Flexible and Lightweight Cryptographic Engines for Constrained Systems

Ege Gulcan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Patrick Schaumont, Chair
Leyla Nazhandali
Michael S. Hsiao

May 4, 2015
Blacksburg, Virginia

Keywords: Lightweight Cryptography, Block Ciphers, Flexible Architectures, SIMON, FPGA

Flexible and Lightweight Cryptographic Engines for Constrained Systems

Ege Gulcan

(ABSTRACT)

There is a significant effort in building lightweight cryptographic operations, yet the proposed solutions are typically single purpose modules that can only provide a fixed functionality. However, flexibility is an important aspect of cryptographic designs where a module can perform multiple operations with different configurations. In this work, we combine flexibility with lightweight designs and propose two cryptographic engines based on the SIMON block cipher. The first proposed engine is the Flexible SIMON, which can execute all configurations of SIMON thus enables an adaptive security with variable key sizes. Our second proposed implementation is BitCryptor, a bit-serialized Compact Crypto Engine that can perform symmetric key encryption, hash computation and pseudo-random-number-generation. The implementation results on a Spartan-3 s50 FPGA show that the proposed engines occupies 90 and 95 slices respectively, which are more compact than the majority of their single purpose counterparts. Therefore, these engines are suitable cryptographic blocks for resource-constrained systems.

This work was supported in part by the National Science Foundation grant no 1115839.

Dedication

Dedicated to my Grandfather
Captain M. Vecdi Tiryaki
Turkish Navy

Anısına,
Albay M. Vecdi Tiryaki
Türk Deniz Kuvvetleri

Acknowledgments

First, I would like to thank my advisor Dr. Patrick Schaumont for his guidance and support throughout my graduate research. Being a part of his research team has been an amazing experience for me and I have learned so much from him in the past two years. I also would like to thank Dr. Leyla Nazhandali for having me as her teaching assistant for a semester and Dr. Michael Hsiao for sparing his time and being on my committee.

I would like to express my gratitude for my mother Neslihan, my father Turgay and the rest of my family for their love and encouragement all my life. Without their support I would not be here. I have a most special thank you to my girlfriend and soul-mate Ayse, who never left me alone although we had thousands of miles between us.

Finally I would like to thank my friends here in the United States and Turkey. Thank you Aydin, Bilgiday and Nahid for your support and contributions to my work in the lab and everyone else who made Blacksburg a better place for me.

Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
ASM	Algorithmic State Machine
BRAM	Block Random Access Memory
CBC	Cipher Block Chaining
CMVP	Cryptographic Module Validation Program
DBL	Double Block Length
DDP	Double Datapath
ECB	Electronic Code Book
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IoT	Internet of Things
IP	Intellectual Property

IV	Initialization Vector
LUT	Look Up Table
Mbps	Megabits Per Second
MHz	Mega Hertz
NIST	National Institute of Standards and Technology
NRE	Non-Recurring Engineering
NSA	National Security Agency
PRNG	Pseudo Random Number Generation
RAM	Random Access Memory
RFID	Radio Frequency Identification
ROM	Read Only Memory
RTL	Register Transfer Level
SBL	Single Block Length
SRD	Shift Register Down
SRL	Shift Register Look Up Table
SRU	Shift Register Up
WC	Wearable Computer
WSN	Wireless Sensor Network

Chapter 1

Introduction

1.1 Introduction

With the advancements in technology, electronic devices have become an essential part of our everyday lives. These devices range from high-end ones such as servers, computers and laptops to more constrained ones like RFID tags, smart cards and wireless sensor nodes. With the Internet of Things on the horizon, all electronic devices like coffee machines and garage doors will be connected via the Internet and communicate with one another to provide the best user experience. However, with a large number of nodes connected to the outside world, the devices become more susceptible to attacks and managing the security of the system becomes an important issue. Some of the devices access sensitive information while some of them only perform simple tasks set by the user. But eventually, all the devices require some level of security to protect itself and the user from attacks from malicious adversaries. Data confidentiality and integrity are two of the security aspects that should be considered to provide the required protection on the system and we rely on cryptographic algorithms to supply this security level.

Block ciphers are the building blocks of secure systems as they enable sending a message

Table 1.1: Internet of Things Examples

Application	**Security Primitive**	**Security Level**
Remotely Controlled Thermostat	Authentication, Confidentiality	64-bit (in-home network), 128-bit (outside home gateway)
Intelligent Light Bulb	Authentication	64-bit
Door/Window Sensor System	Authentication, Integrity	128-bit
Wireless Garage Door	Authentication	128-bit

over a non-secure medium. These ciphers perform symmetric-key encryption by mapping a block of input plaintext to an output ciphertext using a secret key. Once the ciphertext is generated, it can only be decrypted back into the plaintext by using exactly the same secret key. Rijndael is the most widely used block cipher algorithm and it is used as the Advanced Encryption Standard (AES) [17]. Even though Rijndael serves as the AES, its area-cost restricts its use in resource-constrained domains [5]. This is where lightweight cryptography shines. The goal of lightweight cryptography is to minimize the area of implementing and executing an operation while preserving similar or slightly reduced levels of security.

Typically, cryptographic modules are fixed during design time and they perform their operations with some predefined parameters. Due to this restriction, these single purpose modules can only work with a specific configuration. However, by having a flexible module, the system can change the parameters and the operations during runtime, thus a single module can perform multiple task. Flexibility is especially important in the Internet of Things where multiple devices with different capabilities communicate with each other. Depending on the device and/or the sensitivity level of the data, the required security level (ie. key size) or the protocol might change. Table 1.1 gives several example applications in the Internet of Things with different security primitive and security level requirements. For example, the remotely controlled thermostat needs user authentication to get access to the thermostat

and settings confidentiality to keep the settings private. If the user is communicating with the device through the in-home network, then a 64-bit key would be sufficient as the network itself is trusted. However, the device requires a higher security profile (ie. 128-bit key) for any access from outside the home gateway since the communication with the outside world is more vulnerable against attacks. So, a single flexible module with multiple key sizes and cryptographic operations can provide confidentiality and integrity across many Internet of Things devices.

To have a flexible design, the first choice would be using a microprocessor such as MSP 430 that can run any cryptographic algorithm. However, in the domain of lightweight cryptography, using a microprocessor would not be suitable due to the constrained resources of the devices. Therefore, having a dedicated, flexible and compact cryptographic hardware module would yield a more efficient solution.

In this work, we incorporate the merits of lightweight cryptography and flexibility to provide a compact and flexible crypto engine. As the first step, we improve the bit-serial implementation of the SIMON block cipher to add flexibility so that it can work with all configurations with different block and key sizes in a single hardware module. Then, by using the same design methodology, we propose BitCryptor, a compact crypto engine that can execute fundamental cryptographic operations such as symmetric key encryption for data confidentiality, hash function for data integrity and Pseudo-Random-Number-Generator (PRNG).

The results for our flexible implementation of the SIMON block cipher show that it is smaller than all the other block ciphers while working with multiple key and block sizes. Likewise, BitCryptor occupies less area compared to majority of the other single purpose cryptographic modules and flexible designs.

1.2 Contributions

This work has the following contributions:

- We improve the bit-serial implementation of the SIMON block cipher and implement a flexible version that can work with ten different block and key sizes while maintaining its compactness. We present the details of the hardware architecture of the flexible design and show the trade-offs between performance, security and flexibility.

- We propose BitCryptor, a bit-serialized compact crypto engine that can perform fundamental cryptographic operations such as symmetric key encryption, hashing and PRNG. We describe the details of the hardware architecture and the control mechanism of the engine and present an extensive performance comparison with other similar modules.

1.3 Organization

This thesis is organized as follows. Chapter 2 covers the flexible implementation of the SIMON block cipher. It gives the details of the SIMON block cipher, the flexible hardware architecture and finally it shows the results and trade-offs. Chapter 3 shows the architecture details of the BitCryptor, describes how it works and compares performance results with other designs. Chapter 4 concludes the thesis.

1.4 Related Articles

- A. Aysu, E. Gulcan, P. Schaumont. SIMON says: Break Area Records of Block Ciphers on FPGAs. Embedded Systems Letters, IEEE, June 2014

- E. Gulcan, A. Aysu, P. Schaumont. A Flexible and Compact Hardware Architecture for the SIMON Block Cipher. Third International Workshop on Lightweight Cryptography for Security and Privacy - LightSEC 2014

- E. Gulcan, A. Aysu, P. Schaumont. BitCryptor: Bit-Serialized Compact Crypto En-

gine. International Conference on Field-programmable Logic and Applications - FPL 2015, Submitted

- D. Moriyama, A. Aysu, E. Gulcan, P. Schaumont, M. Yung. End-to-end Design of a PUF based Privacy Preserving Authentication Protocol - Workshop on Cryptographic Hardware and Embedded Systems - CHES 2015, Submitted

Chapter 2

Flexible SIMON

2.1 Introduction

Security is a new design dimension for digital systems [25]. Schaumont *et al.* labels this dimension as Risk and shows that flexibility, performance and risk are the main design dimensions of secure embedded systems [37]. Furthermore, they argue that a good design should consider the trade-offs between these dimensions. In that framework, performance refers to the capability of the system for a given target metric (throughput, energy-efficiency, area, etc.), risk is the potential for loss, and flexibility is the ability to (re)define the system parameters and behavior. The dimension of flexibility is even more important especially for applications with a diverse set of requirements. Wireless sensor networks (WSN) are an outstanding example for this scenario. WSN typically consist of a large number of devices (nodes) that are one-time programmed and deployed in the field. The nodes run for long periods of time without human intervention.

A common practice of flexibility is to implement adaptive security for WSN. Younis *et al.* proposes an adaptive security provision for wireless sensor nodes [44]. They propose an efficient protocol in which the encryption strength (key-size) varies between 32-bits to 128-

bits depending on the trust level of the nodes. Obviously, if a node is more trusted, an encryption with a lower level of security allows computation savings. Wang *et al.* argues a similar case for computation savings where the sensitive data within the network is encrypted with a higher security level, while the less important information is encrypted using shorter keys [41]. Sharma *et al.* claims that the application diversity of WSN ranges from military surveillance to agriculture farming, each of which requiring a different set of minimal security mechanisms [38]. Then, they present a comprehensive security framework that can provide security services for a variety of applications. Finally, Portilla *et al.* provides a case study on FPGAs using the Elliptic Curve Cryptography and proposes a solution for a public-key based adaptable security on WSN [35].

Cook *et al.* approaches flexibility from another perspective [14]. If an input plaintext is even one-bit larger than the encryption block-size n, it has to be padded to $2n$ and the encryption should run more than once. Therefore, they introduce an elastic block cipher that improves the inefficiency by allowing a variable block-size. This methodology uses a fixed key-size with a variable block-size.

Our solution combines the merits of both visions. We propose an architecture that can have both variable block-size and key-size. Using such a flexible architecture enables a single device to offer adaptable security for a variety of applications, or multiple levels of security within an application. It can also reduce the redundancy of slightly longer messages by changing the encryption block-size. Our unified architecture also minimizes the licensing/certification efforts since we use a single design for many different use-cases. The complex cryptographic module validation programs like NIST CMVP [33] also make the single hardware running all configurations advantageous over the collection of many that can execute a single configuration. Yet, the proposed architecture is still very compact which makes it very suitable for light-weight applications.

From a design methodology perspective, the proposed hardware provides flexibility (at the expense of area and throughput) to the system by enabling on-the-fly security configuration

Table 2.1: SIMON Parameters

Security Configuration	Block Size (2n)	Key Size	Word Size (n)	Key Words (m)	Rounds
1	32	64	16	4	32
2	48	72	24	3	36
3	48	96	24	4	36
4	64	96	32	3	42
5	64	128	32	4	44
6	96	96	48	2	52
7	96	144	48	3	54
8	128	128	64	2	68
9	128	192	64	3	69
10	128	256	64	4	72

management. It also allows a trade-off between the performance and risk. Our results show that the system can increase the security from 64-bits to 256-bits (from toy-settings to high-profile security) with a throughput degradation of a factor of 2. Moreover, to our best knowledge, the proposed flexible hardware architecture of SIMON is still smaller than other block ciphers of similar security level.

2.2 SIMON Block Cipher

SIMON is a Feistel-based lightweight block cipher recently published by NSA, targeted towards compact hardware implementations [7]. SIMON has ten configurations optimized for different block and key sizes providing a flexible level of security. Table 2.1 shows the parameters for all configurations of SIMON. The word size n is the bit length of each word

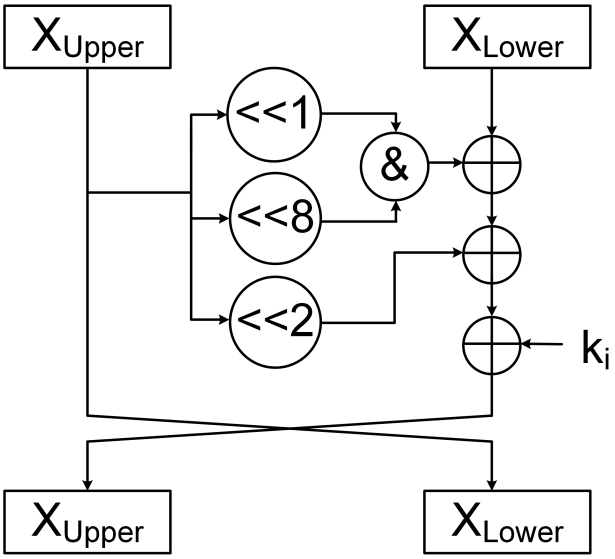

Figure 2.1: SIMON Round Function

in the Feistel network, which makes the block size to be $2n$. The key length is defined as a multiple of the Feistel word size, and the parameter m indicates the number of Feistel words in a key. Security configuration is a new parameter that we introduce to select the desired configuration of SIMON.

2.2.1 Round Function

Figure 2.1 shows the round function for all configurations of SIMON. X_{upper} and X_{lower} respectively denote the upper and lower words of the block and they are n-bits each. These two words hold the initial input plaintext and the output after each round is executed. The round function consists of bitwise AND, bitwise XOR, and circular shift left operations. In each round, shifting and bitwise AND operations are performed on the upper word and it is XORed with the lower word and the round key. The resulting value is written back to the upper word while its content is transferred over to the lower word. The round function continues to run repeatedly until the desired number of rounds is reached.

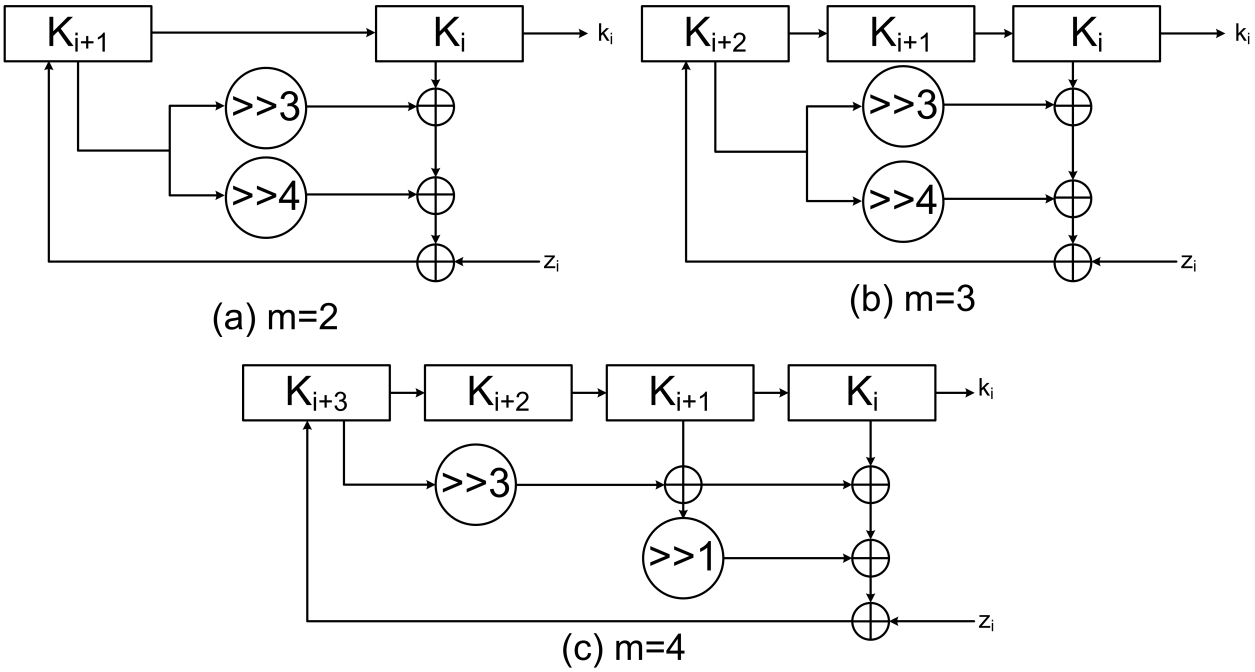

Figure 2.2: SIMON Key Expansions

2.2.2 Key Expansion

SIMON block cipher needs unique keys for each round and the key expansion function generates these round keys. Unlike the round function, there are three different configurations of key expansion as the number of words in a key can be 2, 3 and 4 depending on the configuration. Figure 2.2 shows the key expansion functions for three different key lengths, corresponding to two, three or four Feistel words respectively (m=2, 3 or 4). The block K_i holds the round key for the i^{th} round. For $m = 2$ and $m = 3$, the logical operations of the key expansion function are identical. The most significant word is circular shifted right by 3 and 4, and it is XORed with the least significant word and the round constant z_i. For $m = 4$, there is an extra step where the most significant word (K_{i+3}) is circular shifted right by 3, XORed with K_{i+1}, then circular shifted right by 1 and XORed with the least significant word and the round constant. At the end of each key expansion, the new round key is written into the most significant word, and all the words are shifted one word right. As K_i is the key used in the current round, it will no longer be needed and is overwritten. The key expansion

function has a sequence of one bit round constants (z_i) used for eliminating slide properties and circular shift symmetries. There are five different round constant sequences uniquely tuned for each configuration to provide a cryptographic separation between the different configurations.

2.3 Hardware Implementation

When implementing a block cipher on hardware, there are several parallelism choices(bit level, round level, and encryption level) that affect the area and throughput of the design. In bit level parallelism, the input size of the operators range from one bit to n-bits where n is the block size. In round level parallelism, we can have one round up to r-rounds per clock cycle where r is the total number of rounds of the block cipher. Finally, in encryption level parallelism, we can have one encryption engine up to e encryption engines where e is the maximum number of engines that can fit in our area constraints. Depending of the chosen levels of parallelism, our design space will range from p parallel encryptions per clock cycles to one bit of one round of one encryption engine per clock cycle. In order to keep the area of our design as low as possible, we used the lowest parallelism level of one bit of one round of one engine, which is also called the bit-serial implementation.

2.3.1 Bit-Serial

Figure 2.3 shows the details of the round (a) and key expansion functions (b,c,d) of the bit serial SIMON. The current state holds the words that are used in the current round and the next state holds the words that are generated after the execution of the first round and will be used in the next round. Both of these states share the same set of memory elements and they are overwritten in every round. In the key expansion functions, K_i denotes the key that will be used in the i^{th} round. The highlighted bits indicate the bits that are processed at the first clock cycle of each round.

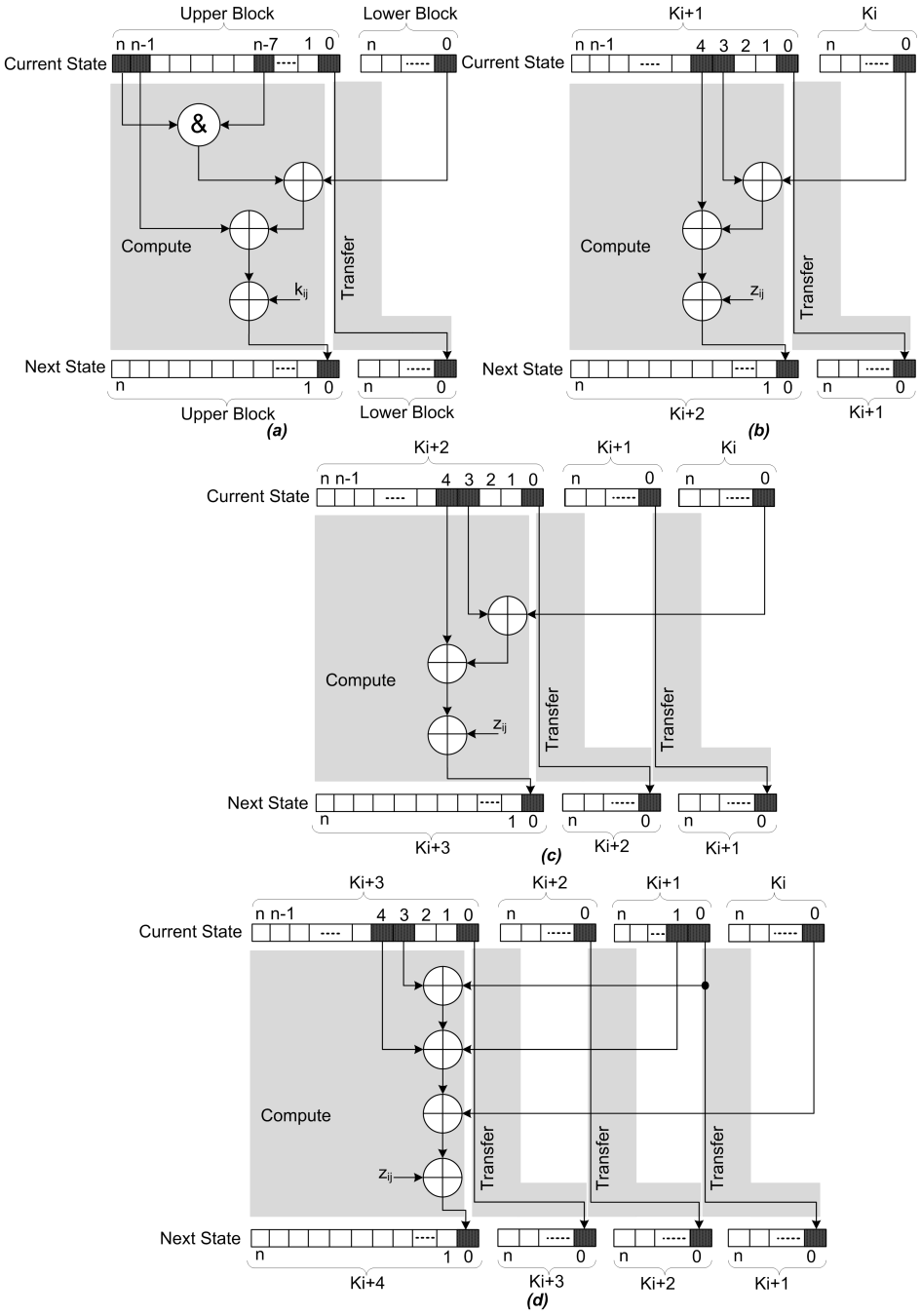

Figure 2.3: (a) SIMON Bit-serial Round Function, (b) SIMON Bit-serial Key Expansion for $m = 2$, (c) SIMON Bit-serial Key Expansion for $m = 3$, (d) SIMON Bit-serial Key Expansion for $m = 4$

Both the key expansion and the round function consist of two phases: Compute and Transfer. The compute phase reads the necessary bits from the current state, performs logic operations on them and writes the resulting bit into the upper block of the next state, while the transfer phase copies the contents of a word in the current states to a lower word in the next state. For the key expansion, there are three different functions depending on the number of key words. The compute phase is the same for $m = 2$ and $m = 3$ where only three bits are necessary from upper and lower words. For $m = 4$, two additional bits are required from the word K_{i+1} to compute the next state bit. The number of transfer phases required to finish one expansion also changes with the key words number.

The bit serial implementation of the SIMON block cipher fits very well into the resources of an FPGA as we can use the Look Up Tables(LUT) as memory elements. In a Spartan-3 family FPGA, each LUT can be configured as an 16x1 Shift Register LUT(SRL), in which we can store the words of the round and key expansion functions. Since we are reading from and writing into the SRL one bit per clock cycle, we will call them FIFOs throughout this thesis. By using these FIFOs we can overlap the compute and transfer phases to process one bit in every clock cycle.

2.3.2 Round Function

The round function of the SIMON block cipher is the same for all ten configurations except for the size of the memory elements (words). In the Feistel network of the round function, the block is separated into two words, each keeping one half of the complete block. As the block size changes with different versions, the size of the FIFOs holding these words also change accordingly. In order to have a round function that can work with any of the ten versions, we need to have a flexible length of FIFOs.

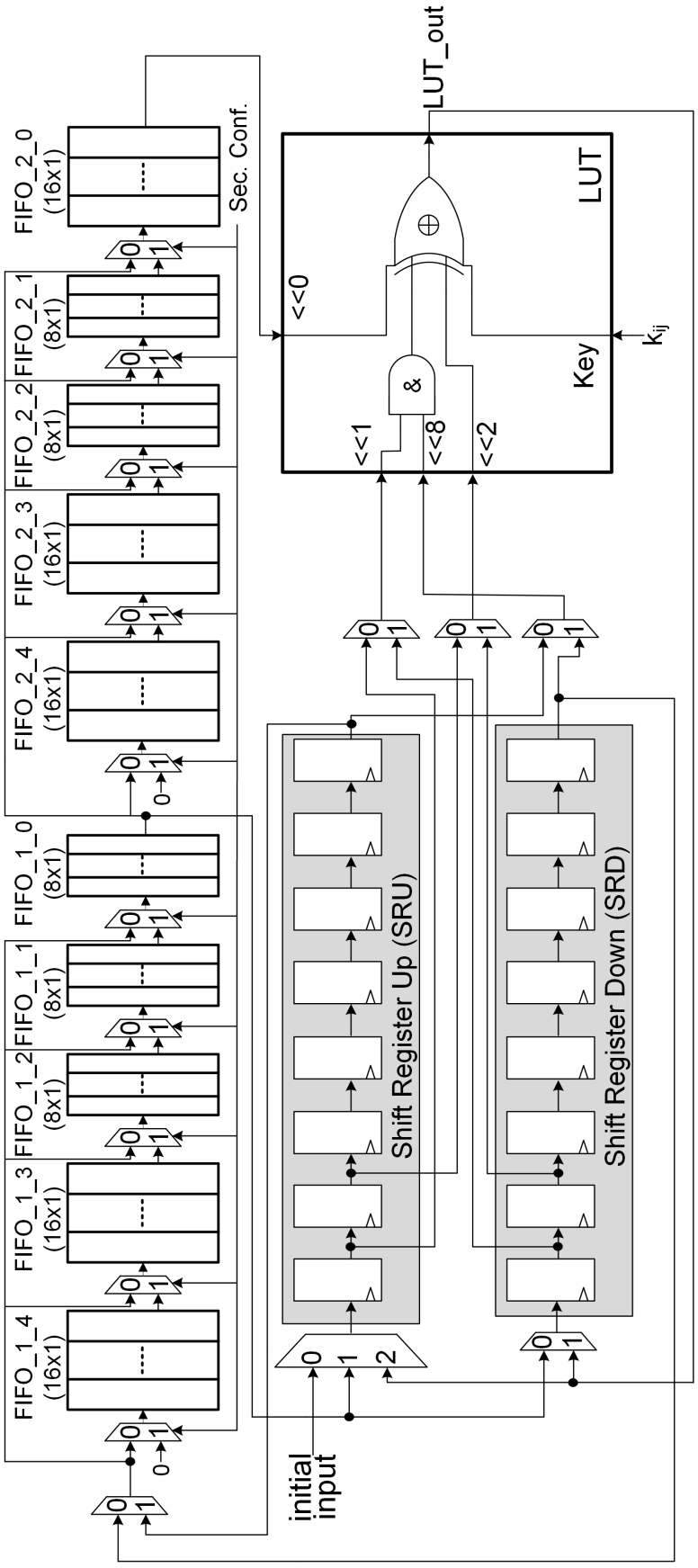

Figure 2.4: SIMON Bit-serial Flexible Round Function

Figure 2.4 shows the bit serial implementation of the flexible round function of SIMON. There are two groups of FIFOs named FIFO_1 and FIFO_2, which hold the upper and lower words of the block respectively. Each group is divided into subsections of FIFOs with different sizes, connected together through multiplexers. The sizes of the subsection FIFOs are selected such that each additional FIFO increases the total size to be equal to the desired word size. FIFO_1 is smaller than FIFO_2 as the eight most significant bits of the upper word are stored in the Shift Registers Up or Down. These shift registers are required due to the circular shift pattern of the round function. As we are using one bit input-output FIFOs, we cannot access the intermediate bits. Therefore, the registers store the first eight bits in flip-flops to enable parallel access. According to the security configuration input, multiplexers select the required size of the FIFOs for both the upper and lower words and route the incoming data to the correct subsection of FIFOs.

Each FIFO has a two input multiplexer at its input that bypasses the unused FIFOs and routes the FIFO group input to the desired subsection FIFO. When input '0' is selected, the FIFO group input is connected to the subsection FIFO and when input '1' is selected, the next FIFOs output is connected. Figure 2.5 shows the required FIFO numbers for all security configurations.

For example, if the security configuration input is 1, the round function uses FIFO_1_0 and FIFO_2_0 while the rest of the FIFOs are grounded. The output of FIFO_1_0 is connected to the input of FIFO_2_0 to perform the transfer operation, and the data coming from SRU or SRD (depending on the round number) is connected to the input of FIFO_1_0. When the security configuration input changes to 2, the word size increases from 16 bits to 24 bits. Therefore, one additional FIFO of size 8 is needed to store the upper and lower words. The multiplexers at the inputs of FIFO_1_0 and FIFO_2_0 now select the output of the FIFOs to their left (select input 1), and the FIFO group inputs are routed to FIFO_1_1 and FIFO_2_1 (select input 0).

One important aspect of the bit serial implementation is the use of two sets of shift registers

Security Conf.	Datapath FIFO No.										Key Expansion FIFO No.																			
	1_0	1_1	1_2	1_3	1_4	2_0	2_1	2_2	2_3	2_4	0_0	0_1	0_2	0_3	0_4	1_0	1_1	1_2	1_3	2_0	2_1	2_2	2_3	2_4	3_0	3_1	3_2	3_3	3_4
1	x					x					x					x				x					x				
2	x	x				x	x				x	x								x	x				x	x			
3	x	x				x	x				x	x				x	x			x	x				x	x			
4	x	x	x			x	x	x			x	x	x							x	x	x			x	x	x		
5	x	x	x			x	x	x			x	x	x			x	x	x		x	x	x			x	x	x		
6	x	x	x	x		x	x	x	x		x	x	x	x											x	x	x	x	
7	x	x	x	x		x	x	x	x		x	x	x	x						x	x	x	x		x	x	x	x	
8	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x										x	x	x	x	x
9	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x	x	x	x	x	x	x
10	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 2.5: FIFO Usage Schedule

named Shift Register Up (SRU) and Shift Register Down (SRD). As the round function of SIMON requires three circular shift left operations (1, 2 and 8) on the upper block, the current state bits required to compute the next state bit do not go in a sequentially ordered manner. For example, when the block size n is 32, in order to compute the bit #0 of the next state, we need to use the bits #31,#30 and #24 of the upper block of the current state. However, the new computed bit #0 should also be stored in the same memory element of the upper block which causes a conflict. We need to use the bit #0 of the current state to compute the bit #1 of the next state so we cannot overwrite it yet. In order to solve this problem, we implemented the ping pong registers SRU and SRD. In the even numbered rounds, the output of the LUT is written to the SRD and the output of the FIFO_1 is written to the SRU. Also for the first eight bits, the input of the FIFO_1 is connected to the output of SRU and for the rest it is connected to the output of SRD. In the odd numbered rounds, we interchange the usage of SRU and SRD. By using this technique, we append the least significant eight bits of the upper block to its most significant bits to solve the circular shift problem and we can finish one round in n clock cycles.

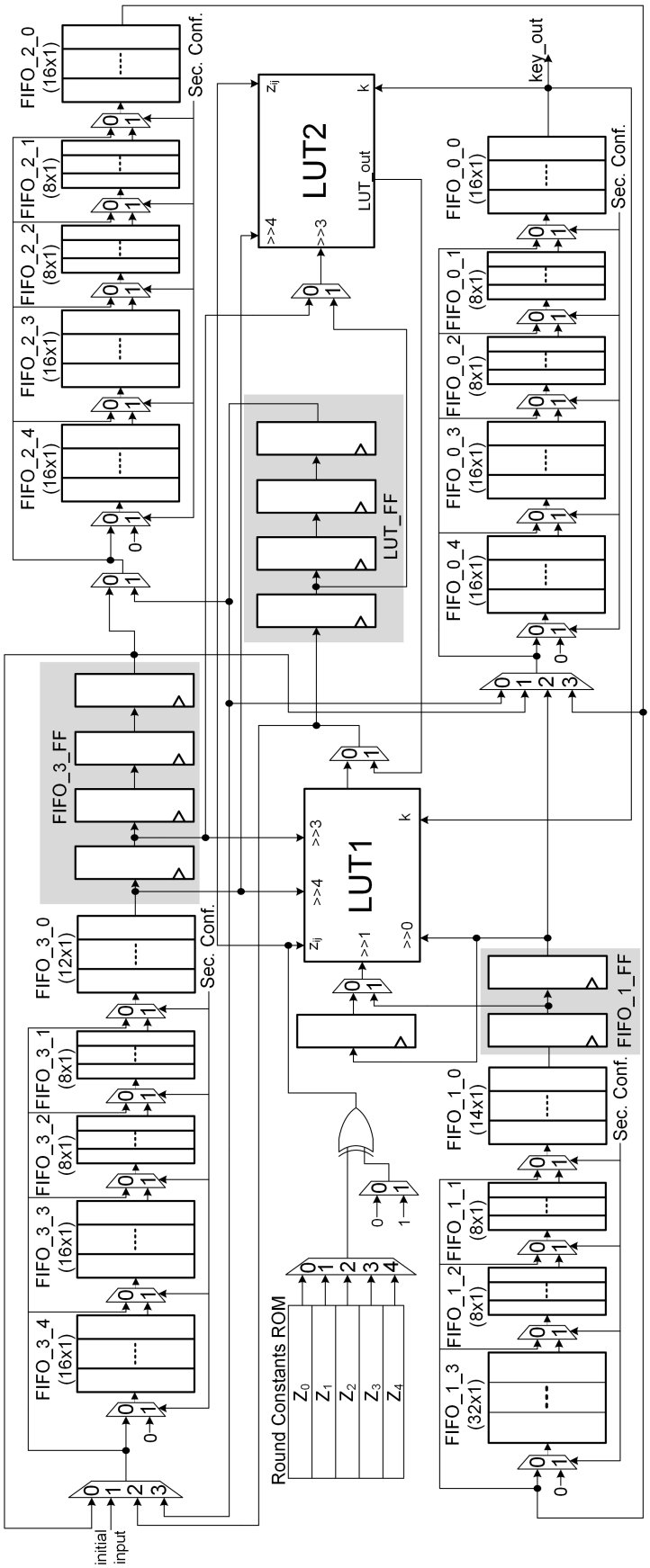

Figure 2.6: SIMON Bit-Serial Flexible Key Expansion

2.3.3 Key Expansion

Unlike the round function, there are three different key expansion functions depending on the block size and the key size. Figure 2.6 shows the flexible bit-serial key expansion of SIMON. There are four groups of FIFOs that store the round keys and similar to the round function, they are divided into subsection FIFOs in order to achieve a flexible size. Since the logical operations for the key word number $m = 4$ are different, we need two LUTs to perform the different key expansion function operations. For $m = 2$ and $m = 3$ the hardware uses LUT2 for the logical operations, while for $m = 4$ it uses LUT1. A LUT based ROM stores the round constants and according to the security configuration input, the multiplexer selects the appropriate sequence.

Another difference of key generation is the dependence of the FIFO group activity to the security configuration input. As there are three possible numbers of key words ($m = 2, 3, 4$), not only the number of subsection FIFOs but also the number of FIFO groups utilized should be flexible. The number of FIFO groups required for each security configuration is equal to the number of key words m of the selected configuration. For $m = 2$ the hardware only uses FIFO_0 and FIFO_3. When $m = 3$ it also utilizes FIFO_2, and if $m = 4$ it enables all four FIFO groups. Additionally, the number of subsection FIFOs changes with the key size. Figure 2.5 gives the details of which FIFOs are used for all the security configurations.

As it can be seen in Figure 2.6, FIFO_3 and FIFO_1 have four (FIFO_3_FF) and two (FIFO_1_FF) additional flip-flops at their outputs, respectively. The necessity of these separate flip-flops come from the circular shift operations of the key expansion function. We used the same technique to overcome the circular shift patterns of the round function, but this time we put the flip-flops at the end of the FIFOs, as the key expansion function uses circular shift right, rather than left. At the first four clock cycles of each round, the input for FIFO_3 is the output of FIFO_3_FF. This way, the least significant four bits of the word are appended into the most significant four bits. At the same time, the output of LUT has to be connected to the same memory element which causes a conflict. Therefore, the

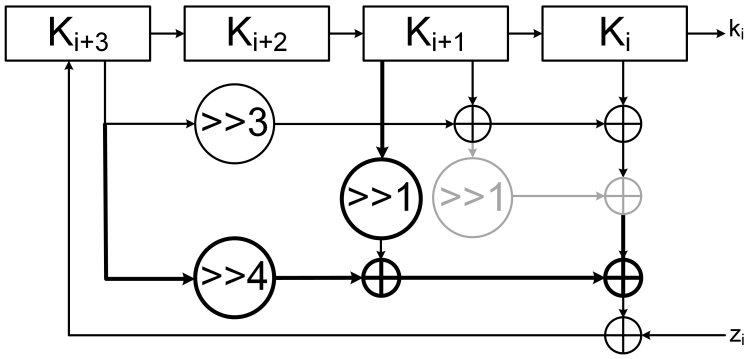

Figure 2.7: SIMON Modified Key Expansion Function for $m = 4$

architecture uses another set of four flip-flops (LUT_FF) that store the output of the LUT for the first four clock cycles. After this period ends, FIFO_3 can directly store the output of LUT since appending more bits is not necessary. At the beginning of the second round, the content of FIFO_3_FF is not fresh as it contains the four bits from the previous round. Therefore, FIFO_3_FF will only be active in the first round. LUT_FF takes its responsibility to append the first four bits to FIFO_3 and also store the outputs of the LUT. Note that in this discussion we do not mention the security configuration input because no matter what the configuration is, FIFO_3 will use this scheduling, only the size of the subsection FIFOs will change.

For $m = 2$, FIFO_3 group stores the upper key word and FIFO_0 group stores the lower one. The transfer operation performs data transfer operation between these two FIFOs. Since the LUT_FF stores the first four bits of each new computed key, during this period the input of FIFO_0 is LUT_FF, and for the rest of the clock cycles, it is FIFO_3. For $m = 3$, in addition to FIFO_0 and FIFO_3, the hardware also utilize FIFO_2 group to store the additional key word. There are two concurrent transfer operations; the first from FIFO_3 to FIFO_2, and the second one from FIFO_2 to FIFO_0. LUT2 computes the logical operations for both $m = 2$ and $m = 3$.

Executing a circular shift operation after a logical operation is problematic for bit-serialized implementation. Therefore, the original form of the key expansion for $m = 4$ is not suitable

for a bit-serial implementation because it requires a circular shift right operation after the XOR of K_{i+3} and K_{i+1}. In order to solve this problem, we modify the key expansion function for $m = 4$. Figure 2.7 shows the required transformation. The gray regions highlight the original operations which are replaced with the bold regions. Originally, the output of the XOR operation has two fanouts, one going directly to another XOR with K_i and the second one to a circular shift operation. We moved the circular shift right by 1 operation from the output to the inputs of the XOR. The XOR from K_{i+3} was originally circular shifted right by 3 and when we shift it one more after the modification, it becomes a circular shift right by 4. Similar to the functionality of FIFO_3_FF, FIFO_1_FF enables the circular access pattern of $m = 4$.

2.4 Implementation Results

The proposed hardware architecture is written in Verilog HDL. The Verilog HDL RTL codes are synthesized to the Xilinx Spartan-3 s50 FPGA using a speed grade of -5, and to the Spartan-6 lx4 FPGA using a speed grade -3. Then, the resulting netlists are placed and routed to the same FPGAs using Planahead. In order to minimize the slice count, we hand-pick our design elements and assign their mapping into the slices.

2.4.1 Area

Comparison with other block ciphers

Figure 2.8 shows hardware resource utilization of our architecture and the previous work. We have compared our work with the smallest version of AES[12], as well as alternative compact block cipher implementations such as PRESENT[43], HIGHT[43], SEA[29], XTEA[23], CLEFIA[11] and ICEBERG[39]. In order to have a fair comparison, we map our hardware into the same FPGA (Spartan-3) with the previous work, but we also show the occupied

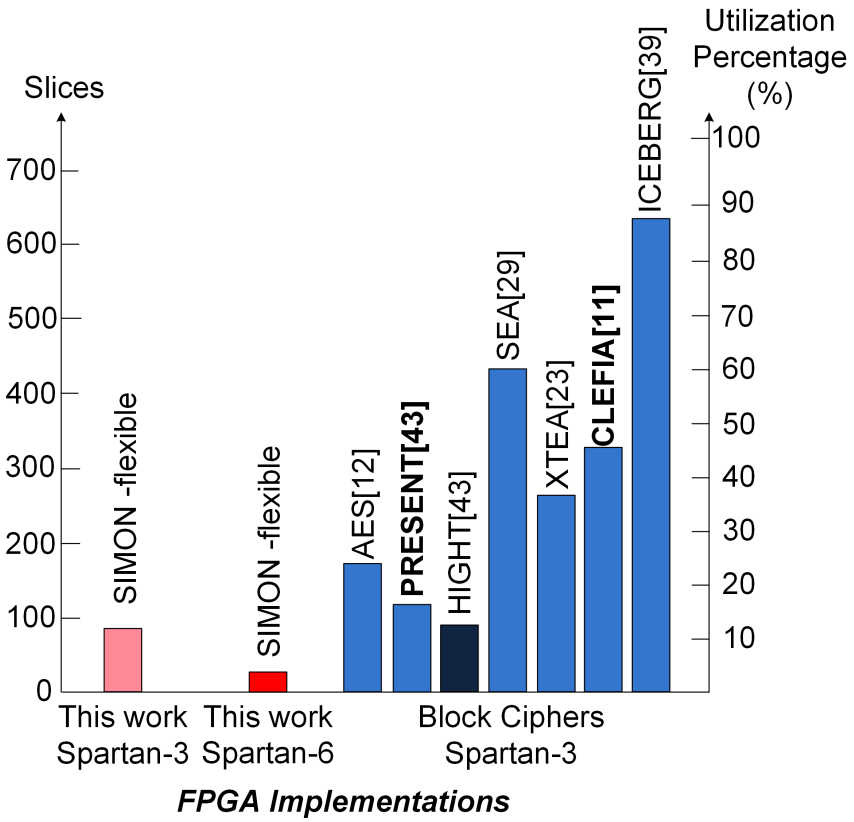

Figure 2.8: Occupied Slices and the Resource Utilization Ratio of Flexible SIMON vs. Previous Work.

area on a more recent FPGA like Spartan-6. The proposed hardware occupies 90 and 32 slices on a Spartan-3 and a Spartan-6 FPGA, respectively. Out of all these implementations, our hardware architecture is the only one that provides the flexibility, whereas the rest of them use a fixed key and block size. Yet, our flexible hardware architecture is still smaller than all block ciphers. These results show that our bit-serial design methodology and our back-end tool-optimization was able to achieve very compact hardware instances while still enabling the flexibility.

Comparison with other flexible architectures

There are several architectures in literature that implement the multiple configurations of AES. However, none of them where targeted for light-weight platforms. AES has three configurations, AES-128, AES-192, and AES-256 all use 128-bit block size with 128,192 and 256 bit key-size, respectively. McLoone *et al.* proposes an architecture that can perform all configurations using 4681 slices [30]. Li *et al.* later optimizes this implementation and reduces the slice count to 3223 slices [27].

Comparison with commercial soft-core processors

One alternative way to implement a flexible encryption engine on an FPGA is through a soft-core processor. Xilinx provides Microblaze whereas Altera proposes NIOS as a commercial soft-core processor. These processors execute software that enables the capability of running all configurations. However, the minimum area-cost of a NIOS and Microblaze is approximately 700 logic elements (logic element is 1 LUT + 1 register) and 600 slices, respectively. Picoblaze is an area-optimized Xilinx processor that can bring the area-cost down to 96 slices and 1 BRAM, which is still higher than our memory-free architecture.

2.4.2 Performance vs. Risk Trade-off

Figure 2.9 shows the trade-off between the performance and the risk. As we increase the size of the key, we decrease the risk of the system. However, we also increase the total time of computation because SIMON requires more rounds to complete, and the bit-serial architecture requires more clock cycles to finish one round. For example, if the system selects the security configuration 1, it takes 32 rounds to complete the encryption of a 32-bit block and one round is processed in 32 clock cycles. Therefore, the throughput of the encryption is 5.27 Mbps. On the other hand, the system will be using a key-length of 64-bits which can be regarded as a toy-setting since dedicated machines like COPACOBANA can break

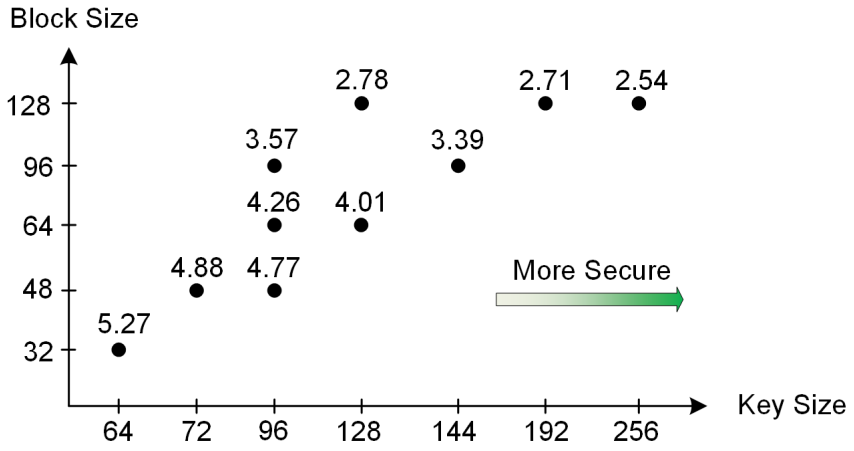

Figure 2.9: Throughput (Mbps) vs. the Security Configuration of SIMON

a block cipher with 57-bits in less than a week [20]. If the system changes its settings to the security configuration of 10, the key size will be 256-bits. Hence, the risk will be much lower, but the throughput will decrease by a factor of 2.

2.4.3 Flexibility vs. Performance Trade-off

Flexibility comes at the expense of performance. Figure 2.10 illustrates the cost of implementing the flexible architecture. We compare our flexible architecture running at the security configuration of 8 to the results of Aysu *et al.*, as they both use 128-bit key size and block size [6]. Since the proposed flexible architecture has to support all available configurations, including the ones that has larger keys and block sizes, the slice count is approximately three times of the fixed implementation. Even though the required clock cycles to complete the encryption is equal for the two architectures, a larger circuit causes longer interconnect delays and a lower maximum achievable frequency. Therefore, compared to the fixed implementation, the throughput of the flexible architecture degrades by 23%.

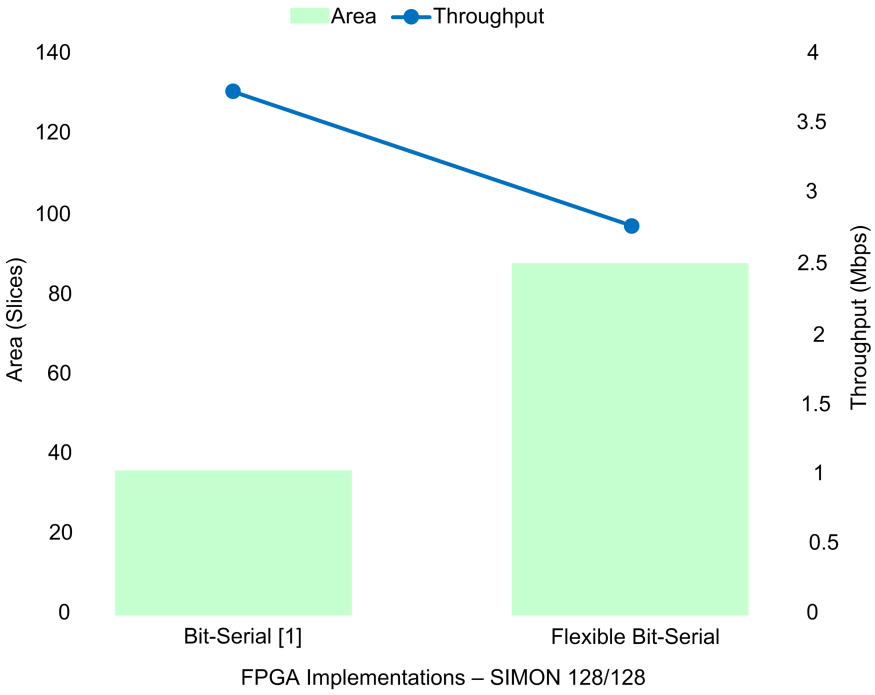

Figure 2.10: The Cost of Flexibility on Area and Throughput

Chapter 3

BitCryptor: Bit-Serialized Compact Crypto Engine

3.1 Introduction

ASIC technology offers a high integration density and a low per-unit price, yet there exist a myriad of applications where FPGAs are preferred over ASICs due to their lower NRE cost and reconfigurable nature. Wireless sensor nodes (WSN) [15], wearable computers (WC) [34], and Internet-of-Things (IoT) [28] are amongst such application domains that require compact solutions and still incorporate FPGAs. Critical applications in these domains inevitably need a method to perform cryptographic operations. Thus, the resource-constrained device should embody this method with minimal area cost.

In this work, we target the FPGA technology and propose BitCryptor, a bit-serialized compact crypto engine that can execute fundamental cryptographic operations. A common practice to improve hardware security is to physically isolate the computational resources of cryptographic procedures from general purpose operations. Then, safety critical operations can be mapped to a relatively smaller hardware that is easier to validate and certify. We

implement such a compact IP block that allows a system designer to bring cryptographic operations into their system. To highlight the area gain of our design, we can compare it with a similar construction. BitCryptor occupies 95 slices, 12% of available resources of a Spartan-3 s50 FPGA whereas the nearest competitor with similar functionalities [26] occupies 916 slices and cannot even fit into the same device. Hence, a system using [26] must migrate to a larger device (eg. Spartan-3 s200), effectively doubling the cost. Although this net cost increase may seem marginal for a single device ($\approx$ \$3.5), it accumulates to a significant quantity if a system utilizes a large population of devices (eg. IoT, WSN, WC).

The challenging area requirements of the target applications enforce designers to apply area-minimizing design methodologies such as bit-serialization. Previously, the routing capabilities of FPGAs did not permit large designs to fit in a single device. Hence, bit-serialization was used to minimize the area of a design and map it into an FPGA [4]. Currently, the routing limitations are not a problem anymore but it is still desirable to use compact architectures to reduce component cost. We revisit and adapt bit-serialization on BitCryptor, targeting an area as small as possible. The design sequentializes datapath and control to process one output bit per clock cycle. The results show that BitCryptor is not only 10$\times$ smaller than a compact AES-based multipurpose design, but it is even smaller than almost all standalone cryptographic modules.

3.1.1 The Need for BitCryptor

Critical applications in constrained embedded systems require a wide range of information security services. These services may include public-key and symmetric-key encryption, true and pseudo-random number generation (PRNG), and hash function. Usually, resource-constrained devices do not utilize public-key encryption and the associated infrastructure with certification authorities or true random number generators due to area and performance limitations. Instead, they use hashing for data integrity, block cipher for data confidentiality, and pseudo-random number generation to generate entropy. Therefore, we design BitCryptor

to support these three operations.

Typical applications for constrained devices involve secure key exchange and entity authentication. Boyd *et al.* enumerate a list of possible protocols for these applications [9]. For example, the protocol with non-reversible functions (section 6.1.5. of [13]) uses a PRNG, hash and encryption, all within a single execution. We are promoting BitCryptor as a generic lightweight crypto engine upon which such protocols can be build as a sequence of BitCryptor commands. The system that utilizes BitCryptor therefore is able to implement cryptographic protocols using minimal area overhead.

3.2 High-Level Description of BitCryptor

The operations of BitCryptor can be handled separately or they can share a single block cipher unit. Since our goal is to minimize the area, we choose a compact block cipher and build all the functionalities around it. Our previous work in Chapter 2 shows that the lightweight block cipher SIMON occupies the smallest area on FPGAs. Therefore, we select SIMON as the core of BitCryptor.

3.2.1 SIMON Block Cipher

The lightweight block cipher SIMON is developed by NSA, targeting compact hardware implementations [7]. So far, conventional cryptanalytic techniques against SIMON did not demonstrate any weaknesses [1], [3], [40]. SIMON has ten configurations with different block and key sizes, giving users the flexibility to choose the best one that fits into their applications requirements. The parameters for all configurations of SIMON are given in Table 3.1. Block size indicates the bit length of the input message to the block cipher while the key size shows the bit length of the key. SIMON is a Feistel-based block cipher and the input block ($2n$) is divided into two words, shown as the word size (n). The key is formed by (m) words making the key size (mn). SIMON using a block size $2n$ and key size mn is denoted as SIMON

Table 3.1: SIMON Parameters for BitCryptor

	Block Size (2n)	Key Size (mn)	Word Size (n)	Key Words (m)	Rounds
Not Secure Enough ↑	32	64	16	4	32
	48	72	24	3	36
	48	96	24	4	36
	64	96	32	3	42
	64	128	32	4	44
	96	**96**	**48**	**2**	**52**
	96	**144**	**48**	**3**	**54**
Too Large ↓	128	128	64	2	68
	128	192	64	3	69
	128	256	64	4	72

$2n/mn$. Equations 3.1 and 3.2 formulate the SIMON round and key expansion functions respectively.

$$R(X_u, X_l) = (X_u \ll 1) \wedge (X_u \ll 8) \oplus (X_u \ll 2) \oplus X_l \oplus k_i \quad (3.1)$$

$$K(i+m) = \begin{cases} k_i \oplus (k_{i+1} \gg 3) \oplus (k_{i+1} \gg 4) \oplus z_i \ for\ m = 1 \\ k_i \oplus (k_{i+2} \gg 3) \oplus (k_{i+2} \gg 4) \oplus z_i \ for\ m = 2 \end{cases} \quad (3.2)$$

3.2.2 Parameter Selection

The parameters we select directly affect the area of the crypto engine. Typically, to reduce the area, lightweight cryptographic systems utilize shorter keys (80-bits). In our design, we aim to find the best configuration that will at least meet this security level while minimizing

the area. Table 3.1 shows the rationale behind the selection process of the configurations. We utilize SIMON 96/96 for symmetric key encryption and PRNG, and SIMON 96/144 for hashing.

One of the challenges in selecting the parameters of the crypto engine is to satisfy the security needs of the hash function. The security level of a hash is determined by the size of the output digest and the probability of a collision on the value of a digest. We choose the most stringent security constraint of strong collision resistance [31] which requires that a hash at a k-bit security level provides a 2k-bit digest. A common practice in building hash functions is to use a block cipher with single-block-length (SBL) constructions like Davies-Meyer [42] or double-block-length (DBL) constructions like Hirose [21]. In SBL, the output size of the hash function is equal to the block size of the underlying block cipher, while in DBL it is twice the block size. To have a strong collision resistance of minimum 80-bits in SBL, the underlying block cipher must have a block size of at least 160-bits. On the other hand, DBL can achieve the same level of security with a block size of only 80-bits.

Figure 3.1 shows the DBL Hirose construction. The input message m_i is concatenated with the chaining value H_{i-1} and fed into the key input. Both block ciphers use the same key generated by a single key expansion function. The Hirose construction requires a block cipher with a key size that is larger than the block size. The digest is the concatenation of the last two chaining values H_i and G_i. The computation equations of H_i and G_i are as follows.

$$H_i = E(G_{i-1} \oplus c, m_i||H_{i-1}) \oplus G_{i-1} \oplus c \tag{3.3}$$

$$G_i = E(G_{i-1}, m_i||H_{i-1}) \oplus G_{i-1} \tag{3.4}$$

The configuration of SIMON that will be used in Hirose construction must have a block size that is at least 80-bits for strong collision resistance and it must have a key size that is larger than the block size. Therefore, out of the three possible candidates from Table 3.1, we select SIMON 96/144 because it gives us the most compact solution and provides a security level even stronger than the minimum requirements. The hash function reads messages in 48-bit

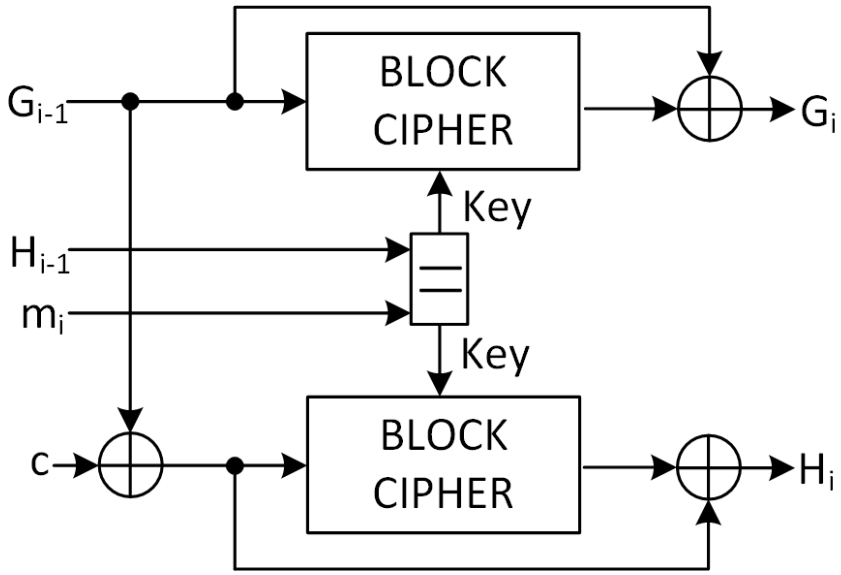

Figure 3.1: Hirose Double-Block-Length Hash Function

blocks and produces a 192-bit digest.

To minimize the area, the crypto engine shares the SIMON block cipher used in hash function to implement symmetric key encryption and PRNG. However, having a 144-bit key is unnecessary in both operations since it is beyond our security requirements. Therefore, it is more feasible to use SIMON 96/96 which has the same block size but a shorter key. In Chapter 2, we show that the flexible architecture of SIMON with all block and key sizes is still very compact. So, the crypto engine uses a flexible SIMON architecture with 96-bit key size for symmetric key encryption and PRNG, and 144-bit key size for hash function. Since only the key size is flexible, the number of words in the key expansion function change while the datapath remains the same.

For the implementation of the PRNG, the crypto engine uses SIMON 96/96 in counter mode of operation. The host system provides a 96-bit key (seed) as the source of entropy for the PRNG and is responsible to reseed the PRNG when necessary. In [16] authors suggest that a single key be used to generate at most 2^{16} blocks of random data. For a block size of 96-bits, this corresponds to approximately 2^{22} bits hence the PRNG module uses a 22-bit counter.

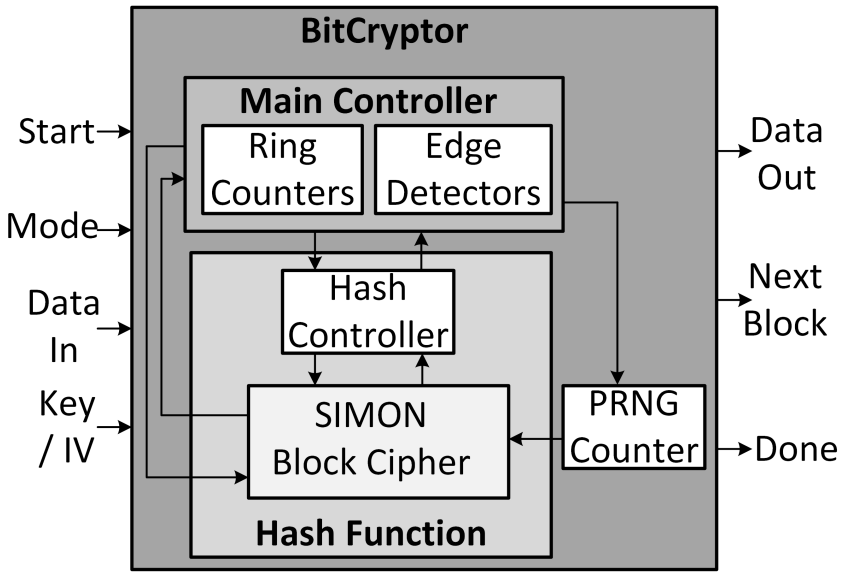

Figure 3.2: Block Diagram of the BitCryptor

3.3 Hardware Implementation

Figure 3.2 shows the block diagram of BitCryptor. The host system indicates the operation *mode* as 1, 2 or 3 for hash, encryption and PRNG respectively. It also provides the *input data*, key/IV (Initialization Vector) and the *start* signal. There are two output signals showing the current status of the engine. The first status signal $Next\ Block$ indicates that a new block of input data can be hashed while the second signal $Done$ states that the operation is completed and the output can be sampled. All the data interfaces ($Data\ In$, Key/IV, $Data\ Out$) are realized as serial ports and the control signals ($Start$, $Mode$, $Next\ Block$, $Done$) are synchronized with the corresponding data.

BitCryptor is an autonomous module and it does not reveal any internal state to outside. To have a secure mode switching, the crypto engine requires the host system to provide a key/IV at the start of each operation. This process overwrites the residues of the key/IV from a previous execution and ensures that no secret information is leaked between two consecutive operations. Output data is revealed together with the done signal if and only if the operation is completed. Hence, an adversary abusing the input control signals cannot

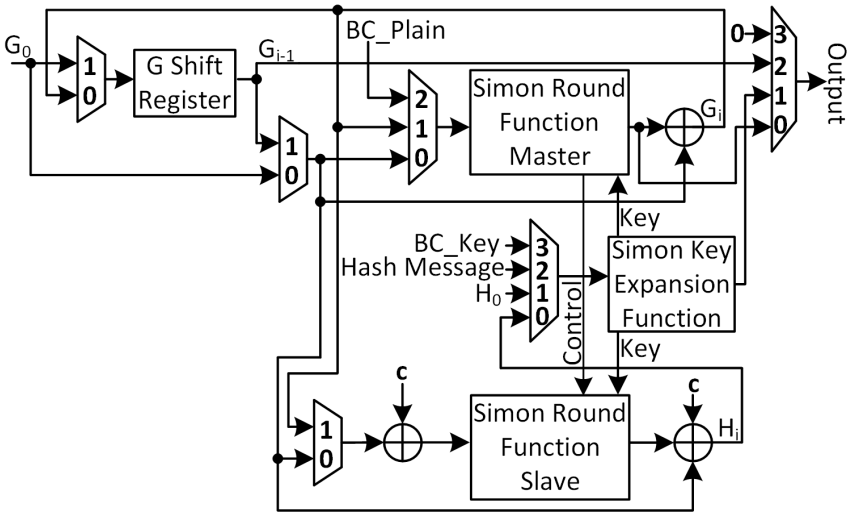

Figure 3.3: Hardware Architecture of the Double-Datapath SIMON and the Hirose Construction

dump out the internal states of the engine.

The main controller of BitCryptor handles selecting the operation modes, starting the functions and reading the output values. Ring counters and edge detectors are used to manage the control hierarchy of modes. The hash function encapsulates the block cipher module and controls it during the hashing operation. Also, the main controller has direct access to the block cipher for encryption and PRNG modes, bypassing the hash controller. Next, we describe the details of the individual operations.

3.3.1 Hash Function

In the Hirose construction, we can use two block ciphers to compute the two halves of the digest. However, this does not necessarily mean that there has to be two full block cipher engines. Since both encryption engines use the same key, they can share a single key expansion function. Moreover, the internal control signals of both round functions are the same so they can share the same control logic. We call this architecture the Double-Datapath (DDP) SIMON with a master round function, slave round function and a shared

key expansion function. The master round function is the full version that is capable of running on its own, independently. On the other hand, the slave round function gets the internal control signals from the master so it can only run while the master is running.

Figure 3.3 shows the DDP SIMON architecture following a master/slave configuration. The hash function has two 96-bit chaining variables G_i and H_i, which are produced by the master and slave round functions respectively. These two variables are loaded with the IV value at the beginning of each operation. A 96-bit shift register (6 SRL-16) stores the G_i value while the shift registers of the key expansion function store H_i. When the hash function is completed, it returns G_i and H_i as the lower and upper 96-bits of the digest respectively.

3.3.2 Symmetric Key Encryption

At the core, the crypto engine uses the SIMON block cipher with a 96-bit block and key size. In [6], Aysu *et al.* implement the bit-serial version of SIMON 128/128 and shows that it is an extremely compact design. To adapt the bit-serial SIMON block cipher to our crypto engine, we modify the implementation in [6] and convert it into SIMON 96/96. We also extend it to perform Cipher-block-chaining (CBC) mode as well as Electronic-code-book (ECB).

Figure 3.3 shows the hardware architecture of the hash function, which also includes the SIMON 96/96 block cipher. When the crypto engine is in encryption mode, it only uses the master round function while the slave round function is inactive. The key expansion function uses the 96-bit key configuration. The input data *BC_plain* and key *BC_key* come directly from the host system through the main controller, bypassing the hash function. When the block cipher completes encryption, it gives the output from the same data output port that is shared with the hash function.

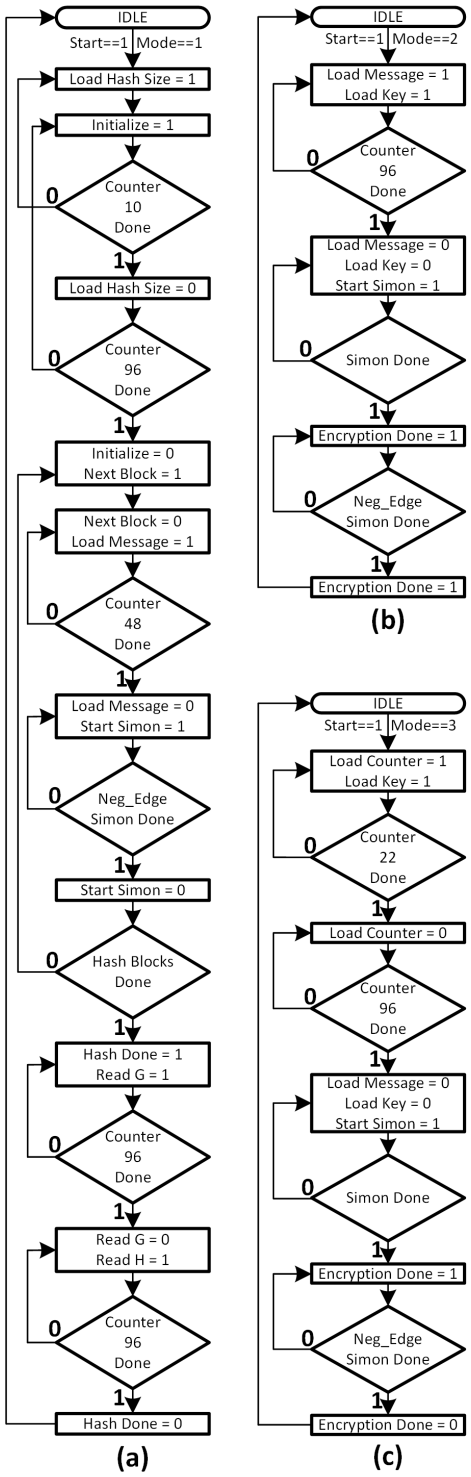

Figure 3.4: ASM Chart for BitCryptor

3.3.3 PRNG

The PRNG uses the SIMON 96/96 in counter mode of operation. When the host system requests a random number, it provides the key as the source of entropy and the PRNG module feeds the 22-bit *PRNG counter* value to the block cipher padded with zeros. The host system is also responsible to change the key after receiving 2^{22} bits of random data. After the block cipher generates the random number, the PRNG module increments the counter value. We verified that the output of the PRNG passes the NIST statistical test suite [36].

3.3.4 BitCryptor Controller

While all three cryptographic modules manage their own internal controls, the top-level controller handles selecting the modes of operation, starting the functions and reading the output values. The bit-serial nature of the crypto engine requires keeping track of the bit position during input and output transmission.

Figure 3.4 (a) shows the control flow of the hash function. When the *start* and *mode* signal for the hash function comes, the controller starts the 96-bit counter and asserts the *initialization* signal while reading in the IV values of G_i and H_i. At the same time, by using the 10-bit counter, the controller reads the 10-bit value that shows how many blocks of data are to be hashed. After 96 cycles the *initialization* signal is deasserted and the controller reads the 48-bit input message and sends the *start* signal to the hash function. When the hashing is completed, the hash function sends a *Hash Done* signal and the controller checks if there are still input blocks to be processed. If the current input block is the final block, the controller sends the *Read G* and *Read H* signals to the hash function for 96 cycles each to read the output digest. Else, the controller sends the *Next Block* signal to the host system, requesting the next block of data.

Figure 3.4 (b) and (c) show the control flow for the symmetric key encryption and PRNG.

The control flow of the two modules are very similar. The difference is that for symmetric key encryption the block cipher input is the message coming from the host system, whereas in PRNG it is the counter value. After the controller loads the inputs to the block cipher, it sends the *start* signal and waits for the done signal. When the *Simon Done* signal is asserted, the block cipher starts to print the output value and it continues to give the output until the *Simon Done* signal is deasserted. Therefore, the controller waits for the negative edge of the *Simon Done* signal so that it can stop giving the output to the host system and return to idle.

3.4 Results

The proposed hardware architecture is written in Verilog HDL and synthesized in Xilinx ISE 14.7 for the target Spartan-3 XC3S50-5 FPGA. In order to minimize the slice count, the synthesized design is manually mapped to the FPGA resources using Xilinx PlanAhead and finally the design is placed and routed. The power consumptions are measured using Xilinx XPower. BitCryptor occupies 95 slices (187 LUTs, 102 Registers) in the target FPGA with a throughput of 4 Mbps for encryption and PRNG, and 1.91 Mbps for hashing at 118 MHz.

Figure 3.5 shows a detailed area comparison of BitCryptor with a compact AES based multipurpose engine [26] and with various standalone block ciphers [6, 11, 12, 23, 29, 32, 39, 43], hash functions [2, 24, 32] and PRNGs [22]. The results show that the bit-serial design methodology and the architecture optimizations using SRL-16 elements can significantly reduce the area. Next, we discuss the details of area comparisons and the performance tradeoff.

Comparison with other multipurpose designs

Most multipurpose designs for FPGAs are optimized for performance and are not suitable for lightweight applications. Bossuet *et al.* survey a number of multipurpose designs and

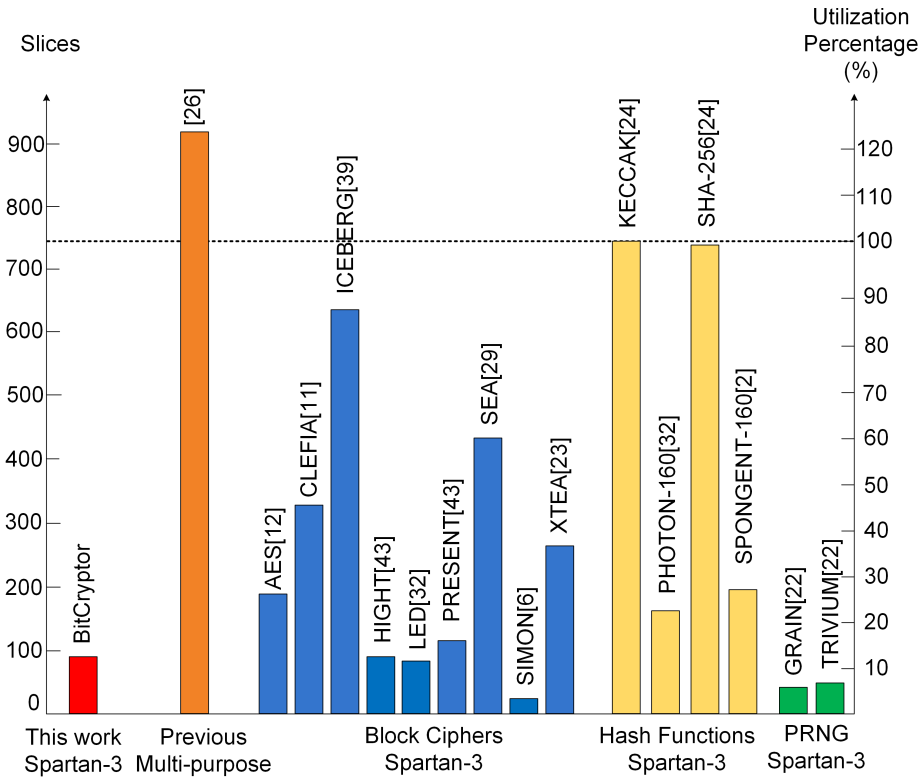

Figure 3.5: Implementation Results and Comparison with Previous Work

reports the smallest to be 847 slices [8]. In [26], Laue *et al.* propose a hardware engine that offers the closest functionality to our design. However, they do not follow a bit-serial design methodology hence it occupies 916 slices on a Virtex-II family FPGA (which has the same slice structure with Spartan-3). Compared to this design, our architecture has an area improvement of almost 10×.

Comparison with single-purpose designs

The results show that our design is more compact than the sum of implementing these functionalities individually. Moreover, it is even smaller than the majority of the lightweight block ciphers and all hash functions. Standalone PRNGs are usually based on simple stream cipher constructions thus making them very compact.

Table 3.2: Area-Performance Tradeoff (@100 MHz)

	Bit-Serial	**Round-Serial**[1]	**Unit**
Block Cipher & PRNG	3.41	169.54	Mbps
Hash Short Block[2]	1.64	83.23	Mbps
Hash Long Block[2]	1.80	86.37	Mbps
Static Power[3]	3.24	14.31	mW
Dynamic Power	7	38	mW
Total Power	10.24	52.31	mW
Average Energy[4]	2.80×10^{-7}	2.85×10^{-8}	J
Energy-Delay	7.79×10^{-12}	1.57×10^{-14}	J-s
Area	95	500	Slice

1. The Round-Serial results are estimated from a simulation of SIMON 96/96 hardware
2. Short block is one 48-bit input block, long block is 1000 48-bit input blocks
3. Static power is scaled with respect to the resource utilization ratio
4. Average energy refers to the averaged energy consumption of three modes

Comparison with soft-core processors

We also compare our area with the software implementations on an FPGA using soft-core processors PicoBlaze [10] and openMSP430 [18]. Good *et al.* provide the smallest soft-core processor in the literature that is capable of running a single-purpose AES encryption [19]. This design utilizes the PicoBlaze processor and it occupies 119 slices and a BRAM ($\approx$ 452 slice equivalent), which is larger than BitCryptor.

Migrating to more recent Xilinx FPGAs

For comparison fairness, we implement our hardware architecture on a Spartan-3 family FPGA. In addition, we also map our design on a more recent Spartan-6 device. On a Spartan-

6 XC6SLX4-3 FPGA, BitCryptor occupies 5% of available resources which corresponds to 35 slices (136 LUTs, 103 Registers) with a maximum frequency of 172 MHz.

Area-Performance Tradeoff

In this work, we focus on area-cost and optimize purely for this design metric. However, the area improvement comes at the expense of throughput and energy-efficiency. Compared to bit-serial architectures, round-serial designs have simpler control and a faster execution time, resulting in a reduced energy consumption and a higher throughput. Table 3.2 quantifies these trade-offs. The round-serial design is approximately two orders of magnitude faster and more energy efficient, but it occupies 5 times the area compared to the bit-serial. However, the power requirement of the bit-serial design is lower due to sequentialization (dynamic) and reduced total area (static).

Chapter 4

Conclusion

In this thesis, we proposed and implemented two flexible and lightweight cryptographic engines for resource constrained devices. We showed that the Flexible SIMON can provide an adaptive security level with on-the-fly configuration of the block and key sizes, but still occupy less area compared to other designs. Then we presented BitCryptor, a bit-serialized crypto engine that can perform fundamental cryptographic operations in a small area. Our results in both cases showed that the bit-serial design methodology and architectural optimizations can significantly minimize the area. We showed the trade-offs between the three design dimensions and compared our designs with other flexible modules. These two cryptographic engines can be very suitable IP blocks in resource constrained devices providing flexibility with a minimal area.

Bibliography

[1] F. Abed, E. List, S. Lucks, and J. Wenzel. Differential and linear cryptanalysis of reduced-round simon. Cryptology ePrint Archive, Report 2013/526, 2013. `http://eprint.iacr.org/`.

[2] M. Adas. On the FPGA based implementation of SPONGENT. 2011.

[3] H. A. Alkhzaimi and M. M. Lauridsen. Cryptanalysis of the SIMON family of block ciphers. Cryptology ePrint Archive, Report 2013/543, 2013.

[4] R. J. Andraka. Building a high performance bit-serial processor in an FPGA. In *Proceedings of Design SuperCon*, volume 96, pages 1–5, 1996.

[5] F. Armknecht, M. Hamann, and V. Mikhalev. Lightweight authentication protocols on ultra-constrained rfids-myths and facts. In *Radio Frequency Identification: Security and Privacy Issues*, pages 1–18. Springer, 2014.

[6] A. Aysu, E. Gulcan, and P. Schaumont. SIMON says: Break area records of block ciphers on FPGAs. *Embedded Systems Letters, IEEE*, 6(2):37–40, June 2014.

[7] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK families of lightweight block ciphers. 2013.

[8] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, and G. Gogniat. Architectures of flexible symmetric key crypto engines—a survey: From hardware coprocessor to multi-crypto-processor system on chip. *ACM Comput. Surv.*, 45(4):41:1–41:32, Aug. 2013.

[9] C. Boyd and A. Mathuria. *Protocols for authentication and key establishment*. Springer, 2003.

[10] K. Chapman. Picoblaze 8-bit microcontroller for virtex-e and spartan-ii/iie devices. *Xilinx Application Notes*, 2003.

[11] R. Chaves. Compact CLEFIA implementation on FPGAs. In P. Athanas, D. Pnevmatikatos, and N. Sklavos, editors, *Embedded Systems Design with FPGAs*, pages 225–243. Springer New York, 2013.

[12] J. Chu and M. Benaissa. Low area memory-free FPGA implementation of the AES algorithm. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 623–626, Aug 2012.

[13] J. A. Clark and J. L. Jacob. A survey of authentication protocol literature: Version 1.0. 1997.

[14] D. L. Cook. *Elastic block ciphers*. PhD thesis, Columbia University, 2006.

[15] A. De la Piedra, A. Braeken, and A. Touhafi. Sensor systems based on FPGAs and their applications: a survey. *Sensors*, 12(9):12235–12264, 2012.

[16] N. Ferguson and B. Schneier. *Practical cryptography*. Wiley, 2003.

[17] FIPS. Pub 197: Advanced encryption standard (aes). *National Institute of Standards and Technology*, 2001.

[18] O. Girard. openmsp430, 2009. `http://opencores.org/project,openmsp430`.

[19] T. Good and M. Benaissa. AES on FPGA from the fastest to the smallest. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 427–440. Springer Berlin Heidelberg, 2005.

[20] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp. Cryptanalysis with COPACOBANA. *Computers, IEEE Transactions on*, 57(11):1498–1513, Nov 2008.

[21] S. Hirose. Some plausible constructions of double-block-length hash functions. In *Fast Software Encryption*, pages 210–225. Springer, 2006.

[22] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj. Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. In *State of the Art of Stream Ciphers Workshop, SASC 2008, Lausanne, Switzerland*, pages 151–162, Feb 2008.

[23] J. Kaps. Chai-tea, cryptographic hardware implementations of xTEA. In D. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology - INDOCRYPT 2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 363–375. Springer Berlin Heidelberg, 2008.

[24] J. Kaps, P. Yalla, K. K. Surapathi, B. Habib, S. Vadlamudi, and S. Gurung. Lightweight implementations of SHA-3 finalists on FPGAs. In *The Third SHA-3 Candidate Conference*, 2012.

[25] P. Kocher, R. Lee, G. McGraw, and A. Raghunathan. Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 753–760, New York, NY, USA, 2004. ACM. Moderator-Ravi, Srivaths.

[26] R. Laue, O. Kelm, S. Schipp, A. Shoufan, and S. Huss. Compact AES-based architecture for symmetric encryption, hash function, and random number generation. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 480–484, Aug 2007.

[27] H. Li. Efficient and flexible architecture for AES. *Circuits, Devices and Systems, IEE Proceedings* -, 153(6):533–538, Dec 2006.

[28] S. Liu, L. Xiang, J. Xu, and X. Li. Intelligent engine room IoT system based on multi-processors. *Microelectronics & Computer*, 9:049, 2011.

[29] F. Mace, F. X. Standaert, and J. J. Quisquater. FPGA implementation(s) of a scalable encryption algorithm. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(2):212–216, 2008.

[30] M. McLoone and J. McCanny. Generic architecture and semiconductor intellectual property cores for advanced encryption standard cryptography. *Computers and Digital Techniques, IEE Proceedings* -, 150(4):239–244, July 2003.

[31] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 2010.

[32] N. Nalla Anandakumar, T. Peyrin, and A. Poschmann. A very compact FPGA implementation of LED and PHOTON. In W. Meier and D. Mukhopadhyay, editors, *Progress in Cryptology – INDOCRYPT 2014*, Lecture Notes in Computer Science, pages 304–321. Springer International Publishing, 2014.

[33] NIST. *Cryptographic Module Validation Program Management Manual*, May 2014.

[34] C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, and L. Thiele. Reconfigurable hardware in wearable computing nodes. In *Wearable Computers, 2002.(ISWC 2002). Proceedings. Sixth International Symposium on*, pages 215–222. IEEE, 2002.

[35] J. Portilla, A. Otero, E. de la Torre, T. Riesgo, O. Stecklina, S. Peter, and P. Langendrfer. Adaptable security in wireless sensor networks by using reconfigurable ECC hardware coprocessors. *IJDSN*, 2010, 2010.

[36] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document, 2001.

[37] P. Schaumont and A. Aysu. Three design dimensions of secure embedded systems. In B. Gierlichs, S. Guilley, and D. Mukhopadhyay, editors, *Security, Privacy, and Applied Cryptography Engineering*, volume 8204 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2013.

[38] K. Sharma and M. Ghose. Cross layer security framework for wireless sensor networks. *International Journal of Security & Its Applications*, 5(1), 2011.

[39] F. X. Standaert, G. Piret, G. Rouvroy, and J. J. Quisquater. FPGA implementations of the ICEBERG block cipher. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 1, pages 556–561 Vol. 1, 2005.

[40] Q. Wang, Z. Liu, K. Varıcı, Y. Sasaki, V. Rijmen, and Y. Todo. Cryptanalysis of reduced-round simon32 and simon48. In *Progress in Cryptology–INDOCRYPT 2014*, pages 143–160. Springer, 2014.

[41] Y. Wang, G. Attebury, and B. Ramamurthy. A survey of security issues in wireless sensor networks. *Communications Surveys Tutorials, IEEE*, 8(2):2–23, Second 2006.

[42] R. S. Winternitz. A secure one-way hash function built from DES. In *2012 IEEE Symposium on Security and Privacy*, pages 88–88. IEEE Computer Society, 1984.

[43] P. Yalla and J. Kaps. Lightweight cryptography for FPGAs. In *Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on*, pages 225–230, 2009.

[44] M. Younis, N. Krajewski, and O. Farrag. Adaptive security provision for increased energy efficiency in wireless sensor networks. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, pages 999–1005, Oct 2009.