BitCryptor: Bit-Serialized Compact Crypto Engine on Reconfigurable Hardware

Ege Gulcan, Aydin Aysu, 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. There is a significant effort in building lightweight cryptographic operations, yet the proposed solutions are typically single-purpose modules that can implement a single functionality. In contrast, we propose BitCryptor, a multi-purpose, bit-serialized compact processor for cryptographic applications on reconfigurable hardware. The proposed crypto engine can perform pseudo-random number generation, strong collision-resistant hashing and variable-key block cipher encryption. The hardware architecture utilizes SIMON, a recent lightweight block cipher, as its core. The complete engine uses a bit-serial design methodology to minimize the area. Implementation results on the Xilinx Spartan-3 s50 FPGA show that the proposed architecture occupies 95 slices (187 LUTs, 102 registers), which is 10× smaller than the nearest comparable multi-purpose design. BitCryptor is also smaller than the majority of recently proposed lightweight single-purpose designs. Therefore, it is a very efficient cryptographic IP block for resource-constrained domains, providing a good performance at a minimal area overhead.

Keywords: Lightweight cryptography, bit-serialization, hardware architecture, crypto engine, SIMON, FPGA

1 Introduction

Lightweight cryptography studies the challenges of enabling security services on resource-constrained platforms. Typical applications on such devices require a protocol execution for secure key exchange or entity authentication. For example, the protocol with non-reversible functions (section 6.1.5. of [12]) uses a PRNG, hash function and encryption, all within a single protocol run. Yet, most compact implementations in the literature are single-purpose and stand alone building blocks that can perform only one of these three operations. How should a designer combine a multi-purpose requirement with an area resource-constraint? Clearly, a solution that uses disjoint kernels (like PRESENT [7] for encryption, [19] for hashing, and TRIVIUM [13] for PRNG) yields a design that is larger than the sum of its composing kernels. It also ignores the opportunity to share the

internal designs for each kernel. Another solution would be to use software on a microcontroller. But such a solution is not ideal either, because the instruction-set of the microcontroller is generic, and not optimized for the multi-purpose kernel which we have in mind. Therefore, we will evaluate a third option: the design of a flexible yet specialized crypto-engine.

In this paper, we propose BitCryptor, a bit-serialized compact crypto engine that can execute fundamental cryptographic operations. BitCryptor is a multi-purpose design that can perform PRNG, encryption and hash operations. Therefore, we are promoting BitCryptor as a generic lightweight crypto engine upon which protocols can be built as a sequence of BitCryptor commands. We show that the BitCryptor is significantly smaller than competing crypto engines and it has a better performance than low-cost microcontrollers.

1.1 Compact and Efficient Crypto Engine on FPGAs

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) [29], wearable computers (WC) [30], and Internet-of-Things (IoT) [25] are amongst such application domains that require compact solutions and still incorporate FPGAs. In addition to their primary functionality, secure systems in FPGAs need a method to perform cryptographic operations. Thus, the resource-constrained device should embody this method with low operational and area costs.

We are not the first to propose a multi-purpose design in FGPA, but our proposal is the smallest so far. BitCryptor occupies 95 slices, 12% of available resources of a Spartan-3 s50 FPGA whereas the nearest competitor with similar functionalities [24] occupies 916 slices and cannot even fit into the same device. Hence, a system using [24] must migrate to a larger device (eg. Spartan-3 s200), effectively increasing the component cost. A larger device also increases the system cost, as it increases static power dissipation, and possibly PCB cost. So, the argument that it is always possible to use a larger FPGA, and thus that FPGA area optimization has little value, is not correct in the context of IoT, WSN, and so on.

One can argue the use of embedded microcontrollers for low-cost reconfigurable systems. However, these platforms are at a disadvantage in terms of operational costs: A recent work [14] shows that, compared to BitCryptor, encryption on a 16-bit MSP430 microcontoller needs 4.8× more clock cycles, 70.8× execution time, and 15.2× energy[1]. Alternatively, to achieve a higher operating frequency, the same general purpose MSP430 microcontroller can also be configured as a soft-core processor on FPGAs. However, this trivial approach is problematic as the resulting hardware occupies a very large area, requiring the system to again move to an expensive board. Therefore, a designer has to find

[1] The results section elaborates on comparisons

the delicate balance between the area-cost, performance, and flexibility. BitCryptor is such a solution that offers multiple cryptographic operations at minimal area-cost and performance hit on reconfigurable hardware.

1.2 Novelty

Achieving the combination of area resource constraints with multi-purpose functionality requires sound cryptographic engineering. It requires picking a lightweight crypto kernel, applying specific functionalities with a careful analysis of modes-of-operations, selecting proper configuration parameters, employing an appropriate design methodology, and back-end engineering for EDA tool optimizations. In this paper, we guide through these steps to reveal how to realize a compact and multi-purpose crypto-engine on FPGA. We also provide detailed analysis on the trade-offs within the design space.

The major contributions of this work are as follows

- We demonstrate a multi-purpose design that is 10× smaller than the nearest comparable crypto-engine [24] and even smaller than the majority of single-purpose encryption and all hash function implementations.
- We develop a systematic design approach with optimizations at several abstraction levels.
- We show area-performance trade-offs between different serialization methods and on multiple platforms.
- We present a comparison with low-cost and moderate microcontroller designs and quantify the performance improvement of our solutions.
- We provide a small isolated security module that is easier to validate and certify.

1.3 Organization

The rest of the paper is organized as follows. Section 2 explains SIMON, the lightweight core of the crypto engine, and discusses high-level design parameters. Section 3 illustrates the bit-serial design methodology with a simple example. Section 4 describes the hardware architecture of BitCryptor. Section 5 shows the implementation results and its comparison to previous work and Section 6 concludes the paper.

2 High-Level Description of BitCryptor

Table 1 summarizes the design of BitCryptor. The heart of BitCryptor is a flexible block cipher, SIMON [6]. The flexibility of SIMON allows multiple key and block lengths. The choice of security-level (96-bits, corresponding to ECRYPT-II Level 5 or 'legacy standard-level' [32]) is a trade-off between selecting the shortest key length possible while offering reasonable security for the intended application domains. Using SIMON as the kernel, we then configure different

Table 1: BitCryptor Construction

Operation	Kernel and Configuration	Modes-Of-Operation	Security-level
Encryption	SIMON 96/96	ECB, CBC	96-bits
Hash function	SIMON 96/144	Hirose[20]	96-bits[1]
PRNG	SIMON 96/96	CTR	96-bits

[1] SIMON 96/144 generates a digest of 192-bits which has 96-bits of strong collision resistance.

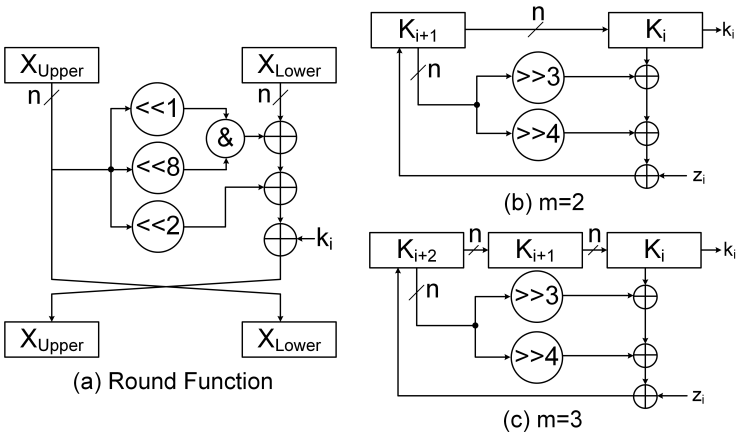

Fig. 1: (a) SIMON Round Function, (b) SIMON Key Expansion Function for m=2, (c) SIMON Key Expansion Function for m=3

mode-of-operations to achieve message confidentiality (encryption), message integrity (hashing), and pseudo-random number generation. Each row in Table 1 describes such a mode-of-operation. In all of these configurations, we maintain the selected 96-bit security level.

2.1 SIMON Block Cipher

The lightweight block cipher SIMON is developed by NSA, targeting compact hardware implementations [6]. So far, conventional cryptanalytic techniques against SIMON did not demonstrate any weaknesses [1], [3], [34]. Equations 1 and 2 formulate the SIMON round and key expansion functions respectively, and Figure 1 illustrates them. 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. Block size indicates the bit length of the input message to the block cipher while the key size is 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

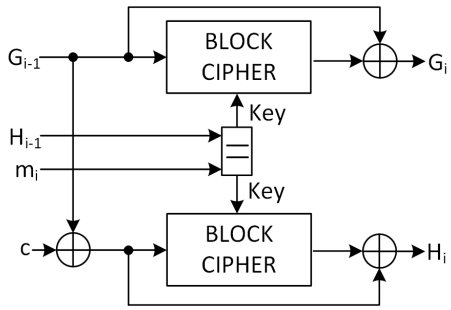

Fig. 2: Hirose Double-Block-Length Hash Function

size (mn). SIMON using a block size $2n$ and key size mn is denoted as SIMON $2n/mn$.

$$R(X_u, X_l) = (X_u \ll 1) \wedge (X_u \ll 8) \oplus (X_u \ll 2) \oplus X_l \oplus k_i \tag{1}$$

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

2.2 Parameter Selection

The parameters we select directly affect the area and performance 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. 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 [27] 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 [35] or double-block-length (DBL) constructions like Hirose [20]. 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 2 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.

$$H_i = E(G_{i-1} \oplus c, m_i||H_{i-1}) \oplus G_{i-1} \oplus c \tag{3}$$

$$G_i = E(G_{i-1}, m_i||H_{i-1}) \oplus G_{i-1} \tag{4}$$

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 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 resulting hash function reads messages in 48-bit 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, the performance of the system improves if we use SIMON 96/96 which has the same block size but a shorter key. In [18], Gulcan *et al.* 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 changes while the datapath remains exactly 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 [15] 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.

3 Design Methodology

The way to systematically reduce the area of a circuit is through sequentialization; dividing operations in time and reusing the same resources for similar computations. In our design, we have applied bit-serialization [4], a sequentialization methodology that processes one output bit at a time. We have adapted and applied this methodology with an architecture optimization using shift register logic (SRL-16) for the target FPGA technology.

3.1 Datapath

Figure 3 illustrates an example where the datapath computes $c = a \oplus b$ by XORing two 16-bit registers a and b, and generates the 16-bit output c. In this

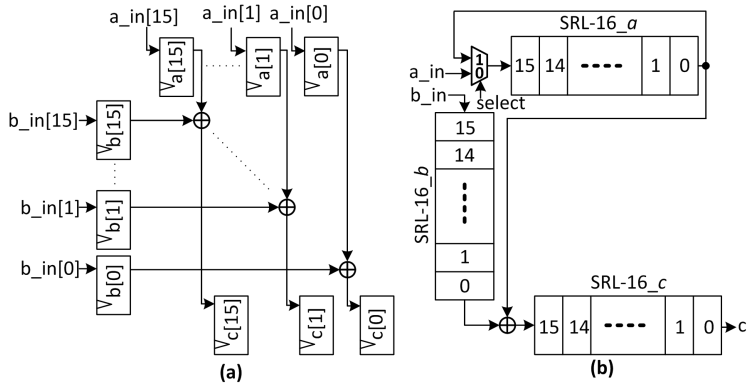

Fig. 3: (a)Bit-Parallel Datapath (b) Bit-Serial Datapath

example, the datapath uses the same value of a multiple times while the value of b changes. If all the bits are processed in the same clock cycle (Figure 3(a)), the datapath produces all bits of c in parallel. This datapath utilizes 48 registers (to store a, b, and c) and 16 LUTs (to compute 16 XOR operations of $c = a \oplus b$). We can map these elements to 24 slices.

If we bit-serialize the entire datapath (Figure 3(b)), the resulting hardware will produce one output bit in one clock cycle. The 16-bit register blocks can now be mapped to SRL-16 logic and the output of a and b can be XORed using a single XOR gate. To keep the value of a, SRL-16_a should have a feedback from its output to input. Thus, the resulting hardware architecture will consist of 5 LUTs (3 SRL-16 to store a,b, and c, 1 LUT to compute the XOR operation and 1 LUT to apply the feedback via a multiplexer). Now, the datapath can be mapped to a total of 3 slices, which is one-eight of the size of the bit-parallel implementation.

3.2 Control

Bit-serialization comes with control overhead. If not dealt carefully, this can counteract the area gain of the datapath. In bit-serial designs, to identify when to start and end loading shift registers, and when to finish operations, we need to keep track of the bit positions during computations. In the example, since the value of a is fixed for a number of $c = a \oplus b$ executions, the control needs to determine the value of the select signal at the input multiplexer of SRL-16_a. It will select 0 while a_in is loaded, otherwise it will select 1. Usually, this is implemented with counters and comparators. Figure 4 (a) shows a 4-bit counter with a corresponding comparator. In each clock cycle, the counter value increments by one and four registers update their values in parallel. A comparator checks the counter value and returns 1 when the check condition occurs. This architecture consists of 5 LUTs (4 LUTs for counter and 1 LUT for comparator) and 4 registers.

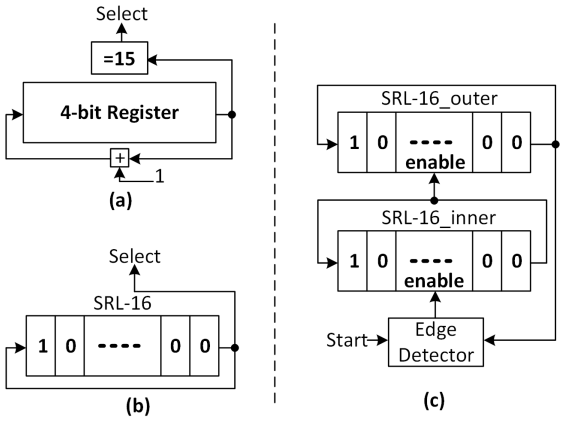

Fig. 4: (a) Control with Up-Counters (b) Control with Ring Counters (c) Control of Nested Loops

Instead of using an up-counter, the same functionality can be realized using a ring counter. Ring counters consist of circular shift registers. Figure 4 (b) shows a 16-bit ring counter. After 16 clock cycles, the output of this counter will return 1 indicating that 16 cycles have passed. The control unit can use a single LUT (SRL-16) to implement the ring counter which is less than one-fifth of a counter-based control mechanism. If the control signal has to remain 1 after 16 clock cycles, the controller can use an edge detector which costs an extra LUT and register, to check when a transition from 1 to 0 occurs.

Managing the hierarchy of control is also simpler using ring counters and edge detectors. Consider an example with two nested loops both counting up to 16. Figure 4 (c) shows the implementation of this nested loop with two ring counters and an edge detector. The outer (SRL-16$_outer$) loop may count the number of rounds while the inner (SRL-16$_inner$) loop counts the number of bits. The *Edge Detector* will convert the *start* pulse into a continuous *enable* signal which will keep SRL-16$_inner$ active until a positive edge is detected at the output of SRL-16$_outer$. Once the SRL-16$_inner$ is active, its output will be 1 every 16 clock cycles and enable the SRL-16$_outer$ for a single clock cycle. This control unit can be realized with 4 LUTs and 3 register (2 LUTs for SRL-16, 2 LUT and 3 register for the Edge Detector).

3.3 Bit-serializing BitCryptor

The datapath of BitCryptor is serialized similar to the example. The bit-parallel operations are converted into bit-serial ones and the necessary data elements are stored in SRL-16. The sequentialization of the control flow is achieved by using ring counters and edge detectors. The ring counters control the internal signals when there is a data transmission with the host system. The I/O structure of

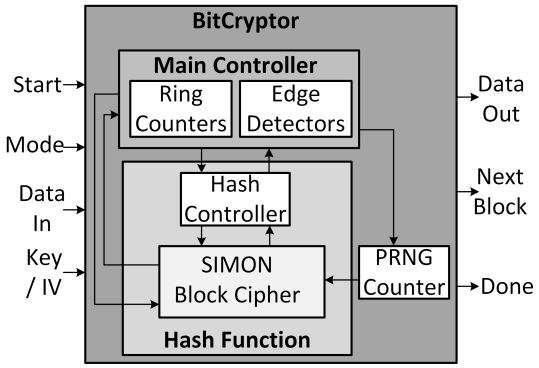

Fig. 5: Block Diagram of the BitCryptor

BitCryptor is also simplified using bit-serial design methodology. The data input and output of the BitCryptor are single bit ports which makes it very suitable for standard serial communication interfaces.

4 Hardware Implementation

Figure 5 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 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 following the methodology in Section 3.2. 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.

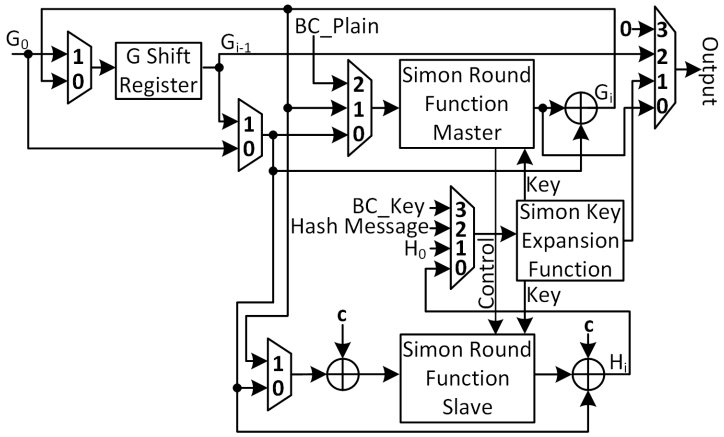

Fig. 6: Hardware Architecture of the Double-Datapath SIMON and the Hirose Construction

4.1 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 have 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 6 shows the DDP SIMON architecture following a master/slave configuration. The architecture is bit-serialized using the design methodology of section 3. 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.

4.2 Symmetric Key Encryption

At the core, the crypto engine uses the SIMON block cipher with a 96-bit block and key size. In [5], Aysu *et al.* implement the bit-serial version of SIMON 128/128 and show that it is an extremely compact design. To adapt the bit-serial SIMON block cipher to our crypto engine, we modify the implementation

in [5] 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 6 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.

4.3 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 [31].

5 Results

In this section, we first focus on BitCryptor, the lightweight bit-serialized implementation. Then, we show the trade-off between the area and performance on a round-serial variant of BitCryptor.

5.1 Smallest Area – BitCryptor

The proposed hardware architecture is written in Verilog HDL and synthesized in Xilinx ISE 14.7 for the target Spartan-3 XC3S50-4 FPGA as well as a more recent Spartan-6 XC3S50-4 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 7 shows a detailed area comparison of BitCryptor with the smallest previous multi-purpose engine [24] and with various standalone area-optimized block ciphers [5, 10, 11, 22, 26, 28, 33, 36], hash functions [2, 23, 28], and PRNGs [21]. The results show the effect of sound cryptographic engineering. Next, we discuss the details of area comparisons and the performance tradeoff.

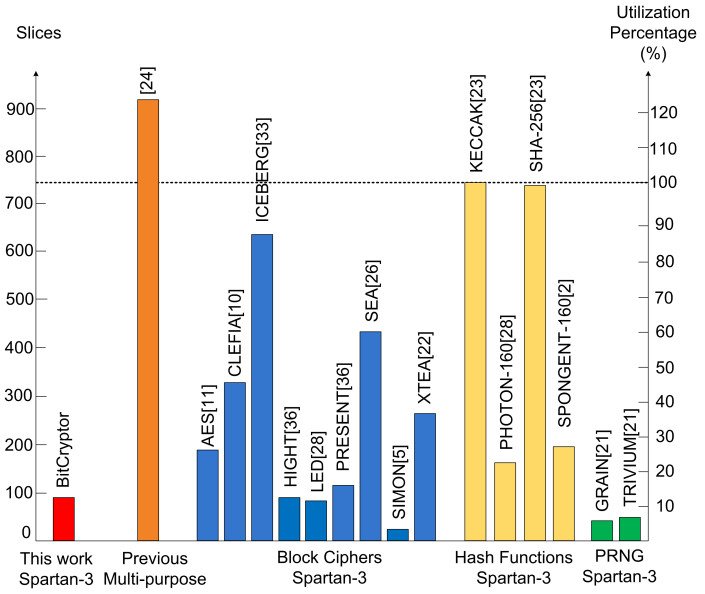

Fig. 7: Implementation Results and Comparison with Previous Work. For comparison fairness with the previous work, we map our architecture on an older Spartan-3 FPGA but we also provide the result on a recent Spartan-6 FPGA.

Migrating to more recent Xilinx FPGAs For comparison fairness with the previous work, we implement our hardware architecture on a Spartan-3 family FPGA (XC3S50-4TQG144C). In addition, we also map our design on a more recent Spartan-6 device. On a Spartan-6 XC6SLX4-2 FPGA, BitCryptor occupies 5% of available resources which corresponds to only 35 slices (136 LUTs, 103 Registers) with a maximum frequency of 172 MHz.

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.

Comparison with other multi-purpose designs Previous multi-purpose designs on FPGAs are optimized primarily for performance and are not suitable for lightweight applications. Bossuet *et al.* survey a number of multi-purpose designs and document the smallest to be 847 slices [8]. In [24], Laue *et al.* propose a hardware engine that offers the closest functionality to our design. However, they do not apply our design and optimization methods. The resulting hardware design is targeted towards high-end applications. It has a throughput of 357.4 Mbps and requires 916 slices on a Virtex-II family FPGA (which has the same

Table 2: Comparison of Encryption Performance with Low-Cost Microcontrollers

Platform	Clock cycles	Max. Frequency (MHz)	Throughput (Kbps)
ATmega128 [14]	24369	16	82.07
MSP430F1611 [14]	12902	8	77.50
This work (bit-serial) XC3S50-4TQG144C	2685	118	4120
This work (bit-serial) XC6SLX4-2TQG144C	2685	172	6005

slice structure with Spartan-3). Compared to this design, our architecture has an area improvement of almost 10×.

Comparison with soft-core and embedded processors We also compare our results with the software implementations on actual microcontrollers and on FPGAs using soft-core processors. Good *et al.* provide the smallest soft-core processor in the literature that is capable of running only a single-purpose AES encryption [17]. This design utilizes the 8-bit PicoBlaze processor [9], achieves 0.71 Mbps, and occupies 119 slices and a BRAM (≈ 452 slice equivalent), making it larger and slower than BitCryptor. Likewise, the 16-bit MSP430 softcore processor [16] on FPGAs occupies more than 10× of BitCryptor and it can not even fit into the same device.

Table 2 shows the comparison of a SIMON block cipher encryption on FPGAs vs. low-cost 8-bit and 16-bit microcontrollers. BitCryptor is two orders of magnitude better than ATmega128 and MSP430 based microcontroller implementations. Note that the previous work [14] uses a fixed-key implementation that requires fewer operations and we provide throughput results to compensate for different SIMON configurations. Unfortunately, the power and energy results of Dinu *et al.* is not available, but we can make a rough estimation on TI MSP430F1611. The typical energy consumption of this microcontroller at an energy optimized configuration of 2.2 V and 1 MHz is 330μJ. A SIMON execution with this setting takes 1.3 ms and consumes 4.26×10^{-6} J of energy which is 15.2× of our bit-serial compact design. Table 3 shows the details of the performance figures.

5.2 Relaxing Area – Round-Serial Variant

Area-Performance Tradeoff A bit-serial design exchanges performance for area savings. We have evaluated the relative impact of this trade-off, by comparing a bit-serial implementation of BitCryptor with a round-serial version of BitCryptor. 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 quantifies these trade-offs. The round-serial design is approximately two orders of magnitude faster and more

Table 3: Area-Performance Tradeoff (@100 MHz XC3S50-4)

	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	$7.2.579 \times 10^{-12}$	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

Table 4: Comparison of Encryption Performance with Moderate Microcontrollers

Platform	Clock cycles	Max. Frequency (MHz)	Throughput (Mbps)
ATSAM3A8 ARM-CORTEX-M3[14]	1406	84	7.29
This work (round-serial) XC3S50-4TQG144C	54	112	189.88
This work (round-serial) XC3S50-2TQG144C	54	162	274.66

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).

The round-serial variant of BitCryptor is still smaller than previous multi-purpose implementations and can also fit into the same Spartan-3 and Spartan-6 FPGA with the bit-serial design. Table 4 shows that this architecture can achieve a two orders of magnitude performance improvement compared to a capable 32-bit ARM microcontroller.

6 Conclusion

BitCryptor is a multi-purpose engine that supports a variety of cryptographic operations with minimal area overhead. We showed that selecting the optimum encryption kernel and parameters, using a bit-serial design methodology, targeting the architecture optimization for the shift register logic (SRL-16), and manual placement of LUTs and registers can significantly minimize the area. The resulting hardware architecture is 10× smaller than a previous multi-purpose design and smaller than majority of single-purpose crypto modules. BitCryptor can fit into the smallest FPGA in Spartan-3 and Spartan-6 family with only

12% and 5% resource utilization respectively, leaving a large amount of logic for other embedded functionalities. Hence, the proposed hardware architecture is a promising IP block for system designers who seek compact and efficient solutions on reconfigurable hardware.

Acknowledgements This project was supported in part by the National Science Foundation grant no 1115839 and 1314598.

References

1. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential and linear cryptanalysis of reduced-round SIMON. Cryptology ePrint Archive, Report 2013/526 (2013), http://eprint.iacr.org/
2. Adas, M.: On the FPGA based implementation of SPONGENT (2011)
3. Alkhzaimi, H.A., Lauridsen, M.M.: Cryptanalysis of the SIMON family of block ciphers. Cryptology ePrint Archive, Report 2013/543 (2013)
4. Andraka, R.J.: Building a high performance bit-serial processor in an FPGA. In: Proceedings of Design SuperCon. vol. 96, pp. 1–5 (1996)
5. 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)
6. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013), http://eprint.iacr.org/
7. Bogdanov, A., Knudsen, L., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, Lecture Notes in Computer Science, vol. 4727, pp. 450–466. Springer Berlin Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74735-2_31
8. Bossuet, L., Grand, M., Gaspar, L., Fischer, V., Gogniat, G.: 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), http://doi.acm.org/10.1145/2501654.2501655
9. Chapman, K.: Picoblaze 8-bit microcontroller for virtex-e and spartan-ii/iie devices. Xilinx Application Notes (2003)
10. 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
11. 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)
12. Clark, J.A., Jacob, J.L.: A survey of authentication protocol literature: Version 1.0 (1997)
13. De Cannire, C.: TRIVIUM: A stream cipher construction inspired by block cipher design principles. In: Katsikas, S., Lpez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) Information Security, Lecture Notes in Computer Science, vol. 4176, pp. 171–186. Springer Berlin Heidelberg (2006), http://dx.doi.org/10.1007/11836810_13

14. Dinu, D., Corre, Y.L., Khovratovich, D., Perrin, L., Groschdl, J., Biryukov, A.: Triathlon of lightweight block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/209 (2015), http://eprint.iacr.org/
15. Ferguson, N., Schneier, B.: Practical cryptography. Wiley (2003), http://books.google.com/books?id=7SiKtxPrrRMC
16. Girard, O.: openmsp430 (2009)
17. Good, T., Benaissa, M.: AES on FPGA from the fastest to the smallest. In: Rao, J., Sunar, B. (eds.) Cryptographic Hardware and Embedded Systems CHES 2005, Lecture Notes in Computer Science, vol. 3659, pp. 427–440. Springer Berlin Heidelberg (2005)
18. Gulcan, E., Aysu, A., Schaumont, P.: A flexible and compact hardware architecture for the SIMON block cipher. In: Eisenbarth, T., Ozturk, E. (eds.) Third International Workshop on Lightweight Cryptography for Security and Privacy - LightSEC 2014. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2014)
19. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011, Lecture Notes in Computer Science, vol. 6841, pp. 222–239. Springer Berlin Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-22792-9_13
20. Hirose, S.: Some plausible constructions of double-block-length hash functions. In: Fast Software Encryption. pp. 210–225. Springer (2006)
21. Hwang, D., Chaney, M., Karanam, S., Ton, N., Gaj, K.: Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. In: State of the Art of Stream Ciphers Workshop, SASC 2008, Lausanne, Switzerland. pp. 151–162 (Feb 2008)
22. Kaps, J.: 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)
23. Kaps, J., Yalla, P., Surapathi, K.K., Habib, B., Vadlamudi, S., Gurung, S.: Lightweight implementations of SHA-3 finalists on FPGAs. In: The Third SHA-3 Candidate Conference (2012)
24. Laue, R., Kelm, O., Schipp, S., Shoufan, A., Huss, S.: 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. pp. 480–484 (Aug 2007)
25. Liu, S., Xiang, L., Xu, J., Li, X.: Intelligent engine room IoT system based on multi-processors. Microelectronics & Computer 9, 049 (2011)
26. 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)
27. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC press (2010)
28. Nalla Anandakumar, N., Peyrin, T., Poschmann, A.: A very compact FPGA implementation of LED and PHOTON. In: Meier, W., Mukhopadhyay, D. (eds.) Progress in Cryptology – INDOCRYPT 2014, pp. 304–321. Lecture Notes in Computer Science, Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-13039-2_18
29. De la Piedra, A., Braeken, A., Touhafi, A.: Sensor systems based on FPGAs and their applications: a survey. Sensors 12(9), 12235–12264 (2012)

30. Plessl, C., Enzler, R., Walder, H., Beutel, J., Platzner, M., Thiele, L.: Reconfigurable hardware in wearable computing nodes. In: Wearable Computers, 2002.(ISWC 2002). Proceedings. Sixth International Symposium on. pp. 215–222. IEEE (2002)
31. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Tech. rep., DTIC Document (2001)
32. Smart, N., Babbage, S., Catalano, D., Cid, C., Weger, B.d., Dunkelman, O., Ward, M.: ECRYPT II yearly report on algorithms and keysizes (2011-2012). European Network of Excellence in Cryptology (ECRYPT II), Sept (2012)
33. 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)
34. Wang, Q., Liu, Z., Varıcı, K., Sasaki, Y., Rijmen, V., Todo, Y.: Cryptanalysis of reduced-round SIMON32 and SIMON48. In: Progress in Cryptology–INDOCRYPT 2014, pp. 143–160. Springer (2014)
35. Winternitz, R.S.: A secure one-way hash function built from DES. In: 2012 IEEE Symposium on Security and Privacy. pp. 88–88. IEEE Computer Society (1984)
36. Yalla, P., Kaps, J.: Lightweight cryptography for FPGAs. In: Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on. pp. 225–230 (2009)