A Flexible and Compact Hardware Architecture for the SIMON Block Cipher

Ege Gulcan, Aydin Aysu, and Patrick Schaumont

Secure Embedded Systems
Center for Embedded Systems for Critical Applications
Bradley Department of ECE
Virginia Tech, Blacksburg, VA 24061, USA
{egulcan,aydinay,schaum}@vt.edu

Abstract. SIMON is a recent, light-weight block cipher developed by NSA. Previous work on SIMON shows that it is a very promising alternative of AES for resource-constrained platforms. While SIMON offers a range of block sizes and key lengths, a straightforward implementation would select fixed values in order to achieve a compact design. In contrast, we propose a flexible hardware architecture on FPGAs that still preserves the compactness of SIMON. The proposed implementation can execute all configurations of SIMON, and thus provides a versatile architecture that enables adaptive security using a variable key-size. Moreover, it also reduces the inefficiency of encrypting slightly longer messages by supporting a variable block-size. The implementation results show that the proposed architecture occupies 90 and 32 slices on Spartan-3 and Spartan-6 FPGAs, respectively. To our best knowledge, these area results are smaller than other block ciphers of similar security level. Furthermore, we also quantify the cost of flexibility and show the trade-off between the security level, throughput and area.

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

1 Introduction

Block ciphers are the building blocks of secure systems as they enable sending a message 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) [8].

Even though Rijndael serves as the AES, its area-cost restricts its use in resource-critical domains like RFID tags. 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. With the aim of reducing the area of the AES, two alternatives named PRESENT and CLEFIA were previously developed and later standardized by ISO [10]. Likewise, DARPA has an ongoing SHIELD project that is targeted towards tackling counterfeit electronics [6]. The goal of the project is to enable supply-chain management by means of a light-weight secure hardware of 100 micron $\times$ 100 micron size [7]. Therefore, there are important incentives to build the basic encryption block that are much smaller than the available ones. SIMON is such an alternative which is optimized for compact hardware implementations [2]. Aysu *et al.* showed that SIMON can break the area records of block ciphers on FPGAs [1]. They implement a fixed 128/128 configuration of SIMON that can only encrypt blocks of 128-bit messages using a 128-bit key. However, the design space of digital systems are not solely composed of fixed elements, and flexibility among others is an important design dimension.

1.1 Motivation

Security is a new design dimension for digital systems [12]. Schaumont *et al.* labels this dimension as Risk and shows that flexibility, performance and risk are the main design dimensions of secure embedded systems [18]. 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 [23]. 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 [21]. 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 [19]. 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 [17].

Cook *et al.* approaches flexibility from another perspective [5]. 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 [16] 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 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.

1.2 Organization

The rest of the paper is organized as follows. Section 2 gives a brief overview of SIMON block cipher and its configurations. Section 3 highlights the methodology behind the compact block cipher architectures and how to extend it for flexibility. Section 4 shows the implementation results and presents the trade-off between flexibility, performance and risk. Section 5 concludes the paper and comments on possible future extensions.

2 SIMON Block Cipher

SIMON is a Feistel-based lightweight block cipher recently published by NSA, targeted towards compact hardware implementations [2]. SIMON has ten configurations optimized for different block and key sizes providing a flexible level of security. Table 1 shows the parameters for all configurations of SIMON. The word size n is the bit length of each word 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.

Table 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

2.1 Round Function

Figure 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.

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 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

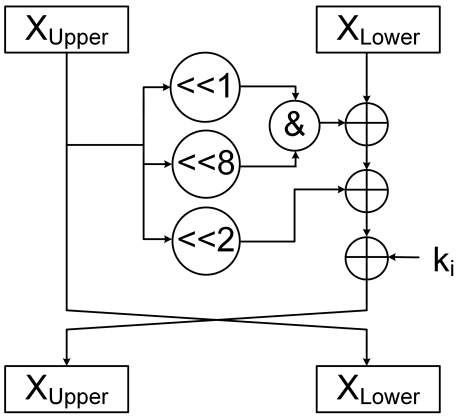

Fig. 1: SIMON Round Function

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 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.

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.

3.1 Bit-Serial

Figure 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

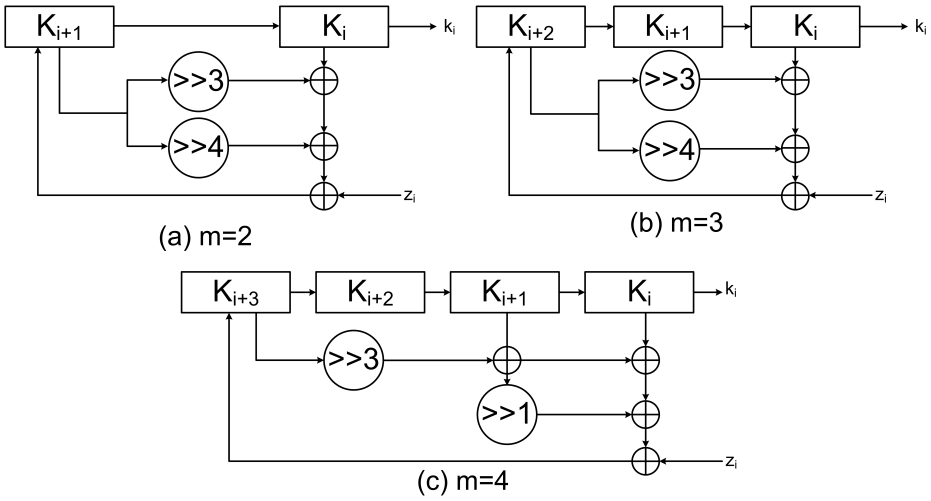

Fig. 2: SIMON Key Expansions

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.

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 paper. By using these FIFOs we can overlap the compute and transfer phases to process one bit in every clock cycle.

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 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. 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 5 shows the required FIFO numbers for all security configurations.

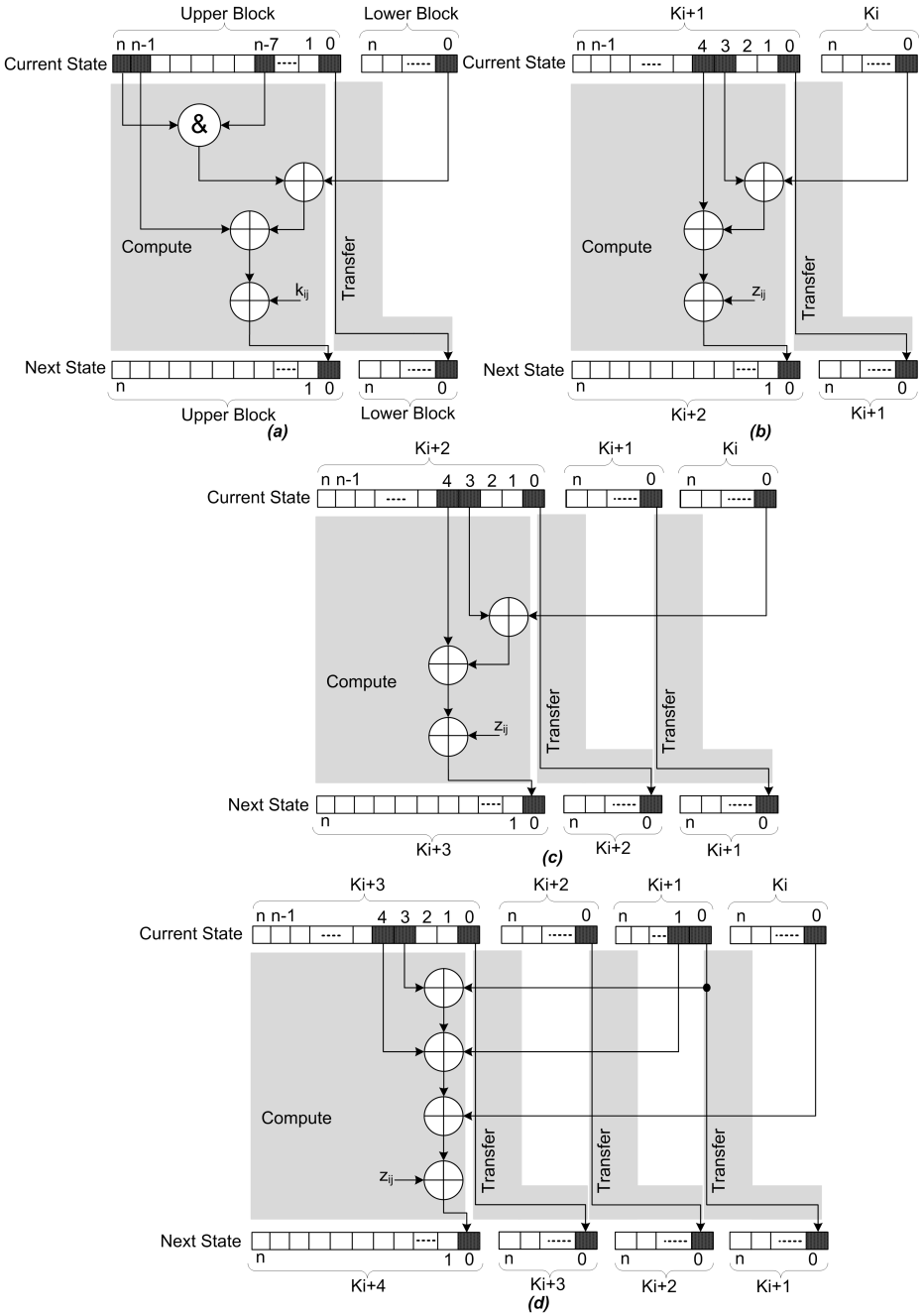

Fig. 3: (a) SIMON Bit-serial Round Function, (b) SIMON Bit-serial Key Expansion for $m = 2$, (c) SIMON Bit-serial Key Expansion for $m = 3$, (d) SIMON Bit-serial Key Expansion for $m = 4$

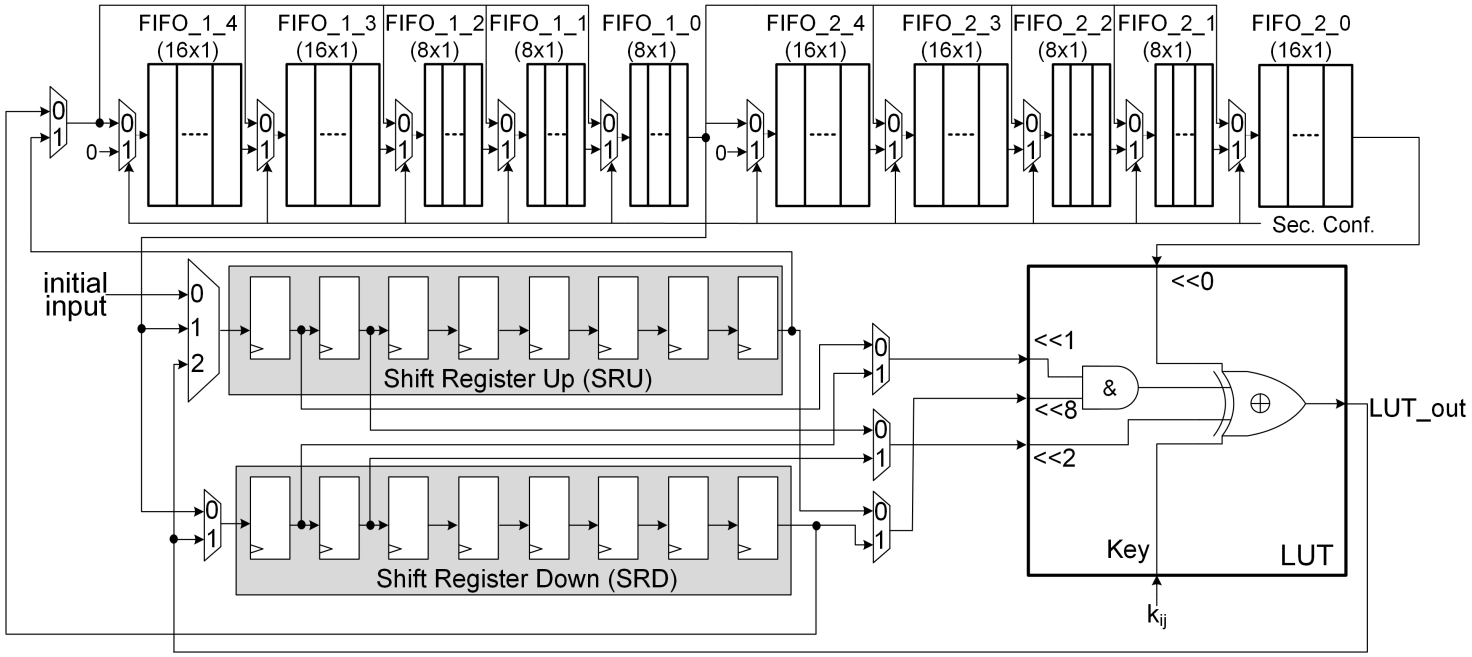

Fig. 4: SIMON Bit-serial Flexible Round Function

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

Fig. 5: FIFO Usage Schedule

For example, if the security configuration input is 1, the round function use 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 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.

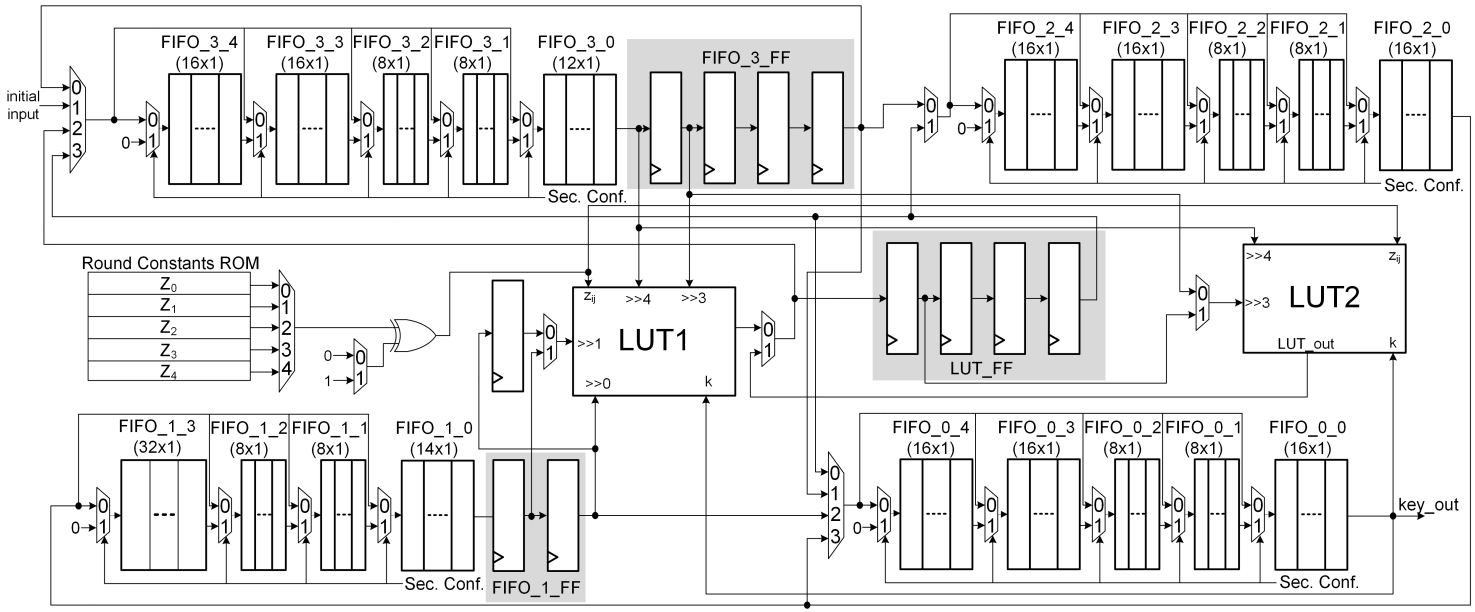

Fig. 6: SIMON Bit-Serial Flexible Key Expansion

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 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 5 gives the details of which FIFOs are used for all the security configurations.

As it can be seen in Figure 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 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

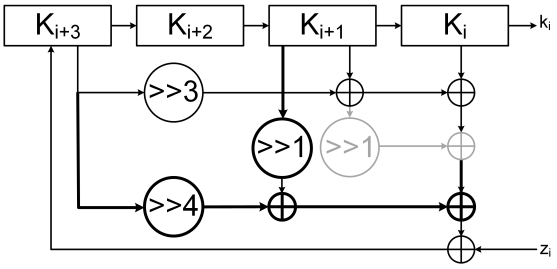

Fig. 7: SIMON Modified Key Expansion Function for $m = 4$

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 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 $m = 4$.

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.

4.1 Area

Comparison with other block ciphers Figure 8 shows hardware resource utilization of our architecture and the previous work. We have compared our work with the smallest version of AES[4], as well as alternative compact block

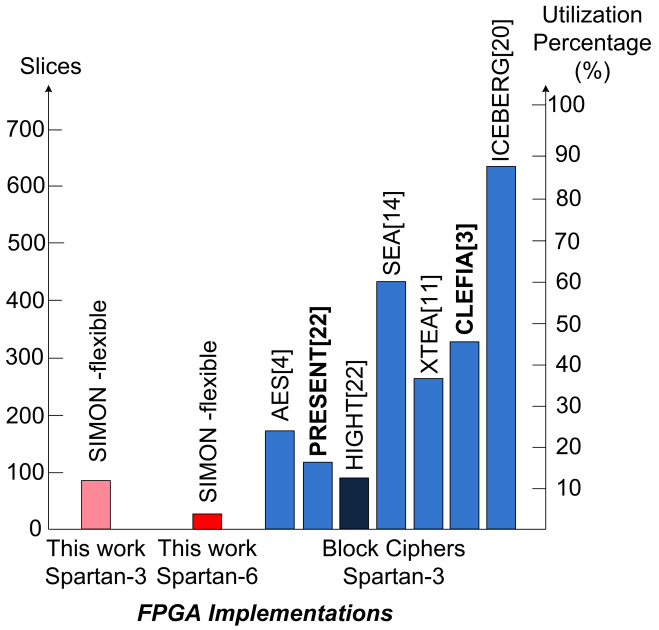

Fig. 8: Occupied slices and the resource utilization ratio of flexible SIMON vs. previous work.

cipher implementations such as PRESENT[22], HIGHT[22], SEA[14], XTEA[11], CLEFIA[3] and ICEBERG[20]. 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 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 [15]. Li *et al.* later optimizes this implementation and reduces the slice count to 3223 slices [13].

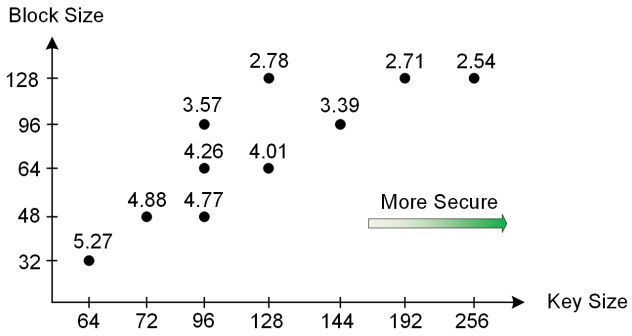

Fig. 9: Throughput (Mbps) vs. the security configuration of SIMON

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.

4.2 Performance vs. Risk Trade-off

Figure 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 16 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 a block cipher with 57-bits in less than a week [9]. 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.

4.3 Flexibility vs. Performance Trade-off

Flexibility comes at the expense of performance. Figure 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

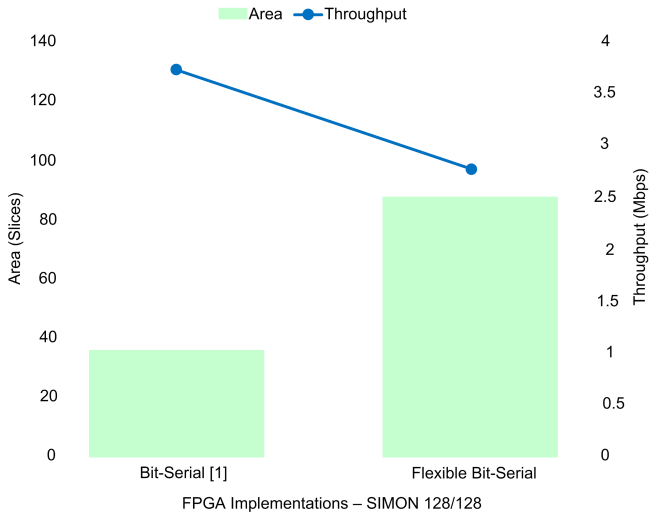

Fig. 10: The cost of flexibility on area and throughput

both use 128-bit key size and block size [1]. 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%.

5 Conclusion and Future Work

In this paper, we propose a flexible and compact architecture for the block cipher SIMON. SIMON is a very promising alternative of AES for resource-constrained platforms and we show that the bit-serialized flexible implementation of SIMON is still smaller than other block ciphers. The proposed architecture can implement all configurations of SIMON and enables on-the-fly security configuration management. Thus, we propose a light-weight, yet flexible and adaptive solution for secure systems. We also show the trade-offs that a designer can utilize regarding the flexibility, performance and risk. A further extension of this work may be proposing a complete system that can use the proposed architecture in an adaptive security protocol. Such a protocol provides different levels of security to its users based on some pre-defined criteria or may scale-up/down the risk on-the-fly, to meet the real-time performance requirements.

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

References

1. Aysu, A., Gulcan, E., Schaumont, P.: SIMON says: Break area records of block ciphers on FPGAs. Embedded Systems Letters, IEEE 6(2), 37–40 (June 2014)
2. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers (2013)
3. Chaves, R.: Compact CLEFIA implementation on FPGAs. In: Athanas, P., Pnevmatikatos, D., Sklavos, N. (eds.) Embedded Systems Design with FPGAs, pp. 225–243. Springer New York (2013), http://dx.doi.org/10.1007/978-1-4614-1362-2_10
4. Chu, J., Benaissa, M.: Low area memory-free FPGA implementation of the AES algorithm. In: Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on. pp. 623–626 (Aug 2012)
5. Cook, D.L.: Elastic block ciphers. Ph.D. thesis, Columbia University (2006)
6. DARPA: SHIELD:supply chain hardware integrity for electronics defense proposers day (Feb 2014)
7. DARPA: Tiny, cheap, foolproof: Seeking new component to counter counterfeit electronics (Feb 2014), http://www.darpa.mil/NewsEvents/Releases/2014/02/24.aspx
8. FIPS PUB 197: AES: Advanced encryption standard. Federal Information Processing Standards Publication (2001)
9. Guneysu, T., Kasper, T., Novotny, M., Paar, C., Rupp, A.: Cryptanalysis with COPACOBANA. Computers, IEEE Transactions on 57(11), 1498–1513 (Nov 2008)
10. ISO/IEC 29192-2:2012: Information technology - security techniques - lightweight cryptography - part 2: Block ciphers (2012)
11. Kaps, J.P.: CHAI-TEA, cryptographic hardware implementations of XTEA. In: Chowdhury, D., Rijmen, V., Das, A. (eds.) Progress in Cryptology - INDOCRYPT 2008, Lecture Notes in Computer Science, vol. 5365, pp. 363–375. Springer Berlin Heidelberg (2008)
12. Kocher, P., Lee, R., McGraw, G., Raghunathan, A.: Security as a new dimension in embedded system design. In: Proceedings of the 41st Annual Design Automation Conference. pp. 753–760. DAC '04, ACM, New York, NY, USA (2004), http://doi.acm.org/10.1145/996566.996771, moderator-Ravi, Srivaths
13. Li, H.: Efficient and flexible architecture for AES. Circuits, Devices and Systems, IEE Proceedings - 153(6), 533–538 (Dec 2006)
14. Mace, F., Standaert, F.X., Quisquater, J.J.: FPGA implementation(s) of a scalable encryption algorithm. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 16(2), 212–216 (2008)
15. McLoone, M., McCanny, J.: Generic architecture and semiconductor intellectual property cores for advanced encryption standard cryptography. Computers and Digital Techniques, IEE Proceedings - 150(4), 239–244 (July 2003)
16. NIST: Cryptographic Module Validation Program Management Manual (May 2014), http://csrc.nist.gov/groups/STM/cmvp/documents/CMVPMM.pdf
17. Portilla, J., Otero, A., de la Torre, E., Riesgo, T., Stecklina, O., Peter, S., Langendrfer, P.: Adaptable security in wireless sensor networks by using reconfigurable ECC hardware coprocessors. IJDSN 2010 (2010), http://dblp.uni-trier.de/db/journals/ijdsn/ijdsn2010.html#PortillaOTRSPL10
18. Schaumont, P., Aysu, A.: Three design dimensions of secure embedded systems. In: Gierlichs, B., Guilley, S., Mukhopadhyay, D. (eds.) Security, Privacy, and Applied Cryptography Engineering, Lecture Notes in Computer Science, vol.

8204, pp. 1–20. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-41224-0_1
19. Sharma, K., Ghose, M.: Cross layer security framework for wireless sensor networks. International Journal of Security & Its Applications 5(1) (2011)
20. Standaert, F.X., Piret, G., Rouvroy, G., Quisquater, J.J.: FPGA implementations of the ICEBERG block cipher. In: Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on. vol. 1, pp. 556–561 Vol. 1 (2005)
21. Wang, Y., Attebury, G., Ramamurthy, B.: A survey of security issues in wireless sensor networks. Communications Surveys Tutorials, IEEE 8(2), 2–23 (Second 2006)
22. Yalla, P., Kaps, J.: Lightweight cryptography for FPGAs. In: Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on. pp. 225–230 (2009)
23. Younis, M., Krajewski, N., Farrag, O.: Adaptive security provision for increased energy efficiency in wireless sensor networks. In: Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on. pp. 999–1005 (Oct 2009)