Optimized Hardware Implementations of Lightweight Cryptography

by

Gangqiang Yang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2017

Abstract

Radio frequency identification (RFID) is a key technology for the Internet of Things era. One important advantage of RFID over barcodes is that line-of-sight is not required between readers and tags. Therefore, it is widely used to perform automatic and unique identification of objects in various applications, such as product tracking, supply chain management, and animal identification. Due to the vulnerabilities of wireless communication between RFID readers and tags, security and privacy issues are significant challenges. The most popular passive RFID protocol is the Electronic Product Code (EPC) standard. EPC tags have many constraints on power consumption, memory, and computing capability. The field of lightweight cryptography was created to provide secure, compact, and flexible algorithms and protocols suitable for applications where the traditional cryptographic primitives, such as AES, are impractical. In these lightweight algorithms, tradeoffs are made between security, area/power consumption, and throughput.

In this thesis, we focus on the hardware implementations and optimizations of lightweight cryptography and present the Simeck block cipher family, the WG-8 stream cipher, the Warbler pseudorandom number generator (PRNG), and the WGLCE cryptographic engine.

Simeck is a new family of lightweight block ciphers. Simeck takes advantage of the good components and design ideas of the SIMON and SPECK block ciphers and it has three instances with different block and key sizes. We provide an extensive exploration of different hardware architectures in ASICs and show that Simeck is smaller than SIMON in terms of area and power consumption.

For the WG-8 stream cipher, we explore four different approaches for the WG transformation module, where one takes advantage of constant arrays and the other three benefit from the tower field constructions of the finite field $\mathbb{F}_{2^8}$ and also efficient basis conversion matrices. The results in FPGA and ASICs show that the constant arrays based method is the best option. We also propose a hybrid design to improve the throughput with a little additional hardware.

For the Warbler PRNG, we present the first detailed and smallest hardware implementations and optimizations. The results in ASICs show that the area of Warbler with throughput of 1 bit per 5 clock cycles (1/5 bpc) is smaller than that of other PRNGs and is in fact smaller than that of most of the lightweight primitives. We also optimize and improve the throughput from 1/5 bpc to 1 bpc with a little additional area and power consumption.

Finally, we propose a cryptographic engine WGLCE for passive RFID systems. We merge the Warbler PRNG and WG-5 stream cipher together by reusing the finite state machine for both of them. Therefore, WGLCE can provide data confidentiality and generate pseudorandom numbers. After investigating the design rationales and hardware architectures, our results in ASICs show that WGLCE meets the constraints of passive RFID systems.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors Professor Mark Aagaard and Professor Guang Gong for their tremendous support, guidance, and encouragement during the past four and a half years. The continuous help and advice have solved a lot of difficulties I encountered during my research. Their extensive knowledge and expertise in the field have always inspired and motivated me for my research from both the theoretical and practical points of view. Throughout their guidance, I have gained a lot of confidence, which help me think deeply to shorten the gap of the theoretical cryptographic research, hardware implementations, and actual applications. Thanks to the positive attitude towards work and life that the professors have provided, the past four and a half years have been an unforgettable time in my entire life that has given me a tremendous development, both academically and personally.

I would also like to express my appreciation to Professor Yunsi Fei From Northeastern University for serving as my external examiner and giving me many valuable suggestions and comments. I am also grateful to my thesis committee members Professor Anwar Hasan, Professor Hiren Patel, and Professor Alfred Menezes for their valuable comments and time spent on my thesis, which help to improve the quality of my thesis a lot. This thesis would not have been possible without their guidance and assistance.

I would like to give special thanks to all my colleagues and friends during my PhD studies for their enormous support and valuable discussions. I thank Dr. Xinxin Fan for his help, guidance, and collaboration during the first two years of my study. I thank Dr. Bo Zhu and Dr. Valentin Suder for collaborating on writing a paper with me. I thank Dr. Fei Huo and Nusa Zidaric for helping me polish my English writings. I thank Dr. Kalikinkar Mandal for his help during my last term of studies. Thanks are also given to all the members from the Communication Security (ComSec) lab at University of Waterloo for their support and friendship. They are: Dr. Yin Tan, Kaveh Fazli, Dr. Teng Wu, Shasha Zhu, Yao Chen, Qiao Liu, Bo Yang, Khaled Nassar, Dr. Hayssam EI-Razouk, Ahmed Ayoub, Meng Yang, and many others.

I would like to thank ComSec seminars and the conferences that I have attended (WESS 2013, NIST lightweight workshop 2015, CHES 2015, SAC 2016), which broadened my knowledge and provided me opportunities to exchange ideas with others.

Last but not least, I would like to thank my family for their unconditional and endless love. I am indebted to my parents and my brother for their support, encouragements and sacrifices throughout my life. My deepest gratitude goes to my mother for her endless support and love. None of my work would have been possible without them.

Dedication

To my dearest father
To my loved mother
To my brother

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, the Internet of Things (IoT) has become pervasive, with many resource constrained and tiny devices deployed on a large scale and communicating wirelessly with each other and with the Internet at large. Traditional cryptographic primitives, which are designed for desktop computing, do not fit into the constraints of these tiny devices. Therefore, developing security and privacy solutions for these devices and protecting the transmitted and stored data are increasingly important.

1.1 Motivation

Among IoT devices, the Radio Frequency Identification (RFID) system is widely used to perform automatic and unique identification of objects [93]. One important advantage of RFID over barcodes is that line-of-sight is not required. Therefore, it is deployed in various applications, such as product tracking, supply chain management, and animal identification [40]. The growth of RFID has been astounding, and the number of RFID tags is expected to grow to 25 billion by 2020 just for retail apparel and shoes with more tags deployed on high value items [4].

A typical RFID system consists of three components: tags, readers and a back-end database [110, 40, 43]. Each tag is issued with an unique identification number and is attached to an object. Complex tags store information about the object, such as model and serial number, date of production, etc. The readers wirelessly communicate and track these objects via the interrogation process to the tag in order to obtain their data. After that, the readers exchange information about the object with the database through a secure channel. Depending on the power sources of RFID tags, they can be classified into three categories: active, semi-passive, and passive.

Active and semi-passive tags contain batteries. In contrast, passive tags perform computation and communication by using the energy received from the reader's RF electromagnetic signal. We focus on passive RFID systems in this thesis.

The most popular and widely adopted standard for passive RFID systems is the Electronic Product Code (EPC) Class 1 Generation 2 (EPC C1 G2) standard [2, 3], which is also included in the ISO 18000-6 standard. EPC systems operate on the ultra high frequency (UHF) band (860 MHz-960 MHz), the unique identification number is an EPC number, and the read range is around 10 meters. The passive EPC RFID tags are required to be very tiny and inexpensive, *i.e.,* about 5 to 10 cents for each tag, due to large scale deployment [110]. These properties dictate that EPC tags have inherent limited capabilities, such as very limited power consumption, constrained memory and computing capability.

Since the first version of the EPC C1 G2 standard released in 2008 [2], the security and privacy concerns have attracted a lot of attention [104, 59, 9], because there are no cryptographic mechanisms to protect the tags' data. Due to the vulnerabilities of wireless communication between readers and tags, the attacker can easily get the EPC number and access the stored data through an unauthorized reader by eavesdropping, leading to unexpected behaviour, such as malicious tracking of the object and modification of tags' stored data, etc.

Traditional cryptographic primitives which are normally well-suited for desktop computing, such as AES, are often too big and impractical due to the tags' constrained resources. Moreover, public-key cryptography is infeasible for these applications [77]. In general, the well accepted maximum area limit of the security functions for the tags are 2000 GEs (Gate Equivalents) [60, 9, 119, 100], which is around 10% of the total area of an entire tag. However, the smallest available hardware implementation of AES in CMOS 180nm Application Specific Integrated Circuit (ASIC) requires 2400 GEs [82]. In order to overcome this challenge, lightweight cryptography is devised to provide secure, compact, and flexible algorithms and protocols that fit into the constraints of resource constrained devices. The term lightweight is used broadly to mean that an algorithm is suitable for use on a constrained platform. In these lightweight algorithms, tradeoffs are made between security, area/power consumption, and throughput. Generally, a lower security level than AES is often sufficiently enough, because the amount of encrypted data is tiny during the device's lifetime [12]. There are currently no well accepted power consumption and throughput requirements. The power consumption depends on multiple factors, such as activity factor, clock speed, operating voltage, and the adopted CMOS technology. Thus, it is hard to give an upper bound on the allowed power or energy, but it should be kept as small as possible. The minimum throughput requirement relies on the specific applications.

In recent years, a lot of lightweight algorithms were proposed, such as the stream ciphers Trivium [27], Grain [55] and lightweight WG (Welch-Gong) stream ciphers (WG-5 [7], WG-7 [71], WG-8 [35]); block ciphers TEA [112], XTEA [86], HIGHT [56], SEA [105], PRESENT [17], KATAN and KTANTAN [26], CLEFIA [103], LED [49], PRINCE [18], EPCBC [117], K-LEIN [47], LBlock [113], Piccolo [102], Twine [106], SIMON and SPECK [11] and QTL [68]. In particular, CLEFIA, PRESENT, and Trivium have been adopted by the ISO/IEC Standard 29192. PRESENT-80 and Grain-128 have been adopted by ISO/IEC Standard 29167, which provides the cipher suite for the RFID air interfaces. This cipher suite has been specified in the second version of the EPC C1 G2 standard released in 2013 [3], which includes a security extension framework, such as encryption, authentication, etc. Recently, National Institute of Standards and Technology (NIST) began an effort to standardize lightweight cryptography [5].

According to the aforementioned area, power consumption and throughput requirements, the highly optimized hardware implementations of lightweight cryptography are important for constrained applications. Different cryptographic primitives, such as block ciphers, stream ciphers, and pseudorandom number generators (PRNGs) have different parameters and structures, which affect the hardware implementations and optimizations. Moreover, the hardware performance of the primitive will also influence the parameters and structure selections. As a result, significant emphasis has been given to the hardware implementations in cryptography competition projects, such as stream ciphers in eSTREAM [34], lightweight cryptography in NIST [5], and authenticated encryption candidates in CAESAR [20]. To support multiple security needs, implementing multiple cryptography primitives on a constrained device is non-trivial. For example, integrating a PRNG and an encryption algorithm to EPC tags with minimal cost is challenging.

The main research presented in this thesis is to explore efficient low area/power consumption hardware implementations and optimizations of lightweight cryptography, including lightweight stream cipher, lightweight block cipher, lightweight PRNG, and a lightweight cryptographic engine with multiple security functions. With evaluations of the hardware architectures, the impact on selecting appropriate design parameters for a smaller cipher design is also investigated.

1.2 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 provides some background and related work. The mathematical background related to finite field, feedback shift register sequences, and the WG transform are firstly reviewed. Then, the related work of lightweight cryptography is provided, and lightweight block ciphers SIMON and SPECK, lightweight stream ciphers Trivium, Grain, and WG are described in detail. We give several efficient hardware implementation techniques, including exploitation of parallelism, clock gating, component reuse and optimizations achieved by choosing appropriate subfields and different bases representations. Finally, we present the description of the EPC C1 G2 passive RFID system. Additional related work appears in each chapter.

Chapter 3 proposes Simeck, a new family of lightweight block ciphers, which is very suitable for resource constrained devices. Simeck takes advantage of the good components and design ideas of SIMON and SPECK, and it targets low area/power consumption implementations while still keeping a reasonable throughput and maximum frequency. We provide an extensive exploration of different hardware architectures in order to make a balance between area, throughput, and power consumption for SIMON and Simeck in both CMOS 65nm and CMOS 130nm ASICs. We show that it is possible to design a smaller cipher than SIMON in terms of area and power consumption.

Chapter 4 presents the design space exploration of the lightweight stream cipher WG-8 for low-cost FPGA and CMOS 65nm and CMOS 130nm ASICs. We explore four different approaches for the WG transformation module, where one takes advantage of the constant arrays and the other three benefits from the tower constructions of finite field $\mathbb{F}_{2^8}$ and also efficient basis conversion matrices. Many design options have been explored to make trade-offs in terms of area, power consumption as well as throughput. Consequently, the results for FPGA and ASICs are given, and comparisons with other lightweight stream ciphers are provided.

Chapter 5 evaluates hardware implementations and optimizations of Warbler PRNG in C-MOS 65nm and CMOS 130nm ASICs. We propose an architecture and a standard interface for the implementations of Warbler with throughput of 1 bit per 5 clock cycles (1/5 bpc). More importantly, we also improve the throughput from 1/5 bpc to 1 bpc with a little additional area and power consumption. In addition, the LFSR counter-based design is better than the binary counter-based design in terms of smaller area and lower total power consumption. The comparisons with other PRNGs and lightweight primitives are also given.

Chapter 6 proposes a lightweight cryptographic engine WGLCE for passive RFID systems. WGLCE is a fusion of the Warbler PRNG and the lightweight stream cipher WG-5, which can be easily integrated into the RFID systems. We investigate the rationale and design choices for WGLCE and explore its different hardware architectures and implementations in CMOS 65nm and CMOS 130nm ASICs. Moreover, we provide an interface for its usage. Finally, we compare our results with other lightweight cyptographic engines.

Chapter 7 concludes this thesis and discusses the future potential research directions.

1.3 Contributions

The main contributions of this thesis are summarized as follows:

1. The Simeck family of lightweight block ciphers.
    - The smallest block cipher family with key schedule.
    - Improved SIMON's round function by choosing new shift numbers.
    - Reduced size of Simon's key schedule by choosing a new LFSR polynomial.
    - First published detailed implementation of fully-serialized Feistel architecture.
2. The lightweight WG-8 stream cipher.
    - A hybrid design architecture for providing parallelism from 1 bit per clock cycle (bpc) to 11 bits per clock cycle.
    - Hardware implementation and analysis of constant arrays and three tower field based methods.
3. The lightweight Warbler PRNG.
    - Detailed and smallest hardware implementations of Warbler.
    - Throughput improvement from 1 bit per five clock cycles to 1 bit per clock cycle.
4. The lightweight WGLCE cryptographic engine.
    - A cryptographic engine which merges WG-5 and Warbler to provide two functionalities: data confidentiality and generating pseudorandom numbers.
    - Hardware implementation analysis of different architectures.
    - Loading pattern that simplifies the work of the external environment.
5. Overall lessons.
    - Analysis of effects on area and power consumption by use of clock gating and binary vs LFSR counters.

Chapter 2

Background

In this chapter, we first recall some mathematical background in Section 2.1. Then, Section 2.2 presents the description of symmetric key cryptography. Section 2.3 reviews a selection of related lightweight symmetric ciphers. In addition, several typical hardware optimization techniques are discussed in Section 2.4. Finally, Section 2.5 describes the system and security issues for a typical passive RFID system.

2.1 Mathematical Background

In this section, we review some mathematical background on finite fields, sequences, and the WG transformation. For details, the readers are referred to [45, 44, 52, 80].

2.1.1 Finite Field

Let $\mathbb{F}_2$ be the finite field with two elements 0 and 1, and $\mathbb{F}_{2^t}$ be an extension field over the finite field $\mathbb{F}_2$ with 2^t elements, defined by an irreducible polynomial.

A polynomial $f(x)$ is *irreducible* over $\mathbb{F}$ if $f(x)$ is only divisible by c or by $cf(x)$, where $c \in \mathbb{F}$. An *irreducible* polynomial $f(x)$ of degree t over $\mathbb{F}_2$ is *primitive polynomial* if $f(x) \mid (x^{2^t-1} - 1)$ but $f(x) \nmid (x^r - 1)$ when $r < 2^t - 1$.

Polynomial and Normal Bases

The basis is used to represent the elements in finite field.

Definition 1 *Let α be a defining element of $\mathbb{F}_{2^t}$, which is a root of an irreducible polynomial $p(x)$, i.e., $p(\alpha) = 0$, where $p(x)$ is an irreducible polynomial of degree t over $\mathbb{F}_2$. Then, the polynomial basis of $\mathbb{F}_{2^t}$ is given by $\{1,\ \alpha,\ \alpha^2,\ \cdots,\ \alpha^{t-1}\}$. Thus, any element in $\mathbb{F}_{2^t}$ can be represented as A = $a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{t-1}\alpha^{t-1}, a_i \in \mathbb{F}_2$.*

Definition 2 *Let $\beta \in \mathbb{F}_{2^t}$, and if the t elements of the set $\{\beta, \beta^2, \beta^{2^2}, \cdots, \beta^{2^{t-1}}\}$ are linearly independent over $\mathbb{F}_{2^t}$, then we call this set as a normal basis of $\mathbb{F}_{2^t}$ over $\mathbb{F}_2$. Thus, every element in F_{2^t} can be written as $A = a_0\beta + a_1\beta^2 + a_2\beta^{2^2} + \cdots + a_{t-1}\beta^{2^{t-1}}, a_i \in \mathbb{F}_2$.*

Change of Bases

For every element $A \in \mathbb{F}_{2^t}$, it can be represented using any basis. We choose two bases $\Psi_1 = \{\alpha_0, \alpha_1, \cdots, \alpha_{t-1}\}$ and $\Psi_2 = \{\beta_0, \beta_1, \cdots, \beta_{t-1}\}$ of $\mathbb{F}_{2^t}$. Assume $\{a_i\} \in \mathbb{F}_2$ and $\{b_i\} \in \mathbb{F}_2$ are coordinates of A with respect to Ψ_1 and Ψ_2 respectively. Let $\mathbf{a} = (a_0, a_1, \cdots, a_{t-1})$ and $\mathbf{b} = (b_0, b_1, \cdots, b_{t-1})$, and assume $\Psi_2^T = \mathbf{C} \cdot \Psi_1^T$, then,

$$\begin{array}{rclclcl} A & = & \sum_0^{t-1} a_i\alpha_i & = & \mathbf{a} \cdot \Psi_1^T, & & \\ & = & \sum_0^{t-1} b_i\beta_i & = & \mathbf{b} \cdot \Psi_2^T & = & \mathbf{b} \cdot \mathbf{C} \cdot \Psi_1^T. \end{array}$$

Thus, $\mathbf{a} = \mathbf{b} \cdot \mathbf{C}$, where $\mathbf{C}$ is defined as the conversion matrix.

Subfield

Definition 3 *Let $\mathbb{F}_{2^n}$ be a finite field with $q = 2^n$ elements, and $\mathbb{F}_{2^n}$ contains a subfield $\mathbb{F}_{2^m}$ if and only if m is a positive divisor of n. An element $\alpha \in \mathbb{F}_{2^n}$ is in the subfield $\alpha \in \mathbb{F}_{2^m}$ if and only if $\alpha^{2^m} = \alpha$.*

Let $n = m \cdot k$. Then we may consider $\mathbb{F}_{2^n}$ as an extension field of $\mathbb{F}_{2^m}$. Thus, an element of $\mathbb{F}_{2^n}$ (*i.e.*, $A \in \mathbb{F}_{2^n}$) can be expressed as a linear combination of elements in $\mathbb{F}_{2^m}$ based on a basis with *k* elements. Assume $\{\gamma_0, \gamma_1, \cdots, \gamma_{k-1}\}$ is a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_{2^m}$, then we have $A = \sum_0^{k-1} a_i\gamma_i, a_i \in \mathbb{F}_{2^m}$.

Trace

Definition 4 *Let $x \in \mathbb{F}_{2^t}$, and the trace function for x from $\mathbb{F}_{2^t}$ to $\mathbb{F}_2$ is defined by: $Trace(x) = x + x^2 + x^{2^2} + \cdots + x^{2^{t-1}}$. For $\alpha \in \mathbb{F}_{2^t}$, we simply denote it as $Tr(\alpha)$.*

Theorem 1 *The trace function satisfies the following properties.*

i. $Tr(x+y) = Tr(x) + Tr(y)$ *for all* $x, y \in \mathbb{F}_{2^t}$.

ii. $Tr(cx) = cTr(x)$ *for all* $c \in \mathbb{F}_2$, *and* $x \in \mathbb{F}_{2^t}$.

iii. $Tr(c) = tc \ (mod\ 2)$ *for* $c \in \mathbb{F}_2$.

iv. $Tr(x^{2^i}) = Tr(x)^{2^i} = Tr(x)$ *for any positive* i.

2.1.2 Feedback Shift Register Sequences

Feedback shift registers (FSRs) are common building block for cryptography. A binary sequence $\{a_i\}_{i\geq 0}$, $a_i \in \mathbb{F}_2$ can be generated by an n-stage FSR, as shown in Figure 2.1. The recursive

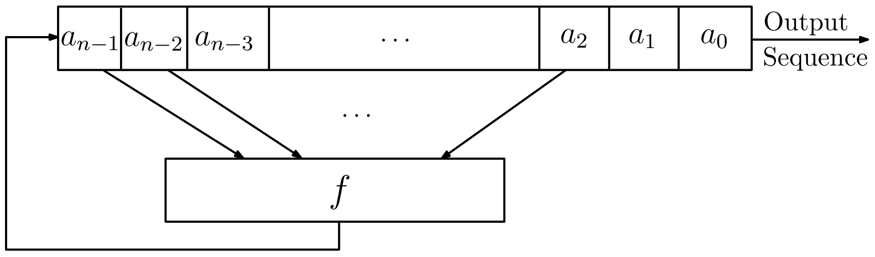


Figure 2.1: Feedback Shift Register

relation of the FSR is defined as

$$a_{n+k} = f(a_k, a_{k+1}, \cdots, a_{n+k-1}), \quad k \geq 0,$$

where $(a_0, a_1, \cdots, a_{n-1})$ is an *initial state* and any *n* consecutive terms of the sequence represent a state of the shift register, *i.e.*, $S_k = (a_k, a_{k+1}, \cdots, a_{n+k-1})$ is the *k-th* state of the shift register.

The sequence $\{a_i\}_{i\geq 0}$ is called a linear feedback shift register (LFSR) sequence if the function *f* is linear, *i.e.*, it is of the form:

$$f(x_0, x_1, \cdots, x_{n-1}) = c_0x_0 + c_1x_1 + \cdots + c_{n-1}x_{n-1}, \ c_i \in \mathbb{F}_2.$$

Otherwise, it is called a nonlinear feedback shift register (NLFSR) sequence.

The internal state of the FSR can be a non-binary variable, for example, it takes from $\mathbb{F}_{2^t}$, and in this case, a sequence generated by the FSR is a non-binary sequence. In other words, the *k-th* state of the FSR is $S_k = (a_k, a_{k+1}, \cdots, a_{n+k-1}), a_i \in \mathbb{F}_{2^t}$. The coefficients of *f* are also non-binary, i.e., $c_i \in \mathbb{F}_{2^t}$, in order to generate the corresponding feedback for the FSR.

Definition 5 *The sequence* $\{a_i\}_{i\geq 0}$ *is called a periodic sequence with period* T *if* $a_i = a_{i+T}$, $i \geq 0$.

Definition 6 *Let* $\boldsymbol{a} = \{a_i\}_{i\geq 0}$ *be a periodic binary sequence with period* T*. The autocorrelation function of* $\boldsymbol{a}$*, denoted by* $C_{\boldsymbol{a}}(\tau)$*, is defined as*

$$C_{\boldsymbol{a}}(\tau) = \sum_{i=0}^{T-1} (-1)^{a_{i+\tau}+a_i}.$$

Definition 7 *A binary sequence with period* $2^n - 1$ *generated by an n-stage LFSR is called an m-sequence.*

Definition 8 *The linear span or linear complexity of a sequence is the length of the shortest LFSR that can generate the entire sequence.*

Definition 9 *A binary sequence with period* $2^n - 1$ *is called a span n sequence if each non-zero n-tuple occurs exactly once in one period.*

2.1.3 The WG Transformation

Let $t \not\equiv 0 \bmod 3$, $3k \equiv 1 \bmod t$, and $h(x) = x + x^{q_1} + x^{q_2} + x^{q_3} + x^{q_4}$, where q_i are given by

$$\begin{aligned} q_1 &= 2^k + 1, \\ q_2 &= 2^{2k} + 2^k + 1, \\ q_3 &= 2^{2k} - 2^k + 1, \\ q_4 &= 2^{2k} + 2^k - 1. \end{aligned}$$

Then the function $\mathsf{WGP}(\cdot) : \mathbb{F}_{2^t} \to \mathbb{F}_{2^t}$ given by

$$\mathsf{WGP}(x) = h(x+1) + 1$$

is called the Welch-Gong (WG) permutation and the function $\mathsf{WGT}(\cdot) : \mathbb{F}_{2^t} \to \mathbb{F}_2$ given by

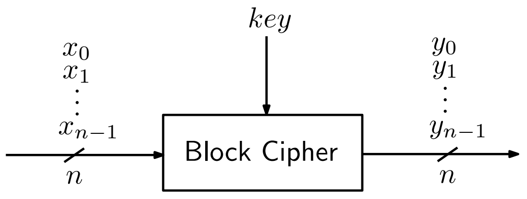
$$\mathsf{WGT}(x^d) = \mathsf{Tr}(\mathsf{WGP}(x^d))$$

is known as the Welch-Gong (WG) transformation with decimation d, where d is coprime to $2^t - 1$ [46].

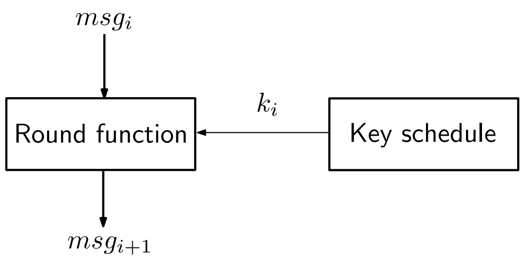
2.2 Symmetric Key Cryptography

In symmetric key cryptography, a secret key is shared between the sender and the receiver. Public key cryptography uses two different keys for encryption and decryption. Public key cryptography involves intensive computations, and it is not feasible, at current research stage, for EPC tags due to the area constraint [77]. The security service of confidentiality can be achieved by ciphers, and authentication can be achieved by challenge/response based protocol, where random numbers are used as challenges. In order to meet these requirements, we mainly focus on the block cipher, stream cipher, and pseudorandom number generator in our following work, therefore we give an overview of them.

Block cipher: A block cipher encrypts an n-bit block of plaintext with a secret key and outputs an n-bit block of ciphertext at a time. Figure 2.2(a) shows the general structure of a block cipher. In general, the block cipher includes two parts: round function and key schedule, as shown in Figure 2.2(b). The round function is iterated multiple times (called round number), in order to increase the unpredictability between the plaintext and ciphertext [23]. The key schedule is used to provide round keys (k_i) for the round function in each round. There are two common architectures for round functions: the Feistel structure and substitution permutation network (SPN) structure, as shown in Figure 2.3. In the Feistel structure, the input message for each round is split into two parts: left half part and right half part. The right half part of the output message directly comes from the input left part and the output left part equals the XORed result of input right half and the output of the function F with inputs of round key and input left half. For the SPN structure, each round function includes adding round key, substitution, and permutation layers. The typical cipher of Feistel structure is DES, and that of SPN structure is AES.



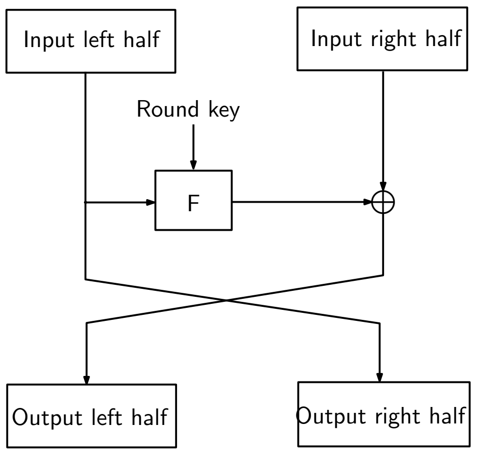
(a) General Structure of a Block Cipher



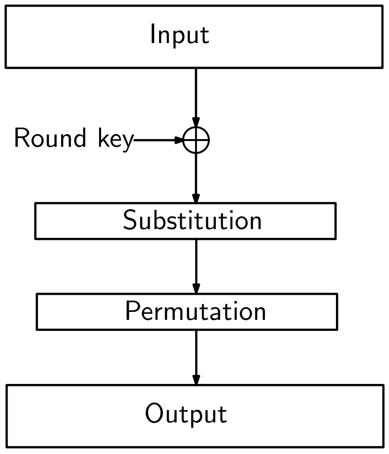
(b) Round Function and Key Schedule

Figure 2.2: Block Cipher

Stream cipher: A stream cipher encrypts each bit of plaintext individually and the general structure of a stream cipher is given in Figure 2.4. Each bit of the ciphertext is obtained by



(a) Feistel Structure



(b) SPN Structure

Figure 2.3: Block Cipher Structures

conducting a bitwise exclusive-or (XOR) operation of each bit of the plaintext and a key bit stream. The key bit stream is implemented by a pseudorandom number generator using a secret key as input [23].

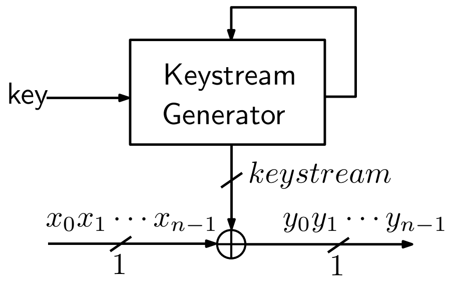


Figure 2.4: General Structure of a Stream Cipher

Pseudorandom number generator: PRNG is also known as deterministic random number generator, which is constructed from a deterministic function with a secret key, and its output has good statistical properties and approximates a sequence of truly random numbers [89].

2.3 Lightweight Cryptography

The goal of lightweight cryptography is to achieve a balance of the tradeoffs between security, area and throughput. To illustrate it in a block cipher, as depicted in Figure 2.5, the key size

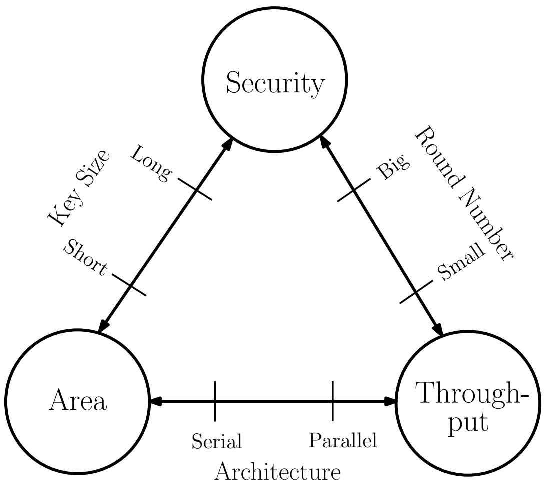


Figure 2.5: Design Tradeoffs between Security, Area, and Throughput

provides a tradeoff between security and area, the round number provides a tradeoff between security and throughput, and the implementation architecture provides the area and throughput tradeoff. In general, it is easy to optimize two of them, but it is difficult to consider all of them [93, 62, 63].

Generally, the proposals of lightweight symmetric ciphers can be divided into three approaches [93]. The first approach involves optimized and compact hardware implementations of standardized algorithms, e.g., compact AES implementations [82, 50, 39, 38]. The second method leverages the slight modification of a well-studied cipher, such as DESXL [94], a lightweight variant of DES. Finally, the third method is to design new ciphers with the goal of having low hardware implementation costs.

Recent proposals include lightweight block ciphers TEA [112], XTEA [86], HIGHT [56], SEA [105], PRESENT [17], KATAN and KTANTAN [26], CLEFIA [103], LED [49], PRINCE [18], EPCBC [117], KLEIN [47], LBlock [113], and Piccolo [102], Twine [106], and the more recent SIMON and SPECK [11]. There exist also some lightweight stream ciphers, such as Trivium [27], Grain [55] and lightweight WG (Welch-Gong) stream ciphers (WG-5 [7], WG-7 [71], WG-8 [35]). Additionally, several surveys of recently published lightweight cryptographic implementations can be found in [29, 75, 66]. In particular, lightweight ciphers have attracted a lot of attention from industry. CLEFIA, PRESENT, and Trivium have been adopted by the ISO/IEC Standard 29192. PRESENT-80 and Grain-128 have been adopted by ISO/IEC Standard 29167, which provides the cipher suites for the RFID air interfaces.

In this section, we give a review of the lightweight block ciphers SIMON and SPECK which are related to our cipher Simeck, lightweight stream ciphers Trivium, Grain and WG.

2.3.1 Lightweight Block Ciphers

SIMON and SPECK are two lightweight block cipher families, designed by the NSA's researchers in 2013 [11, 13, 12]. Each of SIMON and SPECK contains ten instances with various block sizes and key sizes. SIMON and SPECK offer excellent performance on both hardware and software platforms, such as ASIC, FPGA, and 4/8/16/32-bit microcontrollers, and are designed to perform well across the full spectrum of lightweight applications [11]. SIMON is optimized for hardware implementations, and SPECK is tuned for optimal performance in software. The round functions of SIMON and SPECK are based on the Feistel structure.

The following notations are used to describe SIMON, SPECK and our Simeck in Chapter 3.

- $x \lll c$ and $x \ggg c$ denote the cyclic shift of x to the left and right by c bits respectively.
- $x \odot y$ is the bitwise AND of x and y.
- $x \oplus y$ is the exclusive-or (XOR) of x and y.
- $x \boxplus y$ is the integer addition modular 2^n of x and y, where n is the word size.

The SIMON and SPECK block ciphers with n-bit word size ($2n$-bit block size), and m-word (mn-bit) key is denoted as SIMON$2n/mn$ and SPECK$2n/mn$. There are ten instances for each of SIMON and SPECK family for the combination of n and m, where $n \in \{16, 24, 32, 48, 64\}$ and $m \in \{2, 3, 4\}$. The details of them are listed in Table 2.1.

In this thesis, we only consider SIMON$2n/mn$ and SPECK$2n/mn$ with word size n equals 16, 24, and 32, the key words m equals 4 here, in order to be consistent with our Simeck in Chapter 3. They are SIMON32/64, SIMON48/96, SIMON64/128 and SPECK32/64, SPECK48/96, SPECK64/128.

Table 2.1: Ten SIMON and SPECK Instances

SIMON	**SPECK**
SIMON32/64	SPECK32/64
SIMON48/72	SPECK48/72
SIMON48/96	SPECK48/96
SIMON64/96	SPECK64/96
SIMON64/128	SPECK64/128
SIMON96/96	SPECK96/96
SIMON96/144	SPECK96/144
SIMON128/128	SPECK128/128
SIMON128/192	SPECK128/192
SIMON128/256	SPECK128/256

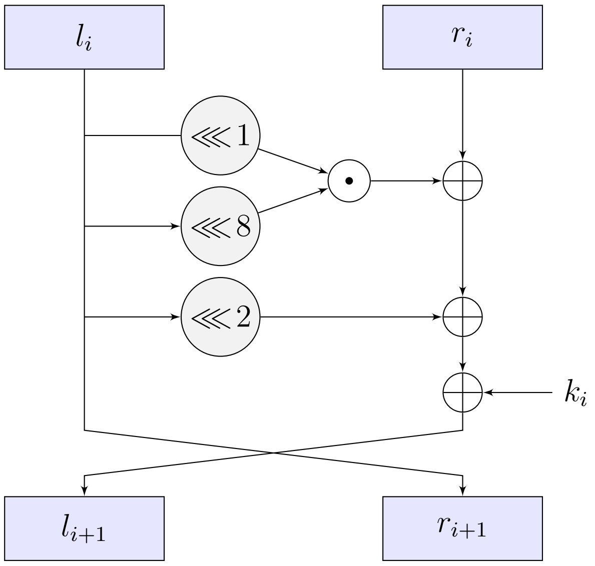


Figure 2.6: The Round Function of SIMON

SIMON

The *i-th* round function of SIMON (as shown in Figure 2.6) is a two-stage Feistel map R_{k_i} which is defined by

$$(l_{i+1}, r_{i+1}) = R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i),$$

where $f(l_i) = ((l_i \lll 1) \odot (l_i \lll 8)) \oplus (l_i \lll 2)$, k_i is the round key, and (l_i, r_i), (l_{i+1}, r_{i+1}) are two internal words in the *i-th* and *(i+1)-th* rounds respectively. l_i and r_i are the left part and right part of the internal words respectively. k_i is generated from the key schedule and $0 \leq i \leq T-1$, where T is the number of rounds. The number of rounds for different SIMON instances are given in Table 2.2.

The key schedule (expansion) of SIMON is depicted in Figure 2.7. The round keys for the first four rounds (k_0, k_1, k_2, k_3) are the four key words, *i.e.*, the input key, of the key schedule, which is used to generate other round keys. The other round keys are generated by

$$k_{i+4} = C \oplus (z_j)_i \oplus k_i \oplus [k_{i+3} \ggg 3 \oplus k_{i+1}] \oplus [(k_{i+3} \ggg 3 \oplus k_{i+1}) \ggg 1],$$

where $0 \leq i < T-4$. The constant C equals $2^n - 4 = 0xff \cdots fc$. The version dependent constant sequence z_j is listed in Table 2.2. $z_0 = 1111101000100101011000011100110$, a period 31 sequence generated by the primitive polynomial $X^5 + X^4 + X^2 + X + 1$. $z_1 = 1000111011111001001100001011010$, a period 31 sequence generated by the primitive polynomial $X^5+X^3+X^2+X+1$. $z_2 = 11011011101011000110010111100000010010001010011100110100001111$, a period 62 sequence formed by bitwise XOR of the period 2 sequence (01) with z_1. In addition, $(z_j)_i$ is the *i-th* bit of z_j.

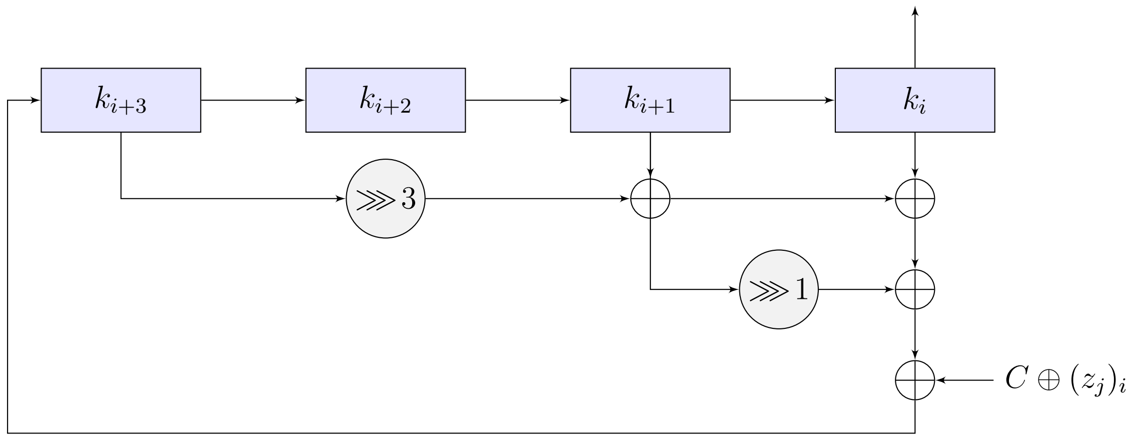


Figure 2.7: The Key Expansion of SIMON

Table 2.2: SIMON and SPECK Parameters

block size $2n$	key size $4n$	word size n	SIMON constant sequence z_j	SIMON rounds T	SPECK rotation α	SPECK rotation β	SPECK rounds T
32	64	16	z_0	32	7	2	22
48	96	24	z_1	36	8	3	23
64	128	32	z_2	44	8	3	27

SPECK

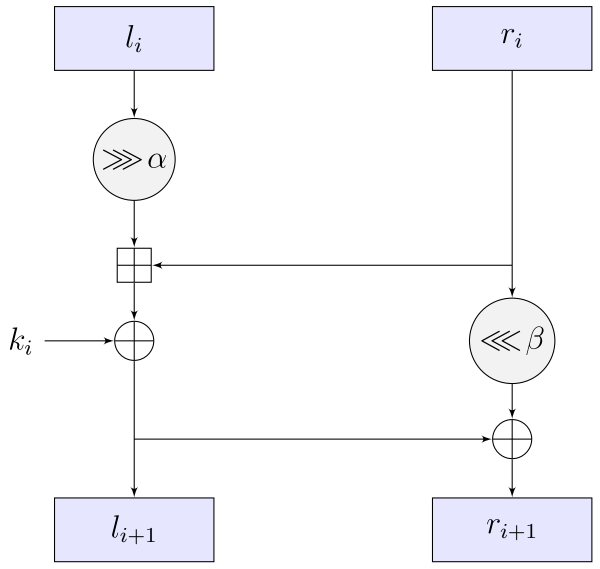


Figure 2.8: The Round Function of SPECK

The *i-th* round function of SPECK,shown in Figure 2.8, is defined by

$$(l_{i+1}, r_{i+1}) = R_{k_i}(l_i, r_i) = ((l_i \ggg \alpha \boxplus r_i) \oplus k_i, r_i \lll \beta \oplus (l_i \ggg \alpha \boxplus r_i) \oplus k_i),$$

where $0 \leq i \leq T - 1$. The number of rounds T and the rotations α and β are different for each instance and they are listed in Table 2.2.

The key schedule reuses the round function to generate the round key k_i, as shown in Figure 2.9. The four key words, which are taken from the input key, of SPECK are $(t_2$, t_1, t_0, $k_0)$, where k_0 is the first round key. The other round keys are generated by

$$\begin{aligned} t_{i+3} &= (t_i \ggg \alpha \boxplus k_i) \oplus i, \\ k_{i+1} &= (k_i \lll \beta) \oplus t_{i+3}, \end{aligned}$$

where $0 \leq i \leq T-1$.

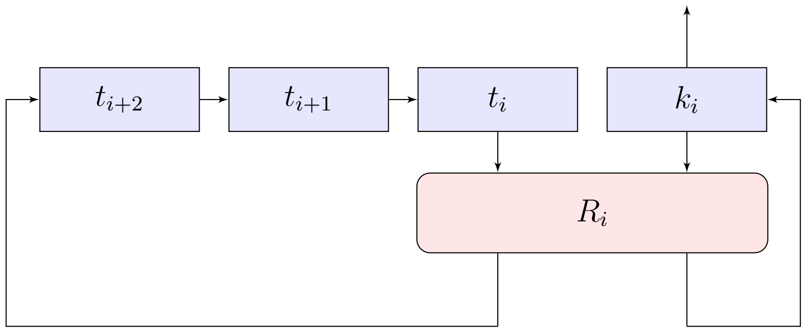


Figure 2.9: SPECK Key Expansion, where R_i is the SPECK Round Function with i acting as the Round Key

2.3.2 Lightweight Stream Ciphers

In this subsection, we present three hardware-oriented lightweight stream ciphers (Trivium, Gain, and WG), where Trivium and Gain are the top finalist of eSTREAM [34] project, and the original WG parameter are also in the phase 2 of eSTREAM [34] project.

Trivium

Trivium [24, 25] was designed in 2005 to be compact in constrained environments. It can generate up to 2^{64} bits of keystream from an 80 bits key $(k_0, \cdots, k_{79})$ and an 80 bits initialization vector $(IV_0, \cdots, IV_{79})$. The cipher itself consists of three NLFSRs (93-bit NLFSR1, 84-bit NLFSR2, 111-bit NLFSR3), and they are denoted as $\{a_0, a_1, \cdots, a_{92}\}$, $\{b_0, b_1, \cdots, b_{83}\}$, and $\{c_0, c_1, \cdots, c_{110}\}$ respectively. The feedback of one NLFSR is generated by the output of another NLFSR, as shown in Figure 2.10. The process of Trivium contains initialization phase and keystream generation phase. More specifically, the 288-bit states of NLFSRs are first initialized with the key and IV as follows.

$$\begin{aligned}(a_0, a_1, \cdots, a_{92}) &= (0, \cdots, 0, k_{79}, \cdots, k_0),\\ (b_0, b_1, \cdots, b_{83}) &= (0, \cdots, 0, IV_{79}, \cdots, IV_0),\\ (c_0, c_1, \cdots, c_{110}) &= (1, 1, 1, 0, \cdots, 0).\end{aligned}$$

Then, the states are rotated for 4×288 times based on the following update function.

$$\begin{aligned}a_{i+93} &= c_i + c_{i+45} + c_{i+1}c_{i+2} + a_{i+24},\\ b_{i+84} &= a_i + a_{i+27} + a_{i+1}a_{i+2} + b_{i+6},\\ c_{i+111} &= b_i + b_{i+15} + b_{i+1}b_{i+2} + c_{i+24}.\end{aligned}$$

After that, assume the values in the NLSFRs are the starting point of the key generation phase, an output keystream, denoted by $\mathbf{s} = \{s_i\}_{i\geq 0}$, can be generated by

$$s_i = a_i + a_{i+27} + b_i + b_{i+15} + c_i + c_{i+45}.$$

The output sequence does not possess any determined randomness properties [23].

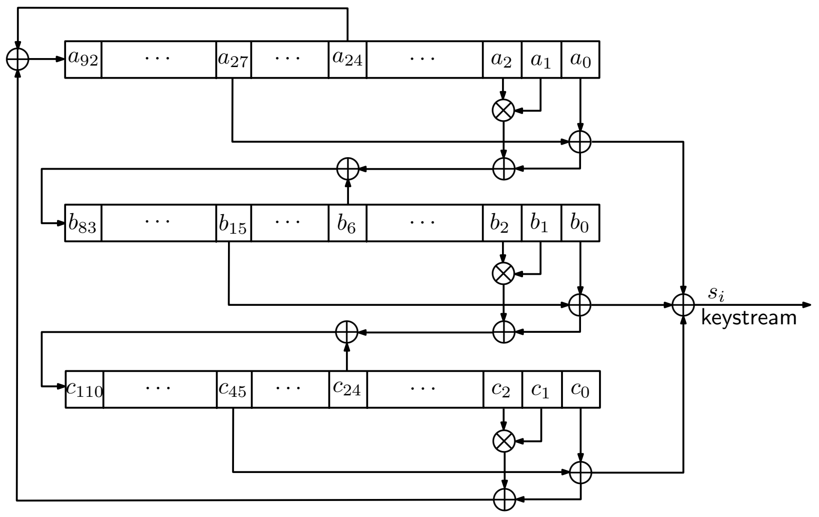


Figure 2.10: The Trivium Stream Cipher

Grain

Grain [54, 55, 53] is a lightweight stream cipher designed in 2005 for applications which have very limited hardware resources. The first version of Grain uses an 80 bits key and a 64 bits

IV [54], and the second version supports key size of 128 bits and IV size of 96 bits [53]. This subsection specifies the details of the first version. Grain consists of three main building parts, a pair of linked 80 bits shift registers and a non-linear filter function. One of the two 80 bits shift registers is LFSR and another one is NLFSR, which are denoted as $\{a_0, a_1, \cdots, a_{79}\}$ and $\{b_0, b_1, \cdots, b_{79}\}$ respectively. In particular, the feedback function of the NLFSR is masked by the output of the LFSR. The nonlinear filter function is used to introduce nonlinearity to the cipher. Moreover, Grain has a lower bound of periods of output sequences [23].

Before the keystream generation phase, the cipher must be initialized with the key (k_i, $0 \leq i \leq 79$) and IV (IV_i, $0 \leq i \leq 63$). During this phase, the NLFSR is loaded with the key bits, where $b_i = k_i$, and the first 64 bits of LFSR are loaded with IV, as $a_i = IV_i, 0 \leq i \leq 63$. The remaining bits of the LFSR are filled with ones, i.e., $a_i = 1, 64 \leq i \leq 79$. Then the cipher is clocked for 160 times without producing any keystream, which is called the initialization phase. After that, the running phase starts, the output keystream s_i is generated clock cycle by clock cycle. The description of them are shown in Figure 2.11.

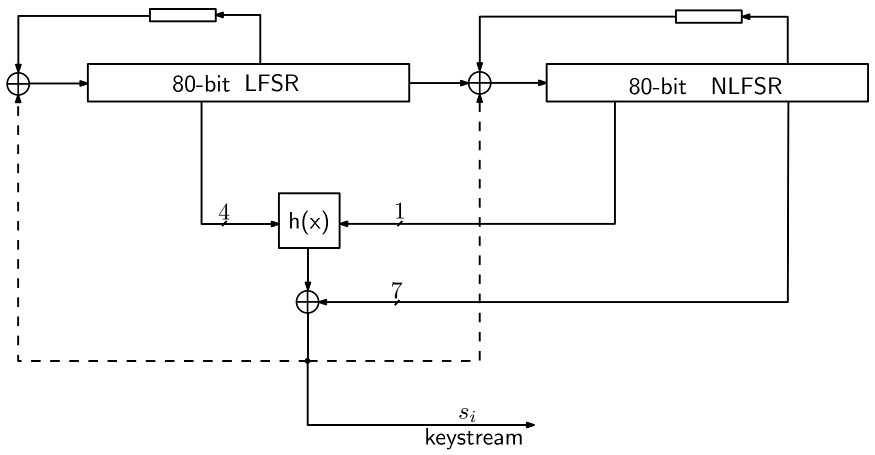


Figure 2.11: The Grain Stream Cipher

In the initialization phase, the feedback update functions of LFSR and NLFSR are XORed with the output keystream s_i (as shown in the dashed lines of Figure 2.11). Therefore, the feedback function of the LFSR is
$a_{i+80} = s_i a_{i+62} + a_{i+51} + a_{i+38} + a_{i+23} + a_{i+13} + a_i, 0 \leq i \leq 159,$

and the feedback function of the NLFSR is

$$\begin{aligned} b_{i+80} = {} & s_i + a_i + b_{i+62} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} + \\ & b_{i+14} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + \\ & b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} + \\ & b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + \\ & b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} + \\ & b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}, \end{aligned}$$

where $0 \leq i \leq 159$.

The output sequence s_i is filtered by a non-linear function $h(x)$. The filtering function $h(x)$ is defined as:

$$h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4,$$

where x_0, x_1, x_2, x_3, x_4 correspond to the tap positions $a_{i+3}, a_{i+25}, a_{i+46}, a_{i+64}$, and b_{i+63} respectively. The output keystream, denoted by $\mathbf{s} = \{s_i\}_{i\geq 0}$, is defined as:

$$s_i = h(a_{i+3}, a_{i+25}, a_{i+46}, a_{i+64}, b_{i+63}) + b_{i+1} + b_{i+2} + b_{i+4} + b_{i+10} + b_{i+31} + b_{i+43} + b_{i+56}.$$

In the running phase, the feedback function of LFSR is
$a_{i+80} = a_{i+62} + a_{i+51} + a_{i+38} + a_{i+23} + a_{i+13} + a_i$, $i \geq 160$,
and the feedback function of the NLFSR is

$$\begin{aligned} b_{i+80} = {} & a_i + b_{i+62} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} + \\ & b_{i+14} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + \\ & b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} + \\ & b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + \\ & b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} + \\ & b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}, \end{aligned}$$

where $i \geq 160$. Accordingly, the keystream $\mathbf{s} = \{s_i\}_{i\geq 160}$ is outputed as shown in the solid lines of Figure 2.11.

WG

The WG stream cipher family is hardware-oriented stream ciphers based on the WG transformation. It was firstly proposed in 2005 [84] and a complete version was published in 2008 [85].

The structure of the WG stream cipher family is described in Figure 2.12. The WG stream cipher is composed of a length l LFSR, followed by a WG transformation with decimation d over $\mathbb{F}_{2^m}$. The optimal decimation parameters which can achieve the highest security level for each WG instance are investigated in [74]. The WG transformation module can be split into WG permutation module and trace module. The characteristic polynomial of the LFSR is a primitive polynomial *f(x)* of degree l over $\mathbb{F}_{2^m}$, i.e., $f(x) = x^l + \sum_0^{l-1} c_i x^i, c_i \in \mathbb{F}_{2^m}$. The WG transformation with decimation d is described in Section 2.1.3, and it exists only when $m \neq 0\ (mod\ 3)$. Thus, the WG stream cipher family is defined when $m \neq 0\ (mod\ 3)$ over $\mathbb{F}_{2^m}$. We denote each specific instance in the family as WG-m stream cipher. Several instances of them have been explored in hardware, such as WG-29 [30], WG-16 [36, 37, 31], WG-7 [67], and WG-5 [7]. The lightweight WG stream ciphers, WG-5[7], WG-7 [71], and WG-8 [35] have been proposed for the resource constrained environments. Their security has been analyzed [87, 99, 74] and can be used in protecting communication in these applications, such as ensuring data confidentiality and performing entity authentication [71].

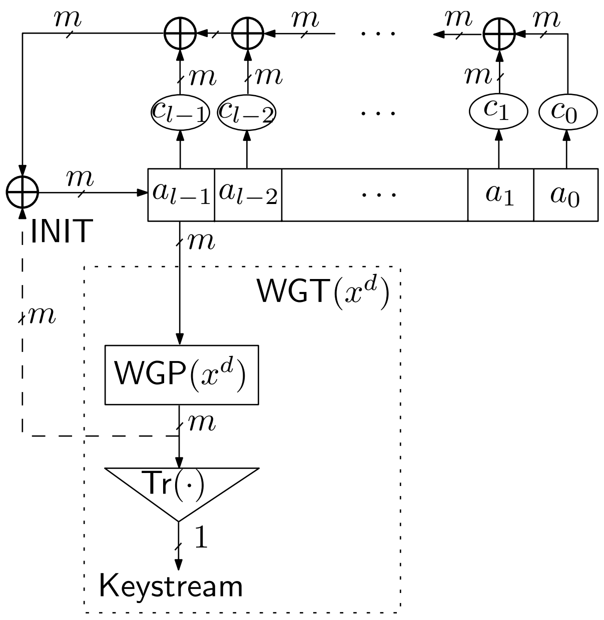


Figure 2.12: The WG Stream Cipher Family

The WG stream cipher contains two phases, the initialization phase and the running phase as shown in the dashed line of Figure 2.12. It is executed *2l* clock cycles in the initialization phase with the recursive values from WGP(x^d) and feedback value of the LFSR. After the initialization phase, the running phase starts and output the one bit keystream generated by the trace function clock cycle by clock cycle with the recursive value from LFSR only. We define the internal states of the LFSR over $\mathbb{F}_{2^m}$ by $\{a_k\}_{k\geq 0}$ and the output keystream over $\mathbb{F}_2$ by $\{s_k\}_{k\geq 0}$. Then, the mathematical expressions for updating the LFSR internal states and generating the output keystream sequence of WG-m are given by:

$$a_{k+l} = \begin{cases} \sum_{i=0}^{l-1} c_i a_{i+k} + \mathsf{WGP}(a_{k+l-1}^d) & 0 \leq k < 2l, \\ \sum_{i=0}^{l-1} c_i a_{i+k} & k \geq 2l. \end{cases}$$

$$s_k = \mathsf{WGT}(a_{k+3l}^d), \quad k \geq 0.$$

Theorem 2 *The keystream sequence of the WG-m stream cipher has the following properties.*

i. Period is $2^n - 1$, where $n = ml$.

ii. It is balanced, i.e., the number of 0's is only one less than the number of 1's in one period of the keystream.

iii. It has an ideal 2-level autocorrelation property.

iv. Any t-tuple is equally likely distributed (ideal t-tuple distribution) for $1 \leq t \leq l$.

v. The linear span or linear complexity of the keystream can be determined exactly, and increased exponentially with m.

2.4 Hardware Design and Optimization

In this section, we discuss hardware implementation technologies and several efficient approaches for cryptographic primitives.

2.4.1 Hardware Implementations

Application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs) are typical hardware implementation techniques. ASIC is much cheaper than FPGA when the quantity of items is large, while FPGA provides more flexibility. Due to the massive deployment of passive RFID tags, the ASIC implementation of cryptographic primitive is more critical. Our optimizations and benchmarking emphasize ASICs more than FPGAs. We use CMOS 130n-m and CMOS 65nm for our implementations because EPC tags use older processes with the consideration of cost and compatibility with analog components.

A typical hardware design process is described in Figure 2.13. Firstly, we write our ideas and specification into Register Transfer Level (RTL) code. Then, we synthesis a RTL code to gate level netlist using logic synthesis tool. We use physical synthesis tool to map the gate level netlist to the actual hardware resources in ASIC or FPGA. Finally, the area, power consumption, and clock speed are generated after the physical implementation.

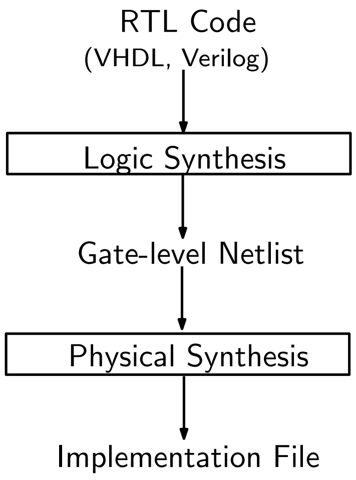


Figure 2.13: Hardware Design Process

The total power consumption is a combination of static and dynamic power consumption, *i.e.*,

$$P_{total} = P_{Static} + P_{Dynamic}.$$

The static power consumption is caused by leakage currents inside transistors. The dynamic power consumption is caused by switching activity by charging and discharging of load capacitance C_{load}, and short circuit currents when transistors switch [58]. However, the switching power consumption dominates the dynamic power consumption, thus the dynamic power consumption is approximately defined as

$$P_{Dynamic} = \alpha \cdot C_{load} \cdot f_{CLK} \cdot V^2,$$

where, α is the switching activity, defined as the number of signal transitions in a clock period. f_{CLK} is the operating clock frequency and V is the supply voltage. For ASIC designs, the total power consumption correlates to the used area. However, for the FPGA designs, the static power consumption correlates to the entire FPGA device and the dynamic power consumption correlates to the used area for a specific design [58].

2.4.2 Reuse, Parallelism and Clock Gating

To get an efficient design in hardware, we can use a number of techniques to optimize the design, such as, reuse of a component, parallelism, and clock gating [97, 67, 82, 98]. Reuse means that we can reuse the existing modules (combinational modules or registers) in different clock cycles in order to decrease the total area. An example of reuse technique over two clock cycles is shown in Figure 2.14.

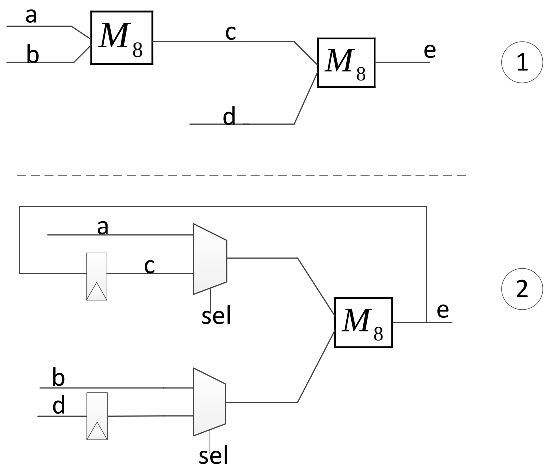
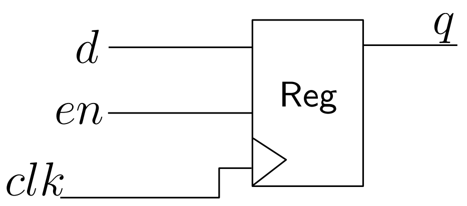


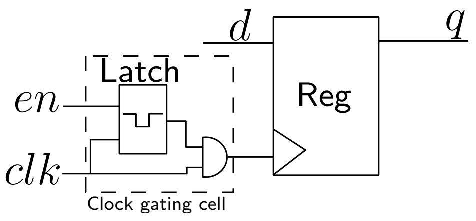
Figure 2.14: Reuse the Multiplier in Two Consecutive Clock Cycles.

Parallelism means that we can improve throughput by adding some extra logic but with s-maller area than using several identical designs. These techniques are popular in hardware design which are used to optimize the designs for achieving better results in terms of area, clock speed, and throughput.

Clock gating is a technique for reducing dynamic power consumption and area. For reducing dynamic power consumption, the clock gating reduces the activity factor by turning off the clock when the circuit is not needed. For reducing area, clock gating can replace n registers with chip-enable with n registers without chip-enable and a clock gating cell, as shown in Figure 2.15. The area of a register with chip-enable is larger than that of a register without chip-enable, as a result the total area is decreased. In our implementation, the power consumption results are



(a) A Register with chip-enable



(b) A Register with Clock Gating Cell

Figure 2.15: Clock Gating

conventionally got by letting the primitives continuous run with stopping. Clock gating is part of the circuit and it is for the cryptographic benchmarking purpose.

2.4.3 Choice of Bases

The design of many cryptosystems involve finite field arithmetic operations. Arithmetic addition can be performed using bit-wise XOR operation under the binary representation, but multiplication operation is complex and time consuming. The complexity is based on the selection of irreducible polynomial for $\mathbb{F}_{2^m}$ and the basis used to represent the finite field elements. During the past years, a lot of research have been given to efficient implementations of arithmetic computations in finite field [69]. In addition, the hardware cost of the ciphers is related to the basis selected for performing computation in $\mathbb{F}_{2^m}$ [23].

Polynomial basis is very good for exploring hardware optimization of multiplication operation, because the multiplication operation using polynomial basis can be implemented using simple shift and XOR operations [28]. Normal basis is efficient and cost effective for hardware implementation of squaring. It is simply a cyclic shift of coordinates of the element [61]. However, the multiplication operation in normal basis is more complex in terms of hardware resources. Optimal normal basis (ONB) [41] is proposed to deal with the constraints in normal basis, which allows not only fast squarings but also fast multiplications with less XOR and AND gates. There are two types of optimal normal bases in $\mathbb{F}_{2^m}$, i.e., type-I and type-II, defined as follows:

1) Type-I: *m* + 1 is a prime *p*, and 2 is primitive modulo *p*.

2) Type-II: *2m* + 1 is a prime *p* and either

- **i.** 2 is primitive modulo *p*, or
- **ii.** $p \equiv 3 \ (mod\ 4)$ and the multiplicative order of 2 modulo *p* is *m*.

However, most of $\mathbb{F}_{2^m}$ do not have a type-I ONB, such as odd prime m. Thus, type-II ONB gives more advantages when m is odd prime [109]. In conclusion, we need to make a trade-off by choosing an irreducible polynomial for the finite field and by carefully selecting a basis in order to achieve the best performance for the entire design.

2.4.4 Tower Field

A wide variety of efficient hardware implementations of AES has been proposed during the past years. Among them, the best method for achieving a very compact S-box is to use subfield [96, 101, 21, 108]. Rijmen [96] suggested using subfield arithmetics in the crucial step of computing an inverse in $\mathbb{F}_{2^8}$ by reducing an 8-bit calculation to several 4-bit ones. Satoh *et al.* [101] further extended this idea, using the tower field approach of Paar [88], by breaking up 4-bit calculations into 2-bit ones, which resulted in a smaller AES circuit. A number of different tower field constructions for S-box had been explored by Canright in [21], and the smallest one is to use three level tower constructions with normal basis in each level. Recently, Ueno *et al.* [108] introduced using PRR (Polynomial Ring Representation), RRB (Redundantly Represented Basis) in the tower field constructions for the inversion circuit in S-box. This method leads to a smaller area of the inversion circuit than Canright's and the corresponding area and time product of S-box is also smaller than that of Canright. Therefore, from the observations of AES, the tower constructions could be useful for efficient implementations of other ciphers.

2.5 EPC Passive RFID Systems

EPCglobal Inc.[2] promotes and leads the standardization of RFID systems, especially for RFID systems operating at 860 MHz - 930 MHz. The Electronic Product Code (EPC) provides identification of items for companies worldwide by storing the EPC number in the so-called EPC memory inside the tags. The EPC Class 1 Generation 2 (EPC C1 G2) UHF (Ultra High Frequency) RFID standard [2, 3] is the most popular one for passive RFID systems. In this section, we give an overview of this type of RFID system.

A typical RFID system consists of three entities: a database, a reader, and a tag. In the current EPC C1 G2 standard [2, 3], there is a tag selection phase for the scenario where the reader communicates with multiple tags. After that, the reader has selected one tag. Therefore, we only consider the situation where one reader communicates with one tag. The diagram of an RFID system is shown in Figure 2.16. We assume that the back-end database and the reader are

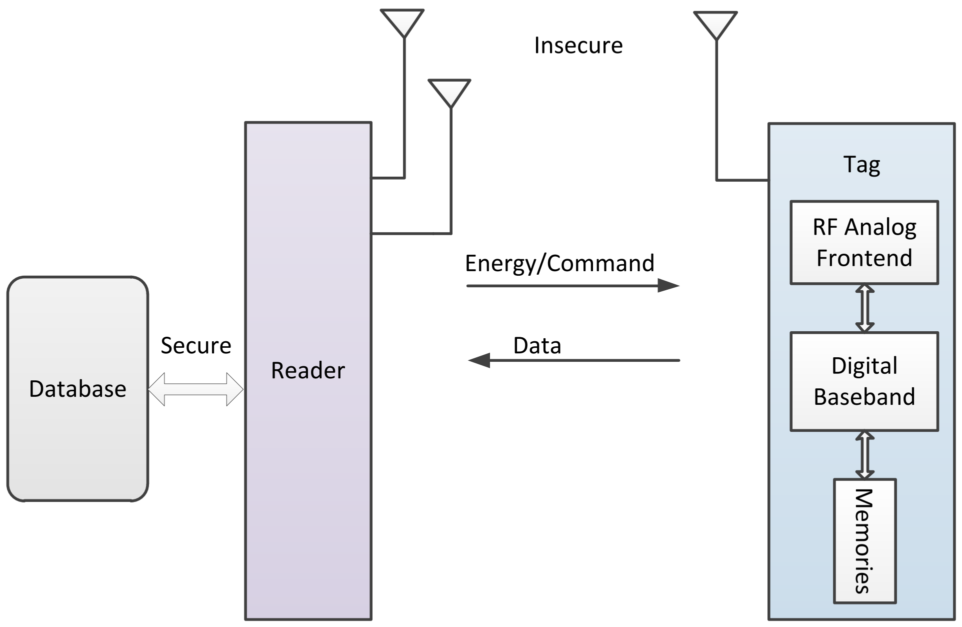


Figure 2.16: The Description of an RFID System

connected with a secure channel via communication protocols such as SSL/TLS. The wireless communication channel between the reader and the tag is insecure.

The reader initiates all the communication between the reader and tag, and it follows a command/response pattern, where the reader sends a command and the tag responds. As shown in Figure 2.16, each tag contains an RF analog frontend, a digital baseband, and four memories. The RF part is used to perform two-way communication with the reader and harvest energy from the reader's signal. The digital baseband is used to process all the commands and data. The four memories are EPC memory, TID (Tag Identification) memory, Reserved memory, and User memory. The EPC number is used for unique identification, and it is stored in the EPC memory. The TID memory stores the unique tag identification number set by the manufacturer. The kill and access passwords are stored in the Reserved memory. The extra item information, such as the weight, place of origin, and so forth, are stored in the optional User memory. The readers' database contains the EPC numbers and the associated passwords of all tags.

According to the EPC C1 G2 standard, the inventory and access protocol between the reader and the tag is shown in Figure 2.17. In which, the reader has two states: inventory and access. The tag has five states: ready, reply, acknowledged, open, and secure. More specifically, the inventory and access protocol is executed as follows. The reader sends a Query command to the tag, and the tag replies with a 16-bit random number, denoted as RN16_Init. After the reader

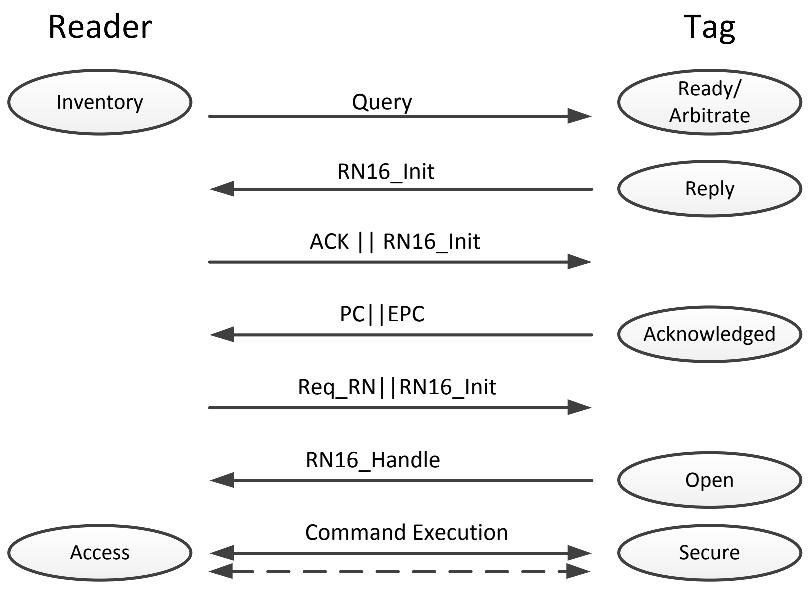


Figure 2.17: The Inventory and Access Protocol between the Reader and the Tag

receives the RN16_Init, it sends an ACK command with the same RN16_Init to the tag. Then the tag sends the EPC number and PC code to the reader if the RN16_Init it received is correct. PC code is the protocol control bits. Upon receiving it, the reader sends a Req_RN command with the RN16_Init to the tag, the tag will reply with a RN16_Handle to the reader after checking the correctness of RN16_Init. The RN16_Handle is another 16-bit random number. After the reader receives the RN16_Handle, the reader may request the tag to execute a command.

Command execution is a multi-round process of bi-directional communication between the reader and tag. The command here can be an access command or a kill command, where the access command transfers the tag from the open state to the secure state, and the kill command transfers the tag from the open state to the killed state. The kill command will disable the tag forever, and the tag cannot respond to the reader any more. However, after the reader successfully sends an access command to the tag which means that the 32-bit access password is correct, the tag will send the same RN16_Handle back and ends up in the secure state. Note that the RN16_Handle is used in the entire procedure of the access command. Then, the reader can send read, write, etc., commands to the tag and the tag will only execute them when it is in the secure state. The exchanged message between the reader and the tag, such as EPC number, RN16 and so on, are very short, and they are typically below 100 bits. Furthermore, RN16 is used for providing verification of the reader identity, and provide cover-code (mask) for the data in access, kill, and write commands.

It is worth mentioning that the two 32-bit passwords for the access and kill commands are designed to be two security features in the first version of the EPC C1 G2 standard [2]. However, the research [111, 40, 43, 51, 22] shows that the two 32-bit passwords can be broken easily if the attacker has the ability to eavesdrop on the bi-directional communications between the reader and the tag. As a result, some optional security requirements are provided in the second version of the EPC C1 G2 standard [3], which was released in 2013. For example, some of these optional security requirements include *Authenticate* and *SecureComm* commands. The *Authenticate* command allows the RFID system to perform reader, tag or mutual authentication. The *SecureComm* command allows encrypted communication between the reader and the tag.

Since the link between the reader and tag is a wireless channel, all the existing attacks on the wireless link also apply to the link between the reader and tag [23]. Moreover, the resource constrained properties of the tag and insecure channel between the reader and tag lead to many vulnerabilities [59, 111, 95]. A diagram about threats on passive RFID systems is depicted in Figure 2.18. It describes the authorized and unauthorized entities, including an authorized read-

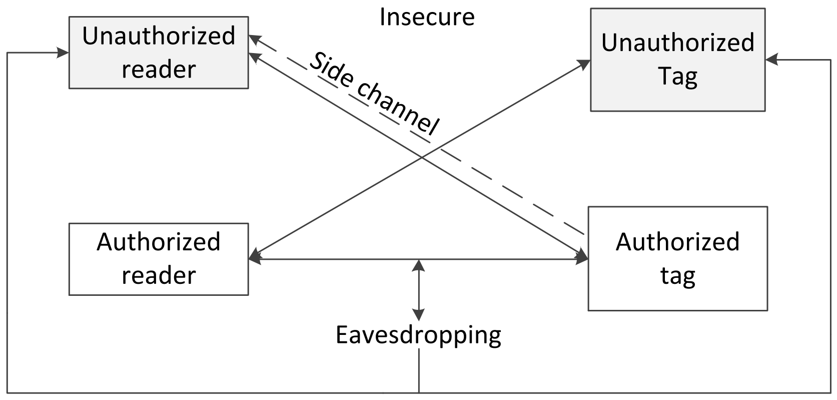


Figure 2.18: Threats of an RFID System

er, an unauthorized reader, an authorized tag, and an unauthorized tag. Figure 2.18 also shows the interactions of the entities in the security model. An unauthorized reader can interrogate an authorized tag, and the unauthorized tag can impersonate an authorized one. The unauthorized reader and unauthorized tag can eavesdrop on the insecure wireless channel between the authorized reader and authorized tag. The unauthorized reader can observe the external information emitted from the authorized tag through the side channel.

Consequently, the attacker can take advantage of the capabilities of the unauthorized reader and unauthorized tag to launch many attacks [42, 90, 81, 95], such as unauthorized tag reading attack, tag data modification attack, man-in-the-middle attack, eavesdropping attack and side channel attack. These attacks may lead to information disclosure and cause the system to respond in an unexpected or damaging way.

The typical security solution for the attacks is to adopt cryptography to the passive RFID system, which can provide confidentiality, integrity check, and authentication. Due to the resource constrained properties of the passive RFID tag, lightweight cryptography is devised to solve this problem. Therefore, exploring the hardware performance and optimizations of lightweight cryptography is increasingly important in the IoT era.

Chapter 3

The Simeck Family of Lightweight Block Ciphers

The security of the recently proposed lightweight block ciphers, SIMON and SPECK [11], have been investigated [15, 8, 19] by the research community since they were public. These ciphers are recognized to be the smallest block ciphers in each of the block/key size categories when used in resource-constrained environments. SIMON is optimized for hardware implementation, while SPECK is optimized for software. Inspired by the designs of SIMON and SPECK, we combine their good components in order to get a new block cipher family, called Simeck We use a slightly modified version of SIMON's round function, and reuse it in the key schedule like SPECK does. Moreover, we take the benefits of using Linear Feedback Shift Register (LFSR) based constants in the key schedule in order to further reduce hardware implementation footprints. The new family of lightweight block ciphers Simeck aims to have comparable security levels but more efficient hardware implementations.

Based on the aforementioned motivations, we have the detailed design goals as follows.

Hardware. First, we want to minimize the area and power consumption of the Application Specific Integrated Circuit (ASIC) implementations. Secondly, we also wish to allow a range of options in the area, throughput, and power consumption. Finally, we would like to keep the maximum operating frequency as high as possible.

Applications. Considering the application of passive RFID tags as an example, Simeck should satisfy the following requirements in order to be used in practice: 1) The area of Simeck should be less than 2000 GEs [60, 9]. 2) The power consumption of Simeck should be very small. 3) The typical passive RFID tag's operating frequency is 2 MHz and the data rate is from 5 Kbps to 320 Kbps depending on different modulation settings [3, 118]. Thus the throughput is from 5K/2M = 1/400 to 320K/2M = 4/25. Therefore, if the tag's operating frequency is 100 KHz (for a benchmarking purpose), the throughput of Simeck should at least be from 100 K · 1/400 bps = 0.25 Kbps to 100 K · 4/25 bps = 16 Kbps.

Security. Although SIMON and SPECK were designed with small, simple round functions, they are iterated a sufficient number of times in order to resist attacks. We follow the same strategy with Simeck, and due to its similarity with SIMON, we benefit from its analysis carried so far.

Remark 1 *The design and cryptographic analysis of the Simeck algorithm were done in collaboration with colleagues in the ComSec lab. My contribution to the design of the Simeck algorithm was to provide in-depth knowledge of hardware design and guide the mathematical design choices to optimize the hardware. The hardware design and analysis are entirely my own work.*

This chapter is organized as follows. In Section 3.1, we describe the specifications and design rationales of the Simeck family. Section 3.2 first presents our metrics and design flow in CMOS 130nm and CMOS 65nm ASICs. Then, we give two different hardware architectures of Simeck in order to make a trade-off between area, throughput, and power consumption. Later, the hardware evaluations in CMOS 130nm and CMOS 65nm are given with a thorough analysis. In Section 3.3, we compare our results of Simeck and SIMON with the results in [11]. The comparisons with other lightweight ciphers are given in Section 3.4. Section 3.5 concludes this chapter.

3.1 Design Specification and Rationale

In this section, we give specification as well as design rationale of our block cipher family Simeck.

3.1.1 Specification of Simeck

Our lightweight block cipher family Simeck is denoted as Simeck$2n/mn$, where $2n$ is the block size and mn is the key size. The n is the word size and n is required to be 16, 24, or 32 and m equals 4. Thus, our Simeck family includes Simeck32/64, Simeck48/96, and Simeck64/128.

For example, Simeck32/64 operates on 32-bit message blocks using a 64-bit key. These choices of the ciphers aim to fit different applications of embedded systems including RFID systems.

Simeck is designed to be extremely small in hardware. The round function and the key schedule follow the Feistel structure. A plaintext block of $2n$ bits to be encrypted is first divided into two words l_0 and r_0, where l_0 contains the most significant n bits, and r_0 consists of the least significant n bits. Then these two words are processed by the Simeck round function for a certain number of rounds, and finally the two output words l_T and r_T are concatenated to form a complete ciphertext, where T denotes the total number of rounds. In the following, we give the details of the design. The notations are the same as that in SIMON and SPECK described in Section 2.3.

Round Function

We define the round function (of the i-th round) as the following function,

$$R_{k_i}(l_i,\ r_i) = (r_i \oplus f(l_i) \oplus k_i,\ l_i),$$

where l_i and r_i are the two words of the internal state of Simeck at the i-th round , k_i is the round key, and the function f is defined as

$$f(x) = (x \odot (x \lll 5)) \oplus (x \lll 1).$$

Figure 3.1 illustrates the operations of the round function R_{k_i}. The inverse of the round function, used for decryption, is

$$R_{k_i}^{-1}(l_i, r_i) = (r_i,\ l_i \oplus f(r_i) \oplus k_i).$$

Key Schedule/Expansion

To generate the round key k_i from a given master key K, the master key K is first segmented into four words and loaded as the initial states (t_2, t_1, t_0, k_0) of the feedback shift registers as shown in Figure 3.2. The least significant n bits of K are loaded into k_0; while the most significant n bits are put into t_2. To update the registers and generate round keys, we reuse the round function with a round constant $C \oplus (z_j)_i$ acting as the round key, i.e., $R_{C\oplus(z_j)_i}$. The updating operation can be expressed as

$$\begin{cases} k_{i+1} &= t_i, \\ t_{i+3} &= k_i \oplus f(t_i) \oplus C \oplus (z_j)_i, \end{cases}$$

where $0 \le i \le T-1$. The value k_i is used as the round key of the i-th round.

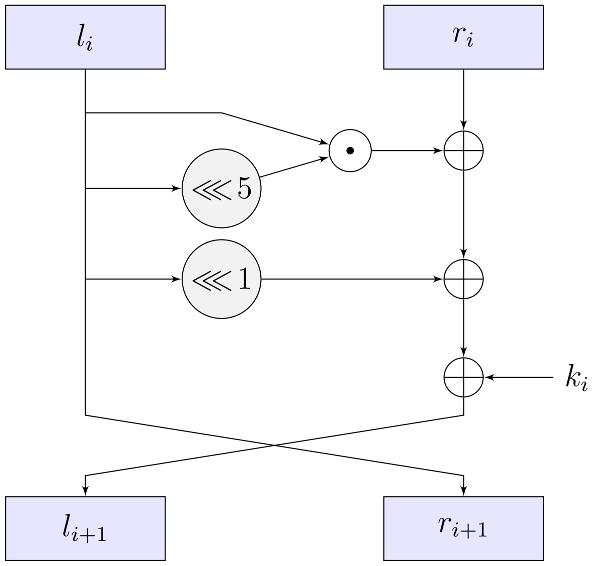


Figure 3.1: The Round Function of Simeck

The value of the constant C is defined by $C = 2^n - 4$, where n is the word size. $(z_j)_i$ denotes the i-th bit of the sequence z_j. Simeck32/64 and Simeck48/96 use the same sequence z_0, i.e. $j = 0$, which is an m-sequence with period 31 and can be generated by the primitive polynomial $X^5 + X^2 + 1$ with the initial state $(1, 1, 1, 1, 1)$. When the rounds number is larger than 31, the sequence repeats itself. Simeck64/128 uses another m-sequence z_1 with period 63, which is generated by the primitive polynomial $X^6 + X + 1$ with the initial state $(1, 1, 1, 1, 1, 1)$.

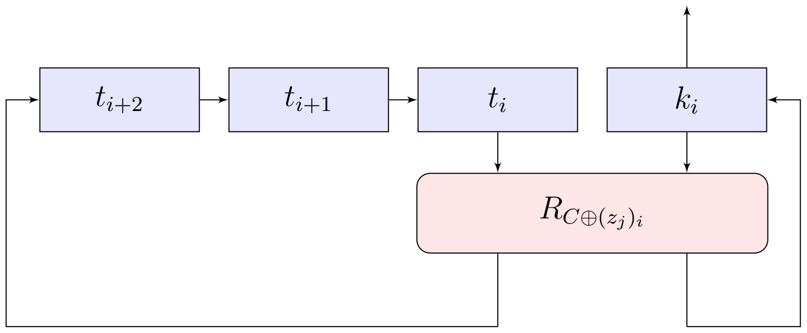


Figure 3.2: The Key Expansion of Simeck, where $R_{C\oplus(z_j)_i}$ is the Simeck Round Function with $C \oplus (z_j)_i$ Acting as the Round Key

Number of Rounds

The number of rounds T for Simeck32/64, Simeck48/96, and Simeck64/128 are 32, 36, and 44 respectively.

3.1.2 Design Rationale

In Simeck, we use a slightly simplified version of the round function of SIMON. The round function of SIMON can be expressed as

$$R'_{k_i}(l_i,\ r_i) = (((l_i \lll 1) \odot (l_i \lll 8)) \oplus (l_i \lll 2) \oplus r_i \oplus k_i,\ l_i),$$

where l_i and r_i are the input words, and k_i is the round key. The operations of the round function only contain bitwise AND, XOR and cyclic shifts, and they are very efficient for hardware implementations. In particular, for Simeck, we change these shift numbers from $(1, 8, 2)$ to $(0, 5, 1)$. We choose our shift numbers in order to realize an acceptable trade-off between hardware performance and security. These modifications will improve the efficiency of hardware implementations, but will have comparable security strengths against certain attacks. More discussions will be given in the following sections.

For the key expansion/schedule algorithm of Simeck, we learn the idea of re-using the round function to update the round-key registers from the design of SPECK.

Concerning the number of rounds for Simeck, we choose the same numbers as the corresponding block ciphers in the SIMON family, in order to have comparable security levels and fair hardware implementation evaluations.

To defeat certain self-similarity attacks such as slide attacks and rotational attacks, we add the round constants C and $(z_j)_i$ into the key expansion process. The constant $C = 2^n - 4$ is also used in the key expansion of SIMON. The polynomials for the two m-sequences z_0 and z_1 are chosen to have minimum numbers of non-zero coefficients, such that their hardware implementations will have small footprints.

Due to its similarity with SIMON and SPECK, most of the security analysis of Simeck follow from the best recent known attacks against the SIMON and SPECK families of block ciphers. The initial security analysis of Simeck is shown in [116], where the security level of Simeck is comparable to those of SIMON and it is reasonable to be used in practice. After that, more security analysis of Simeck is given in [10, 65, 83, 70].

In summary, the differences of the round function and key schedule in SIMON and Simeck are shown in Table 3.1. The round function and key schedule of SIMON are given in Section 2.3.

Table 3.1: Differences between SIMON and Simeck

	SIMON	Simeck	Comments
Round function	three shift numbers $(1, 8, 2)$	two shift numbers $(5, 1)$	
Key schedule	4 XORs two shift numbers $(3, 1)$	3 XORs + 1 AND two shift numbers $(5, 1)$	except the $C \oplus (z_j)_i$
Key constant (generating polynomial)	SIMON32/64: $X^5 + X^4 + X^2 + X + 1$ SIMON48/96, SIMON64/128: $X^5 + X^3 + X^2 + X + 1$	Simeck32/64, Simeck48/96: $X^5 + X^2 + 1$ Simeck64/128: $X^6 + X + 1$	

3.2 Hardware Implementations

We discuss the hardware implementations of the Simeck family of block ciphers in this section.

3.2.1 Metrics and Design Flow

We use the Synopsys Design Compiler Version D-2010.03-SP4 to synthesize the RTL of the designs into netlist based on the STMicroelectronics CMOS 65nm CORE65LPLVT_1.20V and IBM CMOS 130nm CMR8SF-LPVT Process SAGE v2.0 standard cell libraries with both having a typical 1.2V voltage, and 25°C temperature. Cadence SoC Encounter v09.12-s159_1 is used to finish the Place and Route phase in order to generate the layout of the designs. We use Mentor Graphics ModelSim SE 10.1a to conduct functional simulation of the designs and perform timing simulation by using the timing delay information generated from SoC Encounter as well. The areas of the designs after the logic synthesis are provided for comparison with previous ciphers, and a more accurate area after the Place and Route is also provided for deploying the ciphers in practical cases. The densities used for the Place and Route phase for CMOS 130nm and 65nm are 0.92 and 0.93 respectively, in order to make a trade-off between area and maximum operating frequency when the densities are high enough. We choose them because the area after the place and route phase will decrease when the density is higher. Correspondingly, the critical path will also increase in this case; leading to potential DRC (Design Rule Check) and LVS (Layout Versus Schematic) violations. As usual, the area is measured in gate equivalents (GEs), and one GE is equivalent to the physical area required for the two-input one-output NAND gate with the lowest driving strength of the corresponding technology. The areas of one GE are 2.08 $(\mu m)^2$ and 5.76 $(\mu m)^2$ for ST CMOS 65nm and IBM CMOS 130nm respectively.

We use SoC Encounter v09.12-s159_1 to generate the accurate power consumption based on the activity information generated from the timing simulation with frequencies of 100 KHz and

2 MHz, and a duration time of 0.1 s and 5 ms respectively. We do so because the 100 KHz clock frequency is widely used for benchmarking purpose in resource-constrained applications and 2 MHZ is the typical practical operating frequency in passive RFID tags [100]. 0.1 s and 5 ms are long enough to provide an accurate activity information for all the signals in these two cases. Moreover, the maximum clock frequency which can be operated for a specific design is obtained by using the critical path after the place and route phase.

Table 3.2: The Areas of Basic Gates in the Libraries

	IBM130nm-8RF (NSA [11])	IBM130nm-CMR8SF-LPLVT	ST CMOS65nm
NAND	1	1	1
AND	1.25	1.25	1.25
OR	1.25	1.25	1.5
NOT	0.75	0.75	0.75
XOR	2	2	2.25
XNOR	2	2	2.25
2-1 MUX	2.25	2.25	2
DFF	4.25	4.25	3.75
1-bit full adder	5.75	5.75	4.5
Scan FF	6.25	5.5	4.75

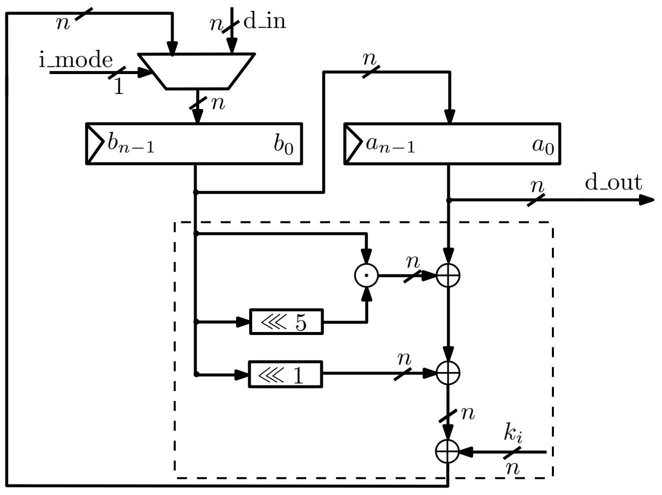
In fact, during the analysis of the previous results [11, 26, 82, 92, 98], the ASIC results for various implementations differ not only in the basic gate technology but also in the types of flip-flops used. In order to compare our results with the previous ones fairly, we provide the areas of some basic gates in our specific libraries and the library used in [11] by the researchers from the NSA for SIMON in Table 3.2. In addition, all the areas of basic gates provided here are the smallest ones in the library, and we normalize the area of the two-input one-output NAND gate to be 1. We observe that our IBM 130nm library is almost the same as the IBM 130nm library used by the researchers from the NSA [11] except the scan flip-flops in terms of the areas of the basic gates.

3.2.2 Two Different Hardware Architectures for Simeck

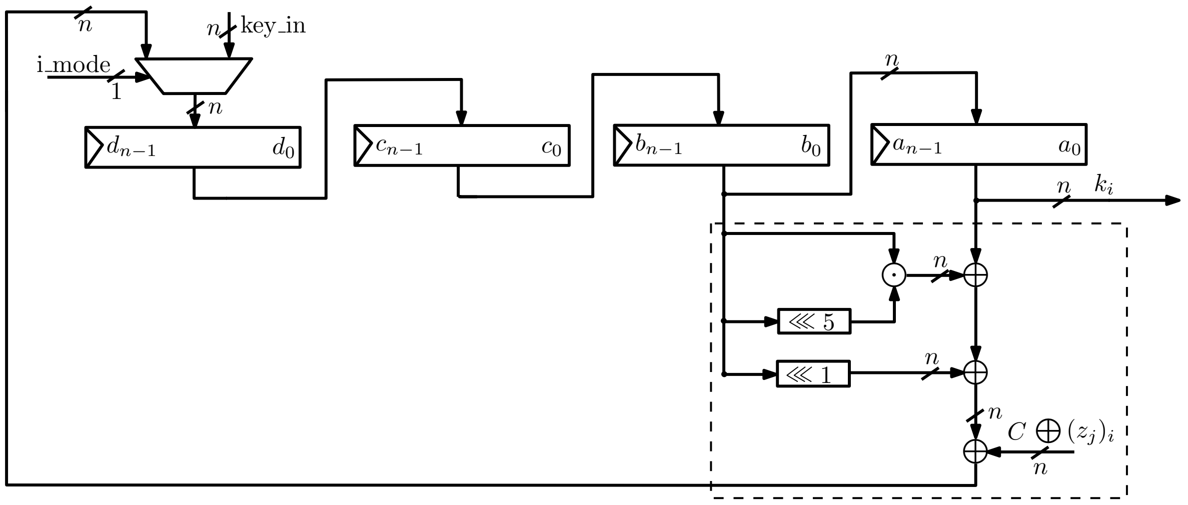
In this section, we target low-area implementations of Simeck and make a trade-off between area and throughput. Meanwhile, we still keep a very high operating frequency. We give two architectures for the implementations: one is parallel architecture, and another is fully serialized architecture. Moreover, we provide a block diagram of the top-level I/O interface between the cipher and the outside environment in order to provide a benchmark for the future implementations and comparison with other ciphers.

Parallel Architecture

The parallel architecture processes one round of the message in one clock cycle, and one round of the key schedule at the same clock cycle, as shown in Figure 3.3. This architecture provides a very high throughput while keeping a compact design. The round function in Figure 3.3(a) in-



(a) Parallel Datapath for the Round Function



(b) Parallel Datapath for the Key Schedule

Figure 3.3: Parallel Architecture for Simeck

cludes three parts: $2n$ flip-flops, a n-bit width 2-to-1 multiplexer, a combinational circuit (dashed box) to compute the feedback data for the multiplexer. Inside the $2n$ flip-flops, n flip-flops are for the message b, and the other n ones are for the message a, where b and a are the left and right parts of the entire message respectively. The multiplexer is used to select the initial plaintext or the feedback data from the combinational circuit for the message b. The combinational circuit includes one n-bit AND gate, three n-bit XOR gates, and two shift modules (cyclic shift to the left by 5 bits and 1 bit). The shift modules cost no extra hardware resources, because they can be done by rewiring the corresponding signals. When the cipher runs, the n-bit data from the message block b shifts to message block a, and simultaneously, the message block b loads a new n-bit data from the multiplexer until the cipher stops. The round key k_i in the combinational circuit for every round comes from the key schedule function, which generates a key for every rounds until the cipher outputs the ciphertext.

The key schedule in Figure 3.3(b) is similar to the round function in Figure 3.3(a), where the key schedule has four n-bit key blocks and one input to the combinational circuit (dashed box) is different, compared to the round function. This n-bit input to the key schedule is a combination of an $(n-1)$-bit constant and a 1-bit signal generated from the control circuit.

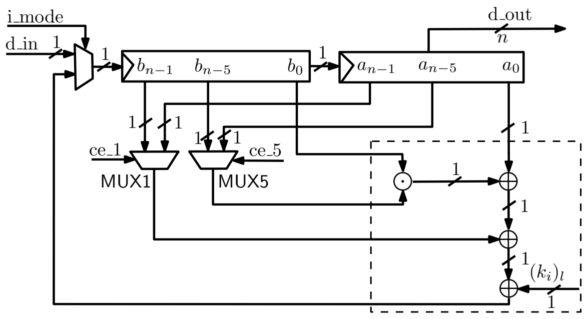
All the flip-flops in the round function and key schedule are standard flip-flops without chip-enable in our architecture. In addition, there are only two n-bit width 2-to-1 multiplexers in total in our architecture to select the initial data or feedback data, where one is for the round function, and the other is for the key schedule. Moreover, the latency for generating a ciphertext using our parallel architecture is $T+4$, where T is the total number of rounds.

Partially Serialized Architecture

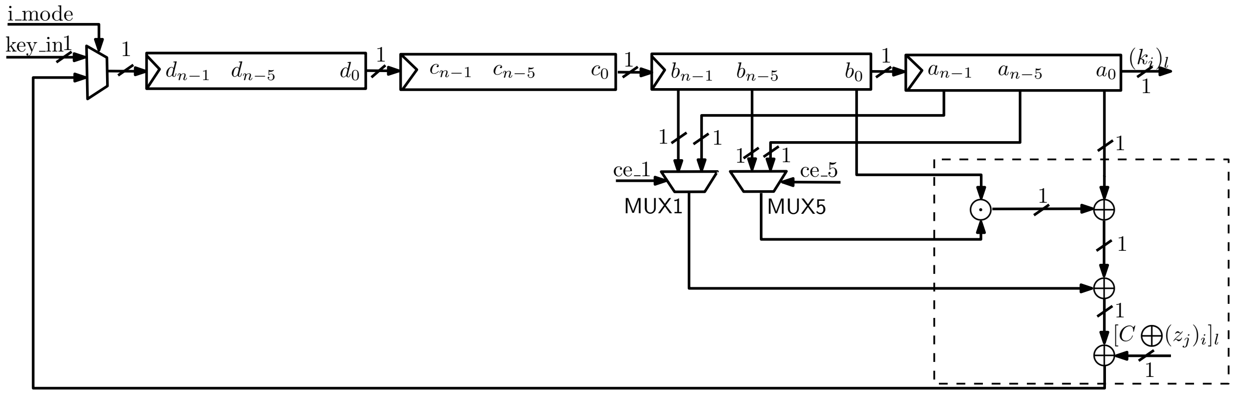
In order to make a trade-off between area, throughput, and power consumption, we provide a partially serialized architecture. This architecture processes only several bits in the round function and the key schedule during one clock cycle. The specific partially serialized size (par_sz) of Simeck are summarized as follows:

$$\begin{aligned} \textsf{Simeck32/64} &: 1, 2, 4, 8, \\ \textsf{Simeck48/96} &: 1, 2, 3, 4, 6, 8, 12, \\ \textsf{Simeck64/128} &: 1, 2, 4, 8, 16. \end{aligned}$$

Besides the round counter (i in Figures 3.3 and 3.4) in the control circuit, there is another counter to control the rounds of the specific serialized size in the partially serialized architecture. The range of this serialized counter (l in Figure 3.4) is between 0 and $n/\text{par\_sz}-1$. In total, the latency for generating a ciphertext is $(n/\text{par\_sz})\cdot(T+4)$, where T is the total number of rounds.



(a) Fully Serialized Datapath for the Round Function


i_mode
key_in
d_{n-1} d_{n-5} d_0
c_{n-1} c_{n-5} c_0
b_{n-1} b_{n-5} b_0
a_{n-1} a_{n-5} a_0
$(k_i)_l$
ce_1
ce_5
MUX1
MUX5
$[C \bigoplus (z_j)_i]_l$

(b) Fully Serialized Datapath for the Key Schedule

Figure 3.4: Fully Serialized Architecture for Simeck

A fully serialized architecture is shown in Figure 3.4. In this architecture, the multiplexer (MUX), and combinational circuit (dashed box) are all 1-bit width, which save a lot of area. Compared to the parallel architecture, there are two more multiplexers, due to the two shift numbers. They are used to select the cyclic shift inputs. The MUX1 is used for the left shift by 1 bit, and MUX5 is used for left shift by 5 bits. The MUX1 selects b_{n-1} as input when the serialized counter equals 0, and chooses a_{n-1} when the serialized counter is larger than 0. Similarly, the MUX5 selects b_{n-5} when the serialized counter is smaller than or equal to 4, and chooses a_{n-5} when the serialized counter is larger than 4.

The partially serialized architecture with par_sz larger than 1 is similar to the fully serialized architecture, where the multiplexer and combinational circuit are par_sz-bit width and the selection signals for the multiplexers (MUXes selection circuitry) are different for various values of par_sz.

The Top-level I/O Interface for Different Architectures

As discussed in Section 3.2.1, the area of the chip depends on not only the area of the basic gates, but also the adopted types of flip-flops. We provide a top-level I/O interface between the cipher and the outside environment as shown in Figure 3.5. We do not have a Finite State Machine

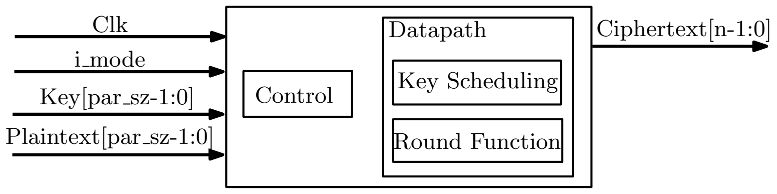


Figure 3.5: The Top-level I/O Interface between the Cipher and the Outside Environment

(FSM) to control the circuit with the purpose of reducing the entire area as much as possible. In our top-level architecture, the cipher is always running and it is controlled by the outside signal `i_mode`. Therefore, we only have two modes in our architecture: loading phase and running phase. The cipher goes into loading phase when `i_mode` equals 0, and it loads the initial data from the inputs `Key` and `Plaintext`. Later on, the cipher begins running phase when `i_mode` equals 1. The user obtains the `Ciphertext` at the end of the running phase. Then, `i_mode` returns back to 0, another `Plaintext` encryption begins. As our architecture never stops, all the flip-flops in the datapath are standard flip-flops without chip-enable signals. This property makes our design ever smaller in terms of area. This architecture presents a benchmark ASIC implementation of Simeck and can be used to fairly compare with the hardware results of other ciphers.

It is worth mentioning that the parallel architecture can be viewed as a special case of the partially serialized case when par_sz equals n. However, the two cases have different architectures as depicted in Figure 3.3 and Figure 3.4.

Our top-level architecture includes two parts: the control circuit and the datapath. The control circuit for the parallel architecture is used to provide the key constant from the LFSR as described in Section 3.1. However, an extra serialized counter in the control circuit is needed for the partially serialized architecture. The binary counter is used for our serialized counter. The datapath includes round function and key scheduling, and they are described as above for the parallel architecture and partially serialized architecture.

3.2.3 Hardware Evaluations of Simeck

We use three different compilation techniques in the Design Compiler to perform hardware optimizations: simple compile, compile ultra and compile ultra with clock gating. The simple compile option can provide us the hierarchical architectures of the design, and the areas of specific sub-modules. The compile ultra option can make deeper optimizations in a way of optimizing the entire module together, thereby reducing the area and power consumption significantly [26, 64]. The clock gating technique can further reduce the area and power consumption [26] We use all standard flip-flops without chip-enable signals for the parallel architecture. For the partially serialized architecture, only the LFSR, which generates the key constant in the control circuit, uses the flip-flops with chip-enable signals, which costs 5, 6, and 6 flip-flops for Simeck32/64, Simeck48/96, and Simeck64/128 respectively. Therefore, the clock gating optimization affects only a little of our results in terms of area and power consumption.

Table 3.3: Our Implementation Results of Simeck32/64, 48/96, 64/128 in 130nm

Simeck	Partial serial	CMOS 130nm					
		Area (GEs)		Max Frequency (MHz)	Throughput @100 KHz (Kbps)	Total Power @100 KHz (μW)	Total Power @2 MHz (μW)
		Before P&R	After P&R				
Simeck32/64	1-bit	505^*	549^*	292	5.6	0.417	8.3
	2-bit	$510^\dagger$	$555^\dagger$	288	11.1	0.431	8.5
	4-bit	$533^\dagger$	$579^\dagger$	312	22.2	0.463	9.2
	8-bit	$591^\dagger$	$642^\dagger$	289	44.4	0.523	10.4
	16-bit	695^*	756^*	526	88.9	0.606	11.9
Simeck48/96	1-bit	$715^\dagger$	$778^\dagger$	299	5.0	0.576	11.4
	2-bit	$722^\dagger$	$785^\dagger$	294	10.0	0.593	11.8
	3-bit	$731^\dagger$	$794^\dagger$	268	15.0	0.611	12.1
	4-bit	$748^\dagger$	$813^\dagger$	284	20.0	0.628	12.5
	6-bit	$770^\dagger$	$837^\dagger$	287	30.0	0.651	12.9
	8-bit	$801^\dagger$	$871^\dagger$	284	40.0	0.688	13.6
	12-bit	$858^\dagger$	$933^\dagger$	283	60.0	0.742	14.7
	24-bit	1027^*	1117^*	512	120.0	0.875	17.3
Simeck64/128	1-bit	924^*	1005^*	288	4.2	0.754	14.9
	2-bit	$933^\dagger$	$1015^\dagger$	303	8.3	0.778	15.4
	4-bit	$958^\dagger$	$1041^\dagger$	271	16.7	0.803	15.9
	8-bit	$1013^\dagger$	$1101^\dagger$	280	33.3	0.834	16.6
	16-bit	$1132^\dagger$	$1231^\dagger$	301	66.7	0.977	19.4
	32-bit	1365^*	1484^*	512	133.3	1.162	23.0

* Area obtained by using compile ultra only.

† Area obtained by using compile ultra and clock gating.

The ASIC implementation results of Simeck and SIMON in CMOS 130nm are shown in Table 3.3 and Table 3.4, and the corresponding results of Simeck and SIMON in CMOS 65nm are shown in Table 3.5 and Table 3.6.

Table 3.4: Our Implementation Results of SIMON32/64, 48/96, 64/128 in 130nm

SIMON	Partial serial	CMOS 130nm						
		Area (GEs)			Max Frequency (MHz)	Throughput @100 KHz (Kbps)	Total Power @100 KHz (μW)	Total Power @2 MHz (μW)
		Before P&R	After P&R	NSA Before P&R				
SIMON32/64	1-bit	517†	562†	523	331	5.6	0.421	8.3
	2-bit	532*	578*	535	306	11.1	0.439	8.7
	4-bit	563†	612†	566	283	22.2	0.479	9.5
	8-bit	623*	677*	627	367	44.4	0.540	10.7
	16-bit	715*	778*	722	456	88.9	0.645	12.8
SIMON48/96	1-bit	733†	796†	739	258	5.0	0.579	11.5
	2-bit	745†	810†	750	289	10.0	0.601	11.9
	3-bit	756†	822†	763	291	15.0	0.615	12.2
	4-bit	778†	846†	781	287	20.0	0.642	12.7
	6-bit	800†	869†	804	289	30.0	0.670	13.3
	8-bit	833†	905†	839	238	40.0	0.706	13.9
	12-bit	895†	973†	898	307	60.0	0.777	15.4
	24-bit	1055*	1147*	1062	467	120.0	0.929	18.4
SIMON64/128	1-bit	944†	1026†	958	225	4.2	0.762	15.1
	2-bit	955†	1038†	968	244	8.3	0.780	15.4
	4-bit	988†	1074†	1000	290	16.7	0.818	16.2
	8-bit	1043†	1134†	1057	296	33.3	0.866	17.2
	16-bit	1174†	1276†	1185	293	66.7	1.024	20.3
	32-bit	1403*	1524*	1417	465	133.3	1.239	24.6

* Area obtained by using compile ultra only.

† Area obtained by using compile ultra and clock gating.

We provide the best area results before and after the Place and Route phase using compile ultra or compile ultra plus clock gating. These results can be used for comparing with other ciphers or for practical purpose. The maximum frequency corresponding with the best optimization technique is given and it is calculated by using the critical path. The calculated throughput is based on the latency in our architectures and it is the same as SIMON. The difference of the total power consumption among the three different optimizations is marginal. Therefore, we provide the total power consumption using compile ultra at both 100 KHz and 2 MHz. 100 KHz is typical for benchmarking purpose and 2 MHz is used for the passive RFID tags in practice.

For the total power consumption at 100 KHz, it is larger in CMOS 65nm than that in CMOS 130nm. The reason is that because the 100 KHz operating frequency is so small, the static power consumption dominates the total power consumption. However, the static power consumption is larger in CMOS 65nm than that in CMOS 130nm in this case. The opposite case is true for the

Table 3.5: Our Implementation Results of Simeck32/64, 48/96, 64/128 in 65nm

Simeck	Partial Serial	CMOS 65nm Area (GEs) Before P&R	After P&R	Max Frequency (MHz)	Throughput @100 KHz (Kbps)	Total Power @100 KHz (μW)	Total Power @2 MHz (μW)
Simeck32/64	1-bit	454*	488*	1754	5.6	1.292	5.5
	2-bit	465†	500†	1428	11.1	1.311	5.6
	4-bit	494†	531†	1388	22.2	1.376	5.9
	8-bit	550*	592*	1250	44.4	1.512	6.4
	16-bit	644*	692*	1428	88.9	1.716	6.8
Simeck48/96	1-bit	645†	693†	1562	5.0	1.805	7.8
	2-bit	656†	706†	1538	10.0	1.825	8.0
	3-bit	663†	712†	1282	15.0	1.857	8.4
	4-bit	686†	738†	1333	20.0	1.886	8.2
	6-bit	701†	753†	1282	30.0	1.919	8.4
	8-bit	732†	787†	1388	40.0	2.009	8.8
	12-bit	794*	854*	1219	60.0	2.212	9.3
	24-bit	951*	1022*	2325	120.0	2.44	9.6
Simeck64/128	1-bit	828*	891*	1369	4.2	2.304	10.2
	2-bit	838†	901†	1408	8.3	2.325	10.3
	4-bit	869†	935†	1098	16.7	2.372	10.5
	8-bit	918†	987†	1190	33.3	2.492	10.9
	16-bit	1042*	1121*	1086	66.7	2.869	12.3
	32-bit	1263*	1358*	1282	133.3	3.316	13.1

* Area obtained by using compile ultra only.

† Area obtained by using compile ultra and clock gating.

total power consumption at 2 MHz.

Besides having a very small area, our another observation is that most part of the area for all the architectures are built of the sequential logics, especially for the fully serialized architecture. Take Simeck32/64 for example. 86%, 85%, 82%, 76%, and 70% of the entire area are sequential logics for the cases that par_sz equals 1, 2, 4, 8, and 16 respectively. From the data provided, we can obtain that the fully serialized architecture is built of about 90% sequential logics. Similar conclusions can be obtained for Simeck48/96 and Simeck64/128.

We provide a range of options between the area, throughput, and power consumption in our ASIC implementations. Taking Simeck32/64 in CMOS 130nm for illustration, we can achieve a throughput of 5.6 Kbps at the area cost of 505 GEs (before the Place and Route) and 549 GEs (after the Place and Route) with the power consumption of 0.417 μW at 100 KHz. However, a two-fold throughput (11.1 Kbps) can be obtained with only 5 and 6 extra GEs (before and after the Place and Route respectively), and 0.014 μW at 100 KHz extra power consumption. With

Table 3.6: Our Implementation Results of SIMON32/64, 48/96, 64/128 in 65nm

SIMON	Partial Serial	CMOS 65nm Area (GEs) Before P&R	CMOS 65nm Area (GEs) After P&R	Max Frequency (MHz)	Throughput @100 KHz (Kbps)	Total Power @100 KHz (μW)	Total Power @2 MHz (μW)
SIMON32/64	1-bit	466*	501*	1428	5.6	1.311	5.6
	2-bit	476*	512*	1562	11.1	1.331	5.7
	4-bit	506*	544*	1408	22.2	1.381	5.9
	8-bit	570*	613*	1075	44.4	1.585	6.8
	16-bit	666*	716*	2222	88.9	1.751	6.8
SIMON48/96	1-bit	661†	711†	1204	5.0	1.812	7.9
	2-bit	670†	720†	1136	10.0	1.889	9.5
	3-bit	682†	733†	1086	15.0	1.86	8.1
	4-bit	699†	752†	1041	20.0	1.915	8.3
	6-bit	724†	779†	1369	30.0	1.962	8.5
	8-bit	757†	814†	1282	40.0	2.122	9.0
	12-bit	819*	881*	1176	60.0	2.305	9.7
	24-bit	982*	1056*	2222	120.0	2.542	9.9
SIMON64/128	1-bit	845†	908†	1282	4.2	2.336	10.2
	2-bit	858†	922†	1265	8.3	2.366	10.4
	4-bit	887†	954†	1250	16.7	2.423	10.6
	8-bit	944†	1015†	1265	33.3	2.577	11.2
	16-bit	1076*	1156*	1176	66.7	3.068	12.8
	32-bit	1305*	1403*	1694	133.3	3.398	13.4

* Area obtained by using compile ultra only.
† Area obtained by using compile ultra and clock gating.

more extra area and power consumption, we can get even higher throughput.

3.3 Results Comparison between Simeck and SIMON

We compare our area results before the Place and Route of Simeck and SIMON in CMOS 130nm with the SIMON results of the NSA researchers [11]. This is because the NSA researchers only provide the area results before the Place and Route in CMOS 130nm. The comparison is shown in Figure 3.6. We can observe that our SIMON results are all smaller than that of NSA's results, and our Simeck results are even smaller than SIMON for all the cases shown in Figure 3.6.

From the theoretical point of view, Simeck is designed to have a smaller area due to the following considerations: the simplified key schedule, the simplified LFSR to generate the key constant, and the decreased shift numbers in the round function. It is worth noting that the

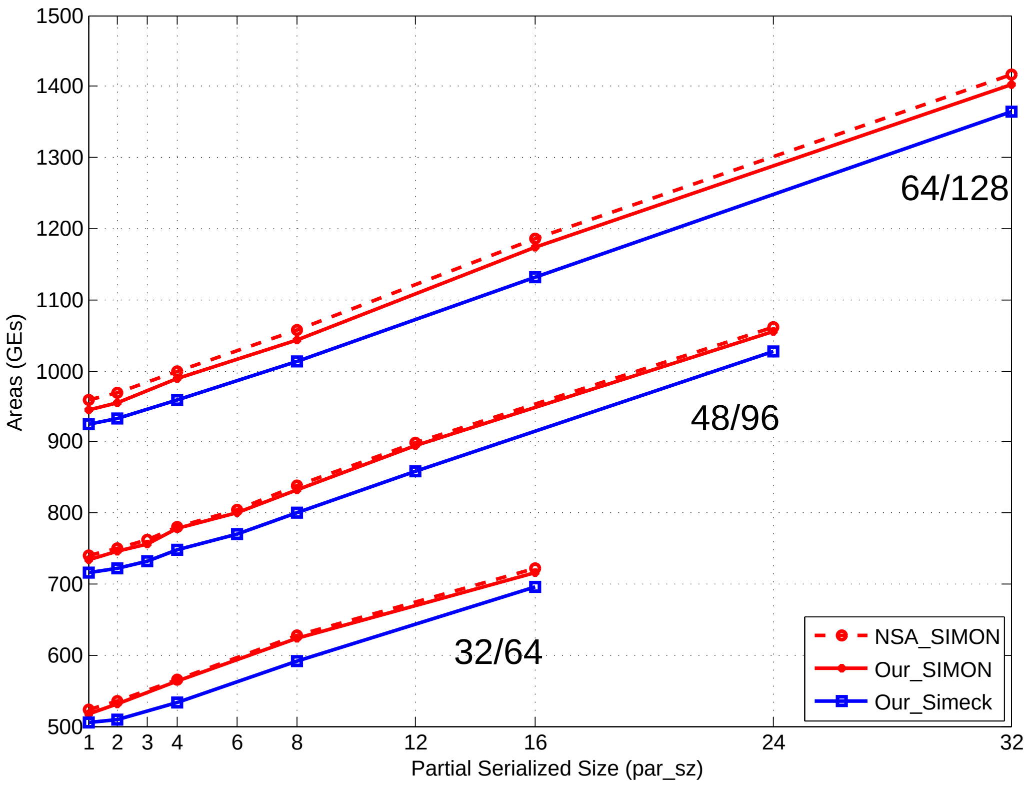


Figure 3.6: Comparison of Areas (before the Place and Route) between the Implementation Results of the NSA Researchers' and Ours in CMOS 130nm

decreased shift numbers do not affect any area in the parallel architecture, and it only affect the area in the partially serialized architecture.

The construction of the combinational circuit in the key schedule of SIMON32/64, 48/96, 64/128 and Simeck32/64, 48/96, 64/128 in the parallel architecture are shown as follows:

SIMON	$(2n+3)$ XOR + $(n-2)$ XNOR
Simeck	$(n+3)$ XOR + $(n-2)$ XNOR + n AND

In general, one XOR gate is larger than one AND gate. Therefore, the key schedule of SIMON is larger than that of Simeck. The LFSR used to generate the key constants for SIMON32/64 is defined by the primitive polynomial $X^5+X^4+X^2+X+1$, and the LFSRs for SIMON48/96 and SIMON64/128 are defined by $X^5+X^3+X^2+X+1$. They are all 2 XOR gates (4 GEs) bigger than the ones used in corresponding Simeck, as described in Section 3.1. The decreased shift numbers of the round function and key schedule reduce 1 MUX for the inputs to the combinational circuits of the round function and the key schedule respectively (2 MUXes in total, $2 \cdot 2.25$ GEs/MUX = 4.5 GEs), and also some logics to select the MUXes.

Table 3.7: Breakdown of the Implementation Results for Simeck before the Place and Route in 130nm

Components		Simeck32/64 (130nm) Parallel (GEs)	Simeck32/64 (130nm) Fully Serialized (GEs)	SIMON32/64 (130nm) Parallel (GEs)	SIMON32/64 (130nm) Fully Serialized (GEs)
Control		31	71	35	75
Datapath	Round_combinational Circuit	112	7	112	7
	Key_combinational Circuit	80	5	96	8
	Sequential + MUXes	474	434	474	443
Totals	Compile simple	697	517	717	533
	Compile ultra	695	505	717	520
	Compile ultra + clock gating	695	506	715	517

From the practical point of view, we break down the area results before the Place and Route in CMOS 130nm for Simeck32/64, and SIMON32/64 in our implementations, as shown in Table 3.7. For parallel architectures, the differences of the control circuits and the key combinational circuits between Simeck32/64 and SIMON32/64 are 4 GEs (key constant) and 16 GEs respectively. The results are almost the same as the theoretical analysis. For the fully serialized architecture, the control circuit is reduced by 4 GEs (key constant), the key combinational circuit (dashed box in Figure 4) is reduced by 3 GEs, and the 2 MUXes plus the MUXes selection circuitry are reduced by 9 GEs for Simeck32/64 (i.e., a total saving of 16 GEs), compared to that of SIMON32/64. Therefore, the practical results match the theoretical analysis. Simeck is smaller than SIMON for both parallel architecture and partially serialized architecture.

The main area cost for SIMON comes from the registers storing the message block and the key. In order to design a smaller cipher than SIMON, we can reduce the areas of only the round function, key schedule, key constant, and multiplexers. For fully serialized architecture of SIMON32/64 (see Table 3.7), the combined area of these blocks is 34.5 GEs (7 + 8 + 6 + 6 MUX $\cdot$ 2.25/MUX), which accounts for only about 6.4% (34.5/533) of the total area. Simeck32/64 reduces this by 16 GEs, a saving of more than 46%. This reduction leads to 2.3% smaller total area in comparison to our implementations of SIMON32/64 in CMOS 130nm, and 3.4% smaller in comparison to the original SIMON32/64 results (see Tables 3.3 and 3.4). Similarly, the fully serialized architectures of Simeck48/96, 64/128 are 2.5%, 2.1%, respectively, smaller than our implementations of SIMON48/96, 64/128 and they are 3.3% and 3.5%, respectively, smaller than the original implementation results of SIMON48/96, 64/128 in CMOS 130nm (see Tables 3.3 and 3.4). For the parallel architectures of SIMON, these blocks consume a larger fraction (about 29%) of the total area (see Table 3.7). Simeck32/64, 48/96, 64/128 achieve the saving of 3.7%, 3.3%, and 3.7% respectively, compared to the original results of SIMON32/64, 48/96, 64/128 (see Tables 3.3 and 3.4). The choice of the values of the shift numbers plays a significant role in the area reduction of the partially serialized architecture. Because the parallel architecture does not contain the MUXes for the inputs to the combinational circuit (dashed box), the total area reduction is only slightly greater than that of the fully serialized architecture.

From Tables 3.3 and 3.4, we can also observe that the power consumption of Simeck is smaller than SIMON for all the cases in CMOS 130nm using the same optimizations. This is easy to understand because the area of Simeck is smaller than that of SIMON. This conclusion also holds for CMOS 65nm in Tables 3.5 and 3.6.

Overall, Simeck is smaller than SIMON in terms of area and power consumption in both CMOS 130nm and CMOS 65nm techniques.

3.4 Comparisons with Other Lightweight Block Ciphers

In Section 3.2.3, we offer a wide range of options between area, throughput, and power consumption for the implementations of Simeck. All the Simeck's family members can meet our security, hardware, and applications design goals. We compare our results to the previous constructions with comparable block sizes and key sizes as given in Table 3.8. Table 3.8 gives our smallest area results for all the instances of Simeck from before and after the Place and Route (P&R) phase in CMOS 130nm and CMOS 65nm ASICs. In addition, the corresponding throughput and power consumption at 100 KHz after the Place and Route are also provided. In particular, Table 3.8 presents our hardware implementation results of SIMON which cost less area than the original results in [11]. In addition to the analysis of the comparison between Simeck and SIMON in CMOS 130nm in Section 3.3, the smallest Simeck32/64, Simeck48/96, and simeck64/128 in CMOS 65nm are 2.6%, 2.4%, and 2.0% smaller respectively than our implementations of the corresponding SIMON as shown in Table 3.8. Moreover, with only a little extra area (GEs) and power consumption, we can increase Simeck's throughput a lot.

Furthermore, the area of Simeck is smaller than other lightweight ciphers (EPCBC, LED, PRESENT) with the same block size and key size as shown in Table 3.8. The area of Simeck48/96 is 731 GEs (before Place and Route Phase) when the throughput is 15.0 Kbps, and that of Simeck64/128 is 958 GEs when the throughput is 16.7 Kbps (see Table 3.3). Therefore, even when compared with the similar throughput, the area of Simeck is smaller than that of EPCPC and PRESENT.

Table 3.8: Comparisons of Hardware Implementations of Lightweight Block Ciphers

Size	Algorithm	Tech (nm)	Area Before P&R (GEs)	Area After P&R (GEs)	Throughput @100KHz (Kbps)	Power @100KHz (μW)	Source
32/64	SIMON	130	523	-	5.6	-	[11]
	SPECK		580	-	4.2	-	[11]
	SIMON		**517**	**562**	5.6	0.421	**here**
	Simeck		**505**	**549**	5.6	0.417	**here**
	SIMON	65	**466**	**501**	5.6	1.311	**here**
	Simeck		**454**	**488**	5.6	1.292	**here**
48/96	SIMON	130	739	-	5.0	-	[11]
	SPECK		794	-	4.0	-	[11]
	SIMON		**733**	**796**	5.0	0.579	**here**
	Simeck		**715**	**778**	5.0	0.576	**here**
	SIMON	65	**661**	**711**	5.0	1.812	**here**
	Simeck		**645**	**693**	5.0	1.805	**here**
	EPCBC	180	1008	-	12.1	-	[117]
64/128	SIMON	130	958	-	4.2	-	[11]
	SPECK		966	-	3.4	-	[11]
	SIMON		**944**	**1026**	4.2	0.762	**here**
	Simeck		**924**	**1005**	4.2	0.754	**here**
	SIMON	65	**845**	**908**	4.2	2.336	**here**
	Simeck		**828**	**891**	4.2	2.304	**here**
	LED	180	1265	-	3.4	-	[49]
	PRESENT		1339	-	12.1	-	[117]
	SKINNY		1172	-	2.03	-	[14]

3.5 Summary

In this chapter, we have presented Simeck, a new family of lightweight block ciphers. Simeck is very suitable for resource-constrained devices, such as passive RFID tags. We have provided an extensive exploration for different hardware architectures in order to make a balance between area, throughput, and power consumption for SIMON and Simeck in both CMOS 130nm and CMOS 65nm technologies. We have shown that it is possible to design a smaller cipher than SIMON in terms of area and power consumption. Moreover, we have improved the hardware implementations of SIMON given in the original paper. In conclusion, all of the instances in the Simeck family can meet the area, power consumption, and throughput requirements in the passive RFID tags and they are promising candidates for resource-constrained devices.

Chapter 4

Design Space Exploration of the Lightweight Stream Cipher WG-8

WG-8 [35] is a lightweight instance of the WG stream cipher family [85], which inherits the good randomness and cryptographic properties of the WG stream cipher family.

The first hardware architecture for implementing WG-29 was described in [85], and the hardware design of MOWG, a multi-bit output variant of the original WG cipher, was proposed in [67] using signal reuse as well as pipelining with reuse techniques. The hardware implementation of the ultra-lightweight instance WG-5 has been reported in the context of passive RFID applications [7]. Recently, a new efficient hardware design for WG-29 was proposed in [30], using the nice property of the trace of product of two finite field elements with type-II optimal normal basis (ONB) representations. A compact hardware implementation of large size stream cipher WG-16 using tower field constructions with excellent performance was also proposed [37]. In addition, the S-box in AES based on the tower field constructions [101] achieved quite good performance. In this chapter, we investigate different tower field constructions and explore the design space for WG-8 on FPGA and ASIC platforms in terms of area, speed, and power consumption.

The main focus of this chapter is to explore different constructions in $\mathbb{F}_{2^8}$ for WG-8 stream cipher and analyze the effect on hardware architectures and the area, power, and performance of hardware implementations on low-cost FPGA and ASICs. Four different hardware architectures have been proposed in this work. Multiplication is the most expensive operation and its area is affected by the choice of basis. The first architecture directly employs an 8×8 constant array over $\mathbb{F}_{2^8}$. With the goal of replace multiplication by small constant arrays, the second one is based on the tower construction $\mathbb{F}_{(2^4)^2}$ together with small 4×4 constant arrays for arithmetic in $\mathbb{F}_{2^4}$, due to the existence of primitive element. The third architecture is slightly different from the second one, the multiplication is small due to the usage of a type-I ONB for efficient computations in $\mathbb{F}_{2^4}$. Finally, the fourth architecture takes advantage of the tower construction $\mathbb{F}_{((2^2)^2)^2}$ coupled with a nice property for computing the trace of the product of two finite field elements.

We propose a novel hybrid design with the parallel width from one to eleven for each proposed architecture. With additional hardware resources, the parallel implementations can achieve a high throughput without incurring significantly critical path delay.

The experimental results show that the direct constant array based hardware architecture is best in terms of clock speed, area, and power consumption, when compared to the tower field arithmetic based approaches. The main reason is due to the small field size as well as the relatively complicated architecture of WG-8 permutation/transformation module. Although the tower field based approaches for WG-8 are not efficient, the proposed architecture and extensive experimental results still provide valuable guidance for efficient hardware implementations of medium or large instances of the WG stream cipher family.

This chapter is organized as follows. In Section 4.1, we give the description and hardware architecture of WG-8. Four different design strategies for the WG-8 transformation module and the hybrid design architectures for WG-8 are presented in Section 4.2. In Section 4.3, we give the design methods to calculate the multiplication by ω module in WG-8 design. The hardware implementations of WG-8, including the Finite State Machine (FSM) and the FPGA and ASIC results are presented in Section 4.4. Before concluding this chapter, we give the results analysis and comparison with other lightweight stream ciphers in Section 4.5.

4.1 Description of WG-8

The detailed description of WG-8 is given in this section.

4.1.1 Parameters for WG-8

We define the terms and notations that will be used to describe the lightweight stream cipher WG-8 and its hardware implementations.

- $p(x) = x^8 + x^4 + x^3 + x^2 + 1$, a primitive polynomial of degree 8 over $\mathbb{F}_2$.
- $\mathbb{F}_{2^8}$, the extension field of $\mathbb{F}_2$ defined by the primitive polynomial $p(x)$ with 2^8 elements. Each element in $\mathbb{F}_{2^8}$ is represented as an 8-bit binary vector. Let ω be a primitive element of $\mathbb{F}_{2^8}$ such that $p(\omega) = 0$.

- $\mathrm{Tr}(x) = x + x^2 + x^{2^2} + x^{2^3} + x^{2^4} + x^{2^5} + x^{2^6} + x^{2^7}$, the trace function from $\mathbb{F}_{2^8} \to \mathbb{F}_2$.
- $l(x) = x^{20} + x^9 + x^8 + x^7 + x^4 + x^3 + x^2 + x + \omega$, the feedback polynomial of LFSR (which is also a primitive polynomial over $\mathbb{F}_{2^8}$).
- $q(x) = x + x^{2^3+1} + x^{2^6+2^3+1} + x^{2^6-2^3+1} + x^{2^6+2^3-1}$, a permutation polynomial over $\mathbb{F}_{2^8}$.
- WGP-8$(x^d) = q(x^d + 1) + 1$, the WG-8 permutation with decimation d from $\mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$, where d is coprime to $2^8 - 1$.
- WGT-8$(x^d) = \mathrm{Tr}($WGP-8$(x^d)) = \mathrm{Tr}(q(x^d + 1))$, the WG-8 transformation with decimation d from $\mathbb{F}_{2^8} \to \mathbb{F}_2$, where d is coprime to $2^8 - 1$.
- Polynomial basis (PB) of $\mathbb{F}_{2^8}$: A polynomial basis of $\mathbb{F}_{2^8}$ over $\mathbb{F}_2$ is a basis of the form $\{1, \omega, \omega^2, \cdots, \omega^7\}$.
- Normal basis (NB) of $\mathbb{F}_{2^8}$: A normal basis of $\mathbb{F}_{2^8}$ over $\mathbb{F}_2$ is a basis of the form $\{\theta, \theta^2, \cdots, \theta^{2^7}\}$, i.e., a basis consisting of all the algebraic conjugates of a fixed element θ. Some basis facts when performing computations in $\mathbb{F}_{2^8}$ with normal basis are as follows:
  - The exponentiation of a field element v to the power 2^t $(t \geq 1)$ can be done by cyclically shifting v to the right by t positions.
  - If θ is a generator of the normal basis, then $\sum_{i=0}^{7} \theta^{2^i} = 1$. Therefore, the addition of a field element v with 1 can be done by a bitwise complement operation.
  - The trace of all basis element is one, i.e., $\mathrm{Tr}(\theta^{2^i}) = 1$ for $i = 0, 1, \cdots, 7$. Hence, the trace of a field element $v = \sum_{i=0}^{7} v_i \theta^{2^i}$ can be calculated as $\mathrm{Tr}(v) = \sum_{i=0}^{7} v_i$, i.e., the modulo-2 sum of its coordinates with respect to the normal basis.
- $\oplus_m$, the bitwise addition operator for two operands of m bits (i.e., XOR).
- $\odot_m$, the bitwise and operator for two operands of m bits (i.e., AND).
- $\otimes_m$, the multiplication operator for two operands over $\mathbb{F}_{2^m}$ or its isomorphic fields.
- 8×8 constant array, a constant array that contains 256 elements with 8-bit input and 8-bit output.
- 8×1 constant array , a constant array that contains 256 elements with 8-bit input and 1-bit output.
- 4×4 constant array, a constant array that contains 16 elements with 4-bit input and 4-bit output.

4.1.2 Overview of WG-8

WG-8 is a lightweight stream cipher with 80-bit key and 80-bit initial vector (IV), which can be regarded as a nonlinear filter generator over finite field $\mathbb{F}_{2^8}$. The stream cipher WG-8 consists of a 20-stage LFSR with the feedback polynomial $l(x)$ followed by a WG-8 transformation module with decimation 19, i.e., $d = 19$, which is different from [35], and operates in two phases, namely an initialization phase and a running phase.

Initialization Phase

The key/IV initialization phase of the stream cipher WG-8 is shown in Figure 4.1. Let the 80-bit secret key be $K = (K_0, \ldots, K_{79})_2$, the 80-bit IV be $IV = (IV_0, \ldots, IV_{79})_2$, and the internal states of LFSR be $S_0, \ldots, S_{19} \in \mathbb{F}_{2^8}$, where $S_i = (S_{i,0}, \ldots, S_{i,7})_2$ for $i = 0, \ldots, 19$. The key and IV initialization process is conducted as follows: $S_{2i} = (K_{8i}, \ldots, K_{8i+3}, IV_{8i}, \ldots, IV_{8i+3})_2$ and $S_{2i+1} = (K_{8i+4}, \ldots, K_{8i+7}, IV_{8i+4}, \ldots, IV_{8i+7})_2$ for $i = 0, \ldots, 9$.

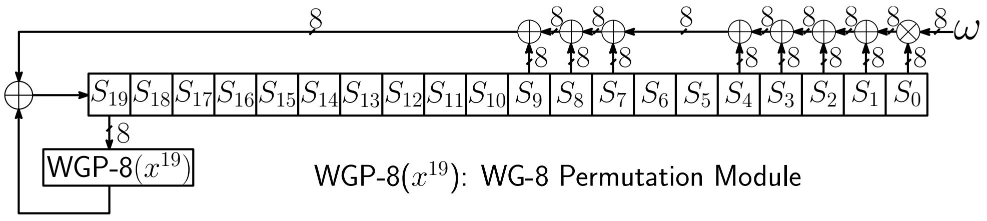

8
ω
S_{19} S_{18} S_{17} S_{16} S_{15} S_{14} S_{13} S_{12} S_{11} S_{10} S_9 S_8 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
WGP-8(x^{19})
WGP-8(x^{19}): WG-8 Permutation Module

Figure 4.1: The Initialization Phase of the Stream Cipher WG-8

Once the LFSR is loaded with the key and IV, the apparatus runs for 40 clock cycles. During each clock cycle, the 8-bit internal state S_{19} passes through the nonlinear WG-8 permutation with decimation 19 (i.e., the WGP-8(x^{19}) module) and the output is used as the feedback to update the internal state of the LFSR. The LFSR update follows the recursive relation:

$$\begin{aligned} S_{k+20} \;&=\; (\omega \otimes_8 S_k) \oplus_8 S_{k+1} \oplus_8 S_{k+2} \oplus_8 S_{k+3} \oplus_8 S_{k+4} \oplus_8 \\ &\quad\; S_{k+7} \oplus_8 S_{k+8} \oplus_8 S_{k+9} \oplus_8 \text{WGP-8}(S_{k+19}^{19}), \end{aligned}$$

where $0 \le k < 40$. After the key/IV initialization phase, the stream cipher WG-8 goes into the running phase and 1-bit keystream is generated after each clock cycle.

Running Phase

The running phase of the stream cipher WG-8 is illustrated in Figure 4.2. During the running

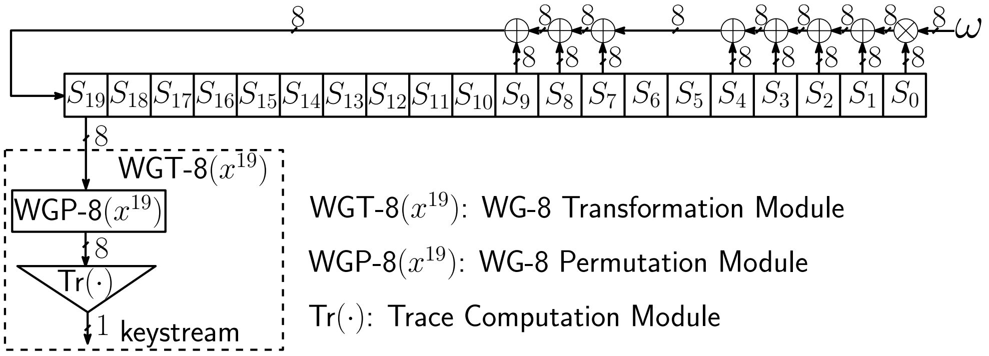

8
ω
S_{19} S_{18} S_{17} S_{16} S_{15} S_{14} S_{13} S_{12} S_{11} S_{10} S_9 S_8 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
WGT-8(x^{19})
WGP-8(x^{19})
Tr(·)
1 keystream
WGT-8(x^{19}): WG-8 Transformation Module
WGP-8(x^{19}): WG-8 Permutation Module
Tr(·): Trace Computation Module

Figure 4.2: The Running Phase of the Stream Cipher WG-8

phase, the 8-bit internal state S_{19} passes through the nonlinear WG-8 transformation with decimation 19 (i.e., the WGT-8(x^{19}) module) and the output is the keystream. Note that the only feedback in the running phase is within the LFSR and the recursive relation for updating the LFSR is given below:

$$S_{k+20} \quad = \quad (\omega \otimes_8 S_k) \oplus_8 S_{k+1} \oplus_8 S_{k+2} \oplus_8 S_{k+3} \oplus_8 S_{k+4} \oplus_8 S_{k+7} \oplus_8 S_{k+8} \oplus_8 S_{k+9},$$

where $k \geq 40$. The WG-8 transformation module WGT-8(x^{19}) comprises of two sub-modules: a WG-8 permutation module WGP-8(x^{19}) followed by a trace computation module Tr(·). While the WGP-8(x^{19}) module permutes elements over $\mathbb{F}_{2^8}$, and the Tr(·) module transforms an 8-bit input to one keystream bit.

4.1.3 Hardware Architecture

The stream cipher WG-8 is a lightweight keystream generator that consists of four main components: the 20-stage LFSR over finite field $\mathbb{F}_{2^8}$, the WG-8 transformation module WGT-8(x^{19}), the multiplication by ω module, and the finite state machine. A high-level hardware architecture that contains both initialization and running phases is shown in Figure 4.3. Note that the WG-8 transformation module WGT-8(x^{19}) has been further split into the WG-8 permutation module WGP-8(x^{19}) and the trace computation module Tr(·) in order to accommodate the initialization phase.

Under the control of the finite state machine, the 80-bit key and the 80-bit IV will be first loaded into LFSR within 20 clock cycles through the pin DIN. After loading the required key and IV, the initialization phase will be performed in the next 40 clock cycles without any output. The running phase will start from the 61 clock cycle and one bit keystream will be output in every clock cycle.

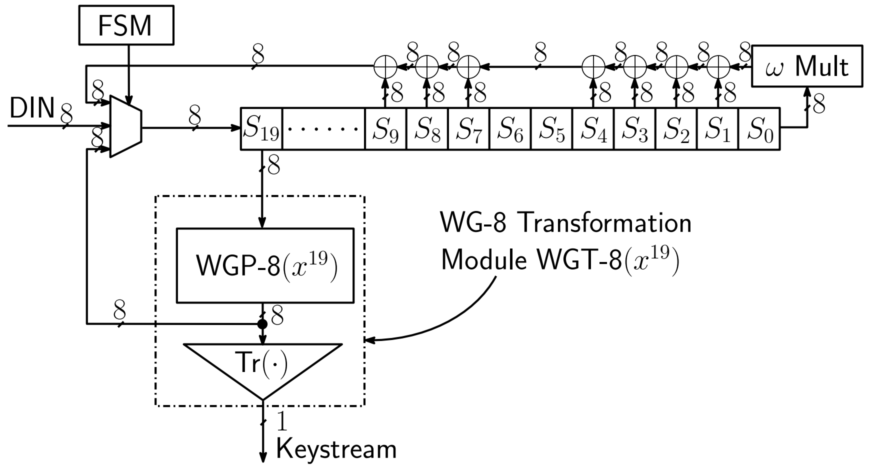

FSM
DIN
ω Mult
S_{19} S_9 S_8 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
8
WG-8 Transformation Module WGT-8(x^{19})
WGP-8(x^{19})
Tr$(\cdot)$
1
Keystream

Figure 4.3: The High-Level Hardware Architecture of the Stream Cipher WG-8

The above procedure characterizes a simple design for implementing the WG-8 stream cipher since only one bit is output per clock cycle during the running phase. To further increase the throughput, we also consider a parallel approach that enables the WG-8 core to output several bits per clock cycle during the running phase. Due to the tap position distribution of the LFSR's feedback polynomial in WG-8, the maximum parallel width is eleven in this case. Although a parallel approach can be applied to the running phase, the initialization phase has to be implemented in a serial fashion, which makes our hardware architecture a hybrid design. In the following sections, we will elaborate the hardware design for the main components of WG-8.

4.2 Design Strategies for the WG-8 Transformation Module

In this section, we discuss the different design strategies for the WG-8 transformation module. The hardware implementation of the WGT-8(x^{19}) module involves the arithmetic (i.e., addition, multiplication, inversion, and exponentiation) over finite field $\mathbb{F}_{2^8}$. The most complicated component is the WGP-8(x^{19}) module that can be implemented using either an 8×8 constant array or the tower field (TF) arithmetic. The $\mathrm{Tr}(\cdot)$ module can also be realized either by an 8×1 constant array or several XOR gates in a straightforward manner.

Since a number of exponentiations of the form 2^t $(t \geq 1)$ need to be performed for computing WGP-8(x^{19}), and we employ normal basis to implement fast exponentiation in $\mathbb{F}_{2^8}$ by using cyclic shift operations. Therefore, the conversion matrices between the TF and NB representations will also be deduced.

In the following subsections, we use four different methods for implementing the WGT-8 module, which are the constant array based method and three tower field methods.

4.2.1 Using Constant Array

Depending on the bases used, one can precompute the WGP-8 module with decimation 19

$$\text{WGP-8}(x^{19}) = q(x^{19} + 1) + 1$$

as well as the trace function

$$\text{Tr}(x) = x + x^2 + x^{2^2} + x^{2^3} + x^{2^4} + x^{2^5} + x^{2^6} + x^{2^7}$$

for all elements $x \in \mathbb{F}_{2^8}$. Hence, an 8×8 constant array (CA) can be generated to store the results for the WGP-8 module, and an 8×1 constant array (CA) can be used to keep the results for the trace function. Moreover, an 8×1 constant array (CA) can also be generated for the entire WGT-8 module by using the relation $\text{WGT-8}(x^{19}) = \text{Tr}(\text{WGP-8}(x^{19}))$. Thus, the area of the WGT-8 module and the area of the trace function are the same, and they are much smaller than that of the WGP-8 module. We will use these constant arrays to implement WGT-8 and WGP-8 in our hardware implementations of WG-8.

4.2.2 Using Tower Field 1

In this subsection, we first construct a finite field $\mathbb{F}_{2^4}$ and then construct $\mathbb{F}_{(2^4)^2}$ for the WGT-8 module. We define this tower construction as Tower Field 1 (TF 1).

Tower Construction $\mathbb{F}_{(2^4)^2}$ and Its Arithmetic

To obtain the tower construction $\mathbb{F}_{(2^4)^2}$, we first construct $\mathbb{F}_{2^4}$ by using an irreducible polynomial $e_1(X)$ of degree 4 over $\mathbb{F}_2$, and then construct $\mathbb{F}_{(2^4)^2}$ by using a certain irreducible polynomial $f_1(X)$ of degree 2 over $\mathbb{F}_{2^4}$. Note that the efficiency of the arithmetic over $\mathbb{F}_{(2^4)^2}$ is closely related to the selection of the irreducible polynomials as well as the bases of the towerings. In this case, we adopt mixed bases for two towerings. More specifically, we use $e_1(X) = X^4 + X^3 + 1$ with its polynomial basis $\{1, \alpha, \alpha^2, \alpha^3\}$ for $\mathbb{F}_{2^4}$ and $f_1(X) = X^2 + X + \alpha$ with its normal basis $\{\beta, \beta^{16}\}$ for $\mathbb{F}_{(2^4)^2}$, where $\alpha \in \mathbb{F}_{2^4}$ and $\beta \in \mathbb{F}_{(2^4)^2}$ are zeros of the polynomials $e_1(X)$ and $f_1(X)$, respectively. Moreover, the irreducible polynomial $e_1(X)$ here is also a primitive polynomial and

α is also a primitive element. According to the property of primitive polynomials, the elements of a finite field generated by them can be represented using both the exponentiation and binary forms. Therefore, we can use two small constant arrays to conduct arithmetic operations in $\mathbb{F}_{2^4}$ (see below for details). We summarize the bases and the defining polynomials for the tower fields in Table 4.1. Furthermore, we choose $\theta = \omega^{91}$ to construct the normal basis $\{\theta, \theta^2, \cdots, \theta^{2^6}, \theta^{2^7}\}$ in $\mathbb{F}_{2^8}$ defined by the polynomial $p(x)$. The values of α, β, θ are chosen in order to minimize the Hamming weight of conversion matrices. As a result, the number of XOR gates used for implementing the conversion matrices will be quite small.

Table 4.1: Tower Construction $\mathbb{F}_{(2^4)^2}$

Finite Field	Basis	Defining Polynomial
$\mathbb{F}_{2^8} \cong \mathbb{F}_{(2^4)^2}$	NB, $\{\beta, \beta^{16}\},\ \beta = \omega^7$	$f_1(X) = X^2 + X + \alpha$
$\mathbb{F}_{2^4} \cong \mathbb{F}_{(2)^4}$	PB, $\{1, \alpha, \alpha^2, \alpha^3\},\ \alpha = \omega^{119}$	$e_1(X) = X^4 + X^3 + 1$

Conversion matrices between TF *and* NB *representations.* For every element v in the tower field $\mathbb{F}_{(2^4)^2}$, the TF representation can be described as follows:

$$\begin{aligned} v &= v_0\beta + v_1\beta^{16}, v_0, v_1 \in \mathbb{F}_{2^4}, \\ &= (v_{000} + v_{001}\alpha + v_{010}\alpha^2 + v_{011}\alpha^3)\beta + (v_{100} + v_{101}\alpha + v_{110}\alpha^2 + v_{111}\alpha^3)\beta^{16}, \end{aligned}$$

where $v_{000}, v_{001}, v_{010}, v_{011}, v_{100}, v_{101}, v_{110}, v_{111} \in \mathbb{F}_2$. Using the relations $\alpha = \omega^{119}, \beta = \omega^7$, we can obtain

$$\begin{aligned} v = {} & v_{000}\omega^7 + v_{001}\omega^{126} + v_{010}\omega^{245} + v_{011}\omega^{109} + \\ & v_{100}\omega^{112} + v_{101}\omega^{231} + v_{110}\omega^{95} + v_{111}\omega^{214}. \end{aligned} \tag{4.1}$$

Writing each $\omega^i, i \in \{7, 126, 245, 109, 112, 231, 95, 214\}$ in terms of the normal basis gives a matrix $\mathbf{M}_{T\to N}$ that converts a finite field element from TF representation to NB representation. Hence, we can also get the inverse matrix $\mathbf{M}_{N\to T}$, which converts a finite field element back. In addition, the hamming weight for the matrix $\mathbf{M}_{T\to N}$ and matrix $\mathbf{M}_{N\to T}$ are 30 and 22, respectively. The $\mathbf{M}_{T\to N}$ and $\mathbf{M}_{N\to T}$ are given below:

$$\mathbf{M}_{T\to N} = \begin{pmatrix} 1&1&0&0&0&1&1&0\\ 0&1&1&0&1&0&1&0\\ 1&1&0&1&0&0&1&0\\ 0&1&1&0&1&0&0&0\\ 0&1&1&0&1&1&0&0\\ 1&0&1&0&0&1&1&0\\ 0&0&1&0&1&1&0&1\\ 1&0&0&0&0&1&1&0 \end{pmatrix}, \quad \mathbf{M}_{N\to T} = \begin{pmatrix} 0&1&0&0&1&0&0&1\\ 1&0&0&0&0&0&0&1\\ 0&0&0&0&0&1&0&1\\ 1&0&1&1&1&0&0&0\\ 1&0&0&1&0&1&0&0\\ 0&0&0&1&1&0&0&0\\ 0&1&0&1&0&0&0&0\\ 1&0&0&0&1&0&1&1 \end{pmatrix}.$$

Arithmetic operations in $\mathbb{F}_{2^4}$. The arithmetic in $\mathbb{F}_{2^4}$ is conducted with the aid of a 4×4 exponentiation table T_{exp} (see Table 4.2) and a 4×4 logarithm table T_{log} (see Table 4.3) below. While the table T_{exp} stores exponentiation $\alpha^i, i = 0, 1, \dots, 14$, and the table T_{log} keeps the exponent i for each $\alpha^i, i = 0, 1, \dots, 14$.

Table 4.2: Exponentiation Table T_{exp} in Hexadecimal Notation

i	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
α^i	8	4	2	1	9	D	F	E	7	A	5	B	C	6	3

Table 4.3: Logarithm Table T_{log} in Hexadecimal Notation

α^i	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
i	3	2	E	1	A	D	8	F	4	9	B	C	5	7	6

Let $A = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3$ and $B = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$ be two non-zero elements in $\mathbb{F}_{2^4}$, where $a_i, b_i \in \mathbb{F}_2, i = 0, 1, 2, 3$. We can perform the arithmetic in $\mathbb{F}_{2^4}$ as follows:

$$\begin{aligned} AB &= T_{exp}[(T_{log}[(a_0, a_1, a_2, a_3)] + T_{log}[(b_0, b_1, b_2, b_3)]) \bmod 15],\\ A^2 &= T_{exp}[(T_{log}[(a_0, a_1, a_2, a_3)] \ll 1) \bmod 15],\\ \alpha A &= T_{exp}[(T_{log}[(a_0, a_1, a_2, a_3)] + 1) \bmod 15]. \end{aligned}$$

Arithmetic operations in $\mathbb{F}_{(2^4)^2}$. Let $A = a_0\beta + a_1\beta^{16}$ and $B = b_0\beta + b_1\beta^{16}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{2^4}$. A multiplication AB in $\mathbb{F}_{(2^4)^2}$ is computed as follows:

$$\begin{aligned} AB &= (a_0\beta + a_1\beta^{16})(b_0\beta + b_1\beta^{16}),\\ &= (c\alpha + a_0b_0)\beta + (c\alpha + a_1b_1)\beta^{16}. \end{aligned}$$

where $c = (a_0 + a_1)(b_0 + b_1)$. For a non-zero element $A \in \mathbb{F}_{(2^4)^2}$, the squaring of A is calculated as follows :

$$\begin{aligned} A^2 &= (a_0\beta + a_1\beta^{16})^2 = a_0^2\beta^2 + a_1^2\beta^{32}, \\ &= [(a_0 + a_1)^2\alpha + a_0^2]\beta + [(a_0 + a_1)^2\alpha + a_1^2]\beta^{16}. \end{aligned}$$

Implementation of WGT-8(x^{19}) Module

For an element $x \in \mathbb{F}_{2^8}$, WGP-8(x^{19}) can be computed as follows:

$$\begin{aligned} \text{WGP-8}(x^{19}) &= q(x^{19} + 1) + 1, \\ &= y + y^{2^3+1} + y^{2^6}(y^{2^3+1} + y^{2^3-1}) + y^{2^3(2^3-1)+1} + 1, \end{aligned}$$

where $y = x^{19} + 1 = x^{2^4} \cdot x^2 \cdot x + 1$, and $y \in \mathbb{F}_{2^8}$. The WGT-8 module is combined by WGP-8 module and Tr$(\cdot)$ module, as described in section 4.1.3. Therefore, the WGT-8(x^{19}) can be computed as follows:

$$\begin{aligned} \text{WGT-8}(x^{19}) &= \text{Tr}(\text{WGP-8}(x^{19})), \\ &= \text{Tr}\left(y + y^{2^3+1} + y^{2^6}(y^{2^3+1} + y^{2^3-1}) + y^{2^3(2^3-1)+1}\right). \end{aligned}$$

Note that for this tower construction $\mathbb{F}_{(2^4)^2}$, 1 can be denoted by the vector $(1, 0, 0, 0, 1, 0, 0, 0)$. Therefore, the addition with 1 under the TF representation is equivalent to XORing with a constant 0x88, which only needs two NOT gates. Moreover, from Equation 4.1 we also obtain

$$\begin{aligned} \text{Tr}(v) = \text{Tr}(&v_{000}\omega^7 + v_{001}\omega^{126} + v_{010}\omega^{245} + v_{011}\omega^{109} + \\ &v_{100}\omega^{112} + v_{101}\omega^{231} + v_{110}\omega^{95} + v_{111}\omega^{214}), \\ = &v_{001} \oplus_1 v_{010} \oplus_1 v_{011} \oplus_1 v_{101} \oplus_1 v_{110} \oplus_1 v_{111}. \end{aligned}$$

Based on the above observations, the hardware architecture of the WGT-8(x^{19}) module is illustrated in Figure 4.4, where we list the output of each component and its basis representation. There are two outputs for the WGT-8(x^{19}) module, and the 8-bit WGP-8(x^{19}) output is used as the feedback for the initialization phase and the 1-bit WGT-8(x^{19}) output is used for the running phase. Furthermore, as shown in Figure 4.5, the exponentiation module $(\cdot)^{2^3-1}$ can be efficiently implemented by performing two squarings and two multiplications over the tower field $\mathbb{F}_{(2^4)^2}$.

4.2.3 Using Tower Field 2

The tower construction 2 (TF 2) is similar as the TF 1. In this case, the finite field $\mathbb{F}_{2^4}$ is firstly constructed and then $\mathbb{F}_{(2^4)^2}$.

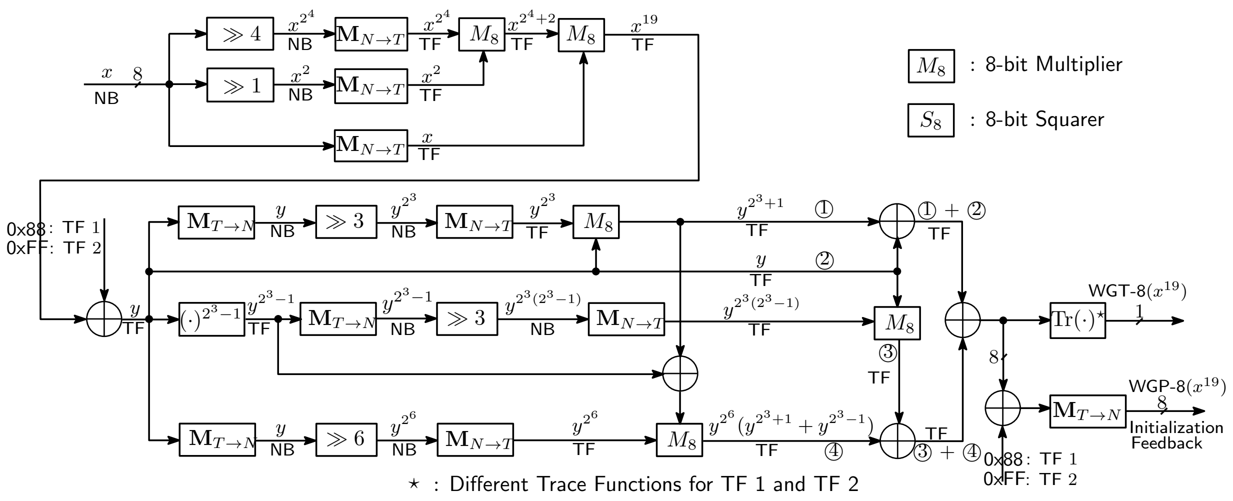


Figure 4.4: The Hardware Architecture of the WG-8 Transformation Module WGT-8(x^{19})

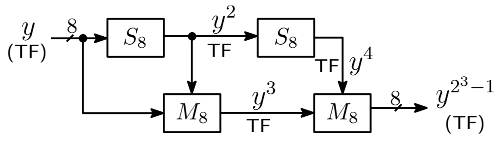


Figure 4.5: The Hardware Architecture of Module $(\cdot)^{2^3-1}$

Tower Construction $\mathbb{F}_{(2^4)^2}$ and Its Arithmetic

We use the same approach as that in TF 1 to build the tower construction $\mathbb{F}_{(2^4)^2}$, but with different irreducible polynomials. More specifically, we use $e_2(X) = X^4 + X^3 + X^2 + X + 1$ with its type-I optimal normal basis $\{\alpha, \alpha^2, \alpha^{2^2}, \alpha^{2^3}\}$ for $\mathbb{F}_{2^4}$ and $f_2(X) = X^2 + X + \alpha$ with its normal basis $\{\beta, \beta^{16}\}$ for $\mathbb{F}_{(2^4)^2}$, where $\alpha \in \mathbb{F}_{2^4}$ and $\beta \in \mathbb{F}_{(2^4)^2}$ are zeros of the polynomials $e_2(X)$ and $f_2(X)$, respectively. However, $f_2(X)$ is the same as $f_1(X)$, and we take a polynomial $e_2(X)$ with type-I optimal normal basis to conduct the arithmetic operations in $\mathbb{F}_{2^4}$. Where, $e_2(X)$ is only an irreducible polynomial but not a primitive polynomial. Therefore, the elements generated by $e_2(X)$ can only be represented by binary form, hence we cannot take advantage of the small constant arrays as in TF 1. However, upon our observations, the arithmetic operations in $\mathbb{F}_{2^4}$ using the type-I optimal normal basis is very efficient, where the coefficient of the multiplication is symmetric and simple (see below for details). Due to the nice property of the arithmetic calculations of $e_2(X)$, the area and power consumption will decrease significantly, compared to TF 1. Moreover, the normal basis $\{\theta, \theta^2, \cdots, \theta^{2^6}, \theta^{2^7}\}$ in this method is given by $\theta = \omega^{21}$. As we did before, the values of α, β, and θ are chosen in order to minimize the hamming weight of conversion matrices. As we can see later, the number of XOR gates used for implementing the conversion matrices is smaller than TF 1, and hence the total area will be decreased (see below for details).

Table 4.4: Tower Construction $\mathbb{F}_{(2^4)^2}$ with Normal Bases

Finite Field	Basis	Defining Polynomial
$\mathbb{F}_{2^8} \cong \mathbb{F}_{(2^4)^2}$	NB, $\{\beta, \beta^{16}\},\ \beta = \omega^{123}$	$f_2(X) = X^2 + X + \alpha$
$\mathbb{F}_{2^4} \cong \mathbb{F}_{(2)^4}$	NB, $\{\alpha, \alpha^2, \alpha^{2^2}, \alpha^{2^3}\},\ \alpha = \omega^{51}$	$e_2(X) = X^4 + X^3 + X^2 + X + 1$

Conversion matrices between TF and NB representations. For every element v in this tower field $\mathbb{F}_{(2^4)^2}$, the TF representation is as follows:

$$\begin{aligned} v &= v_0\beta + v_1\beta^{16},\ v_0, v_1 \in \mathbb{F}_{2^4}, \\ &= (v_{000}\alpha + v_{001}\alpha^2 + v_{010}\alpha^{2^2} + v_{011}\alpha^{2^3})\beta + (v_{100}\alpha + v_{101}\alpha^2 + v_{110}\alpha^{2^2} + v_{111}\alpha^{2^3})\beta^{16}, \end{aligned}$$

where $v_{000}, v_{001}, v_{010}, v_{011}, v_{100}, v_{101}, v_{110}, v_{111} \in \mathbb{F}_2$. Using the relations $\alpha = \omega^{51}$ and $\beta = \omega^{123}$, we can get the conversion matrices $\mathbf{M}_{T\to N}$, and $\mathbf{M}_{N\to T}$ with hamming weight 24 and 24, respectively. The $\mathbf{M}_{T\to N}$ and $\mathbf{M}_{N\to T}$ are given below:

$$\mathbf{M}_{T\to N} = \begin{pmatrix} 0&1&0&1&0&1&0&0 \\ 1&0&0&0&0&0&0&0 \\ 1&1&1&0&1&0&1&0 \\ 0&1&0&0&0&1&1&0 \\ 0&1&0&0&0&1&0&1 \\ 0&0&0&0&1&0&0&0 \\ 1&0&1&0&1&1&1&0 \\ 0&1&1&0&0&1&0&0 \end{pmatrix}, \quad \mathbf{M}_{N\to T} = \begin{pmatrix} 0&1&0&0&0&0&0&0 \\ 0&1&1&1&0&1&0&1 \\ 0&0&1&0&0&0&1&1 \\ 1&0&1&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0 \\ 0&1&0&1&0&1&1&1 \\ 0&0&1&1&0&0&1&0 \\ 0&0&1&0&1&0&1&0 \end{pmatrix}.$$

Arithmetic operations in $\mathbb{F}_{2^4}$. Thanks to the type-I optimal normal basis, the arithmetic in $\mathbb{F}_{2^4}$ has been directly implemented using logics in this case. Let $A = a_0\alpha + a_1\alpha^2 + a_2\alpha^{2^2} + a_3\alpha^{2^3}$ and $B = b_0\alpha + b_1\alpha^2 + b_2\alpha^{2^2} + b_3\alpha^{2^3}$ be two non-zero elements in $\mathbb{F}_{2^4}$, where $a_i, b_i \in \mathbb{F}_2, i = 0, 1, 2, 3.$ A multiplication AB in $\mathbb{F}_{2^4}$ is computed as follows:

$$\begin{aligned} AB &= (a_0\alpha + a_1\alpha^2 + a_2\alpha^{2^2} + a_3\alpha^{2^3}) \cdot (b_0\alpha + b_1\alpha^2 + b_2\alpha^{2^2} + b_3\alpha^{2^3}), \\ &= c_0\alpha + c_1\alpha^2 + c_2\alpha^{2^2} + c_3\alpha^{2^3}, \end{aligned}$$

where

$$\begin{aligned}
c_0 &= a_0b_2 + a_1(b_2+b_3) + a_2(b_0+b_1) + a_3(b_1+b_3),\\
c_1 &= a_1b_3 + a_2(b_3+b_0) + a_3(b_1+b_2) + a_0(b_2+b_0),\\
c_2 &= a_2b_0 + a_3(b_0+b_1) + a_0(b_2+b_3) + a_1(b_3+b_1),\\
c_3 &= a_3b_1 + a_0(b_1+b_2) + a_1(b_3+b_0) + a_2(b_0+b_2).
\end{aligned}$$

The squaring of A and multiplication of A with α, where $A \in \mathbb{F}_{2^4}$, are calculated as follows:

$$\begin{aligned}
A^2 &= (a_0\alpha + a_1\alpha^2 + a_2\alpha^{2^2} + a_3\alpha^{2^3})^2,\\
&= a_3\alpha + a_0\alpha^2 + a_1\alpha^{2^2} + a_2\alpha^{2^3}.\\
\alpha A &= \alpha(a_0\alpha + a_1\alpha^2 + a_2\alpha^{2^2} + a_3\alpha^{2^3}),\\
&= a_2\alpha + (a_0+a_2)\alpha^2 + (a_2+a_3)\alpha^{2^2} + (a_1+a_2)\alpha^{2^3}.
\end{aligned}$$

Hence, the squaring of A can be obtained by cyclically shifting one bit to the right, and αA can be implemented by using 3 XOR gates.

Arithmetic operations in $\mathbb{F}_{(2^4)^2}$. Let $A = a_0\beta + a_1\beta^{16}$ and $B = b_0\beta + b_1\beta^{16}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{2^4}$. The multiplication AB and the squaring of A in $\mathbb{F}_{(2^4)^2}$ can be computed in the same way as TF 1.

Implementation of WGT-8(x^{19}) Module

The hardware architecture of WGT-8(x^{19}) in TF 2 is almost the same as that in TF 1 (see Figure 4.4) and the only difference lies in the computation of trace and addition with 1. Under the tower construction in TF 2, the trace of an element is computed by XORing all component coordinates and 1 can be represented by $(1,1,1,1,1,1,1,1)$. As a result, 7 XOR gates and 8 NOT gates are required to implement the trace computation module and addition with 1.

4.2.4 Using Tower Field 3

The tower construction 3 (TF 3) is built by using three level construction, which are $\mathbb{F}_{2^2}$, $\mathbb{F}_{(2^2)^2}$, and $\mathbb{F}_{((2^2)^2)^2}$.

Tower Construction $\mathbb{F}_{((2^2)^2)^2}$ and Its Arithmetic

To obtain the tower construction $\mathbb{F}_{((2^2)^2)^2}$, we first construct $\mathbb{F}_{2^2}$ by using the irreducible polynomial $e_3(X)$ over $\mathbb{F}_2$, and then construct $\mathbb{F}_{(2^2)^2}$ by using a certain irreducible polynomial $f_3(X)$ of degree 2 over $\mathbb{F}_{2^2}$, and finally construct $\mathbb{F}_{((2^2)^2)^2}$ by using a certain irreducible polynomial $g_3(X)$ of degree 2 over $\mathbb{F}_{(2^2)^2}$. Where, $\alpha \in \mathbb{F}_{2^2}$, $\beta \in \mathbb{F}_{(2^2)^2}$ and $\gamma \in \mathbb{F}_{((2^2)^2)^2}$ are zeros of polynomials $e_3(X)$, $f_3(X)$ and $g_3(X)$ respectively. The specific polynomials and their normal bases are described in Table 4.5. Moreover, the normal basis $\{\theta, \theta^2, \cdots, \theta^{2^6}, \theta^{2^7}\}$ in this method is given by $\theta = \omega^{39}$ in $\mathbb{F}_{2^8}$. As we did before, the values of $\alpha, \beta, \gamma, \theta$ are chosen in order to get the low hamming weight conversion matrices (see below for details). Hence, the number of XOR gates used for implementing the conversion matrices will be quite small. The reason why we use this construction is that we want to have the nice property of trace function for two elements in $\mathbb{F}_{((2^2)^2)^2}$ (see below for details). Through the property of the trace function, a different architecture for implementing WGT-8(x^{19}) module will be used.

Table 4.5: Tower Construction $\mathbb{F}_{((2^2)^2)^2}$ with Normal Bases

Finite Field	Basis	Defining Polynomial
$\mathbb{F}_{2^8} \cong \mathbb{F}_{((2^2)^2)^2}$	NB, $\{\gamma, \gamma^{16}\},\ \gamma = \omega^{56}$	$g_3(X) = X^2 + X + \lambda,\ \lambda = \alpha^2\beta$
$\mathbb{F}_{2^4} \cong \mathbb{F}_{(2^2)^2}$	NB, $\{\beta, \beta^4\},\ \beta = \omega^{17}$	$f_3(X) = X^2 + X + \alpha$
$\mathbb{F}_{2^2} \cong \mathbb{F}_{(2)^2}$	NB, $\{\alpha, \alpha^2\},\ \alpha = \omega^{85}$	$e_3(X) = X^2 + X + 1$

Conversion matrices between TF and NB representations. For every element v in this tower field $\mathbb{F}_{((2^2)^2)^2}$, the TF representation is as follows:

$$\begin{aligned} v &= v_0\gamma + v_1\gamma^{16},\ v_0, v_1 \in \mathbb{F}_{(2^2)^2}, \\ &= [(v_{000}\alpha + v_{001}\alpha^2)\beta + (v_{010}\alpha + v_{011}\alpha^2)\beta^4]\gamma + \\ &\quad [(v_{100}\alpha + v_{101}\alpha^2)\beta + (v_{110}\alpha + v_{111}\alpha^2)\beta^4]\gamma^{16}, \end{aligned}$$

where $v_{000}, v_{001}, v_{010}, v_{011}, v_{100}, v_{101}, v_{110}, v_{111} \in \mathbb{F}_2$. Using the relations $\alpha = \omega^{85}$, $\beta = \omega^{17}$, and $\gamma = \omega^{56}$, we can get the conversion matrices $\mathbf{M}_{T\to N}$, and $\mathbf{M}_{N\to T}$ with hamming weight 20 and 28, respectively. The $\mathbf{M}_{T\to N}$ and $\mathbf{M}_{N\to T}$ are given below:

$$\mathbf{M}_{T\to N} = \begin{pmatrix} 1&0&0&1&0&0&1&0\\ 0&1&1&0&0&1&0&0\\ 0&0&1&0&1&1&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&1&0&1&0&0&1\\ 0&1&0&0&0&1&1&0\\ 1&1&0&0&0&0&1&0\\ 0&0&0&0&1&0&0&0 \end{pmatrix}, \quad \mathbf{M}_{N\to T} = \begin{pmatrix} 0&0&0&1&0&0&0&0\\ 0&1&1&0&0&0&0&1\\ 0&0&1&1&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 0&0&0&0&0&0&0&1\\ 0&0&0&1&0&1&1&0\\ 0&1&1&1&0&0&1&1\\ 0&0&1&1&1&1&1&0 \end{pmatrix}.$$

Arithmetic operations in $\mathbb{F}_{2^2}$. Let $A = a_0\alpha + a_1\alpha^2$ and $B = b_0\alpha + b_1\alpha^2$, where $a_i, b_i \in \mathbb{F}_2, i = 0, 1$. A multiplication AB in $\mathbb{F}_{2^4}$ is computed as follows:

$$\begin{aligned} AB &= (a_0\alpha + a_1\alpha^2)(b_0\alpha + b_1\alpha^2), \\ &= [(a_0 + a_1)(b_0 + b_1) + a_0b_0]\alpha + [(a_0 + a_1)(b_0 + b_1) + a_1b_1]\alpha^2. \end{aligned}$$

The squaring of A and multiplication of A by α, where $A \in \mathbb{F}_{2^2}$, are carried out as follows:

$$\begin{aligned} A^2 &= (a_0\alpha + a_1\alpha^2)^2 = a_1\alpha + a_0\alpha^2, \\ \alpha A &= a_0\alpha^2 + a_1(\alpha + \alpha^2) = a_1\alpha + (a_0 + a_1)\alpha^2. \end{aligned}$$

Arithmetic operations in $\mathbb{F}_{(2^2)^2}$. Let $A = a_0\beta + a_1\beta^4$ and $B = b_0\beta + b_1\beta^4$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{2^2}$. The arithmetic operations of A, B in $\mathbb{F}_{(2^2)^2}$, including multiplication, squaring, and multiplication by λ, are computed as follows:

$$\begin{aligned} AB &= (a_0\beta + a_1\beta^4)(b_0\beta + b_1\beta^4), \\ &= [(a_0 + a_1)(b_0 + b_1)\alpha + a_0b_0]\beta + [(a_0 + a_1)(b_0 + b_1)\alpha + a_1b_1]\beta^4, \\ A^2 &= (a_0\beta + a_1\beta^4)^2, \\ &= [(a_0^2 + a_1^2)\alpha + a_0^2]\beta + [(a_0^2 + a_1^2)\alpha + a_1^2]\beta^4, \\ \lambda A &= \alpha^2\beta A = (a_0\alpha + a_1)\beta + (a_0 + a_1)\beta^4. \end{aligned}$$

Arithmetic operations in $\mathbb{F}_{((2^2)^2)^2}$. Let $A = a_0\gamma + a_1\gamma^{16}$ and $B = b_0\gamma + b_1\gamma^{16}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{(2^2)^2}$. The arithmetic operations of A, B in $\mathbb{F}_{((2^2)^2)^2}$, including multiplication, and squaring, are computed as follows:

$$\begin{aligned} AB &= (a_0\gamma + a_1\gamma^{16})(b_0\gamma + b_1\gamma^{16}), \\ &= [(a_0 + a_1)(b_0 + b_1)\lambda + a_0b_0]\gamma + [(a_0 + a_1)(b_0 + b_1)\lambda + a_1b_1]\gamma^{16}, \\ A^2 &= (a_0\gamma + a_1\gamma^{16})^2, \\ &= [(a_0^2 + a_1^2)\lambda + a_0^2]\gamma + [(a_0^2 + a_1^2)\lambda + a_1^2]\gamma^{16}. \end{aligned}$$

Properties of the Trace Function Under this Tower Construction

As described in [30], the trace of the product of any two elements can be efficiently computed as the inner product of their binary vectors, when the elements in $\mathbb{F}_{2^m}$ are represented in a type-II ONB. Although there is no Gaussian normal basis over $\mathbb{F}_{2^8}$, the above nice property of trace function still holds if we use the isomorphic tower field construction $\mathbb{F}_{((2^2)^2)^2}$ in $\mathsf{TF}\ 3$. Fan *et al.* [37] proved the following theorem and two corollaries, which characterize the aforementioned property.

Theorem 3 *Given the tower construction in $\mathsf{TF}\ 3$, the trace of the field multiplication of any two elements U and V in $\mathbb{F}_{((2^2)^2)^2}$ can be computed as the inner product of U and V, i.e.,*

$$Tr(UV) = \sum_{i=0}^{7}(u_i \odot_1 v_i).$$

Corollary 1 *Given the tower construction in $\mathsf{TF}\ 3$, for any elements X, U and V in $\mathbb{F}_{((2^2)^2)^2}$, we have $Tr(X^{2^\omega}) = Tr(X) = \sum_{i=0}^{7} x_i$ and $Tr(UV) = Tr(U^{2^\omega} \odot_8 V^{2^\omega})$, where ω is an integer.*

Corollary 2 *Given the tower construction in $\mathsf{TF}\ 3$, for any elements U, V and W in $\mathbb{F}_{((2^2)^2)^2}$, we have*

$$Tr(U \odot_8 W) \oplus_1 Tr(V \odot_8 W) = Tr((U \oplus_8 V) \odot_8 W).$$

Hardware Architecture of $\mathsf{WGT\text{-}8}(x^{19})$ Module for the Running Phase by Using the Trace Property

Using the above theorem and two corollaries, the $\mathsf{WG\text{-}8}$ transformation $\mathsf{WGT\text{-}8}(x^{19})$ can be computed as follows:

$$\begin{aligned}
\mathsf{WGT\text{-}8}(x^{19}) &= \mathrm{Tr}(\mathsf{WGP\text{-}8}(x^{19})),\ x \in \mathbb{F}_{2^8}\\
&= \mathrm{Tr}\left(y \oplus_8 y^{2^3+1} \oplus_8 y^{2^6}(y^{2^3+1} \oplus_8 y^{2^3-1}) \oplus_8 y^{2^3(2^3-1)+1}\right),\\
&= \mathrm{Tr}(y \oplus_8 y^{2^3+1}) \oplus_1 \mathrm{Tr}(y^{2^6}(y^{2^3+1} \oplus_8 y^{2^3-1})) \oplus_1 \mathrm{Tr}(y^{2^3(2^3-1)}y),\\
&= \mathrm{Tr}(y \oplus_8 y^{2^3+1}) \oplus_1 \mathrm{Tr}(y^{2^6} \odot_8 (y^{2^3+1} \oplus_8 y^{2^3-1})) \oplus_1 \mathrm{Tr}(y^{2(2^3-1)}y^{2^6}),\\
&= \mathrm{Tr}(y \oplus_8 y^{2^3+1}) \oplus_1 \mathrm{Tr}(y^{2^6} \odot_8 (y^{2^3+1} \oplus_8 y^{2^3-1} \oplus_8 y^{2(2^3-1)})),
\end{aligned}$$

where $y = x^{19} \oplus_8 1 = x^{2^4} \cdot x^2 \cdot x \oplus_8 1$. The $\mathsf{WGT\text{-}8}(x^{19})$ module can be implemented using five multipliers in total for the running phase, where two of them are used to generate y, one for computing y^{2^3+1}, and another two for generating y^{2^3-1}. Due to the property of trace function, the two multiplications $y^{2^6}(y^{2^3+1} \oplus_8 y^{2^3-1})$ and $y^{2^3(2^3-1)}y$ inside the trace function has been replaced by the bitwise AND and XOR operations, which enables us to reduce the area and delay significantly. Using the property of the trace function, we can directly get the trace result of the field multiplication of two arbitrary elements A and B, but the corresponding multiplication result AB is lost. Therefore, some strategy will be taken to recover the value of AB for generating WGP-8 for feedback in the initialization phase.

Hardware Architecture of the WGP-8(x^{19}) Module for the Initialization Phase

In the initialization phase, the WG-8 permutation is computed below:

$$\begin{aligned}
\mathsf{WGP\text{-}8}(x^{19}) &= q(x^{19}+1)+1,\ x \in \mathbb{F}_{2^8} \\
&= (1 \oplus_8 y \oplus_8 y^{2^3+1}) \oplus_8 (y^{2^6}(y^{2^3+1} \oplus_8 y^{2^3-1})) \oplus_8 (y^{2^3(2^3-1)}y), \\
&= (x^{19} \oplus_8 y^{2^3+1}) \oplus_8 (y^{2^6}(y^{2^3+1} \oplus_8 y^{2^3-1})) \oplus_8 (y^{2^3(2^3-1)}y).
\end{aligned}$$

The two multiplication values $y^{2^6}(y^{2^3+1} \oplus_8 y^{2^3-1})$ and $y^{2^3(2^3-1)}y$ are missing due to the property of the trace function. However, y^{2^3+1} and y^{2^3-1} can be obtained from the WGT-8(x^{19}) module. Noting that there are five multipliers in the WGT-8(x^{19}), we can reuse two of them to calculate the two missing intermediate multiplication values over two consecutive clock cycles. This is done by keep the throughput of the running phase to be 1 bit per clock cycle while decreasing the throughput of the initialization phase (i.e., <1 bit/clock). In other words, it is to increase the length of the initialization phase and reuse the existing multipliers in the running phase.

Figure 4.6 describes an integrated architecture for computing WGT-8(x^{19}) and WGP-8(x^{19}). To compute the WGP-8(x^{19}) within two clock cycles, six extra multiplexers (i.e., MUX_1, MUX_2, MUX_3, MUX_4, MUX_5, MUX_6) and three additional 8-bit registers (i.e., Reg_1, Reg_2, Reg_3) are added, as shown in Figure 4.6. The outputs of the six multiplexers and the updated states of the three registers are summarized in Table 4.6, under the control of sel and clk signals. It takes 2 times more clock cycles than before, 80 clock cycles, to finish the initialization phase, while other phases takes the same clock cycles as before. During the initialization phase, when sel equals 0, the output of the WGP-8(x^{19}) is an intermediate value; therefore it will not be sent to the input of the LFSR. When sel becomes 1, the initialization feedback will be sent to update the LFSR when the rising clock comes, along with the linear feedback within the LFSR. Moreover, the sel signal will stay 0 in the running phase, and it will output WGT-8(x^{19}) directly.

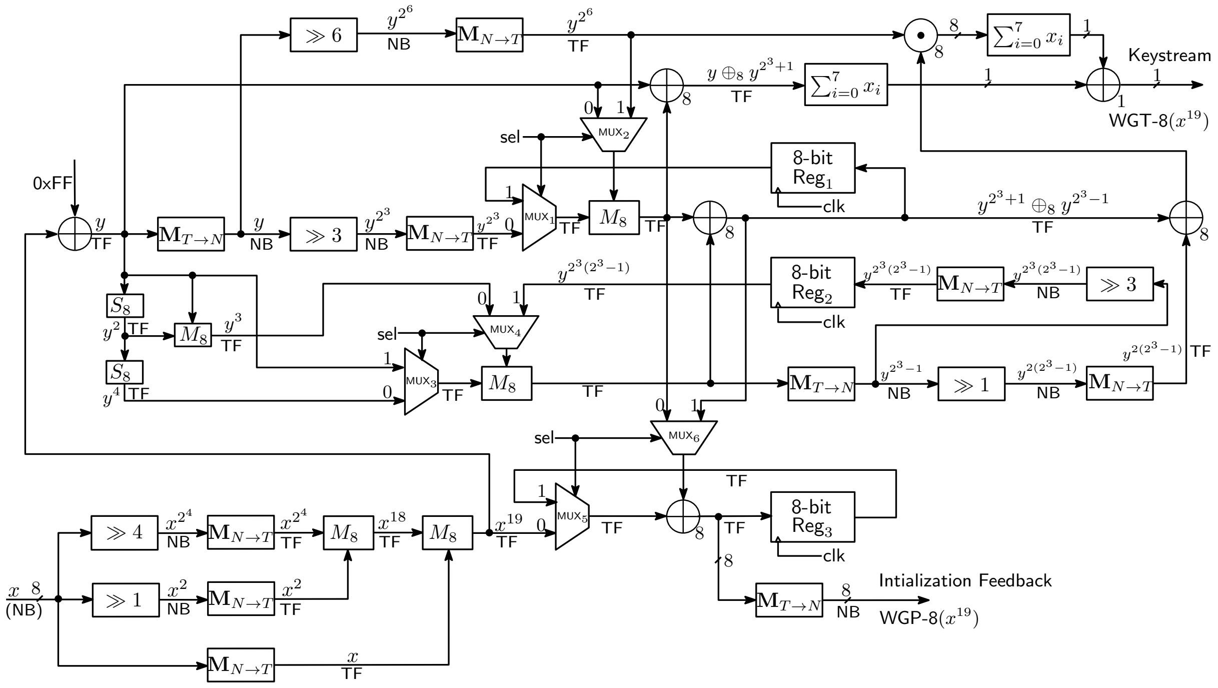


Figure 4.6: The Integrated Hardware Architecture for Computing WGP-8(x^{19}) and WGT-8(x^{19})

Table 4.6: Multiplexers and Registers During the Two-clock Computation of WGP-8(x^{19})

Component		Control Signal sel	
		0	1
Multiplexer	MUX$_1$	y^{2^3}	$y^{2^3+1} \oplus_8 y^{2^3-1}$
	MUX$_2$	y	y^{2^6}
	MUX$_3$	y^4	y
	MUX$_4$	y^3	$y^{2^3(2^3-1)}$
	MUX$_5$	x^{19}	$x^{19} \oplus_8 y^{2^3+1}$
	MUX$_6$	y^{2^3+1}	$y^{2^6}(y^{2^3+1} \oplus_8 y^{2^3-1}) \oplus_8 y^{2^3(2^3-1)}y$
Register	Reg$_1$	$y^{2^3+1} \oplus_8 y^{2^3-1}$	Not Relevant
	Reg$_2$	$y^{2^3(2^3-1)}$	Not Relevant
	Reg$_3$	$x^{19} \oplus_8 y^{2^3+1}$	WGP-8(x^{19})

4.2.5 Hybrid Design Architectures for WG-8

In this subsection, we propose a hybrid architecture for implementing WG-8 based on the above four methods. The purpose of this hybrid architecture is to realize the parallel implementations of WG-8 for parallel width from one to eleven. The maximum number eleven depends on the tap positions of the LFSR's feedback function in WG-8. According to the description in Section 4.1.2, we know the maximum tap position of the feedback function of WG-8 is nine and the significant state of the LFSR is S_{19}, so the maximum parallel width is eleven. The idea for the parallelism is that we can precompute

the feedback values from the 20-stage LFSR through the feedback function for different parallel widths. Hereafter, we only illustrate the strategies for the parallel width equals one and eleven, and the case for the other parallel width numbers is similar. For parallel width equals eleven, we can precompute the eleven feedback values from the LFSR and then load them to the LFSR in the running phase at one clock cycle. In this way, the eleven bits per clock cycle can be generated through eleven copies of WGT-8 module.

In order to achieve parallelism for the initialization phase, we need to precompute eleven feedback values from the WGP-8 module and the LFSR's feedback functions. However, the parallelism is impractical since the input to the WGP-8 is dependent upon the previous feedback value, making the eleven feedback values cannot be obtained in one clock cycle. Therefore, we use a serial design for the initialization phase, where only one feedback value is generated for the LFSR. In our architecture, we define the combined serial design for the initialization phase and the parallel design for the running phase as the hybrid design architecture.

For the parallel width from one to eleven in our hybrid design, we can use one to eleven copies of WGT-8 module in the running phase and one WGP-8 module for the initialization phase or one to eleven copies of WGP-8 and trace modules for the running phase and reuse one of the WGP-8 modules for the initialization phase.

The area of the WGP-8 module is bigger than that of the WGT-8 module in the constant array based method (see Section 4.2.1). Therefore, for parallel width from two to eleven, the total area of the WGP-8 and trace modules is definitely bigger than that of the WGT-8 modules plus one WGP-8 module. Even though the total area of one WGP-8 module plus one WGT-8 module is equal to that of one WGP-8 module plus one trace module for parallel width one, the critical path is longer for the latter design than that of the former design. Thus, in order to balance the area and clock speed, we choose the hybrid design with one to eleven WGT-8 modules and one WGP-8 module (for parallel width from one to eleven respectively) for our constant array based method. The details of this hybrid design architecture for parallel width eleven is shown in Figure 4.7.

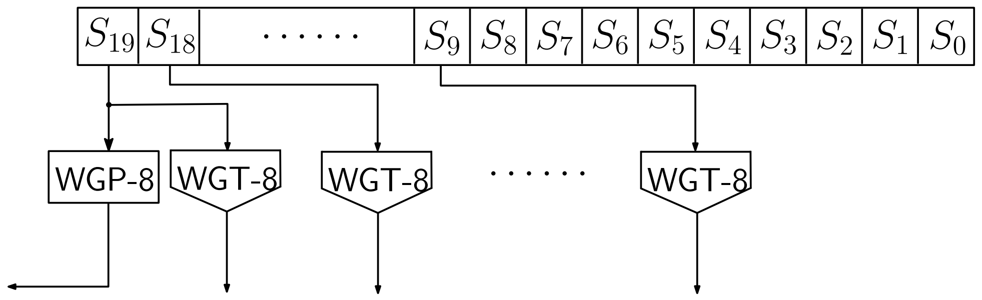


Figure 4.7: The Hybrid Design for the Constant Array based Method

For the three different tower field methods, the area of the WGT-8 module is bigger than that of the WGP-8 module (see Section 4.2.2, 4.2.3, and 4.2.4). Therefore, the hybrid design with one to eleven WGP-8 and trace modules (for parallel width from one to eleven respectively) is chosen in this case.

4.3 Design Strategies for the Multiplication by ω Module

The multiplication by ω module can be implemented using either finite field arithmetic or an 8×8 constant array. The finite field arithmetic based method is used for our implementations.

4.3.1 Using Finite Field Arithmetic

With the PB representation, the multiplication of an element $X \in \mathbb{F}_{2^8}$ by ω can be computed as follows:

$$\begin{aligned} X \cdot \omega &= x_0\omega + x_1\omega^2 + \cdots + x_6\omega^7 + x_7\omega^8 \\ &= x_7 + x_0\omega + (x_1 \oplus_1 x_7)\omega^2 + (x_2 \oplus_1 x_7)\omega^3 + \\ &\quad (x_3 \oplus_1 x_7)\omega^4 + x_4\omega^5 + x_5\omega^6 + x_6\omega^7. \end{aligned}$$

Therefore, the result of $X \cdot \omega$ is represented as the 8-bits vector $(x_7, x_0, x_1 \oplus_1 x_7, x_2 \oplus_1 x_7, x_3 \oplus_1 x_7, x_4, x_5, x_6)$ with respect to the PB, which corresponds to a cyclic right shift operation plus three XOR operations in hardware implementation.

With the NB representation, the multiplication of an element $X \in \mathbb{F}_{2^8}$ by ω can be calculated as follows:

$$\begin{aligned} X \cdot \omega &= (x'_0\theta + x'_1\theta^2 + \cdots + x'_6\theta^{2^6} + x'_7\theta^{2^7}) \cdot \omega \\ &= \mathbf{M} \cdot (x'_0, x'_1, \cdots, x'_6, x'_7)^T, \end{aligned}$$

where the matrix M can be calculated using the specific normal basis, i.e., θ. The matrices M are given below for $\mathsf{TF}\ 1$, $\mathsf{TF}\ 2$, and $\mathsf{TF}\ 3$.

$$\mathbf{M}_{\mathsf{TF1}} = \begin{pmatrix} 0&1&0&0&0&0&1&0 \\ 1&0&0&0&1&1&0&0 \\ 1&0&0&0&0&1&0&1 \\ 1&0&0&0&0&0&1&1 \\ 1&0&0&1&0&0&0&0 \\ 1&1&1&1&0&0&1&1 \\ 0&1&1&1&0&1&1&1 \\ 1&1&1&1&0&1&1&1 \end{pmatrix}, \quad \mathbf{M}_{\mathsf{TF2}} = \begin{pmatrix} 0&0&1&1&0&1&1&1 \\ 1&0&0&0&0&1&1&1 \\ 1&0&1&0&0&1&0&1 \\ 1&0&0&0&1&0&0&0 \\ 1&1&1&0&0&1&0&1 \\ 0&1&0&1&0&0&1&0 \\ 1&1&1&1&1&1&0&1 \\ 0&0&0&0&0&0&1&1 \end{pmatrix},$$

$$\mathbf{M}_{\mathsf{TF3}} = \begin{pmatrix} 0&1&1&0&0&1&1&1 \\ 1&0&0&1&1&0&0&0 \\ 0&1&0&1&1&0&0&0 \\ 0&1&0&1&0&0&1&1 \\ 1&0&0&0&0&1&1&1 \\ 0&0&1&0&1&0&0&1 \\ 1&1&1&1&1&0&1&1 \\ 0&0&0&1&1&1&0&0 \end{pmatrix}.$$

4.3.2 Using Constant Array

Based on the finite field arithmetic or multiplication matrices, one can easily precompute the corresponding 8×8 constant arrays with respect to the chosen bases.

4.4 Hardware Implementations

In this section, we present the hardware implementations of WG-8 in FPGAs and ASICs.

4.4.1 Finite State Machine

According to the hardware architecture in Figure 4.3, the finite state machine (FSM) controls the input to the LFSR. The FSM begins at the IDLE state and the counter returns to 0 when the reset signal rst equals 1. Once the reset signal rst is pulled down to 0, the FSM enters the LOAD state. During the LOAD state, the FSM starts loading Key and IV, and the counter is increased by one at each clock cycle. For the hybrid design with parallel width one, when the counter reaches a value of 19, the FSM goes into the next state INIT_PHASE. During the INIT_PHASE state, the counter continues increasing by one at every clock cycle until it hits a value of 59 for the constant array based method, and TF 1 and 2. However, for the TF 3, the counter needs to reach 99. The reason is that the initialization phase in the TF 3 takes twice number of clock cycles than that of other methods. More specifically, the FSM will generate the third output signal sel in the TF 3, which controls the multiplexers in Figure 4.6. The FSM then transfers to the next state RUN_PHASE and begins to output keystream bits. The sel signal in the TF 3 should be set to 0 in the RUN_PHASE.

In addition, for the hybrid design with the parallel width eleven, the state transition values are different compared to the parallel width one. In this case, the FSM first goes from LOAD state to INIT_PHASE state when the counter reaches 1, followed by the RUN_PHASE state when the counter equals 41 for the constant array based method, TF 1 and 2, but 81 for the TF 3. Moreover, the FSM will remain in RUN_PHASE state unless a reset signal rst is pulled up. During the RUN_PHASE state, eleven bits keystream per clock cycle will be generated .

4.4.2 FPGA Implementations and Results

Our FPGA area and speed results are for the low-cost Spartan-3 XC3S1000 (Package FG320 with speed grade -5) FPGA device from Xilinx using Synopsys Synplify Premier with Design Planner E-201103-SP2 for synthesis, ISE for physical implementation, and Xpower Analyzer for power analysis. All results, including flip-flops, area (slices), speed (maximum frequency), and dynamic power consumption are obtained after Place and Route phase and the dynamic power consumption are recorded at a frequency of 33.3 MHz for all the designs except WG-8 (TF 1), the dynamic power consumption of which is evaluated at a frequency of 16.7 MHz.

Table 4.7 summarize the flip-flop numbers, area, speed, power consumption, optimality results for FPGA implementations. For each method, we calculate three common metrics for optimality that made trade-offs between maximum throughput (maximum frequency * throughput), area, and power: MT/A is maximum throughput over area, MT/P is throughput divided by power, and T/(A*P) is maximum throughput divided by product of area and power.

Table 4.7: The Area, Speed, and Power Consumption Results of FPGA Implementations

Spartan-3(xc3s1000)											
Implementations	Key Size (bits)	IV Size (bits)	Throughput (bits/ cycle)	#FFs	Area (Slices)	Maximum Frequency (MHz)	Dynamic Power (W)	Maximum Throughput (Mbps)	Optimality		
									MT/A (Mbps/ #Slices)	MT/P (Mbps/ W)	MT/(A*P) (Mbps/ Slices*W)
WG-8 (CA)	80	80	1	85	137	190	0.005	190	1.39	38000	277.4
WG-8 (CA)			11	207	398	192	0.016	2112	5.31	132000	331.7
WG-8 (TF 1)			1	83	678	19	0.671	19	0.03	28.3	0.042
WG-8 (TF 1)			11	279	5106	19	4.282	209	0.04	48.8	0.010
WG-8 (TF 2)			1	83	343	42	0.339	42	0.12	123.9	0.36
WG-8 (TF 2)			11	306	2369	42	1.686	462	0.20	274	0.12
WG-8 (TF 3)			1	114	436	49	0.267	49	0.11	183.5	0.42
WG-8 (TF 3)			11	470	2795	44	2.399	484	0.17	201.7	0.07
Grain [57]	80	64	1	–	44	196	–	196	4.45	–	–
Trivium [57]	80	80	1	–	50	240	–	240	4.80	–	–

For the hybrid design with parallel width one, we take advantages of the SRL16 shift register cells that exist in the Spartan-3 devices. The SRL16 is a very efficient way to create shift registers without using flip-flop resources. After writing the 20-stage LFSR VHDL code properly based on the user guide [1] from Xilinx, the synthesis tool can successfully design the 20-stage LFSR using SRL16 shift register cells. In Spartan-3 FPGA, one look-up table (LUT) in a SLICEM slice can be configured as a 16-bit shift register. Shift-in operations of SRL16 cell are synchronous with the clock, and output address is dynamically selectable, ranging from 1 to 16. By using the SRL16 cells to implement the 20-stage LFSR, we are able to reduce the flip-flop numbers and area significantly. However, for the hybrid design with parallel width larger than one, the 20-stage LFSR cannot be implemented by SRL16 cells, due to the parallel inputs of the 20-stage LFSR. Moreover, we use the "-Power" option in the Mapping and Place and Route phases to further reduce the power consumption.

A very obvious observation is that there are a 20-stage LFSR and the width of each state is 8 bits. Therefore, we need 160 registers in total for the 20-stage LFSR in the normal implementation. But there are only 85 registers for the parallel width one in Spartan-3 FPGA. The reason is that the synthesis tool has used 16 SRLE16 modules for the implementation with parallel width one, decreasing the register numbers significantly. More specifically, we can see that there are two seqshift modules (seqshift, seqshift0) with data width equals eight, which is exactly 16 SRLE16 modules, from the RTL schematic. For seqshift, the input to this module is r19 and the address number of this module is 1001. After the set of these two parameters, we can see the output of seqshift is r9. This is exactly why the register numbers will decrease dramatically. The same strategy applies to the seqshift0 module. For the parallel width larger than one, the synthesis tool cannot map SRL16 modules to the design. We can clearly see that the 20-stage LFSR are implemented using 160 registers in this case. Therefore, the total register numbers will be larger than 160.

4.4.3 ASIC Implementations and Results

Our previous published ASIC results in CMOS 65nm are given in [115]. During the past years of research on other primitives, our new technique can further reduce the area and power consumption. Therefore, we refresh our results of WG-8 in CMOS 65nm and provide new results in CMOS 130nm ASIC.

We use the same implementation strategy as Simeck and SIMON in Chapter 3. The corresponding area and total power consumption results are obtained using compile ultra and clock gating technique. The area is dependent upon the clock period, and we need to run the constraints, in which contain the expected clock period, for the synthesis and Place and Route phase many times in order to get the smallest area with smallest actual clock period for our WG-8 circuit. By repeating this strategy, we can get all the results for our four approaches of WG-8. The total power is given, instead of only dynamic power. It is because that the static and dynamic powers are both dependent upon our design in ASIC. The power consumption analysis is conducted under the operating frequency at 100 KHz and 2 MHz for all the designs after Place and Route phase using random data for the key and initialization vector, where 100 KHz is for the benchmarking purpose and 2 MHz is for the practical deployment in RFID systems. The Modelsim simulation runs for 2000 clock cycles, including initialization and running phases.

Tables 4.8 and 4.9 summarize area, speed, power consumption, and optimality results for ASIC implementations. For each method, we calculate three common metrics for optimality that made trade-offs between maximum throughput (maximum frequency*throughput), area, and power: MT/A is maximum throughput over area (after P&R), MT/P is maximum throughput divided by power (total power consumption at 2 MHz), and MT/(A*P) is maximum throughput divided by product of area (after P&R) and power (total power consumption at 2 MHz).

4.5 Results Analysis and Comparison

For both the FPGA and ASIC results in Tables 4.7, 4.8, and 4.9, we provide all the results for throughputs of 1 bit per clock cycle (bpc) and 11 bpc by using our hybrid design architecture. From the results, we can achieve the design with throughput of 11 bpc from the design with throughput of 1 bpc by using some extra area. However, the extra area is relatively smaller than

Table 4.8: Area, Speed, and Power Consumption Results for ASIC Implementations in 65nm

CMOS 65nm											
Implementations	Key Size (bits)	IV Size (bits)	Throughput (bits/ cycle)	Area Before P&R (GEs)	Area After P&R (GEs)	Max Freq (MHz)	Total Power (μ W) @100KHz	Total Power (mW) @2MHz	Optimality MT/A (Mbps/ #GE)	Optimality MT/P (Mbps/ mW)	Optimality MT/(A*P) (Mbps/ GE*mW)
WG-8 (CA)	80	80	1	1493	1606	504	4.049	0.016	0.314	31500	19.614
WG-8 (CA)			11	3699	3978	427	10.066	0.039	1.177	120428	30.250
WG-8 (TF 1)			1	3634	3908	105	17.177	0.179	0.027	587	0.150
WG-8 (TF 1)			11	32422	34863	94	161.610	1.788	0.033	583	0.022
WG-8 (TF 2)			1	2314	2488	163	8.350	0.064	0.065	2547	1.024
WG-8 (TF 2)			11	15488	16654	153	56.670	0.443	0.099	3795	0.231
WG-8 (TF 3)			1	2317	2492	189	8.752	0.060	0.075	3150	1.260
WG-8 (TF 3)			11	18370	19753	170	76.388	0.604	0.099	3102	0.154
Grain [115]	80	64	1	–	1126	1020	–	–	–	–	–
Grain [115]			11	–	1126	1098	–	–	–	–	–
Trivium [115]	80	80	1	–	1986	962	–	–	–	–	–
Trivium [115]			11	–	2028	990	–	–	–	–	–

ten copies of the area of design with throughput of 1 bpc. Meanwhile, the increase of throughput for the design does not affect the corresponding maximum frequency too much for all the cases. Therefore, the hybrid design architecture is very useful for WG-8 in order to provide various throughput.

In addition, we notice that the WG-8 (CA) method is the best hardware solution for the WG-8 stream cipher, compared to the tower field based approaches, i.e., the areas are smaller and the three metrics in optimality are all higher for both throughput of 1 and 11 bpcs. The reason is that building WG-8 permutation/transformation module with an 8×8 constant array occupies smaller area than using logic equations and multipliers. However, when the size of finite field increases, the medium and large instances of WG stream cipher family will benefit from the tower construction, thereby achieving a better performance, as illustrated in [37].

4.5.1 Different Tower Field Methods Analysis

For these three tower field methods, the best choice of them is relevant to the throughput and different metrics, as summarized in Table 4.10. The reason for this results is due to different tower field architectures and the number of multipliers. Since the multiplier occupies a large part of the area and consumes considerable power, we can perform analysis according to the number of multiplier and its area. There are seven multipliers in TF 1 and TF 2 with almost the same

Table 4.9: Area, Speed, and Power Consumption Results for ASIC Implementations in 130nm

CMOS 130nm											
Implementations	Key Size (bits)	IV Size (bits)	Throughput (bits/cycle)	Area Before P&R (GEs)	Area After P&R (GEs)	Max Freq (MHz)	Total Power (μ W) @100KHz	Total Power (mW) @2MHz	Optimality MT/A (Mbps/#GE)	Optimality MT/P (Mbps/mW)	Optimality MT/(A*P) (Mbps/GE*mW)
WG-8 (CA)	80	80	1	1466	1593	97	0.960	0.019	0.060	5105.26	3.205
WG-8 (CA)			11	3536	3843	70	2.336	0.046	0.198	16738.70	4.356
WG-8 (TF 1)			1	3300	3586	20	12.370	0.247	0.006	80.97	0.023
WG-8 (TF 1)			11	25961	28219	21	83.880	1.674	0.0077	137.94	0.004
WG-8 (TF 2)			1	2371	2577	39	4.770	0.095	0.0151	410.50	0.159
WG-8 (TF 2)			11	15020	16326	37	31.249	0.622	0.022	654.39	0.044
WG-8 (TF 3)			1	2401	2610	38	4.432	0.088	0.0145	431.81	0.165
WG-8 (TF 3)			11	18717	20345	37	44.713	0.891	0.0198	456.72	0.022
Grain [7]	80	64	1	–	1259	–	0.780	–	–	–	–
Trivium [7]	80	80	1	–	2088	–	1.440	–	–	–	–

Table 4.10: The Best Choice of the Tower Field Methods for Different Metrics

Throughputs (bpc)	FPGA			65nm ASIC			130nm ASIC		
	MT/A	MT/P	MT/(A*P)	MT/A	MT/P	MT/(A*P)	MT/A	MT/P	MT/(A*P)
1	TF 2	TF 3	TF 3	TF 3	TF 3	TF 3	TF 2	TF 3	TF 3
11	TF 2	TF 2	TF 2	TF 2/3	TF 2	TF 2/3	TF 2	TF 2	TF 2

architecture. Unlike TF 1 and 2, TF 3 is intended to reduce the number of multipliers, with only five multipliers needed. However, six extra multiplexers are required due to the architecture of TF 3. Therefore, for analysis of TF 3, we should combine the effects of multipliers and multiplexers. As we have described before, the multiplication in TF 1 are based on the small 4×4 constant arrays due to the existence of primitive element, and the multiplication in TF 2 and TF 3 are performed using the efficient logic equations directly. We give the area of one multiplier in $\mathbb{F}_{2^8}$ for TF 1, 2, and TF 3 in FPGA (Spatan-3) for analysis, which is 53, 29, and 26 slices, respectively, as shown in Table 4.11. From this observation, the TF 1 is worse than TF 2 in terms of area and power consumption, due to similar architecture. However, for the TF 3, we should also consider the effects of multiplexers, making it hard to compare with TF 2. Therefore, the best tower field method is different with respect to the data rates, implementation approaches, i.e., FPGA or ASIC, and metrics in optimality. Nevertheless, the TF 2 is the best choice in most cases, and it has the smallest area among the three tower field methods. Moreover, to our best knowledge, the area of one multiplier is quadratic with respect to the size of finite field, and the area of one multiplexer and other miscellaneous are linear with respect to the size of the finite

Table 4.11: The Number of Multipliers and Multiplexers, and the Area of Them in FPGA

Implementations	Multipliers (Number)	Multiplier (Single area) (slices)	Multiplier (Total areas) (slices)	Multiplexers (Number)
TF 1	7	53	371	0
TF 2	7	29	203	0
TF 3	5	26	130	6

field. Thus, for arithmetic in small finite field $\mathbb{F}_{2^8}$, e.g., TF 3, the decreased area by the number of multipliers will be balanced to the increased area of multiplexers. However, the decreased area will be larger for large size finite field, thereby achieving a better performance.

4.5.2 Comparisons with Other Lightweight Stream Ciphers

Table 4.7 also compares the performance of our WG-8 implementations against two lightweight stream ciphers Grain [55] and Trivium [24] in the hardware profile of the eSTREAM project on the same Xilinx Spartan-3 FPGA platform [57]. For the FPGA results, Grain and Trivium are better than WG-8 (CA) method in metric of MT/A, and their areas are smaller than that of WG-8 (CA). However, when we compare the WG-8 (CA) method with Grain and Trivium in ASIC, we have the following conclusions. For throughput of 1 bpc, the area of WG-8 (CA) is smaller than that of Trivium [115], but larger than that of Grain [115] in CMOS 65nm. The area of WG-8 (CA) is larger than that of Grain and Trivium for throughput of 11 bpc. In addition, the maximum frequency is smaller than that of Grain and Trivium. For CMOS 130nm, the same conclusion holds for throughput of 1 bpc, and the total power consumption at 100 KHz are larger than that of Grain, but smaller than that of Trivium [7]. Moreover, the WG-8 stream cipher has desired randomness properties like period, balance, ideal two-level autocorrelation, ideal tuple distribution, and exact linear complexity [35]. While Grain can only provide a lower bound for the period of a keystream [55], and Trivium does not have any determined randomness properties by design [24]. Therefore, WG-8 is a promising candidate for providing strong encryption and authentication solutions for RFID systems.

4.6 Summary

In this chapter, we have presented the design space exploration of the lightweight stream cipher WG-8 for FPGA and ASIC implementations. Four implementation approaches have been proposed, where one takes advantage of the constant arrays and the other three benefit from the tower constructions of finite field $\mathbb{F}_{2^8}$ and efficient basis conversion matrices. A wide range of design options and strategies have been thoroughly explored to make trade-offs in terms of area, speed, and power consumption. More specifically, we have proposed a hybrid architecture with parallel width from one to eleven. Since the S-box in AES and large size stream cipher WG-16 demonstrated the advantages of tower field constructions, the tower field approaches have been extensively investigated in WG-8. From the results, we can obtain that the tower field constructions affected both the area of one multiplier and the number of the multipliers in their architectures. Moreover, among the three tower field constructions, TF 2 with the type-I optimal normal basis is the best choice in most cases for WG-8. For future work, we will propose more efficient tower field constructions and explore the design space using tower field for larger size WG stream ciphers, i.e., WG-10, WG-11, WG-14, etc. In summary, WG-8 is a good lightweight candidate for securing RFID systems.

Chapter 5

Warbler Pseudorandom Number Generator

Pseudorandom number generators (PRNGs) are a critical component of EPC C1 G2 standard, and are widely used to provide security in both the first and second versions of the standard [2, 3]. For example, a 16-bit random number (RN16) is used in many commands, for anti-collision mechanism, to provide verification of the reader identity, and to mask the data [1]. More specifically in the inventory process, the reader first obtains the RN16 from the tag through sending a *Query* command to the tag. Afterwards, the reader sends an *ACK* command with the received RN16 to the tag. Then, the tag checks the correctness of the RN16 and it will send back its EPC number if the RN16 is correct. In addition, using the RN16 to do data masking is adopted in the *access*, *kill*, and *write* commands in the EPC standard [2, 3]. Random numbers can also be used in security extensions of this standard, such as the challenge-response based lightweight mutual authentication protocols [32] between the readers and tags, where both the readers and tags use random numbers as challenges.

The randomness properties of the RN16, drawn from the PRNG, shall at least satisfy the EPC C1 G2 standard's statistical tests [2, 3]: 1) The probability for any RN16 shall be bounded by $0.8/2^{16} < P(RN16) < 1.25/2^{16}$. 2) The probability that any two or more tags generate the same sequence of RN16s shall be less than 0.1%. 3) The probability of predicting an RN16 which is 10 ms later, shall not be greater than 0.025%.

Motivated by the above applications, several lightweight PRNGs have been devised in recent years, such as Warbler [72], Melia-Segui *et al.* [78], J3Gen [79], LAMED [91], and AKARI1B [76]. The key components of them are, nonlinear feedback shift registers (NLFSRs), linear

[1]mask is to provide cover-code to XOR with the data.

feedback shift register (LFSR) with random source, and ARX structure (*i.e.*, addition modular operation, bit rotations, and XOR operation).

The Warbler PRNG is proposed by Mandal and Gong in 2012 [72]. The output sequence of Warbler has guaranteed randomness properties such as long period and high linear span. Fan *et al.* has a preliminary implementation of Warbler [73]. Our contribution is to provide the first detailed and smallest hardware implementations of Warbler.

In this chapter, we present the low-area implementation and optimizations of Warbler in CMOS 65nm and CMOS 130nm ASICs, and provide area, maximum clock frequency, and total power consumption results. This chapter is organized as follows. In Section 5.1, we describe the specification of Warbler. Then, we give our ASIC architecture of Warbler, including the top-level architecture, finite state machine (FSM), and datapathes for both throughputs of $1/5$ and 1 bpc in Section 5.2. Later on, we present our ASIC results and thorough analysis in both CMOS 65nm and CMOS 130nm in Section 5.3. Section 5.4 provides some related work and compares Warbler with other lightweight primitives. Finally, Section 5.5 concludes this chapter.

5.1 Description of Warbler

This section gives a detailed description of the Warbler.

Warbler is mainly built upon three NLFSRs and four WG-5 transformation modules, as shown in Figure 5.1. All the computation in Warbler use the polynomial basis $\{1, \alpha, \alpha^2, \alpha^3, \alpha^4\}$ in $\mathbb{F}_{2^5}$, which is defined by a primitive polynomial of degree 5 over $\mathbb{F}_2$:

$$p(x) = x^5 + x^4 + x^3 + x + 1,$$

and α is a primitive root such that $p(\alpha) = 0$. NLFSR1 and NLFSR2 are both 1 bit wide for each state, and their lengths are 18 and 17 respectively, where the lengths are chosen to be relatively prime. The output of NLFSR1 and NLFSR2 are firstly XORed together, and then is sent to a 5-bit shift register (SR), which is one feedback input to NLFSR3. NLFSR3 is 5 bits wide for each state and it has 6 stages. Warbler mainly includes two phases: initialization phase and running phase, and the feedbacks to the NLFSR1 and NLFSR2 are different during these two phases as shown in Figure 5.1. For the initialization phase, the goal is to scramble the internal states as quickly as possible, and so an additional bit of feedback from NLFSR3 is included. For the running phase, NLFSR1 and NLFSR2 generate span-n sequences and NLFSR3 outputs a sequence with 1/5 bpc throughput, where the *span n* sequence is a binary sequence with period $2^n - 1$ and each non-zero n-tuple occurs exactly once in one period [72].

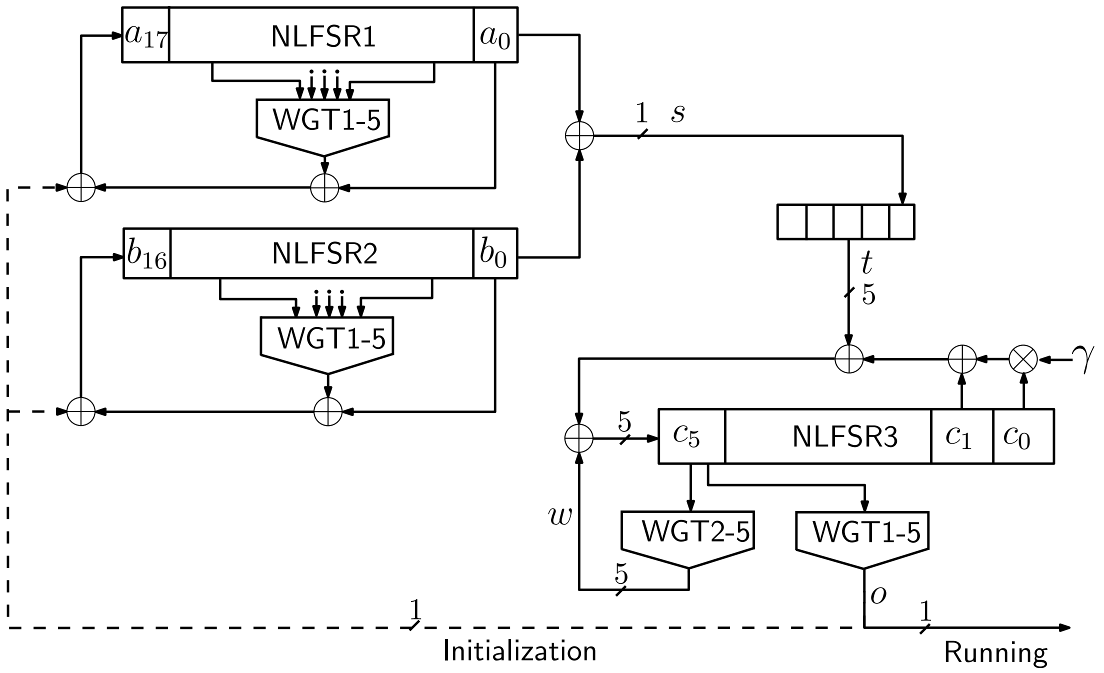


Figure 5.1: Key/IV Initialization and Running Phases of Warbler

The WG-5 transformation of an element $x \in \mathbb{F}_{2^5}$ is constructed from a WG-5 permutation followed by a trace function, where the WG-5 permutation is defined as

$$\text{WGP-5}(x) = x + (x+1)^5 + (x+1)^{13} + (x+1)^{19} + (x+1)^{21},$$

and the trace function is defined as

$$\text{Tr}(x) = x + x^2 + x^{2^2} + x^{2^3} + x^{2^4}.$$

The input to the three WGT1-5 modules use a decimation of degree 3 (*i.e.,* WGT-5(x^3)), and that for the WGT2-5 module use a decimation of degree 1 (*i.e.,* WGT-5(x)). After simplifications, the final equations for the WGT-5 modules are:

$$\begin{array}{lrcl} \text{WGT1-5}: & \text{WGT-5}(x^3) & = & \text{Tr}(x^{13}), \\ \text{WGT2-5}: & \text{WGT-5}(x) & = & \text{Tr}(x^7). \end{array}$$

The rationale of the two WGT1-5 modules (with decimation 3) used for the feedback of NLF-SR1 and NLSFR2 is to guarantee their maximum sequence period. The WGT1-5 module used for the filtering purpose of NLFSR3 is to provide larger nonlinearity, algebraic immunity, and algebraic degree. In addition, the WGT2-5 module used for the feedback of NLFSR3 is to provide higher nonlinearity for its internal states.

As shown in Figure 5.1, the NLFSR1 generates a sequence $\mathbf{a} = \{a_i\}_{i \geq 0}$, where $a_i \in \mathbb{F}_2$; the NLFSR2 generates a sequence $\mathbf{b} = \{b_i\}_{i \geq 0}$, where $b_i \in \mathbb{F}_2$; the NLFSR3 generates a sequence $\mathbf{c} = \{c_i\}_{i \geq 0}$, where $c_i \in \mathbb{F}_{2^5}$. A new sequence $\mathbf{s} = \{s_i \mid s_i = a_i \oplus b_i, i \geq 0\}$ is generated based on the sequences **a** and **b**, and will be sent to a 5-bit shift register (SR) immediately. The element of this 5-bit shift register is used as one feedback for NLFSR3, and it is represented as $t_k \in \mathbb{F}_{2^5},\ k \geq 0$. The feedback primitive polynomial for NLFSR3 is

$$g(x) = x^6 + x + \gamma$$

over $\mathbb{F}_{2^5}$ and $\gamma = \alpha^{15}$. Therefore, the recursive relation for NLFSR3 is defined as

$$c_{k+6} = \gamma c_k + c_{k+1} + w_k + t_k,$$

where the least significant bit of w_k is generated by the WGT2-5 module from the most significant element of NLFSR3, and the other bits of w_k are set to all zeros.

Warbler has a 65-bit Key $(K_0, K_1, K_2, \cdots, K_{64})$ and a 65-bit IV $(IV_0, IV_1, IV_2, \cdots, IV_{64})$. The Key and IV are mixed by XORing each bit together ($m_i = K_i \oplus IV_i, 0 \leq i \leq 64$) to generate the 65-bit internal states of NLFSR1, NLFSR2, and NLFSR3. A similar Key and IV mixing scheme can be found in AES [6] and CBC mode of block ciphers [23]. Then the 65-bit internal states m_i are loaded into registers in the NLFSRs as follows.

$$\begin{aligned}
a_{17}, \cdots, a_0 &= m_{17}, \cdots, m_0, \\
b_{16}, \cdots, b_0 &= m_{34}, \cdots, m_{18}, \\
c_0 &= m_{39}, \cdots, m_{35}, \\
c_1 &= m_{44}, \cdots, m_{40}, \\
c_2 &= m_{49}, \cdots, m_{45}, \\
c_3 &= m_{54}, \cdots, m_{50}, \\
c_4 &= m_{59}, \cdots, m_{55}, \\
c_5 &= m_{64}, \cdots, m_{60}.
\end{aligned}$$

A 45-bit Key and a 20-bit IV case for generating the 65-bit internal states of Warbler are described in [72, 114]. However, the 65-bit Key scenario here provides higher security level than the 45-bit Key case due to the increased key size.

After loading the internal states, a 36 clock cycle initialization phase is performed to mix the internal states and then follows the running phase where the output sequence is generated.

$$\text{NLFSR1: } \begin{cases} x &= (a_{k+4}, a_{k+7}, a_{k+8}, a_{k+10}, a_{k+15}), \\ o_0 &= 0, \\ a_{k+18} &= a_k \oplus \textsf{WGT-5}(x^3) \oplus o_k, \quad 0 \leq k \leq 35. \end{cases}$$

$$\text{NLFSR2: } \begin{cases} y &= (b_{k+4}, b_{k+7}, b_{k+8}, b_{k+9}, b_{k+12}), \\ o_0 &= 0, \\ b_{k+17} &= b_k \oplus \textsf{WGT-5}(y^3) \oplus o_k, \quad 0 \leq k \leq 35. \end{cases}$$

$$\text{5-bit SR: } \begin{cases} s_j &= 0, \quad j = 0, 1, 2, 3, \\ s_{k+4} &= a_k \oplus b_k, \\ t_k &= (s_k, s_{k+1}, s_{k+2}, s_{k+3}, s_{k+4}),\ 0 \leq k \leq 34. \end{cases}$$

$$\text{NLFSR3: } \begin{cases} w_k &= (0, 0, 0, 0, \textsf{WGT-5}(c_{k+5})), \\ c_{k+6} &= \gamma c_k + c_{k+1} + w_k + t_k, \\ o_{k+1} &= \textsf{WGT-5}(c_{k+5}^3), \quad 0 \leq k \leq 34. \end{cases}$$

Equation 1. Initialization Method

In the initialization phase, the output signal $\{o_i\}_{i\geq 0}$ from the WGT1-5 module in NLFSR3 is used as a feedback to the inputs of NLFSR1 and NLFSR2 in every clock cycle. The Warbler initialization method is described in Equation 1. The first output sequence bit o_0 from NLFSR3 is manually set to 0, which is used for the feedback from NLFSR3 to NLFSR1 and NLFSR2 in the first initialization clock cycle. The reason for this setting is that there is a 5-bit shift register between NLFSR1 and NLFSR2 together and NLFSR3. The result of $a_0 \oplus b_0$ needs to take one clock cycle in order to shift into this 5-bit shift register. Therefore, NLFSR3 needs to wait for one clock cycle until the first element t_0 in the 5-bit shift register is ready for the feedback computation of NLFSR3. As a result, in the initialization phase, the NLFSR1 and NLFSR2 run for 36 clock cycles, and the NLFSR3 runs only for 35 clock cycles.

After the NLFSRs finish the initialization phase, they simultaneously go to the running phase. The random sequences $\{o_i\}_{i\geq 36}$ are generated from the output of the WGT1-5 module in NLFSR3 in this phase, and it is worth noting that there is no feedback from WGT1-5 module in NLFSR3 to the inputs of NLFSR1 and NLFSR2 as shown in Figure 5.1. The sequences of **a** and **b** generated in this phase are *span* 18 and *span* 17 sequences respectively. The running method is shown in Equation 2. It takes five clock cycles to obtain t_k from the 5-bit shift register, which results in a $1/5$ bpc throughput of the Warbler output sequence.

$$\text{NLFSR1: } \begin{cases} x &= (a_{k+4}, a_{k+7}, a_{k+8}, a_{k+10}, a_{k+15}), \\ a_{k+18} &= a_k \oplus \textsf{WGT-5}(x^3), \quad k \geq 36. \end{cases}$$

$$\text{NLFSR2: } \begin{cases} y &= (b_{k+4}, b_{k+7}, b_{k+8}, b_{k+9}, b_{k+12}), \\ b_{k+17} &= b_k \oplus \textsf{WGT-5}(y^3), \quad k \geq 36. \end{cases}$$

$$\text{5-bit SR: } \begin{cases} s_{k+4} &= a_k \oplus b_k, \\ t_k &= (s_{5(k-27)-1}, s_{5(k-27)}, s_{5(k-27)+1}, \\ &= s_{5(k-27)+2}, s_{5(k-27)+3}), \quad k \geq 35. \end{cases}$$

$$\text{NLFSR3: } \begin{cases} w_k &= (0, 0, 0, 0, \textsf{WGT-5}(c_{k+5})), \\ c_{k+6} &= \gamma c_k + c_{k+1} + w_k + t_k, \\ o_{k+1} &= \textsf{WGT-5}(c_{k+5}^3), \quad k \geq 35. \end{cases}$$

Equation 2. Running Method

5.2 ASIC Architecture

In this section, we first provide the top-level architecture of Warbler and then present the architectures of FSM and datapath.

5.2.1 Entire Architecture

We provide a top-level architecture for Warbler in Figure 5.2, which includes FSM and datapath. FSM is used to provide the control signals for the datapath. The datapath is used to load the initial data for the registers (Section 5.1) using the input ports (i_d1, i_d2, i_d3), to process the internal sates in the registers, and then to output the sequence (o_data) and the valid signal (o_valid).

Compared to the entire architecture of Warbler in [114], we add i_valid signal to control when to input the valid data and when to generate the output sequence, and it is important for the real applications. As a result, ce1, ce2, ce3, ce5 chip-enable signals are created to control the internal shift registers. By doing this extension, we can improve usability of Warbler.

5.2.2 FSM

Our architecture has three states: loading, initialization, and running. The loading state takes 18 clock cycles, the initialization state lasts for 36 clock cycles, and the running state lasts forever

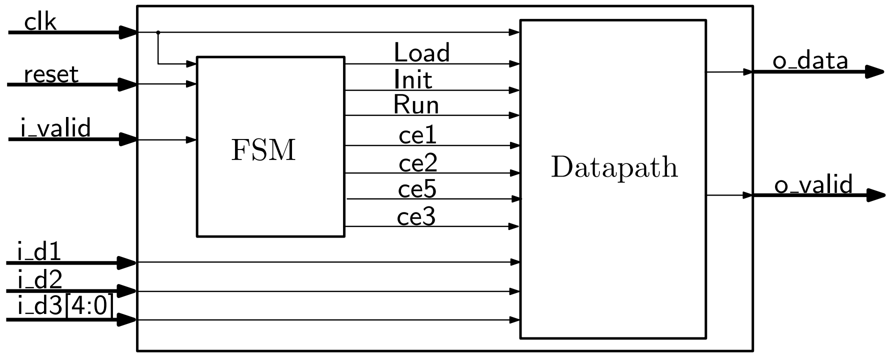


Figure 5.2: The Top-level Architecture of Warbler

unless Warbler is reset. Specifically, our FSM goes into loading state immediately when reset equals 1. Warbler reads the initial data from i_d1 into the datapath once reset goes to 0 again, and it reads data from i_d2 and data from i_d3 in the 2th and 13th clock cycles of the loading state respectively. Once the loading state is finished, the initialization and running states will run. The state transition signal Load stays at 1 when the FSM is in the loading state; otherwise, it equals 0. The similar case is for the Init and Run signals.

The chip-enable signals (ce1, ce2, ce3, ce5) for throughput of $1/5$ bpc are generated according to the states and i_valid values. The ce1, ce2, ce5 signals are set to i_valid in the loading state, and 1 in the initialization states, and i_valid in the running state. Due to the 5-bit shift register, the ce3 is set to 1 in the loading state when i_valid is asserted, 1 in the initialization state except the first clock cycle, and $(0, 0, 0, 0, 1)$ in every five clock cycles in the running state when i_valid is asserted.

Recently, LFSR based counters have been used to replace binary counter in the FSM in hardware [64], because the combinational logic of the LFSR counter is smaller than the full-adder of the binary counter. We evaluate both options in Warbler. To design our LFSR counter, we use a primitive polynomial ($X^6 + X + 1$) with an initial value (1, 1, 1, 1, 1, 1). We use one-hot encoding for the three states: loading (100), initialization (010), and running (001). For the binary counter-based design, the counter starts from 0 in each state. Similarly, the counter starts from 63 in each state for the LFSR counter-based design. The states transition conditions for these two designs are summarized in Table 2.

Table 5.1: States Transition Conditions for FSM

States	Binary counter-based	LFSR counter-based
Loading (100) $\rightarrow$ initialization (010)	17	17
Initialization (010) $\rightarrow$ running (001)	35	39

5.2.3 Datapath

The datapath for Warbler for throughput of 1/5 bpc in our ASIC architecture is shown in Figure 5.3. This figure follows directly from Figure 5.1. It includes five parts: NLFSR1, NLFSR2, NLFSR3, Shift5, and o_valid. NLFSR1 contains a 18-stage register, a WGT1-5 module, and other feedback logic. Similarly, NLFSR2 contains a 17-stage register, a WGT1-5 module, and other feedback logic. NLFSR3 contains a 6-stage register, a Gamma_Mult module, a WGT1-5 module, a WGT2-5 module, and other feedback logic. Shift5 is used for the 5-bit shift register and is comprised of a 5-stage register and other combinational logic. Moreover, o_valid provides a valid signal for the Warbler output sequence.

According to Section 5.1, the feedback values vary for different states. Therefore, Init, Load, and ce3 are used to select the correct feedback values for the NLFSRs in each state. Furthermore, the ce3 is used to control the throughput of the output sequence (o_data) and the output valid signal (o_valid).

The Gamma_Mult module is used for the calculation of γc_k in $\mathbb{F}_{2^5}$. Under the polynomial basis representation, the element $X \in \mathbb{F}_{2^5}$ ($X = x_0 + x_1\alpha + x_2\alpha^2 + x_3\alpha^3 + x_4\alpha^4$) multiplied by $\gamma = \alpha^{15}$ can be computed as follows:

$$\begin{aligned} X \cdot \alpha^{15} &= (x_0 + x_1\alpha + x_2\alpha^2 + x_3\alpha^3 + x_4\alpha^4) \cdot \alpha^{15}, \\ &= x_0\alpha^{15} + x_1\alpha^{16} + x_2\alpha^{17} + x_3\alpha^{18} + x_4\alpha^{19}, \\ &= (x_2 + x_4) + (x_2 + x_3 + x_4)\alpha + \\ &\quad (x_0 + x_3 + x_4)\alpha^2 + (x_0 + x_1 + x_2)\alpha^3 + \\ &\quad (x_1 + x_3 + x_4)\alpha^4. \end{aligned}$$

Therefore, the result of $X \cdot \gamma$ is represented as a 5-bit vector $(x_2 \oplus x_4, x_2 \oplus x_3 \oplus x_4, x_0 \oplus x_3 \oplus x_4, x_0 \oplus x_1 \oplus x_2, x_1 \oplus x_3 \oplus x_4)$. Thus, we can implement our Gamama_Mult module by using the finite field logic directly.

Similarly, we can compute WGT-5(x^3) and WGT-5(x) in polynomial basis for every $x \in \mathbb{F}_{2^5}$ by using the finite field logic directly or pre-storing them to two constant arrays (WGT1-5 and WGT2-5 respectively), as in Chapter 4. However, the hardware implementations of WGT1-5

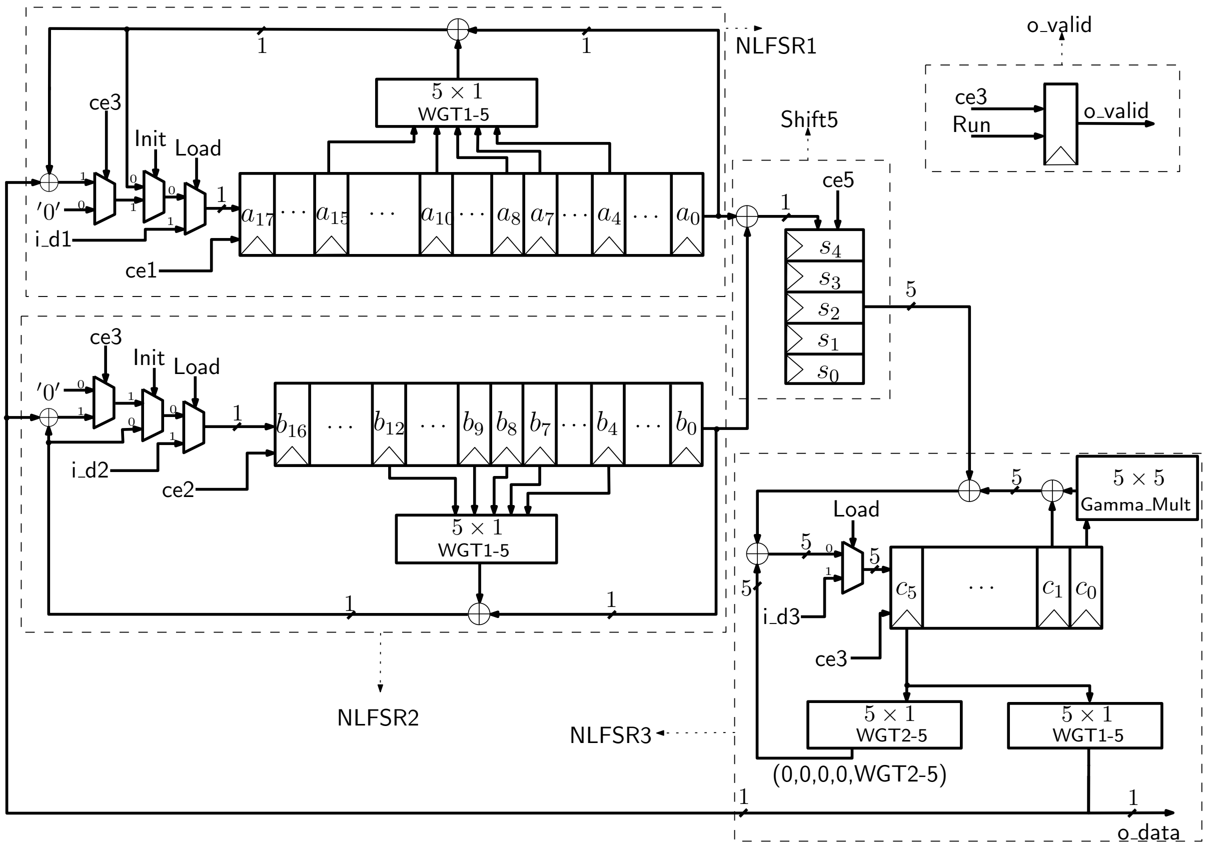


Figure 5.3: Datapath of Warbler for Throughput of 1/5 bpc

module (with decimation 3) and WGT2-5 module (with decimation 1) are more efficient if the constant array based method is used rather than the finite field logic methods, shown in Chapter 4. Therefore, we use the constant array based methods for implementing the WGT1-5 and WGT2-5 modules.

5.2.4 Throughput Improvement

It is important to have a design with a throughput of 1 bpc for the passive RFID systems, as illustrated in [100]. Therefore, we consider the following two options (1 and 2) in Table 5.2, in order to improve the throughput of Warbler from $1/5$ bpc to 1 bpc. In the original architecture as described in Sections 5.2.2 and 5.2.3, the throughput of NLFSR1/2 in initialization and running states are both 1 bpc, and that of NLFSR3 in these two states are 1 and $1/5$ bpcs respectively.

Table 5.2: Throughput Improvement of Warbler

Warbler	**States**	**Throughput (bpc)**	
		NLFSR1/2	NLFSR3
Original	Initialization	1	1
	Running	1	$1/5$
Option 1	Initialization	1	1
	Running	5	1
Option 2	Initialization	5	5
	Running	5	1

We increase the throughput of the running state by five times, but keep the throughput of the initialization state unchanged for option 1. In this case, we need four extra WGT1-5 modules and four extra XORs in the recursive relations for both NLFSR1 and NLFSR2, and four extra XORs to generate the 5-bit values for the feedback function of NLFSR3 from the first five registers of NLFSR1 and NLFSR2. Because we use parallel NLFSR1 and NLFSR2 in the running state but serial NLFSR1 and NLSFR2 in the loading and initialization states, we need 33 extra multiplexers (MUXes) to select different inputs to each register in the NLFSR1 and NLFSR2 for the loading and initialization states, and running state. Therefore, the total extra area needed for improving the throughput from $1/5$ bpc to 1 bpc is 8 WGT1-5s + 8 XORs + 33 MUXes + 4 XORs.

The hardware architecture for option 1 is shown in Figure 5.4. From it, we can clearly see the 33 multiplexers, four more WGT1-5 modules, and four more XORs in the architecture of NLFSR1 and NLFSR2, and four more XORs for the feedback computation of NLFSR3. In addition, by adding these four XORs, we need to add an extra multiplexer to select the 5-bit feedback values from the Shift5 module in the initialization state and from the five XORs (shown in Figure 5.4) in the running state for NLFSR3. As a result, the ce3 signal in the FSM should be adjusted to always equal i_valid in the running state. For the ce1 and ce2 signals, they should stop at the last clock cycle of the initialization state. The reason for this change is that the feedback value for NLFSR3 is directly taken from the first five registers of NLFSR1 and NLFSR2 in the running state of the architecture with throughput of 1 bpc. However, there is one clock cycle delay for loading the data from NLFSR1 and NLFSR2 to shift5 module in the running state of the architecture with throughput of $1/5$ bpc.

For option 2, we increase the throughput for both the initialization and running states by five times. Hence, we need to use four extra copies of all the modules in the feedback functions of NLFSR1, NLFSR2, and NLFSR3. Besides, we need four extra XORs to output the 5-bit values for the feedback function of NLFSR3 from NLFSR1 and NLFSR2. However, in order to reuse this five XORs' output for the initialization phase, we have to change the behavior of Warbler by reducing the 5-bit shift register but using this five XORs instead. In this case, even though the required clock cycles for the initialization state are decreased, but the total extra area needed to increase the throughput is 8 WGT1-5s + 8*2 XORs + 4 XORs + 4 Gamma_Mults + 4 * 3 XORs + 4 WGT2-5s + 4 WGT1-5s - 5 Registers, which is considerably larger than that of option 1. Therefore, option 1 is selected for the architecture with throughput of 1 bpc, shown in Figure 5.4, considering both the functionality and extra area consumption.

5.3 Results Evaluation

We use the same design flow and metrics as Simeck and SIMON in Chapter 3. We first present the hardware architecture and then provide the ASIC implementation results in CMOS 130nm and CMOS 65nm. Then, we give a comprehensive analysis of the area that use different technologies and various compilation techniques.

5.3.1 ASIC Results

We use the three different compilation techniques in Design Compiler to perform hardware optimizations: simple compile, compile ultra, and compile ultra with clock gating.

The metric in our ASIC optimization is the low-area implementation, while still maintain a very high maximum frequency. In general, when the clock period constraint in the Design Compiler is very small, we can get a circuit with higher maximum frequency but also bigger area. However, in order to get a low-area implementation, we make a trade-off between the area and the maximum frequency by setting the clock period constraint loosely. Based on this, the ASIC implementation results of Warbler by using our architecture with throughput of $1/5$ and 1 bpcs in CMOS 65nm and CMOS 130nm are shown in Table 5.3. Notably, the area, maximum frequency, and power consumption results are provided by using compile ultra and clock gating techniques.

From Table 5.3, we can see that, for the throughput of $1/5$ bpc case, the area of the LFSR counter-based design is smaller than that of binary counter-based design (*i.e.*, 6 GEs smaller in both CMOS 65nm and CMOS 130nm after the place and route phase). Similarly, the total power consumption of the LFSR counter-based design is smaller than that of the binary counter-based design in both CMOS 65nm and CMOS 130nm. As a result, only the LFSR counter-based design for throughput of 1 bpc is provided. The smallest area of Warbler for throughput of $1/5$ and 1 bpcs are 513 GEs and 750 GEs respectively in CMOS 65nm and that are 551 GEs and 750 GEs

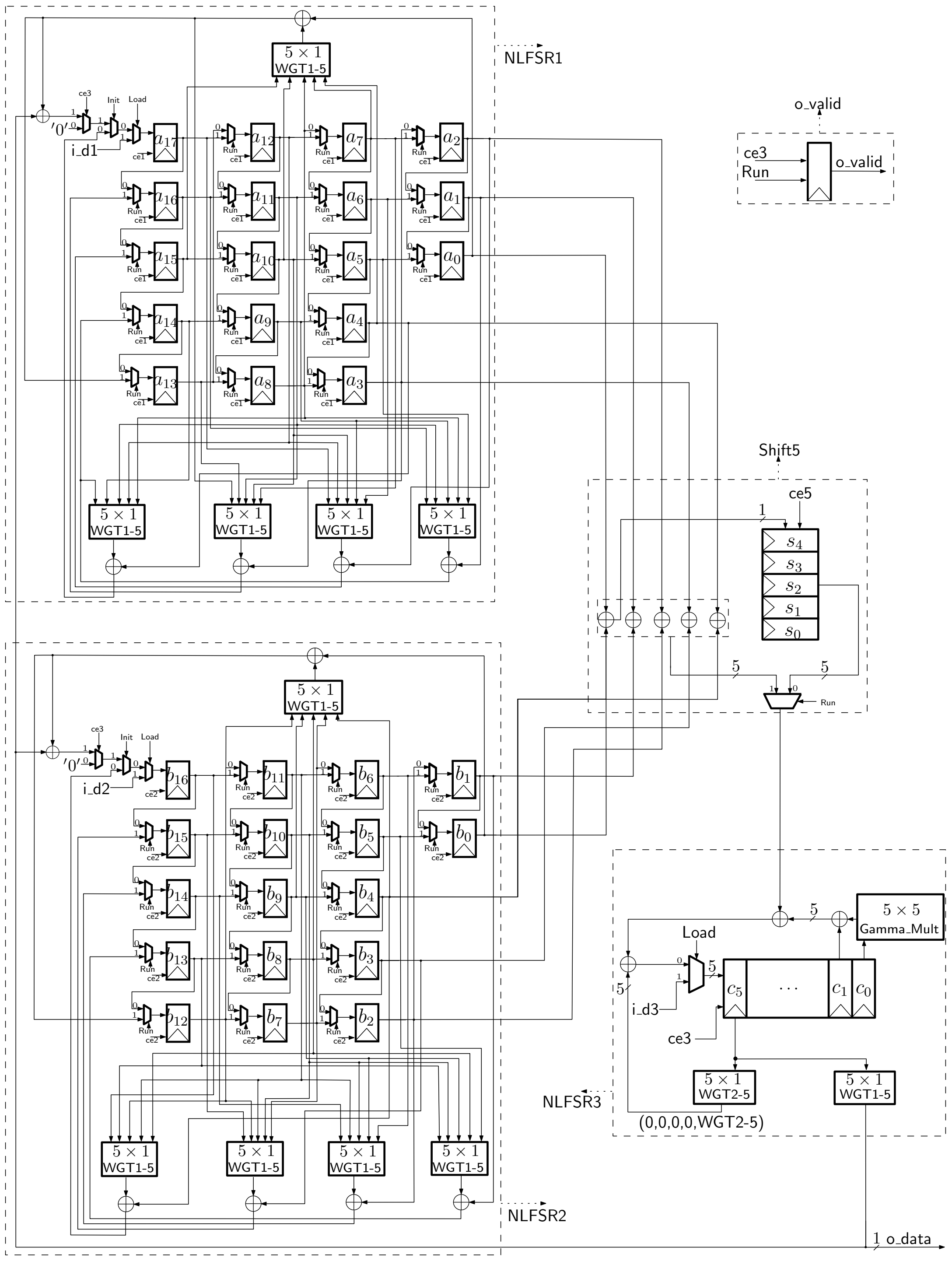


Figure 5.4: Datapath of Warbler for Throughput of 1 bpc

Table 5.3: Our Implementation Results of Warbler in CMOS 65nm and CMOS 130nm

Warbler	**Tech** (nm)	**Area** (GEs) Before P&R	After P&R	**Max Freq** (MHz)	**Throughput** (bpc)	**Total Power** @100KHz (μW)	**Total Power** @2MHz (μW)	**Optimality** (TP/(#GEs *power)*10^3)
Binary counter-based	65	483	519	1808	$1/5$	1.800	5.340	0.071
LFSR counter-based		477	513	1872	$1/5$	1.760	5.200	0.075
LFSR counter-based		698	750	1328	1	2.212	8.045	0.165
Binary counter-based	130	512	557	235	$1/5$	0.299	5.900	0.059
LFSR counter-based		507	551	233	$1/5$	0.279	5.500	0.066
LFSR counter-based		690	750	207	1	0.497	9.837	0.135

Table 5.4: The Sequential Logic Ratios of Warbler

Warbler	**Throughput** (bpc)	**CMOS 65nm** Compile simple	Compile ultra	Compile ultra + clock gating	**CMOS 130nm** Compile simple	Compile ultra	Compile ultra + clock gating
Binary counter-based	1/5	65.5%	67.0%	65.6%	72.8%	72.5%	72.1%
LFSR counter-based	1/5	65.6%	67.2%	66.6%	73.6%	75.0%	72.8%
LFSR counter-based	1	47.9%	43.4%	46.2%	55.7%	53.4%	55.1%

respectively in CMOS 130nm. For the total power consumption at 100 KHz, the static power consumption dominates the total power consumption because the operating frequency is so low. However, it is opposite in the 2 MHz case, where the dynamic power consumption increases a lot. The static power consumption is larger in CMOS 65nm than that in CMOS 130nm. Therefore, the total power consumption at 100 KHz in CMOS 65nm is larger than that in CMOS 130nm, and it is the opposite case for 2 MHz case, as shown in Table 5.3.

We improve the throughput from $1/5$ bpc to 1 bpc through increasing the area by 237 GEs (a 46% increased amount from 513 GEs to 750 GEs) in CMOS 65nm, by 199 GEs (a 36% increased amount from 551 GEs to 750 GEs) in CMOS 130nm. The maximum frequency for the throughput of 1 bpc architecture is slower than that of throughput of $1/5$ bpc architecture in both CMOS 65nm and CMOS 130nm. This is because the critical path for the throughput of 1 bpc is from register a_8 to register a_{17} as shown in Figure 5.4, and that for throughput of $1/5$ bpc is between the two registers of the counter in the FSM. The maximum tap position of WGT1-5 module for NLFSR1 is 15, but the maximum state index of NLFSR1 is 17. As a result, we need to have a chain of 2 WGT1-5 modules in order to provide five feedback values for the NLFSR1 in each clock cycle during the parallelism of NLFSR1, which incurs the increase of the maximum clock period. From Figure 5.4, we can clearly see that there are 2 linked WGT1-5 modules along the path from a_8 to a_{17}.

The optimality is defined as throughput/(area (after place and route)* total power at 2 MHz). For the throughput of $1/5$ bpc, the optimality of LFSR counter-based design is slightly larger than that of binary counter-based design. However, the optimality for the design with throughput of 1 bpc is much larger than that of the design with throughput of $1/5$ bpc.

Our another observation is that the sequential logic dominates the area in the throughput of $1/5$ bpc case. The proportion of sequential logic (Table 5.4) in the area before the place and route phase in this case, depends on the adopted technologies and compilation techniques, are all above 65%. However, due to the increased combinational area to improve the throughput, the sequential logic proportion has been reduced to around 50% for 1 bpc.

5.3.2 Results Analysis

In order to thoroughly analyze the constitution of the area of Warbler in CMOS 65nm and CMOS 130nm, we break down the area from before the place and route phase into separate submodules, as shown in Table 5.5.

The areas of the datapath for the binary counter-based and the LFSR counter-based designs are the same under the same technology, and only the areas of the FSM are different. The LFSR counters are 9.4% and 10.0% smaller than the binary counters in CMOS 65nm and CMOS 130nm respectively for throughput of $1/5$ bpc. However, the total areas of the FSM are essentially the same for CMOS 65nm and reduced by 5.7% for CMOS 130nm, due to the area of state transitions and chip-enable logic are larger in the LFSR counter-based FSM than that in the binary counter-based FSM. By using compile ultra technique, we can reduce the area by optimizing the hardware design as a whole. We can see that compile ultra decreases the area by from 2.2% to 8.1% for all our designs as shown in Table 5.6, compared to using the compile simple technique directly. Furthermore, we can change all the registers with chip-enable signals to registers without chip-enable signals and a latch by using the clock gating technique. As a result, with the combination of compile ultra, the areas are decreased a lot by from 12.5% to 20.9% for all our designs, as shown in Table 5.6.

The areas of combinational logic depend on the areas of basic gates in the different technologies. For example, the areas of the WGT1-5 module are different in CMOS 65nm and CMOS 130nm, which are 15.50 GEs and 14.50 GEs respectively. The same situation exists for other submodules. Moreover, the WGT1-5 and WGT2-5 modules are different in the same technology (*i.e.*, 15.50 GEs and 9.25 GEs respectively in CMOS 65nm, and 14.50 GEs and 9.50 GEs respectively in CMOS 130nm). We give the specific contents of these two constant arrays in Table 5.7. As we can see, the position distributions of 1 and 0 are different for the WGT1-5 and WGT2-5 modules, and they are computed based on WG-5 transforms with decimation 3 and 1

Table 5.5: Breakdown of the Implementation Results of Warbler before the Place and Route Phase

			CMOS 65nm			CMOS 130nm		
			Throughput (bpc)			Throughput (bpc)		
			1/5		1	1/5		1
Components			Binary counter-based (GEs)	LFSR counter-based (GEs)	LFSR counter-based (GEs)	Binary counter-based (GEs)	LFSR counter-based (GEs)	LFSR counter-based (GEs)
FSM	State transitions + chip_enable logic		38.50	43.75	41.50	38.75	39.00	39.75
	Counter		55.75	50.50	41.50	57.50	51.75	47.25
Datapath	NLFSR1	18-stage register	90.00		90.00	108.00		108.00
		WGT1-5	15.50		77.50	14.50		72.50
		Other feedback logic	10.00		53.25	9.50		47.50
	NLFSR2	17-stage register	85.00		85.00	102.00		102.00
		WGT1-5	15.50		77.50	14.50		72.50
		Other feedback logic	10.00		52.25	9.50		45.75
	NLFSR3	6-stage register	150.00		150.00	180.00		180.00
		Gamma_Mult	13.75		13.75	14.00		14.00
		WGT1-5	15.50		15.50	14.50		14.50
		WGT2-5	9.25		9.25	9.50		9.50
		Other feedback logic	32.50		32.50	30.75		30.75
	Shift5	5-stage register	18.75		18.75	32.50		32.50
		Other combinational logic	15.00		32.50	2.75		19.75
	O_valid		6.50		7.50	8.50		7.25
Totals	Compile simple		581	581	798	647	641	845
	Compile ultra		569	556	768	618	613	780
	Compile ultra + clock gating		483	477	698	512	507	690

Table 5.6: The Area Reduction Percentages by using Compile Ultra and Compile Ultra plus Clock Gating

Techniques	CMOS 65nm			CMOS 130nm		
	Throughput (bpc)			Throughput (bpc)		
	1/5		1	1/5		1
	Binary counter-based	LFSR counter-based	LFSR counter-based	Binary counter-based	LFSR counter-based	LFSR counter-based
Compile ultra	2.2%	4.3%	3.8%	4.5%	4.4%	8.1%
Compile ultra + clock gating	17.0%	17.9%	12.5%	20.8%	20.9%	18.3%

respectively. The synthesis tool is able to optimize WGT2-5 to the simpler logic in hardware than WGT1-5; therefore, the area of WGT1-5 is bigger than WGT2-5's as shown in Table 5.5.

Table 5.7: The WGT1-5 and WGT2-5 Constant Arrays

Address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
WGT1-5	0	0	1	0	0	1	1	1	0	1	1	1	1	0	0	0	1	0	1	0	0	0	1	1	0	1	1	0	0	1	0	1
WGT2-5	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1

The distinct areas of different WG-5 transformation tables give us a worthwhile idea to select a decimation value in order to make the WG-5 transformation table as small as possible for a new primitive design.

Furthermore, the area of the FSM and the area of the counter in the throughput of 1 bpc case are smaller than that in the throughput of $1/5$ bpc case because the ce3 signal is set to 1 in each clock cycle instead of 1 in every five clock cycles. For example in CMOS 130nm, the area of LFSR counter in throughput of 1 bpc is 47.25 GEs, whereas that in throughput of $1/5$ bpc is 51.75 GEs. Similarly in CMOS 130nm, the area of FSM in throughput of 1 bpc is 87.00 GEs, which is smaller than 90.75 GEs, the area of FSM in throughput of $1/5$ bpc. For the areas of NLFSR1 and NLFSR2 in the throughput of 1 bpc case, we can clearly see that the WGT1-5 module is five times larger than that in the throughput of $1/5$ bpc case. The total areas of their feedback logic are increased by 85.50 GEs and 84.25 GEs respectively for CMOS 65nm and CMOS 130nm, which are slightly smaller than the estimated area of the increased gates (8 XORs + 33 MUXes) in Figure 5.4 for NLFSR1 and NLFSR2. The area of NLFSR3 is identical for both cases. The area of shift5 module is increased by 17.50 GEs and 17.00 GEs respectively in CMOS 65nm and CMOS 130nm because the increased four XORs and one 5-bit width multiplexer. Overall, we can improve the throughput of Warbler without costing the area too much.

Compared with our results in [114], the area is increased by 15 GEs and 17 GEs in CMOS 65nm and CMOS 130 nm respectively for our new architecture with throughput of $1/5$ bpc, which are around 3% of the entire area. However, with this small fraction increase of the area, we can improve the feasibility and usability of Warbler.

5.4 Comparisons with Other Lightweight Primitives

During the past years, a few lightweight PRNGs have been proposed in the literature, such as Melia-Segui *et al.* [78], J3Gen [79], LAMED [91], and AKARI1B [76]. Melia-Segui *et al.*'s PRNG [78] and J3Gen [79] combine a linear shift register (LFSR) with a random source from a truly random number generator (TRNG). The TRNG transforms thermal noises from two registers to random bits, which are updated once for each pseudorandom sequence. They are used to select one of primitive polynomials for the LFSR, which introduces nonlinearity of the LFSR

Table 5.8: Comparisons with Hardware Implementations of Lightweight Primitives

Algorithms		Key Size	IV/ Block Size*	Internal State Size	Area (GEs) Before P&R	Area (GEs) After P&R	Max Freq (MHz)	TP @100KHz (Kbps)	Total Power @100KHz	Tech (nm)	Source
PRNG	Warbler	65	65	65	**507**	**551**	233	20.00	0.279 μW	130	**here**
	Warbler	65	65	65	**477**	**513**	1872	20.00	1.760 μW	65	**here**
	Warbler	65	65	65	**690**	**750**	207	100.00	0.497 μW	130	**here**
	Warbler	65	65	65	**698**	**750**	1328	100.00	2.212 μW	65	**here**
	Warbler	45	20	65	–	937	–	20.00	–	65	[73]
	Melia-Segui *et al.*	16	0	16	$761^{\triangle}$	–	–	–	–	–	[78]
	J3Gen	64	0	64	$1419^{\triangle}$	–	–	–	–	–	[79]
	LAMED	32	32	64	$1585^{\triangle}$	–	–	–	–	–	[91]
	AKARI1B	–	–	64	1749	–	–	14.20	0.182 $\mu W^{\triangledown}$	90	[76]
Stream Cipher	Grain	80	64	160	–	1259	–	100.00	0.780 μW	130	[7]
	Trivium	80	80	288	–	2088	–	100.00	1.440 μW	130	[7]
	Grain	80	64	160	–	1126	1020	100.00	–	65	[115]
	Trivium	80	80	288	–	1986	962	100.00	–	65	[115]
Block Cipher	Simeck$^{\diamond}$	64	32	96	505	549	292	5.60	0.417 μW	130	[116]
	SIMON$^{\diamond}$	64	32	96	517	562	331	5.60	0.421 μW	130	[116]
	SPECK$^{\diamond}$	64	32	96	580	–	–	4.20	–	130	[11]
	Simeck$^{\diamond}$	64	32	96	454	488	1754	5.60	1.292 μW	65	[116]
	SIMON$^{\diamond}$	64	32	96	466	501	1428	5.60	1.311 μW	65	[116]
Hash Function	PHOTON-80/20/16	○	○	100	865	–	–	2.82	–	180	[48]
	SPONGENT-88	○	○	88	738	–	–	0.81	1.570 μW	130	[16]

* IV is for PRNGs and stream ciphers, and Block is for block ciphers.
○ The corresponding value is not related; – The corresponding value is not provided by the authors.
$^{\triangle}$ The estimated area; $^{\triangledown}$ The estimated power consumption in UMC Faraday 90nm library.
$^{\diamond}$ The smallest one in the Simeck, SIMON, SPECK families.

and improves the corresponding security. The Melia-Segui *et al.*'s PRNG contains eight primitive polynomials candidate and a LFSR with length 16, while J3Gen contains a variety of sizes for the number of primitive polynomials and length of LFSR. In addition, their pseudorandom sequences satisfy the EPC C1 G2 standard's randomness test. LAMED [91] is designed to be able to generate both 32-bit and 16-bit random numbers, which is compatible with the EPC C1 G2 standard, where the 16-bit output is the XOR of two halves of the 32-bit output. The internal states are updated with the ARX (*i.e.*, addition modular 2^{32}, bit rotations, and XOR operation) structure. The random numbers of LAMED can pass the NIST randomness tests. The estimated areas of Melia-Segui *et al.*'s PRNG [78] and J3Gen [79] with an internal state size 64 are 761 GEs and 1419 GEs respectively. The estimated area of LAMED [91] is 1585 GEs. AKARI1B [76] is designed based on the T-function and a non-linear filter function, and its output sequence can pass the NIST randomness tests. The area before the place and route phase for AKARI1B [76] with an internal state size 64 is 1749 GEs, synthesized using the UMC Faraday 90nm technology, and the corresponding throughput and estimated power consumption are 14.2 Kbps and 0.182 μW at 100 KHz respectively. We compare our Warbler results with that of those PRNGs in Table 5.8. From the table, we can see that the areas from both before and after the place and route phase of Warbler are smaller than the estimated areas of LAMED, Melia-Segui *et al.*'s PRNG, and J3Gen, and also smaller than the area of AKARI1B in both CMOS 65nm and C-MOS 130nm. More importantly, the area of Warbler at throughput of 20.00 Kbps is around 45% smaller than the result in CMOS 65nm in [73] .

We also compare Warbler with other lightweight primitives (the smallest area of them) in Table 5.8, including stream ciphers, block ciphers, and hash functions. The areas and the total power consumption at 100 KHz of Warbler are both smaller than that of Grain and Trivium in CMOS 65nm and CMOS 130nm, while the maximum frequency of Warbler is larger than that of Grain and Trivium in CMOS 130nm.

The area of Warbler at throughput of 20.00 Kbps is slightly larger than the smallest area of Simeck and smaller than that of SIMON and SPECK in CMOS 130nm. However, it is slightly larger than that of Simeck and SIMON in CMOS 65nm. The maximum frequency of Warbler is larger than that of Simeck and SIMON at throughput of 20.00 Kbps in CMOS 65nm, and smaller than them at throughput of 100.00 Kbps. Meanwhile, the maximum frequency of Warbler at CMOS 130nm is smaller than that of Simeck and SIMON. The total power consumption at 100 KHz for Warbler in CMOS 130nm is smaller than that of Simeck and SIMON at throughput of 20.00 Kbps, but larger than them at throughput of 100.00 Kbps. However, it is larger than that of Simeck and SIMON in CMOS 65nm.

Similarly, the area of Warbler is smaller than that of PHOTON-80/20/16 and SPONGENT-88, and its power consumption at 100 KHz is smaller than that of SPONGENT-88 in CMOS 130nm.

Overall, Warbler has a small area and power consumption with a flexibility in throughput, and hence it is very suitable for passive RFID applications.

5.5 Summary

In this chapter, we have presented hardware implementations and optimizations of Warbler in CMOS 65nm and CMOS 130nm ASICs. We have proposed an architecture for hardware implementations of Warbler and with thorough analysis. We can achieve the areas of 551 GEs and 513

GEs after the place and route phase in CMOS 130nm and CMOS 65nm respectively for throughput of $1/5$ bpc. More importantly, we have improved the throughput from $1/5$ bpc to 1 bpc by increasing 199 GEs and 237 GEs area respectively in CMOS 130nm and CMOS 65nm. By doing so, the corresponding total power consumption has increased by 0.218 μW and 0.452 μW at 100 KHz respectively, and 4.337 μW and 2.845 μW at 2 MHz respectively. We have determined that the LFSR counter-based design is better than the binary counter-based design in terms of smaller area and lower total power consumption. Moreover, the sequential logic ratios for all our designs are bigger than 65% for throughput of $1/5$ bpc and are around 50% for throughput of 1 bpc. Our analysis has verified that the areas of NLFSRs and combinational logic are dependent upon the type of registers and the adopted technologies. The area of the WG-5 transformation table depends upon the selected decimation value, giving us some suggestions for future ciphers and pseudorandom number generator designs using WG-5 transformations. When compared with other lightweight primitives, the areas of our Warbler implementations are smaller than the estimated areas of LAMED, Melia-Segui *et al.*'s PRNG, and J3Gen, and also smaller than the areas of AKARI1B, Grain, Trivium, PHOTON-80/20/16, and SPONGENT-88 in both CMOS 65nm and CMOS 130nm. Additionally, the area of Warbler is comparable with that of SIMON and Simeck at throughput of $1/5$ bpc. In conclusion, Warbler can fit into passive RFID systems.

Chapter 6

WGLCE: A Cryptographic Engine for Passive RFID Systems

In this chapter, we design a cryptographic engine, which can generate pseudorandom numbers and provide data confidentiality for passive RFID systems. Based on the EPC Class 1 Generation 2 standard [3], the 16-bit random number (RN16) is used in many commands. An encryption/decryption algorithm is needed to meet the security requirements for data confidentiality. During the interrogation process between RFID readers and tags, the RN16s and the encrypted data are sent at different time slots from the tag to the reader. The pseudorandom number generator and the encryption engine never works at the same time. Therefore, we can integrate the pseudorandom number generator and cipher together to build a cryptographic engine.

This chapter is organized as follows. We give an overview of the cryptographic engine in Section 6.1. Then, the hardware implementations of WG-5 is investigated in Section 6.2. In Section 6.3, we explore the architectures of our cryptographic engine in detail. We compares our cryptographic engine with others in Section 6.4. Finally, the conclusion is given in Section 6.5.

6.1 WGLCE: Overview

Instead of using a separate cipher and pseudorandom number generator, we design a cryptographic engine which can perform data encryption/decryption and generate pseudorandom numbers. By doing so, the total area of the security solutions can be reduced and the reusability can be improved. Because the cryptographic engine is a single module, and it can be easily integrated into a complete RFID digital baseband.

Some of the previous works adopt a design of separate encryption engine and pseudorandom number generator. For example, Ertl *et al.* [33] proposed an enhanced security solution for the UHF RFID tag. In their paper, they used AES to achieve encryption/decryption functionality, and the lightweight stream cipher Grain to generate pseudorandom numbers. Todd *et al.* [107] designed a passive RFID system with data confidentiality and pseudorandom numbers as well. In this design, the block cipher Present-80 is used to provide data confidentiality and the pseudorandom number generator LAMED [91] is used to generate pseudorandom numbers. However, the areas of AES plus Grain, and Present-80 plus LAMED are both larger than 2000 GEs.

Different from the previous work, we explore our cryptographic engine by using the Warbler pseudorandom number generator [114] and lightweight stream ciphers (WG-5 [7], WG-7 [67], WG-8 [115]) that are all based on the WG transformation. Therefore, our cryptographic engine is named WGLCE, which represents the lightweight cryptographic engine based on WG transformations. More specifically, WGLCE is a combination of lightweight pseudorandom number generator and lightweight stream cipher. WG-5, WG-7, and WG-8 are all lightweight stream ciphers, but with different security level, hardware cost, and power consumption. In order to achieve a smaller design of WGLCE, we will use WG-5 as an example in this chapter because its area is smaller than both that of WG-7 and WG-8.

According to the security analysis of WG-5, the most effective attack is the discrete fourier transform attack which is related to the linear complexity [7]. The linear complexity of Warbler is at least 2^{18} [72], and the linear complexity of WG-5 with decimation one is 2^{17} [7]. The exchanged information between the reader and the tag is EPC number, RN16, encrypted data, etc. These are very short messages which are below 100 bits. Hence, it is very difficult for an attacker to get 2^{17} consecutive data during the life time of the RFID tag. Therefore, Warbler and WG-5 are both secure enough for the passive RFID tags [72, 7]. As a result, we can use WG-5 as a stream cipher and Warbler as a pseudorandom number generator. Motivated by that the stream cipher Grain was used as the pseudorandom number generator in the passive RFID tags in [33], we have the following design options for our WGLCE in Table 6.1.

Table 6.1: Different Design Options for WGLCE

Option		PRNG	Encryption/Decryption	Discussions
$\surd$	1	Warbler	WG-5	good
	2	WG-5	WG-5	larger
	3	Warbler	Warbler	lower security level
	4	WG-5		extra memory and add delay
	5	Warbler		extra memory and add delay

The area of Warbler is smaller than that of WG-5. Therefore, the option 2 is not a good idea because the area of two WG-5 is larger than that of Warbler plus WG-5. The key length of Warbler is shorter than that of WG-5, which results in a lower security level. Hence, the two combined Warbler designs in option 3 is not as good as option 1 considering the balance between the hardware cost and security. Options 4 and 5 use one hardware module of WG-5 or Warbler, to provide two functionalities. In this case, we have to use the memory to store the internal states of the LFSR when WGLCE is switching between cipher and pseudorandom numbers. Option 5 is not as good as option 4 due to the same reason as option 3. The area of option 4 is less than that of option 1. Nevertheless, option 4 needs extra clock cycles to load and store the LFSR's internal states to the memory, which decreases the performance and increases the energy consumption. Moreover, the operations to load and store the internal states to the memory may be vulnerable to potential side channel attacks. As a result, we choose option 1 for our WGLCE design, where Warbler is used to generate pseudorandom numbers and WG-5 is deployed to provide data confidentiality.

6.2 A New Efficient Hardware Implementations of Ultra-lightweight Stream Cipher WG-5

This section is mainly used to provide background information for the WGLCE. We will present the architecture of WGLCE in next section.

In 2013, Aagaard and Gong [7] proposed an ultra-lightweight stream cipher WG-5 with two versions. One version uses decimation one for the WG transformation function *i.e.*, WGT-5(x), and another one uses decimation eleven for the WG transformation function *i.e.*, WGT-5(x^{11}). These two versions have different security levels and are both sufficiently secure for passive RFID tags. The area of the version with decimation one is smaller than that of the version with decimation eleven. In this section, we present an efficient hardware implementation of WG-5 with decimation one.

6.2.1 Description of WG-5

In this subsection, we first give the parameters for describing the WG-5 and then give a brief description of it.

Parameters for WG-5

- The primitive polynomial $p(x)$ for generating $\mathbb{F}_{2^5}$, the primitive element α, and the trace function are the same as that of Warbler in chapter 5.
- The WG-5 permutation and WG-5 transformation with decimation 1 are the same as that of Warbler in chapter 5.
- $g(x) = x^{32} + x^7 + x^6 + x^5 + x^4 + x + \omega$, a feedback primitive polynomial of degree 32 over $\mathbb{F}_{2^5}$ for LFSR32, where $\omega = \alpha^{15}$.

Behaviour of WG-5

WG-5 is an ultra-lightweight variant of the WG stream cipher family with 80-bit key and 80-bit initial vector (IV). The stream cipher WG-5 consists of a 32-stage LFSR (LFSR32) with the feedback polynomial $g(x)$ followed by a WG-5 transformation module.

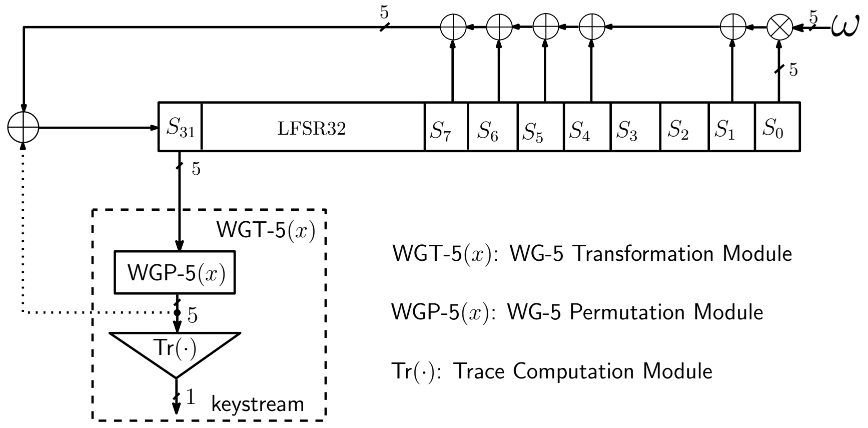

5
5
ω
5
S_{31}
LFSR32
S_7
S_6
S_5
S_4
S_3
S_2
S_1
S_0
5
WGT-5(x)
WGP-5(x)
5
Tr$(\cdot)$
1
keystream
WGT-5(x): WG-5 Transformation Module
WGP-5(x): WG-5 Permutation Module
Tr$(\cdot)$: Trace Computation Module

Figure 6.1: The Initialization and Running Phases of WG-5

The initialization and running phases of the WG-5 are shown in Figure 6.1. Before the initialization and running phases starts, we first load the key and IV to the internal states. Let the 80-bit secret key be $K = (K_0, \ldots, K_{79})$, the 80-bit IV be $IV = (IV_0, \ldots, IV_{79})$, and the internal states of LFSR32 be $S_0, \ldots, S_{31} \in \mathbb{F}_{2^5}$, where $S_i = (S_{i,0}, \ldots, S_{i,4})$ for $i = 0, \ldots, 31$. The initial values of the internal states are computed as follows:

$$\begin{aligned} S_i &= (K_{5i}, \cdots, K_{5i+4}), & i &= 0, 1, \cdots, 15, \\ &= (IV_{5(i-16)}, \cdots, IV_{5(i-16)+4}), & i &= 16, 17, \cdots, 31. \end{aligned}$$

1) Initialization Phase

Once the LFSR32 is loaded with initial states, the apparatus runs for 64 clock cycles. During each clock cycle, the 5-bit internal state S_{31} passes through the nonlinear WG-5 permutation module (i.e., the WGP-5(x)) and the output is used as the feedback to update the internal states of the LFSR32. The LFSR32 update follows the recursive relation:

$$S_{k+32} = (\omega \otimes_5 S_k) \oplus_5 S_{k+1} \oplus_5 S_{k+4} \oplus_5 S_{k+5} \oplus_5 S_{k+6} \oplus_5 S_{k+7} \oplus_5 \text{WGP-5}(S_{k+31}),$$

where $0 \leq k < 64$.

2) Running Phase

After the initialization phase, WG-5 goes into the running phase and 1-bit keystream is generated in each clock cycle. During the running phase, the 5-bit internal state S_{31} passes through the nonlinear WG-5 transformation module (i.e., WGT-5(x)) and the output is the 1-bit keystream. The recursive relation for updating the LFSR32 is given below:

$$S_{k+32} = (\omega \otimes_5 S_k) \oplus_5 S_{k+1} \oplus_5 S_{k+4} \oplus_5 S_{k+5} \oplus_5 S_{k+6} \oplus_5 S_{k+7},$$

where $k \geq 64$.

6.2.2 Hardware Architecture of WG-5

The top level hardware architecture of WG-5 is shown in Figure 6.2, which includes the FSM and the Datapath. The input d is used to provide the initial values for the internal states. i_valid and o_valid indicate when the inputs and outputs are valid respectively. o_WG-5 represents the output keystream of WG-5. We use the architecture presented in Chapter 4, and the optimization techniques in chapter 3, chapter 4, and chapter 5. Compared with the implementations in [7], the new one is much smaller.

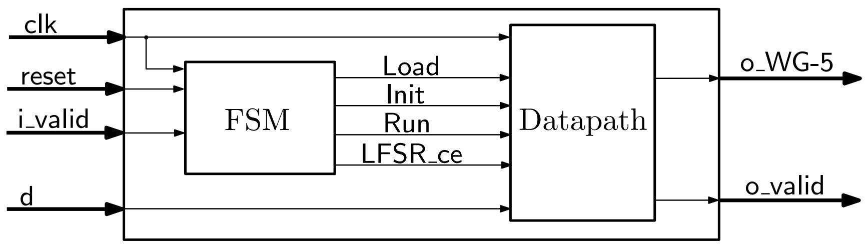


Figure 6.2: The Top Level Hardware Architecture of WG-5

FSM

Our FSM has three states, including loading, initialization, and running. When reset equals one, the WG-5 goes into the loading state, in which the initial values of the internal states are loaded into the LFSR32 by using 32 clock cycles. The initialization state begins when the loading state finished, and it will run for 64 clock cycles and then the running state starts, where the keystream is generated in each clock cycle.

Similar to Warbler in Chapter 5, we use one-hot encoding for the three states and using both binary counter and LFSR counter in the FSM. The LFSR counter is designed based on the primitive polynomial ($X^7 + X + 1$) with an initial value (1,1,1,1,1,1,1). The state transition conditions are given in Table 6.2.

Table 6.2: States Transition Conditions for FSM in WG-5

States	**Binary counter-based**	**LFSR counter-based**
Loading (100) $\rightarrow$ initialization (010)	31	79
Initialization (010) $\rightarrow$ running (001)	63	112

LFSR_ce equals one when WG-5 is in the loading and initialization states, and equals one when WG-5 is in the running state and i_valid equals one as well.

Datapath

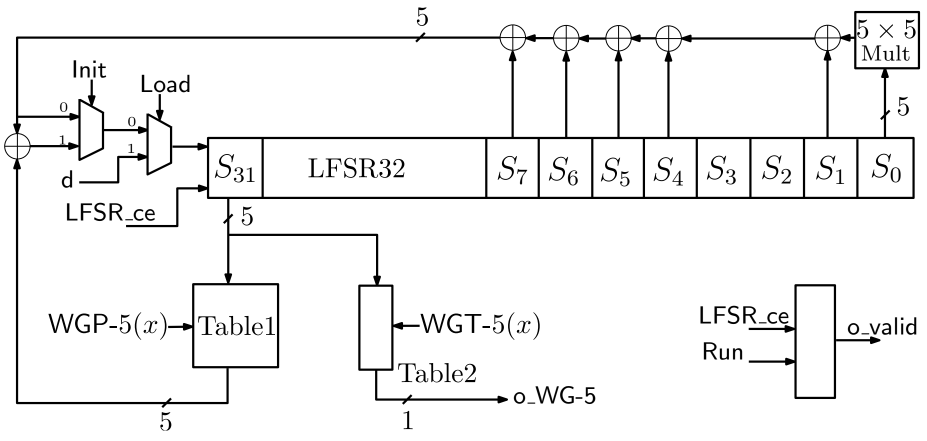


Figure 6.3: The Datapath of WG-5

The datapath of WG-5 is shown in Figure 6.3. The results analysis from WG-8 in Chapter 4 shows that the constant array based method is the best solution for the small size WG permutation and transformation modules. Hence, the constant array based method is used to implement WG-5, and all the computations in WG-5 are calculated based on polynomial basis. More specifically, we use one 5×5 constant array to store the results of WGP-5(x) for every $x \in \mathbb{F}_{2^5}$, and one 5×1 constant array to store the results of WGT-5(x) for every $x \in \mathbb{F}_{2^5}$. In addition, we use one more 5×5 constant array to store the results of the WG multiplication $\alpha^{15} S_k$ for every $S_k \in \mathbb{F}_{2^5}$.

LFSR_ce is used to control LFSR32 and is also employed with run state to control o_valid. Init and Load signals control the various input values for the LFSR32 in different states.

6.2.3 ASIC Results

We use the same design flow and metrics, and compilation techniques as Simeck and SIMON in Chapter 3 for our low area implementations of WG-5. Our ASIC results of WG-5 with compile ultra compilation and clock gating technique in CMOS 65nm and CMOS 130nm are shown in Table 6.3. Two design strategies are used to implement WG-5, which are binary counter based and LFSR counter based FSM designs respectively. The areas of the before and after place and route phase are given and the corresponding maximum operating frequency and total power consumption at 2 MHz after place and route phase are provided as well. We choose 2 MHz for the power consumption because the passive RFID tag is operated at around 2 MHz in practice.

Table 6.3: ASIC Implementation Results of WG-5 in CMOS 65nm and 130nm

WG-5	**Tech**	**Area** (GEs)		**Max Freq** (MHz)	**Throughput** @100KHz (Kbps)	**Total Power** @2MHz (μW)	**Optimality** (MHz/#GEs)	**Source**
		Before P&R	After P&R					
Binary	65nm	832	894	1234	100	9.875	1.38	here
LFSR	65nm	828	890	1250	100	9.911	1.40	here
Binary	130nm	917	997	235	100	13.61	0.23	here
LFSR	130nm	913	992	241	100	13.25	0.24	here
Binary	130nm	–	1259	–	100	–	–	[7]

As shown in Table 6.3 in both CMOS 65nm and 130nm, the area of the LFSR counter based design is smaller than that of the binary counter based design. Accordingly, the optimality (with respect to max frequency/area) is slightly larger in the LFSR counter based design, compared to the binary counter design. In addition, the total power consumption at 2 MHz is almost identical for both designs. In summary, the smallest area for WG-5 in CMOS 65nm is 890 GEs after

Place and Route phase with the power consumption of $9.911\mu W$ at 2 MHz. Similarly, in CMOS 130nm, the smallest area for WG-5 is 992 GEs with the power consumption of $13.25\mu W$ at 2 MHz.

Compared with the previous implementation, the area in CMOS 130nm are 21.2% smaller than that of result in [7].

6.3 Design and Implementations of WGLCE

As discussed in Section 6.1, WGLCE is a cryptographic engine, which integrates the Warbler pseudorandom number generator and the WG-5 cipher. In this section, we will discuss the hardware architecture and implementation results of WGLCE.

In order to make our WGLCE to generate pseudorandom numbers or to perform encryption, we need one selection signal mode to choose from these two functionalities. When mode is asserted, WGLCE works as the pseudorandom number generator. Otherwise, WGLCE works as the encryption engine.

6.3.1 Hardware Architectures

In this subsection, we propose the architecture for WGLCE based on the design rational that the pseudorandom number generator and encryption engine do not need to work simultaneously. It merges Warbler and WG-5 together by taking advantage of reusing the finite state machine for both of them, due to the fact that Warbler and WG-5 have the same architecture of the FSM but with different state transition conditions. We design a combined FSM to control the datapath of both WG-5 and Warbler for WGLCE, which reuses the state registers and the counter for both Warbler and WG-5. Inside the combined FSM, the different state transition conditions are selected by the mode signal.

During the discussion of design options, we also explored the option with reusing the registers in WG-5 and Warbler, and the option with reusing the WGT-5 module. However, reusing the registers needs the extra memory and reusing WGT-5 increases the total area due to the required extra multiplexer.

The hardware architecture with FSM reuse for WGLCE is shown in Figure 6.4. The mode signal is used to select one from Warbler and WG-5, i_valid and o_valid indicate when the input data and output sequence or keystream are valid respectively. Only when i_valid equals one, the input data can be sent into WGLCE. The input d is used to provide initial values for the internal

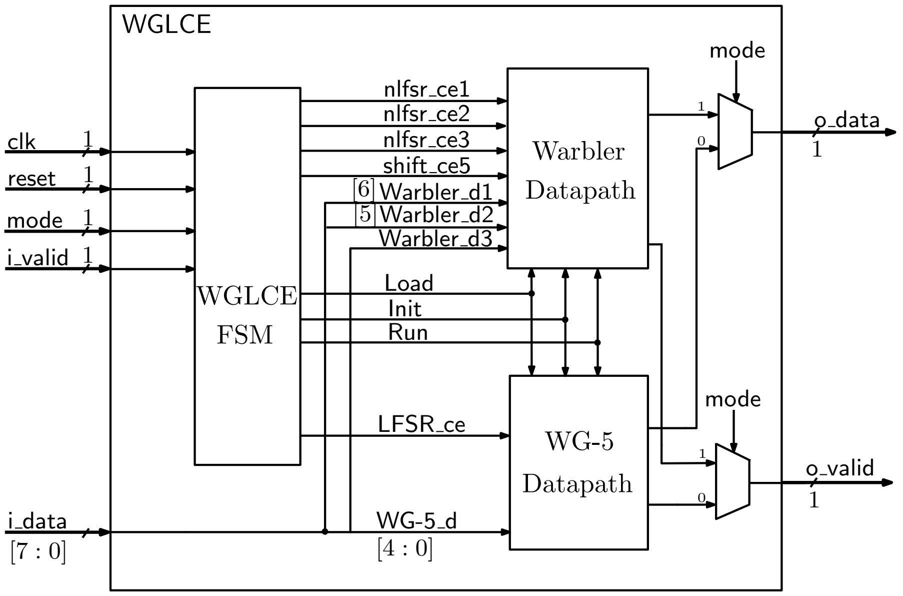


Figure 6.4: Hardware Architecture with FSM Reuse for WGLCE

states of Warbler and WG-5, with a data width eight. This is because the typical operands in the instruction set is multiples of eight. More specifically, d [6] is connected to Warbler_d1, d [5] is connected to Warbler_d2, and d [4:0] is connected to Warbler_d3 and WG-5_d. o_data is used to output the sequence of WG-5 or keystream of Warbler.

Table 6.4: States Transition Conditions for FSM in WGLCE

States	**Binary counter-based**		**LFSR counter-based**	
	Warbler	WG-5	Warbler	WG-5
Loading (100) $\rightarrow$ initialization (010)	17	31	12	79
Initialization (010) $\rightarrow$ running (001)	35	63	68	112

According to the description of Warbler and WG-5, their state transition conditions are listed in Table 6.4. If a LFSR counter is employed, the degree of the primitive polynomial should be at least 7, because the maximum binary counter number is 63. Therefore, the same primitive polynomial (X^7+X+1) as WG-5 is used for WGLCE. The LFSR counter-based state transition conditions are also listed in Table 6.4.

The hardware architecture without FSM reuse for WGLCE which directly connect Warbler and WG-5, shown in Figure 6.5, is also provided for comparison.

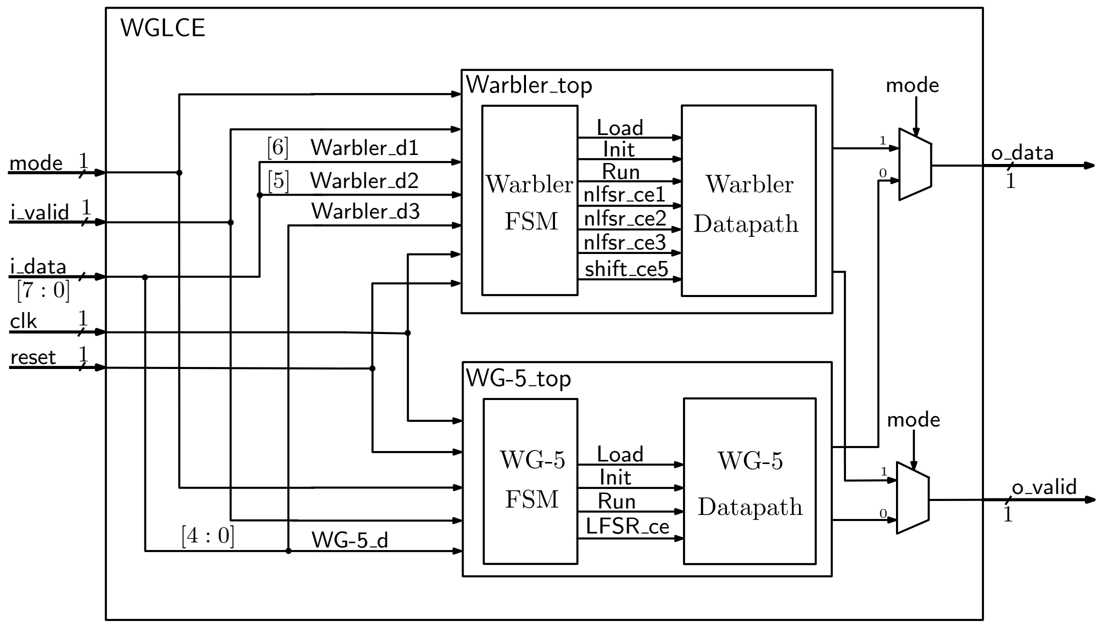


Figure 6.5: Hardware Architecture without FSM Reuse for WGLCE

6.3.2 Implementation Results and Analysis

We use the same low area implementation strategy as WG-5, and the ASIC results of WGLCE with compile ultra compilation and clock gating techinique in CMOS 65nm and 130nm are shown in Tables 6.5 and 6.6 respectively. According to the results, the architecture with FSM reuse for WGLCE is smaller than the architecture without FSM reuse in terms of area and power consumption. In addition, the area of LFSR counter-based design is smaller than that of the binary counter-based design in both CMOS 65nm and 130nm. The maximum frequency and optimality of the LFSR counter-based design are both higher than that of the binary counter-based design in CMOS 130nm, while they are slightly different in CMOS 65nm. The power consumption at 2 MHz is similar for the LFSR counter-based design and the binary counter-based design. Moreover, the power consumption of WGLCE is lower than that of the total of Warbler plus WG-5, because WGLCE only enables one functionality at each time.

In order to analyze the advantages of the architecture with FSM reuse and the effects of LFSR counter based designs, we provide the breakdown areas of WGLCE from before the place and route phase in CMOS 130nm, as shown in Table 6.7. For the binary based designs, the combined areas of Warbler_FSM and WG-5_FSM in the architecture without FSM reuse are 200 GEs. By reusing the 6-bit register for binary counter and 3-bit register for the states in the

Table 6.5: ASIC Implementation Results of WGLCE in CMOS 65nm

Architectures	WGLCE	**Area** (GEs) Before P&R	**Area** (GEs) After P&R	**Max Freq** (MHz)	**Total Power** @2MHz (μW)	**Optimality** (MHz/#GEs)
No FSM reuse	Binary	1332	1432	1193	9.784	0.83
	LFSR	1322	1422	1203	9.604	0.84
FSM resuse	Binary	1264	1359	1175	9.309	0.86
	LFSR	1260	1355	1040	9.293	0.77

Table 6.6: ASIC Implementation Results of WGLCE in CMOS 130nm

Architectures	WGLCE	**Area** (GEs) Before P&R	**Area** (GEs) After P&R	**Max Freq** (MHz)	**Total Power** @2MHz (μW)	**Optimality** (MHz/#GEs)
No FSM reuse	Binary	1439	1565	224	12.82	0.14
	LFSR	1426	1550	262	12.86	0.16
FSM resuse	Binary	1371	1490	228	12.44	0.15
	LFSR	1356	1474	237	12.47	0.16

Table 6.7: Breakdown of the Area Results for WGLCE before the Place and Route in 130nm

WGLCE Architectures	Components	Binary Counter-based (GEs)	LFSR Counter-based (GEs)
No FSM Reuse	Warbler_FSM	104.75	97.75
	WG-5_FSM	95.25	86.75
	Warbler_CORE	546.5	546.5
	WG-5_CORE	1106	1106
	Total Area*	1866	1850
	Total Area‡	1439	1426
FSM Reuse	WGLCE_FSM	127.75	114.25
	Warbler_CORE	546.5	546.5
	WG-5_CORE	1106	1106
	Total Area*	1793	1779
	Total Area‡	1371	1356

* Compile Simple, ‡ Compile ultra + clock gating.

architecture with FSM reuse, we can get the WGLCE_FSM's area equals 127.75 GEs, resulting in a 36.1% reduction. As a result, a 4.7% reduction in the total area can be achieved by using the FSM reuse architecture in the binary counter based designs. Similarly, we can achieve 38.1% area less in WGLCE_FSM than that of the combined areas of Warbler_FSM and WG-5_FSM in the LFSR counter based designs. In this case, there is a 4.9% reduction in the total area by using the architecture with FSM reuse. Hence, the FSM reuse architecture with LFSR counter based design is the best hardware implementation and is chosen as the architecture of WGLCE. Moreover, this architecture can also be generalized for other stream ciphers.

6.3.3 Interface of WGLCE

In order to practically use WGLCE, we provide a way for efficiently loading initial values to internal states. This is because the usual practice for considering the pattern of the initial values is from the perspective of the primitive only, not taking into account the environment. By doing this, the work of the external environment will be simplified. The old method for the initial value pattern of the internal states of Warbler is shown in Table 6.8. However, it is hard in terms of the size of the assembly code to load from the memory where keys and IVs are stored. In order to make it easier, we change the initial value pattern for the internal states of Warbler to a new method, as shown in Table 6.9. In the new method, we assign all the NLFSRs simultaneously from the last state with the initial values from the end, instead of assigning them from the beginning individually, in order to improve efficiency. The initial value pattern of the internal states for WG-5 conforms with this rule, thus it is not needed to change.

Table 6.8: The Old Initial Value Pattern for the Internal States of Warbler

Clock cycles	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NLFSR1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NLFSR2	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	
	60	55	50	45	40	35												
	61	56	51	46	41	36												
NLFSR3	62	57	52	47	42	37												
	63	58	53	48	43	38												
	64	59	54	49	44	39												

We assume the Keys and IVs of WG-5 and Warbler are stored in the following addresses of the memory, shown in Table 6.10. Take the WG-5 as an example. Since the size of the internal states of WG-5 is 5-bit, therefore we need to get the correct 5-bit pattern for each state and send it to WGLCE through the last 5 bits of i_data. However, the Keys and IVs are stored in a 8-bit

Table 6.9: The New Initial Value Pattern for the Internal States of Warbler

Clock cycles	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NLFSR1	58	51	44	37	30	23	21	19	17	15	13	11	9	7	5	3	1	0
NLFSR2	59	52	45	38	31	24	22	20	18	16	14	12	10	8	6	4	2	
	60	53	46	39	32	25												
	61	54	47	40	33	26												
NLFSR3	62	55	48	41	34	27												
	63	56	49	42	35	28												
	64	57	50	43	36	29												

Table 6.10: The Location of Keys and IVs

(a) The Location of Keys and IVs for WG-5

Keys	IVs	data (bit positions)							
10	20	7	6	5	4	3	2	1	0
11	21	15	14	13	12	11	10	9	8
12	22	23	22	21	20	19	18	17	16
13	23	31	30	29	28	27	26	25	24
14	24	39	38	37	36	35	34	33	32
15	25	47	46	45	44	43	42	41	40
16	26	55	54	53	52	51	50	49	48
17	27	63	62	61	60	59	58	57	56
18	28	71	70	69	68	67	66	65	64
19	29	79	78	77	76	75	74	73	72

(b) The Location of Keys and IVs for Warbler

Keys	IVs	data (bit positions)							
30	39	7	6	5	4	3	2	1	0
31	40	15	14	13	12	11	10	9	8
32	41	23	22	21	20	19	18	17	16
33	42	31	30	29	28	27	26	25	24
34	43	39	38	37	36	35	34	33	32
35	44	47	46	45	44	43	42	41	40
36	45	55	54	53	52	51	50	49	48
37	46	63	62	61	60	59	58	57	56
38	47								64

pattern in the memory. We propose an efficient algorithm for loading them by using the loop statement in order to minimize the code size, and reduce the number of instructions to load data from memory. We use the first 16 states of WG-5 to illustrate the algorithm in pseudo-assembly code as it is loaded from the Keys, which is shown in Algorithm 1.

R1 to R10 are the registers from the processor and they are used to obtain the correct 5-bit pattern values for internal states. We first use R2 and R3 to store the lowest and highest bit positions of the 5-bit pattern, and then transform them to byte positions (R4, R6) in the 8-bit pattern as well as the bit positions (R5, R7) in each byte. By using the byte positions, we can load the data in the memory to the register. We then construct a new 8-bit data with the last 5 effective bits are the initial values under two different cases, depending on whether the byte positions are equal.

Algorithm 1: The Algorithm to Load the Initial Values for the First 16 States

```
   Mov R1, 0                   R1 stores the states index of WG-5 from 0 to 15
S: MUL R2, R1, 5               R2 stores the lowest bit position of 5-bit pattern
   ADD R3, R2, 4               R3 stores the highest bit position of 5-bit pattern
   DIV R4, R2, 8               R4 stores the lowest byte position of 8-bit pattern
   MOD R5, R2, 8               R5 stores the lowest bit position of R4
   DIV R6, R3, 8               R6 stores the highest byte position of 8-bit pattern
   MOD R7, R3, 8               R7 stores the highest bit position of R6
   Bneq R4, R6 B               If R4 equals R6, go to branch A; otherwise go to branch B
A: ADD R4, R4, 10              Refresh R4 with the lowest byte address in memory
   Load R8, R4                 Read the 8-bit data with address R4 to R8
   SRL R8, R8, R5              Shift R8 to right by R5 bit
   AND R8, R8, b``00011111"    Get the effective 5-bit pattern
   Send R8, i_data             Send R8 to the i_data input
   JMP C                       Go to branch C
B: ADD R4, R4, 10              Refresh R4 with the lowest byte address in memory
   Load R8, R4                 Read the 8-bit data with address R4 to R8
   SRL R8, R8, R5              Shift R8 to right by R5 bit
   Mov R9, b``11111111"        Generate a mask for the effective bits in R8 after shifting
   SRL R9, R9, R5
   AND R8, R8, R9
   ADD R6, R6, 10              Refresh R6 with the highest byte address in memory
   Load R10, R6                Read the 8-bit data with address R6 to R10
   Sub R5, 8, R5
   SLL R10, R10, R5            Shift R10 to left by new R5 bit
   Mov R9, b``11111111"        Generate a mask for the effective bits in R10 after shifting
   SLL R9, R9, R5
   SLL R9, R9, 3
   SRL R9, R9, 3
   AND R10, R10, R9
   OR R8, R8, R10              Construct a new R8 with the last 5 effective bits
   Send R8, i_data             Send R8 to the i_data input
C: ADD R1, 1                   Increment R1
   SUB R2, R1, 16
   BLT R2, S                   Go back to branch S
```

Similar algorithm can be used for loading the initial values of Warbler by using the new method pattern.

6.4 Comparisons

The areas after the Place and Route phase of our WGLCE in CMOS 130nm and 65nm ASICs are 1474 GEs and 1355 GEs respectively. They are both smaller than the well accepted area limit (2000 GEs) in the passive RFID tags. We compare our WGLCE with the cryptographic engines in the previous papers, such as AES plus Grain [33], and Present-80 plus LAMED [107], as shown in Table 6.11. Although there is no actual hardware implementation of LAMED, the estimate area of LAMED is larger than that of Warbler based on the analysis in [72]. The smallest area of AES reported by Moradi *et al.* [82] is 2400 GEs in CMOS 180nm ASIC. Considering all the above factors together, the area of WGLCE is smaller than that of the other two designs.

Table 6.11: Comparisons with the Existing Cryptographic Engines

Design	Crypto Engines	Area (GEs)	Power (@2MHz μW)	Tech (nm)	Common Hardware Reuse
Ertl *et al.* [33]	AES, Grain	2770 2450	– –	130 130	No
Todd *et al.* [107]	Present-80, LAMED	1900 –	– –	65 –	No
This work	WGLCE	1474 1355	12.47 9.29	130 65	Yes

6.5 Summary

In this chapter, we designed a lightweight cryptographic engine (WGLCE) for the passive R-FID systems. WGLCE is a fusion of the Warbler pseudorandom number generator and the lightweight stream cipher WG-5, which can be easily integrated into RFID systems. Firstly, we investigated the rationales and design choices for WGLCE and then explored the hardware implementation of WG-5. Later on, we discussed the design, hardware architectures, and implementations of our WGLCE. Finally, we compared our WGLCE results with other cryptographic engines. Overall, our WGLCE can satisfy the area requirement for the security extension in the passive RFID tags, and it is a promising candidate for this kind of applications.

Chapter 7

Conclusions and Future Work

This chapter concludes the thesis and provides future work. Section 7.1 presents a summary of contributions and concluding remarks, and the potential future work directions are discussed in Section 7.2.

7.1 Conclusions

In this thesis, we concentrated on the efficient hardware implementations and optimizations of lightweight cryptography, including the block cipher Simeck, stream cipher WG-8, pseudo-random number generator Warbler, and cryptographic engine WGLCE, in order to meet the constraints in resource constrained applications. We have shown that they can meet the area, power consumption, and throughput requirements in passive RFID tags and they are promising candidates for resource constrained applications.

Motivated by the designs of SIMON and SPECK, we first proposed Simeck, a new family of lightweight block ciphers with Feistel structure. Simeck takes advantage of the good components and design ideas of SIMON and SPECK, and it has three instances with different block and key sizes: Simeck32/64, Simeck48/96, and Simeck64/128. Simeck is designed to have a smaller area than that of SIMON with the following considerations: the reduced shift numbers in the round function, the simplified key schedule, and the simplified LFSR to generate the key constant. We provided an extensive exploration for different hardware architectures in order to make a balance between area, throughput, and power consumption for SIMON and Simeck in both CMOS 130nm and CMOS 65nm ASICs. We verified that Simeck is indeed smaller than that of SIMON in terms of area and power consumption, and a thorough analysis for the area reductions for parallel and fully serialized architectures is given. Moreover, our SIMON's area is smaller than the results of SIMON given in the original paper. In addition, the security analysis showed that even though the round function of Simeck is quite simple, this round function is iterated a sufficient number of time to provide an adequate security against known attacks.

For WG-8, we explored four different constructions for the WG transformation module. The first architecture directly employs an 8×8 constant array over $\mathbb{F}_{2^8}$, the second one is based on the tower construction $\mathbb{F}_{(2^4)^2}$ together with small 4×4 constant arrays for arithmetic in $\mathbb{F}_{2^4}$, due to the existence of primitive element. The third architecture is slightly different from the second one, due to the usage of a type-I ONB for efficient computations in $\mathbb{F}_{2^4}$. Finally, the fourth architecture takes advantage of the tower construction $\mathbb{F}_{((2^2)^2)^2}$ coupled with a nice property for computing the trace of product of two finite field elements under this certain tower construction. We also proposed a novel hybrid design with the parallel width from one to eleven for each proposed architecture. We gave the results on low-cost FPGA and CMOS 65nm and CMOS 130nm A-SICs in terms of area, clock speed, throughput and power consumption The experimental results showed that the lightweight stream cipher WG-8 with the direct constant array based hardware architecture is optimal in terms of throughput, area, and power consumption, when compared to the tower field arithmetic based approaches. The main reason is due to the small field size as well as the relatively complicated architecture of WG-8 permutation/transformation module. Although the tower field based approaches for WG-8 are not efficient, the proposed architecture and extensive experimental results still provide valuable guidance for efficient hardware implementation of medium or large instances of the WG stream cipher family. Moreover, with a little additional hardware resources, the parallel implementations can achieve a high throughput without decreasing the clock speed two much.

Later on, we presented first detailed and smallest hardware implementations and optimizations of Warbler PRNG in CMOS 65nm and CMOS 130nm ASICs. We proposed an architecture for hardware implementations of Warbler with thorough analysis. We improved the throughput from $1/5$ bpc to 1 bpc by increasing 46% and 36% of the area respectively in CMOS 65nm and CMOS 130nm. Moreover, the sequential logic ratios for all our designs are bigger than 65% for throughput of $1/5$ bpc and are around 50% for throughput of 1 bpc. We determined that the LFSR counter-based design is better than the binary counter-based design in terms of area and total power consumption. The area of the WG-5 transformation table depends upon the selected decimation value, giving us some suggestions for future ciphers and pseudorandom number generator designs using WG-5 transformations. When compared with other lightweight primitives, the areas of our Warbler implementations are smaller than that of other PRNGs and are in fact smaller than most of lightweight primitives.

Finally, we proposed a lightweight cryptographic engine WGLCE, which can be easily integrated into passive RFID systems. WGLCE merges Warbler PRNG and WG-5 stream cipher, and it takes advantages of reusing the FSM. WGLCE has two functionalities: data encryption/decryption and random numbers generation. We provided the design rationales, architectures, and implementation results in CMOS 65nm and CMOS 130nm ASICs. Moreover, an interface with outside environment for WGLCE was provided as well. When compared with other cryptographic engines, the area of WGLCE is smaller than that of them. Overall, the results showed that it can satisfy the requirement for the security extension in passive EPC RFID systems.

7.2 Future Work

Some future interesting research directions can be conducted from extension of the current results, which are shown as follows:

- **Exploring the hardware implementations and optimizations of other instances in SIMON and SPECK families.** We provided a wide range of architectures and optimizations for the implementations of Simeck and SIMON with tradeoffs of area, power consumption, and throughput in Chapter 3. However, we only finished three instances of SIMON family in order to compare with Simeck family. Therefore, we can use the proposed architectures and techniques to implement the other seven instances of SIMON family (SIMON48/72, SIMON64/96, SIMON96/96, SIMON96/144, SIMON128/128, SIMON128/192, SIMON128/256) and the ten instances of SPECK family (SPECK32/64, SPECK48/72, SPECK48/96, SPECK64/96, SPECK64/128, SPECK96/96, SPECK96/144, SPECK128/128, SPECK128/192, SPECK128/256) which can not only improve the existing area results, but also provide detailed explorations. Moreover, we can try to add new instances to Simeck family by investigating the simpler key schedule designs than that of SIMON family with a thorough area and cryptographic analysis.

- **Hardware evaluations of other WG ciphers and cryptographic engines.** We evaluated the hardware implementations and optimizations of WG-8 using constant array method and three tower field approaches in Chapter 4 and proposed a cryptographic engine WGLCE in Chapter 6. Similar optimization techniques, such as different tower constructions, parallelism, component reuse, different architectures, and so on, can be used for other WG ciphers, such as WG-10, WG-11, WG-14, etc. These techniques can provide tradeoffs of area, power consumption and throughput for the primitives. In addition, WGLCE can be generalized for other primitives which can provide a cryptographic engine with different security levels.

- **Design and implement a secure mutual authentication protocol for the passive RFID readers and tags.** The mutual authentication protocol is very important for the passive RFID systems. It provides the authentication of both the reader and the tag, leading to resist to the reader or tag impersonation attacks. We will adopt the privacy preserving authentication protocol, specified in EPC standard with our cryptographic engine to design the protocol. Then, we can perform the security analysis. Finally, we can evaluate the hardware optimizations of this mutual authentication protocol in order to make it feasible for the current passive RFID systems.

- **A pseudorandom number generator with 128-bit key.** Our current Warbler PRNG has pretty small area but with 65-bit key. In order to provide more flexibility in security for more applications, it will be good for us to consider a new PRNG with 128-bit key and with a similar architecture as Warbler.

- **Side channel attack analysis.** The side channel attack is based on exploiting the information measured externally from the implementation of the primitive to reveal its secret key. There are three common types of side channel attacks: timing attack, electromagnetic analysis attack and power analysis attack. Therefore, evaluating the side channel analysis of the primitives is one of our future work for secure use in practice.

References

[1] Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 Generation FPGAs. *Xilinx Inc., available at http://www.xilinx.com/support/documentation/ application_notes/xapp465.pdf*, May 2005.

[2] EPC Radio Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHZ - 960 MHz Version 1.2. *EPCglobal Inc., available at http://www.gs1.org/sites/default/files/docs/epc/uhfc1g2_1_2_0-standard-20080511.pdf*, Oct 2008.

[3] EPC Radio Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHZ - 960 MHz Version 2. *EPCglobal Inc., available at http://www.gs1.org/sites/default/files/docs/epc/uhfc1g2_2_0_0_standard-20131101.pdf*, Nov 2013.

[4] What is the Future of RFID Technology? *available at http://www.lowrysolutions.com/blog/future-rfid-technology/*, January 2015.

[5] Lightweight Cryptography Standardization Investigation. *NIST, available at http://www.nist.gov/itl/csd/ct/lwc-project.cfm*, 2015, 2016.

[6] NIST FIPS 197. Specification for the Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication*. http://csrc.nist.gov/archive/aes/.

[7] Mark D. Aagaard, Guang Gong, and Rajesh K. Mota. Hardware Implementations of the WG-5 Cipher for Passive RFID Tags. In *6th IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 24–29, June 2013.

[8] Javad Alizadeh, Hoda AlKhzaimi, Mohammad Reza Aref, Nasour Bagheri, Praveen Gauravaram, Abhishek Kumar, Martin M. Lauridsen, and Somitra Kumar Sanadhya. Crypt-

analysis of SIMON Variants with Connections. In *Radio Frequency Identification: Security and Privacy Issues - 10th International Workshop, RFIDSec 2014, Oxford, UK, July 21-23, 2014, Revised Selected Papers*, pages 90–107, 2014.

[9] Frederik Armknecht, Matthias Hamann, and Vasily Mikhalev. Lightweight Authentication Protocols on Ultra-Constrained RFIDs – Myths and Facts. In *Radio Frequency Identification: Security and Privacy Issues*, pages 1–18. Springer, 2014.

[10] Nasour Bagheri. Linear Cryptanalysis of Reduced-Round SIMECK Variants. In *Progress in Cryptology–INDOCRYPT 2015*, pages 140–152. Springer, 2015.

[11] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/.

[12] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: Block Ciphers for the Internet of Things. In *NIST Lightweight Cryptography Workshop*, 2015.

[13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Lightweight Block Ciphers. In *Proceedings of the 52nd Annual Design Automation Conference*, page 175. ACM, 2015.

[14] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and its Low-Latency Variant MANTIS. Cryptology ePrint Archive, Report 2016/660, 2016. https://eprint.iacr.org/2016/660.pdf.

[15] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK. In *FSE 2014*, LNCS. Springer, 2014.

[16] Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varıcı, and Ingrid Verbauwhede. SPONGENT: A Lightweight Hash Function. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 312–325. Springer, 2011.

[17] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An Ultra-lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 450–466. Springer, 2007.

[18] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, et al. PRINCE–A Low-Latency Block Cipher for Pervasive Computing Applications. In *Advances in Cryptology*, pages 208–225. Springer, 2012.

[19] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 179–199, 2014.

[20] CAESAR. CAESAR: Competition for Authenticated Encryption. https://competitions.cr.yp.to/caesar.html, 2012.

[21] David Canright. A Very Compact S-box for AES. In *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 441–455. Springer, 2005.

[22] Qi Chai, Guang Gong, and Daniel Engels. How to Develop Clairaudience-active Eavesdropping in Passive RFID Systems. In *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2012.

[23] Lidong Chen and Guang Gong. *Communications System Security*. CRC Press, 2012.

[24] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In *Information Security*, pages 171–186. Springer, 2006.

[25] Christophe De Cannière and Preneel Bart. Trivium Specifications. *available at http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf*, 2005.

[26] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN: A Family of Small and Efficient Hardware-oriented Block Ciphers. In *The 11th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 272–288. Springer, September 2009.

[27] Christophe De Cannière and Bart Preneel. Trivium Specifications, 2005. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030.

[28] Jean-Pierre Deschamps. *Hardware Implementation of Finite-field Arithmetic*. McGraw-Hill, Inc., 2009.

[29] Thomas Eisenbarth and Sandeep Kumar. A Survey of Lightweight-cryptography Implementations. *IEEE Design & Test of Computers*, 24(6):522–533, 2007.

[30] Hayssam El-Razouk, Arash Reyhani-Masoleh, and Guang Gong. New Implementations of the WG Stream Cipher. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(9):1865–1878, 2014.

[31] Hayssam El-Razouk, Arash Reyhani-Masoleh, and Guang Gong. New Hardware Implementationsof WG-29 and WG-16 Stream Ciphers Using Polynomial Basis. *IEEE Transactions on Computers*, 64(7):2020–2035, 2015.

[32] Daniel Engels, Xinxin Fan, Guang Gong, Honggang Hu, and Eric M. Smith. Ultra-lightweight Cryptography for Low-cost RFID Tags: Hummingbird Algorithm and Protocol. *Centre for Applied Cryptographic Research (CACR) Technical Reports*, 29, 2009.

[33] Johann Ertl, Thomas Plos, Martin Feldhofer, Norbert Felber, and Luca Henzen. A Security-enhanced UHF RFID Tag Chip. In *Euromicro Conference on Digital System Design (DSD)*, pages 705–712. IEEE, 2013.

[34] eSTREAM. The ECYPT Stream Cipher Project. http://www.ecrypt.eu.org/stream/, 2008.

[35] Xinxin Fan, Kalikinkar Mandal, and Guang Gong. WG-8: A Lightweight Stream Cipher for Resource-constrained Smart Devices. In *9th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, 2013.

[36] Xinxin Fan, Teng Wu, and Guang Gong. An Efficient Stream Cipher WG-16 and Its Application for Securing 4G-LTE Networks. In *Applied Mechanics and Materials*, volume 490, pages 1436–1450. Trans Tech Publ, 2014.

[37] Xinxin Fan, Nusa Zidaric, Mark Aagaard, and Guang Gong. Efficient Hardware Implementation of the Stream Cipher WG-16 with Composite Field Arithmetic. In *Proceedings of the 2013 ACM CCS Workshop on Trustworthy Embedded Devices (TrustED 2013)*, November 2013.

[38] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithm. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 357–370. Springer, 2004.

[39] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES Implementation on A Grain of Sand. *IEE Proceedings-Information Security*, 152(1):13–20, 2005.

[40] Klaus Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. Wiley, 2003.

[41] Shuhong Gao and Hendrik W Lenstra Jr. Optimal Normal Bases. *Designs, Codes and Cryptography*, 2(4):315–323, 1992.

[42] Simson L Garfinkel, Ari Juels, and Ravi Pappu. RFID Privacy: An Overview of Problems and Proposed Solutions. *IEEE Security & Privacy*, (3):34–43, 2005.

[43] Bill Glover and Himanshu Bhatt. *RFID Essentials*. O'Reilly Media, Inc., 2006.

[44] Solomon W. Golomb and Guang Gong. *Signal Design for Good Correlation: for Wireless Communication, Cryptography, and Radar*. Cambridge University Press, 2005.

[45] Solomon W. Golomb, Lloyd R. Welch, Richard M. Goldstein, and Alfred W. Hales. *Shift Register Sequences*, volume 78. Aegean Park Press Laguna Hills, CA, 1982.

[46] Guang Gong and Amr M. Youssef. Cryptographic Properties of the Welch-Gong Transformation Sequence Generators. *IEEE Transactions on Information Theory*, 48(11):2837–2846, 2002.

[47] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A New Family of Lightweight Block Ciphers. In *Proceedings of the 7th international conference on RFID Security and Privacy*, pages 1–18. Springer-Verlag, 2011.

[48] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In *Advances in Cryptology–CRYPTO 2011*, pages 222–239. Springer, 2011.

[49] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED Block Cipher. In *The 13th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2011*, pages 326–341. Springer, September 2011.

[50] Panu Hamalainen, Timo Alho, Marko Hannikainen, and Timo D. Hamalainen. Design and Implementation of Low-area and Low-power AES Encryption Hardware Core. In *The 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, 2006 (DSD 2006).*, pages 577–583. IEEE, 2006.

[51] Gerhard Hancke. Eavesdropping Attacks on High-frequency RFID Tokens. In *4th Workshop on RFID Security (RFIDSec)*, pages 100–113, 2008.

[52] Anwar Hasan. Lecture Notes. *ECE720 - Selected Topics in Cryptographic Computations*, Fall 2013.

[53] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. In *IEEE International Symposium on Information Theory (ISIT 2006)*, 2006.

[54] Martin Hell, Thomas Johansson, and Willi Meier. Grain - A Stream Cipher for Constrained Environments. In *2005/010, ECRYPT (European Network of Excellence for Cryptology)*, 2005.

[55] Martin Hell, Thomas Johansson, and Willi Meier. Grain: A Stream Cipher for Constrained Environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, May 2007.

[56] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bon-Seok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, et al. Hight: A New Block Cipher Suitable for Low-resource Device. In *The 8th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2006*, pages 46–59. Springer, October 2006.

[57] David Hwang, Mark Chaney, Shashi Karanam, Nick Ton, and Kris Gaj. Comparison of FPGA-targeted Hardware Implementations of eSTREAM Stream Cipher Candidates. *The State of the Art of Stream Ciphers*, pages 151–162, 2008.

[58] Peter Jamieson, Wayne Luk, Steve JE Wilton, and George A Constantinides. An Energy and Power Consumption Analysis of FPGA Routing Architectures. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 324–327. IEEE, 2009.

[59] Ari Juels. RFID Security and Privacy: A Research Survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, 2006.

[60] Ari Juels and Stephen A. Weis. Authenticating Pervasive Devices with Human Protocols. In *Advances in Cryptology – CRYPTO 2005*, pages 293–308. Springer, 2005.

[61] Chang Han Kim, Yongtae Kim, Sung Yeon Ji, and IlWhan Park. A New Parallel Multiplier for Type II Optimal Normal Basis. In *Computational Intelligence and Security*, pages 460–469. Springer, 2007.

[62] Micoslav Knežević. Lightweight Cryptography: from Smallest to Fastest. Technical report, NIST Lightweight Cryptography Workshop, July, 2015.

[63] Miroslav Knežević, Ventzislav Nikov, and Peter Rombouts. Low-Latency Encryption–Is Lightweight= Light+ Wait? In *Cryptographic Hardware and Embedded Systems–CHES 2012*, pages 426–446. Springer, 2012.

[64] Lars Knudsen, Gregor Leander, Axel Poschmann, and Matthew J.B. Robshaw. PRINTcipher: A Block Cipher for IC-printing. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 16–32. Springer, 2010.

[65] Stefan Kölbl and Arnab Roy. A Brief Comparison of Simon and Simeck. Technical report, Cryptology ePrint Archive, Report 2015/706, 2015.

[66] Jia Hao Kong, Li-Minn Ang, and Kah Phooi Seng. A Comprehensive Survey of Modern Symmetric Cryptographic Solutions for Resource Constrained Environments. *Journal of Network and Computer Applications*, 49:15–50, 2015.

[67] Chiu Hong Lam, Mark Aagaard, and Guang Gong. Hardware Implementations of Multi-output Welch-Gong Ciphers. *Centre for Applied Cryptographic Research Technical Reports, CACR 2011-01.*

[68] Lang Li, Botao Liu, and Hui Wang. QTL: A New Ultra-lightweight Block Cipher. *Microprocessors and Microsystems*, 2016.

[69] Rudolf Lidl. *Introduction to Finite Fields and Their Applications*. Cambridge university press, 1994.

[70] Lingyue Qin and Huaifeng Chen. Linear Hull Attack on Round-Reduced Simeck with Dynamic Key-guessing Techniques. Technical report, Cryptology ePrint Archive, Report 2016/066, 2016.

[71] Yiyuan Luo, Qi Chai, Guang Gong, and Xuejia Lai. A Lightweight Stream Cipher WG-7 for RFID Encryption and Authentication. In *IEEE Global Telecommunications Conference (GLOBECOM 2010)*, pages 1–6. IEEE, 2010.

[72] Kalikinkar Mandal, Xinxin Fan, and Guang Gong. Warbler: A Lightweight Pseudorandom Number Generator for EPC C1 Gen2 Tags. In *Radio Frequency Identification System Security: RFIDsec'12 Asia Workshop Proceedings*, page 73. IOS Press, 2013.

[73] Kalikinkar Mandal, Xinxin Fan, and Guang Gong. Design and Implementation of Warbler Family of Lightweight Pseudorandom Number Generators for Smart Devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):1, 2016.

[74] Kalikinkar Mandal, Guang Gong, Xinxin Fan, and Mark Aagaard. Optimal Parameters for the WG Stream Cipher Family. *Cryptography and Communications*, pages 1–19, 2013.

[75] Charalampos Manifavas, George Hatzivasilis, Konstantinos Fysarakis, and Konstantinos Rantos. Lightweight Cryptography for Embedded Systems–A Comparative Analysis. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 333–349. Springer, 2014.

[76] Honorio Martin, Enrique San Millán, Pedro Peris-Lopez, and Juan E. Tapiador. Efficient ASIC Implementation and Analysis of Two EPC C1 G2 RFID Authentication Protocols. *Sensors Journal, IEEE*, 13(10):3537–3547, 2013.

[77] Joan Melià Seguí. Lightweight PRNG for Low-cost Passive RFID Security Improvement. In *Ph.D. Thesis*. Universitat Oberta de Catalunya, 2011.

[78] Joan Melià-Seguí, Joaquin Garcia-Alfaro, and Jordi Herrera-Joancomarti. Analysis and Improvement of A Pseudorandom Number Generator for EPC Gen2 Tags. In *Financial Cryptography and Data Security*, pages 34–46. Springer, 2010.

[79] Joan Melià-Seguí, Joaquin Garcia-Alfaro, and Jordi Herrera-Joancomartí. J3Gen: A PRNG for Low-cost Passive RFID. *Sensors*, 13(3):3816–3830, 2013.

[80] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC press, 1996.

[81] Aikaterini Mitrokotsa, Melanie R. Rieback, and Andrew S. Tanenbaum. Classifying RFID Attacks and Defenses. *Information Systems Frontiers*, 12(5):491–505, 2010.

[82] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and A Threshold Implementation of AES. In *Advances in Cryptology – EUROCRYPT 2011*, pages 69–88. Springer, 2011.

[83] Venu Nalla, Rajeev Anand Sahu, and Vishal Saraswat. Differential Fault Attack on SIMECK. In *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, pages 45–48. ACM, 2016.

[84] Yassir Nawaz and Guang Gong. The WG Stream Cipher. *eSTREAM PHASE 2 Achive, available at http://www.ecrypt.eu.org/stream/p2ciphers/wg/wg_p2.pdf*, 2005.

[85] Yassir Nawaz and Guang Gong. WG: A Family of Stream Ciphers with Designed Randomness Properties. *Information Sciences*, 178(7):1903–1916, 2008.

[86] Roger M. Needham and David J. Wheeler. TEA Extensions, October 1997. Technical Report, University of Cambridge.

[87] Mohammad Ali Orumiehchiha, Josef Pieprzyk, and Ron Steinfeld. Cryptanalysis of WG-7: A Lightweight Stream Cipher. *Cryptography and Communications*, 4(3-4):277–285, 2012.

[88] Christof Paar. *Efficient VLSI Architectures for Bit Parallel Computation in Galios Fields*. VDI-Verlag, 1994.

[89] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Science & Business Media, 2009.

[90] R.K. Pateriya and Sangeeta Sharma. The Evolution of RFID Security and Privacy: A Research Survey. In *International Conference on Communication Systems and Network Technologies (CSNT), 2011*, pages 115–119. IEEE, 2011.

[91] Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M. Estevez-Tapiador, and Arturo Ribagorda. LAMED–A PRNG for EPC Class-1 Generation-2 RFID Specification. *Computer Standards & Interfaces*, 31(1):88–97, 2009.

[92] Thomas Plos, Christoph Dobraunig, Markus Hofinger, Alexander Oprisnik, Christoph Wiesmeier, and Johannes Wiesmeier. Compact Hardware Implementations of the Block Ciphers mCrypton, NOEKEON, and SEA. In *Progress in Cryptology – INDOCRYPT 2012*, pages 358–377. Springer, 2012.

[93] Axel Poschmann. Lightweight Cryptography: Cryptographic Engineering for A Pervasive World. In *Ph.D. Thesis*. Department of Electical Engineering and Information Sciences, Ruhr-University Bochum, Germany, 2009.

[94] Axel Poschmann, Gregor Leander, Kai Schramm, and Christof Paar. New Light-weight Crypto Algorithms for RFID. In *IEEE International Symposium on Circuits and Systems, 2007 (ISCAS 2007)*, pages 1843–1846. IEEE, 2007.

[95] Damith C. Ranasinghe and Peter H. Cole. Confronting Security and Privacy Threats in Modern RFID Systems. In *Fortieth Asilomar Conference on Signals, Systems and Computers, 2006 (ACSSC'06)*, pages 2058–2064. IEEE, 2006.

[96] Vincent Rijmen. Efficient Implementation of the Rijndael S-box. *Katholieke Universiteit Leuven, Department of ESAT in Belgium*, 2000.

[97] Francisco Rodríguez-Henríquez, Nazar Abbas Saqib, Arturo Díaz Pérez, and Cetin Kaya Koc. *Cryptographic Algorithms on Reconfigurable Hardware*. Springer Science & Business Media, 2007.

[98] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents. In *Smart Card Research and Advanced Applications*, pages 89–103. Springer, 2008.

[99] Sondre Rønjom. Improving Algebraic Attacks on Stream Ciphers based on Linear Feedback Shift Register over $\mathbb{F}_{2^k}$. *Designs, Codes and Cryptography*, pages 1–15.

[100] Markku-Juhani O Saarinen and Daniel W. Engels. A Do-it-all-cipher for RFID: Design Requirements. *IACR Cryptology ePrint Archive*, 2012:317, 2012.

[101] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A Compact Rijndael Hardware Architecture with S-box Optimization. In *Advances in Cryptology ASIACRYPT 2001*, pages 239–254. Springer, 2001.

[102] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An Ultra-lightweight Block Cipher. In *The 13th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2011*, pages 342–357. Springer, September 2011.

[103] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit Block Cipher CLEFIA. In *Fast Software Encryption*, pages 181–195. Springer, 2007.

[104] L. SIMSON, Ari Juels, and Ravi Pappu. RFID Privacy: An Overview of Problems and Proposed Solutions. 2005.

[105] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In *The 7th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications-CARDIS 2006*, pages 222–236. Springer, April 2006.

[106] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. Twine: A Lightweight Versatile Block Cipher. In *ECRYPT Workshop on Lightweight Cryptography*, pages 146–169, 2011.

[107] Michael Todd, Wayne Burleson, and Russell Tessier. The Design and Assessment of A Secure Passive RFID Sensor System. In *New Circuits and Systems Conference (NEWCAS), 2011 IEEE 9th International*, pages 494–497. IEEE, 2011.

[108] Rei Ueno, Naofumi Homma, Yukihiro Sugawara, Yasuyuki Nogami, and Takafumi Aoki. Highly Efficient GF (2ˆ8) Inversion Circuit Based on Redundant GF Arithmetic and Its Application to AES Design. In *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 63–80. Springer, 2015.

[109] Joachim Von Zur Gathen, Amin Shokrollahi, and Jamshid Shokrollahi. Efficient Multiplication using Type II Optimal Normal Bases. In *Arithmetic of Finite Fields*, pages 55–68. Springer, 2007.

[110] Roy Want. *RFID Explained: A Primer on Radio Frequency Identification Technologies*, volume 1. Morgan & Claypool Publishers, 2006.

[111] Stephen A. Weis, Sanjay E. Sarma, Ronald L. Rivest, and Daniel W. Engels. Security and Privacy Aspects of Low-cost Radio Frequency Identification Systems. In *Security in pervasive computing*, pages 201–212. Springer, 2004.

[112] David J. Wheeler and Roger M. Needham. TEA: A Tiny Encryption Algorithm. In *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, pages 363–366, 1994.

[113] Wenling Wu and Lei Zhang. LBlock: A Lightweight Block Cipher. In *Proceedings of Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011.*, pages 327–344. Springer, 2011.

[114] Gangqiang Yang, Mark D. Aagaard, and Guang Gong. Efficient Hardware Implementations of the Warbler Pseudorandom Number Generator. *NIST Lightweight Cryptography Workshop*, 2015.

[115] Gangqiang Yang, Xinxin Fan, Mark D. Aagaard, and Guang Gong. Design Space Exploration of the Lightweight Stream Cipher WG-8 for FPGAs and ASICs. In *Proceedings of the Workshop on Embedded Systems Security*, page 8. ACM, 2013.

[116] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. The Simeck Family of Lightweight Block Ciphers. In *Cryptographic Hardware and Embedded Systems–CHES 2015*, pages 307–329. Springer, 2015.

[117] Huihui Yap, Khoongming Khoo, Axel Poschmann, and Matt Henricksen. EPCBC: A Block Cipher Suitable for Electronic Product Code Encryption. In *Proceedings of the 10th international conference on Cryptology and Network Security*, pages 76–97. Springer-Verlag, 2011.

[118] Daniel J. Yeager, Alanson P. Sample, Joshua R. Smith, and Joshua R. Smith. WISP: A Passively Powered UHF RFID Tag with Sensing and Computation. *RFID Handbook: Applications, Technology, Security, and Privacy*, pages 261–278, 2008.

[119] Yu Yu, Yuqing Yang, Na Yan, and Hao Min. A Novel Design of Secure RFID Tag Baseband. *RFID Convocation, Brussels, Belgium*, 2007.