discovering a *good* signal per iteration can be interpreted as conditional probability for dependent events. Hence, with reference to basic probability theory [88], the chances of discovering the desired outcome is inversely proportional to the size of the sample space. This provides the incentive to limit the size of the sample space for a more refined search algorithm.

**Proposition 4.** Let \( E \) be the desired event and \( e_1, e_2, \ldots e_n \) be the conditional events that had to be satisfied in sequence to achieve \( E \). If the size of the sample space \( S \) expands after each experiment, then for each time an event \( e_i \) fails, the probability of \( e_i, e_{i+1}, \ldots, e_n, E \) decrease.

**Proof.** Let \( n(e_n) \) be the number of ways an event \( e_n \) can occur in a sample space \( S \). By
the definition of probability,

\[ P(e_n) = \frac{n(e_n)}{\text{size}(S)} \]

Given events \( e_1 \) and \( e_2 \) are dependent,

\[ P(e_2 \mid e_1) = \frac{P(e_1 \cap e_2)}{P(e_1)} \]

Since \( P(e_1 \cap e_2) \) is a two-staged event,

\[ P(e_2 \mid e_1) = \frac{\frac{n(e_1 \cap e_2)}{\text{size}(S)}}{\frac{n(e_1)}{\text{size}(S)}} = \frac{n(e_1 \cap e_2)}{n(e_1) \times \text{size}(S)} \] (3.3.3)

Given the premise that \( \text{size}(S) \) increases after each experiment, then each time the event \( e_1 \) fails, \( \text{size}(S) \) increases while \( n(e_1 \cap e_2) \) and \( n(e_1) \) remain constant. By Equation 3.3.3 it follows that \( P(e_2 \mid e_1) \) decreases. The same can be proven for any subsequent events that are dependent on \( e_2 \).

Based on Proposition 4 each time the AND-minimization step generates a signal that is not relevant to the construction of the optimal circuit, it becomes less likely for the algorithm to be able to produce a signal that is relevant subsequently. In the original AND-minimization algorithm, it is suggested that the user sets a sample size limit so that the algorithm would abort the current sample space and reinitialize once the limit is exceeded. However, the suggestion is vague and there is room for improvement in regards to a clearer rule for sample size limitation.

There are three parameters that directly influence the sample size in the AND-minimization step: (a) the number of XOR operations per iteration \( a \), (b) the number of AND operations per iteration \( m \) and (c) the total number of iterations before reinitialization \( i \).

Coincidentally, previous proposition in Section 3.3.1 has decided the values of \( m \) and \( i \) in the attempt to reduce background tasks required to facilitate the algorithm. This leaves the remaining parameter \( a \) for adjustment. The goal is to determine the value of \( a \) which results in minimal expansion to the sample size without affecting the ability to discover all potential optimal solutions.

**Proposition 5.** Let \( n \) be the number of input variables. If \( m = 1 \) and \( i = c_\wedge(f) \), the AND-minimization step will not be able to solve for all possible optimal solutions (in terms of multiplicative complexity) for a problem with \( c_\wedge(f) = 1 \) if \( a < n - 1 \).

**Proof.** To prove Proposition 5 it is only necessary to provide an example where the
Chapter 3: Enhanced Boyar-Peralta Algorithm for LMC Logic Optimization

AND-minimization step is unable to produce a possible optimal solution for a \( c_A(f) = 1 \) problem, given \( m = 1, i = c_A(f) \) and \( a < n - 1 \). Consider a \( c_A(f) = 1 \) problem with \( n = 4 \) represented with (3.3.4).

\[
f = x_1 \oplus x_1 x_2 \oplus x_1 x_3 \oplus x_1 x_4
\]  
(3.3.4)

It is easy to identify one of the possible optimal solution with minimal AND-count to be (3.3.5).

\[
f = x_1(x_1 \oplus x_2 \oplus x_3 \oplus x_4)
\]  
(3.3.5)

In the AND-minimization step, XOR operations on the randomly selected pairs of signals are performed before the AND operations (see Figure 2.3). To obtain solution (3.3.5), it must be possible for the algorithm to produce the signal \( x_1 \oplus x_2 \oplus x_3 \oplus x_4 \) during the window for XOR operations. Given \( a < n - 1 \), the algorithm is able to perform a maximum of only two XOR operations per iteration. Since the desired signal requires three XOR operations, it follows that the AND-minimization step is unable to discover solution (3.3.5) under the given circumstances.

Following Proposition [5], it is ideal to perform \( a = n - 1 \) XOR operations per iteration. In conjunction with the previous proposal in Section 3.3.1, the new AND-minimization step is subjected to minimal sample space expansion per iteration. As a result, there is a higher probability for each randomized selection to generate a desired outcome.

Overall, there is a sharp distinction between the new AND-minimization step and the original in concept. Whereas the original approach favors signal selection in the same sample space for large number of iterations with few resets, the new AND-minimization step takes the opposite approach with minimal number of iterations but frequent reinitialization of the sample space. Given the rationale discussed previously regarding the advantages of a restricted sample size, the new approach is conjectured to outperform the original algorithm by virtue of improved probability.

3.3.3 Solving Sequence for Multiple-Output Problem

In Section 2.5, the product sharing feature adopted by the original algorithm to solve multiple-output problems is described in detail. However, the algorithm can benefit from a more guided approach to the solving sequence of the functions involved. Specifically, the proposal is to solve the functions in ascending order of their respective multiplicative complexity.

Firstly, it is relevant to reiterate that the main purpose of the multiple-output approach
is to maximize the potential of circuit sharing between the functions, especially the opportunity for product sharing which influences the total number of AND gates required in the final circuit. The sequence in which the functions are solved has interesting implications in regards to the aforementioned objectives.

Consider a simple two-output problem consisting of the functions $f_1$ and $f_2$ sharing some $n = 3$ inputs as described in (3.3.6).

$$
\begin{align*}
  f_1 &= x_1 x_2 \oplus x_1 \\
  f_2 &= x_1 x_2 x_3 \oplus x_1 x_3 \oplus x_1
\end{align*}
\quad (3.3.6)
$$

Using the lower bound rule and the upper bound rule of multiplicative complexity introduced through Lemmas 1 and 2 in Section 2.2.3, it is apparent that $c_\wedge(f_1) = 1$ and $c_\wedge(f_2) = 2$. Assuming the AND-minimization algorithm is applied to solve $f_2$ first, a potential optimal solution is $f_2 = x_3 (x_1 x_2 \oplus x_1) \oplus x_1$. In this case, among the intermediate signals carried over to solve $f_1$ are the two products $p_1 = x_1 x_2$ and $p_2 = x_3 (x_1 x_2 \oplus x_1)$. While it is true that product $p_1$ is valuable to the construction of the circuit for $f_1$, it is also important to highlight the fact that $p_2$ will never be useful for any function with multiplicative complexity of $c_\wedge(f) < 2$ in the context of LMC optimization. This establishes the first reasoning: Functions of higher multiplicative complexity require products of higher order which do not contribute to the construction of functions with lower multiplicative complexity. Worse still, said products dilute the sample space for the AND-minimization algorithm, making the optimization of subsequent functions more difficult.

Solving $f_2$ first may also yield another optimal solution $f_2 = x_1 (x_2 x_3 \oplus x_3) \oplus x_1$. In this case, the products $p_1 = x_2 x_3$ and $p_2 = x_1 (x_2 x_3 \oplus x_3)$ are both unrelated to the function $f_1$. This leads to the second reasoning: Functions of higher multiplicative complexity generally have more alternative solutions. This is attributed to properties of the distributive law that allow the factorization of XOR-AND Boolean functions as demonstrated in (3.3.7).

$$
\begin{align*}
  f &= x_1 x_2 x_3 \oplus x_2 x_3 x_4 \\
     &= x_2 (x_1 x_3 \oplus x_3 x_4) \\
     &= x_3 (x_1 x_2 \oplus x_2 x_4)
\end{align*}
\quad (3.3.7)
$$

A function with higher multiplicative complexity generally allows more room for manipulation using the distributive law compared to a function with lower multiplicative complexity, hence the increase in number of possible solutions. However, only a fraction of these solutions usually have good compatibility for product sharing with the other
functions. As low complexity functions have less number of possible optimal solutions in general, solving them first introduces less variation at the early stages of the algorithm. In fact, the signals carried over from lower complexity functions can help to reduce result variation in high complexity functions as they skew the randomized selection algorithm towards results with common circuitry, which is in line with the purpose of the endeavor.

As a side note, considering the fact that LMC functions are generally smaller in size, solving for them first provides the added benefit of keeping the sample size small for subsequent functions when carrying over the intermediate signals. As discussed previously, a small sample size is always desirable for a randomized selection algorithm for better odds at the desired outcomes.

3.3.4 Inclusion of Non-Linear Circuit in XOR-Minimization

The two-step algorithm for LMC optimization is designed to optimize non-linear and linear portions of a circuit separately through the AND-minimization step and XOR-minimization step respectively (see Figure 2.2). For example, Figure 3.1 shows a circuit with an upper linear component, a middle non-linear component, and a bottom linear component (similar to the depiction of the AES S-Box in [79]). The procedure to solve for an optimal solution for the circuit would be to apply the AND-minimization step on the non-linear component, followed by the XOR-minimization step on the remaining linear components.

The fourth proposal is the inclusion of non-linear components in the XOR-minimization step for further reduction in circuit size. In theory, this may seem contradictory due to incompatibility of the AND gates present in the non-linear circuit with the XOR-minimization step which is designed to operate only on XOR gates. However, the proposal can be made possible by treating each product in the non-linear circuit as a unique variable in the XOR-minimization step. The procedure is demonstrated in Example 1.

Example 1. Given a function \( f = x_1 + x_4 + x_1(x_2 + x_3(x_1 + x_4)) \) that is optimal in terms of multiplicative complexity with \( c_A(f) = 2 \), the function can be transformed into three
purely linear expressions as demonstrated in (3.3.8).

\[
f = x_1 + x_4 + x_1(x_2 + x_3(x_1 + x_3))
\]  (3.3.8)

\[
\begin{align*}
z_1 &= w_1 + w_4 \\
z_2 &= w_2 + w_5 \\
z_3 &= w_1 + w_4 + w_6
\end{align*}
\]  (3.3.9-3.3.11)

Where,

\[
\begin{align*}
w_1 &= x_1 \\
w_2 &= x_2 \\
w_3 &= x_3 \\
w_4 &= x_4 \\
w_5 &= x_3(x_1 + x_4) \\
w_6 &= x_1(x_3(x_1 + x_4) + x_2)
\end{align*}
\]

As shown in Example 1, it is possible translate the non-linear function \(f\) into three linear expressions \(z_1, z_2, z_3\). Signals generated from AND gates are treated as “new variables” alongside the original inputs as denoted by \(w_1, w_2, ..., w_6\). The XOR-minimization step can then be applied to optimize the linear expressions without further complications.

There is one major benefit to this proposition. As mentioned in Section 2.4.1, the AND-minimization step is designed to focus solely on achieving optimal multiplicative complexity while ignoring the cost XOR gates in its execution. As a result, the XOR circuitry in the non-linear component is often constructed inefficiently especially due to the reliance on randomness. By converting the resultant circuit to a linear optimization problem, the existing XOR-minimization algorithm can be leveraged for further reduction to the non-linear component.

### 3.3.5 Circuit Depth Criterion for XOR-Minimization

The XOR-minimization step offers an SLP algorithm to solve for linear circuit construction with minimal number of XOR gates. The algorithm described in Section 2.4.2 allows for XOR cancellation and has shown better results in comparison to algorithms which do not [86]. However, the algorithm is focused solely on the shortest path for minimal gate
count. This means that two solutions that are of the same gate count will be of equal “value” to the algorithm even if one circuit has advantages in different metrics compared to the other.

Figure 3.2: Two linear circuits (a) and (b) computing the same functions $z_1, z_2, ..., z_4$. 

Figure 3.2 shows two 4-input linear circuits computing the same functions $z_1, z_2, ..., z_4$ with the following relations:

\[
\begin{align*}
    z_1 &= w_1 \oplus w_2 \\
    z_2 &= w_1 \oplus w_2 \oplus w_3 \\
    z_3 &= w_1 \oplus w_2 \oplus w_3 \oplus w_4 \\
    z_4 &= w_3 \oplus w_4
\end{align*}
\]  

(3.3.12)

Both circuit (a) and (b) from Figure 3.2 are optimal in terms of gate count. However, it is easy to infer that circuit (a) has an advantage in terms of circuit depth compared to circuit (b). This is important as lower circuit depth reduces propagation delay for the circuit and no notable trade-off is incurred in this instance. However, the XOR-minimization step as described in Section 2.4.2 has no means to distinguish between circuit (a) and (b) as they both satisfy the SLP criterion for minimal gate count. For this purpose, an additional circuit depth criterion is proposed for the XOR-minimization step.

**Proposition 6.** Let $p_n$ be a pair of signals, $D_{p,n}$ be the distance vector for the signals in $p_n$ and $\max(D_{p,n})$ be the largest value in $D_{p,n}$. If $\text{XOR}(p_1) = \text{XOR}(p_2)$, then $\text{XOR}(p_1)$ will have a shallower circuit depth if $\max(D_{p,n}) < \max(D_{p,n})$.

**Proof.** Assuming distances and circuit depth are measured in number of logic gates, the output from an XOR gate resulting from input pair $p_n$ will have a circuit depth of $\max(D_{p,n}) + 1$ gates. It follows that if $\max(D_{p,n}) < \max(D_{p,n})$, $\text{XOR}(p_1)$ has a shallower circuit depth.

From Proposition 6, the pair of signals with the lowest $\max(D_{p,n})$ should be selected for minimal circuit depth. To facilitate this proposal, the proposed XOR-minimization step needs to track the distances of each signal added to set $S$. Fortunately, contrary to the complications associated with product tracking in the original AND-minimization
step (see Section 3.3.1), calculation of signal distance for the output is computationally trivial if the distances of the input pair are known. Hence, it does not impact the overall complexity of the XOR-minimization step. Regardless, there are two important caveats which concern the implementation of the circuit depth criterion.

Firstly, it is important to keep the original SLP criterion as the primary benchmark for signal selections in the XOR-minimization step since minimal gate count is the main goal of the algorithm. Therefore, the best approach is to implement the proposed circuit depth criterion as the tiebreaker criterion when multiple pairs of signals can reduce the distance vector by the same maximum amount. This ensures that the algorithm remains favorable towards low gate count solutions whenever possible. The circuit depth criterion will generally come into play in later iterations of the algorithm where the remaining signals are likely to have the same impact on the distance vector.

Secondly, it is suggested in [29] to resolve tiebreakers by choosing to XOR the pair of signals which will produce a new distance vector with the largest Euclidean norm\(^3\). For example, given two possible distance vectors of equal magnitude \(D_1 = [1 \ 1 \ 1 \ 1]\) and \(D_2 = [0 \ 0 \ 3 \ 1]\), the signal pair which produces \(D_2\) is preferred as \(\|D_2\|_2 > \|D_1\|_2\). The reasoning is that \(D_1\) would require four more gates to realize while \(D_2\) may require only three. In other words, this gives the SLP algorithm a higher chance to maximize distance vector reduction in subsequent iterations. Nevertheless, instances in which tiebreakers remain undecided post Euclidean norm evaluation are commonplace in practice. Ultimately, given the implication of the Euclidean norm criterion on gate count reduction, it is sensible for it to take precedence over the circuit depth criterion in the decision hierarchy.

3.3.6 Summary

Overall, the proposed algorithm incorporates a total of five enhancements to the Boyar-Peralta algorithm for LMC logic optimization. It retains the two-step framework of the original algorithm but with several differences to how the AND-minimization step and the XOR-minimization step are executed. To summarize the proposed algorithm: Algorithm [1] describes the enhanced AND-minimization step, Algorithm [2] describes the enhanced XOR-minimization step, and Algorithm [3] gives the proposed approach to multiple-output problems.

\(^{3}\)Euclidean norm may also be referred to as square root of the sum of squares.
Algorithm 1 Pseudocode for the enhanced AND-minimization step.

1: begin
2: Initialization
3: Define single target non-linear function \( f \)
4: Form sample space \( S \) with all \( n \) input variables \( x_1, x_2, ..., x_n \)
5: for \( i = 1 \) to \( i = c^*(f) \) do
6:     for \( a = 1 \) to \( a = n - 1 \) do
7:         Randomly select a pair of signals in \( S \)
8:         XOR selected pair of signals
9:         Include new signal into \( S \)
10:    end for
11: Randomly select one pair of signals in \( S \)
12: AND selected pair of signals
13: Include new signal into \( S \)
14: end for
15: if target function \( f \) discovered then
16:     return solution
17: else
18:     restart from line 4
19: end if
20: end

3.4 Evaluation of Proposed Algorithm

To observe the actual improvements provided by the proposed enhancements, comparisons are done between the original algorithm and the enhanced algorithm by applying both algorithms to solve identical problems. For this purpose, both algorithms are implemented using MATLAB R2012b on a system with the Intel Core i5-4690 processor @ 3.50GHz and 8GB of RAM.

Due to elements of randomness associated with both algorithms, a large number of trails are required for fair comparisons. In this case, 100 trials are performed per algorithm for each problem. Results are represented using box plot to avoid making assumptions on the underlying statistical distribution. In fact, majority of the parameters of interest have a discrete distribution rather than a continuous distribution and as such, median is a more meaningful measure than mean. The sample size limitation allocated for the original Boyar-Peralta AND-minimization step is set to 1000 signals. Further increase in the value appears to incur undesirable memory management issues within the computing system which noticeably increases CPU time and may bias the results.

Since the resultant circuits are constructed using more than one type of logic gates, circuit sizes will be measured in NAND gate equivalent (GE) to more accurately reflect the actual area. However, comparing the computation time of the algorithms can be challenging. Firstly, algorithm complexity analysis is not useful for both algorithms as the randomized nature leads to unbounded time complexity in the worst-case scenario.
Algorithm 2 Pseudocode for the enhanced XOR-minimization step.

1: begin
2: Initialization
3: Define all \( m \) target linear functions \( z_1, z_2, \ldots, z_m \)
4: Form matrix \( M \) representing the linear functions
5: \( \triangleright \) see Section 2.4.2 for derivation of matrix \( M \)
6: Form set \( S \) with all \( n \) input variables \( w_1, w_2, \ldots, w_n \)
7: Calculate distance vector \( D \) for matrix \( M \)
8: Set \( D_{\text{min}} = D \)
9: Set \( EN_{\text{min}} = \) Euclidean norm of \( D \)
10: while \( D_{\text{min}} \neq 0 \) do
11: \quad while NOT(all possible pairs in \( S \) are evaluated) do
12: \quad \quad Select a pair of signals \( p \) from \( S \) for XOR
13: \quad \quad Calculate new distance vector \( D_{\text{new}} \) and Euclidean norm \( EN_{\text{new}} \)
14: \quad \quad if \( D_{\text{new}} < D_{\text{min}} \) then \( \triangleright \) SLP criterion
15: \quad \quad \quad Mark \( p \) as best pair
16: \quad \quad \quad Set \( D_{\text{min}} = D_{\text{new}}, EN_{\text{min}} = EN_{\text{new}} \) and \( D_{p,\text{max}} = \max(D_p) \)
17: \quad \quad else if \( D_{\text{new}} = D_{\text{min}} \) and \( EN_{\text{new}} > EN_{\text{min}} \) then \( \triangleright \) Euclidean norm tiebreaker criterion
18: \quad \quad \quad Mark \( p \) as best pair
19: \quad \quad \quad Set \( D_{\text{min}} = D_{\text{new}}, EN_{\text{min}} = EN_{\text{new}} \) and \( D_{p,\text{max}} = \max(D_p) \)
20: \quad \quad else if \( EN_{\text{new}} = EN_{\text{min}} \) and \( \max(D_p) < D_{p,\text{max}} \) then \( \triangleright \) circuit depth tiebreaker criterion
21: \quad \quad \quad Mark \( p \) as best pair
22: \quad \quad \quad Set \( D_{\text{min}} = D_{\text{new}}, EN_{\text{min}} = EN_{\text{new}} \) and \( D_{p,\text{max}} = \max(D_p) \)
23: \quad \quad end if
24: \quad end while
25: \quad Set \( D = D_{\text{min}} \)
26: \quad Add XOR of best pair \( p \) into set \( S \)
27: end while
28: return solution
29: end


1: begin
2: Initialization
3: Define all \( m \) non-linear functions \( f_1, f_2, \ldots, f_m \) to be optimized
4: Form sample space \( S \) with all \( n \) input variables \( x_1, x_2, \ldots, x_n \)
5: Sort functions in ascending order of multiplicative complexity
6: while NOT(all functions are optimized) do
7: \quad Select the next target function \( f \)
8: \quad Run Algorithm 1 on \( f \) with \( S \)
9: \quad Keep intermediate signals in \( S \)
10: end while
11: return solution
12: end

38
At the same time, measuring computation time in real time is not ideal as it is subjected to external factors such as background tasks running in the PC. In this experiment, number of operations is used as a pseudo-representation of computation time, indicating the total number of XOR and AND operations that each algorithm has to perform until the target function is discovered (this value does not reset when the AND-minimization step is reinitialized).

Last but not least, inferential statistics is used to determine whether there is a significant difference between the means of the random samples. This is important to provide some degree of confidence that the differences observed between the samples have not occurred by chance. Specifically, independent-samples t-tests are conducted for each experiment using the typical alpha significance level of $p < 0.05$.

### 3.4.1 Single-Output Problem

The performance of each algorithm is first evaluated on single-output problems. Equation (3.4.1) gives the truth vectors for three random functions $f_1, f_2, f_3$ of $n = 4$ inputs.

$$f_1(x_1, x_2, x_3, x_4) = [0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0]^T$$

$$f_2(x_1, x_2, x_3, x_4) = [0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1]^T$$

$$f_3(x_1, x_2, x_3, x_4) = [1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1]^T \quad (3.4.1)$$

Both algorithms are applied to solve functions $f_1, f_2, f_3$ individually. The distributions of results for each function are illustrated in Figures 3.3, 3.4 and 3.5 respectively. A summary of important statistics for all results are tabulated in Table 3.1.

**Figure 3.3:** Distributions of optimized results for $f_1$: (a) Circuit size and (b) no. of operations.

Function $f_1$ has a multiplicative complexity of $c_\lambda(f_1) = 1$ and represents the least com-
plex minimization problem for the LMC heuristic. Reviewing the results in Figure 3.3, box plot (a) shows indistinguishable differences between the two algorithms in quality of results. This is attributed to the low complexity of function $f_1$ which generally has less possible solutions that are optimal in AND-count. Specifically, only two unique solutions are discovered for $f_1$ with one requiring an extra XOR gate (4 GE in area). Both algorithms are able to derive the better solution in majority of the trials and no clear advantage can be given to either approach in quality of results. However, despite the low complexity, a significant disparity can already be observed in the number of operations required. Said metric is recorded at a median of 28k operations for the original algorithm, more than nine-fold the value observed for the enhanced algorithm. This improvement is the result of minimal sample space expansion featured in the enhanced algorithm. It is observed that if the original algorithm fails to discover the target function in the early
iterations, subsequent experiments in the same sample space rarely yield the desired outcome until reinitialization. This agrees with the probability theory discussed in Section 3.3.2. Therefore, the observed reduction in number of operations makes sense for the enhanced algorithm due to the frequent reinitialization.

Function $f_2$ has a multiplicative complexity of $c_n(f_2) = 2$ and represents a step-up in complexity from the previous function. From Figure 3.4 (b), the enhanced algorithm retains the advantage in number of operations, having a median of 358k operations over the 563k of the original algorithm. More importantly, the higher complexity of function $f_2$ results in increased number of possible solutions (of varying qualities) which enables comparison between the performances of both algorithms in quality of results. Specifically, the results on circuit area as represented by Figure 3.4 (a) are interesting in many aspects. First of all, the enhanced algorithm showed increased instances of solutions that achieved the local minimum in circuit area: 60% for the enhanced algorithm against 36% for the original algorithm. Secondly, result variation is also reduced as evident by the shorter box plot and inter-quartile range for the enhanced algorithm. These improvements are important for consistency and can be attributed to a few factors. The proposal to translate non-linear circuits into linear forms for XOR-minimization undoubtedly played a role in area reduction. When examining the solutions derived from the original algorithm, it is noticed that the XOR interconnections in some instances are not efficient. Subjecting said circuitry to XOR-minimization generally yields a single XOR gate reduction which makes the results closer to that of the enhanced algorithm. Nevertheless, it is also surmised that much of the improvement is contributed by the strict sample size proposed for the AND-minimization step. Because of the sample size limitation, the enhanced algorithm is much less “tolerant” of large solutions by nature. This is because large solutions require more good signal pairings to be happened in a single execution which is less likely to occur when the number of pairings per iteration is minimal.

Function $f_3$ is another $c_n(f_3) = 2$ problem. From Figure 3.5, the enhanced algorithm once again demonstrated obvious advantages over the original algorithm in both circuit area and number of operations. A total of 66% of the solutions by the enhanced algorithm achieve the local minimum area, outperforming the original algorithm at 41%. Overall, the result distributions for $f_3$ are similar to the previous function $f_2$ and the same commentary applies to explain the improvements. Regardless, the results serve to reinforce the established observations on both algorithms.

The t-test results for each experiment can be referred from Table 3.1. In each instance, the two-tailed significance value $p$ is lower than the alpha significance level of 0.05. Therefore, it can be concluded with reasonable confidence that there is a significant statistical difference between the two samples.
Table 3.1: Summary of results and t-tests for single-output problems.

\[
f_1(x_1, x_2, x_3, x_4) = [0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0]\t
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(c_r(f_1))</th>
<th>Circuit size (GE)</th>
<th>No. of operations ((\times 10^4))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(\text{med}.)</td>
<td>(\text{min}.)</td>
</tr>
<tr>
<td>Original</td>
<td>1</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Proposed</td>
<td></td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

\[
f_2(x_1, x_2, x_3, x_4) = [0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1]\t
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(c_r(f_2))</th>
<th>Circuit size (GE)</th>
<th>No. of operations ((\times 10^4))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(\text{med}.)</td>
<td>(\text{min}.)</td>
</tr>
<tr>
<td>Original</td>
<td>2</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>Proposed</td>
<td></td>
<td><strong>20</strong></td>
<td><strong>20</strong></td>
</tr>
</tbody>
</table>

\[
f_3(x_1, x_2, x_3, x_4) = [1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0]\t
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(c_r(f_3))</th>
<th>Circuit size (GE)</th>
<th>No. of operations ((\times 10^4))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(\text{med}.)</td>
<td>(\text{min}.)</td>
</tr>
<tr>
<td>Original</td>
<td>2</td>
<td>21</td>
<td>17</td>
</tr>
<tr>
<td>Proposed</td>
<td></td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>
3.4.2 Multiple-Output Problem

While the results on single-output problems give significant advantages to the enhanced algorithm, experiments on multiple-output problems are more relevant to practical applications. For this purpose, the three functions $f_1, f_2, f_3$ from (3.4.1) are collectively treated as a three-output problem $f_1, f_2, f_3 \in F$. Both algorithms are applied to solve the multiple-output problem $F$ and the results are illustrated in Figure 3.6.

![Comparison of Circuit Size](image1)
![Comparison of No. of Operations](image2)

Figure 3.6: Distributions of optimized results for $F$: (a) Circuit size and (b) no. of operations.

Observing the distributions in Figure 3.6, it is evident that the improvements in single-output problems translate well to the multiple-output problem. Circuit area showed reduction from a median of 51 GE to 39 GE while the number of operations is reduced from a median of 716k to 102k. Although the same reasoning can be drawn as in the case of single-output problems, further examination reveals other interesting factors which are unique to multiple-output problems. Unlike single-output problems where the lower bound rule (Lemma 1) and upper bound rule (Lemma 2) allow a generous estimate of the multiplicative complexity of a function, it is difficult to do the same for a multiple-output problem due to difficulties in determining the optimal product sharing between the functions. When solving a function in a multiple-output problem, the AND-count limitation is applied per function basis based on the respective multiplicative complexity during the AND-minimization step. However, because the AND-minimization step “encourages” product sharing with previously solved functions (see Section 2.5), it is possible to arrive at a solution requiring less AND gates than the function’s multiplicative complexity. The possibility of these solutions is dependent on the product set carried over from solved functions which varies significantly every execution. For this reason the AND-count limitation cannot be reduced below the multiplicative complexity lest the algorithm be permanently stuck with no available solutions due to incompatible prod-
uct set. Therefore, unlike single-output problems, the Boyar-Peralta algorithm does not guarantee optimal AND-count for multiple-output problems even when the multiplicative complexity of a problem is known. To be precise, the AND-count of the solutions always fall between a lower bound given by the actual multiplicative complexity of the problem $c_\Lambda(F)$ and an upper bound given by the sum of multiplicative complexity of its individual functions $\sum^n_{i=1} c_\Lambda(f_i)$. Since the AND-count of the solutions is closely related to the circuit area as suggested by the LMC heuristic, it is interesting to observe the performance of each algorithm in this regard to gain further insight into the differences in results.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{comparison.png}
\caption{Distribution of number of AND gates required in optimized results for $F$.}
\end{figure}

With reference to the distributions in Figure 3.7, it can be verified that the number of AND gates does range between the aforementioned lower and upper boundaries. More importantly, notable advantage can be observed for the enhanced algorithm in deriving solutions of low AND-count for the multiple-output problem. This disparity in AND-count is the main cause for the pronounced difference in circuit area as depicted in Figure 3.6 (a). It is conjectured that the lower AND-count can be attributed to the strict sample size limitation proposed for the AND-minimization step as the other propositions have no discernible effect on the AND-count of the solutions (except for the solving sequence which is a common factor in this experiment). A curbed sample space gives higher probability of selecting the free products carried over from previous functions by virtue of a low sample size. Although these products are not always useful for the target function, maximizing the attempts to utilize them is critical in LMC optimization especially given the inability to preemptively determine the applicability of said products. On the contrary, the leniency to mix and match more combinations of signals in the original algorithm makes it easier to arrive at alternative solutions (of higher AND-count) when the good products in the sample space are missed.

To examine the impact of the proposed solving sequence for multiple-output problems,
the enhanced algorithm is applied to solve $F$ in both ascending order ($f_1, f_2, f_3$) and descending order ($f_3, f_2, f_1$) of multiplicative complexity. The distributions of results are portrayed in Figure 3.8.

Figure 3.8: Distributions of results by proposed algorithm with opposing solving sequence: (a) Circuit size, (b) no. of operations and (c) no. of AND gates.

Solving sequence in a multiple-output problem has severe implications on the results as shown in Figure 3.8. In fact, the difference in results is comparable to that between the original algorithm and the enhanced algorithm in Figures 3.6 and 3.7. There is a 6 GE disparity between the medians of both samples and a significant $14.27 \times 10^5$ difference in number of operations. When describing the motivation behind the proposed solving sequence in Section 3.3.3, the incompatibility of high order products in the construction of functions with lower multiplicative complexity is highlighted. The results on the number of AND gates agree with the sentiment as the descending solving sequence appears to hinder the ability of the enhanced algorithm to produce solutions of optimal multiplicative complexity. Consequently, the same implication is reflected on circuit size as per the Boyar-Peralta heuristic. It is interesting to note that the disparity in number
of operations between the solving sequences is greater than that between the original and enhanced algorithm. The descending solving sequence populates the sample space with high order products which severely lower the odds at good signal pairings. This problem is further compounded by the increasing number of output functions which adds to the number of products. Overall, solving a multiple-output problem in ascending order of the multiplicative complexity of its functions clearly demonstrated a significant advantage over the opposite approach.

The t-test statistics are presented in Table 3.3. Once again, given the low significance level \( p \) in each experiment, the statistical differences between samples are deemed to be significant.

### Table 3.2: Summary of results for multiple-output problem \( F \).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Circuit size (GE)</th>
<th>No. of operations ((\times 10^5))</th>
<th>No. of AND gates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>med.</td>
<td>min.</td>
<td>sd.</td>
</tr>
<tr>
<td>Original</td>
<td>51</td>
<td>35</td>
<td>7.13</td>
</tr>
<tr>
<td>Proposed</td>
<td>39</td>
<td>35</td>
<td>3.33</td>
</tr>
<tr>
<td>Proposed*</td>
<td>45</td>
<td>35</td>
<td>5.36</td>
</tr>
</tbody>
</table>

* Proposed algorithm with descending solving sequence.

### Table 3.3: t-tests for multiple-output problem \( F \).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Circuit size (GE)</th>
<th>No. of operations ((\times 10^5))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>( t(99) )</td>
</tr>
<tr>
<td>Original</td>
<td>50.62</td>
<td>14.54</td>
</tr>
<tr>
<td>Proposed</td>
<td><strong>39.18</strong></td>
<td>1.47</td>
</tr>
</tbody>
</table>

* Proposed algorithm with descending solving sequence.

### 3.4.3 Circuit Depth

A separate subsection is dedicated to evaluate the circuit depth criterion for the proposed XOR-minimization. The large sample sizes in the previous experiments make it difficult to observe the actual impact of the enhancement. In fact, any observable differences in circuit depth may just be attributed to the different solutions instead of actual improvements to the circuits. Conveniently, XOR-minimization is not a randomized process and can be easily evaluated by subjecting both algorithms to applicable problems. Matrices \( M_1, M_2, M_3 \) represents three artificially selected linear optimization problems for the XOR-minimization step where solutions of optimal gate count exist with varying circuit depth.
depth. Note that matrix $M_2$ represents the same circuit portrayed in Figure 3.2. The optimized results are tabulated in Table 3.4.

$$M_1 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad M_2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad M_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$ (3.4.2)

Table 3.4: Summary of results for linear optimization problems $M_1, M_2$ and $M_3$.

<table>
<thead>
<tr>
<th>Problem</th>
<th>XOR-count</th>
<th>Circuit depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$M_2$</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$M_3$</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

As reported in Table 3.4, the enhanced XOR-minimization step is able to identify the lower depth solution in all three problems while the original algorithm only managed to do so for $M_1$. Due to the absence of the circuit depth criterion, the original algorithm returns the first solution with optimal gate count that it finds as subsequent solutions are considered to be of equal value. In contrast, the enhanced XOR-minimization step compares the depth of solutions when an equal gate count is observed to always return the better solution. Since the gate count of the circuits remains the primary criterion, there is no trade-off in circuit area associated with this approach.

3.4.4 Discussion

Based on the results for both single and multiple-output problems, the enhanced two-step algorithm for LMC logic optimization demonstrated significant improvements over the original algorithm in three main areas: (a) average quality of results, (b) computation time and (c) overall consistency. As such, the enhanced algorithm can more reliably produce optimal results for logic optimization based on the Boyar-Peralta heuristic.

While acknowledging the improvements over the original algorithm, it is important to note that the proposed algorithm does not produce results that are strictly better than the original algorithm. In all experiments, the best results produced by both algorithms are of equal quality (as evident by the min. metric in Tables 3.1 and 3.2). Thus, it is emphasized that the contributions of the proposed enhancements are in increasing the odds of outcomes that are or close to the best case scenario as indicated by the improved median and shorter range. At the same time, the amount of variation or dispersion in the
results is reduced overall as measured by the standard deviation. As long as a randomized selection procedure remains necessary in the AND-minimization step, improvements to the median and standard deviation of the result distributions are indispensable to a more reliable algorithm.

Improvements in number of operations (hence computation time) are generally less valued than the quality of results in most use cases. Regardless, given the exponential increase in complexity for logic minimization problems following an increase in number of inputs, the considerable shortening of computation time by the enhanced algorithm goes a long way toward improving the “ease of use” of the heuristic.

3.5 MCNC Benchmarks

The Microelectronics Center of North Carolina (MCNC) International Workshop on Logic Synthesis [89] provides an extensive set of combinational multilevel benchmark circuits. Given that existing works report only on the performance of the LMC heuristic on cryptographic circuits, 11 MCNC benchmark circuits are selected for further evaluation on a wider scope of functions. Each selected circuit is subjected to optimization using the enhanced Boyar-Peralta algorithm and the best solution found is recorded. The results are tabulated in Table 3.5.

On smaller circuits with low number of inputs and outputs such as b1 and cm82a, the enhanced Boyar-Peralta algorithm can be applied directly for optimization. On the other hand, it is generally impractical to apply the algorithm on functions with over five inputs due to the complexity of the algorithm. Regardless, these circuits tend to feature intermediate signals that are functions of a much smaller number of variables. By subjecting said intermediate signals for logic optimization, it is possible to achieve gate count reduction for large circuits using the enhanced Boyar-Peralta algorithm.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Input(s)</th>
<th>Output(s)</th>
<th>Gate count</th>
<th>MCNC [89]</th>
<th>This work</th>
<th>Δ%</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>3</td>
<td>4</td>
<td>13</td>
<td>5</td>
<td>61.54</td>
<td></td>
</tr>
<tr>
<td>cm42a</td>
<td>4</td>
<td>10</td>
<td>17</td>
<td>23</td>
<td>-35.29</td>
<td></td>
</tr>
<tr>
<td>cm82a</td>
<td>5</td>
<td>3</td>
<td>27</td>
<td>10</td>
<td>62.96</td>
<td></td>
</tr>
<tr>
<td>cm85a</td>
<td>11</td>
<td>3</td>
<td>38</td>
<td>50</td>
<td>-31.58</td>
<td></td>
</tr>
<tr>
<td>cm150a</td>
<td>21</td>
<td>1</td>
<td>69</td>
<td>47</td>
<td>31.88</td>
<td></td>
</tr>
<tr>
<td>cm151a</td>
<td>12</td>
<td>2</td>
<td>33</td>
<td>23</td>
<td>30.30</td>
<td></td>
</tr>
<tr>
<td>cm162a</td>
<td>14</td>
<td>5</td>
<td>43</td>
<td>39</td>
<td>9.30</td>
<td></td>
</tr>
<tr>
<td>cm163a</td>
<td>16</td>
<td>5</td>
<td>42</td>
<td>37</td>
<td>11.90</td>
<td></td>
</tr>
<tr>
<td>parity</td>
<td>16</td>
<td>1</td>
<td>68</td>
<td>15</td>
<td>77.94</td>
<td></td>
</tr>
<tr>
<td>pc1</td>
<td>19</td>
<td>9</td>
<td>68</td>
<td>56</td>
<td>17.65</td>
<td></td>
</tr>
<tr>
<td>tcon</td>
<td>17</td>
<td>16</td>
<td>41</td>
<td>24</td>
<td>41.46</td>
<td></td>
</tr>
</tbody>
</table>
Overall, the enhanced Boyar-Peralta algorithm showed impressive results with few exceptions. Over 60% reduction in gate count was observed for small circuits such as b1 and cm82a where the full potential of the algorithm can be leveraged. Otherwise, the algorithm still managed notable savings for large circuits with percentage reduction in gate count ranging from approximately 10% to 40%. The parity circuit achieved an isolated case of over 70% reduction in gate count despite the large number of inputs. This is due to the natural association of parity function with XOR operations which can be implemented more efficiently over the logic basis (AND, XOR, NOT) rather than (AND, OR, NOT).

Optimizing circuits cm42a and cm85a based on the LMC heuristic resulted in increased gate count of approximately 30%. Upon closer inspection, it is observed that the two circuits have pre-optimized descriptions with SOPs that feature very small number of product terms. This leads to compact implementations over the logic basis (AND, OR, NOT). Constructing the same functions over (AND, XOR, NOT) results in strictly worse gate count than the original implementations.

In summary, the LMC heuristic is a powerful tool for gate reduction in logic optimization. Although not unanimously beneficial, the enhanced Boyar-Peralta algorithm is proven to provide significant area savings for majority of the benchmark circuits. On larger circuits, it is generally wise to first optimize the problem using efficient heuristics such as Espresso [28] and then attempt further optimization using the enhanced Boyar-Peralta algorithm on intermediate signals with smaller number of variables. The LMC heuristic is especially potent on functions with no efficient implementations over alternative logic basis.

3.6 Application: Stochastic Random Number Generator

Stochastic computing (SC) [90] is an alternative approach to conventional binary computing. The main feature of SC is the representation of operands using streams of random bits known as stochastic numbers (SNs). In doing so, computations that are normally expensive in conventional binary computing can be approximated using stochastic logic (which requires minimal hardware) for efficient implementations. At the same time, stochastic circuits have good fault tolerance property [91] and low power requirement. Example applications for SC include image processing [92–94], error control coding (ECC) [95] and digital filter design [96–99].

To generate SNs from the operands, SC relies on stochastic number generators (SNGs). The core principle behind the generation of SN is that the probability of ‘1’s in the stream of random bits is determined by the value of the operand it represents. Moreover, the accuracy of a SC circuit is dependent on the interacting SNs being uncorrelated [100].
Hence, it is not uncommon for each input to a SC circuit to require its own independent SNG. In fact, given how economical the hardware requirements to compute stochastic logic are, SNGs can occupy more than 80% of the area for a SC circuit as reported in [92].

To remedy the excessive circuit area dedicated to SNGs, Neugebauer et al. [48] proposed a new random number source (RNS) known as SBoNG for the generation of SNs. The new design differentiates itself from normal SNGs by including a non-linear substitution circuit in its RNS in addition to the typical LFSR. The authors showed that the new design does not interfere with common de-correlation methods unlike typical LFSR-based RNS. Hence, one instance of SBoNG is sufficient in the generation of multiple SNs.

The non-linear substitution circuit used in SBoNG is referenced from [101]. It is a 4-bit RNS. Hence, one instance of SBoNG is sufficient in the generation of multiple SNs.

The enhanced two-step algorithm. The best observed result is depicted in Figure 3.9. In [48], the substitution circuit is directly synthesized using the Altera Quartus Prime software. For a competitive implementation, the same design does not interfere with common de-correlation methods unlike typical LFSR-based RNS. Hence, one instance of SBoNG is sufficient in the generation of multiple SNs.

The non-linear substitution circuit used in SBoNG is referenced from [101]. It is a 4-bit substitution \( \{f_{S1}, f_{S2}, f_{S3}, f_{S4}\} \in F_{SBoNG} \) with the following truth vectors:

\[
\begin{align*}
    f_{S1} &= [0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0]T \\
    f_{S2} &= [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0]T \\
    f_{S3} &= [1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0]T \\
    f_{S4} &= [0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1]T
\end{align*}
\]

In [48], the substitution circuit is directly synthesized using the Altera Quartus Prime software. For a competitive implementation, the same \( F_{SBoNG} \) circuit is optimized using the enhanced two-step algorithm. The best observed result is depicted in Figure 3.9.

| \( t_1 \) | \( t_2 \) | \( t_3 \) | \( t_4 \) | \( t_5 \) | \( t_6 \) | \( t_7 \) | \( t_8 \) | \( t_9 \) | \( t_{10} \) | \( t_{11} \) | \( t_{12} \) | \( t_{13} \) | \( t_{14} \) | \( t_{15} \) | \( y_1 \) | \( y_2 \) | \( y_3 \) | \( y_4 \) |
| \( x_1 \oplus x_2 \) | \( x_2 \oplus x_3 \) | \( t_1 \times t_2 \) | \( x_4 \oplus t_3 \) | \( t_5 = t_4 \oplus t_2 \) | \( t_6 = x_1 \oplus x_4 \) | \( t_7 = t_6 \times t_4 \) | \( t_8 = x_1 \times t_4 \) | \( t_9 = t_6 \times t_5 \) | \( t_{10} = t_3 \oplus t_9 \) | \( y_1 = t_7 \oplus t_1 \) | \( y_2 = (t_8 \oplus t_{10})' \) | \( t_{11} = t_3 \oplus y_1 \) | \( t_{12} = t_1 \oplus t_5 \) | \( t_{13} = x_1 \oplus t_3 \) | \( t_{14} = t_{12} \times t_{11} \) | \( t_{15} = t_{12} \times t_{10} \) | \( y_3 = (x_3 \oplus t_{14})' \) | \( y_4 = t_{15} \oplus t_{13} \) |

**Figure 3.9:** 19-gate implementation of the SBoNG substitution circuit. The 4-bit inputs are \( X = \{x_1, x_2, x_3, x_4\} \) and the 4-bit outputs are \( Y = \{y_1, y_2, y_3, y_4\} \).

The original SBoNG substitution circuit in [48] has a reported gate count of 15 AND gates and 4 OR gates. However, a significant number of the gates used have three or more inputs. For fairer comparison, Table 3.6 tabulates the number of two-input gates required by both circuits. An additional metric in NAND2 gate equivalent is provided for further context.

To observe the impact on power and speed performances by the proposed implementation, two 8-bit SBoNG RNS circuits are implemented using the original and the proposed substitution circuits respectively. The LFSRs used in both design use the same characteristic polynomial \( x^8 + x^6 + x^5 + x^4 + 1 \) as in [48]. Both circuits are synthesized using
Table 3.6: Comparison of circuit size between the original and proposed SBoNG substitution circuits.

<table>
<thead>
<tr>
<th>Work</th>
<th>Gate count</th>
<th>Gate equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AND</td>
<td>OR</td>
</tr>
<tr>
<td>Original</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>This work</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Quartus Prime Version 17.1.0 for Intel FPGA Cyclone IV EP4CE6E22C8 and the results are tabulated in Table 3.7.

Table 3.7: FPGA implementation results for both 8-bit SBoNG RNSs.

<table>
<thead>
<tr>
<th>Work</th>
<th>Logic element (LE)</th>
<th>$F_{\text{max}}$ (MHz)</th>
<th>Power (mW)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>32</td>
<td>358.94</td>
<td>43.19</td>
<td>43.33</td>
</tr>
<tr>
<td>This work</td>
<td>30</td>
<td>383.00</td>
<td>42.82</td>
<td>42.95</td>
</tr>
</tbody>
</table>

The proposed 19-gate implementation of the SBoNG substitution circuit achieved a circuit size improvement of 17.5% in NAND2 equivalent. At the same time, approximately 24% reduction in dynamic power is observed from the FPGA implementations. A small improvement of 6.7% is also noted for the maximum supported frequency $F_{\text{max}}$.

All in all, the SBoNG RNS is a major contribution towards the adoption of SC in modern computing for specific applications that can benefit from its features such as fault tolerance and low power. For instances where larger range of values for the operands or higher accuracy is necessary, the SBoNG circuit would require several more substitution circuits in its implementation to generate SNs of sufficient lengths. The enhanced two-step algorithm allowed the derivation of a more economical implementation of the SBoNG substitution circuit to mitigate the hardware cost in these scenarios.
Deterministic AND-Minimization through Reed-Muller Decomposition

4.1 Introduction

In Chapter 3, an enhanced two-step algorithm is proposed to solve for optimal multiplicative complexity circuit based on the LMC heuristic. The enhancements enable a more reliable approach to solve for optimal multiplicative complexity implementations through improved computational time and average quality of results. However, it is undeniable that a certain degree of undesired inconsistencies still remains in the algorithm as randomness is still a core element of the AND-minimization step.

In this chapter, the potential for a deterministic algorithm is explored for LMC logic optimization. Observations on applications where the LMC logic optimization algorithms were previously applied (mainly cryptographic substitution circuits [29,78]) revealed that most of them involve functions that are of lower bounded multiplicative complexity. With knowledge on the lower bound rule in [83], a new approach to optimize lower bounded expressions is devised based on decomposition of Reed-Muller expressions.

4.2 Preliminaries

4.2.1 Modulo-2 Arithmetic

Modular arithmetic is an arithmetic system for integers where values exceeding the modulus “wrap around”, i.e. no values will be equal or larger than the defined modulus [102]. For example, a classic 12-hour clock is an example for a modulo-12 system. Mathematically, the value of an integer after modulo operation can be obtained by finding the remainder after division by the modulus. While modular arithmetic is an extensive field of study, discussion in this chapter will be restricted to modulo-2 arithmetic, as it has
significant implications to the logic basis (AND, XOR, NOT).

Modulo-2 arithmetic is a specific arithmetic system where the modulus is 2. Consequently, the only possible values to exist within this arithmetic system are ‘0’s and ‘1’s. Therefore, the modulo-2 arithmetic system resembles the binary number system used in modern computing operations [103]. More importantly, mathematical operations that exist within the modulo-2 field such as additions and multiplications are closely related to the logic basis (AND, XOR, NOT), which is the foundation for LMC logic optimization.

For instance, the XOR function is equivalent to mod-2 addition and can be expressed in both logical and mathematical notations as shown in (4.2.1). In this case, $c$ represents the sum resulting from the addition of $a$ and $b$.

$$a \oplus b = a + b = c \quad (4.2.1)$$

Similarly, the AND function is equivalent to mod-2 multiplication. The logical and mathematical notations are shown in (4.2.2), where $c$ is the product resulting from the multiplication of $a$ and $b$.

$$a \odot b = a \times b = c \quad (4.2.2)$$

Finally, the logic NOT is mathematically equivalent to mod-2 addition with the value ‘1’. With reference to (4.2.3), $c$ is referred to as the inverse or negation of $a$.

$$\overline{a} = a + 1 = c \quad (4.2.3)$$

Understanding the mathematical significance of the XOR and AND gates has vital implications on the discussions in the subsequent sections. In this regard, it is emphasized that the use of the ‘+’ sign in this thesis indicates logical XOR (mod-2 addition) and not to be misinterpreted as logical OR.

4.2.2 Reed-Muller Expression

Reed-Muller expressions have significant implications in the field of multilevel logic optimization [66]. In this case, they are used as a form of logic expression to describe the circuit output as a function of the circuit inputs over the logic basis (AND, XOR, NOT) [104]. Reed-Muller expressions can be generalized into two main categories, namely Fixed Polarity Reed-Muller (FPRM) expressions [105] and Mixed Polarity Reed-Muller (MPRM) expressions [106]. FPRM implies that all variables in the Reed-Muller
expression appear either uncomplemented (Positive Polarity) or complemented (Negative Polarity) but not both whereas MPRM expressions include both uncomplemented and complemented variables. In subsequent sections, discussions are mostly limited to the use of Positive Polarity Reed-Muller (PPRM) expressions. However, most of the properties demonstrated can be applied to NPRM expressions without significant adjustment.

**Definition 10.** Given a function with \( n \)-bit inputs \( x_1, x_2, \ldots, x_n \), its PPRM expression can be defined as shown in (4.2.4).

\[
f(x_1, x_2, \ldots, x_n) = a_0 \pi_0 + a_1 \pi_1 + \ldots + a_{2^n-1} \pi_{2^n-1}
\]  

(4.2.4)

Where,

\[
a_0, a_1, \ldots, a_{2^n-1} \in \{0, 1\}
\]

\[
\pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \ldots, \pi_{2^n-1} = 1, x_1, x_2, x_2x_1, x_3, \ldots, x_nx_{n-1}x_1
\]

While PPRM expressions resemble conventional SOP, it is important to distinguish the minor differences between the both. Specifically, PPRM expressions are ESOP, where the additions invoked in the expressions are functionally equivalent to XOR function as per descriptions in Section 4.2.1.

For any arbitrary function, there is only one unique representation in PPRM form. This unique PPRM expression can be derived easily by multiplying the truth vector of the function with the \( n \)-variable transform matrix \( T_n \) given in (4.2.5) where \( T_0 = [1] \).

\[
T_n = \begin{bmatrix} T_{n-1} & 0 \\ T_{n-1} & T_{n-1} \end{bmatrix}
\]  

(4.2.5)

The PPRM expressions allow polynomial-esque expressions for logic functions which are relevant to show important mathematical manipulation techniques in LMC logic optimization. However, arithmetic operations such as additions and multiplications between PPRM expressions also have unique properties in comparison to conventional arithmetic.

Addition of PPRM expressions are performed through mod-2 additions on the *coefficients* of each unique cube in the expressions. This process is illustrated in (4.2.6).

\[
\begin{align*}
\text{exp.1} & \quad (x_1 + x_1x_2 + x_1x_3) \\
\text{exp.2} & \quad (x_2 + x_1x_2 + x_2x_3)
\end{align*}
\]

\[
= x_1 + x_2 + x_1x_3 + x_2x_3
\]  

(4.2.6)

On the other hand, multiplications of PPRM expressions also perform slightly different
than normal arithmetic. When multiplying two cubes that include one or more identical variable(s), the order of that variable does not increase as demonstrated in (4.2.7).

\[
(x_1x_2) \times (x_2x_3) = x_1x_2x_3 \\
\text{exp.1} \quad \text{exp.2}
\]

Regardless, multiplications of longer expressions follow the distributive law in conventional multiplication of polynomials but with the cancellation of repeated cubes through mod-2 additions (if applicable) as shown previously in (4.2.6).

\[
(x_1 + x_2) \times (x_3 + x_4) = x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 \\
\text{exp.1} \quad \text{exp.2}
\]

### 4.3 Decomposition of PPRM Expressions

In conventional arithmetics, the reduction on number of multiplications is associated with the decomposition (a.k.a. factorization) process. Since Reed-Muller expressions over the logic basis (AND, XOR, NOT) are essentially equivalent to modulo-2 arithmetic over $GF(2)$, it stands to reason that the same process can be leveraged to achieve implementations with low number of multiplications. In this section, the relevance of PPRM decomposition on optimal multiplicative complexity is discussed using illustrative examples.

**Example 2.** Given $f = x_1x_2 + x_2x_3$,

\[
f = x_2(x_1 + x_3)
\]

Example 2 shows a simple case of decomposition where a function $f$ requiring two AND gates and one XOR gate in its PPRM form can be reduced to just one AND gate and one XOR gate post-decomposition. However, expressions of higher degree and/or number of variables tend to require further manipulations to achieve the same effect. For this purpose, three important sections on a decomposed expression can be defined for ease of discussion.

Given $f \in \langle x_1, x_2, \ldots, x_n \rangle$ and let $x_i$ be the chosen factor for decomposition,

\[
f = (x_i)_{a} (f_b)_{b} + (f_c)_{c}
\]

55
Chapter 4: Deterministic AND-Minimization through Reed-Muller Decomposition

Where,

\[ a = \text{multiplier} \]
\[ b = \text{factored expression} \]
\[ c = \text{remainder} \]

Given that a function can be decomposed using any of its input literals, the first stage of decomposition will result in \( n \) different factored expressions, each using a unique factor \( x_1, x_2, ..., x_n \). Subsequently, each of these factored expressions can be subjected to further stages of decomposition. However, since the factored expressions are all functions of \( n-1 \) literals, the second stage will thus have only \( n-1 \) potential factors for decomposition. This process of multi-stage decomposition can be performed until the factored expressions have a degree of \( d = 1 \) where no further decomposition can be attempted.

*Example 3.* Given \( f \in \langle x_1, x_2, ..., x_4 \rangle \) and

\[ f = x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3 \quad (4.3.3) \]

The first stage of decomposition results in \( n = 4 \) possible factored expressions \( f_1, f_2, ..., f_4 \):

\[ f_1 = x_1(x_2x_3 + x_2x_4 + x_3x_4) + x_2x_3 \]
\[ f_2 = x_2(x_1x_3 + x_1x_4 + x_3) + x_1x_3x_4 \]
\[ f_3 = x_3(x_1x_2 + x_1x_4 + x_2) + x_1x_2x_4 \]
\[ f_4 = x_4(x_1x_2 + x_1x_3) + x_1x_2x_3 + x_2x_3 \]

Since the factored expressions for \( f_1, f_2, ..., f_4 \) are all of degree \( d > 1 \), further decomposition is still possible. Using the factored expression for \( f_2 \) as example:

\[ f_2 = x_2 \underbrace{(x_1x_3 + x_1x_4 + x_3)}_{b} + x_1x_3x_4 \quad (4.3.4) \]

The factored expression \( b \) of \( f_2 \) is a function of three literals \( x_1, x_3, x_4 \). As such, there are \( n-1 = 3 \) possible factors for the second stage of decomposition, resulting in \( f_{2,1}, f_{2,2}, f_{2,3} \) respectively:

\[ f_{2,1} = x_1(x_3 + x_4) + x_3 \]
\[ f_{2,2} = x_3(x_1) + x_1x_4 + x_3 \]
\[ f_{2,3} = x_4(x_1) + x_1x_3 + x_3 \]
Chapter 4: Deterministic AND-Minimization through Reed-Muller Decomposition

Naturally, the second stage of decomposition is applied to \( f_1, f_3, f_4 \) as well. At this point, the factored expressions for all of \( f_{2,1}, f_{2,2}, f_{2,3} \) are of degree \( d = 1 \). This completes the multi-stage decomposition for the original function \( f \).

The entire process of decomposition can be illustrated as a tree diagram. In this case, an \( n \)-input function to be decomposed \( f \in \langle x_1, x_2, \ldots, x_n \rangle \) would be the root of the tree. Each stage of decomposition is equivalent to forming a number of *children branches* with a unique set of multipliers \( a \), factored expressions \( b \) and remainders \( c \). Finally, the tree will be terminated with *leaves* containing factored expressions of degree \( d = 1 \). Figure 4.1 shows an example of the tree diagram for a function \( f \) of \( n = 3 \) inputs and degree \( d = 3 \).

![Figure 4.1: Tree diagram for an \( n = 3, d = 3 \) function.](image)

4.4 Tree Search Algorithm for Lower Bounded Problems

Thus far, a procedure is established to leverage multi-stage decomposition to arrive at factored expressions with degrees of \( d = 1 \) in a tree-like structure. This process has significant implications to the construction of optimal multiplicative complexity circuits for a lower bounded function.

**Lemma 3.** Given a function \( f \) in PPRM form, the maximum depth of the tree diagram generated through decomposition is \( d - 1 \).

**Proof.** Let \( f' \) be an exclusive disjunction of cubes in \( f \) which contains the chosen factor \( x_i \). It can be inferred that \( d_{f'} \leq d \) where \( d_{f'} \) and \( d \) are the degrees of \( f' \) and \( f \) respectively. This is due to \( f' \) being a subset of cubes from \( f \).
Decomposition using the chosen literal $x_i$ forms a factored expression $b$ equivalent to $f'$ sans the literal $x_i$. Hence, the degree of the factored expression $b$ is $d_b = d_{f'} - 1$. Given the relationship between $d_{f'}$ and $d$, it then follows that $d_b \leq d - 1$. In other words, each stage of decomposition would reduce the degree of the factored expression by at least one compared to the original function $f$. Given the multi-stage decomposition is terminated when $d_b = 1$, it follows that the maximum level of decomposition is equal to $d - 1$. \[\square\]

Lemma 3 enables quantification of the maximum depth of each leaf generated in the tree diagram. Coincidentally, the lower bound rule of multiplicative complexity (see Lemma 1) states that the minimum number of multiplications required to compute a function $f$ is equal to $d - 1$.

**Lemma 4.** Given a tree diagram generated through multi-stage decomposition of a lower bounded function $f$, if each stage of decomposition for a leaf can be constructed using only one multiplication, then the leaf represents an optimal multiplicative complexity implementation for the function $f$.

**Proof.** Given a function $f$ in PPRM factored form $f = a \times b + c$ as per the definitions in (4.3.2), the decomposition is said to cost one multiplication if the multiplication between $a$ and $b$ is the only new product required to construct $f$ from $b$.

Lemma 3 gives the maximum depth of the tree to be $d - 1$. Hence, if each stage of decomposition for a leaf is verified to cost one multiplication, the total number of multiplications for the leaf would be $d - 1$ which agrees with the lower bound rule in Lemma 1. \[\square\]

Lemma 4 provides a clear-cut approach to discovering optimal multiplicative complexity solutions for a lower bounded problem. Given a fully expanded tree diagram obtained through multi-stage decomposition, the goal is to check for each leaf of the tree diagram for ones that satisfy the requirement in Lemma 4. The proposed verification process is best demonstrated through an illustrative example.

**Example 4.** Given one of the factored expressions from the last stage of decomposition from Example 3 (one of the leaves in the tree diagram):

$$f_{2,1} = x_1 \times (x_3 + x_4) + x_3$$  \tag{4.4.1}

From (4.4.1), it is immediately evident that $f_{2,1}$ can be constructed from $b = x_3 + x_4$ with one multiplication. Hence, the verification process continues on the parent of the current leaf:

$$f_2 = x_2 \times (f_{2,1}) + x_1x_3x_4$$  \tag{4.4.2}
At first glance, $f_2$ seems to require more than one multiplications due to the presence of $c = x_1x_3x_4$ in the remainder. However, it is possible to manipulate the remainder $c$ through addition of literals in multiplier $a$ as per the equation below:

$$
 f = \left( \frac{x_i}{a} \right) (f_b) + \left( \frac{f_c}{c} \right) \\
 = \left( \frac{x_i + x_j}{a} \right) (f_b) + \left( \frac{f_c + x_j f_b}{c} \right) 
$$

(4.4.3)

In (4.4.3), the addition of the literal $x_j$ into multiplier $a$ results in a “compensating expression” $x_j(f_1)$ to be added to $c$ to maintain equality. Due to the cancellation property of mod-2 addition (XOR), this process allows the manipulation of cubes in $c$. Note that the addition of literals in multiplier $a$ does not cost additional multiplication which is important in regards to Lemma 4. Returning to the example on $f_2$:

$$
 f_2 = \left( \frac{x_2 + x_1}{a} \right) (f_{2,1}) + x_1x_3x_4 + x_1x_3 + x_1x_4 + x_1x_3 \\
 = \left( \frac{x_2 + x_1}{a} \right) (f_{2,1}) + x_1x_3x_4 + x_1x_4 
$$

(4.4.4)

By adding the literal $x_1$ to the multiplier $a$, the function $f_2$ can be manipulated into a new remainder $c = x_1x_3x_4 + x_1x_4$. However, it is obviously still not multiplication-free.

$$
 f_2 = \left( \frac{x_2 + x_1 + x_3}{a} \right) (f_{2,1}) + x_1x_3x_4 + x_1x_4 + x_1x_3 + x_1x_3x_4 + x_3 \\
 = \left( \frac{x_2 + x_1 + x_3}{a} \right) (f_{2,1}) + x_1x_4 + x_1x_3 + x_3 \\
 = \left( \frac{x_2 + x_1 + x_3}{a} \right) (f_{2,1}) + f_{2,1} 
$$

(4.4.5)

As shown in (4.4.5), addition of another literal $x_3$ into $a$ manipulates $f_2$ into a form where $f_2$ can be constructed from its factored expression $f_{2,1}$ with just one multiplication.

Since it is verified that the two-stage decomposition for $f_{2,1}$ can be constructed with one multiplication per stage, $f_{2,1}$ is verified as a solution with optimal multiplicative complexity $c_\Lambda(f) = d - 1 = 2$.

From Example 4, a tree search algorithm (TSA) is proposed to identify leaves in the decomposition tree diagram that satisfy Lemma 4 for optimal multiplicative complexity construction. The one-multiplication-per-level verification process is the key element of the TSA. In this process, the algorithm is designed to attempt additions of $2^n$ possible combinations of literals into the multiplier $a$ in order to ascertain the possibility of a multiplication-free remainder $c$. For clarification, a multiplication-free remainder $c$ can
Chapter 4: Deterministic AND-Minimization through Reed-Muller Decomposition

either (a) requires no multiplication or (b) requires only multiplications that exist in factored expression $b$. Let $\pi_0, \pi_1, \pi_2, \pi_3, \ldots, \pi_{2^n-1} = 0, x_1, x_2, x_1 + x_2, \ldots, x_1 + x_2 + \ldots + x_n$ represents the $2^n$ combinations of literals, the procedure to verify the possibility of a multiplication-free remainder on a node of the tree diagram can be summarized in Algorithm 4.

**Algorithm 4** Pseudocode for verifying a multiplication-free remainder

```plaintext
1: begin
2:   valid = 0
3: for $i = 0$ to $2^n - 1$ do
4:     comp = $(\pi_i)(b)$
5:     $c' = c + \text{comp}$
6:     if degree of $c' \leq 1$ then
7:       valid = 1
8:     else if all multiplications in $c'$ exist in $b$ then
9:       valid = 1
10: end if
11: end for
12: return valid
13: end
```

The verification process is to be attempted on every leaf of the expanded tree diagram. Once a node is verified to be valid, the same verification process is to be attempted on the parent node. If all nodes on the path between a leaf and the root of the tree satisfy the one-multiplication-per-level rule, an optimal solution is obtained. On the contrary, if at any point on the path a node fails the verification process, the associated leaf is rejected without the need of further verification on its ancestors. It is important for the verification process to be applied in the “upwards” direction on the tree diagram. This is because knowledge on the multiplications used in its descendants is required when verifying a node. Algorithm 5 gives the procedure for a full search on an expanded tree diagram for all optimal solutions.

### 4.4.1 Regarding Interchangeable Literals

In the process of expanding the tree diagram, each stage of decomposition involves factorization using all $n$ possible literals (see Example 3). This results in a number of child nodes equal to the number of possible factors for each parent node. However, it is possible to reduce the number of branches for a parent node while still retaining the same number of optimal solutions by exploiting the property of interchangeable literals.

By definition, interchangeable literals refer to a pair of literals in a function $f$ whereby the output of $f$ will not be affected if their values are interchanged. For instance, consider
Algorithm 5 Pseudocode for full search on an expanded tree

1: begin
2: \( S = \{ \} \)
3: for \( i = 1 \) to \( n_l \) do \( \triangleright n_l = \) number of leaves
4: \( \text{level} = d_l \) \( \triangleright d_l = \) depth of leaf
5: while \( \text{level} > 0 \) do
6: \( \text{run Algorithm 4 on current node} \)
7: \( \text{if valid} = 1 \) then
8: \( \text{level} = \text{level} - 1 \)
9: \( \text{if level} = 0 \) then
10: \( \text{record optimal solution in} \ S \)
11: \( \text{end if} \)
12: \( \text{else} \)
13: \( \text{exit} \)
14: \( \text{end if} \)
15: \( \text{end while} \)
16: end for
17: return \( S \)
18: end

the function \( f \) from Example 3:

\[
f = x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3
\]

In this case, \( x_2 \) and \( x_3 \) are interchangeable for the function \( f \) as demonstrated below:

\[
f = x_1x_3x_2 + x_1x_3x_4 + x_1x_2x_4 + x_3x_2
\]
\[
= x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3
\]

The benefit of having interchangeable literals in a function is that their interchangeability pertains to the solutions as well. From Example 4 an optimal solution for \( f \) with \( c_\lambda(f) = 2 \) can be obtained as follows:

\[
f = x_2(x_1(x_3 + x_4) + x_3) + x_1(x_3 + x_4) + x_3
\]

By leveraging the property of interchangeable literals, it follows that replacing all instances of \( x_2 \) with \( x_3 \) and vice versa would result in another optimal solution:

\[
f = x_3(x_1(x_2 + x_4) + x_2) + x_1(x_2 + x_4) + x_2
\]

The implication of this is that when interchangeable literals are present, the tree expansion process only needs to include decomposition using one of the literals to effectively discover the solutions associated with the other. Hence, there will be less leaves in the
4.5 Product Sharing for Multiple-Output Problem

Product sharing is an attempt to use products required in previously solved functions in the construction of subsequent functions in order to further reduce the total number of multiplications in a multiple-output problem. In Section 2.5, the importance of product sharing in deriving low multiplicative complexity solutions is explained. The same principle applies to the current approach to LMC optimization. However, due to the significant differences in execution between the original randomized search algorithm and the proposed TSA, there is a need to identify instances where products from previously solved functions can be applied in the latter in a meaningful way.

First, there are avenues to apply the free products in the PPRM decomposition process. Let \( p_1, p_2, ..., p_m \in P \) be a set of \( m \) products accumulated from solving previous functions. Given the function \( f \) as the subsequent function to be optimized, it is possible to change the expression at each stage of decomposition completely by adding any number of elements from \( P \) to the function \( f \).

\[
f' = f + p_i
\]  (4.5.1)

In (4.5.1), a new expression \( f' \) is obtained by adding a product \( p_i \) to the original function \( f \). Since the process to generate \( f' \) does not cost any multiplication, \( f' \) can be subjected to the same multi-stage decomposition in conjunction with \( f \) for more potential solutions. This results in a new tree diagram structure where additions with products from \( P \) are executed in between stages of decomposition as illustrated in Figure 4.2. Due to the extra levels of branching associated with product sharing, the decomposition process now occurs once per two depth level of the tree diagram. Since the number of multiplications corresponds to the levels of decomposition, the cost of multiplications is now evaluated as one per two depth level instead.

The true benefit of product sharing in the decomposition process occurs when additions with certain products result in a function \( f' \) where \( d_{f'} < d_f \). When possible, these functions of lower degree have the potential to be solved using less multiplication than the original function \( f \) as the lower bound of multiplicative complexity is associated with the degree of a function as per Lemma 1. In the actual decomposition process, all \( f' \) with \( d_{f'} = d_f \) should still be considered valid branches in the tree diagram as \( f' \) with lower degree (if available) may fail the one-multiplication-per-decomposition verification that is applied post-decomposition. Nevertheless, all \( f' \) with \( d_{f'} > d_f \) can be safely ignored as they exceed the known multiplicative complexity of the target function.
Figure 4.2: Tree diagram with product sharing.
Aside from the decomposition process, the product set $P$ can be applied in the proposed TSA described in Section 4.4 (specifically in Algorithm 4). In Example 4, it is shown that the addition of literals to the multiplier $a$ can manipulate expression in the remainder $c$ into a form that does not require additional multiplications. The free products in $P$ broaden the available options for this purpose in two ways.

First of all is the ability to form a new remainder $f'_c$ through addition of $p_i$ to the original $f_c$ as shown in (4.5.2). Since $p_i$ is a free product, verification (through Algorithm 4) now involves a new remainder $f'_c$ instead of the original remainder $f_c$.

$$f = \left( x_i \right)_a \left( f_b \right)_b + \left( f'_c \right)_c + p_i \quad (4.5.2)$$

Where,

$$f'_c = f_c + p_i$$

The second approach involves adding free products $p_i$ into the multiplier $a$ to generate unique additions to the remainder $c$ as illustrated in (4.5.3). This is the same concept as adding literals to the multiplier $a$ shown in Example 4.

$$f = \left( x_i + p_i \right)_a \left( f_b \right)_b + \left( f_c + p_i f_b \right)_c \quad (4.5.3)$$

Together, both options add to the flexibility in manipulating the remainder $c$ as evident in (4.5.2) and (4.5.3). As a result, nodes in a tree diagram that would normally fail the verification process by Algorithm 4 have the potential to produce optimal solutions with the “aid” of product sharing.

From the algorithm perspective, the verification procedure described in Algorithm 4 needs to be enhanced to incorporate the feature of product sharing in a multiple-output problem. Given a function $f$ with $n$ input literals $x_1, x_2, ..., x_n \in X$ and a current product set of $m$ elements $p_1, p_2, ..., p_m \in P$, let $y_1, y_2, ..., y_{n+m} \in Y$ be a joint set of elements from the input literals and the product set $\{X, P\} \in Y$ in which $y_1, y_2, ..., y_{n+m} = x_1, x_2, ..., x_n, p_1, p_2, ..., p_m$. From there, let $\pi_0, \pi_1, \pi_2, \pi_3, ..., \pi_{2^n+m-1} = 0, y_1, y_2, y_1 + y_2, ..., y_1 + y_2 + ... + y_{n+m}$ represent the $2^{n+m}$ possible combinations of elements in $Y$. Similarly, let $\theta_0, \theta_1, \theta_2, \theta_3, ..., \theta_{m-1} = 0, p_1, p_2, p_1 + p_2, ..., p_1 + p_2 + ... + p_m$ represent the $2^m$ possible combinations of elements in just the product set $P$. A new verification algorithm with consideration for product sharing is described as per Algorithm 6.
Algorithm 6 Pseudocode for verifying a multiplication-free remainder with product sharing

1: begin
2: valid = 0
3: for $i = 0$ to $2^{n+m} - 1$ do
4: \hspace{1em} comp = $(\pi_i)(b)$
5: \hspace{1em} \hspace{1em} $c' = c + \text{comp}$
6: \hspace{1em} for $j = 0$ to $2^m - 1$ do
7: \hspace{2em} \hspace{1em} $c'' = c' + \theta_j$
8: \hspace{2em} \hspace{1em} if degree of $c'' \leq 1$ then
9: \hspace{2em} \hspace{2em} \hspace{1em} \hspace{1em} valid = 1
10: \hspace{2em} \hspace{1em} else if all multiplications in $c''$ exist in $b$ then
11: \hspace{2em} \hspace{2em} \hspace{1em} \hspace{1em} valid = 1
12: \hspace{2em} \hspace{1em} end if
13: \hspace{1em} end for
14: \hspace{1em} end for
15: return valid
16: end

4.5.1 Regarding Leaves of Higher Depth

In the previous section, it was briefly mentioned how product sharing can result in a new $f'$ with $d_{f'} < d_f$ when applied during the decomposition process. Due to the potential reduction of degree in $f'$, certain branches of the tree diagram may terminate earlier since the depth of a leaf is associated with the degree of $f'$ as per Lemma 3. These leaves of shallower depth are more valuable than the others as they represent the potential for solutions with lower number of multiplications.

For this reason, when the TSA is executed on the tree diagram of a function as a part of a multiple-output problem, priority is given to leaves of shallower depth. Whenever a leaf is verified to be a valid solution to the function, any remaining leaves of higher depth can be discarded as they would naturally cost more multiplications equal to the difference in depth. Algorithm 7 gives the procedure used by the proposed TSA on a function in a multiple-output problem.

4.6 Performance

4.6.1 Time Complexity

In this section, the worse-case time complexity for the proposed TSA on a multiple-output problem is discussed. For this purpose, it is important to first analyze the time complexity of Algorithm 6 as a sub-algorithm for the TSA. In this case, the time complexity is a function of two variables $T(n, m)$ where $n$ is the number of inputs and $m$ is the number of elements in the products set $P$. The operations in the nested loop for
Algorithm 7 Pseudocode for full search on an expanded tree with product sharing

1: begin
2: $S = \emptyset$
3: sort leaves in ascending order of depth
4: $next = 1$
5: for $i = 1$ to $n_l$ do \hfill $\triangleright n_l =$ number of leaves
6: \hfill $\triangleright d_l =$ depth of leaf
7: \hfill $\triangleright$ while $level > 0$ do
8: \hfill run Algorithm 6 on current node
9: \hfill if $valid = 1$ then
10: \hfill $level = level - 1$
11: \hfill if $level = 0$ then
12: \hfill record optimal solution in $S$
13: \hfill add all used products into $P$
14: \hfill $next = 0$
15: \hfill end if
16: \hfill else
17: \hfill exit
18: \hfill end if
19: \hfill end while
20: \hfill $\triangleright$ if all leaves in current depth are done then
21: \hfill $\triangleright$ if $next = 0$ then
22: \hfill $\triangleright$ ignore leaves of higher depth if solution(s) found
23: \hfill $\triangleright$ end if
24: \hfill end if
25: end for
26: return $S, P$
27: end

Algorithm 6 is dominated by the mod-2 addition of expressions in terms of complexity. It is executed by first concatenating the two addends, followed by the sorting of cubes and removal of consecutive duplicates through scanning\(^1\). The entire process has a time complexity of $O(2^n \log(2^n))$\(^2\).

Given the number of iterations for the outer and inner loops in Algorithm 6 to be $2^{n+m}$ and $2^m$ respectively, the worst-case time complexity of Algorithm 6 can be deduced to be $O(2^{2(n+m)} \log(2^n))$.

The proposed TSA (Algorithm 7) executes Algorithm 6 recursively in a two-level nested loop. The number of iterations for each loop is determined by the number of leaves and the tree depth respectively as follows:

- From the multi-stage decomposition illustrated in Figure 4.2, it can be inferred

\(^1\)This is equivalent to the process of XOR-ing cubes between expressions as shown in (4.2.6).
\(^2\)A general algorithm to eliminate duplicates through sorting and scanning has a known time complexity of $O(n \log n)$. However, the variable $n$ in this context refers to the number of elements to be sorted. Mod-2 addition in Algorithm 6 involves the sorting of cubes. Given $n$ as the number of inputs as per our initial definition, the maximum number of cubes to be sorted is thus $2^n$. Consequently, the time complexity of the mod-2 addition is deduced to be $O(2^n \log(2^n))$. 
that the number of leaves is at most \((2^m n)^{d-1}\).

- Given the application on lower bounded functions, the maximum tree depth can be determined as \(d - 1\) following Lemma 1.

Consolidating all the information thus far enables the conclusion that the worse-case time complexity for the proposed TSA as \(O(d(2^m n)^{d-1}(2^{2(n+m)}) \log(2^n))\). For comparison, Algorithm 5 has a worst-case time complexity of \(O(d(n)^{d-1}(2^{2n}) \log(2^n))\) without the consideration for product sharing.

### 4.6.2 Computation Time

Using worst-case time complexity as a metric for comparison between the proposed TSA and the Boyar-Peralta algorithm is difficult as the metric is unbounded for the latter. In order to compare the performance of both algorithms, this experiment resorts to using average computation time as an ad hoc metric for this purpose. Both algorithms are executed using MATLAB R2012b on a system equipped with Intel Core i5-4690 processor @ 3.50GHz and 8GB of RAM. To mitigate the influence from external factors, 100 executions of both algorithms are done per problem to obtain a fair average. As for the problems for optimization, four functions of the PRESENT 4-bit S-Box \([42]\) are chosen as pseudo representation of a real world application \(\{f_{P1}, f_{P2}, f_{P3}, f_{P4}\}\). In addition, a function challenged in \([29]\) is included as \(f_{BP}\) for further experimentation.

The truth vectors for the aforementioned functions are given as follows:

\[
\begin{align*}
    f_{P1} &= [0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0]^T \\
    f_{P2} &= [0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]^T \\
    f_{P3} &= [1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0]^T \\
    f_{P4} &= [1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0]^T \\
    f_{BP} &= [0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1]^T \\
\end{align*}
\]

Naturally, both algorithms must be compared based on their performance on multiple-output problems. For this reason, the four functions of the PRESENT S-Box are collectively treated as \(\{f_{P1}, f_{P2}, f_{P3}, f_{P4}\} \in F_{PRESENT}\) and optimized with product sharing in mind. The same experiment is applied to Canright’s \(GF(2^4)\) multiplicative inversion circuit \([75]\) represented as a four-output function \(F_{inv}\). The results are reported in Table 4.1.
Table 4.1: Comparison of computation time

<table>
<thead>
<tr>
<th>Function</th>
<th>( c_A(f) )</th>
<th>Computation time (s)</th>
<th>% Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original</td>
<td>Proposed</td>
</tr>
<tr>
<td>( f_{P1} )</td>
<td>1</td>
<td>0.0638</td>
<td>0.0093</td>
</tr>
<tr>
<td>( f_{P2} )</td>
<td>2</td>
<td>5.3874</td>
<td>0.5107</td>
</tr>
<tr>
<td>( f_{P3} )</td>
<td>2</td>
<td>7.9096</td>
<td>0.4087</td>
</tr>
<tr>
<td>( f_{P4} )</td>
<td>2</td>
<td>7.3758</td>
<td>0.4743</td>
</tr>
<tr>
<td>( f_{BP} )</td>
<td>3</td>
<td>688.4390</td>
<td>26.2880</td>
</tr>
<tr>
<td>( F_{PRESENT} )</td>
<td>4</td>
<td>931.7820</td>
<td>64.5891</td>
</tr>
<tr>
<td>( F_{inv} )</td>
<td>5</td>
<td>4041.3000</td>
<td>147.3512</td>
</tr>
</tbody>
</table>

4.6.3 Discussion

Looking at the time complexity analysis of the proposed TSA, it is obvious that the algorithm is not designed to solve large logic optimization problems due to exponential growth in complexity. In fact, the notion is generally true for the LMC heuristic as a whole. To reiterate the authors in [29], the heuristic is effective on small components that exist within a complex function. It contributes to the reduction of gate count in a circuit by isolating these smaller components and optimizing them individually. Therefore, the time complexity is generally acceptable for the nature of applications for the algorithm.

Discussions on computation time, while generally extraneous to the application of the TSA, act as supplements to highlight noteworthy differences between the proposed TSA and the original approach. From Table 4.1, over 85% computation time reduction can be observed in each experiment by the proposed algorithm. This improvement is achieved through a combination of several factors.

When proposing enhancements to the Boyar-Peralta two-step algorithm in Chapter 3, it was discussed how randomness in the original approach results in unpredictability during the optimization process. To be specific, the algorithm allows occurrences in which the same sums or products that already existed in the sample space to be repeatedly generated by the randomized selection procedure. This happens not only when the same pair of signals are selected, but also due to the properties of mod-2 addition and multiplication where different signal pairs may produce the same sum or product. This exacerbates the sample space expansion problem and is mainly responsible for the excessive computation time in the original algorithm. Of course, this problem does not exist in the proposed TSA.

It is also important to underline a major benefit of the proposed TSA when applied on multiple-output problems specifically: it always returns solutions with the best achievable multiplicative complexity provided the functions are lower bounded. In Section 3.4.2, it was discussed how the results obtained from the Boyar-Peralta algorithm may require number of multiplications ranging between the true multiplicative complexity of
the overall problem and a certain upper bound. The ability of the original algorithm to arrive at an optimal solution is heavily dependent on the extent the randomly chosen products can be shared across multiple functions. Because the Boyar-Peralta algorithm gives one random solution per execution, in the unfortunate scenario where the optimal degree of product sharing cannot be achieved, the only solution is to rerun the algorithm on the entire multiple-output problem in hope for a different product set. On the contrary, the proposed TSA collects all optimal solutions (verified leaves) and their respective product set in one execution. Hence, the proposed algorithm can attempt optimization on subsequent functions with the different product sets without the need to redo previous functions. This severely reduces the effective computation time for the proposed TSA.

Last point to note is the exponential increase in computation time for both algorithms following the increase in multiplicative complexity of a problem. This is to be expected following the time complexity analysis in Section 4.6.1 Regardless, due to the much shorter computation time of the proposed algorithm, the exponential increase has a lesser impact on the practicability of the algorithm. Most notably, the proposed algorithm showed tangible improvement over the original algorithm when solving for $F_{\text{PRESENT}}$ and $F_{\text{inv}}$. Given that the original algorithm requires multiple executions to ascertain the quality of the results, the magnitude of improvement is essentially several-fold of the values tabulated in Table 4.1.

### 4.7 Quality of Results

Discussion on the quality of results is much more important pertaining to the role of the proposed algorithm. As an algorithm based on the LMC heuristic for logic optimization, it is important to clarify that the proposed algorithm is generally unable to discover all possible optimal solutions. The reasoning behind this statement is due to the assumption on all decomposed expressions to be in their minimal form when factorizing the first literal $x_i$, i.e. $f = (x_i)(f_b) + f_c$ where $x_i \notin f_b$.

**Example 5.** Given $f = x_1x_2x_3 + x_1x_2 + x_1$,

\begin{align*}
  f & = x_1(x_2x_3 + x_2) + x_1 \\   (4.7.1) \\
  f & = x_1(x_1x_2x_3 + x_2) + x_1 \\   (4.7.2) \\
  f & = x_1(x_2x_3 + x_1x_2) + x_1 \\   (4.7.3)
\end{align*}

Example 5 gives three possible expressions resulting from a mod-2 decomposition of a function $f$. Among the expressions, (4.7.1) represents the minimal form (which the proposed TSA always assumes) whereas (4.7.2) and (4.7.3) are both the non-minimal
form of decomposition. The general consensus is that minimal-form decomposition is strictly better than the latter for maximum reduction to the number of multiplications. However, when a multiple-output problem is concerned, the same statement is no longer true due to the existence of product sharing. The additional multiplications remained in non-minimal form decomposition can potentially be nullified through product sharing whereas the minimal form may not. Hence, it is undeniable that non-minimal decomposition has the potential to require equal or less multiplications than minimal form in this scenario. Regardless, verifying all possible non-minimal forms at every stage of decomposition adds another layer of exponential complexity to the search algorithm.

Another compromise occurs in the remainder manipulation process described in Section 4.4. The proposed TSA considers the additions of all combinations of input literals and/or free products into the multiplier \( a \) portion of the factored expression with the goal of generating a multiplication-free remainder \( c \). However, this process excludes the examination of all \( 2^n \) potential cubes that can be added into \( a \) (unless they are elements of the collective product set \( P \)). Disregarding the impact on algorithm complexity, the main reasoning against this approach is that cubes innately requires additional multiplications as conjunction of literals (see Definition 3). As the algorithm has no means of guaranteeing the sharing of a cube with subsequent functions to nullify the multiplication cost, adding a cube into multiplier \( a \) often results in non-optimal solution for a multiple-output problem. In fact, the proposal to add free products from \( P \) into multiplier \( a \) in Section 4.5 is, to some degree, an attempt to compensate for this issue through the use of products or cubes that have already been determined as mandatory components for solved functions (hence guaranteeing product sharing if used).

Following the discussions above, it is paramount that the proposed TSA be evaluated on the quality of results against the original approach. When examining the best solutions obtained from both algorithms on the problems described in Section 4.6.2, it was observed that they are generally identical in terms of gate count, save for the fact that the original algorithm relies on the factor of chance to discover said solutions within the limited number of trials. However, there are a few notable findings to be reported.

### 4.7.1 Substitution Circuits

In Section 4.6.2 the multiple-output problems \( F_{PRESENT} \) and \( F_{inv} \) are chosen for optimization because existing works have attempted optimization on the same problems following the Boyar-Peralta heuristic in \(^{78}\) and \(^{29}\) respectively. Thus, results from both works serve as a good benchmark for quality of results to evaluate the proposed TSA. Table 4.2 compares the best results achieved by the proposed TSA to the aforementioned works.
Table 4.2: Comparison of logic gate count on $F_{\text{inv}}$ and $F_{\text{PRESENT}}$

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Gate count</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XOR</td>
<td>AND</td>
</tr>
<tr>
<td>Function: $F_{\text{inv}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boyar [29]</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Boyar [79]</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>This work</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Function: $F_{\text{PRESENT}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courtois [78]</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>Courtois [78]^1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>This work</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>This work^2</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

^1 After further optimization through affine AND-OR replacement.

^2 After further optimization through AND/XOR to NAND/XNOR replacement.

Looking at the results on Canright’s $GF(2^4)$ inversion circuit $F_{\text{inv}}$, it appears that the best result discovered by the proposed TSA achieved a reduction of one XOR gate in comparison to the implementation reported in [29]. In fact, the proposed TSA discovered more than one size-15 (total gate count of 15) implementations of $F_{\text{inv}}$. Among the solutions, the best instance achieved a critical path of depth-8 (critical path of 8 gates) compared to the depth-9 implementation of [29]. On the other hand, work in [79] reported a size-17 depth-4 implementation of the same function. Although the implementation in [79] is not exactly optimal in terms of multiplicative complexity, it shows that improvements in other metrics are possible at the cost of gate count by making some adjustments to the Boyar-Peralta algorithm. Figure 4.3 gives the best implementation of the $F_{\text{inv}}$ function by the proposed algorithm.

Results on the 4-bit PRESENT S-Box $F_{\text{PRESENT}}$ provide interesting comparisons as well. In [78], a size-25 implementation of $F_{\text{PRESENT}}$ was reported. The circuit is obtained through a unique approach by converting the $F_{\text{PRESENT}}$ function to a Boolean satisfiability problem (SAT). SAT solver software is then used to obtain a solution that is optimal in regards to the Boyar-Peralta heuristic. The authors commented on the result as “unsatisfactory” in terms of gate count. An exhaustive search where all $2^4$ possible cases of replacing the AND gates in the initial implementation with OR gates was
Their approach resulted in a final size-14 implementation of $F_{\text{PRESENT}}$. In comparison, the proposed TSA returned a size-15 implementation which is significantly better than the initial result in [78] but slightly inferior post affine gate replacement. However, similar gate replacement techniques can also be applied on the result. In fact, since $f_{P3}$ and $f_{P4}$ from $F_{\text{PRESENT}}$ are negative functions (i.e. $f(0) = 1$), the two NOT gates at the end of the circuit can easily be combined with the AND/XOR gate prior into NAND/XNOR gate as shown below:

$$a \times b + 1 = (a \times b)' = \text{NAND}(a, b)$$
$$a + b + 1 = (a + b)' = \text{XNOR}(a, b)$$

Through this adjustment, the resulting implementation of $F_{\text{PRESENT}}$ has a gate count of 13 which is the lowest gate count reported for the 4-bit PRESENT S-Box at the time of this writing to the best of our knowledge. Figures 4.4 and 4.5 give the implementations of $F_{\text{PRESENT}}$ before and after gate replacement respectively.

The comparisons through applications on $F_{\text{inv}}$ and $F_{\text{PRESENT}}$ showed the ability of the proposed TSA to produce results of better quality when benchmarked against existing works that utilize the LMC heuristic. Even when restricted to the use of only the logic basis (AND, XOR, NOT), the results was at least comparable as with the case on $F_{\text{PRESENT}}$. Regardless, it is emphasized that the Boyar-Peralta algorithm should theoretically be able to discover the same results (if not potentially better) as the proposed TSA given infinite number of executions. The experiments also do not invalidate the potential of better results that exist over the logic basis (AND, XOR, NOT) for both

---

3AND gates and OR gates are affine equivalent.
Chapter 4: Deterministic AND-Minimization through Reed-Muller Decomposition

\[
\begin{align*}
    t_1 &= x_3 \oplus x_4 \\
    t_2 &= x_1 \oplus x_2 \\
    t_3 &= x_2 \oplus x_3 \\
    t_4 &= x_4 \oplus x_5 \\
    t_5 &= t_1 \times t_4 \\
    t_6 &= t_5 \oplus t_1 \\
    t_7 &= t_4 \oplus t_6 \\
    t_8 &= t_3 \times t_7 \\
    t_9 &= t_8 \oplus t_6 \\
    t_{10} &= x_3 \oplus t_8 \\
    t_{11} &= t_2 \times t_9 \\
    y &= t_{11} \oplus t_{10}
\end{align*}
\]

Figure 4.6: Size-12 implementation of the majority function with \( n = 5 \).

\( F_{inv} \) and \( F_{PRESENT} \). This experiment only claims that the proposed TSA is competitive with other approaches based on the LMC heuristic in quality of results and is able derive better solutions than competing works in the two instances above.

### 4.7.2 Majority Functions

A majority function is an \( n \)-to-one transformation which is true when at least \( \frac{n}{2} \) of its \( n \) inputs are true but false otherwise. In [107], the authors reported on the difficulties associated with finding a representation that is optimal in terms of multiplicative complexity for a majority function when \( n \) is odd. Given that majority functions with \( n = 3 \) and \( n = 5 \) are lower bounded functions, it is interesting to compare the best results achievable by the proposed TSA against the best circuit reported in [107].

For the majority function with \( n = 3 \), a total of six solutions are discovered to be optimal in terms of multiplicative complexity \( c_\lambda(f) = 1 \). Each solution is a size-4 implementation consisting of one AND gate and three XOR gates. These results are in line with the quality reported in [107]. Granted, functions of low multiplicative complexity (and low number of inputs) show less variety in the possible solutions. Hence, notable differences between the two approaches are not expected.

On the other hand, the proposed algorithm discovered a total of 5760 optimal solutions\(^4\) with \( c_\lambda = 3 \) when given a majority function with \( n = 5 \). Sorting through the large set of solutions, 1440 instances of size-12 implementations are identified: 3 AND gates and 9 XOR gates. These implementations edge out the circuit reported in [107] by one XOR gate. While single-gate reduction is by no means a significant breakthrough, this comparison is another instance where the proposed TSA outperforms a different approach in LMC logic optimization. Example of a size-12 implementation of the majority-5 function is illustrated in Figure 4.6.

\(^4\) Some of the solutions are essentially the same circuits with different arrangement of literals. This is the result of different decomposition and manipulation procedures producing the same final expression for the target function.
4.8 Application: AES 8-bit S-Box

Canright’s $GF(2^4)$ inversion circuit ($F_{\text{inv}}$ in Section 4.7.1) is part of the tower field architecture for the AES S-Box in [75]. As an extension on the work to optimize the $GF(2^4)$ inversion circuit, a compact circuit for the full 8-bit S-Box for AES is presented in this section.

The S-Box is widely known to be the most expensive component of the AES cipher. The substitution comprises a multiplicative inverse over $GF(2^8)$ followed by an affine transformation [108]. The high order of the finite field makes optimization on the multiplicative inverse circuit difficult. Hence, a common approach to AES S-Box optimization is the use of composite field arithmetic (CFA) to map the inversion process to a subfield of lower order [73,75,77,109–111]. In general, the process to implement multiplicative inverse in less complex subfields through CFA can be summarized as follows:

1. Isomorphism function to map elements from the original field to a subfield.
2. Compute multiplicative inverse over the subfield.
3. Inverse isomorphism function to map results from the subfield back to the original field.

Incidentally, Boyar and Peralta [79] noted that the tower field architecture from [75] can be viewed as a three-part circuit illustrated in Figure 4.7. The three components are: (a) 8-to-22-bit top linear component $U$, (b) 22-to-18-bit middle non-linear component $M$ and (c) 18-to-8-bit bottom linear component $B$. The clear distinction between the linear and non-linear components allows the Boyar-Peralta two-step algorithm to be applied without further complications. Given the similarities in applicability, the proposed TSA benefits from the same viewpoint as well.

![Figure 4.7: The AES S-Box as a three-part circuit: top linear component $U$, middle non-linear component $M$ and bottom linear component $B$.](image)

Gate-level description of the proposed AES S-Box can be referenced in Appendix A. Table 4.3 tabulates the comparison between the proposed implementation and existing works on metrics such as circuit size and depth.
Table 4.3: Comparison of circuit complexities between the proposed AES S-Box and existing works

<table>
<thead>
<tr>
<th>Work</th>
<th>Size</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AND</td>
<td>XOR</td>
</tr>
<tr>
<td>73</td>
<td>36</td>
<td>126</td>
</tr>
<tr>
<td>75</td>
<td>36</td>
<td>91</td>
</tr>
<tr>
<td>Case I [111]</td>
<td>36</td>
<td>118</td>
</tr>
<tr>
<td>Case II [111]</td>
<td>36</td>
<td>106</td>
</tr>
<tr>
<td>Case III [111]</td>
<td>36</td>
<td>96</td>
</tr>
<tr>
<td>79</td>
<td>34</td>
<td>90</td>
</tr>
<tr>
<td>29</td>
<td>32</td>
<td>79</td>
</tr>
<tr>
<td>87</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>This Work</td>
<td>32</td>
<td>76</td>
</tr>
</tbody>
</table>

1 Information on the number of individual gates is not available.

Works from [73, 75, 111] include the most influential results on CFA-optimized AES S-Box. In comparison, it is observed that approaches based on the Boyar-Peralta heuristic in [29, 79, 87] and this work generally outperforms the aforementioned designs in both size and depth. With an efficient circuit for $F_{inv}$ generated by the proposed TSA, the proposed implementation achieves the lowest gate count for the AES S-Box.

Specifically, the proposed implementation is compared to the circuit reported in [29] to which they share the same top and bottom linear components. The improved circuit quality for the proposed $GF(2^4)$ inversion circuit translates to three XOR gates reduction to the middle non-linear component. At the same time, the circuit depth also sees a reduction of three logic gates. The gate count reduction is particularly important as a full AES implementation requires multiple instances of the S-Box for 128-bit data block encryption and key scheduling [112]. Hence, the improvement can be perceived to be several-fold of the reported values in the full AES circuit depending on the width of the datapath.

As discussed in Section 4.7.1 better circuit depth can be achieved at the cost of gate count as demonstrated by [79, 87]. The opposing designs essentially provide options to meet different design constraints: the proposed implementation is suitable for environments with hardware limitations whereas low depth variants are more attractive for high speed applications.

---

5 A standard round-based AES requires 20 instances of S-Box.
5.1 Introduction

In conjunction with optimizing logic circuits for applications in constrained environments, security and privacy concerns are also a major topic in this scope of study. To protect sensitive information against unwanted intervention by unauthorized parties, cryptographic countermeasures are necessary on exposed devices. There are a variety of cryptographic primitives designed for different functions under the main goal of ensuring information security. They can generally be categorized under: (a) block ciphers, (b) hash functions, (c) message authentication codes (MAC) and (d) stream ciphers. Chief among them is the block cipher family of primitives as they are commonly used as components for cryptographic protocols in complex computer security systems.

The most popular block cipher in modern computing is AES. It is appointed as the standard by NIST of the US government since 2001 [108]. Over the last two decades, active research on hardware optimization of the block cipher achieved significant breakthroughs in important metrics such as area, power and speed [73,75,76,110]. Our contribution to the AES S-Box is reported in Section 4.8. However, given the complexity of the primitive and increased demand on solutions for low end devices, researches have taken the opportunity to design new ciphers that are more suitable for constrained environments.

In 2017, NIST reported on the current state of lightweight cryptography and deduced a plan for the standardization of new primitives [40]. This further escalated the interests in this field of study.

This chapter reviews popular lightweight block ciphers introduced over the last decade. Specifically, focus is given on seven lightweight block ciphers with configurations that fulfill the requirements specified by NIST to be candidates for standardization. Existing works on hardware optimization for lightweight block ciphers are also studied to
determine the state-of-the-art.

5.2 Lightweight Block Ciphers

Lightweight block cipher is a subset of the block cipher family that targets applications on lower end of the device spectrum. Examples include embedded systems, RFID and sensor networks. As such, they are mostly designed with area and power concerns in mind.

An extensive (but not exhaustive) list of lightweight block ciphers can be referenced in [36]. The list covers a total of 38 block ciphers (as of this writing) and outlines important properties for each item. In this work, seven lightweight block ciphers are chosen for study based on their relevance to ultra-lightweight applications. These include mCrypton [41], PRESENT [42], Piccolo [43], LED [44], PRINCE [45], SIMON [46], and Midori [47].

5.2.1 mCrypton

The mCrypton cipher is one of the very first lightweight block cipher, introduced in 2005 by Lim and Korkishki [41]. It is derived from the CRYPTON cipher [113], one of the original candidate for AES. It operates on 64-bit block and supports options for 64/96/128-bit key. The structure of the cipher is a substitution-permutation network (SPN) involving 12 cycles of four transformations: non-linear substitution, bit permutation, state transposition and key addition. A full encryption process is illustrated in Figure 5.1.

![Figure 5.1: mCrypton encryption process.](image-url)
5.2.2 PRESENT

Arguably the most popular lightweight block cipher, the PRESENT cipher is introduced in 2007 by Bogdanov et al. [42]. The cipher supports block size of 64-bit and key size of 80/128-bit. It also adopts a SPN-based structure but differs from many other SPN ciphers by using bit-oriented permutation. 31 iterations of three transformations form the backbone of the cipher: addRoundKey, sBoxLayer and pLayer. Figure 5.2 shows the encryption process of the cipher. The PRESENT S-Box is revered for its small footprint and good cryptographic properties and is reused by other cryptographic primitives such as GOST-revisited [114], LED [44] and PHOTON [115]. The design of PRESENT cipher also inspired lightweight hash functions such as DM-PRESENT [116] and SPONGENT [117]. PRESENT cipher is one of the two lightweight block ciphers standardized under ISO/IEC 29192-2:2012 [118].

![PRESENT encryption process diagram](image)

**Figure 5.2:** PRESENT encryption process.

5.2.3 Piccolo

Shibutani et al. (from Sony Corporation) introduced the Piccolo cipher in 2011 [43]. The cipher supports the same block and key sizes as the PRESENT cipher with 64-bit block and 80/128-bit key. The design of Piccolo is based on the general Feistel network (GFN) structure. Figure 5.3 shows the GFN structure of Piccolo with three transformations: \( F \)-function, round key addition and round permutation. The \( F \)-function is a mini SPN which includes two S-Box layers separated by a diffusion matrix. The transformations are repeated for 25/31 rounds depending on the length of the key. Piccolo uses a specially designed 4-bit S-Box with very low gate count.
Figure 5.3: Piccolo encryption process. Note that each instance of round key $r_k$ is unique as generated through key scheduling.

5.2.4 LED

The LED cipher was proposed by Guo et al. in 2011 [44]. 64-bit block and 64/128-bit key are supported by the cipher. LED is an SPN cipher heavily inspired by AES with a unique property where the round key addition is performed once every four rounds. Regardless, four transformations are involved in LED at a per round basis: AddConstants, SubCells, ShiftRows and MixColumnsSerial. The S-Box used in the SubCells process is identical to the PRESENT S-Box [42]. The cipher also does not implement key scheduling to generate individual round keys.

5.2.5 PRINCE

The PRINCE cipher was introduced in [45] by Borghoff et al. in 2012 and it operates on 64-bit data blocks with support for only 128-bit key. The cipher is categorized as an SPN cipher involving four main transformations: key addition, S-Layer, M-Layer and round constant addition. One unique feature in the design of PRINCE cipher is its $\alpha$-reflection property. Essentially, the transformations involved in the second half of the encryption rounds are the inverse of the transformations in the first half. This property
5.2.6 SIMON

The SIMON cipher is designed by the American National Security Agency (NSA) and is proposed in 2013 (alongside a software-oriented version SPECK) [46]. It has a Feistel network (FN) structure and offers a wide range of options for block and key sizes. The main feature of SIMON cipher is the simplicity in the design of its Feistel function which involves only bitwise XOR, bitwise AND and bit rotations. All these operations are very area-efficient when implemented on hardware. It is also important to note that SIMON cipher does not utilize a non-linear S-Box unlike most other lightweight block ciphers.
5.2.7 Midori

Midori cipher was introduced in 2015 by Banik et al. (including some designers from Sony Corporation) [47]. Design-wise, Midori cipher uses the popular SPN structure with supports for 64/128-bit blocks and 128-bit key. The core transformations are very similar to AES-like ciphers as well: SubCell, ShuffleCell, MixColumn and KeyAdd. The cipher is advertised as an energy-efficient choice for hardware implementation. Key scheduling mechanism is also absent in its design.

![Diagram of SIMON-64/128 encryption process.](image)

**Figure 5.6:** SIMON-64/128 encryption process. The 64-bit input block is split into two 32-bit blocks $x_1$ (most significant bits) and $x_0$ (least significant bits) in the Feistel network.

![Diagram of Midori-64 encryption process.](image)

**Figure 5.7:** Midori-64 encryption process. Key additions in the middle 15 rounds include a sparse round constant per round.
5.2.8 Summary

The seven lightweight block ciphers vary significantly in designs and features. Table 5.1 tabulates some general properties of the ciphers regarding the available key-size options and the associated number of rounds needed to complete encryption/decryption of a data block.

Table 5.1: Summary of the seven chosen lightweight block ciphers.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Reference</th>
<th>Year</th>
<th>Type</th>
<th>Block size (bits)</th>
<th>Key size (bits)</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCrypton</td>
<td>[41]</td>
<td>2005</td>
<td>SPN</td>
<td>64</td>
<td>64</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>PRESENT</td>
<td>[42]</td>
<td>2007</td>
<td>SPN</td>
<td>64</td>
<td>80</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Piccolo</td>
<td>[43]</td>
<td>2011</td>
<td>GFN</td>
<td>64</td>
<td>80</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>LED</td>
<td>[44]</td>
<td>2011</td>
<td>SPN</td>
<td>64</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>PRINCE</td>
<td>[45]</td>
<td>2012</td>
<td>SPN</td>
<td>64</td>
<td>128</td>
<td>12</td>
</tr>
<tr>
<td>SIMON</td>
<td>[46]</td>
<td>2013</td>
<td>FN</td>
<td>32</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>48</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72/96</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td>42/44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96/128</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96/144</td>
<td>52/54</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td>68/69/72</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128/192/256</td>
<td></td>
</tr>
<tr>
<td>Midori</td>
<td>[47]</td>
<td>2015</td>
<td>SPN</td>
<td>64</td>
<td>128</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

As noted previously, the choice of the seven ciphers are made based on features pertaining to applications in ultra-constrained environments. Specifically, the requirements outlined by NIST in [40] are given heavy consideration.

Given the main motivation for the existence of lightweight block ciphers, attention is directed towards ciphers with hardware requirements that are substantially smaller than the current AES standard. Due to the extensive security analysis on AES as the standardized block ciphers since 2001 [119–121], it can be argued that the cipher is less likely to be vulnerable to cryptographic attacks resulting from unidentified weakness in the primitive. Hence, without suitable justification in hardware savings, AES remains attractive in most applications. For example, CLEFIA cipher [122] is a very popular lightweight primitive. However, when observing compact implementations of the cipher such as the case reported in [123], it is evident that hardware requirement of the cipher is not substantially different from a comparable implementation of AES [124].

Another obvious reasoning is the concern on security strength of a cipher. In order to be relevant, a cipher must not be proven vulnerable to cryptographic attacks. For this reason, lightweight ciphers such as GOST [114] and HIGHT [123] are not considered...
due to vulnerabilities observed in [126] and [127,128] respectively. For the same security reason, it is also obligatory to respect the minimum key length requirement of 112 bits specified by NIST [129]. Hence, ciphers such as KLEIN [130], KATAN/KTANTAN [131] and SEA [132] which do not have any configurations that fulfill the minimum key length are also excluded.

Last but not least, attention is given to specific requirements demanded by the nature of applications. Circuit area and power requirements come to mind but these metrics are heavily dependent on the platform/technology on which a cipher is implemented. Nevertheless, it was pointed out in [39] that a 50-cycle latency limit is recommended for applications in RFID (which is relevant for lightweight block ciphers). Although loop unrolling techniques can be applied on any block ciphers to meet the latency requirement, doing so normally comes at several-fold increase in hardware costs. Hence, it is preferable to have ciphers that naturally fulfill the latency requirement.

The seven lightweight block ciphers are the results of filtering the large family of lightweight primitives to satisfy the few conditions described above. To streamline the different configurations available for each cipher, all seven ciphers will be studied and optimized based on their 64-bit block with 128-bit key configurations.

### 5.3 Related Works in Hardware Optimization

This section reviews existing works on the hardware implementation of lightweight block ciphers to understand the state-of-the-art. Due to the sheer popularity of the PRESENT cipher, a majority of the optimization efforts have been focused on the cipher. Regardless, most of them - or at least the concept - can be applied on the other ciphers to achieve the same improvements.

#### 5.3.1 General Architecture-Based Optimization

Architecture-based optimization is the most prevalent form of hardware optimization observed for lightweight block ciphers. Generally, they can be grouped into three main categories as per [116,133]: (a) Round-based, (b) Serial and (c) Parallel.

1. **Round-based**: Round-based architectures are characterized by the latency requirements that are equivalent to the number of rounds in the specifications of a block cipher. The hardware is set up to compute one encryption round in one clock cycle. As such, the hardware requirement is essentially proportional to the number of transformations involved in a single encryption round of the targeted cipher. However, a small amount of additional hardware is usually required for control
logic which generally consists of a counter to coordinate the encryption process. Because round-based architectures resemble the algorithmic description of block ciphers, it is often referred to as the default architecture.

2. **Serial**: In a serial architecture, one round of encryption is spread across multiple clock cycles. This is done by performing only a fraction of the transformations required per encryption round on the data block in each cycle as characterized by the use of narrower datapaths. The width of the serialized datapath is typically limited by the size of the S-Box used in a block cipher. For example, a cipher utilizing 4-bit S-Boxes can adjust the datapath to any width divisible by four. Consequently, hardware savings are achieved at the cost of increased latency. For this reason, serialization is the most popular approach to hardware implementations of lightweight block ciphers. Most notable are the attempts on PRESENT cipher in [37,38] where the authors experimented with different degrees of serialization. Based on the hardware results reported, serial architectures are considered the current state-of-the-art in lightweight block cipher implementation.

3. **Parallel**: Parallel (a.k.a. loop unrolled) architecture is the exact opposite of a serial architecture. Through this implementation, multiple rounds of encryption are performed in one clock cycle. As such, the parallel approach trades circuit compactness for improvement in latency. In fact, for a fully unrolled architecture, full encryption of a block of data is completed in one clock cycle and control logic is usually unnecessary in this form of implementation. Nevertheless, hardware cost of parallelization is essentially several times that of their counterpart using a round-based approach. Consequently, parallelization is generally not popular for applications in constrained environments.

### 5.3.2 Memory-Based Implementation

Memory-based implementation targets devices with excess memory resources for applications outside of its primary function. Essentially, expensive circuitry such as state storage and S-Box for a block cipher can be realized using resources such as RAM blocks for area savings. Kavun and Yalcin [134] reported the benefits of this approach through implementation on the PRESENT cipher on FPGA.

### 5.4 Discussion

Hardware optimization of lightweight block ciphers at this early stage is not as well-explored as the robust optimization works done on the AES cipher. Among the popular
optimization methodologies, many of them raise concerns regarding the trade-offs incurred.

Architecture serialization may be effective in area and power reduction but the trade-off in latency can cripple the block cipher in applications where real-time response is required. To put it into perspective, a serial architecture for the PRESENT cipher in [38] has a latency of 136 cycles. This more than doubles the recommended latency threshold suggested in [39] for applications on the widely deployed EPCglobal Gen2 [135] and ISO/IEC 18000-63 [136] RFID.

On the other hand, memory-based approach offers a simple means to implement complex components in a block cipher. Circuits that are difficult to be constructed efficiently such as the S-Boxes can just be implemented as a lookup table (LUT) using available memory storage. However, considering the target applications of lightweight block ciphers, memory resources are extremely limited and are often reserved for the intended function of the device rather than for security mechanisms. NIST [40] specifically highlights microcontrollers such as the TI COP912C [137], NXP RS08 [138] and Microchip PIC10/12/16 [139] which have as little as 16 to 64 bytes of RAM.

As a summary, it is challenging to meet the stringent design goals imposed by the many fields of application for lightweight block ciphers. It is the classical task of balancing trade-offs in VLSI designs. The state-of-the-art offers opportunities for new hardware optimization methodologies to better address area and power reductions while managing trade-offs in metrics that would potentially disrupt proper operation of the target devices.
Chapter 6

Area and Power Optimization for Lightweight Block Ciphers

6.1 Introduction

In the previous chapter, a total of seven lightweight block ciphers are selected for study by virtue of their aptitudes for applications in ultra-constrained environments. Through literature review on limited existing works regarding the hardware optimization of lightweight ciphers, opportunities to contribute in this field of study were identified in the form of optimization techniques that do not compromise the latency of a cipher.

Reviewing the designs of the seven ciphers described in Section 5.2, it is observed that many of them exhibit similarities in terms of the transformations involved. Many ciphers rely on common cryptographic transformations in their encryption/decryption processes such as non-linear substitution, linear mix layer, as well as key and round constant additions. Hardware optimization techniques targeting these transformations are attractive as they can potentially benefit multiple ciphers that utilize said transformations. In this chapter, these transformations are studied extensively to identify avenues for optimization with the goal of area and/or power reductions.

6.2 Methodologies

6.2.1 Low Multiplicative Complexity S-Boxes

The non-linear substitution circuit a.k.a. S-Box is responsible for the fulfillment of Claude Shannon’s confusion property for a secure cipher. The substitution circuit is often the most expensive component of a block cipher as evident from decades of opti-

\[\text{confusion property necessitates the relationship between the secret key and ciphertext to be complex and difficult to understand.}\]
mization efforts on AES and represents the bottleneck to achieve hardware compactness. In the specifications of a lightweight block cipher, the S-Boxes are often presented as truth tables describing the non-linear transformations.

The most impactful contribution to the hardware optimization of the AES S-Box is the concept of CFA which was touched upon in Section 4.8. This approach is possible due to the understanding of the mathematical operations behind the substitution as portrayed in [108]. However, it is difficult to replicate the same approach on most contemporary lightweight S-Boxes as the mathematical descriptions of these substitutions are unknown and not directly available in their specifications.

Consequently, to construct economical circuits for non-linear substitutions in lightweight ciphers is to discover efficient implementations through logic optimization given specifications in the form of truth tables. The 4-bit S-Boxes utilized in the chosen ciphers are studied extensively to identify meaningful characteristics and properties that can be useful for optimization. Specifically, each of the four functions in an S-Box is examined. Table 6.1 tabulates some interesting findings from the study.

An important observation drawn from Table 6.1 is regarding the multiplicative complexity of the functions examined. Surprisingly, all individual functions involved in the S-Boxes of interest are lower bounded in complexity i.e. \( c_\Lambda(f) = d - 1 \). The implication of this observation is that the TSA proposed in Chapter 4 (which has shown good results on PRESENT cipher in Section 4.7.1) can be applied on all nine S-Boxes in Table 6.1 for potential low gate count implementations.

Using the truth tables provided in the specifications of the respective cipher, each S-Box is subjected to the TSA to observe the best achievable results. In the case for mCrypton and PRINCE ciphers which use multiple different S-Boxes in their design, each unique S-Box is subjected to the TSA independently as they do not share the same inputs in actual implementations (hence product sharing is not possible). The best result for PRESENT S-Box (which is also used in LED cipher) was reported in Section 4.7.1 and details regarding the best implementations for the rest of the S-Boxes are reported in Appendix B.

### 6.2.2 Circuit Sharing in Finite Field Multiplication

In combination with the confusion property, the **diffusion** property\(^2\) completes Shannon’s properties of a secure cipher. Most block ciphers achieve diffusion through simple permutation (rearranging the bits) or linear transformations. In the latter, the transformation

\[ F(x) = \alpha x + \beta \mod 2^8 \]

where \( \alpha \) and \( \beta \) are the coefficients. The diffusion property requires each output bit to be changed with the probability of one half when an input bit is flipped. See Strict Avalanche Criterion (SAC) \(^3\).

---

\(^2\) AES S-Box is equivalent to calculating the 8-bit multiplicative inverse over \(GF(2^8)\) for the input followed by a predefined affine transformation.

\(^3\) Diffusion property requires each output bit to be changed with the probability of one half when an input bit is flipped. See Strict Avalanche Criterion (SAC).
### Table 6.1: Properties of S-Boxes under study. \( f_1, f_2, \ldots, f_4 \) are the four functions of a 4-bit S-Box in ascending order of bit significance.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>S-Box</th>
<th>Function</th>
<th>Complexity ( c_{\lambda}(f) )</th>
<th>Degree ( d )</th>
<th>Negative(^1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCrypton</td>
<td>( S_0 )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>mCrypton</td>
<td>( S_1 )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>mCrypton</td>
<td>( S_2 )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>mCrypton</td>
<td>( S_3 )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>PRESENT LED</td>
<td>( S )</td>
<td>( f_1 )</td>
<td>1</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Piccolo</td>
<td>( S )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>1</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>1</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>PRINCE</td>
<td>( S )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>PRINCE</td>
<td>( S^{-1} )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Midori</td>
<td>( Sb_0 )</td>
<td>( f_1 )</td>
<td>2</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_2 )</td>
<td>1</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_3 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( f_4 )</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\(^1\) A negative function implies \( f(0) = 1 \).

Mathematical transformations are generally equivalent to finite field multiplications between the data blocks and predefined diffusion matrices.

If the elements in a diffusion matrix include only integers ‘1’s and ‘0’s, the linear transformation is just simple XORs of the relevant inputs. For example, (6.2.1) shows one such matrix \( M_{\text{Midori}} \) (used in Midori cipher) and how the multiplication between \( M_{\text{Midori}} \) and
a 4-bit input $X$ can be implemented as a linear circuit.

$$M_{Midori} = \begin{bmatrix}
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{bmatrix}$$

$$M_{Midori} \times X = \begin{bmatrix}
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{bmatrix} \times \begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix} = \begin{bmatrix}
x_2 + x_3 + x_4 \\
x_1 + x_3 + x_4 \\
x_1 + x_2 + x_4 \\
x_1 + x_2 + x_3 \\
\end{bmatrix}$$ (6.2.1)

Multiplication with a simple matrix is not uncommon in cryptography. Some ciphers such as mCrypton and PRINCE use much larger diffusion matrices instead of reiterating the same transformation on smaller chunks of a data block. Economic implementations of these circuits are fairly simple\footnote{XOR-minimization step from the Boyar-Peralta two-step algorithm (see Section 2.4.2) is a good fit for this application.} Regardless, diffusion matrices which include integers of magnitude greater than one are more complex to implement as they necessitate finite field multiplications.

$$M_{Piccolo} = \begin{bmatrix}
2 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 \\
1 & 1 & 2 & 3 \\
3 & 1 & 1 & 2 \\
\end{bmatrix}$$ (6.2.2)

Equation (6.2.2) shows the diffusion matrix $M_{Piccolo}$ used in the $F$-function of the Piccolo cipher. The linear transformation in said cipher involves multiplication between matrix $M_{Piccolo}$ and a 16-bit input (hence each element in the matrix is multiplied to a nibble of the input). Nibbles subjected to multiplications with elements ‘2’s and ‘3’s in matrix $M_{Piccolo}$ are to be evaluated as finite field multiplications over $GF(2^4)$ defined by selected irreducible polynomial $p = x^4 + x + 1$. Since $M_{Piccolo}$ is a circulant matrix and each unique element in $M_{Piccolo}$ is present in every column, derivations of $\times 2$ and $\times 3$ values of all nibbles are necessary.

On hardware, finite field multiplications are realized as logical shifts and additions using XOR gates. Further additions using the irreducible polynomial $p$ is required whenever a carry is induced. Figure 6.1 illustrates two independent circuits computing $\times 2$ and $\times 3$ respectively. Note that although the term finite field multiplication is implied, these transformations are purely linear as they do not require AND gates.
A degree of redundancy can be observed in hardware from the two circuits in Figure 6.1. Given $X$ as the 16-bit input, the $\times 2$ and $\times 3$ finite field multiplications can be expressed in polynomial form as (6.2.3) and (6.2.4) respectively.

\[
X \times 2 = [x \times X] \mod p \quad \text{(6.2.3)}
\]

\[
X \times 3 = [(x + 1) \times X] \mod p \quad \text{(6.2.4)}
\]

Through careful manipulation of the $\times 3$ expression, the two operations can be re-expressed as shown below:

\[
X \times 3 = [(x + 1) \times X] \mod p
= [(x) \times X] \mod p + X
= (X \times 2) + X \quad \text{(6.2.5)}
\]

Figure 6.2 shows the resultant circuit achieved through the relation in (6.2.5).

The same approach can be extended to more complex diffusion matrices which involves elements of higher magnitudes. For instance, the diffusion matrix for LED cipher necessitates derivations for $\{\times 2, \times 4, \times 5, \times 6, \times 8, \times 9, \times A, \times B, \times E, \times F\}^5$. Circuit sharing between the different magnitudes of finite field multiplications can provide significant

---

5Magnitudes in hexadecimal.
Figure 6.2: Single finite field multiplication circuit for $\times2 \mod p$ and $\times3 \mod p$ computations.

hardware reduction for the cipher.

6.2.3 Circuit Gating

Power requirement is a another major constraint alongside circuit area for lightweight applications. In both ASIC and FPGA designs, total power requirements are often divided into two separate measurements: static power $P_S$ and dynamic power $P_D$. In particular, CMOS circuits generally have very low static power consumption, which is a function of the supply voltage $V_{CC}$ and leakage current $I_{CC}$:

$$P_S = V_{CC} \times I_{CC}$$

On the other hand, dynamic power often contributes significantly to the total power consumption (especially at high operating frequencies) and can be attributed to two sources: transient power and capacitive-load power [142]. Looking at the formula given in (6.2.6), dynamic power $P_D$ is expressed as a function of switching activity $N_{SW}$, switched capacitance $C$, supply voltage $V_{CC}$ and operating frequency $F$ [143].

$$P_D = N_{SW} C V_{CC}^2 F$$  \hspace{1cm} (6.2.6)

The dynamic power contribution by the switching activity $N_{SW}$ is of notable interest. It is understood that most lightweight block ciphers are designed with consideration for power efficiency in every clock cycle. In this regard, an interesting deviation is observed in the round-based architecture of the PRINCE cipher.

Figure 6.3 depicts the round-based architecture for PRINCE cipher as per [45]. The functional blocks can be referenced from the cipher specification where $S$, $SR$ and $M'$ represent the non-linear substitution block, the shift rows block and the linear transformation block respectively. The exponent $-1$ indicates an inverse of the original trans-
formation. As highlighted in Section 5.2.5, the first half of the 12 encryption rounds for PRINCE cipher differ from the second half (the latter involves transformations that are the inverse of the first) to achieve $\alpha$-reflexivity. As a result, there are two distinct datapaths in the round-based circuit.

Ideally, the inactive half of the circuit should exhibit zero switching activity to minimize dynamic power as per (6.2.6). Given the original architecture in Figure 6.3 when the state in the single 64-bit register is updated in each clock cycle, logic circuits in both datapaths are triggered simultaneously due to direct connection to the register. Consequently, both datapaths are effectively “working” for the full 12 rounds of encryption.

In order to mitigate the surge in dynamic power, it is possible to add “demultiplexers” at specific locations in the circuit to prevent signal toggling in the idle half. The role of the demultiplexers is to ensure a constant input of zeros to the inactive half of the circuit. In doing so, the respective datapath should exhibit zero switching activity throughout the rounds of inactivity. This approach is referred to as circuit gating as the principle is similar to gating off specific portions of a circuit from the input (a register in this case). The proposed circuit is illustrated in Figure 6.4.

On the actual hardware, these demultiplexers are synthesized as additional combinational logic (mostly AND gates). The select lines \{c_0, c_1, c_2\} are shared with the original multiplexers hence no added control logic is required. They are sourced from the round counter with the following behaviors:

- \(c_0\) is TRUE for round 0, FALSE otherwise.
Chapter 6: Area and Power Optimization for Lightweight Block Ciphers

- \( c_1 \) is **TRUE** from round 7 to 11, **FALSE** otherwise.
- \( c_2 \) is **TRUE** from round 6 to 11, **FALSE** otherwise.

It is understood that the circuit gating approach incurs additional hardware costs to implement the demultiplexers. Therefore, the value of the proposition hinges upon the magnitude of power savings achievable weighted against the trade-off in area (which differs significantly depending on the process technology). In general, this approach should only be recommended for applicable ciphers in which the duration of inactivity is significant. While only PRINCE cipher benefits from this proposition among the chosen ciphers, similarly inspired primitive such as the MANTIS cipher [144] can potentially draw from the same concept for comparable results.

### 6.2.4 Logic Circuits for Round Constants

Round constants are round-specific fixed sequence of bits commonly used for additions on the data block or the round key in each encryption round. The purpose of adding unique round constant in each round is to reduce symmetry between the round functions which can be exploited in cryptographic attacks.

The hardware descriptions regarding the implementation of round constants are fairly different across multiple ciphers. For example, Midori cipher suggests employing two LUTs to store the round constants for encryption and decryption. However, due to the same reasoning against implementation of non-linear substitutions through LUTs, this approach is not feasible for constrained environment. To put it into perspective, round constants for Midori cipher require a total of 38 bytes of storage (30 bytes for Midori-64). As discussed in Section 5.4, this requirement immediately rules out applicability on common microcontrollers with as low as 16 bytes of memory.

PRINCE cipher [45] describes an implementation in which XOR operations with constants are reduced to inverters (NOT gates). These inverters can then be combined with the preceding XOR gates as XNOR gates which essentially removes the hardware cost to implement the additions of round constants. However, this approach is only possible in a fully unrolled architecture and as clarified in Section 5.3.1 loop unrolled architectures require several-fold increase in circuit area and power requirement in comparison to their round-based counterparts. Hence, this approach loses appeal in regards to the optimization goals.

Perhaps most interestingly is the implementation suggested in the specification for SI-MON cipher [46]. Since round constants are periodical sequences of bits that are cyclically repeated for the encryption of each new data block, linear feedback shift registers (LFSRs) can be used to generate the sequences. To do so, the appropriate feedback
function has to be understood to configure the LFSRs for correctness in the round constant produced per round. Using SIMON-64/128 for demonstration, the suggested LFSR configuration uses the companion matrix $V$ in (6.2.7). This matrix is multiplied to a 5-bit seed given as a column vector $S = [00001]^T$ per encryption round to generate the next sequence. The round constant is simply derived from the last bit in the generated sequence. The equivalent circuit is illustrated in Figure 6.5.

$$V = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.2.7}$$

Figure 6.5: LFSR configuration for the round constants in SIMON-64/128 encryption.

Alternatively, this work proposes a different approach to the generation of round constants as combinational functions of the round counter. Since the value of the round constant fundamentally depends only on the round of encryption/decryption, it makes sense to derive the value from this relationship. The premise is that an $n$-bit round counter is only sufficient to generate up to $2^n$ unique round constants of any arbitrary length. Since most ciphers generally require use of one round constant per round, this requirement is easily satisfied.

The reasoning behind the proposal is that registers, which are the fundamental building blocks for the LFSR approach, are much more expensive than combinational logic in terms of circuit area. With reference to [145], a register is approximately four times the size of an AND gate in NAND equivalent. In fact, given a 5-bit binary counter as input, the same sequence of round constants in Figure 6.5 can be obtained through a circuit of 8 AND gates, 4 OR gates and 3 XOR gates. If compared based on NAND equivalent, that is approximately 25% reduction in area. Granted, if the cost for the round counter is considered, the combinational logic approach would be much more expensive than LFSR. However, given the round counter is an essential component to coordinate the encryption/decryption process, it can be argued that it does not constitute additional

---

6Circuit synthesized using Intel Quartus Prime Version 17.1.0 software.
hardware to utilize the round counter for the generation of round constants.

### 6.2.5 Circuit Reduction for Static Round Keys

Key scheduling is an algorithm that derives multiple round keys from the master key of a cipher. Similar to the round constants, the round keys are often unique per encryption/decryption round and are absorbed through XORs into the data blocks. The purpose of the round keys is to strengthen a cipher against linear and differential cryptanalysis [146].

Nevertheless, key scheduling can be costly to implement and often is the second most expensive component in terms of area and power requirements after the non-linear substitution circuit. In fact, approximately 25% and 32% of total energy consumed in the AES cipher and PRESENT cipher respectively is contributed by the key scheduling circuit [17]. Hence, when security for a cipher can still be proven (especially against related-key attacks) without the need of key scheduling, it is often desirable to omit the algorithm for hardware savings. This is especially common in the design of lightweight block ciphers due to their nature of applications.

Lightweight ciphers that do not implement key scheduling often reuse the master key (or sections of the master key) repeatedly throughout the encryption/decryption rounds as “pseudo round keys”. Since the values of said keys remain unchanged throughout the process, they are referred to as static round keys. A good example is the Midori cipher in which the round keys alternate between the first and second halves of the 128-bit master key. The actual value of the 128-bit key does not undergo any form of transformation in a full encryption. In actual hardware, a multiplexer is placed after the key register to select the designated half of the master key for key addition. Figure 6.6 depicts the circuit where the two halves are viewed separately. Note that the two multiplexers that precede the key registers are necessary to register new master keys.

On ciphers utilizing the same approach to round keys, this work proposes the swapping of values between the key registers per clock cycle instead. This is done by feeding the output of each register to the multiplexer preceding the other register. Through this simple tweak to the circuit, the need for the multiplexer after the key registers can be eliminated. The resultant circuit is portrayed in Figure 6.7.

The elimination of the 64-bit data selector is important as it constitutes a significant area among the hardware dedicated to the round keys. At the same time, the size or complexity of the preceding multiplexers are not affected in the case of Figure 6.7. Regardless, it is anticipated that the proposed implementation will experience an increase in dynamic power due to increased switching activities in the two registers. It will be interesting to evaluate the hardware savings (and the associated reduction in static power) for various lightweight ciphers.
Figure 6.6: Round key implementation for Midori cipher with a 64-bit multiplexer.

Figure 6.7: Reduced round key implementation for Midori cipher.

power) against the added dynamic power requirement.

6.2.6 Summary

To summarize, the five proposed methodologies target specific transformations in the ciphers of interest to achieve one or more of the following goals:

- Circuit area reduction. This is achieved through a variety of methods such as logic optimization of complex transformations, circuit sharing between functions and removal of redundant or unnecessary components.

- Static power reduction. Although the proposed methodologies do not specifically target the reduction of static power, the metric can be correlated to the area requirement of a circuit [40]. Hence, improvements in this metric are concurrent with area reduction in many instances.
Dynamic power reduction. This is achieved through minimization of switching activities in a circuit and as a positive side effect of area reduction in some instances.

Table 6.2 gives a summary of the constraint problems addressed by each proposed methodology. The methodologies are denoted with alphabets (A to E) for easier reference in subsequent discussions. Because the methodologies target specific transformations in a cipher, they are not universally applicable on all seven ciphers of interest. The applicability of each methodology and the general requirement is summarized in Table 6.3.

It is also important to note that none of the proposed methodologies are mutually exclusive. Hence, it is possible for multiple applicable methodologies to be implemented simultaneously on a cipher for further benefits. Conversely, the proposed methodologies are expected to have varying effects on different applicable ciphers due to unique designs and properties even between the same cryptographic transformation. Therefore, it is necessary to evaluate each methodology on the applicable ciphers through actual synthesis to make meaningful comments on the value of each proposal.

6.3 Experimental Setup

To preface the actual evaluation of the proposed methodologies, this section outlines the experimental setup used to obtain the hardware implementation results on the ciphers. It was decided to evaluate the designs on ASIC instead of FPGA. While extensive arguments can be made for both sides, the main point of contention is the basic building block for the integrated circuits. In FPGA, the combinational logic is ultimately synthesized onto logic blocks (often referred to as logic element or slice). The fundamental components in a logic block include LUTs, multiplexers, full adder and D-type flip-flop [147]. As pointed out in [148], a 4-input LUT can represent combinational functions requiring anywhere between one to over 20 logic gates. By contrast, the same circuit would be implemented as standard logic gates on ASIC depending on availability in the technology library. Because part of the proposed methodologies heavily involve optimization on combinational logic, ASIC implementation is deemed to be the preferable approach to more accurately observe the implications of each proposal.

6.3.1 Technology

When comparing ASIC designs, attention must be given to ensure the circuits are synthesized using the same technology library. This is because evaluation metrics such as area, power and performance are technology dependent. Synthesis reports for two identical register-transfer level (RTL) designs can vary significantly when implemented using
Table 6.2: Optimization goals addressed by individual methodology.

<table>
<thead>
<tr>
<th>Denotation</th>
<th>Methodology</th>
<th>Optimization goal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Area</td>
</tr>
<tr>
<td>A</td>
<td>Low multiplicative complexity S-Boxes</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>Circuit sharing in finite field multiplication</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>Circuit gating</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Logic circuits for round constants</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>Circuit reduction for static round keys</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 6.3: Applicability of proposed methodologies on targeted lightweight block ciphers.

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Lightweight block cipher</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mCrypton</td>
<td>PRESENT</td>
</tr>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
different process technology. Such comparisons are not meaningful to comment upon the strengths and weaknesses of the subjects. In this work, the ciphers are described in VHDL and synthesized using Silterra CL180G 180nm and CL130G 130nm logic processes. Two technology libraries are used in the experiments to observe consistency of the results across different technologies.

6.3.2 Environment

ASIC implementations in this work is facilitated using Synopsys Design Compiler version I-2013.12-SP2. For power simulation, the foundry’s typical values for core voltage and temperature are used and the suggested wireload model is applied. Simulations for functional verification are performed using Synopsys VCS version J-2014.12-SP3-13.

6.3.3 Metrics

The ASIC designs are to be evaluated in three main aspects: area, power and speed. Table 6.4 provides a summary of the relevant metrics used for comparing the various designs. Some notable commentaries are as follows:

- A fixed frequency \( F_{\text{fixed}} \) is used to derive the throughput and energy metrics instead of the maximum supported frequency \( F_{\text{max}} \). In lightweight applications, the ciphers rarely utilize the maximum frequencies (typically in the range of tens to hundreds of MHz). Instead, they are often limited by the operating frequencies of the devices which are much lower than \( F_{\text{max}} \). Using an identical \( F_{\text{fixed}} \) across all designs allows fairer comparison given the nature of applications. \( F_{\text{fixed}} \) of 100 kHz is chosen as it is the most commonly used frequency for throughput calculations in the specifications of the targeted lightweight ciphers [42,44,46].

- Power and energy metrics are both included in the evaluation and their importance vary depending on applications. The power metric is more relevant in applications where the devices are expected to harvest power from their surroundings (e.g. passive RFID). On the contrary, energy consumption is a better measurement for battery-operated devices.

6.4 Evaluations of Proposed Methodologies on Applicable Ciphers

This section discusses the implications of the proposed methodologies on the applicable ciphers pertaining to the hardware area and power metrics. In addition to quantifying
Table 6.4: Summary of relevant metrics.

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric</th>
<th>Denotation</th>
<th>Unit</th>
<th>Derivation/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area</strong></td>
<td>Circuit area</td>
<td>$A$</td>
<td>$\mu m^2$</td>
<td>Compilation report from design compiler</td>
</tr>
<tr>
<td></td>
<td>Gate equivalent</td>
<td>$GE$</td>
<td>$GE$</td>
<td>$A/A_{NAND}$</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>Static power</td>
<td>$P_S$</td>
<td>$\mu W$</td>
<td>Power report from design compiler</td>
</tr>
<tr>
<td></td>
<td>Dynamic power</td>
<td>$P_D$</td>
<td>$\mu W$</td>
<td>Power report from design compiler</td>
</tr>
<tr>
<td></td>
<td>Total power</td>
<td>$P$</td>
<td>$\mu W$</td>
<td>$P_S + P_D$</td>
</tr>
<tr>
<td></td>
<td>Energy</td>
<td>$E$</td>
<td>$nJ$</td>
<td>$(P \times \text{Lat})/F_{\text{fixed}}$</td>
</tr>
<tr>
<td></td>
<td>Energy per bit</td>
<td>$E/\text{bit}$</td>
<td>$nJ/\text{bit}$</td>
<td>$E/B_{\text{size}}$</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td>Latency</td>
<td>$\text{Lat}$</td>
<td>cycle</td>
<td>Equivalent to number of encryption rounds in round-based architecture</td>
</tr>
<tr>
<td></td>
<td>Critical delay</td>
<td>$T_{\text{min}}$</td>
<td>ns</td>
<td>Timing report from design compiler</td>
</tr>
<tr>
<td></td>
<td>Maximum frequency</td>
<td>$F_{\text{max}}$</td>
<td>MHz</td>
<td>$1/T_{\text{min}}$</td>
</tr>
<tr>
<td></td>
<td>Throughput</td>
<td>$T$</td>
<td>Mbp/s</td>
<td>$(F_{\text{max}} \times B_{\text{size}})/\text{Lat}$</td>
</tr>
<tr>
<td></td>
<td>Throughput at 100kHz</td>
<td>$T^*$</td>
<td>Mbp/s</td>
<td>$(F_{\text{fixed}} \times B_{\text{size}})/\text{Lat}$</td>
</tr>
<tr>
<td></td>
<td>Throughput per GE</td>
<td>$T^*/GE$</td>
<td>Mbp/s/GE</td>
<td>$T^*/GE$</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>NAND area</td>
<td>$A_{\text{NAND}}$</td>
<td>$\mu m^2$</td>
<td>Silicon area of drive-strength-one NAND2 gate (in selected technology library)</td>
</tr>
<tr>
<td></td>
<td>Block size</td>
<td>$B_{\text{size}}$</td>
<td>bit</td>
<td>Fixed block length of cipher (from specifications)</td>
</tr>
<tr>
<td></td>
<td>Fixed frequency</td>
<td>$F_{\text{fixed}}$</td>
<td>kHz</td>
<td>Constant frequency of 100kHz</td>
</tr>
</tbody>
</table>
Chapter 6: Area and Power Optimization for Lightweight Block Ciphers

the magnitude of improvements achievable, it is equally important to identify instances where the lack of improvements are observed. Due to the intricate differences in the algorithmic design of each lightweight cipher, some of the proposed methodologies may not achieve the expected benefits on specific ciphers. Studying such instances enables a better understanding on the limitations of each proposal and allows conclusive remarks to be drawn on the recommended configurations for each cipher.

Method A: Low Multiplicative Complexity S-Boxes

Table 6.5 reports the hardware results on the applicable ciphers before and after the proposed S-Box optimization. The reference designs are implemented through behavioral modeling of the respective S-Boxes (as described in their specifications) and subjected to the logic synthesis procedure of the design compiler. For Piccolo and Midori ciphers, an additional implementation is reported for each cipher using the specific gate-level description available in their specifications. A competitive S-Box from [78] is also attempted for the PRESENT cipher.

Overall, there are noticeable improvements in area and power across all ciphers with two exceptions: Piccolo and Midori ciphers. Specifically, the lack of improvements are noted when measured against the reference S-Box designs with specific gate-level descriptions. In an effort to minimize hardware requirements, the non-linear substitution circuits for the aforementioned ciphers are designed with special care:

- Piccolo S-Box requires four NOR gates, three XOR gates and one XNOR gate [43].
- Midori S-Box requires nine NAND gates, five nor gates, two AND gates, four OR gates and one NOT gate [47].

Gate-level descriptions for the two ciphers are hardware efficient and the proposed low multiplicative complexity implementations are unable to outperform the respective designs in both area and power metrics. Nevertheless, the proposed S-Box optimization proved to be beneficial to the rest of the ciphers which are presented only as truth tables in their specifications. While the low multiplicative complexity S-Boxes generally cost only a few gate counts less than the implementations synthesized by the design compiler, the magnitudes of improvement are multiplied by severalfold due to the number of instances required (it is not uncommon to see more than 16 instances of 4-bit S-Boxes in a round-based architecture).

In Section 4.7.1 a size-13 implementation of the PRESENT S-Box was reported which requires one less XOR gate compared to the implementation in [78]. From the results in Table 6.5 it can be affirmed that the improvement translates well to ASIC implementation with reductions in both area and power.
### Table 6.5: Comparison of applicable ciphers before and after low multiplicative complexity S-Box optimization.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Optimization</th>
<th>Area ((\mu m^2))</th>
<th>Area (GE)</th>
<th>Static power ((\mu W))</th>
<th>Dynamic power ((\mu W))</th>
<th>Total power ((\mu W))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Silterra 180nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mCrypton</td>
<td>No</td>
<td>39780</td>
<td>3986</td>
<td>0.1048</td>
<td>9.8252</td>
<td>9.9300</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>36334</td>
<td>3641</td>
<td>0.1151</td>
<td>9.6567</td>
<td>9.7718</td>
</tr>
<tr>
<td></td>
<td></td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRESENT</td>
<td>No</td>
<td>23910</td>
<td>2396</td>
<td>0.0724</td>
<td>7.2574</td>
<td>7.3298</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>23431</td>
<td>2348</td>
<td>0.0678</td>
<td>7.2149</td>
<td>7.2828</td>
</tr>
<tr>
<td>Piccolo</td>
<td>No</td>
<td>36983</td>
<td>3706</td>
<td>0.0921</td>
<td>9.3750</td>
<td>9.4671</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>30616</td>
<td>3068</td>
<td>0.0843</td>
<td>8.4612</td>
<td>8.5455</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>36381</td>
<td>3646</td>
<td>0.1036</td>
<td>8.5577</td>
<td>8.6613</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>32746</td>
<td>3281</td>
<td>0.0866</td>
<td>8.8125</td>
<td>8.9991</td>
</tr>
<tr>
<td>LED</td>
<td>No</td>
<td>52377</td>
<td>5249</td>
<td>0.1760</td>
<td>11.7255</td>
<td>11.9015</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>51589</td>
<td>5170</td>
<td>0.1423</td>
<td>11.6383</td>
<td>11.7805</td>
</tr>
<tr>
<td>PRINCE</td>
<td>No</td>
<td>39346</td>
<td>3943</td>
<td>0.1067</td>
<td>9.3633</td>
<td>9.4700</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>34959</td>
<td>3503</td>
<td>0.0971</td>
<td>8.9921</td>
<td>9.0892</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No†</td>
<td>39346</td>
<td>3943</td>
<td>0.1067</td>
<td>9.3633</td>
<td>9.4700</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>38457</td>
<td>3854</td>
<td>0.1200</td>
<td>9.2820</td>
<td>9.4020</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Silterra 130nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mCrypton</td>
<td>No</td>
<td>21205</td>
<td>3154</td>
<td>2.2222</td>
<td>2.4473</td>
<td>4.6695</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>18445</td>
<td>2743</td>
<td>2.2772</td>
<td>2.3846</td>
<td>4.6618</td>
</tr>
<tr>
<td></td>
<td></td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRESENT</td>
<td>No</td>
<td>12001</td>
<td>1785</td>
<td>1.4885</td>
<td>2.1264</td>
<td>3.6149</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>11880</td>
<td>1767</td>
<td>1.4401</td>
<td>2.1138</td>
<td>3.5539</td>
</tr>
<tr>
<td>Piccolo</td>
<td>No</td>
<td>19843</td>
<td>2951</td>
<td>2.1297</td>
<td>2.4481</td>
<td>4.5778</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>15931</td>
<td>2369</td>
<td>1.8903</td>
<td>2.2480</td>
<td>4.1383</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No†</td>
<td>19843</td>
<td>2951</td>
<td>2.1297</td>
<td>2.4481</td>
<td>4.5778</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>17366</td>
<td>2583</td>
<td>2.0232</td>
<td>2.3257</td>
<td>4.3489</td>
</tr>
<tr>
<td>LED</td>
<td>No</td>
<td>19221</td>
<td>2859</td>
<td>2.1563</td>
<td>2.2726</td>
<td>4.4289</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>16704</td>
<td>2484</td>
<td>2.1066</td>
<td>2.1718</td>
<td>4.2784</td>
</tr>
<tr>
<td>PRINCE</td>
<td>No</td>
<td>29033</td>
<td>4318</td>
<td>3.5320</td>
<td>2.6151</td>
<td>6.1471</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>26984</td>
<td>4013</td>
<td>3.1490</td>
<td>2.6640</td>
<td>5.8130</td>
</tr>
<tr>
<td>Midori</td>
<td>No</td>
<td>21877</td>
<td>3254</td>
<td>2.3682</td>
<td>2.4833</td>
<td>4.8515</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>19235</td>
<td>2861</td>
<td>2.1531</td>
<td>2.3678</td>
<td>4.5209</td>
</tr>
</tbody>
</table>

* With structural S-Box in cipher specification [43].
† With structural S-Box in cipher specification [47].
Method B: Circuit Sharing in Finite Field Multiplication

Table 6.6 tabulates the hardware results on Piccolo and LED ciphers with and without optimization on the finite field multiplication circuits.

Table 6.6: Comparison of Piccolo and LED ciphers before and after circuit sharing in finite field multiplication.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Optimization</th>
<th>Area ($\mu m^2$)</th>
<th>Area (GE)</th>
<th>Static power ($\mu W$)</th>
<th>Dynamic power ($\mu W$)</th>
<th>Total power ($\mu W$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Silterra 180nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Piccolo</td>
<td>No</td>
<td>30616</td>
<td>3068</td>
<td>0.0843</td>
<td>8.4612</td>
<td>8.5455</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>27842</td>
<td><strong>2790</strong></td>
<td>0.0787</td>
<td>7.5611</td>
<td><strong>7.6398</strong></td>
</tr>
<tr>
<td>LED</td>
<td>No</td>
<td>36381</td>
<td><strong>3646</strong></td>
<td>0.1036</td>
<td>8.5577</td>
<td><strong>8.6613</strong></td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>40383</td>
<td>4047</td>
<td>0.1119</td>
<td>9.4135</td>
<td>9.5254</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Silterra 130nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Piccolo</td>
<td>No</td>
<td>15931</td>
<td>2369</td>
<td>1.8903</td>
<td>2.2480</td>
<td>4.1383</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>14383</td>
<td><strong>2139</strong></td>
<td>1.7420</td>
<td>2.0179</td>
<td><strong>3.7599</strong></td>
</tr>
<tr>
<td>LED</td>
<td>No</td>
<td>19221</td>
<td><strong>2859</strong></td>
<td>2.1563</td>
<td>2.2726</td>
<td><strong>4.4289</strong></td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>20375</td>
<td>3030</td>
<td>2.2641</td>
<td>2.4317</td>
<td>4.6958</td>
</tr>
</tbody>
</table>

Results on the Piccolo cipher show expected improvements as per the reasoning outlined in Section 6.2.2. However, the same improvements do not apply to the LED cipher as strictly worse results are observed compared to the reference design. With reference to [44], the MDS matrix $M_{LED}$ applied in the linear transformation of the cipher is as shown in (6.4.1). On first glance, the matrix is much more complicated than $M_{Piccolo}$ for the Piccolo cipher with 10 different magnitudes of multiplication. $M_{LED}$ is also not a circulant matrix and each input nibble is subjected to different circuit sharing as follows (values in hexadecimal notations):

- The first nibble (most significant) shares $\{\times 4, \times 8, \times B, \times 2\}$.
- The second nibble shares $\{\times 6, \times E, \times 2\}$.
- The third nibble shares $\{\times 2, \times 5, \times A, \times F\}$.
- The last nibble (least significant) shares $\{\times 2, \times 6, \times 9, \times B\}$.

While the proposed circuit sharing should still benefit the implementation, an interesting property of $M_{LED}$ is that it can be decomposed into four iterations of a less complex matrix $A_{LED}$ as demonstrated in (6.4.1). Through this approach, only the computations involving the last row in $A_{LED}$ require finite field multiplications. In each of the four iterations, the first three rows of $A_{LED}$ imply simple upward shifts of the input nibbles. This process can be implemented through hardware wiring (similar to bit permutations) and thus is essentially free. Consequently, this unique property of the diffusion matrix...
Chapter 6: Area and Power Optimization for Lightweight Block Ciphers

in the LED cipher leads to a more efficient hardware implementation than the proposed optimization on matrix $M_{LED}$.

$$M_{LED} = \begin{bmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 4 & 1 & 2 & 2 \end{bmatrix}^4 = A^4_{LED} \quad (6.4.1)$$

Similar to the case of the S-Boxes, the magnitude of improvement on the Piccolo cipher is amplified by the number of instances the matrix multiplication is applied per encryption round. To be precise, eight instances of the linear transformation are required per round in the FN structure of the Piccolo cipher.

**Method C: Circuit Gating**

Table 6.7 shows the effect of circuit gating on the PRINCE cipher.

**Table 6.7**: Comparison of PRINCE cipher before and after circuit gating.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Optimization</th>
<th>Area ($\mu m^2$)</th>
<th>Area (GE)</th>
<th>Static power ($\mu W$)</th>
<th>Dynamic power ($\mu W$)</th>
<th>Total power ($\mu W$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Silterra 180nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINCE</td>
<td>No</td>
<td>52377</td>
<td>5249</td>
<td>0.1760</td>
<td>11.7255</td>
<td>11.9015</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>56128</td>
<td>5624</td>
<td>0.2241</td>
<td>10.4658</td>
<td><strong>10.6899</strong></td>
</tr>
<tr>
<td>Silterra 130nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINCE</td>
<td>No</td>
<td>29033</td>
<td>4318</td>
<td>3.5320</td>
<td>2.6151</td>
<td>6.1471</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>31467</td>
<td>4680</td>
<td>3.7128</td>
<td>2.1076</td>
<td><strong>5.8204</strong></td>
</tr>
</tbody>
</table>

The implications of circuit gating are rather interesting. Immediately, the expected trade-offs between area and dynamic power are obvious: 7% increase in area for 11% reduction in power on the 180nm technology and 8% increase in area for 19% reduction in power for the 130nm technology. While the numbers make the proposed approach a compelling method for area-power balancing, the increased area cost also comes with an increased static power requirement. Implementations on the 180nm technology are less affected by the increased static power as they constitute only a small portion of the total power budget. However, the opposite is true for the 130nm technology, tallied up to only 5% reduction in total power post-optimization. Hence, circuit gating is best suited for hardware implementations in which a majority of the power budget is contributed by dynamic power. Regardless, the power reduction from the proposed approach is non-negligible and should be attractive for applications in which the power constraint is more prominent than the area restriction.
Method D: Logic Circuit for Round Constants

Table 6.8 compares the hardware results of applicable ciphers between using LFSR and combinational logic for round constant generation.

Table 6.8: Comparison of applicable ciphers using LFSR and combination logic for round constant generation.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Approach</th>
<th>Area ($\mu$m²)</th>
<th>Area (GE)</th>
<th>Static Power ($\mu$W)</th>
<th>Dynamic Power ($\mu$W)</th>
<th>Total Power ($\mu$W)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Silterra 180nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINCE</td>
<td>LFSR</td>
<td>52377</td>
<td>5249</td>
<td>0.1760</td>
<td>11.7255</td>
<td>11.9015</td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td>49523</td>
<td>4963</td>
<td>0.1626</td>
<td>10.1893</td>
<td>10.3520</td>
</tr>
<tr>
<td>SIMON</td>
<td>LFSR</td>
<td>22739</td>
<td>2279</td>
<td>0.0660</td>
<td>7.5750</td>
<td>7.6410</td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td>21628</td>
<td>2167</td>
<td>0.0623</td>
<td>6.8614</td>
<td>6.9237</td>
</tr>
<tr>
<td>Midori</td>
<td>LFSR</td>
<td>34959</td>
<td>3503</td>
<td>0.0971</td>
<td>8.9921</td>
<td>9.0892</td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td>31385</td>
<td>3145</td>
<td>0.0837</td>
<td>7.8712</td>
<td>7.9549</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Silterra 130nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINCE</td>
<td>LFSR</td>
<td>29033</td>
<td>4318</td>
<td>3.5320</td>
<td>2.6151</td>
<td>6.1471</td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td>27142</td>
<td>4037</td>
<td>3.2407</td>
<td>2.4811</td>
<td>5.7218</td>
</tr>
<tr>
<td>SIMON</td>
<td>LFSR</td>
<td>11341</td>
<td>1687</td>
<td>1.3786</td>
<td>2.2822</td>
<td>3.6608</td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td>10762</td>
<td>1601</td>
<td>1.2987</td>
<td>2.0410</td>
<td>3.3397</td>
</tr>
<tr>
<td>Midori</td>
<td>LFSR</td>
<td>19235</td>
<td>2861</td>
<td>2.1531</td>
<td>2.3678</td>
<td>4.5209</td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td>16918</td>
<td>2516</td>
<td>1.9111</td>
<td>2.0146</td>
<td>3.9258</td>
</tr>
</tbody>
</table>

Deriving round constants as a combinational function of the round counter proved to be advantageous over the LFSR approach in both area and power. The results are straightforward and consistent across the three ciphers and the different technologies used. The magnitude of improvement corresponds to the length and number of round constants required. Hence, larger reduction in area and power are observed for PRINCE and Midori ciphers which use 64-bit and 4-bit round constants respectively compared to the SIMON cipher which uses 1-bit round constants.

Method E: Circuit Reduction for Static Round Keys

Hardware results on Midori and LED ciphers before and after the proposed circuit reduction for round keys are given in Table 6.9. Note that all implementations of Midori cipher in this experiment include the round constant optimization (method D).

Commentaries on the results are best done per cipher basis. On Midori cipher, the desired reduction in circuit area is noted for the proposed designs. The values are approximately 4% on the 180nm process and 13% on the 130nm process. There is a notable discrepancy on the magnitudes of improvement between the two technology processes. This is influenced by the standard cells available in the technology libraries to implement the multiplexer. In this case, the selected 130nm technology library apparently has a hefty
Table 6.9: Comparison of Midori and LED ciphers before and after round key circuit reduction.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Optimization</th>
<th>Area ($\mu m^2$)</th>
<th>Static Power ($\mu W$)</th>
<th>Dynamic Power ($\mu W$)</th>
<th>Total Power ($\mu W$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Area (GE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Silterra 180nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Midori</td>
<td>No</td>
<td>31385</td>
<td>0.0837</td>
<td>7.8712</td>
<td>7.9549</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>30011</td>
<td>0.0924</td>
<td>8.0635</td>
<td>8.1559</td>
</tr>
<tr>
<td>LED</td>
<td>No</td>
<td>36381</td>
<td>0.1036</td>
<td>8.5577</td>
<td>8.6613</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>36745</td>
<td>0.1057</td>
<td>8.8144</td>
<td>8.9201</td>
</tr>
<tr>
<td>Silterra 130nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Midori</td>
<td>No</td>
<td>16918</td>
<td>1.9111</td>
<td>2.0146</td>
<td>3.9258</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>14762</td>
<td>1.8622</td>
<td>2.2705</td>
<td>4.1327</td>
</tr>
<tr>
<td>LED</td>
<td>No</td>
<td>19221</td>
<td>2.1563</td>
<td>2.2726</td>
<td>4.4289</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>19509</td>
<td>2.2210</td>
<td>2.3181</td>
<td>4.5391</td>
</tr>
</tbody>
</table>

area cost for multiplexer implementations hence the proposed circuit reduction produced greater hardware savings compared to the 180nm process. Trade-off in dynamic power consumption is also identified due to the increased switching activities through swapping of values between the 64-bit key registers every cycle. Regardless, the compromise is less significant than the achieved area reduction: valued at 2.5% and 5% increase in total power consumption for the 180nm and 130nm processes respectively. Therefore, depending on the standard cell library, the proposed methodology can be useful for area-power balancing in a similar fashion to the proposed method C for PRINCE cipher.

Frankly, the same implications are not reflected on the LED cipher. A unique property of the LED cipher is the performance of round key addition once per four clock cycles. By removing the 64-bit multiplexer in the manner described in Section 6.2.5, the values in the two key registers need to be held for four cycles before swapping. This results in added complexity for the remaining multiplexers as they are required to select between three input options: (a) new 128-bit key, (b) current 64-bit subkeys and (c) the alternate 64-bit subkeys. In other words, the reduction of a 64-bit multiplexer is contradicted by increased hardware demanded by the two remaining 64-bit multiplexers. Consequently, the same circuit reduction method is not recommended for LED cipher.

6.5 Proposed Implementations

The extensive evaluation process performed in the previous section enabled better understanding of the trade-offs associated with each proposed methodology. Since the methodologies are not mutually exclusive, the best implementation for each lightweight cipher is proposed by incorporating all the methodologies in this work that showed positive results in the previous evaluation. For PRINCE and Midori ciphers, two con-
Figurations are suggested for each cipher to represent an area-optimized version and a power-optimized version (due to area-power trade-offs associated with method C and E respectively). Table 6.10 summarizes the implementations proposed for each cipher.

**Table 6.10:** Proposed implementations for the seven lightweight ciphers of interest.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Configuration</th>
<th>Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCrypton</td>
<td>C1</td>
<td>✓</td>
</tr>
<tr>
<td>PRESENT</td>
<td>C1</td>
<td>✓</td>
</tr>
<tr>
<td>Piccolo</td>
<td>C1</td>
<td>✓</td>
</tr>
<tr>
<td>LED</td>
<td>C1</td>
<td>✓</td>
</tr>
<tr>
<td>PRINCE</td>
<td>C1</td>
<td>✓</td>
</tr>
<tr>
<td>SIMON</td>
<td>C1</td>
<td>✓</td>
</tr>
<tr>
<td>Midori</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.5.1 Area and Performance Results

Area and performance related metrics are tabulated in Tables 6.11 and 6.12 for the 180nm and 130nm processes respectively. Emboldened values highlight notable improvements achieved by the proposed implementations. Figure 6.8 provides a graphical representation of the results on circuit area (in GE).

### 6.5.2 Power and Energy Consumption Results

Power and energy related metrics are reported in Tables 6.13 and 6.14 for the 180nm and 130nm processes respectively. The results on total power requirement are illustrated graphically in Figure 6.9

### 6.5.3 Discussion

Collectively, the proposed implementations for all seven lightweight block ciphers showed noteworthy improvements in both area and power metrics in comparison to their reference counterparts. Circuit area sees improvements of up to 15% on the 180nm process and up to 23% on the 130nm process. On the other hand, total power requirement achieves up to 23% reduction on the 180nm process and up to 15% reduction on the 130nm process. The same percentages of improvements are reflected on the energy consumption as per the formula in Table 6.4. The achieved reductions in circuit area and power/energy requirements allow the selected ciphers to better suit their nature of applications in constrained environments.
Table 6.11: Area and performance results for implementations on Silterra 180nm process.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Configuration</th>
<th>$B_{size}$ (bit)</th>
<th>$K_{size}$ (bit)</th>
<th>Area $(\mu m^2)$</th>
<th>$T_{min}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
<th>$Lat$ (cycle)</th>
<th>$T$ (Mbps)</th>
<th>$T^*$ (kbps)</th>
<th>$T^*/Area$ (kbps/GE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCrypton</td>
<td>Ref.</td>
<td>39780</td>
<td>3986</td>
<td>39780</td>
<td>19.33</td>
<td>51.73</td>
<td>12</td>
<td>275.91</td>
<td>533.33</td>
<td>0.1338</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>36344</td>
<td>3641</td>
<td>20.04</td>
<td>49.90</td>
<td>266.13</td>
<td>31</td>
<td>770.34</td>
<td>206.45</td>
<td>0.1465</td>
</tr>
<tr>
<td>PRESENT</td>
<td>Ref.</td>
<td>27682</td>
<td>2774</td>
<td>2.68</td>
<td>373.13</td>
<td>522.66</td>
<td>266.13</td>
<td>206.45</td>
<td>0.0744</td>
<td>0.0879</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>23431</td>
<td>2348</td>
<td>3.95</td>
<td>253.16</td>
<td>206.45</td>
<td>0.0673</td>
<td>206.45</td>
<td>0.0740</td>
<td>0.0408</td>
</tr>
<tr>
<td>Piccolo</td>
<td>Ref.</td>
<td>30616</td>
<td>3068</td>
<td>7.16</td>
<td>139.66</td>
<td>531.74</td>
<td>31</td>
<td>288.34</td>
<td>133.33</td>
<td>0.0638</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>27842</td>
<td>2790</td>
<td>7.68</td>
<td>130.21</td>
<td>268.82</td>
<td>48</td>
<td>182.90</td>
<td>0.0366</td>
<td>0.0074</td>
</tr>
<tr>
<td>LED</td>
<td>Ref.</td>
<td>36381</td>
<td>3646</td>
<td>7.29</td>
<td>137.17</td>
<td>168.14</td>
<td>48</td>
<td>133.33</td>
<td>0.0408</td>
<td>0.0040</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>32602</td>
<td>3267</td>
<td>7.93</td>
<td>126.10</td>
<td>531.74</td>
<td>12</td>
<td>404.96</td>
<td>145.45</td>
<td>0.0638</td>
</tr>
<tr>
<td>PRINCE</td>
<td>C1</td>
<td>52377</td>
<td>5249</td>
<td>10.03</td>
<td>99.70</td>
<td>531.74</td>
<td>12</td>
<td>404.96</td>
<td>145.45</td>
<td>0.0101</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>52723</td>
<td>5238</td>
<td>11.63</td>
<td>85.98</td>
<td>458.58</td>
<td>44</td>
<td>1202.10</td>
<td>0.0618</td>
<td></td>
</tr>
<tr>
<td>SIMON</td>
<td>Ref.</td>
<td>22739</td>
<td>2279</td>
<td>1.21</td>
<td>826.45</td>
<td>583.33</td>
<td>48</td>
<td>555.06</td>
<td>0.0671</td>
<td>0.0061</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>21628</td>
<td>2167</td>
<td>2.63</td>
<td>380.23</td>
<td>1219.51</td>
<td>44</td>
<td>1202.10</td>
<td>0.0638</td>
<td></td>
</tr>
<tr>
<td>Midori</td>
<td>Ref.</td>
<td>34959</td>
<td>3503</td>
<td>3.28</td>
<td>304.88</td>
<td>829.88</td>
<td>16</td>
<td>829.88</td>
<td>0.1272</td>
<td>0.0734</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>31385</td>
<td>3145</td>
<td>4.82</td>
<td>207.47</td>
<td>1619.43</td>
<td>16</td>
<td>404.86</td>
<td>0.1330</td>
<td>0.0886</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>30011</td>
<td>3007</td>
<td>2.47</td>
<td>404.86</td>
<td>1619.43</td>
<td>16</td>
<td>404.86</td>
<td>0.1330</td>
<td></td>
</tr>
</tbody>
</table>
### Table 6.12: Area and performance results for implementations on Silterra 130nm process.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Configuration</th>
<th>$B_{size}$ (bit)</th>
<th>$K_{size}$ (bit)</th>
<th>Area ($\mu m^2$)</th>
<th>Area (GE)</th>
<th>$T_{min}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
<th>Lat (cycle)</th>
<th>$T$ (Mbps)</th>
<th>$T^*$ (kbps)</th>
<th>$T^*/$Area (kbps/GE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCrypton</td>
<td>Ref.</td>
<td>64</td>
<td>128</td>
<td>21205</td>
<td>3154</td>
<td>6.92</td>
<td>144.51</td>
<td>12</td>
<td>770.71</td>
<td>533.33</td>
<td>0.1691</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td>18445</td>
<td>2743</td>
<td>7.23</td>
<td>138.31</td>
<td>31</td>
<td>1764.54</td>
<td>206.45</td>
<td><strong>0.1944</strong></td>
</tr>
<tr>
<td>PRESENT</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>14712</td>
<td>2188</td>
<td>1.17</td>
<td>854.70</td>
<td>31</td>
<td>1282.31</td>
<td>206.45</td>
<td>0.0944</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td>11880</td>
<td>1767</td>
<td>1.61</td>
<td>621.12</td>
<td>737.67</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Piccolo</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>15931</td>
<td>2369</td>
<td>1.99</td>
<td>502.51</td>
<td>31</td>
<td>1037.45</td>
<td>206.45</td>
<td>0.0871</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td>14383</td>
<td>2139</td>
<td>2.04</td>
<td>490.20</td>
<td>1012.02</td>
<td></td>
<td></td>
<td><strong>0.0965</strong></td>
</tr>
<tr>
<td>LED</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>19221</td>
<td>2859</td>
<td>2.79</td>
<td>358.42</td>
<td>477.90</td>
<td>401.61</td>
<td>133.33</td>
<td>0.0466</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td>16704</td>
<td>2484</td>
<td>3.32</td>
<td>301.20</td>
<td>1075.27</td>
<td></td>
<td></td>
<td><strong>0.0537</strong></td>
</tr>
<tr>
<td>PRINCE</td>
<td>C1</td>
<td></td>
<td></td>
<td>29033</td>
<td>4318</td>
<td>4.96</td>
<td>201.61</td>
<td>1090.95</td>
<td>533.33</td>
<td>0.1422</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td>25227</td>
<td>3752</td>
<td>5.90</td>
<td>169.49</td>
<td>921.13</td>
<td></td>
<td></td>
<td>0.1312</td>
</tr>
<tr>
<td>SIMON</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>11341</td>
<td>1687</td>
<td>0.74</td>
<td>1351.35</td>
<td>1965.60</td>
<td>145.45</td>
<td>0.0862</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td>10762</td>
<td>1601</td>
<td>1.24</td>
<td>806.45</td>
<td>1173.02</td>
<td></td>
<td></td>
<td><strong>0.0909</strong></td>
</tr>
<tr>
<td>Midori</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>19235</td>
<td>2861</td>
<td>1.05</td>
<td>952.38</td>
<td>3809.52</td>
<td>400.00</td>
<td>0.1398</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td>16918</td>
<td>2516</td>
<td>1.76</td>
<td>568.18</td>
<td>2272.73</td>
<td></td>
<td></td>
<td>0.1590</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td><strong>14762</strong></td>
<td><strong>2195</strong></td>
<td>1.15</td>
<td>869.57</td>
<td><strong>3478.26</strong></td>
<td></td>
<td></td>
<td><strong>0.1822</strong></td>
</tr>
</tbody>
</table>
Figure 6.8: Circuit area for the different configurations of ciphers. (a) Silterra 180nm. (b) Silterra 130nm.
Table 6.13: Power and energy consumption results for implementations on Silterra 180nm process.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Configuration</th>
<th>$B_{size}$ (bit)</th>
<th>$K_{size}$ (bit)</th>
<th>$Lat$ (cycle)</th>
<th>Static power ($\mu W$)</th>
<th>Dynamic power ($\mu W$)</th>
<th>Total power ($\mu W$)</th>
<th>Energy ($nJ$)</th>
<th>Energy/bit ($nJ$/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCrypton</td>
<td>Ref.</td>
<td>64</td>
<td>12</td>
<td>12</td>
<td>0.1048</td>
<td>9.8252</td>
<td>9.9300</td>
<td>1.1916</td>
<td>0.0186</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>0.1151</td>
<td>9.6567</td>
<td>9.7718</td>
<td>1.1726</td>
<td>0.0183</td>
</tr>
<tr>
<td>PRESENT</td>
<td>Ref.</td>
<td>31</td>
<td></td>
<td>31</td>
<td>0.0658</td>
<td>7.6452</td>
<td>7.7110</td>
<td>2.3904</td>
<td>0.0374</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>0.0678</td>
<td>7.2149</td>
<td>7.2828</td>
<td>2.2577</td>
<td>0.0353</td>
</tr>
<tr>
<td>Piccolo</td>
<td>Ref.</td>
<td></td>
<td>31</td>
<td>31</td>
<td>0.0843</td>
<td>8.4612</td>
<td>8.5455</td>
<td>2.6491</td>
<td>0.0414</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>0.0787</td>
<td>7.5611</td>
<td>7.6398</td>
<td>2.3683</td>
<td>0.0370</td>
</tr>
<tr>
<td>LED</td>
<td>Ref.</td>
<td></td>
<td>64</td>
<td>128</td>
<td>0.1036</td>
<td>8.5577</td>
<td>8.6613</td>
<td>4.1574</td>
<td>0.0650</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>0.1054</td>
<td>8.1801</td>
<td>8.2855</td>
<td>3.9770</td>
<td>0.0621</td>
</tr>
<tr>
<td>PRINCE</td>
<td>Ref.</td>
<td>12</td>
<td></td>
<td></td>
<td>0.1760</td>
<td>11.7255</td>
<td>11.9015</td>
<td>1.4282</td>
<td>0.0223</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>0.1315</td>
<td>10.1432</td>
<td>10.2747</td>
<td>1.2330</td>
<td>0.0193</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td>0.1674</td>
<td>9.0535</td>
<td>9.2208</td>
<td>1.1065</td>
<td>0.0173</td>
</tr>
<tr>
<td>SIMON</td>
<td>Ref.</td>
<td>44</td>
<td></td>
<td></td>
<td>0.0660</td>
<td>7.3750</td>
<td>7.6410</td>
<td>3.3620</td>
<td>0.0525</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>0.0623</td>
<td>6.8614</td>
<td>6.9237</td>
<td>3.0464</td>
<td>0.0476</td>
</tr>
<tr>
<td>Midori</td>
<td>Ref.</td>
<td>16</td>
<td></td>
<td></td>
<td>0.0971</td>
<td>8.9921</td>
<td>9.0892</td>
<td>1.4543</td>
<td>0.0227</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>0.0837</td>
<td>7.8712</td>
<td>7.9549</td>
<td>1.2728</td>
<td>0.0199</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td>0.0924</td>
<td>8.0635</td>
<td>8.1559</td>
<td>1.3049</td>
<td>0.0204</td>
</tr>
</tbody>
</table>
Table 6.14: Power and energy consumption results for implementations on Siltegra 130nm process.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Configuration</th>
<th>$B_{size}$ (bit)</th>
<th>$K_{size}$ (bit)</th>
<th>$Lat$ (cycle)</th>
<th>Static power ($\mu W$)</th>
<th>Dynamic power ($\mu W$)</th>
<th>Total power ($\mu W$)</th>
<th>Energy ($nJ$)</th>
<th>Energy/bit ($nJ$/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCrypton</td>
<td>Ref.</td>
<td>64</td>
<td>12</td>
<td>12</td>
<td>2.2222</td>
<td>2.4473</td>
<td>4.6695</td>
<td>0.5603</td>
<td>0.0088</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>2.2772</td>
<td>2.3846</td>
<td>4.6618</td>
<td>0.5594</td>
<td>0.0087</td>
</tr>
<tr>
<td>PRESENT</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>31</td>
<td>1.9690</td>
<td>2.2318</td>
<td>3.7278</td>
<td>1.1556</td>
<td>0.0181</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>1.4401</td>
<td>2.1138</td>
<td>3.5539</td>
<td>1.1017</td>
<td>0.0172</td>
</tr>
<tr>
<td>Piccolo</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>31</td>
<td>1.8903</td>
<td>2.2480</td>
<td>4.1383</td>
<td>1.2829</td>
<td>0.0200</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>1.7420</td>
<td>2.0179</td>
<td>3.7599</td>
<td>1.1656</td>
<td>0.0182</td>
</tr>
<tr>
<td>LED</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>48</td>
<td>2.1563</td>
<td>2.2726</td>
<td>4.4289</td>
<td>2.1259</td>
<td>0.0332</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>2.1066</td>
<td>2.1718</td>
<td>4.2784</td>
<td>2.0536</td>
<td>0.0321</td>
</tr>
<tr>
<td>PRINCE</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>64</td>
<td>2.5320</td>
<td>2.6151</td>
<td>6.1471</td>
<td>0.7377</td>
<td>0.0115</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>2.8893</td>
<td>2.5275</td>
<td>5.4168</td>
<td>0.6500</td>
<td>0.0102</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td>2.9697</td>
<td>2.2633</td>
<td>5.2330</td>
<td>0.6280</td>
<td>0.0098</td>
</tr>
<tr>
<td>SIMON</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>44</td>
<td>1.3786</td>
<td>2.2822</td>
<td>3.6608</td>
<td>1.6108</td>
<td>0.0252</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>1.2987</td>
<td>2.0410</td>
<td>3.3397</td>
<td>1.4695</td>
<td>0.0230</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td>1.9111</td>
<td>2.0146</td>
<td>3.9258</td>
<td>0.6281</td>
<td>0.0098</td>
</tr>
<tr>
<td>Midori</td>
<td>Ref.</td>
<td></td>
<td></td>
<td>16</td>
<td>2.1531</td>
<td>2.3678</td>
<td>4.5209</td>
<td>0.7233</td>
<td>0.0113</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td>1.8822</td>
<td>2.2705</td>
<td>4.1327</td>
<td>0.6612</td>
<td>0.0103</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 6.9: Total power consumption for the different configurations of ciphers. (a) Silterra 180nm. (b) Silterra 130nm.
An important motivation behind the optimization works for the ciphers is to maintain acceptable latency in hardware implementations. As evident in Tables 6.11 and 6.12, the proposed implementations retain their round-based latency and satisfy the 50-cycle limitation by [39] for RFID applications. However, following the norm in VLSI design, trade-offs in some form are to be expected in response to the area/power savings. From the same tables, said trade-offs can be noted in the results for critical path $T_{\min}$. In most cases, the proposed methodologies result in wide circuits for the respective transformations with increased propagation delay. This is a conscious design decision made in regards to the nature of applications for the lightweight ciphers. The implications of increased $T_{\min}$ are twofold: (a) lower maximum supported frequency $F_{\text{max}}$ and (b) lower throughput $T$. A high $F_{\text{max}}$ is desirable to support a higher range of operating frequencies on target devices. However, a common characteristic of low end devices in which lightweight ciphers are employed is the relatively low operating frequencies (typically 2 MHz on EPC passive RFID tags). With a minimum reported $F_{\text{max}}$ of 49.90 MHz (for mCrypton cipher), the proposed implementations can easily satisfy the typical operating frequencies demanded from the targeted applications. In addition, with reference to [40], NIST has emphasized throughput to not be a design goal in lightweight ciphers. Regardless, trade-off in throughput should be managed carefully on specific applications. For example, a security camera capturing footage of 720×480 resolution at 30 fps has a throughput of approximately 250 Mbps (can be further reduced through video compression). In this case, a sufficient throughput for the encryption cipher is desirable for it to not be a bottleneck for the embedded system.

In summary, the proposed implementations improve the lightweight ciphers in metrics that are crucial to applications in constrained environment. The associated trade-offs in performance are also justified as to not critically impair the functionality of the ciphers. Reviewing the results at a common frequency (of 100 kHz), the proposed implementations also outperform the reference designs in efficiency metrics, i.e. throughput per unit area $T^*/GE$ and energy per bit $E/\text{bit}$.

### 6.6 Comparison with Serial Architectures

In Section 5.4, arguments were made against the state-of-the-art approach to hardware implementations of lightweight ciphers due to ramifications associated with severe latency cost. Regardless, it is undeniable that serialization is effective in reducing area and power costs. Hence, this section compares the magnitude of improvements from the proposed optimizations on round-based architectures against their serialized counterparts. The PRESENT cipher is selected as the subject to facilitate the comparisons as it is the only lightweight cipher (among the selected seven) that has seen substantial work done in
architectural optimization over the last decade. Works in [37,38,151] reported hardware implementations of PRESENT cipher with varying degree of serialization. To ensure fair comparisons, the source codes provided by the authors are synthesized using the same technology libraries under identical environment and settings.

An additional design is also attempted based on the serialization of the optimized round-based PRESENT cipher in this work using a datapath of 32-bit. The purpose of this implementation is to observe the effect of single-stage serialization on the metrics of interest without influence from other design differences that may be present in competing works. The results are summarized in Table 6.15.

The serial design from [151] showed the smallest area and power requirement among the six implementations in this experiment. However, its key generation module is intended to be calculated beforehand and described in a ROM module. Therefore, the hardware for key scheduling is not included in the source code and consequently it is difficult to comment on the true performance of the design.

Serial designs from [37] and [38] are next in line for the best results in area and power. The latter outperforms the former with less than half the latency despite the close values in area and power metrics. Compared against the best serial design in [38], the proposed round-based design is approximately 10% worse in area and 21% worse in power using the 180nm process. Similar difference in total power consumption is also observed on the 130nm process. Interestingly, circuit area showed negligible difference between the two designs based on the 130nm process. It is suspected that some of the RTL descriptions for the processes in [38] do not translate well to the available cells in the 130nm technology library used. Nevertheless, it is to be acknowledged that the serial designs, in general, achieved tangible reductions in both area and power which are desirable for lightweight applications. However, this is to be expected given the nature of architecture serialization in which only a fraction of the hardware is necessary compared to a round-based design. The point of interest, however, is in the magnitude of improvements achieved by the serial designs.

Ideally, a serial design should see reduction in circuit area proportional to the degree of serialization, i.e. a $n$-stage serialization (using $1/2^n$ bit-length of the original datapath) should require $1/2^n$ of the original hardware. In reality, most circuits involve components that cannot be serialized optimally. The design in [38] employed two-stage serialization on the PRESENT cipher. However, the hardware result is far from the ideal 75% reduction in circuit area. This can be attributed to two main obstacles concerning the serialization of block ciphers in general.

Firstly, serial architectures of block ciphers require extra control logic to coordinate the encryption/decryption rounds. This is important to ensure the correct values for unique
Table 6.15: Comparison between different architectures for PRESENT cipher.

<table>
<thead>
<tr>
<th>Work</th>
<th>Design</th>
<th>$B_{size}$ (bit)</th>
<th>$K_{size}$ (bit)</th>
<th>Area (GE)</th>
<th>Lat (cycle)</th>
<th>$T^*$ (kbps)</th>
<th>$T^*/$Area (kbps/GE)</th>
<th>Total power (μW)</th>
<th>Energy (nJ)</th>
<th>Energy/bit (nJ/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silterra 180nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37 Iterative</td>
<td></td>
<td>64</td>
<td>128</td>
<td>2921</td>
<td>55</td>
<td>116.36</td>
<td>0.0398</td>
<td>6.5747</td>
<td>3.6161</td>
<td>0.0565</td>
</tr>
<tr>
<td>37 Serial</td>
<td></td>
<td></td>
<td></td>
<td>2166</td>
<td>303</td>
<td>21.12</td>
<td>0.0098</td>
<td>6.0456</td>
<td>18.3182</td>
<td>0.2862</td>
</tr>
<tr>
<td>151 Serial¹</td>
<td></td>
<td></td>
<td></td>
<td>1319</td>
<td>132</td>
<td>48.48</td>
<td>0.0368</td>
<td>2.3255</td>
<td>3.0697</td>
<td>0.0480</td>
</tr>
<tr>
<td>38 Serial</td>
<td></td>
<td></td>
<td></td>
<td>2143</td>
<td>136</td>
<td>47.06</td>
<td>0.0220</td>
<td>6.0163</td>
<td>8.1822</td>
<td>0.1278</td>
</tr>
<tr>
<td>This work Round-based</td>
<td></td>
<td></td>
<td></td>
<td>2348</td>
<td>31</td>
<td>206.45</td>
<td>0.0879</td>
<td>7.2828</td>
<td>2.2577</td>
<td>0.0353</td>
</tr>
<tr>
<td>This work Serial</td>
<td></td>
<td></td>
<td></td>
<td>2110</td>
<td>62</td>
<td>103.23</td>
<td>0.0489</td>
<td>6.7546</td>
<td>4.1878</td>
<td>0.0654</td>
</tr>
<tr>
<td>Silterra 130nm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37 Iterative</td>
<td></td>
<td>64</td>
<td>128</td>
<td>2412</td>
<td>55</td>
<td>116.36</td>
<td>0.0483</td>
<td>3.5063</td>
<td>1.9285</td>
<td>0.0301</td>
</tr>
<tr>
<td>37 Serial</td>
<td></td>
<td></td>
<td></td>
<td>1723</td>
<td>303</td>
<td>21.12</td>
<td>0.0123</td>
<td>3.0305</td>
<td>9.1824</td>
<td>0.1435</td>
</tr>
<tr>
<td>151 Serial¹</td>
<td></td>
<td></td>
<td></td>
<td>1126</td>
<td>132</td>
<td>48.48</td>
<td>0.0431</td>
<td>1.4455</td>
<td>1.9081</td>
<td>0.0298</td>
</tr>
<tr>
<td>38 Serial</td>
<td></td>
<td></td>
<td></td>
<td>1770</td>
<td>136</td>
<td>47.06</td>
<td>0.0266</td>
<td>2.9015</td>
<td>3.9460</td>
<td>0.0617</td>
</tr>
<tr>
<td>This work Round-based</td>
<td></td>
<td></td>
<td></td>
<td>1767</td>
<td>31</td>
<td>206.45</td>
<td>0.1169</td>
<td>3.5539</td>
<td>1.1017</td>
<td>0.0172</td>
</tr>
<tr>
<td>This work Serial</td>
<td></td>
<td></td>
<td></td>
<td>1651</td>
<td>62</td>
<td>103.23</td>
<td>0.0625</td>
<td>2.9999</td>
<td>1.8599</td>
<td>0.0291</td>
</tr>
</tbody>
</table>

¹ The key generation module is not considered.
elements such as round constants or round keys are supplied at the appropriate timing. The serial design of PRESENT in [38] quadruples the number of clock cycles or latency by nature of its two-stage serialization. Hence, a larger round counter is inherently necessary to direct the transformations. Conventionally, additional multiplexers are also incurred in serial design as selectors for the different segments of data block to be transformed in each cycle. Although the design in [38] circumvents the need of additional multiplexers through the use of shift registers to naturally propagate the values of the data block every cycle, the increased switching activities has significant ramifications on the dynamic power consumed by the circuit. These circumstances culminated in additional resources that contradict the ideal scenario in serialization.

Secondly, serialization does not reduce the size of registers required for data block and key storage despite the reduction in datapath. Due to the iterative nature of block ciphers in which the data block is to be transformed over multiple iterations of encryption/decryption, it is inherently impossible to retain full information on the data block and key over the course of the process with smaller registers. At the same time, information on the location of each bit is paramount in the implementation of a block cipher due to bit permutation. This is critical to ensure the diffusion property required in a secure cipher. To retain information on the bit locations, full-width registers are absolutely necessary. With reference to the original specification of PRESENT [42], the hardware results indicated that approximately 55.1% of the area requirement is dedicated to registers for the data block and key state. Consequently, a significant percentage of the hardware for PRESENT sees no benefit from serialization.

The discussions above serve to highlight the inefficiencies associated with the serialization of block ciphers. The magnitude of improvements achieved in circuit area and power reductions is only a fraction of the trade-offs incurred in latency. Besides the obvious problem in real-time applications where fast response times are required, the ramifications of increased latency extend to the energy cost for the cipher as well. This is evident in Table 6.15 where the proposed round-based PRESENT design has an energy budget over 70% lower than the serial design in [38] to encrypt one block of data using the same operating frequency.

Granted, arguments can be made against the above comparison as the serial design in [38] can also benefit to some degree from the same optimizations performed on the proposed design. To address this issue, a single-stage serialized version of the proposed PRESENT cipher is implemented which includes the same optimizations. In this case, the area and power reductions are approximately 8% and 11% respectively when averaged using the results from both technology processes. The improvements are weighted against a twofold increase in latency and approximately 77% increase in averaged energy requirement per data block. The numbers once again suggest a very inefficient trade-off in the serialization
of PRESENT cipher.

Overall, architectural serialization remains a feasible option when area and power constraints are extremely prohibitive to the applicability of a block cipher. However, the demonstrated inefficiencies should be carefully considered in making the design decision. Research effort focused on optimizing the circuitry of different transformations in a cipher may be more attractive for hardware savings to avoid the presented trade-offs.
Conclusion and Future Works

7.1 Summary of Thesis Chapters

In this thesis, extensive studies were carried out on the LMC heuristic as a new approach to logic optimization promising further area reduction for complex circuits. The enhanced Boyar-Peralta algorithm was proposed to offer a more consistent approach to LMC logic optimization with improved average quality of results and computation time. Additionally, we proposed a novel TSA as an alternative approach to solve lower bounded minimization problems without the drawbacks of the Boyar-Peralta algorithm. This thesis then explored state-of-the-art hardware implementations of lightweight block ciphers. Area and power optimizations for seven lightweight block ciphers were proposed to overcome the resource constraints in lightweight applications.

Chapter 2 focused on introducing the concept of LMC heuristic and the original Boyar-Peralta algorithm. The LMC heuristic represented an interesting alternative to the popular Espresso heuristic due to the use of logic basis (AND, XOR, NOT) which had been shown to be beneficial in designing low gate count implementations for arithmetic and error-correcting circuits. The premise of the heuristic was to discover implementations that require the minimal number of AND gates within the logic basis (AND, XOR, NOT). To achieve this, the Boyar-Peralta algorithm emphasized a two-step approach: (a) AND-minimization step based on a randomized selection algorithm followed by (b) XOR-minimization step based on an SLP algorithm. The “two-step” nature of the algorithm provided a good framework for logic optimization using the LMC heuristic. Another highlight of the Boyar-Peralta algorithm was the execution of product sharing through the collection of product terms from solved functions for the potential to be used in the optimization of subsequent functions.

Based on the study on the Boyar-Peralta algorithm, an enhanced version of the two-step algorithm was proposed to remedy several issues with the original algorithm in Chapter 3. The primary objective of the work done was to mitigate the side effects resulting from
Chapter 7: Conclusion and Future Works

the reliance on randomness in the selection process of the AND-minimization step. At the same time, additional parameters were suggested for the algorithm to improve the quality of results in specific scenarios. By allowing only a minimal number operations per round in the AND-minimization step, the proposed algorithm was able to reduce algorithm overhead and restrict sample space expansion for improved probability. A proper solving sequence was also proposed for multiple-output problems based on the ascending order of multiplicative complexity with the goal of maximizing the potential for product sharing. In addition, linear form transformation was proposed to enable optimized non-linear circuits to benefit from the XOR-minimization step. Last but not least, an additional selection criterion using circuit depth information for tiebreaker scenario was proposed for the XOR-minimization step to allow the algorithm to differentiate circuits with lower depth from the alternatives. Executing both the enhanced algorithm and the original algorithm in the MATLAB environment revealed significant advantages for the former. Specifically, the quality of results (in terms of gate count) for the enhanced algorithm showed lower medians that were closer to the best case found as well as reduced variation. Computation time also saw noteworthy improvement due to enhancements that improve probability, resulting in less number of signal pairings required on average in the search for the target functions. Most important was the ability of the proposed algorithm to discover solutions of optimal multiplicative complexity for multiple-output problems at a much higher rate than the original approach. A practical use case of the proposed algorithm was demonstrated on the optimization of the non-linear circuit in a stochastic random number generator.

In Chapter 4, a novel deterministic algorithm was proposed as an alternative for the original AND-minimization step to derive optimal multiplicative complexity implementations for lower bounded functions. With reference to Schnorr’s lower bound rule for multiplicative complexity, an optimal implementation for an applicable function can be derived through a combination of decomposition and manipulation on its FPRM expression. To leverage this property in an algorithm to solve for low gate count circuits, a TSA was proposed to evaluate the decomposition tree of a function to identify leaves capable of producing an optimal solution. A quick selection algorithm was then performed to return the optimal solution with the lowest gate count once the TSA was completed. The new AND-minimization step was designed with heavy emphasis on product sharing, with meaningful usage of the collective product set in the decomposition process and the TSA. The key feature of the proposed algorithm was the lack of randomness in its selection process. Hence, the computation time and quality of result were consistent given the same minimization problem and it was unnecessary to evaluate a large number of solutions through multiple iterations to statistically identify the better solution (unlike the original algorithm). Experiments in the MATLAB environment verified
the expected reduction in computation time for the deterministic algorithm in comparison to the original Boyar-Peralta algorithm. Most notably, when applied on practical minimization problems such as circuits computing non-linear substitutions and majority functions, the proposed algorithm returned better results in terms of gate count than implementations reported in existing works.

A brief introduction to the family of lightweight block ciphers was done in Chapter 5 to facilitate the transition of focus to the hardware optimization on cryptographic primitives for constrained environments. Seven ciphers of interest, namely mCrypton, PRESENT, Piccolo, LED, PRINCE, SIMON and Midori ciphers, were discussed with details on their overall structures and unique design properties. A list of literature regarding existing works in hardware optimization of lightweight block ciphers was provided to recognize the state-of-the-art in this field of study. Since interests in the field of lightweight cryptography only gained momentum recently, hardware optimization techniques on these circuits were relatively unexplored. The most prevalent form of optimization observed from existing works involved varying degree of architectural serialization (a.k.a. narrowing of datapath). While effective in area and power reductions, serial architectures typically exhibited high latency that severely exceeded the recommended threshold for real-time applications (especially RFID). Another notable proposal existed in the form of memory-based implementations for computationally intensive transformations in a block cipher. However, considering the limited memory resources on devices in constrained environment, this approach was less attractive given the nature of applications.

A total of five methodologies were devised in Chapter 6 to achieve savings in area and/or power metrics for the hardware implementations of the seven ciphers of interest. The primary factor that differentiated the proposed methodologies from the state-of-the-art implementations was the preservation of round-based (default) latency for the respective cipher. Instead, the proposed methodologies targeted cryptographic transformations commonly seen in lightweight block ciphers to reduce the amount of hardware resources required to perform said transformations. They included: (a) non-linear substitution, (b) linear transformation, (c) round constant scheduling and (d) key scheduling. ASIC synthesis using Silterra's 180nm and 130nm processes provided the platform to evaluate each proposed methodology on applicable ciphers. The evaluation process revealed instances in which certain ciphers were unable to benefit from specific methodology due to unique design differences in the target transformation. Isolating these instances, recommended implementations were proposed by collectively applying all methodologies that showed positive results in the evaluation process per cipher basis. Experimental results demonstrated notable reduction in circuit area and power consumption for all proposed implementations without increased latency cost. The incurred trade-offs affected metrics related to the increased critical paths of the proposed implementations. Regardless,
they were not the main design concerns for the lightweight block ciphers as the proposed implementations were still able to support the typical operating frequencies required in the targeted environments. Although the proposed optimizations did not achieve the same magnitude of improvements in both area and power when compared against serial architectures, the hardware results were fairly close. The low latency made the proposed implementation more meaningful in practical applications that demand fast response times. The same comparison also highlighted several characteristics of a block cipher that prevented optimal serialization, namely the inability to reduce register count and the added complexity to the control logic.

7.2 Concluding Remarks

LMC heuristic is a powerful tool for multilevel logic optimization with the focus on circuit area reduction. The heuristic contributes to cost savings for applications in ultra-constrained environments where hardware resources are limited. Given an arbitrary function, the enhanced Boyar-Peralta algorithm offers a two-step approach to derive implementations at optimal multiplicative complexity which generally have low area costs as implied by the LMC heuristic. The enhanced algorithm shares the overall framework of the original Boyar-Peralta algorithm but with notable improvements to average quality of results, consistency and computation time. For the subcategory of minimization problems with lower bounded multiplicative complexity, the proposed TSA offers a more efficient alternative by analyzing decomposed FPRM expressions for optimal solutions. When applicable, this algorithm allows optimal multiplicative complexity implementations to be derived through a deterministic model and was proven in this study to outperform existing works (using the same heuristic) in terms of quality of results.

Lightweight cryptography had seen a surge in popularity over the last decade following the advent of ubiquitous computing and IoT. In particular, lightweight block ciphers became a popular field of study to provide sufficient protection for devices operating in ultra-constrained environments. Although a large number of new cryptographic primitives had been introduced to meet specific design constraints, hardware optimization of these primitives had not yet been extensively studied. State-of-the-art implementations reported often exhibited excessive latency costs as a result of serialization which can be critical in real-time applications. By focusing the optimization efforts to design compact and low power circuits for common transformations in these primitives, reduction in circuit area and power can be achieved without ramification on latency which is ideal in lightweight applications.
7.3 Directions for Future Works

In this section, potential future works related to this study are proposed. They include ideas for further improvements to logic optimization algorithms inspired by the LMC heuristic as well as research opportunities in the field of lightweight cryptography.

Several avenues exist for further research based on the established algorithms for LMC optimization. Perhaps the biggest incentive is to generalize the deterministic TSA to be able to solve non-lower bounded problems as well. An interesting approach to consider is to devise a method to split a non-lower bounded function into two or more lower bounded functions. This allows the resultant functions to be optimized as if they are part of a multiple-output problem. In the FPRM decomposition process, it is possible to treat the factored expression and the remainder as two separate functions. Alternatively, other expansion techniques such as Shannon expansion [152] and Reed-Muller expansion [67] can be used to achieve the same effect. The challenge, however, is the difficulty in guaranteeing the resulting functions to be lower bounded in multiplicative complexity. Theoretically, an optimization algorithm can be designed to explore all possible expansions but doing so adds to the already complex algorithm proposed in this study (given the multiple stages of decomposition). Simultaneously, the possibility of product sharing between the resultant functions is also critical to ensure good quality of results. As such, further studies on FPRM manipulation techniques and their relationship to multiplicative complexity can reveal useful properties that can be exploited to simplify the optimization algorithm.

In this study, the decomposition and manipulation techniques are mostly demonstrated on PPRM expressions with the clarification that the same procedure can be applied generally on all FPRM expressions without further adjustment. However, executing the optimization algorithm on both PPRM and NPRM expressions of the same problem will return different results. This provides an opportunity for further study to identify the polarity for each input literal that would result in the best quality of solutions. The circumstances become more complex when MPRM expressions are considered. Optimization of MPRM expressions is an active field of research but the efforts are mostly focused on two-level logic synthesis [62, 153, 154]. At its current state, it is inefficient for the logic optimization algorithm to consider MPRM expressions as an input literal and its negation have to be considered as two different variables, effectively doubling the number of input variables compared to FPRM. Nevertheless, given FPRM is more restrictive than MPRM, multilevel logic synthesis using the latter certainly has the potential to produce circuits of better quality.

In terms of applications, this thesis demonstrated the benefits of LMC heuristic on non-linear substitution circuits mainly in the field of cryptography. However, as evident on
Chapter 7: Conclusion and Future Works

the MCNC benchmark circuits in Chapter 3, arbitrary combinational circuits can benefit from the heuristic provided they can be decomposed into smaller functions to be solved in practical time. The area saving is especially prominent if a circuit does not have compact implementation over the logic basis (AND, OR, NOT). In particular, error detection and correction circuits can potentially benefit from the heuristic as their functions lend themselves to compact implementations using AND and XOR gates. On the other hand, it is interesting to explore potential applications on encoding and decoding circuits as they rely heavily on combinational logic to perform the desired transformations.

The issue of excessive latency observed in many contemporary implementations of lightweight block ciphers has been addressed in this thesis through the proposal of five optimization techniques on common cryptographic transformations. This study intends to inspire future optimization efforts in the same direction instead of general architectural tweaks. Currently, there are a significant number of existing lightweight block ciphers with new primitives are constantly being introduced with further improvements or different specializations. However, given the initiative by NIST to standardize lightweight cryptographic primitives, the announcement of a block cipher as the new standard for lightweight applications is foreseeable and will allow a more focused approach to hardware optimization. As a side note, cryptanalysis of said primitives will always be a valuable field of study to ensure the ciphers are capable of fulfilling their primary role.
References


References


References


References


[90] B. R. Gaines, “Stochastic computing,” in *Proceedings of the April 18- 
http://doi.acm.org/10.1145/1465482.1465505

http://doi.acm.org/10.1145/2465787.2465794

[92] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja, “An architecture for fault- 
tolerant computation with stochastic logic,” *Computers, IEEE Transactions on*, 
vol. 60, no. 1, pp. 93–105, Jan 2011.

[93] A. Alaghi, C. Li, and J. Hayes, “Stochastic circuits for real-time image- 
processing applications,” in *Design Automation Conference (DAC), 2013 50th 
ACM/EDAC/IEEE*, May 2013, pp. 1–6.

[94] P. Li and D. Lilja, “Using stochastic computing to implement digital image pro- 
cessing algorithms,” in *Computer Design (ICCD), 2011 IEEE 29th International 

ldpc codes,” *Signal Processing, IEEE Transactions on*, vol. 59, no. 11, pp. 5617– 
5626, Nov 2011.

[96] Y.-N. Chang and K. Parhi, “Architectures for digital filters using stochastic com- 
puting,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE Inter- 
national Conference on*, May 2013, pp. 2697–2701.

[97] K. Parhi and Y. Liu, “Architectures for iir digital filters using stochastic compu- 
ting,” in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 
June 2014, pp. 373–376.

puting,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE 
References


References


References


Appendices
Proposed AES S-Box

The three components $U$, $M$ and $B$ of our optimized AES S-Box are described in Figures A.1, A.2 and A.3 respectively. Note that the $GF(2^4)$ inversion circuit which computes $F_{inv}$ exists within $M$ and is represented by $M_{23}$ through $M_{37}$ in Figure A.2.

\begin{align*}
U_0 &= x_4 + x_2 \\
U_1 &= x_7 + x_1 \\
U_2 &= x_7 + x_4 \\
U_3 &= x_7 + x_2 \\
U_4 &= x_6 + x_5 \\
U_5 &= U_4 + x_0 \\
U_6 &= U_5 + x_4 \\
U_7 &= U_1 + U_0 \\
U_8 &= U_5 + x_7 \\
U_9 &= U_5 + x_1 \\
U_{10} &= U_9 + U_3 \\
U_{11} &= x_3 + U_7 \\
U_{12} &= U_{11} + x_2 \\
U_{13} &= U_{11} + x_6 \\
U_{14} &= U_{12} + x_0 \\
U_{15} &= U_{12} + U_4 \\
U_{16} &= U_{13} + U_2 \\
U_{17} &= x_0 + U_{16} \\
U_{18} &= U_{15} + U_{16} \\
U_{19} &= U_{15} + U_3 \\
U_{20} &= U_4 + U_{16} \\
U_{21} &= U_1 + U_{20} \\
U_{22} &= x_7 + U_{20}
\end{align*}

\begin{figure}[h]
\centering
\begin{tabular}{ccc}
$U_0 = x_4 + x_2$ & $U_1 = x_7 + x_1$ & $U_2 = x_7 + x_4$ \\
$U_3 = x_7 + x_2$ & $U_4 = x_6 + x_5$ & $U_5 = U_4 + x_0$ \\
$U_6 = U_5 + x_4$ & $U_7 = U_1 + U_0$ & $U_8 = U_5 + x_7$ \\
$U_9 = U_5 + x_1$ & $U_{10} = U_9 + U_3$ & $U_{11} = x_3 + U_7$ \\
$U_{12} = U_{11} + x_2$ & $U_{13} = U_{11} + x_6$ & $U_{14} = U_{12} + x_0$ \\
$U_{15} = U_{12} + U_4$ & $U_{16} = U_{13} + U_2$ & $U_{17} = x_0 + U_{16}$ \\
$U_{18} = U_{15} + U_{16}$ & $U_{19} = U_{15} + U_3$ & $U_{20} = U_4 + U_{16}$ \\
$U_{21} = U_1 + U_{20}$ & $U_{22} = x_7 + U_{20}$
\end{tabular}
\caption{Top linear component $U$ of the proposed AES S-Box. 8-bit inputs are $x_0, x_1, \ldots, x_7$. 22-bit outputs are $x_0, U_0, U_1, \ldots, U_{22}$ excluding $U_4$ and $U_{11}$.}
\end{figure}

The transformations embodied by the top and bottom linear components can be viewed as multiplications with matrices $U$ in (A.0.1) and $B$ in (A.0.2) respectively.
Appendix A: Proposed AES S-Box

<table>
<thead>
<tr>
<th>M_0 = U_7 \times U_{12}</th>
<th>M_1 = U_{10} \times U_{14}</th>
<th>M_2 = M_1 + M_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_3 = U_6 \times x_0</td>
<td>M_4 = M_3 + M_0</td>
<td>M_5 = U_1 \times U_{20}</td>
</tr>
<tr>
<td>M_6 = U_9 \times U_5</td>
<td>M_7 = M_6 + M_5</td>
<td>M_8 = U_8 \times U_{17}</td>
</tr>
<tr>
<td>M_9 = M_9 + M_5</td>
<td>M_{10} = U_2 \times U_{16}</td>
<td>M_{11} = U_0 \times U_{18}</td>
</tr>
<tr>
<td>M_{12} = M_{11} + M_{10}</td>
<td>M_{13} = U_3 \times U_{15}</td>
<td>M_{14} = M_{13} + M_{10}</td>
</tr>
<tr>
<td>M_{15} = M_2 + U_{13}</td>
<td>M_{16} = M_4 + M_{14}</td>
<td>M_{17} = M_7 + M_{12}</td>
</tr>
<tr>
<td>M_{18} = M_9 + M_{14}</td>
<td>M_{19} = M_{15} + M_{12}</td>
<td>M_{20} = M_{16} + U_{19}</td>
</tr>
<tr>
<td>M_{21} = M_{17} + U_{21}</td>
<td>M_{22} = M_{18} + U_{22}</td>
<td></td>
</tr>
</tbody>
</table>

\[ M_{23} = M_{20} \times M_{22} \]
\[ M_{24} = M_{19} + M_{20} \]
\[ M_{25} = M_{23} + M_{24} \]
\[ M_{26} = M_{21} \times M_{25} \]
\[ M_{27} = M_{21} + M_{22} \]
\[ M_{28} = M_{26} + M_{27} \]
\[ M_{29} = M_{26} + M_{23} \]
\[ M_{30} = M_{29} \times M_{27} \]
\[ M_{31} = M_{22} + M_{30} \]
\[ M_{32} = M_{27} + M_{23} \]
\[ M_{33} = M_{19} \times M_{32} \]
\[ M_{34} = M_{24} + M_{33} \]
\[ M_{35} = M_{23} + M_{34} \]
\[ M_{36} = M_{35} \times M_{24} \]
\[ M_{37} = M_{39} + M_{38} \]
\[ M_{39} = M_{37} + M_{31} \]
\[ M_{40} = M_{37} + M_{34} \]
\[ M_{41} = M_{31} + M_{28} \]
\[ M_{42} = M_{39} + M_{38} \]
\[ N_0 = M_{41} \times U_{12} \]
\[ N_1 = M_{28} \times U_{14} \]
\[ N_2 = M_{31} + x_0 \]
\[ N_3 = M_{10} \times U_{20} \]
\[ N_4 = M_{34} \times U_5 \]
\[ N_5 = M_{37} \times U_{17} \]
\[ N_6 = M_{39} \times U_{16} \]
\[ N_7 = M_{12} \times U_{18} \]
\[ N_8 = M_{38} \times U_{15} \]
\[ N_9 = M_{41} \times U_{7} \]
\[ N_{10} = M_{28} \times U_{10} \]
\[ N_{11} = M_{31} \times U_{6} \]
\[ N_{12} = M_{40} \times U_{1} \]
\[ N_{13} = M_{34} \times U_9 \]
\[ N_{14} = M_{37} \times U_8 \]
\[ N_{15} = M_{39} \times U_2 \]
\[ N_{16} = M_{42} \times U_0 \]
\[ N_{17} = M_{38} \times U_3 \]

Figure A.2: Middle non-linear component of the proposed AES S-Box. 22-bit inputs are \( x_0, U_0, U_1, ..., U_{22} \) excluding \( U_4 \) and \( U_{11} \). 18-bit outputs are \( N_0, N_1, ..., N_{17} \).

\[ B_0 = N_{15} + N_{16} \]
\[ B_1 = N_{10} + B_0 \]
\[ B_2 = N_9 + B_4 \]
\[ B_3 = N_0 + N_2 \]
\[ B_4 = N_1 + N_0 \]
\[ B_5 = N_3 + N_4 \]
\[ B_6 = N_{12} + B_3 \]
\[ B_7 = N_7 + B_5 \]
\[ B_8 = N_8 + B_6 \]
\[ B_9 = B_7 + B_8 \]
\[ B_{10} = B_5 + B_4 \]
\[ B_{11} = N_3 + N_5 \]
\[ B_{12} = N_{13} + B_0 \]
\[ B_{13} = B_3 + B_{11} \]
\[ y_4 = B_2 + B_{10} \]
\[ B_{14} = N_6 + B_7 \]
\[ B_{15} = N_{14} + B_9 \]
\[ B_{16} = B_{12} + B_{13} \]
\[ y_0 = (N_{12} + B_{16}) \]
\[ B_{17} = N_{15} + B_{14} \]
\[ B_{18} = B_1 + N_{11} \]
\[ y_7 = B_2 + B_{14} \]
\[ y_1 = (B_9 + B_{16}) \]
\[ y_3 = B_{13} + y_4 \]
\[ y_6 = (y_4 + B_{14}) \]
\[ B_{19} = B_{15} + B_{17} \]
\[ y_5 = (B_{19} + N_{17}) \]
\[ y_2 = B_{18} + B_{15} \]

Figure A.3: Bottom linear component of the proposed AES S-Box. 18-bit inputs are \( N_0, N_1, ..., N_{17} \). 8-bit outputs are \( y_0, y_1, ..., y_7 \).
Appendix A: Proposed AES S-Box

\[
U = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0
\end{bmatrix}
\]  \tag{A.0.1}

\[
B = \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0
\end{bmatrix}
\]  \tag{A.0.2}
Proposed S-Boxes for Lightweight Block Ciphers

This section details the constructions of the low multiplicative complexity S-Boxes for mCrypton, Piccolo, PRINCE and Midori ciphers derived using the algorithm proposed in Chapter 4. Each S-Box is illustrated with the respective gate-level description alongside the PPRM expressions associated with its individual outputs. Note that the S-Boxes for Piccolo and Midori ciphers are less efficient than their reference counterparts as explained in Section 6.4 but are provided regardless for disclosure.

B.1 mCrypton

B.1.1 S-Box $S_0$

PPRM expressions:

\[
\begin{align*}
y_4 & = x_1 x_2 x_3 + x_2 x_3 x_4 + x_1 x_3 + x_2 x_4 + x_1 + x_3 + x_4 \\
y_3 & = x_1 x_2 x_4 + x_1 x_3 x_4 + x_1 x_3 + x_1 x_4 + x_2 x_3 + x_2 + x_4 + 1 \\
y_2 & = x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 + x_2 x_4 + x_1 + x_2 + x_4 \\
y_1 & = x_1 x_2 x_3 + x_1 x_2 x_4 + x_2 x_3 x_4 + x_1 x_4 + x_2 x_4 + x_1 + x_2 + x_3 + x_4
\end{align*}
\]

Gate-level description:
APPENDIX B: PROPOSED S-BOXES FOR LIGHTWEIGHT BLOCK CIPHERS

B.1.2 S-Box $S_1$

PPRM expressions:

\[
\begin{align*}
y_4 &= x_1 x_2 x_3 + x_1 x_3 x_4 + x_2 x_3 x_4 + x_1 x_4 + x_2 x_4 + x_1 + x_4 \\
y_3 &= x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_1 x_3 + x_2 x_3 + x_1 + x_2 + x_3 + x_4 \\
y_2 &= x_1 x_2 x_4 + x_1 x_3 x_4 + x_1 x_3 + x_2 x_4 + x_2 + x_3 + x_4 \\
y_1 &= x_1 x_2 x_3 + x_2 x_3 x_4 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_1 + x_3 + 1
\end{align*}
\]

Gate-level description:

\[
\begin{align*}
t_1 &= x_2 + x_3 & t_2 &= x_1 + x_4 & t_3 &= x_3 + t_2 \\
t_4 &= t_3 + t_1 & t_5 &= x_2 + t_2 & t_6 &= x_4 + t_3 \\
t_7 &= x_4 \times t_2 & t_8 &= t_7 \times t_1 & t_9 &= x_1 + t_1 \\
t_{10} &= t_9 \times t_6 & y_4 &= t_{10} + t_2 & t_{11} &= x_1 + t_4 \\
t_{12} &= t_5 \times t_{11} & y_1 &= (t_{12} + t_1) & t_{13} &= x_4 + t_{12} \\
t_{14} &= t_{13} + t_{13} & t_{15} &= t_{14} \times t_1 & y_3 &= t_{15} + t_5 \\
t_{16} &= x_4 + t_8 & y_2 &= t_4 + t_{16}
\end{align*}
\]

B.1.3 S-Box $S_2$

PPRM expressions:

\[
\begin{align*}
y_4 &= x_1 x_2 x_3 + x_1 x_3 x_4 + x_2 x_3 x_4 + x_2 x_4 + x_1 + x_2 \\
y_3 &= x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 x_4 + x_1 x_2 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_3 + x_4 + 1 \\
y_2 &= x_1 x_2 x_4 + x_1 x_3 x_4 + x_1 x_2 + x_2 x_3 + x_3 x_4 + x_2 + x_3 + 1 \\
y_1 &= x_1 x_2 x_3 + x_2 x_3 x_4 + x_1 x_2 + x_1 x_4 + x_2 x_4 + x_1 + x_2 + x_3 + 1
\end{align*}
\]

Gate-level description:
Appendix B: Proposed S-Boxes for Lightweight Block Ciphers

B.1.4 S-Box $S_3$

PPRM expressions:

\[
\begin{align*}
y_4 &= x_1 x_2 x_3 + x_1 x_2 x_4 + x_2 x_3 x_4 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_3 x_4 + x_1 + 1 \\
y_3 &= x_1 x_2 x_4 + x_1 x_3 x_4 + x_1 x_2 + x_2 x_4 + x_3 + x_4 \\
y_2 &= x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_1 x_2 + x_2 x_4 + x_3 x_4 + x_1 + x_3 + x_4 + 1 \\
y_1 &= x_1 x_2 x_3 + x_2 x_3 x_4 + x_1 x_4 + x_3 x_4 + x_1 + x_2 + x_4 + 1
\end{align*}
\]

Gate-level description:

\[
\begin{align*}
t_1 &= x_1 + x_4 & t_2 &= x_3 + t_1 & t_3 &= x_2 + t_1 \\
t_4 &= t_1 \times t_3 & t_5 &= x_4 + t_4 & t_6 &= x_1 + t_4 \\
t_7 &= x_2 + x_3 & t_8 &= t_6 \times t_2 & t_9 &= x_4 + t_3 \\
t_{10} &= t_7 \times t_9 & t_{11} &= x_2 + t_8 & t_{12} &= x_1 + t_{11} \\
t_{13} &= t_5 \times t_{12} & y_1 &= (t_{11} + t_5)' & y_2 &= (t_{10} + t_2)' \\
y_3 &= t_{13} + t_2 & t_{14} &= y_3 \times t_{11} & y_4 &= (t_{14} + t_1)'
\end{align*}
\]

B.2 Piccolo

PPRM expressions:

\[
\begin{align*}
y_4 &= x_3 x_4 + x_1 + x_3 + x_4 + 1 \\
y_3 &= x_2 x_3 + x_2 + x_3 + x_4 + 1 \\
y_2 &= x_2 x_3 x_4 + x_1 x_2 + x_2 x_3 + x_2 x_4 + x_3 x_4 + x_1 + x_4 + 1 \\
y_1 &= x_1 x_2 x_3 + x_2 x_3 x_4 + x_1 x_2 + x_1 x_3 + x_1 x_4 + x_2 x_4 + x_2 + x_3 + x_4
\end{align*}
\]

Gate-level description:
Appendix B: Proposed S-Boxes for Lightweight Block Ciphers

B.3 PRINCE

B.3.1 S-Box $S$

PPRM expressions:

\[
\begin{align*}
  y_4 &= x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 + x_3 x_4 + x_2 + x_4 + 1 \\
  y_3 &= x_1 x_2 x_4 + x_2 x_3 x_4 + x_1 x_2 + x_1 x_4 + x_2 x_4 + x_1 + x_4 \\
  y_2 &= x_1 x_2 x_3 + x_2 x_3 x_4 + x_1 x_3 + x_2 x_3 + x_2 x_4 + 1 \\
  y_1 &= x_1 x_2 x_3 + x_1 x_2 + x_1 x_4 + x_2 x_3 + x_3 x_4 + x_3 + x_4 + 1
\end{align*}
\]

Gate-level description:

\[
\begin{align*}
  t_1 &= x_2 + x_4 \\
  t_2 &= x_3 + x_4 \\
  t_3 &= x_1 + x_2 \\
  t_4 &= t_1 \times t_2 \\
  t_5 &= x_4 + t_4 \\
  t_6 &= x_1 + t_1 \\
  t_7 &= t_5 \times t_6 \\
  t_8 &= x_1 + t_7 \\
  t_9 &= t_4 + t_8 \\
  t_{10} &= x_3 + t_8 \\
  t_{11} &= x_2 + t_4 \\
  t_{12} &= t_{10} \times t_{11} \\
  y_4 &= (t_7 + t_1)'^l \\
  y_3 &= t_{12} + t_9 \\
  y_2 &= t_{13} = t_{12} + t_3 \\
  t_{14} &= x_3 + t_7 \\
  t_{15} &= t_5 + t_{14} \\
  t_{16} &= t_{15} \times t_9 \\
  t_{17} &= y_3 \times t_{13} \\
  y_2 &= (t_{16} + t_5)'^l \\
  t_{18} &= x_4 + t_{17} \\
  y_1 &= (t_{10} + t_{18})'^l
\end{align*}
\]

B.3.2 Inverse S-Box $S'$

PPRM expressions:

\[
\begin{align*}
  y_4 &= x_1 x_2 x_3 + x_1 x_3 x_4 + x_2 x_3 x_4 + x_1 x_2 + x_1 x_3 + x_2 x_3 + x_3 x_4 + x_1 + x_2 + 1 \\
  y_3 &= x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_2 + x_1 x_3 + x_2 x_3 + x_2 x_4 + x_1 + x_3 \\
  y_2 &= x_1 x_2 x_3 + x_1 x_3 + x_2 x_3 + x_2 x_4 + x_3 x_4 + 1 \\
  y_1 &= x_1 x_2 x_4 + x_1 x_3 x_4 + x_1 x_2 + x_2 x_3 + x_3 x_4 + x_4 + 1
\end{align*}
\]

Gate-level description:
**Appendix B: Proposed S-Boxes for Lightweight Block Ciphers**

<table>
<thead>
<tr>
<th>(t_1)</th>
<th>(t_2)</th>
<th>(t_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1 + x_3)</td>
<td>(x_1 + x_4)</td>
<td>(x_1 + x_2)</td>
</tr>
<tr>
<td>(x_3 \times t_2)</td>
<td>(x_2 + t_4)</td>
<td>(x_3 + t_5)</td>
</tr>
<tr>
<td>(t_4 + t_1)</td>
<td>(t_7 \times t_6)</td>
<td>(t_8 + t_5)</td>
</tr>
<tr>
<td>(x_4 + t_9)</td>
<td>(t_4 + t_{10})</td>
<td>(t_{12} = t_{11} \times t_9)</td>
</tr>
<tr>
<td>(t_3 + t_3)</td>
<td>(t_3 = t_1 \times t_{10})</td>
<td>(t_{14} = t_{13} \times t_5)</td>
</tr>
<tr>
<td>((x_2 + t_{14})')</td>
<td>(t_{15} = t_{14} + t_6)</td>
<td>(t_{16} = x_2 + t_1)</td>
</tr>
<tr>
<td>((t_8 + t_{16})')</td>
<td>(t_{17} = t_{15} \times t_{16})</td>
<td>(t_{18} = t_{14} + t_{17})</td>
</tr>
</tbody>
</table>

\[y_4 = (x_1 x_2 x_4 + x_2 x_3 x_4 + x_1 x_2 + x_2 x_4 + x_3 x_4 + 1)\]
\[y_3 = (x_1 x_2 x_3 + x_1 x_2 x_4 + x_2 x_3 x_4 + x_1 x_4 + x_1 + x_4 + 1)\]
\[y_2 = (x_1 x_3 + x_1 x_4 + x_3 x_4 + x_1 + x_3)\]
\[y_1 = (x_1 x_2 x_3 + x_1 x_2 x_4 + x_2 x_3 x_4 + x_1 x_3 + x_1 x_4 + x_2)\]

**B.4 Midori**

PPRM expressions:

\[y_4 = (t_{10} + t_4)'\]
\[y_3 = (t_{10} + t_4)'\]

**Gate-level description:**

<table>
<thead>
<tr>
<th>(t_1)</th>
<th>(t_2)</th>
<th>(t_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_3 + x_4)</td>
<td>(x_1 + x_3)</td>
<td>(t_1 \times t_2)</td>
</tr>
<tr>
<td>(x_3 + t_3)</td>
<td>(x_1 + t_3)</td>
<td>(x_2 + t_1)</td>
</tr>
<tr>
<td>(t_4 \times t_5)</td>
<td>(x_2 + t_6)</td>
<td>(t_2 \times x_1)</td>
</tr>
<tr>
<td>(x_4 + t_6)</td>
<td>(t_7 + t_8)</td>
<td>(y_1 \times t_9)</td>
</tr>
<tr>
<td>((t_{10} + t_4)')</td>
<td>(t_9)</td>
<td>(t_9)</td>
</tr>
</tbody>
</table>

148
Publications Arising from this Doctoral Study


Appendix B: Proposed S-Boxes for Lightweight Block Ciphers
