FPGA-based Assessment of Midori and GIFT Lightweight Block Ciphers ⋆ ⋆⋆

Carlos Andres Lara-Nino (✉)[1], Arturo Diaz-Perez[2], and Miguel Morales-Sandoval[1]

[1] CINVESTAV Tamaulipas. Victoria City, México
clara@tamps.cinvestav.mx.
[2] CINVESTAV Guadalajara. Zapopan, México.

Abstract. Lightweight block ciphers are today of paramount importance to provide security services in constrained environments. Recent studies have questioned the security properties of PRESENT, which makes it evident the need to study alternative ciphers. In this work we provide hardware architectures for Midori and GIFT, and compare them against implementations for PRESENT and GIMLI under fair conditions. The hardware description for our designs is made publicly available.

Keywords: Midori · GIFT · PRESENT · GIMLI · Lightweight Cryptography · Lightweight Block Ciphers · FPGA.

1 Introduction

A decade has elapsed since the term "lightweight" was first used to describe a block cipher. Back in 2007, the discourse of efficient implementation of cryptographic architectures was centered around AES. However, a new proposal would set the benchmark for encryption in constrained applications. The PRESENT block cipher [2] promised to fill a niche "for extremely constrained environments such as RFID tags and sensor networks" that AES couldn't. In the following years several lightweight cryptographic algorithms, design strategies, and architectures were proposed with similar goals. These developments contributed to consolidate the research line denominated *lightweight cryptography* [3]. Nowadays, PRESENT can be considered as the most widespread lightweight block cipher.

The idea of lightweightness was so enticing that a new branch of cryptography was set to study these primitives (*lightweight cryptography*) and many such proposals have been published since then. However, even after ten years from the first contact and five years after the first standardization efforts [4] several questions remain without a definitive answer: what does lightweight mean? why use a lightweight block cipher? what is the best lightweight block cipher?

Over the past ten years, different lightweight block ciphers have been proposed. As new design goals became necessary, the concept of lightweight gradually evolved. Initially, in what we call the *first generation* of lightweight primitives, the lightweight property was associated with small implementation size. PRESENT is an example of such proposals, being mainly optimized for "security and hardware efficiency." The idea that low area is synonymous of lightweightness was posteriorly contested [5]. These new ideas promoted that "lightweight" also include low processing latencies. Works that consider area and processing latency as key aspects to achieve lightweightness can be denominated of *second generation*. Later on, power efficiency and low energy consumption requirements became the central focus for designers, thus leading to what we consider the *third generation* of lightweight primitives. Midori [6] is a block cipher from the third era. In recent years, the security of lightweight algorithms has regained interest. This has started what might be considered a new wave of lightweight cryptographic algorithms.

⋆ This is an extended version of a paper appearing in ICICS 2018, please also cite as [1]. The final authenticated publication is available online at https://doi.org/10.1007/978-3-030-01950-1_45.

⋆⋆ This work was supported by CONACyT under grant number 336750 and CINVESTAV. This work was also funded by "Fondo Sectorial de Investigación para la Educación", CONACyT Mexico, through the project number 281565.

Proposals like GIFT [7] and GIMLI [8] exemplify this trend. It is in this way that hardware usage, latency, power/energy, and security became key aspects for any lightweight cryptographic design. Nowadays the designers must be aware of the application constraints in each of these fields in order to conceive a lightweight security solution.

In 2015, a possible NIST standard for lightweight cryptography was first mentioned. Over the course of two years NIST published a report detailing the scope and state of the art in lightweight cryptography [9] and the standardization works seem to be in progress. This hints to the fact that lightweight cryptographic primitives are key components in the development of future technologies and applications. Generating solid and reproducible implementation results and benchmarking is undoubtedly primordial for any future standards.

For years, the lightweight block cipher PRESENT [2] has been in the spotlight as the most ideal solution for providing confidentiality under constrained environments. However, recent findings [10–13] call into question the security properties of the scheme. It is clear that the study of alternatives which offer resilience against birthday attacks and linear or differential cryptanalysis is necessary.

In this work we evaluate hardware realizations of the cryptographic algorithms Midori and GIFT, which are believed to be secure. We compare these implementations against State of the Art architectures for GIMLI and PRESENT. Our main contributions are:

1. Novel architectural designs for the Midori and GIFT block ciphers following area-reduction strategies.
2. The benchmark of the proposed architectures against the most relevant results from the literature under fair conditions.
3. The first implementation results for GIFT and the first area-oriented results for Midori in FPGA.
4. All the proposed designs (VHDL) are available at https://www.tamps.cinvestav.mx/~hardware/

The rest of the paper is structured as follows. In Section 2 we discuss the implications of novel attack strategies on the lightweight ciphers under study. Section 3 describes the different architectures for the selected lightweight algorithms, which are implemented and evaluated. Section 4 describes our experimental setup. Section 5 presents our findings. Section 6 concludes this work.

2 On the security of Lightweight Block Ciphers

2.1 The Birthday Bound

For quite some time the research community has been aware of the possible attacks on encryption systems which use states of 64-bits or smaller [14]. As explained in [11], whereas block ciphers with keys of k-bits and states of n-bits are expected to resist attacks with spatial and temporal complexities up to $O(2^n)$ and $O(2^k)$, respectively, most operation modes offer security under the assumption that no more than $2^{n/2}$ blocks are encrypted under a single key. This upper limit for the operation modes is called the *birthday bound* [14] and affects all the basic operation modes: CBC, CFB, OFB, CTR, OCB, GCM.

With this bound in mind, a 128-bits block cipher like AES can, under any of the aforementioned encryption modes, process "safely" up to 2^{64} blocks or 2^{68} bytes. According to Cisco [15], "in 2016, the annual run rate for global IP traffic was 1.2 ZB per year" that is approximately 2^{70} bytes. These figures evidence that the scenario where an operation mode outlives its use is not out of the realm of reality. For the 64-bits ciphers the threshold is even lower, allowing only 2^{32} "safe" blocks or 2^{36} bytes; that is 32 GB. The impact of this attack model on 64-bit block ciphers was first identified in [14].

For many years this attack was considered as impractical since it "requires known plaintext, and reveals only a few bits of a large datastream" [11]. Nonetheless, the authors in [11] managed to apply it successfully to Blowfish in OpenVPN and 3DES in TLS; in both cases to secure HTTP and using the CBC operation mode. They employed the Beastly attack to generate the required traffic in each case. These demonstrations required 785GB data each and took 19 and 38 hours, respectively. A success probability of 0.39 for $n/2$ blocks is reported. This case set a precedent in regards to the feasibility of theoretical attacks empowered with the availability of big data.

Although on first sight it might looks like the use of block ciphers with states of 64-bits or smaller should be avoided, there are countermeasures that can be taken. The following are identified in [11]:

- Stop using legacy 64-bit block ciphers.
- Implement rekeying strategies that swap the key well before the $n/2$ blocks.
- Use operation modes that are not vulnerable to the birthday bound. The CTR mode is unaffected if the underlying block cipher behaves like a pseudo-random function (PRF); and if the underlying cipher is not a PRF, the attack has the same cost but the attacker must perform significant additional work to recover any plaintext.

Both GIFT and Midori offer variants with state size of 64 and 128 bits. Whereas the latter can be considered secure, the former should be used under the appropriate operation mode or with the adequate rekeying strategies.

2.2 Distinguisher Attacks

In 2015, Blondeau et al. proposed an attack for PRESENT under the known-key model [10]. This attack is unhindered by the key length and thus can be applied to PRESENT-80 and PRESENT-128. The authors made two significant contributions in that work: they first extended the differential cryptanalysis of the cipher to reach the linear analysis in terms of attacked rounds, then introduced a Meet-in-the-Middle layer (MtM) to propagate the differential properties for the known-key distinguisher. This attack requires to store 48GB and took a time corresponding to 2^{56} encryptions with a success probability of 0.51 for the full PRESENT.

While authors of [10] concede that their attack is "very generous with the attacker", they provide as application example the case where PRESENT is used to construct hash functions.

Blondeau's work was extended in [12] where the authors generalized the attack method for a kind of permutations that share similar characteristics with PRESENT's. They used as case study attacking the hash function SPONGENT. The main breakthrough on that work is that they extended the MtM layers on a 64-bit state to states with any size. The so called PRESENT-like permutations are iterated applications of a substitution-permutation function with a key addition layer (a particular case of an SPN). The distinguisher exploited in [12] and previous work [16,17] is related to the "capacity of a multidimensional linear approximation", which in the targeted permutations benefits from the bit-permutation linear layer.

The generalization from [12] and their definition for a "PRESENT-like permutation" implies that both GIFT-64 and GIFT-128 are vulnerable to this attack. Even though this is a known-key attack it should not be dismissed on light that the use of the ciphers in combination with operation modes for providing different security services can lead to unexpected vulnerabilities such as the case of the birthday attacks.

Both variants of Midori are unaffected by this attack since the composition used in their round functions differ from a "PRESENT-like permutation."

2.3 Side-channel Attacks

Lightweight block ciphers are often implemented on physically reachable areas. Since an attacker has direct access to the hardware, it gains greater advantages from the information leaked by

the device through "side channels." In [13] the authors present "the first practically realizable side-channel assisted fault attack on PRESENT", that attack can retrieve the last round key with 4 random nibble faults in the best case, and 7-8 faults in the average case.

The authors describe that their attack exploits the fact that PRESENT uses bit-permutations instead of Maximum-Distance-Separable (MDS) layers to generate diffusion. As mentioned in [13], the attack should also be applicable to GIFT-64 and GIFT-128. The silver lining is that the nibble-based permutations used affects a single non-overlapping nibble, this is a pattern used for reducing the width of these permutations [18–20].

The authors in [13] claim that their attack would fail against AES or LED since they use a MDS layer. Although Midori employs an "Almost MDS" layer, its designers claim that by using additional "optimal cell-permutation layers" they can improve the diffusion speed and increase the number of active S-boxes per round. The use of these two layers should provide resilience against the described attack.

2.4 Keynotes

The creation of secure lightweight block ciphers is a challenging task. Designers must be aware of the application requirements and the improvements of attack models. We can reflect on the following notes:

- State sizes of 64-bits should be avoided and deprecated gradually.
- Key sizes of 128-bits should be used for providing long-term security [21].
- The design of the diffusion layer should not be taken lightly as it can provide much information for attackers.
- Involutive designs are interesting, but decryption is seldom used. In practical scenarios forward modes such as CTR offer greater efficiency with little cost.
- Although side channel countermeasures are costly, they should be considered for any design expected to be deployed in unprotected environments.

3 Methods

In this section we focus on encryption functions since these can be used to encrypt and decrypt data under the CTR mode of operation [22]. We use 128-bit key sizes for all the block ciphers.

For each lightweight block cipher we study its *iterative* and *serial* architectures [23]. We define two types of serial architectures. The first type (serial-1) targets a reduction in the number of 4-bit substitution boxes (SBOX) from $n/4$ to two. The second type of architecture (serial-2) seeks to reduce not only the number of substitution boxes, but also the width of other transformations when possible.

3.1 Preliminaries

A block cipher uses as input an n-bit data block, which is a fragment of a message or *plaintext* ($\mathcal{P}$), and stores it in a register denominated *state* (S). A k-bit secret information denominated *key* ($\mathcal{K}$) is used in the processing of S. A round-based function which iterates r rounds over S is applied to encrypt the input data block. The result of this processing is a new n-bit data block ($\mathcal{C}$) which is a fragment of an encrypted message or *ciphertext*.

In this paper, for each selected lightweight block cipher we study its *iterative* and *serial* architectures [23]. For the former, each round takes one cycle and all the internal transformations are applied over the state on parallel. For the latter, one or multiple internal transformations are *serialized*. This means that the word in S is divided in nibbles, which are then processed serially. While iterative architectures are close implementations of the specification algorithm, their serial counterparts explore area/performance trade-offs to reduce the hardware resources required. We

define two types of serial architectures. The first type (serial-1) targets a reduction in the number of 4-bit substitution boxes (SBOX) from $n/4$ to two. These SBOX are substitutions applied to every four bits of the state. Since SBOX require the most resources in a block cipher [2], reducing the number of SBOX should yield the greatest improvement to the implementation size. The second type of architecture (serial-2) seeks to reduce not only the number of substitution boxes, but also the width of other transformations when possible. This design philosophy was used in [20] to achieve smaller designs for PRESENT.

In regards to the processing cycles of an architecture, we define two types of latency. The first one is related with the input and output (IO) ports of the data blocks. Since the number of IO ports available in constrained devices is often limited, we use reduced IO ports in our implementations. This strategy is also employed in [8, 20, 23]. Since to the IO ports are smaller than the state, the input/output data blocks must be divided into d-bit nibbles which are then taken/produced serially. The second type of latency is associated with the data processing (DP). For iterative architectures, the DP latency equals the number of rounds in the algorithm specification. For serial architectures, the DP latency accounts for the cycles required to serialize certain steps of the datapath and the cycle required to apply any parallel transformations remaining.

3.2 PRESENT

The PRESENT block cipher follows a Substitution-Permutation Network (SPN) construction. It has a block size of 64-bit and supports key sizes of 80-bit and 128-bit. The specification for its encryption function is presented in Alg. 1.

Algorithm 1 Encryption in the PRESENT cipher [2] where $n = 64$, $k = 128$, and $r = 31$.

Input: A 64-bit plaintext $\mathcal{P}$, a k-bit secret key $\mathcal{K}$.
Output: A 64-bit ciphertext $\mathcal{C}$.

```
S ← P
K_i ← generateRoundKeys(K)
for i = 1 to r do
    S ← addRoundKey(S, K_i)
    S ← sBoxLayer(S)
    S ← pLayer(S)
end for
C ← addRoundKey(S, K_{r+1})
return C
```

We first study the basic implementation of the block cipher with IO ports of 8-bit shown in Fig. 1. That hardware realization of PRESENT requires 17 substitution boxes (SBOX), 77 XOR gates, and 192 Flip-Flops (FF). In regards to latency, 16 cycles are required to input the plaintext and the cipher key, 31 cycles to encrypt the data, and 8 cycles to produce the output. In total this sums 55 cycles.

In the serial-1 architecture for PRESENT shown in Fig. 2, the the main optimization involves reducing the number of substitution boxes to two. The substitution boxes used in the key generation are also removed and the number of XOR gates is reduced. The trade-off is an increment in the number of cycles required to encrypt the data. The implementation of this design requires 2 SBOX, 21 XOR gates, and 192 FF. The number of cycles needed to take the input and to produce the output are 16 and 8, respectively. Each iteration of the round function uses 8 cycles due to the reduced number of SBOX and an additional cycle to apply the permutation over the state and to update the key register, which is $31 \times 9 = 279$ cycles. With this design 303 latency cycles are needed to encrypt a data block.

The serial-2 PRESENT architecture under study is the one reported in [20]. In that design the main strategy was outlined as reducing the whole datapath to 16-bit, which is a quarter of its

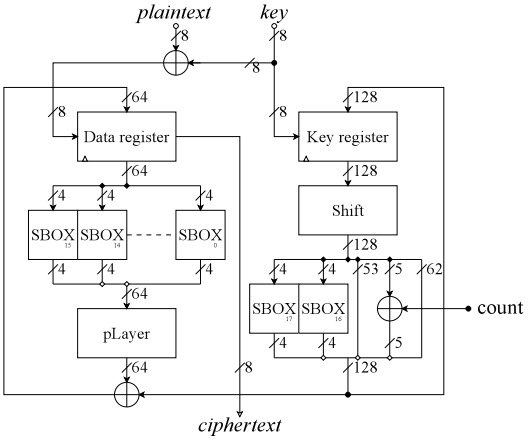

Fig. 1. Iterative architecture for PRESENT as proposed in [23].

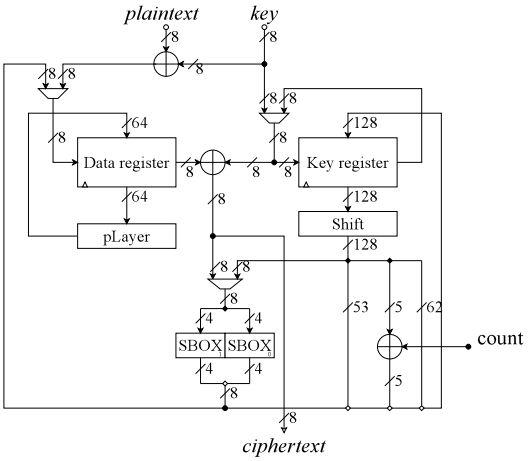

Fig. 2. Serial-1 architecture for PRESENT as proposed in [23].

block size. The hardware realization for this design involves the use of 6 SBOX, 21 XOR gates, and 192 FF. In this case the input and output processes use 8 and 4 cycles, respectively. The data processing takes 4 cycles per iteration of the round function, which amounts to $31 \times 4 = 124$ cycles. The total latency of the design is of 136 cycles. This architecture is illustrated in Fig. 3.

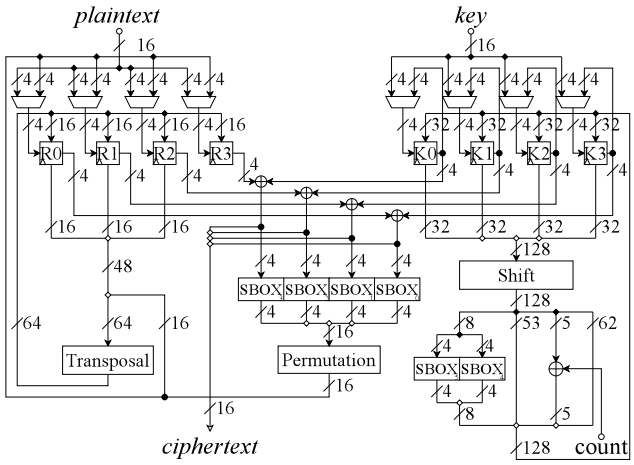

Fig. 3. Serial-2 architecture for PRESENT.

3.3 Midori

Midori is a lightweight block cipher "that is optimized with respect to the energy consumed by the circuit per bit in encryption or decryption operation" [6]. Alg. 2 presents the Midori specification, which can operate over data blocks of 64 or 128 bits. A key size of 128-bit is used in both versions of the algorithm. Midori also has an SPN structure. More details about Midori can be found in [6]. This block cipher operates over data blocks of 64 or 128 bits. A key size of 128-bit is used in both versions of the algorithm. Midori also has an SPN structure.

Algorithm 2 Encryption procedure for Midori as described in [6] where $n = \{64, 128\}$, $k = 128$, and $r = \{16, 20\}$ for {Midori-64, Midori-128}, respectively.

Input: An n-bit plaintext $\mathcal{P}$, a 128-bit secret key $\mathcal{K}$.
Output: An n-bit ciphertext $\mathcal{C}$.

$WK, RK_i \leftarrow f(\mathcal{K})$
$S \leftarrow \text{KeyAdd}(\mathcal{P}, WK)$
for $i = 0$ **to** $r - 2$ **do**
  $S \leftarrow \text{SubCell}(S)$
  $S \leftarrow \text{ShuffleCell}(S)$
  $S \leftarrow \text{MixColumn}(S)$
  $S \leftarrow \text{KeyAdd}(S, RK_i)$
end for
$S \leftarrow \text{SubCell}(S)$
$\mathcal{C} \leftarrow \text{KeyAdd}(S, WK)$
return $\mathcal{C}$

The iterative architecture for Midori created in this work is presented in Fig. 4. This design can describe both Midori-64 and Midori-128 realizations. It follows the algorithm specification closely but uses 8-bit IO ports. In hardware, this architecture requires 16 SBox (which are of 4-bit for Midori-64 and of 8-bit for Midori-128), an $n-$bit transformation which can be simplified as $n/2$ XOR gates (MixColumn), an n-bit XOR layer, 16 XOR gates for the key mechanism, $r - 2$ 16-bit round constants, and $n + 128$ FF. To take the input block and the cipher key 16 cycles are used in both Midori-64 and Midori-128. For the former, 17 cycles are required to process the data and 8 cycles are used to produce the output. For the latter, the data processing takes 21 cycles and producing the output requires 16 cycles. In total the iterative architectures for Midori-64 and Midori-128 have a latency of 41 and 53 cycles, respectively.

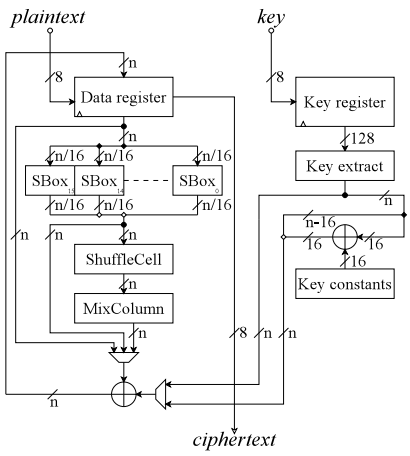

Fig. 4. Iterative architecture for Midori-64 and Midori-128.

Fig. 5 illustrates the serial-1 architecture created in this work for Midori-64 and Midori-128. This version focuses on reducing the SBOX count. For Midori-64, the SBox illustrated represents two 4-bit SBOX. For Midori-128, eight 8-bit permutations are also allocated inside the SBox. These permutations work together with two 4-bit SBOX to produce the output of the substitution layer. Two of the four permutations are selected depending on the position in the state of the data nibble being processed. The hardware realization of this design uses two 4-bit SBOX, the n/2 XOR gates simplification of MixColumn, an n-bit XOR layer, the 16 XOR gates used in the key generation, $r-2$ 16-bit round constants, and $128+n$ FF. In regards to latency, for both instances of Midori, 16 cycles are requires to take the input data. For Midori-64 the data processing takes an initial cycle for the first key addition and 9 cycles per round, thus $1+(16\times 9)=145$ total cycles. To produce the output 8 cycles are used. This Midori-64 architecture has a latency of 169 cycles. For Midori-128 the data processing requires the key addition cycle and 16 cycles for each iteration of the round function, this is $1+(20\times 17)=341$ cycles. Producing the output has a cost of 16 cycles, thus the Midori-128 design requires a total of 373 cycles.

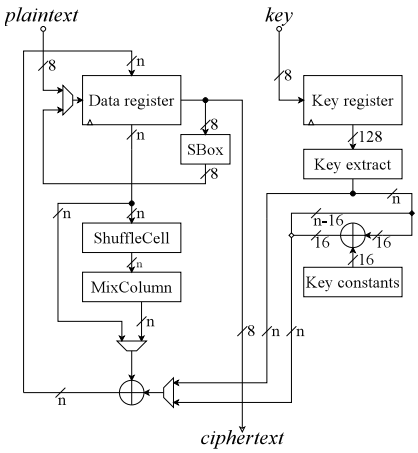

Fig. 5. Serial architecture for Midori-64 and Midori-128 with reduced number of substitution boxes (serial-1).

The serial-2 architecture developed in this work for Midori is shown in Fig. 4 (right). In this design the datapath width d is reduced to 16-bit for Midori-64 and 32-bit for Midori-128. In both cases the operations which can be serialized are the substitution layer, the MixColumn step, and the key addition. The n-bit permutation is performed during an extra cycle in the round. In order to achieve this design we modified the Midori algorithm so that the SubCell and ShuffleCell operations are swapped. This allows pushing the ShuffleCell step from the i iteration back to the $i-1$ iteration. From this, the serializable steps of the algorithm are now grouped at the beginning of the round and can be processed together in 4 cycles. The non serializable part is left at the end of the round and performed in the extra cycle. The cost of this modification only affects Midori-128 due to the 8-bit permutations used inside the SBox which have to be shuffled. For implementing Midori-64 this design requires four 4-bit SBOX, an 8 XOR gates version of MixColumn, 32 XOR gates, 14 16-bit round constants, and 192 FF. The latency of this design amounts to 96 cycles, detailed as eight cycles to input the data, $(16\times 5)+4=84$ cycles for the data processing, and four cycles to produce the output. For implementing Midori-128 the hardware requirements are four 8-bit SBox, a 16 XOR gates version of MixColumn, 48 XOR gates, 18 16-bit round constants, and 256 FF. Four cycles are used to input the data, $(20\times 5)+4=104$ cycles are used for the data processing, and four cycles are used to produce the output, which sums a latency of 112 cycles to encrypt the data with this architecture.

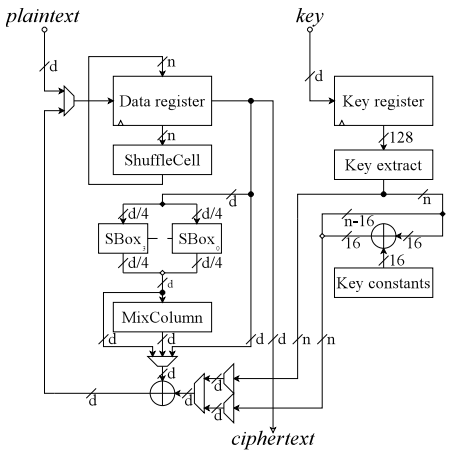

Fig. 6. Serial architecture for Midori-64 and Midori-128 with generalized reduction of the datapath width (serial-2).

3.4 GIFT

The block cipher GIFT is said to be a direct improvement to PRESENT "that provides a much increased efficiency in all domains (smaller and faster)" and also patches security weaknesses of the latter (in regards to linear cryptanalysis). Two specifications of the algorithm were presented in [7] for block sizes of 64 and 128-bit, these are shown in Alg. 3. A key size of 128-bit is used in both versions of the algorithm.

Algorithm 3 Encryption procedure for GIFT as described in [7] where $n = \{64, 128\}$, $k = 128$, and $r = \{28, 40\}$ for {GIFT-64, GIFT-128}, respectively.

Input: An n-bit plaintext $\mathcal{P}$, a 128-bit secret key $\mathcal{K}$.
Output: An n-bit ciphertext $\mathcal{C}$.

$S \leftarrow \mathcal{P}$
$K \leftarrow \mathcal{K}$
$C_0 \leftarrow 0$
for $i = 1$ **to** r **do**
  $S \leftarrow \text{SubCells}(S)$
  $S \leftarrow \text{PermBits}(S)$
  $C_i \leftarrow f(C_{i-1})$
  $S \leftarrow \text{AddRoundKey}(S, K, C_i)$
  $K \leftarrow g(K)$
end for
$\mathcal{C} \leftarrow S$
return $\mathcal{C}$

In Alg. 3 the function f represents updating the round constant, which is generated using a LFSR. The function g represents a permutation which updates the key register in order to generate the next round key.

The iterative architecture created for GIFT is presented in Figure 7 (left). This design is a direct implementation of the specification with 8-bit IO ports. For GIFT-64 or GIFT-128 the design requires $n/4$ 4-bit SBOX, $n/2+6$ XOR gates, a NOT gate, and $n+134$ FF. The latency for GIFT-64 is of 24 cycles for the data input and output and 28 cycles for the data processing, which sums 52 cycles. The latency for GIFT-128, 16 cycles are spent in the data input, 40 cycles are used in the data processing, and 16 cycles are required to produce the output, which amounts to a total latency of is 72 cycles.

Fig. 8 presents our serial-1 architecture for GIFT. This design uses 8-bit IO ports and has a serialized application of the substitution layer based on two 4-bit SBOX. The architecture

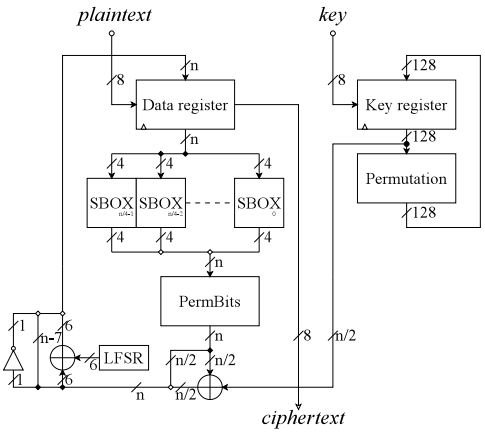

Fig. 7. Iterative architecture for GIFT-64 and GIFT-128.

illustrated describes both GIFT-64 and GIFT-128. In the case of GIFT-64 the implementation requires two 4-bit SBOX, 38 XOR gates, a NOT gate, and 198 FF. For this version 16 cycles are needed to take in the data block and the cipher key, and 8 cycles are used to produce the output. Due to the serialization of the substitution layer, 8 cycles are used per round plus 1 extra cycle for the processing of the rest of the round operations. This sums a latency of $16 + (9 * 28) + 8 = 276$ cycles. For GIFT-128, two 4-bit SBOX, 70 XOR gates, a NOT gate, and 262 FF are used. In this case the latency is of $16 + (17 * 40) + 16 = 712$ cycles, which includes the data input, processing, and output steps.

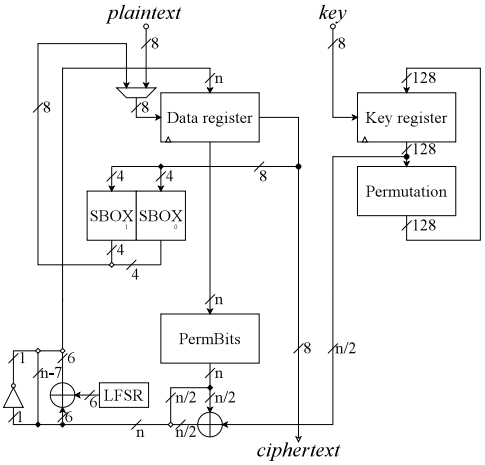

Fig. 8. Serial-1 architecture for GIFT-64 and GIFT-128 with reduced number of substitution boxes.

Our serial-2 architecture for GIFT, shown in Fig. 9, was created by serializing the substitution, permutation, and key addition layers. in Alg. 3. The datapath width d was adjusted to 16-bit for GIFT-64 and to 32-bit for GIFT-128. The reduction of the substitution layer is straightforward for GIFT. We used a regular pattern found in the original permutation to reduce the permutation layer width to a quarter of its original width. However, by using this reduction an additional transposition of the state is required. Let us use a 2-D representation of the state as described in [7]. The new reduced permutation will yield a transposed version of the 2-D state, arranged in 16 $n/16$-bit nibbles. Thus, the additional permutation is a shuffling of the state in 4-bit nibbles for GIFT-64 and 8-bit nibbles for GIFT-128. This strategy is similar to that used in [20] for PRESENT. The *small* permutation is applied on a serialized manner while the *transposition* is applied over the state during an additional cycle. The round key also needs

to be shuffled to accommodate for this intermediate result. In order to serialize the key addition step, we separated the addition of the keying materials and the addition of the round constants. The keying materials are derived from the key register, shuffled, and serialized, before being applied to the state. The round constants are applied to the state during the additional cycle while the key register is updated. Based on this architecture, the implementation of GIFT-64 requires four 4-bit SBOX, 14 XOR gates, a NOT gate, and 198 FF. The implementation of GIFT-128 uses eight 4-bit SBOX, 22 XOR gates, a NOT gate, and 262 FF. In this design, the data input takes $128/d$ cycles and producing the output takes four cycles. The data processing uses four cycles for the serialized steps and a fifth cycle for completing the round processing and updating the key register. The total latency for GIFT-64 and GIFT-128 is of $8+(5*28)+4 = 152$ cycles and $4+(5*40)+4 = 208$ cycles, respectively.

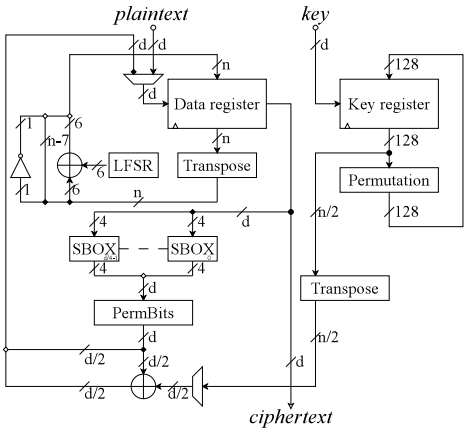

Fig. 9. Serial-2 architecture for GIFT-64 and GIFT-128 with generalized reduction of the datapath. The value d equals $n/4$.

3.5 Gimli

Gimli is a 384-bit permutation "designed to achieve high security with high performance across a broad range of platforms". According to its creators, this permutation can be easily used to build high-security block ciphers. We have included this algorithm into our review since its authors claim it was designed for "energy-efficient hardware" and "compactness". The specification for this function is presented in Alg. 4. Since the implementations provided in [8] do not implement a block cipher, a secret key is not used.

The design shown in Fig. 10 is the iterative implementation for Gimli as presented in [8]. In this instance a block size of 384-bit is used. The application of the parallel SP-box requires two 384-bit permutations, 768 XOR gates, 256 AND gates, and 128 OR gates. The Big-Swap and the Small-Swap can be seen as 384-bit permutations. Finally, 37 XOR gates are used for the addition of the round constants. Since this architecture uses 8-bit IO ports, a latency of 48 cycles is needed to input a 384-bit message. The data processing takes 24 cycles. Lastly another 48 cycles are required to produce a 384-bit output block. In total, this architecture has a latency of 120 cycles.

The serial-1 architecture for Gimli is shown in Fig. 11, this design was retrieved from [8]. The main strategy for reducing resources consists on serializing the application of the SP-box layer. In this instance, 96-bit of the state are processed in parallel so that four cycles are required for each application of the SP-box layer. The other transformations are applied to the state in a fifth cycle, which is present for half of the rounds. The application of the serialized SP-box requires two 96-bit permutations, 192 XOR gates, 64 AND gates, and 32 OR gates. The Big-Swap and the Small-Swap can still be represented as 384-bit permutations and 37 XOR gates

Algorithm 4 The permutation GIMLI as presented in [8].

Input: An r-bit block $\mathcal{P}$.
Output: A r-bit block $\mathcal{C}$.

```
S ←⊕ P
for i = 24 to 1 do
    S ← SP-box(S)
    if i mod 4 = 0 then
        S ← Small-Swap(S)
    else if i mod 4 = 2 then
        S ← Big-Swap(S)
    end if
    if i mod 4 = 0 then
        S ← Add-Constant(S, 0x9e377900, i)
    end if
end for
C ← S
return C
```

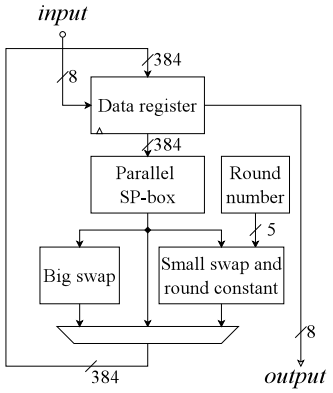

Fig. 10. Iterative architecture for GIMLI provided in [8].

are also used for the addition of the round constants. A latency of 48 cycles is required for the data input and also for the data output. In the data processing half of the rounds only apply the SP-box layer which takes 4 cycles, the other half takes 5 cycles. The total latency for this architecture is of $48 + (12 * 4) + (12 * 5) + 48 = 204$ cycles.

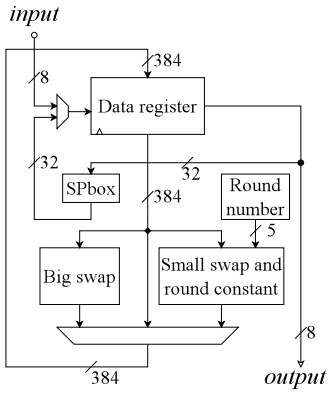

Fig. 11. Serial architecture for GIMLI as presented in [8] (serial-1).

3.6 Summary

Table 1 provides a summary of the different architectures discussed in this section.

Table 1. Summary of the different designs reviewed in this section.

Label	Alg.	State (bits)	Key (bits)	Rounds	Ref.	Arch.	Latency (cycles)	Hardware resources SBOX	Gates	Const.	FFs
C01	PRESENT	64	128	31	[23]	Iterative	55	17	77	1	192
C02					[23]	Serial-1	303	2	21	1	192
C03					[20]	Serial-2	136	6	21	1	192
C04	Midori-64	64	128	16	Ours	Iterative	41	16	112	14	192
C05					Ours	Serial-1	169	2	112	14	192
C06					Ours	Serial-2	96	4	40	14	192
C07	Midori-128	128	128	20	Ours	Iterative	53	32	208	18	256
C08					Ours	Serial-1	373	2	208	18	256
C09					Ours	Serial-2	112	8	64	18	256
C10	GIFT-64	64	128	28	Ours	Iterative	52	16	39	0	198
C11					Ours	Serial-1	276	2	39	0	198
C12					Ours	Serial-2	152	4	15	0	198
C13	GIFT-128	128	128	40	Ours	Iterative	72	32	71	0	262
C14					Ours	Serial-1	712	2	71	0	262
C15					Ours	Serial-2	208	8	23	0	262
C16	GIMLI	384	-	24	[8]	Iterative	120	0	1189	2	384
C17					[8]	Serial-1	204	0	325	2	384

4 Experimental Evaluation

The different designs in Table 1 are used as configurations for our experimental evaluation. The VHDL description for the PRESENT implementations is the one used in [23] and [20]. The hardware descriptions for the different Midori and GIFT architectures were created in this work. Lastly, the VHDL description for the GIMLI architectures is the one used in [8].

All the configurations were implemented for the xc6slx16-3csg324 FPGA using ISE Design Suite 14.2 and for the xc7a15t-1cpg236c FPGA using Vivado Design Suite 2017.3 Version. The synthesis process was configured with *Area* as optimization goal in both instances. The use of RAM/ROM elements was disabled for all the implementations. We provide Post-Place & Route area results in terms of slices (SLC), Look-Up-Tables (LUT), and Flip-Flops (FF) for all the configurations in the two implementation platforms.

In regards to performance, we report the total latency (LAT), the maximum achievable frequency (Fmax) from the Post-Place & Route report, the runtime (Time), and the throughput (Thr) for each configuration. The throughput was calculated for operational frequencies of 100KHz and Fmax as Thr = (state size $\times$ Freq) / LAT.

A power analysis for the xc6slx16-3csg324 FPGA was performed using the Xilinx XPower Analyzer tool version 14.2 for operational frequencies of 100KHz and Fmax. The power estimations were obtained after place and route using Xilinx XPower Analyzer 14.3 with HIGH overall confidence level. This analysis used the Post-Place & Route Design file (ncd), a Physical Constraints file (pcf) specific for the evaluation target, and a Simulation Activity file (saif) generated from a Post-Place & Route simulation in Isim. The Simulation Run Time was of 100ms for all the 100KHz instances and of 100μs for all the Fmax instances. From this evaluation we report the quiescent and dynamic power for each design. The power dissipation and the performance at 100KHz and Fmax were then used to calculate the energy consumption for each configuration.

We use three efficiency (EFF) metrics to evaluate the different configurations. The first figure represents the relation between performance and area and is given in Kbps per SLC.

The second figure represents the relation between energy and area and is given in μJ per SLC. Lastly, the third efficiency indicator represents the relation between the energy spent and the bits processed and is expressed in nJ per bit. These metrics are expected to indicate the prowess of the configurations for different trade offs, which might be attractive for different application scopes.

5 Results

The area and performance results for the implementations in the xc6slx16-3csg324 FPGA are presented in Table 2.

Table 2. Area and performance results for the xc6slx16-3csg324 FPGA using operational frequencies of 100KHz and Fmax.

Cipher	Ref.	Conf.	Size (bits)				Resources			LAT	Fmax	Time (μs)		Thr (Mbps)		EFF (Kbps/SLC)	
			State	Key	IO	DP	FF	LUT	SLC	(Cycles)	(MHz)	100KHz	Fmax	100KHz	Fmax	100KHz	Fmax
PRESENT	[23]	C01	64	128	8	64	200	202	56	55	145.35	550	0.38	0.12	169.13	2.08	3020.24
		C02	64	128	8	8	203	157	45	303	131.87	3030	2.30	0.02	27.85	0.47	618.99
	[20]	C03	64	128	16	16	201	220	61	148	159.21	1480	0.93	0.04	68.85	0.71	1128.65
Midori-64	Ours	C04	64	128	8	64	200	356	118	41	166.17	410	0.25	0.16	259.38	1.32	2198.17
		C05	64	128	8	8	203	262	109	169	141.56	1690	1.19	0.04	53.61	0.35	491.83
		C06	64	128	16	16	202	268	96	96	157.80	960	0.61	0.07	105.20	0.69	1095.86
Midori-128	Ours	C07	128	128	8	128	264	549	157	53	157.95	530	0.34	0.24	381.47	1.54	2429.75
		C08	128	128	8	8	269	390	115	373	139.31	3730	2.68	0.03	47.81	0.30	415.72
		C09	128	128	32	32	267	482	155	112	86.07	1120	1.30	0.11	98.37	0.74	634.64
GIFT-64	Ours	C10	64	128	8	64	205	189	58	52	218.10	520	0.24	0.12	268.43	2.12	4628.17
		C11	64	128	8	8	209	151	44	276	225.53	2760	1.22	0.02	52.30	0.53	1188.56
		C12	64	128	16	16	208	235	67	152	219.44	1520	0.69	0.04	92.40	0.63	1379.06
GIFT-128	Ours	C13	128	128	8	128	270	290	93	72	189.93	720	0.38	0.18	337.66	1.91	3630.75
		C14	128	128	8	8	275	286	81	712	144.15	7120	4.94	0.02	25.92	0.22	319.94
		C15	128	128	32	32	273	256	66	208	217.63	2080	0.96	0.06	133.92	0.93	2029.16
GIMLI	[8]	C16	384	-	8	384	394	587	174	120	121.34	1200	0.99	0.32	388.30	1.84	2231.62
		C17	384	-	8	32	397	493	164	204	148.88	2040	1.37	0.19	280.24	1.15	1708.76

Figures 12 and 14 offer graphic representation for some results from Table 2.

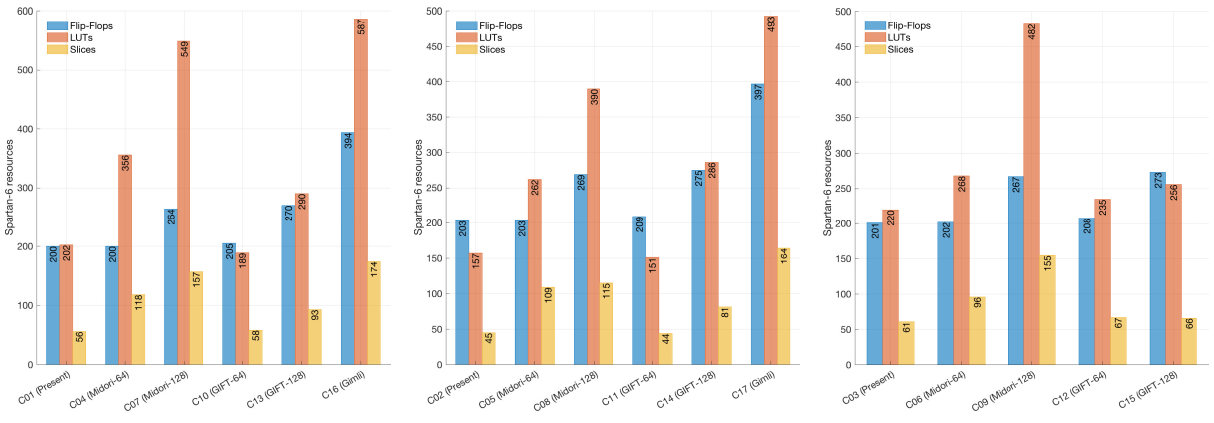

Fig. 12. Area results of lightweight block ciphers using the xc6slx16-3csg324 FPGA. Results obtained after place and route. Plots are divided between iterative (left), serial-1 (center), and serial-2 (right) architectures.

The results for the power analysis and energy consumption calculations for the different configurations implemented in the xc6slx16-3csg324 FPGA are provided in Table 3.

Figure **??** shows a graphic representation for some results from Table 3.

The area results in the xc7a15t-1cpg236c FPGA are shown in Fig. 15.

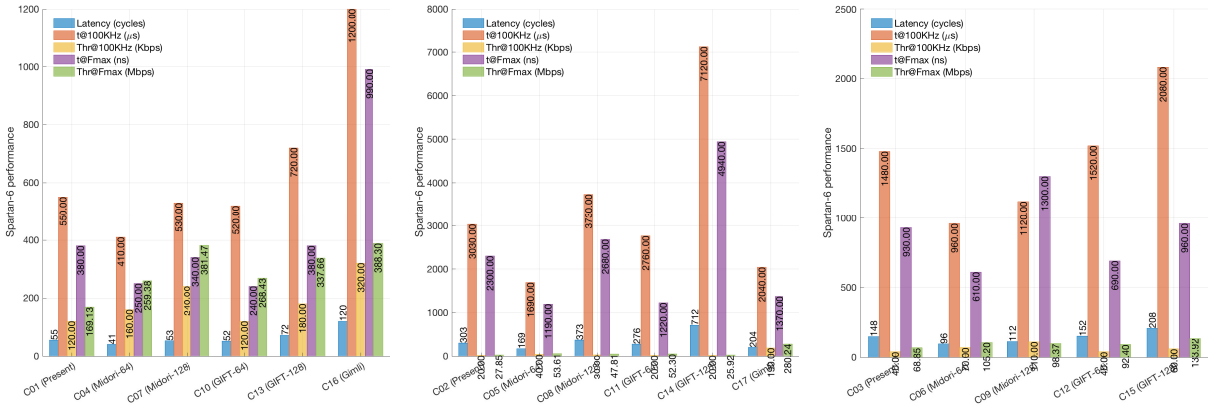

Fig. 13. Performance results of lightweight block ciphers using the xc6slx16-3csg324 FPGA for operational frequencies of 100KHz and Fmax. Results obtained after place and route. Plots are divided between iterative (left), serial-1 (center), and serial-2 (right) architectures.

Table 3. Power and energy results for the xc6slx16-3csg324 FPGA using operational frequencies of 100KHz and Fmax.

Cipher	Ref.	Conf.	POW@100KHz (mW)		ENE@100KHz	EFF@100KHz		POW@Fmax (mW)		ENE@Fmax	EFF@Fmax	
			Quiescent	Dynamic	(nJ)	(nJ/SLC)	(nJ/bit)	Quiescent	Dynamic	(nJ)	(nJ/SLC)	(nJ/bit)
PRESENT	[23]	C01	21.51	0.50	12105.50	216.17	189.15	21.82	31.22	20.07	0.36	0.31
		C02	21.51	0.55	66841.80	1485.37	1044.40	21.68	17.48	89.98	2.00	1.41
	[20]	C03	21.51	0.49	32560.00	533.77	508.75	21.89	37.67	55.37	0.91	0.87
Midori-64	Ours	C04	21.51	0.50	9024.10	76.48	141.00	22.00	48.36	17.36	0.15	0.27
		C05	21.51	0.48	37163.10	340.95	580.67	21.69	18.07	47.47	0.44	0.74
		C06	19.90	0.47	19555.20	203.70	305.55	20.14	24.72	27.29	0.28	0.43
Midori-128	Ours	C07	21.51	0.52	11675.90	74.37	91.22	22.28	75.15	32.69	0.21	0.26
		C08	21.51	0.49	82060.00	713.57	641.09	21.76	24.98	125.14	1.09	0.98
		C09	19.90	0.53	22881.60	147.62	178.76	20.38	48.13	89.15	0.58	0.70
GIFT-64	Ours	C10	21.51	0.49	11440.00	197.24	178.75	21.94	42.29	15.31	0.26	0.24
		C11	21.51	0.46	60637.20	1378.12	947.46	21.68	17.63	48.11	1.09	0.75
		C12	19.90	0.46	30947.20	461.90	483.55	20.10	20.99	28.46	0.42	0.44
GIFT-128	Ours	C13	21.51	0.49	15840.00	170.32	123.75	22.03	51.32	27.81	0.30	0.22
		C14	21.51	0.48	156568.80	1932.95	1223.19	21.65	14.24	177.27	2.19	1.38
		C15	19.90	0.46	42348.80	641.65	330.85	20.20	30.79	48.73	0.74	0.38
GIMLI	[8]	C16	21.51	0.58	26508.00	152.34	69.03	21.74	23.39	44.63	0.26	0.12
		C17	21.51	0.57	45043.20	274.65	117.30	21.73	21.99	59.91	0.37	0.16

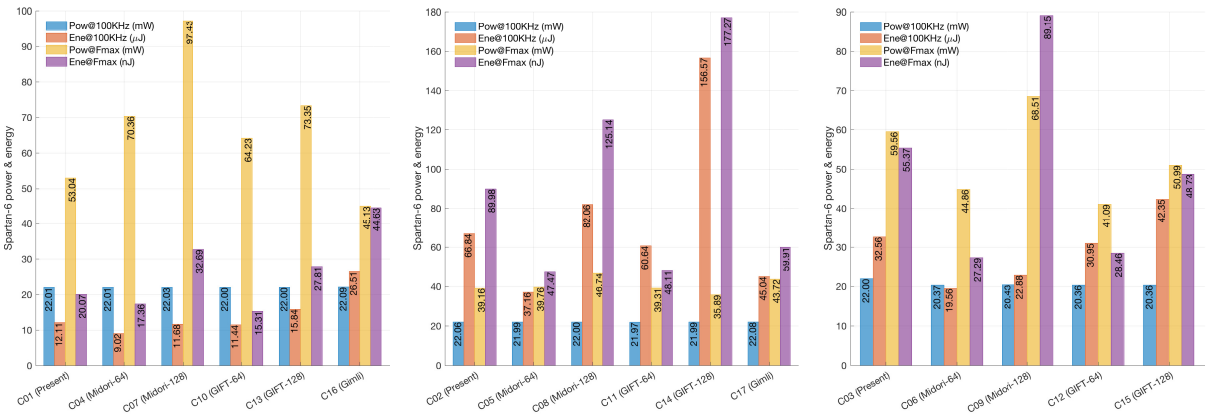

Fig. 14. Power and energy results of lightweight block ciphers using the xc6slx16-3csg324 FPGA for operational frequencies of 100KHz and Fmax. Results obtained after place and route. Plots are divided between iterative (left), serial-1 (center), and serial-2 (right) architectures.

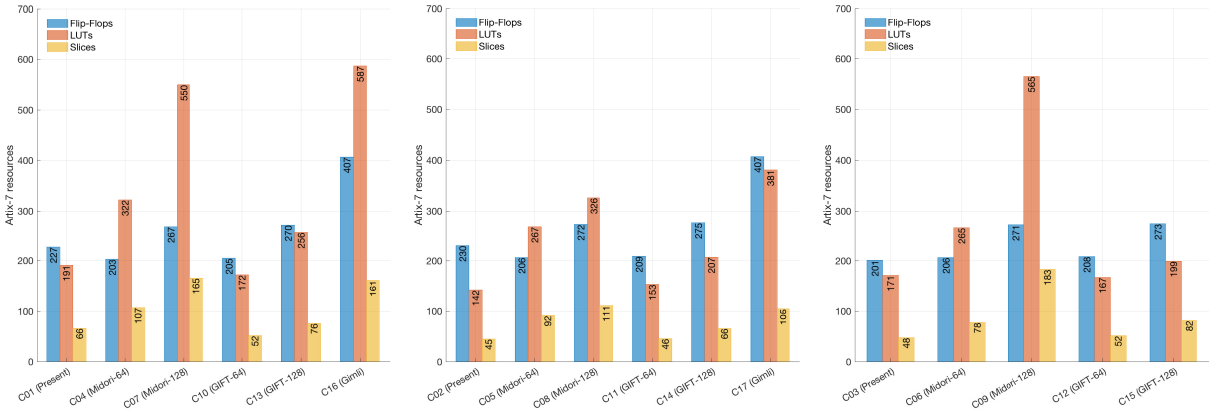

Fig. 15. Area results of lightweight block ciphers using the xc7a15t-1cpg236c FPGA. Results obtained after place and route. Plots are divided between iterative (left), serial-1 (center), and serial-2 (right) architectures.

5.1 Discussion

The iterative architectures presented for Midori and GIFT offer a good balance between area and performance. While iterative implementations are generally more efficient, serial architectures can be used in cases where further area reduction is needed.

The first type of serial architectures described (S1: reduction of the SBOX count) offers a reduction in the hardware resources over the iterative architectures for all the block ciphers reviewed. But the latency is the least favorable for every instance. The second type of serial architectures (S2: general reduction of the datapath) offers better performance than the S1 type. The hardware profile seems to vary from design to design. For PRESENT, the serial-2 architecture (C03) appears to be ineffective compared to C01 in the xc6slx16-3csg324 FPGA. However, the improvement for this design (C03) is palpable when implemented on the xc7a15t-1cpg236c FPGA. Other instances where the serial-2 architecture is advantageous for area occur for Midori-64 and GIFT-128 in the xc6slx16-3csg324 FPGA and for Midori-64 in the xc7a15t-1cpg236c FPGA.

The iterative architectures consistently achieved the smaller energy consumption figures. However, the second type of serial architectures dissipated the least power for Midori and GIFT at low operational frequencies (100KHz). While low energy consumption is a desirable trait for extending the lifetime of battery-powered applications such as WSN motes, low power dissipation is required in passive devices such as RFID tags.

Even though high operational frequencies lead to increased power dissipation, the execution times obtained from the frequency increment, and the resulting energy consumption, are greatly improved. For throughput, the variation from 100KHz to Fmax is generally of three orders of magnitude, which coincides with the reduction of the execution time. The frequency increment causes the power dissipation to double for all the configurations, but due to the delay reduction the final energy consumption is also reduced three orders of magnitude for almost all the configurations. This experiment presents evidence that constrained devices can benefit from high operational frequencies, however, the application scope shall ultimately dictate the operational frequency to be used.

From the results it is possible to note how small IO buffers can be a burden for an implementation. It is known that most constrained devices can not afford to implement wide interfaces. But if the IO width selected is too small, the port interfacing will take longer than the data processing itself. This is more evident with primitives with large block sizes such as Midori-128, GIFT-128 and GIMLI.

The efficiency results allow drawing specific comparisons among the different configurations. From the performance per slice comparison of Fig. 16, it is possible to note that the iterative

architectures (C01, C04, C07, C10, C13, C16) are consistently more efficient compared to the serial realizations. From this set, the iterative implementations of the GIFT block cipher, in both 64 (C10) and 128 bits (C13) instances, resulted to be the most efficient. The results are consistent for both operational frequencies used.

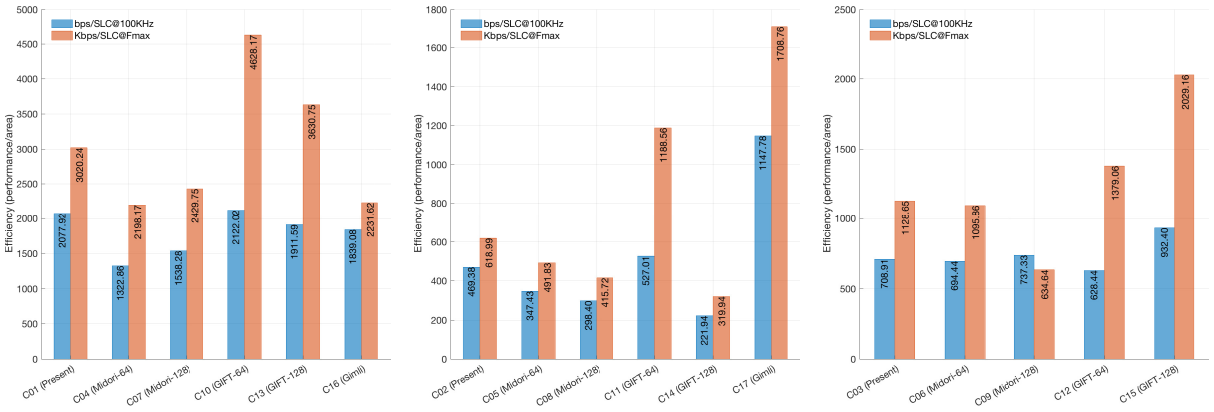

Fig. 16. Efficiency in terms of performance per slice for the different configurations using the xc7a15t-1cpg236c FPGA. Plots are divided between iterative (left), serial-1 (center), and serial-2 (right) architectures.

In terms of energy per slice, Fig. 17, the minimal energy expenditure per slice is observed for the iterative realization of Midori-64 (C04) and Midori-128 (C07). The maximum energy per slice was observed for the serial architectures of GIFT (C14) and PRESENT (C02), these designs both follow the approach of reducing the number of substitution boxes in the design. In this case the behavior for both operational frequencies is similar even though the difference of three orders of magnitude is noticeable.

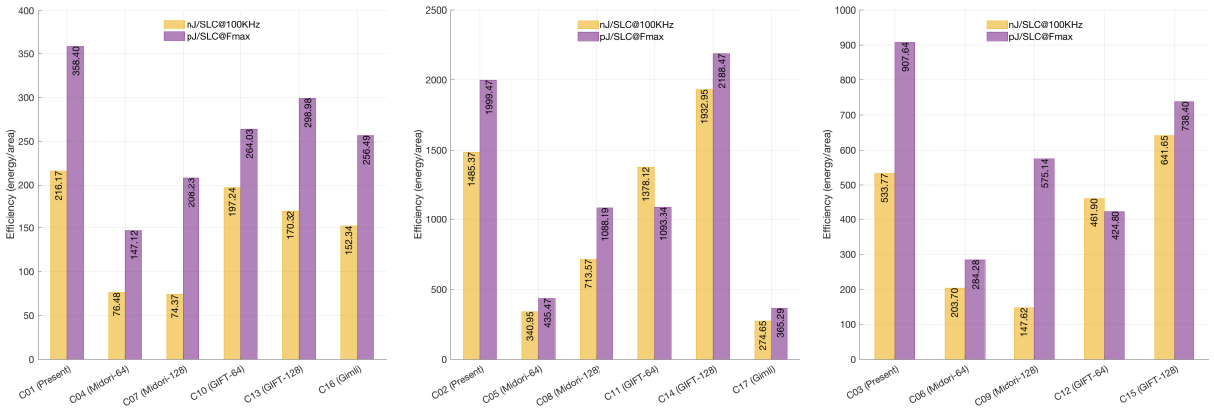

Fig. 17. Efficiency in terms of energy per slice for the different configurations using the xc7a15t-1cpg236c FPGA. Plots are divided between iterative (left), serial-1 (center), and serial-2 (right) architectures.

Both implementations for the GIMLI permutation (C16, C17) obtained the smaller expenditures in the energy per bit efficiency results, as shown in Fig. 18. These were followed by the iterative implementations of GIFT-128 (C13) and GIFT-64 (C10). The same pattern can be discerned for both operational frequencies used.

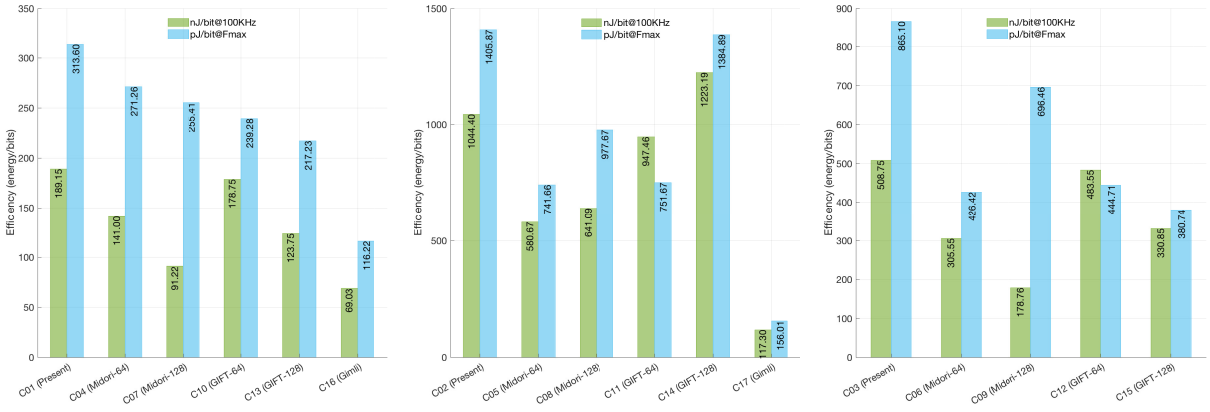

Fig. 18. Efficiency in terms of energy per bit for the different configurations using the xc7a15t-1cpg236c FPGA. Plots are divided between iterative (left), serial-1 (center), and serial-2 (right) architectures.

5.2 Comparison with the State of the Art

In the literature we found one work which implements the Midori block cipher in FPGA [24]. In that reference the authors propose fault-diagnosis schemes for Midori-128 and compare them with the "Original Midori128 Encryption" in an xc7vx330t FPGA. Results in SLC, maximum frequency, power, and throughput are provided for four Midori-128 implementations. Since a different FPGA platform is used and not all the information is available (latency, synthesis criteria) it is difficult to have a fair comparison. In regards to area, the implementations in [24] cost from 155 to 171 SLC while our designs for Midori-128 in the xc6slx16-3csg324 FPGA cost from 112 to 162 SLC. In performance, our fastest implementation of Midori-128 can reach up to 433 Mbps while the range in [24] is 42.52 to 47.41 Gbps. The power requirements for our designs range from 20.42 mW to 22.02 mW while the more modest design in [24] requires 340 mW. Its clear that our implementations were created following different design goals. While the results in [24] were obtained for improved security and high performance, our implementations seek to provide low implementation size and energy consumption.

No FPGA implementations for GIFT were found in our review.

6 Conclusions

In this paper we have studied cryptographic algorithms which can substitute the use of PRESENT and might be considered for future standardization. Even though the modern constructions are efficient, they can not improve the resource requirements of PRESENT for secure state sizes.

We have provided lightweight hardware architectures for the Midori and GIMLI block ciphers. The proposed designs exhibit varying trade-offs which can be attractive for different applications. In order to increase the usability of our work the hardware descriptions for these architectures are made public.

To the best of our knowledge, we have obtained the first FPGA results for the GIFT block cipher and the first area-optimized implementations for Midori.

Extension statement

This paper is an extension of the work presented in [1]. In this version we have expanded on the background and motivation for our research. We have provided further detail of the architectures reviewed. The provided results are now accompanied by graphs for improving the readability of our findings. And lastly, we have included insights on the security of the reviewed block ciphers in relation to modern attacks.

Acknowledgments

The authors would like to thank the authors of [6] and [7] for providing detailed test vectors to validate the implementations presented in this work. And the authors of [23] and [8] for making the source files of their implementations publicly available which contributed to achieve a fairer comparison.

References

1. C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "FPGA-based Assessment of Midori and GIFT Lightweight Block Ciphers," in *Information and Communications Security*, (Cham), Springer International Publishing, 2018.
2. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," in *CHES 2007* (P. Paillier and I. Verbauwhede, eds.), (Berlin, Heidelberg), pp. 450–466, Springer Berlin Heidelberg, 2007.
3. T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A Survey of Lightweight-Cryptography Implementations," *IEEE Design Test of Computers*, vol. 24, pp. 522–533, Nov 2007.
4. "ISO/IEC 29192-2:2012." Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers, Jan 2012.
5. M. Knežević, V. Nikov, and P. Rombouts, "Low-Latency Encryption – Is "Lightweight = Light + Wait"?," in *CHES 2012* (E. Prouff and P. Schaumont, eds.), (Berlin, Heidelberg), pp. 426–446, Springer Berlin Heidelberg, 2012.
6. S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: A Block Cipher for Low Energy," in *Advances in Cryptology – ASIACRYPT 2015* (T. Iwata and J. H. Cheon, eds.), (Berlin, Heidelberg), pp. 411–436, Springer Berlin Heidelberg, 2015.
7. S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A Small Present," in *CHES 2017* (W. Fischer and N. Homma, eds.), (Cham), pp. 321–345, Springer International Publishing, 2017.
8. D. J. Bernstein, S. Kölbl, S. Lucks, P. M. C. Massolino, F. Mendel, K. Nawaz, T. Schneider, P. Schwabe, F.-X. Standaert, Y. Todo, and B. Viguier, "Gimli : A Cross-Platform Permutation," in *CHES 2017* (W. Fischer and N. Homma, eds.), (Cham), pp. 299–320, Springer International Publishing, 2017.
9. K. A. McKay, L. Bassham, M. S. Turan, and N. Mouh, "Report on Lightweight Cryptography," tech. rep., NIST, Information Technology Laboratory, March 2017.
10. C. Blondeau, T. Peyrin, and L. Wang, "Known-key distinguisher on full present," in *Advances in Cryptology – CRYPTO 2015* (R. Gennaro and M. Robshaw, eds.), (Berlin, Heidelberg), pp. 455–474, Springer Berlin Heidelberg, 2015.
11. K. Bhargavan and G. Leurent, "On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 456–467, ACM, 2016.
12. G. Zhang and M. Liu, "A distinguisher on PRESENT-like permutations with application to SPONGENT," *Science China Information Sciences*, vol. 60, p. 072101, Jan 2017.
13. S. Patranabis, J. Breier, D. Mukhopadhyay, and S. Bhasin, "One plus one is more than two: A practical combination of power and fault analysis attacks on present and present-like block ciphers," in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 25–32, Sept 2017.
14. P. Rogaway, "Evaluation of Some Blockcipher Modes of Operation," tech. rep., Cryptography Research and Evaluation Committees (CRYPTREC), feb 2011. [Online].
15. Cisco, "The Zettabyte Era: Trends and Analysis," tech. rep., Cisco public, jun 2017. [Online].
16. J. Y. Cho, "Linear cryptanalysis of reduced-round present," in *Topics in Cryptology - CT-RSA 2010* (J. Pieprzyk, ed.), (Berlin, Heidelberg), pp. 302–317, Springer Berlin Heidelberg, 2010.
17. C. Blondeau and K. Nyberg, "Links between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities," in *Advances in Cryptology – EUROCRYPT 2014* (P. Q. Nguyen and E. Oswald, eds.), (Berlin, Heidelberg), pp. 165–182, Springer Berlin Heidelberg, 2014.
18. P. Yalla and J. P. Kaps, "Lightweight Cryptography for FPGAs," in *International Conference on Reconfigurable Computing and FPGAs, ReConFig '09*, pp. 225–230, Dec 2009.
19. C. A. Lara-Nino, M. Morales-Sandoval, and A. Diaz-Perez, "Novel FPGA-based low-cost hardware architecture for the PRESENT block cipher," in *2016 Euromicro Conference on Digital System Design*, pp. 646–650, Aug 2016.
20. C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Lightweight Hardware Architectures for the Present Cipher in FPGA," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, pp. 2544–2555, Sept 2017.
21. NIST, "NIST Special Publication 800-57. Part 1. Revision 4. Recommendation for Key Management." [Online] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf, 2016. [Last access] 10-08-2018.
22. M. Dworkin, "Recommendation for Block Cipher Modes of Operation ," tech. rep., NIST, Information Technology Laboratory, Dec 2001.
23. N. Hanley and M. ONeill, "Hardware Comparison of the ISO/IEC 29192-2 Block Ciphers," in *2012 IEEE Computer Society Annual Symposium on VLSI*, pp. 57–62, Aug 2012.
24. A. Aghaie, M. M. Kermani, and R. Azarderakhsh, "Fault Diagnosis Schemes for Low-Energy Block Cipher Midori Benchmarked on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 1528–1536, April 2017.