Microelectronics Journal 115 (2021) 105167

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Efficient full data-path width and serialized hardware structures of SPONGENT lightweight hash function

Bahram Rashidi

Department of Elec. Eng., Ayatollah Boroujerdi University, Boroujerd, 69199-69411, Iran

ARTICLE INFO

Keywords:
SPONGENT hash function
Serialized hardware implementation
Full data-path
Lightweight
High-throughput
ASIC

ABSTRACT

In this paper, two efficient low-cost and high-throughput structures of the SPONGENT lightweight hash function are presented. Two structures are called full data-path width (parallel) and 4-bit serialized. The serialized architecture is designed by using one multi-task shift register in the round computations with minimum hardware resources. The area consumed in this structure is lower than that of the full data-path structure but the number of clock cycles is increased. The speed computation of the full data-path is higher compared to the 4-bit serialized structure of the SPONGENT hash function because the data are computed in a parallel form. To improving the timing characteristics, we implement the S-box block as the complex block in the SPONGENT hash function based on an Area×Delay optimized circuit. A large number of gates, in the structure, have been implemented by 2-input NAND and 2-input NOR gates in order to reduce delay and area. The performance measurement of the proposed structures is performed by evaluating the parameters such as area consumption, computation time, critical path delay (CPD), throughput, and throughput/area. The implementation results are achieved for all variants of the SPONGENT hash function in 180 nm CMOS technology. The results of area consumption (for 4-bit serialized structure) and throughput (for full data-path structure) show improvements compared to previous works. For area-constrained applications, the proposed structure with a lower data-path width is an appropriate choice. The full data-path width structure can be used for the high-speed and high-throughout cryptographic applications.

1. Introduction

The main challenge in lightweight cryptographic algorithms is reducing the hardware consumption as far as possible while maintaining an acceptable security level. The lightweight devices might process sensitive data and thus require some security, such as encryption, privacy, or authenticity. Classical cryptographic algorithms are not very suitable for lightweight applications [1–4]. Therefore, many lightweight cryptographic schemes have been proposed in the past few years block ciphers such as PRESENT [5], SIMON and SPECK [6,7], PRINCE [8], LED [9] and lightweight hash functions like SPONGENT [10,11], QUARK [12], and PHOTON [13]. The main focus of lightweight cryptography researches has been on the trade-offs between security and performance in terms of speed, area, and throughput. The lightweight cryptographic algorithms can be implemented either in software or in hardware platforms such as Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC). The hardware-based implementations have better performance and can provide security improvements by protecting secret parameters over software solutions.

SPONGENT is a lightweight hash function with flexible hash value sizes [10,11]. A major application for hash functions is in constructing digital signature algorithms. In this case, the hash of a long message is signed instead of directly signing it. Signing a long message with a public-key cryptosystem (PKC) is slow and leads to a signature as large as the message. Reducing a long message to a small hash value using a hash function and signing the hash value is a lot faster and leads to small fixed-size signatures. The SPONGENT hash function has a sponge construction with a permutation block built as a substitution-permutation network (SPN) like PRESENT block cipher. It is a family of five hash lengths of 88, 128, 160, 224, and 256 bits. This hash function is an ISO/IEC international standard hash function 29192-5:2015 [14] with a simple structure. The core parts of the SPONGENT family of hash functions include three steps: an addition step, a substitution step which is implemented by several parallel 4-bit S-boxes, and a bit-permutation step. The number of rounds in SPONGENT cipher ranges from 45 rounds to 385 rounds. Also, variants of the SPONGENT hash function exist for different parameters of width, rate, and capacity. From the hardware point of view, throughput, area consumption, and speed processing of this hash function are important factors for hardware implementation. Therefore, in this paper, we proposed two efficient

E-mail address: b.rashidi@abru.ac.ir.

https://doi.org/10.1016/j.mejo.2021.105167
Received 19 March 2020; Received in revised form 28 April 2021; Accepted 13 July 2021
Available online 17 July 2021

hardware structures for the implementation of the SPONGENT hash function.

The design of low-area (4-bit data-path width) and high-speed (full data-path width) hardware structures for the SPONGENT hash function is the focus of this paper. For area-constrained applications, the proposed structure with a lower data-path width is an appropriate choice. The full data-path width structure can be used for the high-speed and high-throughout cryptographic applications. The main contributions of this work are as follows:

- A 4-bit serialized architecture by using one multi-task shift register called Shift_Reg_Round in the round computations is designed. This block is implemented for the processing of 4-bit words of data. To implementing the SPONGENT hash function in a 4-bit serialized structure, we 4-bit serialize the round computations.
- For the high-throughput applications a full data-path width architecture is proposed. The data-path width b is equal to {88, 136, 176, 240, 264, 272, 336, 384, 480, 672, 768} for different variants of SPONGENT hash function.
- Low-cost 4-bit S-box with comparable area consumption is proposed. The S-box is optimized in terms of delay and area. It consumes only 28 logic gates. A large number of gates, in the structure, have been implemented by the 2-input NAND and NOR gates in order to reduce delay and area.
- The power analysis and timing attacks based on an experimental setup are performed. The achieved power traces are unified during hash operations. For timing attack, the results show that the computation time is fix in the proposed structures for different input messages.

The rest of the paper is organized as follows. Section 3 describes the SPONGENT hash function. The proposed hardware structures are presented in Section 4. Section 6 shows a comparison between this work and other previous works. And, the paper is concluded in Section 7.

2. Related work

There are various low-cost cryptographic systems and many papers are presented for the comparison of these systems. The works [10,11, 15–17] perform the security analysis on the SPONGENT hash function in terms of collision and preimage resistances. In [11] the cryptanalysis methods are performed based on Differential cryptanalysis, Rebound attack (for Collision attacks), Meet-in-the-middle (for Preimage resistance), and Linear attacks. In [16] the cryptanalysis on the reduced versions (the number of rounds is reduced) of SPONGENT is presented. The results suggest that SPONGENT is secure against pure algebraic attacks. In [17] the authors generalize their method and present a distinguisher on PRESENT-like permutations. It is divided into two phases. The first phase is a truncated differential distinguisher with a strong bias, which describes the unbalance of the output collision on some fixed bits, given the fixed input in some bits. The second phase is the meet-in-the-middle layer, which is pre-added to the truncated differential to propagate the differential properties as far as possible. This distinguisher method is applied to all versions of SPONGENT permutations. In the truncated differential phase, the authors achieve one, two, or three rounds more than the results of [11]. In the meet-in-the-middle phase, we get up to 11 rounds to pre-add to the differential distinguishers. Therefore, the approach of work [17] improves the previous distinguishers on all versions of SPONGENT permutations by up to 13 rounds.

Several FPGA and ASIC implementations of the SPONGENT hash function have been reported in the literature [11–22]. In [11] several ASIC hardware implementations from the lowest area to the highest throughput of SPONGENT are provided. In [18] the hardware realization of SPONGENT hash function on FPGA is presented. The implementation has a latency of 1980 clock cycles and consumed 74 Slices (104 FFs and 143 LUTs), with a maximum frequency of 227 MHz, and provides throughput of 29.32 Mbps on Virtex-5 XC3S50 FPGA. In [19] FPGA-based implementation of hash functions KECCAK-200 and KECCAK-400, PHOTON and SPONGENT are presented. The comparison shows that the SPONGENT implementation provided the highest throughput per area ratios. In [20] various fault diagnosis approaches for hash-based post-quantum signatures are proposed with ASIC implementation. For the inner hash function SPONGENT, the authors have presented several diagnosis methods that are capable of reaching high error coverage with an acceptable area overhead. In [21] the two structures of SPONGENT (for 80-bit) algorithm based on look-up table and Shift-AND-OR techniques are presented. In [22] a lightweight authenticated encryption scheme based on the integrated hardware implementation of the PRESENT block cipher and the SPONGENT hash function is presented. Two works QUARK [12] and BLAKE2 [23,24] are the lightweight hash functions. In work [12] the authors present the six ASIC implementations U-QUARK, U-QUARK× 8, D-QUARK, D-QUARK× 8, S-QUARK, and S-QUARK× 16 of QUARK hash function for comparison with other hash functions. The implementations are realized with the area consumed of 1379, 2392, 1702, 2819, 2296, and 4640 gate equivalent (GE) in 180 nm technology, respectively. The lightest implementation is U-QUARK with at least 64-bit security against all attacks such as collisions, multicollisions, distinguishers, etc. S-QUARK× 16 design is larger, in area point of view, than all previous designs with 112-bit security. Also, in work [24] presents implementations for the permutation function of the BLAKE2 algorithm based on combinational and sequential circuits. In this work use from the DADDA multiplier and the carry-save adder (CSA) (to the addition of three and more values) in the structures. The implementations in work [24] are synthesized in 90 nm technology, with results for area consumption ranging from 14130 GE to 52820 GE, throughput ranging from 97 Mbps to 2426 Mbps, and power consumption ranging from 0.384 mW to 2.104 mW.

3. SPONGENT lightweight hash function

An important class of one-way functions, for cryptography, is the one-way hash functions, that operate on bit strings and map bit strings of arbitrary length to a bit string of a fixed length which is called the hash value [25]. SPONGENT is a lightweight hash function based on sponge construction with a wide PRESENT-type permutation [10,11]. It produces an n-bit hash value, where $n \in \{88, 128, 160, 224, 256\}$. The sponge construction is a simple iterative structure that takes a variable-length input and can produce an output of an arbitrary length based on a permutation π_b operating on a state of a fixed number b of bits. The size of the internal state $b = r + c \geq n$ is called width, where r and c are the rate and the capacity, respectively. The sponge construction starts with an *Initialization phase*. In this phase, the message m is padded by a single bit 1 followed by a necessary number of '0' bits up to a multiple of r bits, for example, if $r = 8$, then the 1-bit message '1' is transformed to ''11000000". Then, it is cut into blocks of r bits. The second phase is called *Absorbing phase*. The r-bit input message blocks m_i are added (XORed) into the first r bits of the state, interleaved with applications of the permutation π_b. After the *Absorbing phase*, we have *Squeezing phase*. The first r bits of the state are returned as output, interleaved with applications of the permutation π_b, until n bits are returned. In the SPONGENT hash function, the b-bit '0' is taken as the initial value before the absorbing phase. The message parts are XORed into the r rightmost bit positions of the state. The same r-bit positions form parts of the hash output.

Fig. 1 shows the sponge construction based on a b-bit permutation π_b. The structure consists of two phases absorbing and squeezing. The m_i are the r-bit message blocks (the input message is padded and divided into blocks of r-bit) and h_i are parts of the hash value. Its size is determined from the size of b. After processing all message blocks, the squeezing phase starts. The several structures of SPONGENT based

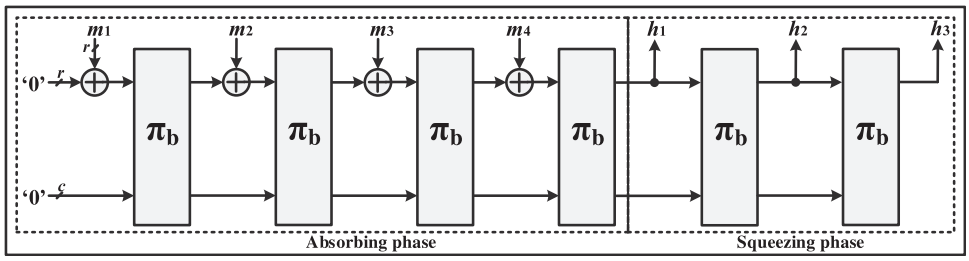

Fig. 1. Sponge construction based on a b-bit permutation π_b.

Table 1
13 cases of the SPONGENT hash function.

Number	Cases	n	b	c	r	R Number of rounds	LFSR size	LFSR initial value
C1	SPONGENT-\88\80\8	88	88	80	8	45	6	0 × 05
C2	SPONGENT-\88\176\88	88	264	176	88	135	8	0xC6
C3	SPONGENT-\128\128\8	128	136	128	8	70	7	0 × 7A
C4	SPONGENT-\128\256\128	128	384	256	128	195	8	0 × FB
C5	SPONGENT-\160\160\16	160	176	160	16	90	7	0 × 45
C6	SPONGENT-\160\160\80	160	240	160	80	120	7	0 × 01
C7	SPONGENT-\160\320\160	160	480	320	160	240	8	0 × A7
C8	SPONGENT-\224\224\16	224	240	224	16	120	7	0 × 01
C9	SPONGENT-\224\224\112	224	336	224	112	170	8	0 × 52
C10	SPONGENT-\224\448\224	224	672	448	224	340	9	0 × 105
C11	SPONGENT-\256\256\16	256	272	256	16	140	8	0 × 9E
C12	SPONGENT-\256\256\128	256	384	256	128	195	8	0xFB
C13	SPONGENT-\256\512\256	256	768	512	256	385	9	0 × 015

Algorithm 1 Permutation π_b Algorithm

Input: b-bit state S.
Output: b-bit output K.
1. **For** i **from** 0 **to** $R-1$ **do**
2. $S \leftarrow RSFL_b(i) \oplus S \oplus LFSR_b(i)$;
3. $S \leftarrow S - box_b(S)$;
4. $S \leftarrow Permutation_b(S)$;
5. **End For**;
6. **Return** $K \leftarrow S$;

on various parameterizations as SPONGENT-\$n\c\r$ for different hash sizes n, capacities c, and rates r are presented in [11]. 13 SPONGENT cases (C1 to C13) are shown in Table 1. These cases provide the five different hash output lengths with multiple security levels.

All SPONGENT cases with the same output size of n bits are referred to as SPONGENT-n. The SPONGENT-88 cases are designed for extremely restricted applications such as RFID protocols and low preimage security requirements. The SPONGENT-128 and SPONGENT-160 can be used in highly constrained applications with low and middle requirements for collision security. The parameters of SPONGENT-224 and SPONGENT-256 to make SPONGENT compatible with the standard interfaces in the usual lightweight embedded applications.

3.1. Permutation π_b

The permutation π_b is an R-round transform of the input state S of b bits that can be presented by Algorithm 1. In this algorithm, only widths b with $4|b$ are allowed. The number of rounds R depends on block size b (see also Table 1). $LFSR_b(i)$ is the state (value) of an LFSR dependent on b at round i that yields the round constant in round i and is added to the rightmost bits of S. $RSFL_b(i)$ is the value of $LFSR_b(i)$ with its bits in reversed order and is added to the leftmost bits of S.

3.1.1. S-box

The SPONGENT hash function uses a 4-bit S-box so that the input nibble $x_3x_2x_1x_0$ is substituted by the output nibble $y_3y_2y_1y_0$. The

Table 2
The values of 4-bit S-box used in SPONGENT.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
y	e	d	b	0	2	1	4	f	7	a	8	5	9	c	3	6

SPONGENT S-box is presented in Table 2. In the SPONGENT for b-bit permutation operation, we require $b/4$ 4-bit S-boxes for round computations.

3.1.2. $Permutation_b$

The bit-permutation $Permutation_b$ is an extension of the PRESENT bit-permutation and transforms bit j of state S to bit position $P_b(j)$ as follows:

$$P_b(j) = \begin{cases} j \times \frac{b}{4} \ mod \ b-1 & if \ j \in \{0,1,\ldots,b-2\} \\ b-1 & if \ j = b-1. \end{cases}$$

3.1.3. $LFSR_b$

In the SPONGENT hash function we have four $L = \lceil log_2 R \rceil$ bits LFSRs, where $L \in \{6,7,8,9\}$. If λ is the root of unity in the corresponding binary finite field, the L-bit LFRSs defined by the polynomials $\lambda^6+\lambda^5+1$, $\lambda^7+\lambda^6+1$, $\lambda^8+\lambda^4+\lambda^3+\lambda^2+1$, $\lambda^9+\lambda^4+1$ for 6-bit, 7-bit, 8-bit and 9-bit LFSRs in the SPONGENT variants, respectively. The initial values of all the LFSRs were presented in Table 1.

4. Proposed structures of the SPONGENT hash function

Hardware structures of the SPONGENT hash function are presented in this section. Two structures called full data-path and 4-bit serialized are proposed. For area-constrained applications, the structure 4-bit serialized is an appropriate choice. The full data-path width structure can used for high-speed and high-throughput applications.

Table 3
Proposed computations of the 4-bit S-box.

Output of S-box	Terms
y_0	$((x_2'x_1) \oplus x_0 \oplus x_3)$
y_1	$x_3x_1'x_0' + (x_2x_1 \odot (x_3 + x_0))$
y_2	$(x_2 \odot x_1)x_3' + x_3x_0(x_2 + x_1) + x_2'x_1'x_0'$
y_3	$((x_2 \oplus x_0)x_1 + ((x_2 \odot x_1)x_2' + x_1x_0)x_3 + x_3'x_2x_1')$

4.1. Proposed full data-path structure of the SPONGENT hash function

The full data-path width structure is implemented based on three 2-to −1 multiplexers with r-bit width, one L-bit LFSR ($L \in \{6, 7, 8, 9\}$), three addition operations, reversed order block, permutation block, $b/4$ 4-bit S-boxes, and one b-bit register R. The operations are started by control signals S_0='1' and S_1='1' for one clock cycle. In this clock cycle, the zero initial values and m_0 are applied to the circuit. At the other clock cycles (clock cycles 2 to R) the control signals S_0 and S_1 are set to '0'. After R clock cycles, one permutation π_b is computed. In the *Absorbing* phase for start of each step (computation of addition with m_i and permutation π_b) the control signals of S_0 and S_1 are set to '1' (for one clock cycle). On the other hands, for the implementation of *Squeezing* phase, these control signals in all clock cycles are set to '0'. Because we have only the permutation π_b step in the *Squeezing* phase. The reversed order block generates $RSFL_b(i)$ bits from $LFSR_b(i)$ bits, $0 \le i \le L-1$. It is implemented based on the wired structure without extra hardware resources. The structure of reversed order block for 6-bit LFSR is shown in Fig. 2.

4.1.1. Structure of the 4-bit S-box

The main difference between this work and work [11] is the design of the S-box. In the proposed structure, the low-cost 4-bit S-box with comparable area consumption is proposed. The critical path delay and area consumption of the SPONGENT structure depend on S-box. The most complex block in the SPONGENT hash function is the 4-bit S-box. Therefore, we present an Area×Delay efficient circuit for implementation of this block. The proposed computations of the 4-bit S-box are shown in Table 3. Fig. 3(a) shows the structure of the 4-bit S-box. As seen from this figure, the S-box block is constructed using only 28 logic gates, with a low critical path delay structure. The circuit is simple and low-cost for hardware implementation. The critical path delay of S-box in the circuit is equal to $T_{NX}+2T_A+T_O+T_{NO}$, where T_{NX}, T_O, T_{NO} and T_A are the time delay of the 2-input XNOR gate, 2-input OR gate, 2-input NOR gate, and the 2-input AND gate, respectively. The hardware consumption of the proposed 4-bit S-box is reduced to 14 2-input AND gates, 7 2-input OR gates, 3 2-input XOR gate, 3 2-input XNOR gate, 1 2-input NOR gates, and 4 NOT gates.

The optimized structure of the S-box based on NAND and NOR gates is shown in Fig. 3(b). The implementation by using NAND and NOR gates is based on the rules $x \oplus y = x' \oplus y'$, $x \odot y = x' \odot y'$, $x' + y' = (xy)'$ and $x'y' = (x + y)'$. Because the 2-input NAND and NOR gates have lower area and delay than those of the 2-input AND and OR gates. In the gate equivalents (GE) measurement, in 180 nm CMOS technology, the area consumption of Fig. 3(a) and (b) are equal to 44 GE and 35 GE, respectively. The critical path delay of Fig. 3(b) is equal to $T_X + 4T_{NO}$, where T_X and T_{NO} are the time delay of the 2-input XOR gate and the 2-input NOR gate, respectively. Table 4 shows the hardware results of the proposed structures of 4-bit S-box and related works. As seen from the table, the proposed 4-bit S-box (Fig. 3(b)) has acceptable hardware resources and timing characteristics compared to other 4-bit S-boxes.

4.2. Proposed 4-bit serial structure of the SPONGENT hash function

The original structure of the SPONGENT hash function has a b-bit data-path [11]. In this case, we need to $\lceil \frac{b}{4} \rceil$ 4-bit S-boxes in the round computations. Given that the S-box is the most complex block in the SPONGENT thus the implementation of structure based on a b-bit data-path (full data-path) has higher hardware resources than that of the structure with the lower number of bit data-path. On the other hands, for area-constrained applications, the structure with a lower data-path width is an appropriate choice. Here, we describe the proposed structure for the low-cost implementation of the SPONGENT hash function. For example, Algorithms 2 and 3 show the proposed 4-bit serialized SPONGENT hash function for cases 6-bit and 8-bit LFSRs, respectively. In these algorithms, the state S is divided to $b/4$4-bit words. The words of the state S are denoted by $S(i)$, $0 \le i \le b/4 - 1$. The $S(0)$ is the least significant word of S.

The proposed structure has a 4-bit data-path is equal to the width of a single S-box. This value is the divisor of the original size of the data-path i.e. $b/4$-bit. The proposed low-cost 4-bit serialized structure of SPONGENT hash function is shown in Fig. 4. The structure is composed based on blocks such as S-box, Shift_Reg_Round block, permutation operation, addition operations, L-bit LFSR, 4-bit converter blocks, and several multiplexers. At each clock cycle, one word of the state S is processed. The block Shift_Reg_Round is a multi-task shift register for processing of 4-bit words of data. The micro-architecture of Shift_Reg_Round block connected to the permutation block is shown in Fig. 5. The structure is constructed based on $b-1$ registers called RR_1 to RR_{b-1} and b 2-to −1 multiplexers. This circuit can performs two operating modes (dual-mode circuit). The first mode is round computations with S_4 ='0' and in the second mode permutation operation is performed with S_4 ='1'. The 4-bit output words of permutation (from the most significant word to the least significant word) are saved

Table 4
Hardware results of the proposed structure of 4-bit S-box and related works.

Works	# AND (or OR)	# NAND (or NOR)	# XOR (or XNOR)	CPD
[26], SS_0	26	–	3	$T_A + T_X + 2T_O$
[26], SS_1	31	–	1	$3T_A + 2T_O$
[26], SS_2	34	–	1	$2T_A + 3T_O$
[26], SS_3	36	–	1	$3T_A + 2T_O$
[27]	–	4	4	$2T_X + 2T_{NO}$
[28]	–	4	4	$2T_X + 2T_{NO}$
[29]	20	–	7	$T_X + T_A + 2T_O$
[30]	4	–	4	$2T_X + T_A + T_O$
[31]	4	–	4	$4T_X + 4T_A$
TW	–	22	6	$T_X + 4T_{NO}$

TW: This work; T_A, T_{NA}, T_X, T_{XN}, T_O, T_{NO}, T_M denote the time delay of a 2-input AND gate, 2-input NAND gate, 2-input XOR gate, 2-input XNOR gate, 2-input OR gate, and 2-input NOR gate, respectively.

Algorithm 2 Proposed Serialized Permutation π_b Algorithm for 6-bit LFSR Case

Input: 4-bit parts ($S(i)$, $0 \le i \le \frac{b}{4} - 1$) of state S.
Output: 4-bit output $K(i)$, $0 \le i \le \frac{b}{4} - 1$.
1. **For** j **from** 0 **to** $R-1$ **do**
2. **For** i **from** 0 **to** $\frac{b}{4} - 1$ **do**
3. **If** $i = 0$ **then**
4. $S(i) = S(i) \oplus LFSR[3:0](j)$;
5. **Elseif** $i = 1$ **then**
6. $S(i) = S(i) \oplus \}\}00\varepsilon||LFSR[5:4](j)$;
7. **Elseif** $i = (\frac{b}{4} - 2)$ **then**
8. $S(i) = S(i) \oplus RSFL[0:1](j)||\}\}00\varepsilon$;
9. **Elseif** $i = (\frac{b}{4} - 1)$ **then**
10. $S(i) = S(i) \oplus RSFL[2:5](j)$;
11. **Else**
12. $S(i) = S(i)$;
13. **End If**;
14. $S(i)$=S-box($S(i)$);
15. **End For**;
16. S=Permutation(S);
17. **End For**;
19. **Return** $K(i) = S(i)$;

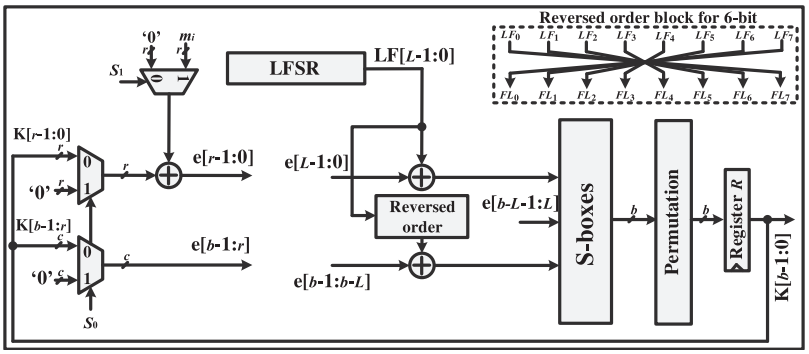

Fig. 2. The proposed full data-path structure of SPONGENT hash function.

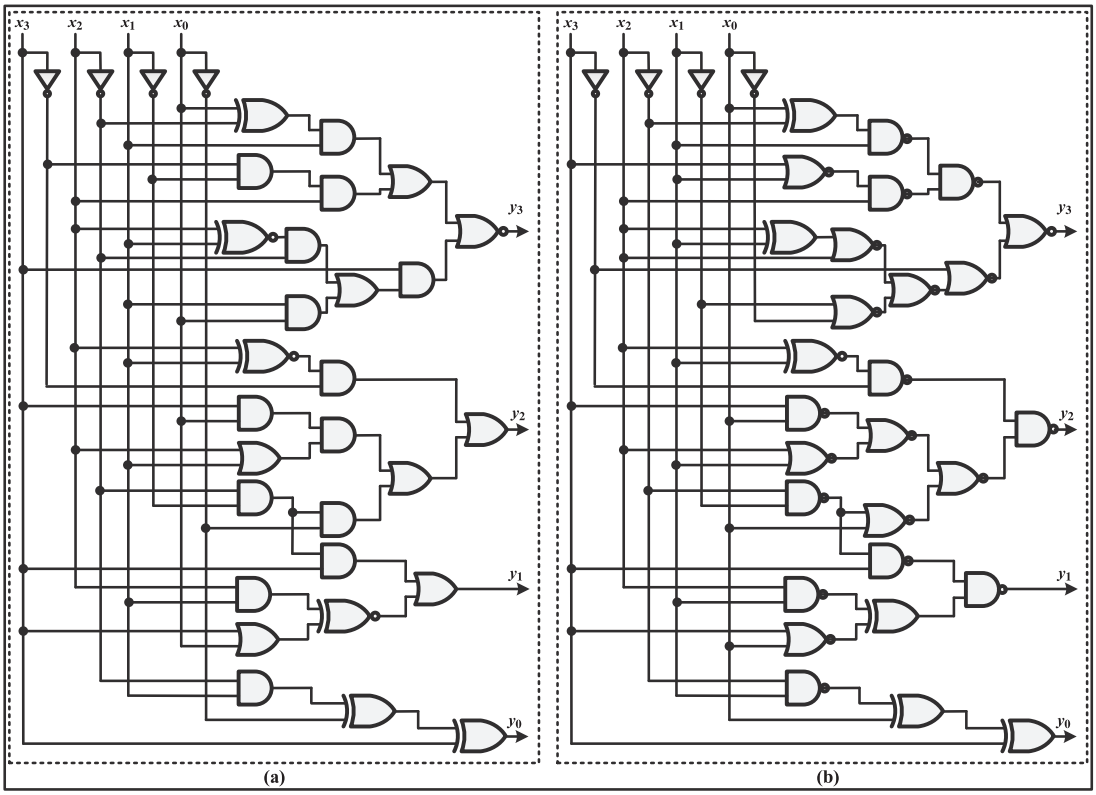

Fig. 3. Proposed structure of S-box in the SPONGENT hash function (a) and the optimized structure based on NAND and NOR gates (b).

in registers RR_1 to $RR_{b/4-1}$, respectively. At the first clock cycle, the control signals (S_0, S_1), and (S_2, S_3) are set to '1' and '0', respectively, and the first zero word of initial value and the first 4-bit part of the LFSR are applied to the structure. In the next clock cycles of the round computations, the intermediate 4-bit words are shifted until all words are processed. In the 4-bit data-path, each round is computed at the $b/4$ clock cycles. In this case, at each clock cycle, one 4-bit data is computed and stored in the Shift_Reg_Round block. After $b/4$ clock cycles, the b-bit data which is stored in registers R, RR_1 to $RR_{b/4-1}$ are applied to the permutation block for the start of the next round computations. During round computations, the control signal S_4 is set to '0' for the shift of 4-bit data to storing into the registers RR_1 to $RR_{b/4-1}$. At the end of round computations, the control signal S_4 is set to '1' for performing permutation operation. Therefore, the stored data in the RR_1 to $RR_{b/4-1}$ are permuted and stored again in these registers. The number of clock cycles for the proposed 4-bit serialized structure is equal to $b/4 \times R$.

In the proposed structure, we have two blocks called 4-bit converter 1 and 4-bit converter 2, for convert the output bits of LFSR and RSFL to 4-bit data, respectively. These blocks are implemented based on 2-to −1 multiplexers. For example, consider 7-bit LFSR case, the output bits of LFSR are denoted by LF_6, LF_5, LF_4, LF_3, LF_2, LF_1, LF_0. The 4-bit converter is controlled by the S_5 control signal. In the first step, the control signal S_5 is set to '0' and the 4-bit outputs of the converter block $F_3F_2F_1F_0$ are equal to $LF_3LF_2LF_1LF_0$. In the next step, this control signal is set '1' and we have $F_3F_2F_1F_0 = 0LF_6LF_5LF_4$(see Fig. 6).

From the hardware point of view, the proposed 4-bit serialized structure of the SPONGENT hash function for case SPONGENT-\88\80\8 as an example for Fig. 4 is presented in Fig. 8. In this structure, only one S-box is used in the 4-bit data-path. The $b/4$ and L parameters are equal to 22 and 6, respectively. Therefore, the part of the round computation is consist of 21 4-bit registers RR_1 to RR_{21} and register R (see Fig. 7).

5. Security analysis of the proposed structures

The security analysis in the work [11] is performed based on theoretical analysis. The proposed structures have the same original

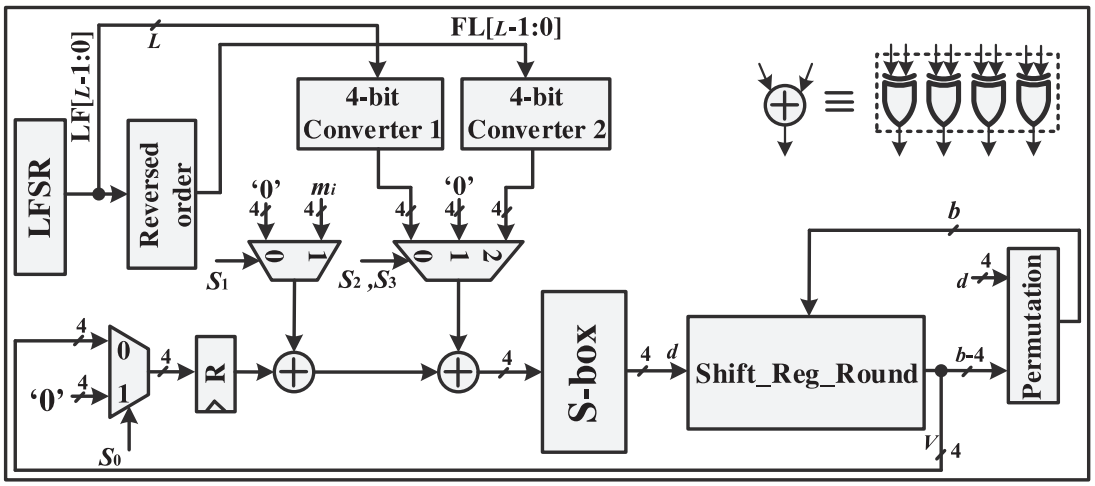

Fig. 4. The proposed low-cost 4-bit serialized structure of SPONGENT hash function.

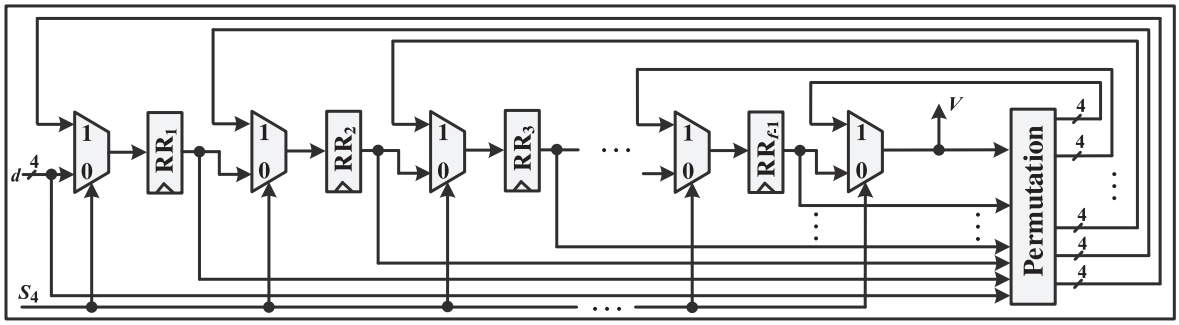

Fig. 5. The proposed micro-architecture of the Shift_Reg_Round block connected to permutation block.

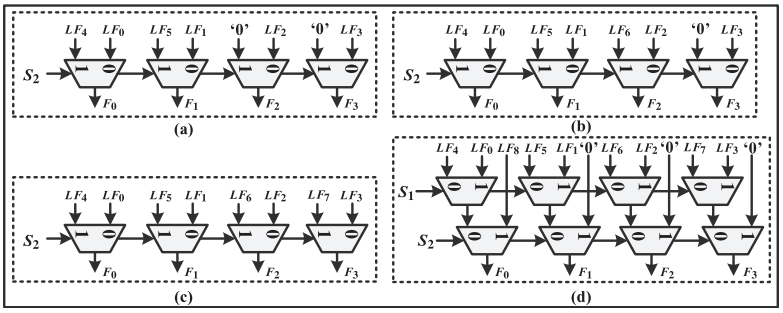

Fig. 6. Structure of the 4-bit converter 1 for 6-bit (a), 7-bit (b), 8-bit (c), and 9-bit (d) LFSRs.

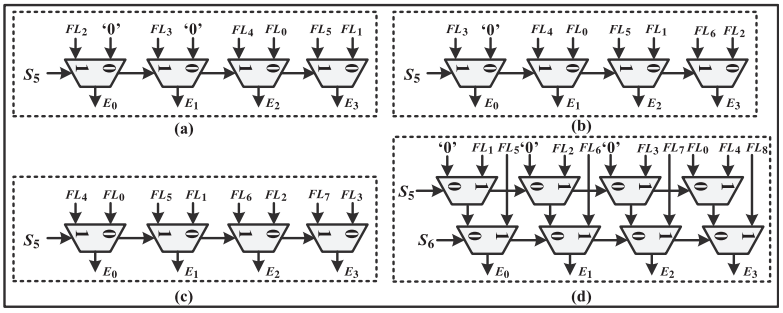

Fig. 7. Structure of the 4-bit converter 2 for 6-bit (a), 7-bit (b), 8-bit (c), and 9-bit (d) RSFLs.

SPONGENT hash function algorithm. Therefore, the resistance of the proposed structures against different attacks such as collision attack and linear attack is the same as work [11] resistance. The main focus of this work is the design and implementation of low-cost and high-speed

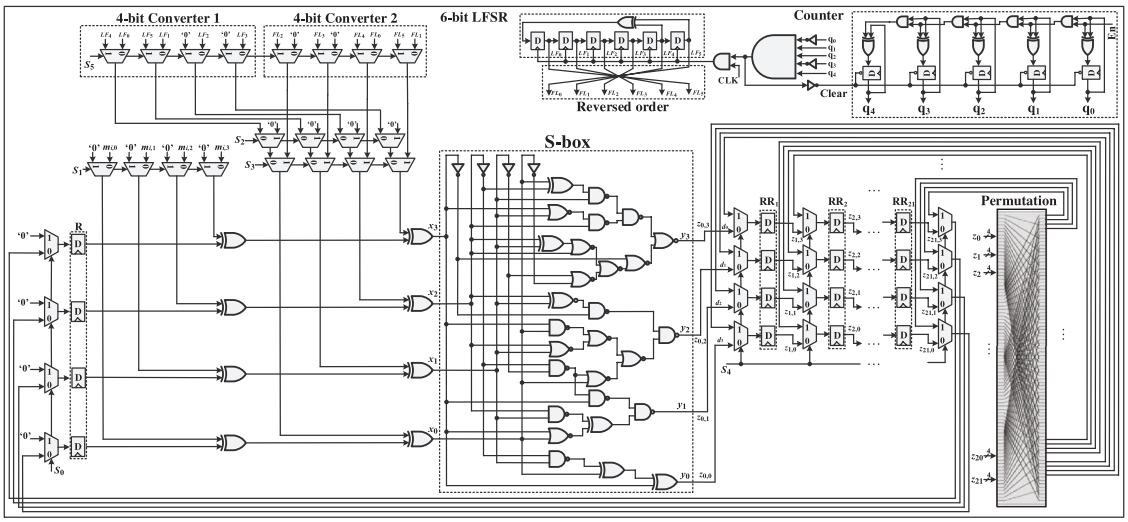

Fig. 8. The proposed 4-bit serialized structure of SPONGENT hash function for case SPONGENT-\88\80\8.

Algorithm 3 Proposed Serialized Permutation π_b Algorithm for 8-bit LFSR Case

Input: 4-bit parts ($S(i)$, $0 \le i \le \frac{b}{4} - 1$) of state S.
Output: 4-bit output $K(i)$, $0 \le i \le \frac{b}{4} - 1$.
1. **For** j **from** 0 **to** $R - 1$ **do**
2. **For** i **from** 0 **to** $\frac{b}{4} - 1$ **do**
3. **If** $i = 0$ **then**
4. $S(i) = S(i) \oplus LFSR[3:0](j)$;
5. **Elseif** $i = 1$ **then**
6. $S(i) = S(i) \oplus LFSR[7:4](j)$;
7. **Elseif** $i = (\frac{b}{4} - 2)$ **then**
8. $S(i) = S(i) \oplus RSFL[0:3](j)$;
9. **Elseif** $i = (\frac{b}{4} - 1)$ **then**
10. $S(i) = S(i) \oplus RSFL[4:7](j)$;
11. **Else**
12. $S(i) = S(i)$;
13. **End If**;
14. $S(i)$=S-box($S(i)$);
15. **End For**;
16. S=Permutation(S);
17. **End For**;
19. **Return** $K(i) = S(i)$;

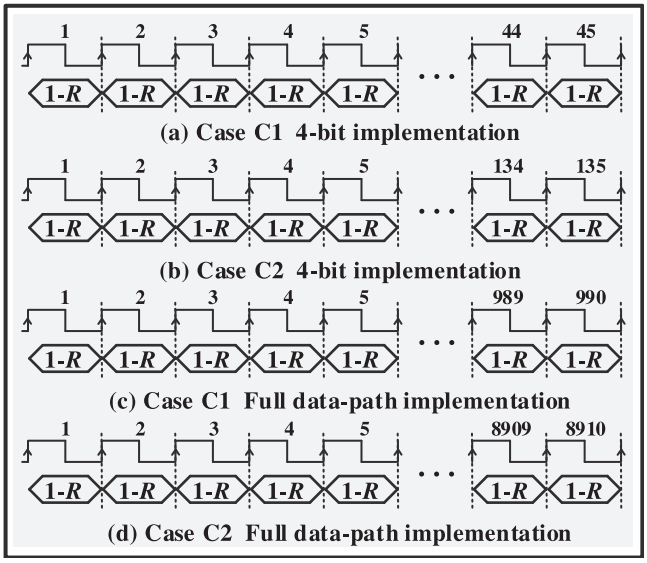

Fig. 9. The computations (number of rounds) of SPONGENT hash function at each clock cycle for cases (a) C1 and (b) C2.

hardware structures of the SPONGENT hash function. However, we analyze the security of structures from a hardware point of view. Side-channel attacks and fault injection attacks are the most threats to the security of cryptographic algorithms. These attacks are used to recover sensitive data such as the main key and plaintext (message). Fault injection attacks are non-invasive active attacks that insert any kind of malfunction on the operation during encryption, using this wrong result to retrieve the secret key of a device. Side-channel attacks on cryptographic devices are non-invasive passive attacks that use certain physical information leaked during normal encryption such as power consumption [32], time delay [33], or electromagnetic radiation [34] to find the secret key. Differential power analysis (DPA) [32] is a power consumption side-channel attack that divides the encryption into several time slots and measures power in each slot for different plaintext input. It can be used for cryptographic implementations (such as lightweight block ciphers) to recover the secret key (or message in hash functions). Simple Power Analysis is a method that interprets power consumption measurements during cryptographic operations. It can achieve information about a device's operation as well as the secret key (or message) based on a power trace. Correlation Power Analysis [35] has advantages like the need for less number of power traces and lack of ghost peak problem.

5.1. Power analysis of the proposed hardware structures

In the proposed structures, at each clock cycle, we have the computation of operations with a similar hardware complexity. In this case, the power consumption at each clock cycle is almost constant. In the proposed structures, the same operations are performed at each clock cycle. For example, at each clock cycle, for the full data-path structure, we have the computation of a round function. In each round, the computation of S-boxes and permutation layer is performed. Figs. 9 (a), (b), (c), and (d) show the computations (number of rounds) of SPONGENT hash function for cases C1, C2, C3, and C4, respectively. As seen from these figures, at each clock cycle the number of computed rounds are similar (1-R 1 rounds at one clock cycle). In this case, the power consumption at each clock cycle is fixed. Therefore, this feature leads to a unified power trace in total clock cycles and the power traces are independent of the message patterns (do not change when the input differs).

The power analysis of the proposed structures based on a hardware measurement is performed. The cryptographic device is connected to

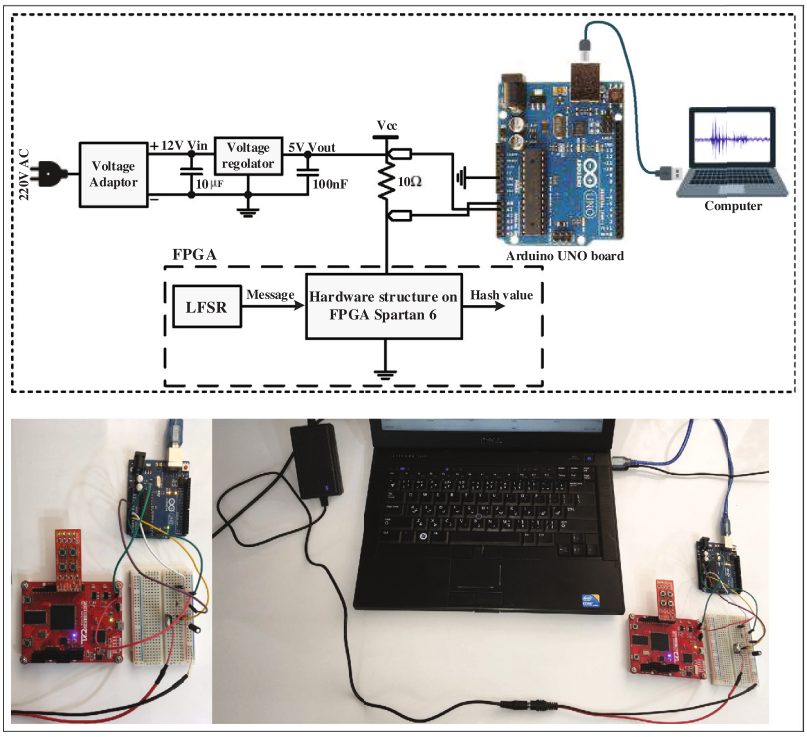

Fig. 10. The experimental setup for the power analysis attack of the proposed structures based on Spartan-6 XC6SLX9 FPGA.

a computer by using an Arduino UNO board [36], where the power traces are captured by using the Arduino board. While the hash process is happening, the waveform that depicts the power consumption of the device is captured and monitored on the computer. Fig. 10 shows the experimental setup based on Spartan-6 XC6SLX9-2TQG144 FPGA as the cryptographic device and an Arduino UNO board for capturing power traces. To compute the power consumption, we used a 10 Ω resistor on the path to the power supply for measurement of the power. The computer is used for communication and capturing data from the Arduino UNO board and also monitoring the power traces. The power traces must be captured for several hundred messages. Therefore, a set of messages must be generated. A Linear Feedback Shift Register (LFSR) is used to generating a set of message for computation hash function.

The cryptographic device and the Arduino board are appropriately connected with a $V_{cc} = 5V$ resistor of 10 Ω selected as the power measurement circuit. The power is measured by connecting the analog-to-digital converter (ADC) pins (A0 and A1) of the Arduino board to the resistor. To achieve a better result, we analyze the hundreds of power traces based on the generated random message by LFSR. For example, the power traces of the proposed structure of the 4-bit serialized SPONGENT hash function (case C1) for different messages (input patterns) are shown in Figs. 11. Also, for the full data-path width of the SPONGENT hash function (case C1), Figs. 12 show the power traces. Therefore, based on the measured power trances from the experimental setup for different message patterns, the power traces have a unified waveform without picks on power trace. In this case, the power traces are independent of the message patterns and we cannot achieve the information about the bit pattern of messages from power measurements during hash operations.

5.2. Timing analysis attack

Another important side-channel attack is the timing attack. In this attack, the time taken to execute hash algorithms is measured precisely by the attacker [33]. If the algorithm execution time for different messages is different, this will lead to the attacker obtaining information about the bit-pattern of a message. Therefore, the implementation of the hash algorithm must reduce data-dependent timing information. The computation time for each hash operation of the proposed 4-bit serialized and full data-path width structures is fixed. It is independent of the message. In this case, the structures leak no information about the bit pattern of the message bit pattern. The computation of one permutation π_b of the messages M_1 to M_n takes the same time t_1 and t_2 for the 4-bit serialized and full data-path width structures, respectively. Fig. 13 shows the waveform of proposed structures for the computation of one permutation π_b of the messages in the full data-path structure. As seen in this figure, the permutation π_b of the messages $(M_1, M_2, \ldots, M_n)$ takes the same time t_2. For example, the timing results for the proposed 4-bit serialized and full data-path width structures of the SPONGENT hash function are shown in Table 5. Here, we measure the time for 50 separate messages $(M_1, M_2, \ldots, M_{50})$. Here, some samples are shown in Table 5. The computation time of one permutation π_b for each of the 50 measurements for the proposed 4-bit serialized and full data-path width structures (for case C1) is equal to 881.1 ns and 42.3 ns, respectively on 180 nm CMOS technology. Therefore, the computation time of the proposed architectures does not change when the bit pattern of the message changes. The computations' timings are independent of the message being manipulated. In this case, the details of the internal computations of the hash algorithm are hidden.

6. Results and comparison

The results of hardware implementation and comparison with other previous works are presented in this section. The implementations of the SPONGENT hash function have been synthesized for the 180 nm CMOS technology in the Synopsys design compiler. The area results are given in gate equivalents (GEs), which represent the number of NAND gates used to implement the algorithms (1 GE represents an area equivalent to a 2-input NAND gate). The results of the hardware implementation for the proposed full data-path structure and 4-bit serialized structure are shown in Tables 6 and 7, respectively. In

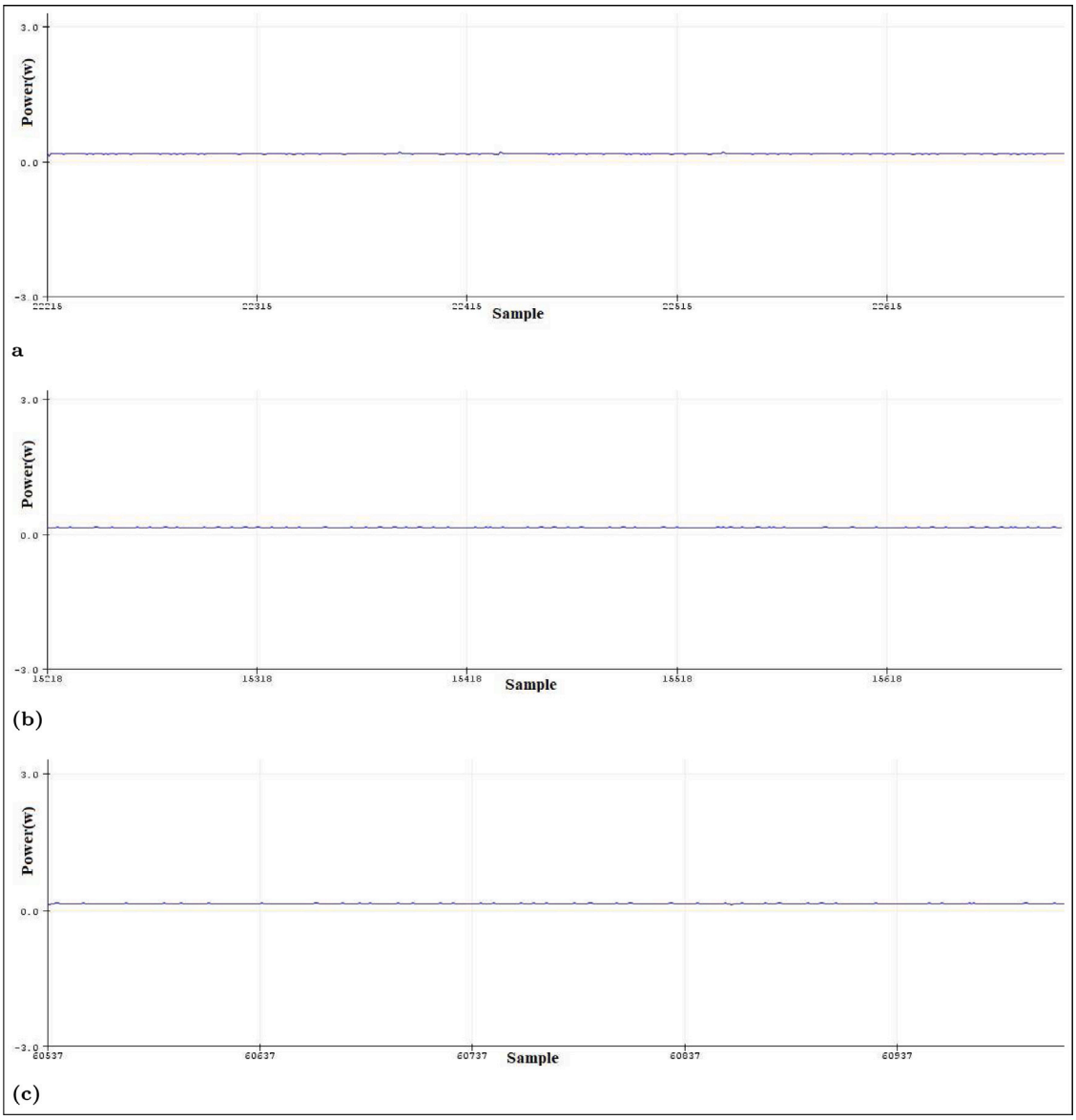

Fig. 11. Power traces of the proposed structure of 4-bit serialized SPONGENT hash function (case C1) for different messages (input patterns).

Table 5
Some samples of hardware results of the proposed 4-bit serialized and full data-path width structures of SPONGENT hash function (for case C1).

Message (Hex)	Hash value	t_1	t_2
M_1=50726573656E74202B2050726573656E74203D2053706F6E67656E	65A551976030022EB8ED6EDB3870F328	881.1 ns	42.3 ns
M_2=42616872616D202B2052617368696469203D204241485252415348	C5D5C88612AA3EF133237903EFE067CA	881.1 ns	42.3 ns
M_3=53616861722B2052617368696469203D2073616D616E6900000000	DFE691FACA21E18E620884DFC077946F	881.1 ns	42.3 ns
M_4=616E204C6967687477656967687420436970686572206861736820	36B8821DFE669741CEFEA3CD5E3CA190	881.1 ns	42.3 ns
M_5=49742063616E2070726F647563652038382D6269742C203132382D	C87289423C5174EEC064C57757B89E39	881.1 ns	42.3 ns
…	…	…	…
M_{46}=6368616E67696E672074686520616C676F726974686D277320636F	C358E7801BA014F81F34BEBBB3FE4396	881.1 ns	42.3 ns
M_{47}=54686520666F6C6C6F77696E672061726520736F6D65206578616D	FE2CD9FEC43B7E5DF6D38E8B2EB2E3BB	881.1 ns	42.3 ns
M_{48}=5468697320706170657220707 26F706F7365732073706F6E67656E	CD364D4209F6D031DEEDB959069CDAF9	881.1 ns	42.3 ns
M_{49}=466F72206861736820636F646573206F662061206465736972656 4	F3A21DAD5940C2042E15004E5A38B5EC	881.1 ns	42.3 ns
M_{50}=486173682066756E6374696F6E732074686174206172652065666 6	1C17F99DD2F23AF2C2A45285D385775E	881.1 ns	42.3 ns

t_1: Time for the computation of one permutation π_b in the 4-bit serialized structure; t_2: Time for the computation of one permutation π_b in the full data-path width structure.

these tables, the parameters such as area consumption, critical path delay, execution time (for the computation of one permutation π_b), throughput, throughput/area, and power consumption for a frequency of 100 KHz are presented. The throughput is defined as: Throughput = $\frac{\text{Message size (bits)}}{\text{CPD} \times \#\text{Clock cycles}} = \frac{\text{Fmax} \times \text{Message size (bits)}}{\#\text{Clock cycles}}$

In work [11] the throughput is computed at the operation frequency of 100 KHz as follows: Throughput = $\frac{100\ \text{KHz} \times \text{Message size (bits)}}{\#\text{Clock cycles}}$

The parameters critical path delay, time, throughput, and power (Tables 6 and 7) are scaled to 90 nm and 65 nm technologies (besides reported results in 180 nm technology) based on presented scaling

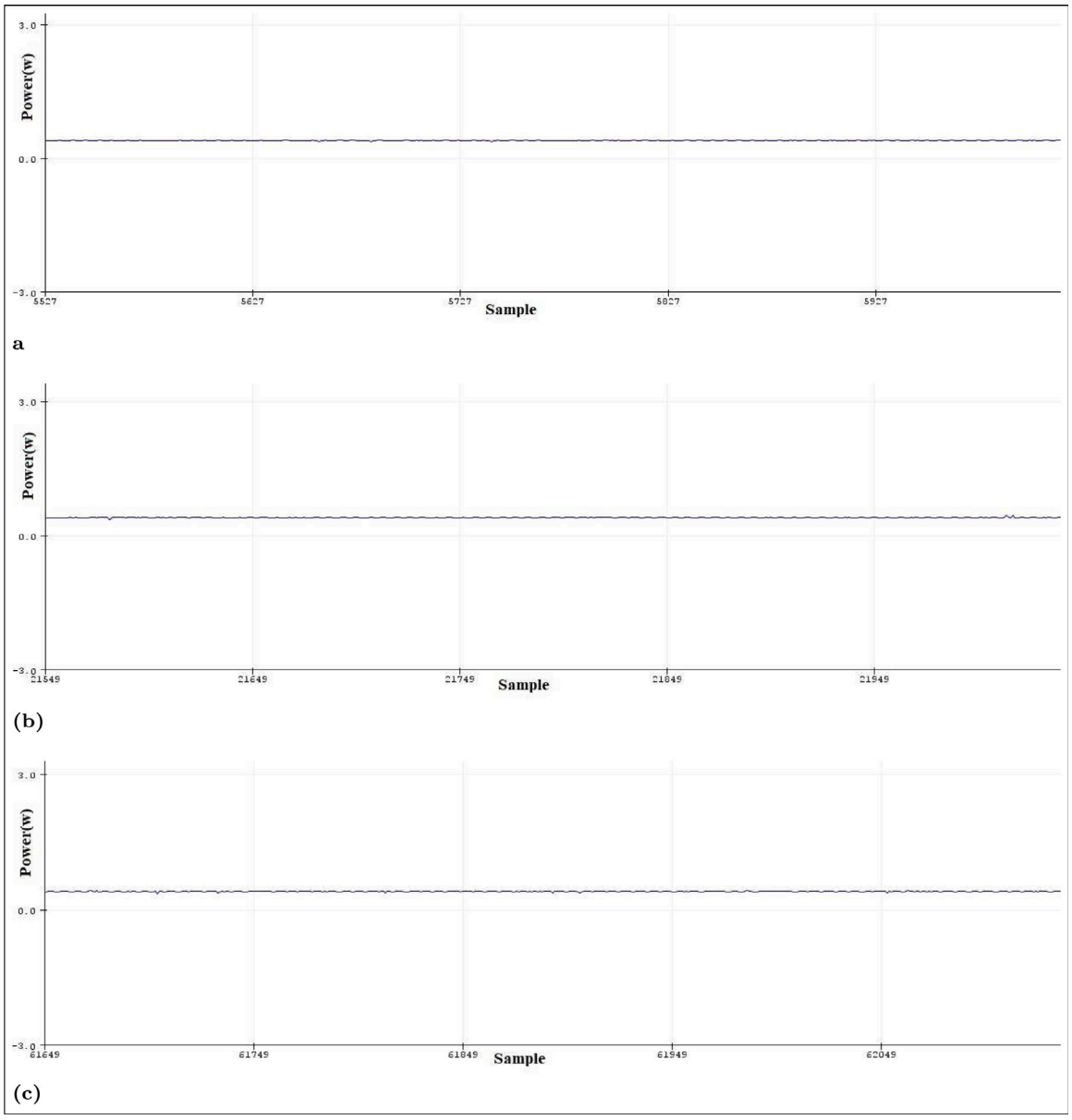

Fig. 12. Power traces of the proposed structure of full data-path width of SPONGENT hash function (case C1) for different messages (input patterns).

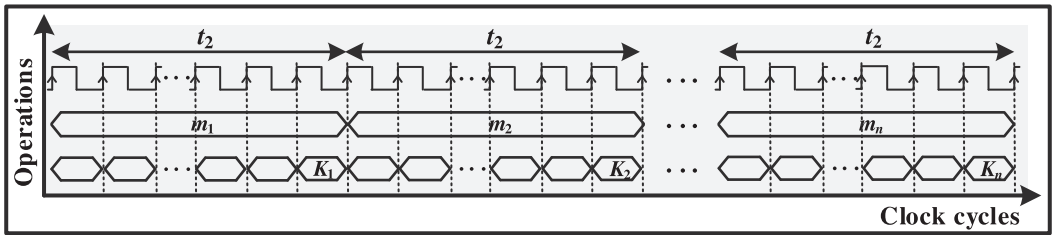

Fig. 13. Waveform of the proposed structures for the computation of one permutation π_b with the same execution time.

methods in works [37,38]. In this case, we have the estimation of these parameters for the proposed method in the two new technologies.

In [11] the several ASIC hardware implementations from the lowest area to the highest throughput of SPONGENT are provided. The work [11] is selected as the baseline for the comparison because this work has reported the results of hardware implementation. The reported hardware results in work [11] are the main reason for this selection. In work [11], the synthesis results such as critical path delay, time, throughput, throughput/area, and power are not reported in more detail. Therefore, the proposed structure in work [11] is re-implemented to get these synthesis results using 180 nm technology. In [20] various fault diagnosis approaches for hash-based post-quantum signatures are proposed with ASIC implementation. For the inner hash function SPONGENT, the authors have presented several diagnosis methods that are capable of reaching high error coverage with an acceptable area overhead. Here, to a fair comparison, we compare the

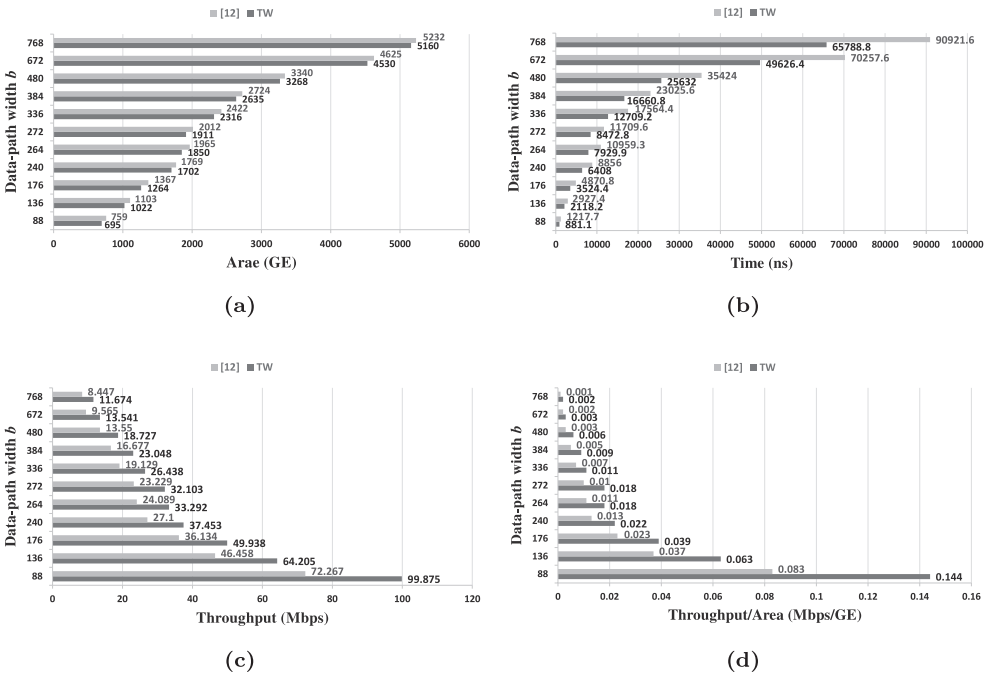

Fig. 14. Column diagrams of the Area (a), Time (b), Throughput (c), Throughput/Area (d) for the proposed 4-bit serialized structure and work [11] for 13 cases of the SPONGENT hash function.

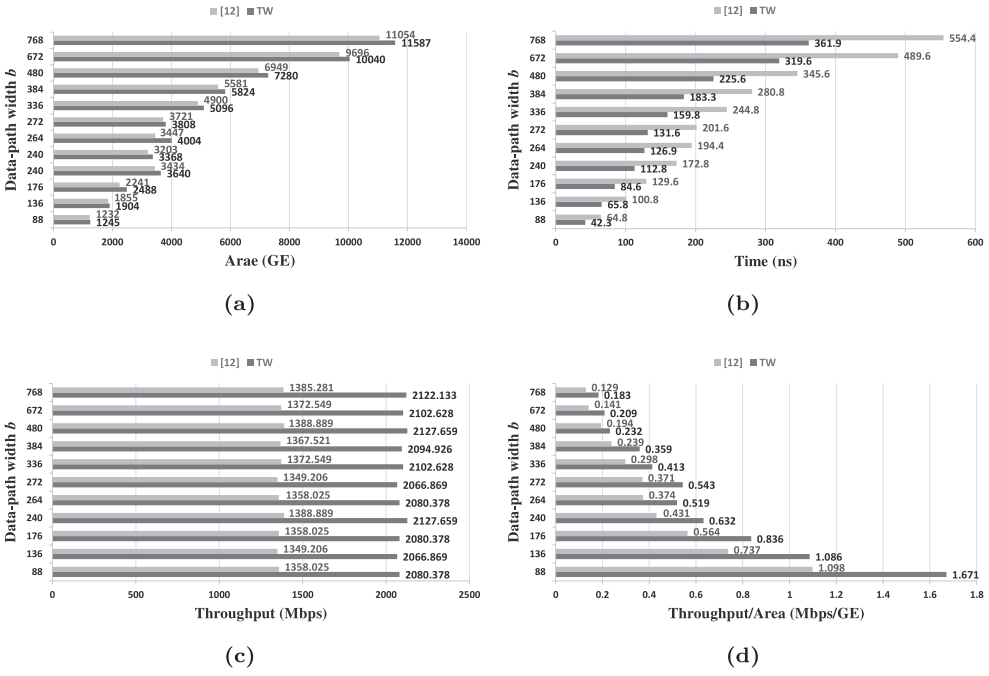

Fig. 15. Column diagrams of the Area (a), Time (b), Throughput (c), Throughput/Area (d) for the proposed full data-path width structure and work [11] for 13 cases of the SPONGENT hash function.

proposed works with the original implementation of SPONGENT (without fault diagnosis) for work [20]. In [21] the two structures of SPONGENT (for 80-bit) algorithm based on look-up table and Shift-AND-OR techniques are presented. The Hardware results of two works [11] and [20] are shown in Table 8. The area consumption (based on GE) in work [11] for cases C1 to C13 are equal to 1232, 3847, 1855, 5581, 2241, 3434, 6949, 3203, 4900, 9696, 3721, 5581, 11054, respectively. Also, for the 4-bit data-path structure, the results of [11] are equal to 759, 1965, 1103, 2724, 1367, 1769, 3340, 1768, 2422, 4625, 2012, 2724, 5232 for cases C1 to C13, respectively. In work [20] the area of original design for cases C1 and C2 are 1412 GE and 1915 GE with throughputs ≃12Kbps and ≃8 Kbps, respectively.

Figs. 14 (a), (b), (c), and (d) show column diagrams of the area, execution time, throughput, and throughput/area, respectively, for the proposed 4-bit serialized structure and work [11] of the 13 cases of SPONGENT hash function. Also, for the case full data-path width these parameters are shown in Fig. 15. Figs. 14(a) and 15(a) shows the comparison of area consumption between this work and work [11]. In the structures with a 4-bit data-path, our work has a lower area than that of work [11]. Based on the hardware results, we get improvements

Table 6
Hardware results of the proposed 4-bit serialized schemes of the SPONGENT hash function on 180 nm CMOS technology.

Works	Area (GE)	CPD (ns)	#Clock cycles	Time (ns)	Thr. Mbps	Thr./Area Mbps/GE	Power (μw)
TW, C1	695	0.89	990	881.1	99.875	0.144	5.89
		(0.29, 0.22)		(287.1, 217.8)	(306.513, 404.040)		(0.84, 0.54)
TW, C2	1850	0.89	8910	7929.9	33.292	0.018	15.27
		(0.29, 0.22)		(2583.9, 1960.2)	(102.171, 134.680)		(1.93, 1.41)
TW, C3	1022	0.89	2380	2118.2	64.205	0.063	8.60
		(0.29, 0.22)		(690.2, 523.6)	(197.044, 259.740)		(1.08, 0.79)
TW, C4	2635	0.89	18720	16660.8	23.048	0.009	22.12
		(0.29, 0.22)		(5428.8, 4118.4)	(70.734, 93.240)		(2.84, 2.04)
TW, C5	1264	0.89	3960	3524.4	49.938	0.039	10.47
		(0.29, 0.22)		(1148.4, 871.2)	(153.257, 202.020)		(1.35, 0.96)
TW, C6	1702	0.89	7200	6408	37.453	0.022	14.26
		(0.29, 0.22)		(2088, 1584)	(114.943, 151.515)		(1.81, 1.32)
TW, C7	3268	0.89	28800	25632	18.727	0.006	26.89
		(0.29, 0.22)		(8352, 6336)	(57.471, 75.758)		(3.41, 2.48)
TW, C8	1702	0.89	7200	6408	37.453	0.022	14.18
		(0.29, 0.22)		(2088, 1584)	(114.943, 151.515)		(1.79, 1.31)
TW, C9	2316	0.89	14280	12709.2	26.438	0.011	19.35
		(0.29, 0.22)		(4141.2, 3141.6)	(81.136, 106.952)		(2.46, 1.78)
TW, C10	4530	0.89	55760	49626.4	13.541	0.003	37.69
		(0.29, 0.22)		(16170.4, 12267.2)	(41.557, 54.780)		(4.78, 3.48)
TW, C11	1911	0.89	9520	8472.8	32.103	0.018	15.81
		(0.29, 0.22)		(2760.8, 2094.4)	(98.522, 129.870)		(2.10, 1.46)
TW, C12	2635	0.89	18720	16660.8	23.048	0.009	21.85
		(0.29, 0.22)		(5428.8, 4118.4)	(70.734, 93.240)		(2.77, 2.02)
TW, C13	5160	0.89	73920	65788.8	11.674	0.002	43.14
		(0.29, 0.22)		(21436.8, 16262.4)	(35.826, 47.226)		(5.47, 3.98)

TW:This work; The values in parentheses are scaled in new technologies 90 nm and 65 nm (90 nm, 65 nm). The power consumption is archived for a frequency of 100 KHz.

Table 7
Hardware results of the proposed full data-path width schemes of the SPONGENT hash function on 180 nm CMOS technology.

Works	Area (GE)	CPD (ns)	#Clock cycles	Time (ns)	Thr. Mbps	Thr./Area Mbps/GE	Power (μw)
[22], C1, 180 nm	967	–	–	–	195.5 Kbps	0.0002	1.5
TW, C1	1245	0.94	45	42.3	2080.378	1.671	8.48
		(0.32, 0.24)		(14.4, 10.8)	(6111.111, 8148.148)		(1.18, 0.83)
TW, C2	4004	0.94	135	126.9	2080.378	0.519	26.78
		(0.32, 0.24)		(43.2, 32.4)	(6111.111, 8148.148)		(3.39, 2.47)
TW, C3	1904	0.94	70	65.8	2066.869	1.086	12.85
		(0.32, 0.24)		(22.4, 16.8)	(6071.429, 8095.238)		(1.63, 1.18)
TW, C4	5824	0.94	195	183.3	2094.926	0.359	39.14
		(0.32, 0.24)		(62.4, 46.8)	(6153.846, 8205.128)		(4.96, 2.61)
TW, C5	2488	0.94	90	84.6	2080.378	0.836	11.03
		(0.32, 0.24)		(28.8, 21.6)	(6111.111, 8148.148)		(1.46, 1.03)
TW, C6	3640	0.94	120	112.8	2127.659	0.585	24.63
		(0.32, 0.24)		(38.4, 28.8)	(6250, 8333.333)		(3.14, 2.21)
TW, C7	7280	0.94	240	225.6	2127.659	0.232	48.91
		(0.32, 0.24)		(76.8, 57.6)	(6250, 8333.333)		(6.21, 4.49)
TW, C8	3368	0.94	120	112.8	2127.659	0.632	22.98
		(0.32, 0.24)		(38.4, 28.8)	(6250, 8333.333)		(2.93, 2.09)
TW, C9	5096	0.94	170	159.8	2102.628	0.413	34.41
		(0.32, 0.24)		(54.4, 40.8)	(6176.471, 8235.294)		(4.36, 3.16)
TW, C10	10040	0.94	340	319.6	2102.628	0.209	67.45
		(0.32, 0.24)		(108.8, 81.6)	(6176.471, 8235.294)		(8.55, 6.2)
TW, C11	3808	0.94	140	131.6	2066.869	0.543	25.92
		(0.32, 0.24)		(44.8, 33.6)	(6071.429, 8095.238)		(3.28, 2.36)
TW, C12	5824	0.94	195	183.3	2094.926	0.359	39.33
		(0.32, 0.24)		(62.4, 46.8)	(6153.846, 8205.128)		(4.97, 3.50)
TW, C13	11587	0.94	385	361.9	2122.133	0.183	78.38
		(0.32, 0.24)		(123.2, 92.4)	(6233.766, 8311.688)		(9.94, 7.05)

TW:This work; The values in parameterize are scaled in new technologies 90 nm and 65 nm (90 nm, 65 nm). The power consumption is archived for a frequency of 100 KHz.

in terms of hardware resources, throughput, and throughput/area for the SPONGENT hash function.

The main differences between the proposed structures and other works for the implementation of SPONGENT lightweight hash function are as follows:

1. Low-cost 4-bit S-box with comparable area consumption is proposed. The S-box is optimized in terms of delay and area. It consumes only 28 logic gates. The S-boxes in work [21] are implemented by look-up table.
2. In order to reduce delay and area, the main part of 4-bit S-box circuit have been implemented by the 2-input NAND and NOR gates.
3. A 4-bit serialized architecture by using one multi-task shift register called Shift_Reg_Round in the round computations is

Table 8
Hardware results of two works [11] (180 nm CMOS technology) and [20] (65 nm CMOS technology).

Works	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
[11], 4-bit Area	869	2264	1257	3183	1572	2066	3931	2070	2827	5430	2323	3138	6163
[11], 4-bit #CCs	990	8910	2380	18720	3960	7200	28800	7200	14280	57120	9520	18720	73920
[11], 4-bit CPD (ns)	1.23	1.23	1.23	1.23	1.23	1.23	1.23	1.23	1.23	1.23	1.23	1.23	1.23
[11], 4-bit Time (ns)	1217.7	10959.3	2927.4	23025.6	4870.8	8856	35424	8856	17564.4	70257.6	11709.6	23025.6	90921.6
[11], 4-bit Thr.(Mbps)	72.267	24.089	46.458	16.677	36.134	27.100	13.550	27.100	19.129	9.565	23.229	16.677	8.447
[11], 4-bit Thr./Area (Mbps/GE)	0.083	0.011	0.037	0.005	0.023	0.013	0.003	0.013	0.007	0.002	0.010	0.005	0.001
[11], 4-bit Power (μw)	5.94	15.37	8.67	22.26	10.78	14.25	27.29	14.48	19.36	37.44	15.80	21.49	43.91
[11], FP Area	1237	3633	1831	5715	2406	3612	7163	3220	4611	9751	3639	5713	10778
[11], FP #CCs	45	135	70	195	90	120	240	120	170	340	140	195	385
[11], FP CPD (ns)	1.44	1.44	1.44	1.44	1.44	1.44	1.44	1.44	1.44	1.44	1.44	1.44	1.44
[11], FP Time (ns)	64.8	194.4	100.8	280.8	129.6	172.8	345.6	172.8	244.8	489.6	201.6	280.8	554.4
[11], FP Thr.(Mbps)	1358.025	1358.025	1349.206	1367.521	1358.025	1388.889	1388.889	1388.889	1372.549	1372.549	1349.206	1367.521	1385.281
[11], FP Thr./Area (Mbps/GE)	1.098	0.374	0.737	0.239	0.564	0.385	0.194	0.431	0.298	0.141	0.371	0.239	0.129
[11], FP Power (μw)	8.36	23.75	12.05	36.87	16.37	24.74	47.14	21.05	31.36	63.32	25.09	37.58	69.08
[20], Area	1412	–	1915	–	–	–	–	–	–	–	–	–	–
[20], Thr.(Kbps)	12	–	8	–	–	–	–	–	–	–	–	–	–

C1 to C13: Case 1 to Case 13 of the SPONGENT hash function; CCs: clock cycles; FP: Full path. In work [11], the synthesis results such as critical path delay, time, throughput, throughput/area, and power are not reported in more detail. Therefore, the proposed structure in work [11] is re-implemented to get these synthesis results using 180 nm technology. The power consumption is archived for a frequency of 100 KHz.

designed. This multi-task shift register can performs two operating modes. The first mode is round computations and in the second mode permutation operation is performed. In the structures with a 4-bit data-path, our work has a lower area than that of work [11].

4. The round computations with b-bit width in the 4-bit serialized structure are implemented in a 4-bit circuit with one S-box.
5. For high-throughput application a full data-path width architecture is proposed. This structure has a higher throughput than that of work [11].

7. Conclusion

Two efficient low-cost and high-throughput structures (full data-path width or parallel structure and 4-bit serialized structure) of the SPONGENT lightweight hash function are presented in this paper. To obtain a low-area implementation with acceptable timing characteristics for SPONGENT, a 4-bit serialized architecture is designed. The serialized architecture is designed by using one multi-task shift register in the round computations with minimum hardware consumption. To improving the timing characteristics, we implement the S-box block as the complex block in the SPONGENT hash function based on an Area×Delay optimized structure. In order to reduce area and delay, a large number of gates have been implemented by 2-input NAND and 2-input NOR gates. The performance measurement of the proposed structures is performed by evaluating the parameters such as area consumption, computation time, critical path delay (CPD), throughput, and throughput/area. The implementation results are achieved for all variants of the SPONGENT hash function in 180 nm CMOS technology. The results of area consumption (for 4-bit serialized structure) and throughput (for full data-path width structure) show improvements compared to previous structures. For area-constrained applications, the proposed structure with a 4-bit data-path width is an appropriate choice with acceptable timing complexities. The full data-path width structure can be used for the high-throughout cryptographic applications.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M.R. Sohizadeh Abyaneh, Security Analysis of Lightweight Schemes for RFID Systems (Dissertation for the Degree of Philosophiae Doctor), Department of informatics university of bergen norway, 2012.

[2] C. Manifavas, G. Hatzivasilis, K. Fysarakis, K. Rantos, Lightweight cryptography for embedded systems- a comparative analysis, in: Proc. 8th Cryptographic Hardware and Embedded Systems-CHES, Springer, Egham, UK, 2013, pp. 333–349.

[3] W. Wang, X. Zhang, Q. Hao, Z. Zhang, B. Xu, H. Dong, T. Xia, X. Wang, Hardware-enhanced protection for the runtime data security in embedded systems, MDPI Electron. 8 (1) (2019) 1–20.

[4] T. Eisenbarth, S. Kumar, C. Paar, L. Uhsadel, A survey of lightweight-cryptography implementations, IEEE Des. Test Comput. 24 (6) (2007) 522–533.

[5] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe, PRESENT: An ultra lightweight block cipher, in Proc. International Workshop on Data Privacy Management, 2007, pp. 450–466.

[6] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, The Simon and Speck Families of Lightweight Block Ciphers, Cryptology ePrint Archive, Report 2013/404, 2013, http://eprint.iacr.orgI2013/404.

[7] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, L. Wingers, The simon and speck lightweight block ciphers, in: Design Automation Conference (DAC), 52nd ACMIEDACIIEEE, IEEE, 2015, pp. 1–6.

[8] J. Borghoff, et al., PRINCE-A low-latency block cipher for pervasive computing applications, in: Proc. 18th International Conference on the Theory and Application of Cryptology and Information Security, ASIACRYPT, in: LNCS, vol. 7658, 2012, pp. 208–225.

[9] J. Guo, T. Peyrin, A. Poschmann, M. Robshaw, The LED block cipher, in: Proc Cryptographic Hardware and Embedded Systems-CHES, 32, Springer, Nara, Japan, 2011, pp. 6–341.

[10] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varıcı, I. Verbauwhede, SPONGENT: A lightweight hash function, in: Proc. Cryptographic Hardware and Embedded Systems-CHES, in: LNCS, vol. 6917 (2011) 312–325.

[11] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varıcı, I. Verbauwhede, SPONGENT: The design space of lightweight cryptographic hashing, IEEE Trans. Comput. 62 (10) (2013) 2041–2053.

[12] J.P. Aumasson, L. Henzen, W. Meier, M. Naya lasencia, Quark: a lightweight hash, J. Cryptol. 26 (2) (2013) 313–339.

[13] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varıcı, I. Verbauwhede, The PHOTON family of lightweight hash functions, in: Proc. CRYPTO: 31st Annual Cryptology Conference, in: LNCS, vol. 6841, Santa Barbara, CA, USA, 2011, pp. 222–239.

[14] International Standardization of Organization (ISO): Information technology-Security techniques-Lightweight cryptography-Part 5: Hash-functions, document ISO/IEC 29192-5, 2016, p. 2016.

[15] D. Toz, Cryptanalysis of Hash Functions (Dissertation for the Degree of Philosophiae Doctor), Katholieke Universiteit Leuven, Belgium, 2013.

[16] M. Walter, Algebraic Methods in Analyzing Lightweight Cryptographic Symmetric Primitives, (Dissertation for the Degree of Philosophiae Doctor), Department of Computer Science, Technical University of Darmstadt, 2013.

[17] G. Zhang, M. Liu, A distinguisher on PRESENT-like permutations with application to SPONGENT, SCI. CHINA Inf. Sci. 60 (2017) 1–13.

[18] C.A. Lara-Nino, M. Morales-Sandoval, A. Diaz-Perez, Small lightweight hash functions in FPGA, in: Proc. IEEE 9th Latin American Symposium on Circuits & Systems, LASCAS, Puerto Vallarta, Mexico, 2018, pp. 1-4.

[19] B. Jungk, L. Rodrigues Lima, M. Hiller, A systematic study of lightweight hash functions on FPGAs, in Proc. International Conference on ReConFigurable Computing and FPGAs, ReConFig14, Cancun, Mexico, 2014, pp. 1-6.

[20] M. Mozaffari Kermani, R. Azarderakhsh, A. Aghaie, Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC, ACM Trans. Embedded Comput. Syst. 16 (2) (2016) 1–19.

[21] M. Prasad Soni, A.R. Pais, Light-weight hash algorithms using GRP instruction, in: Proc. of the International Conference on Security of Information and Networks, ACM, Jaipur, India, 2017, pp. 206–211.

[22] G. Hatzivasilis, G. Floros, I. Papaefstathiou, C. Manifavas, Lightweight authenticated encryption for embedded on-chip systems, Inf. Secur. J. 25 (4–6) (2016) 151–161.

[23] J.P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, C. Winnerlein, BLAKE2: Simpler, smaller, fast as MD5, in: Proc. of the International Conference on Applied Cryptography and Network Security, Springer, Jaipur, India, 2013, pp. 119–135.

[24] J.F. Rossetti, Hardware Design and Performance Analysis for Cryptographic Sponge BLAMKA (Master of thesis), university of sao paulo, 2017.
[25] M. Stevens, Attacks on Hash Functions and Applications, Mathematical Institute, Faculty of Science, Leiden University, 2012.
[26] Rashidi B., Efficient and flexible hardware structures of the 128-bit CLEFIA block cipher, IET Comput. Digit. Tech. (2019) 69–79.
[27] C. Beierle, J. Jean, S. Kolbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, S.M. Sim, The SKINNY family of block ciphers and its low-latency variant MANTIS, in: Proc. 36th Advances in Cryptology-CRYPTO, in: LNCS, vol. 9815, Santa Barbara, CA, USA, 2016, pp. 123–153.
[28] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, T. Shirai, Piccolo: An ultra-lightweight block cipher, in: Proc. CHES: International Workshop on Cryptographic Hardware and Embedded Systems, in: LNCS, vol. 6917, Nara, Japan, 2011, pp. 342–357.
[29] B. Rashidi, Efficient and high-throughput application-specific integrated circuit implementations of HIGHT and PRESENT block ciphers, IET Circuits Dev. Syst. 13 (6) (2019) 731–740.
[30] M. Ullrich, C. De Canniere, S. Indesteege, O. Kucuk, N. Mouha, B. Preneel, Finding optimal bitsliced implementations of 4*4-bit S-boxes, in: Proc. Symmetric Key Encryption Workshop, Copenhagen, DK, 2011, pp. 1–20.
[31] G. Grosso, G. Leurent, F.X. Standaert, K. Varici, LS-Designs: Bitslice encryption for efficient masked software implementations, in: Proc. 21st International Workshop on Fast Software Encryption, in: LNCS, vol. 8540, London, UK, 2014, pp. 18–37.
[32] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: Proc. of Advances in Cryptology, Berlin, Germany, 1999, pp. 388–397.
[33] Kocher, P.C. and , Timing attacks on implementations of Diffie–Hellman, RSA, DSS, in: Proc. of Advances in Cryptology, Berlin, Germany, 1996, pp. 104–113.
[34] Y.I. Hayashi, N. Homma, T. Mizuki, T. Aoki, H. Sone, L. Sauvage, J.L. Danger, Analysis of electromagnetic information leakage from cryptographic devices with different physical structures, IEEE Trans. Electromagn. Compat. 55 (3) (2013) 571–580.
[35] E. Brier, C. Clavier, F. Olivier, Correlation power analysis with a leakage model, in: LNCS, Proc. of Cryptographic Hardware and Embedded Systems, CHES, Cambridge, MA, USA, 2004, pp. 16–29.
[36] https://www.arduino.cc/en/Main/ArduinoBoardUno.
[37] A. Stillmaker, B. Baas, Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm, Integr. VLSI J. 58 (2017) 74–81.
[38] H.-S. Wong, D.J. Frank, P. Solomon, C. Wann, J. Wesler, Nanoscale CMOS, Proc. IEEE 87 (4) (1999) 537–570.