FPGA-Based Assessment of Midori and GIFT Lightweight Block Ciphers

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

CINVESTAV Tamaulipas campus, Victoria, Tamaulipas, México
clara@tamps.cinvestav.mx

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 · FPGA

1 Introduction

For years, the lightweight block cipher PRESENT [6] has been in the spotlight as the most ideal solution for providing confidentiality under constrained environments. However, recent findings [5,11] 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 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 [10] 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.

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 first implementation results for GIFT and the first area-oriented results for Midori in FPGA.

D. Naccache et al. (Eds.): ICICS 2018, LNCS 11149, pp. 745–755, 2018.
https://doi.org/10.1007/978-3-030-01950-1_45

3. All the proposed designs (VHDL) are available at https://www.tamps.cinvestav.mx/~hardware/.

The rest of the paper is structured as follows. Section 2 describes the different architectures for the selected lightweight algorithms, which are implemented and evaluated. Section 3 describes our experimental setup. Section 4 presents our findings. Section 5 concludes this work.

2 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 [7]. We use 128-bit key sizes for all the block ciphers.

For each lightweight block cipher we study its *iterative* and *serial* architectures [8]. 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.

2.1 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 [6].

We first study the basic implementation of the block cipher with IO ports of 8-bit described in [8]. 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 found in [8], 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. 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 [9]. In that design the main strategy was outlined as reducing the whole datapath to 16-bit, which is a quarter of its block size. The hardware realization for this design involves the use of 6 SBOX, 21 XOR gates, and 192 FF. The total latency of the design is of 136 cycles.

2.2 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" [2]. 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.

The iterative architecture for Midori created in this work is presented in Fig. 1 (left). 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. In total the iterative architectures for Midori-64 and Midori-128 have a latency of 41 and 53 cycles, respectively.

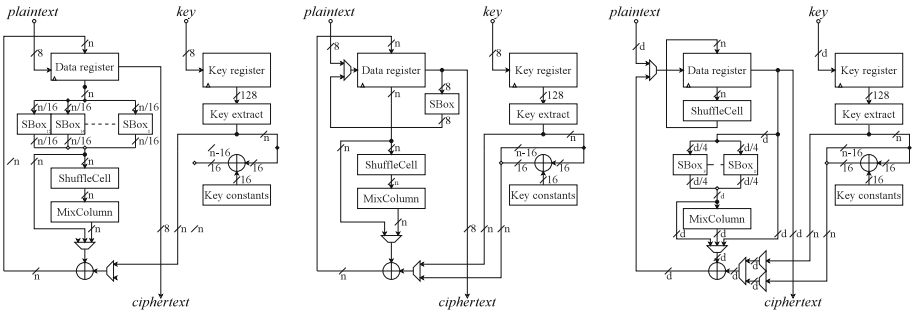

Fig. 1. Iterative (left), serial-1 (center), and serial-2 (right) architectures for Midori-64 and Midori-128

Figure 1 (center) 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. This Midori-64 architecture has a latency of 169 cycles while the Midori-128 design requires a total of 373 cycles.

The serial-2 architecture developed in this work for Midori is shown in Fig. 1 (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. 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. A latency of 112 cycles is required to encrypt the data with this architecture.

2.3 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. Two specifications of the algorithm were presented in [3] for block sizes of 64 and 128-bit. A key size of 128-bit is used in both versions of the algorithm.

The iterative architecture created for GIFT is presented in Fig. 2 (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 52 cycles and the latency for GIFT-128 is 72 cycles.

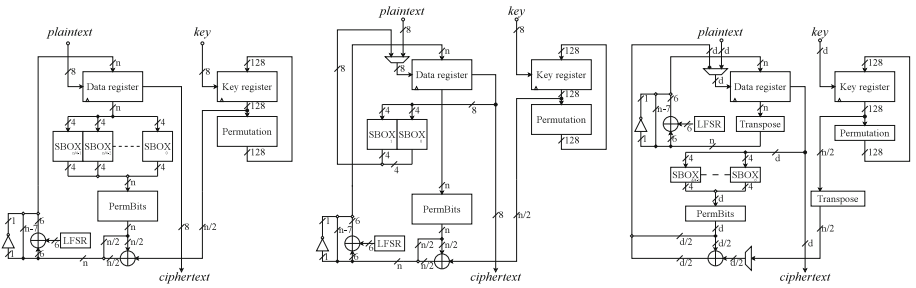

Fig. 2. Iterative (left), serial-1 (center), and serial-2 (right, the value d equals $n/4$) architectures for GIFT-64 and GIFT-128

Figure 2 (center) 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 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 276 latency cycles are required. 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 712 cycles.

Our serial-2 architecture for GIFT, shown in Fig. 2 (right), was created by serializing the substitution, permutation, and key addition layers. 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 [3]. 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 [9] 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. The total latency for GIFT-64 and GIFT-128 is of 152 and 208 cycles, respectively.

2.4 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 [4]. Since the implementations provided in [4] do not implement a block cipher, a secret key is not used.

In the iterative implementation for GIMLI provided in [4] 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. This architecture has a latency of 120 cycles.

A serial-1 architecture for GIMLI was also retrieved from [4]. 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 are also used for the addition of the round constants. A latency of 204 cycles is required for this design.

2.5 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	[8]	Iterative	55	17	77	1	192
C02					[8]	Serial-1	303	2	21	1	192
C03					[9]	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	[4]	Iterative	120	0	1189	2	384
C17					[4]	Serial-1	204	0	325	2	384

3 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 [8] and [9]. 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 [4].

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 100 KHz 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 100 KHz 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 100 ms for all the 100 KHz instances and of 100 μs 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 100 KHz 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.

4 Results

The area and performance results for the implementations in the xc6slx16-3csg324 FPGA are presented in Table 2. 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. The area results in the xc7a15t-1cpg236c FPGA are shown in Fig. 3.

4.1 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

Table 2. Area and performance results for the xc6slx16-3csg324 FPGA using operational frequencies of 100 KHz 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	[8]	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
	[9]	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	[4]	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

Table 3. Power and energy results for the xc6slx16-3csg324 FPGA using operational frequencies of 100 KHz 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	[8]	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
	[9]	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	[4]	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

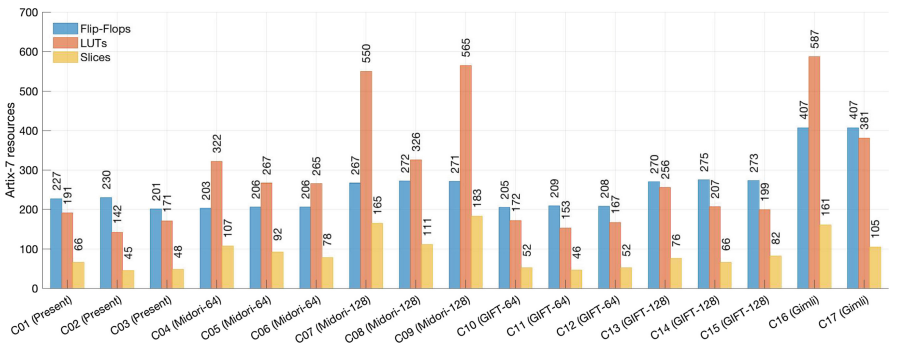

Fig. 3. Area results of lightweight block ciphers using the xc7a15t-1cpg236c FPGA. Results obtained after place and route

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 (100 KHz). 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 100 KHz 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 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.

In terms of energy per slice, 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.

Both implementations for the GIMLI permutation (C16, C17) obtained the smaller expenditures in the energy per bit efficiency results. 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.

4.2 Comparison with the State of the Art

In the literature we found one work which implements the Midori block cipher in FPGA [1]. 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 [1] 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 [1] 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 [1] requires 340 mW. Its clear that our implementations were created following different design goals. While the results in [1] 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.

5 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.

Acknowledgments. 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 México, through the project number 281565.

References

1. Aghaie, A., Kermani, M.M., Azarderakhsh, R.: Fault diagnosis schemes for low-energy block cipher Midori benchmarked on FPGA. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **25**(4), 1528–1536 (2017)
2. Banik, S., et al.: Midori: a block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 411–436. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_17
3. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: a small present. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 321–345. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_16
4. Bernstein, D., et al.: GIMLI : a cross-platform permutation. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 299–320. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_15
5. Bhargavan, K., Leurent, G.: 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, pp. 456–467. CCS 2016. ACM, New York (2016)
6. Bogdanov, A., et al.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_31
7. Dworkin, M.: Recommendation for block cipher modes of operation. Technical report, NIST, Information Technology Laboratory, December 2001
8. Hanley, N., ONeill, M.: Hardware comparison of the ISO/IEC 29192–2 block ciphers. In: 2012 IEEE Computer Society Annual Symposium on VLSI, pp. 57–62, August 2012
9. Lara-Nino, C.A., Diaz-Perez, A., Morales-Sandoval, M.: Lightweight hardware architectures for the present cipher in FPGA. IEEE Trans. Circuits Syst. I: Regul. Pap. **64**(9), 2544–2555 (2017)
10. McKay, K.A., Bassham, L., Turan, M.S., Mouh, N.: Report on lightweight cryptography. Technical report, NIST, Information Technology Laboratory, March 2017
11. Zhang, G., Liu, M.: A distinguisher on PRESENT-like permutations with application to SPONGENT. Sci. China Inf. Sci. **60**(7), 072101 (2017)