Microelectronics Journal 93 (2019) 104615

Contents lists available at ScienceDirect

Microelectronics Journal

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

High throughput implementation of SHA3 hash algorithm on field programmable gate array (FPGA)

hanno assunto un messaggio sempre di 64 bit

Soufiane El Moumni [a,*], Mohamed Fettach [a], Abderrahim Tragha [b]

[a] Information Processing Laboratory, Faculty of Sciences Ben M'sik, Hassan II University, Casablanca, Morocco
[b] Information Processing and Modeling Laboratory, Faculty of Sciences Ben M'sik, Hassan II University, Casablanca, Morocco

LE INFO

Keywords:
High-speed implementation
SHA3 hash function
FPGA device

A B S T R A C T

Cryptographic hash function is an essential element in sensitive communications, such as banking, military and health. It ensures secure communication by checking data integrity, storing passwords and other important roles. Keccak hash function (i.e. SHA3) is the best one in terms of resistance against recent cryptanalysis attacks as well as of hardware performance. However, an efficient improvement in terms of hardware performance is always needed, such as increasing speed or decreasing area consumption. In this paper, we have focused on improving the speed (throughput) of Keccak hash algorithm by proposing a new design which is based on decreasing the number of clock cycles needed to produce a hash value. Consequently, we could achieve 33.35 Gbps as a highest achieved throughput. However, a decrease in terms of maximum frequency has been noticed. Our design has been implemented in Xilinx Virtex5 and Virtex6 FPGA device, and has been compared to recent published implementations.

1. Introduction

Ensuring a secure communication is an essential element to keep stability in many fields especially in sensitive ones like banking, e-commerce and military. Therefore, applying a cryptographic protocol is mandatory technique to overcome this issue. Cryptographic hash function is the main protocol that takes charge of many applications such as data integrity checking, authentication, random number generation, digital signature computing and password storing.

In addition, cryptographic hash function plays an important role in many recent fields including health that requires secure communication between the patient and the doctor [1], Cloud computing [2], internet of things IoT [3], big data [4], Radio frequency identification RFID [5], and Wireless sensor network WSN [6].

As in any field, the competitiveness is demanded to ensure sustainability of the security and the privacy of communication. Therefore, several cryptographic hash functions have been proposed, and their arrangement is based on their resistance against cryptanalysis attacks.

MD5 [7], SHA-1 [8] and SHA-2 [9] were the most popular cryptographic hash functions. However, after the cryptanalysis attacks on MD5 [10], SHA1 [11], and SHA2 [12], the National Institute of Standards and Technology announced a public contest in 2007 [13] to select a new cryptographic hash function SHA3 that should prove its resistance against recent attacks.

After three rounds of SHA3 competition and based on some selection criteria [14] such as security, design simplicity, implementation efficiency, and flexibility; NIST announced in 2012 that Keccak hash function is the winner of that contest [15] and it will represent SHA-3 hash algorithm.

Although the proven strength of Keccak in terms of security and resistance against recent attacks, there remains a need for an efficient implementation be it Software (e.g. CPU) or Hardware (e.g. FPGA, ASIC). Therefore, several approaches have been proposed to efficiently implement Keccak hash algorithm by focusing on minimising area consumption, energy, or increasing speed (throughput).

In this paper, we have focused on the last version of Keccak hash algorithm [16], and we have used FPGA platform to propose a hardware implementation of Keccak. In addition, our proposed design aims to increase throughput and generate all possible output lengths of Keccak hash algorithm.

The rest of this article is outlined as follows: related works are presented in section 2. Section 3 provides an overview of Keccak hash algorithm. In section 4, the reason behind choosing FPGA platform is stated, followed by describing our approach in details. Then, in section 5,

* Corresponding author.
E-mail address: soufianeelmoumni@gmail.com (S. El Moumni).

https://doi.org/10.1016/j.mejo.2019.104615
Received 14 October 2018; Received in revised form 28 July 2019; Accepted 10 September 2019
Available online 14 September 2019
0026-2692/© 2019 Elsevier Ltd. All rights reserved.

our implementation results are shown and compared fairly with recent authors' approaches. In section 6, we discuss our achievements. Finally, section 7 concludes the paper.

2. Related works

Numerous designs of Keccak hash algorithm have been proposed on FPGA devices, each one provides a technique that allows an efficient implementation. This efficiency is represented by increasing throughput, or decreasing area and energy consumption. However, there still is room for performance improvement, especially in terms of increasing throughput.

Many of researchers have focused on improving throughput by several methods including the method used in Ref. [17], where the authors have adopted pipelining and re-timing techniques in the case of multi messages hashing, to improve the performance of the design. In Ref. [18], the authors have presented a technique that consists of two steps, the first one is based on combining all the five steps of the compression function, this allows an elimination of all intermediate states between these steps, and secondly, LUT primitives were used to propose a hardware architecture, that allows a high throughput implementation. The approach used in Ref. [19] has achieved high data throughput by applying loop unrolling and pipelining techniques on Keccak hash algorithm. As well as, the authors of [20] have proposed two-staged pipelined architecture. While, in Ref. [21], the authors could select the best number of stages used in the pipelining technique, which allowed good results in terms of throughput, in the case of single and multi-block messages, and also multi message processing.

All of these works and others [22–30] have shared a common goal, which revolved around increasing the throughput of the cryptographic hash function Keccak. Nevertheless, there is always a need for an improved implementation to enhance the throughput.

The major focus of our proposed design is to provide a high-speed implementation of Keccak by increasing the throughput. Furthermore, the design can dynamically change the output length to allow generating all possible output lengths of the last version of Keccak hash algorithm. Whereas, many works focused on generating only one output length [22–25] or two output lengths: 256 and 512 bits, as mentioned in paper [26].

3. Keccak overview

Cryptographic hash function Keccak was designed by Bertoni et al. [31], it is based on sponge construction [32], this feature made Keccak non-differentiable from random oracle [33], which allows generating a random output with fixed length from an arbitrary length input.

A sponge function has three parameters to be selected: the width 'b', the bitrate 'r' that has the same bloc size of a message, and the capacity 'c' such that c = b-r.

In addition, the sponge function is based on three steps: padding, absorbing and squeezing. During the first phase, a padded message is produced with a length multiple of the bitrate r, this step enhance/reinforce Keccak against length extension attacks [34].

After that, the padded message is split into blocks with 'r' bits length each, and each block is XORed with the bitrate 'r', then, the capacity 'c' is concatenated with the XORed block to generate a state that represents an input of the absorbing phase.

During this step, a function *f* absorbs the generated state, then, internal processing is repeated for desired number of rounds.

Thirdly, the produced state of the absorbing phase is squeezed out. This constitutes the hash value with the desired length. Fig. 1 shows the mechanism of the sponge construction.

This compression function *f* is constituted by five sub-functions called steps as mentioned below. The application of this function *f* is repeated for 24 iterations. For more details about Keccak, we recommend the official paper of Keccak [31].

1-Theta step (Θ):

1) For $0 \le x < 5$ and $0 \le z < 64$, C(x,z) = S(x,0,z) XOR S(x,1,z) XOR S(x,2,z) XOR S(x,3,z) XOR S(x,4,z)
2) For $0 \le x < 5$ and $0 \le z < 64$, D(x,z) = C[(x-1)mod 5,z] XOR C[(x+1) mod 5,(z-1)mod 64]
3) For $0 \le x < 5$, $0 \le y < 5$ and $0 \le z < 64$, S′(x,y,z) = S(x,y,z) XOR D(x,z)
4) Return S'.

Where S represents the initial state (r_0,c_0), while, the intermediate state is represented by S'. 'C' and 'D' are intermediate signals with 5 × 64 bits, and (x,y,z) are the axes of the state (5x5x64 bits).

2-Rho step (ρ):

1) For $0 \le z < 64$ S' (0, 0, z) = S (0, 0, z)
2) Let (x, y) = (1, 0)
3) For $0 \le t \le 23$
   a For $0 \le z < 64$, S′(x, y, z) = S[x, y, (z-(t+1)(t+2)/2)mod 64]
   b (x, y) = (y, (2x+3y) mod 5).
4) Return S'.

3-Pi step (π):

1) For $0 \le x < 5$, $0 \le y < 5$ and $0 \le z < 64$, S′(x, y, z) = S [(x+3y) mod 5, x, z].
2) Return S'.

4- Chi step (χ):

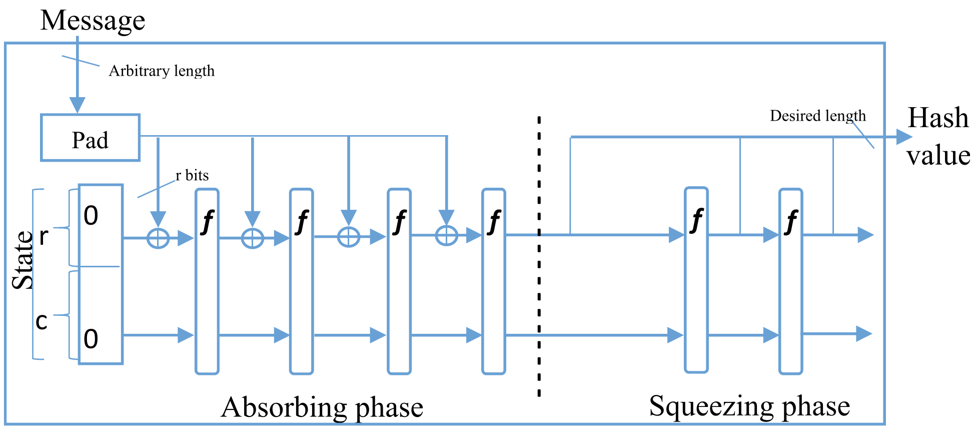

Fig. 1. The sponge construction mechanism.

1) For $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < 64$ $S'(x,y,z) = S(x,y,z)$ XOR [NOT [S((x+1) mod 5, y, z)] AND S((x+2) mod 5, y, z)].
2) Return S'.

5- Iota step (ι):

1) For $0 \leq z < 64$, S (0, 0, z) = S (0, 0, z) XOR RC (z).
2) Return S.

Where RC is a 64 bits signal that represents the round constant. For each round number, a value is assigned to RC signal by following Table 1, which gives a hexadecimal representation of all RC values according to the round number (RC[round number]).

In this work, we are focusing on Keccak-f [1600], which is the largest version of Keccak hash function. In addition, our message padding was done by following a padding rule as mentioned in Ref. [35].

In order to select the output length of Keccak-f [1600] hash function, the bitrate 'r' and the capacity 'c' have to be chosen in advance, as summarised in Table 2.

As mentioned in Table 2, there are four possible output lengths to be generated by the hash function. As it is known, the more the output length is high, the more the hash function is resistant against cryptanalysis attacks that focus on the output size. However, several applications of hash function prefer the smaller output size, especially the ones that are not using them for security. Therefore, in this work, we proposed a design that allows generating all these possible output lengths. Using a selector signal, we could select the desired output length of the hash value.

4. Material and method

4.1. Hardware platform

Despite the presence of software implementations of security algorithms using high powerful CPUs, it is always necessary to provide a hardware description of them using high developed platforms such as FPGAs and ASICs. This hardware implementation gives to the researchers the opportunities to optimize the efficiency of the design rather than being limited in the internal performance of the CPU in the case of software description.

Moreover, hardware implementation is more secure due to its independence from the main processor. In addition, it is faster than software implementation because of its dedicated task.

In this work, we have chosen FPGA device to implement Keccak hash algorithm due to its reputation in this field [36]; also, it is the most commonly used type of programmable device for this task. Thus, we could have a long list of proposed approaches to compare with.

Furthermore, FPGA platform offers many benefits [37] compared to ASIC, such as low cost, reconfigurability, and facility in correcting errors.

4.2. Proposed design

Unlike several works, our design processes data completely including padding and preprocessing parts. Moreover, it is not based on any FPGA resources such as Digital signal processors (DSPs) [38]. Additionally, we did not use any software aid as mentioned in Ref. [39] or any dedicated interfaces as adopted in Ref. [40].

Table 2
The parameters (r,c) and the appropriate output lengths.

Output length	r	c
Keccak-224	1152	448
Keccak-256	1088	512
Keccak-384	832	768
Keccak-512	576	1024

Fig. 2 presents an overview of our proposed design, after that, each part of it will be explained in detail.

Since we are interested in Keccak-*f* [1600] core by providing an efficient architecture, then the data input length is inconsequential. Hence, only a message with 64 bits length has been treated.

As shown in Fig. 2, the first part is the padding, where some bits are padded to the message according to a rule as mentioned in Ref. [35], in order to generate a data with r bits length. This rule is summarised in Table 3:

Fig. 3 shows the mechanism of padding part. When the desired output length is 224 bits, then, the first padding rule (padding 1) will be selected by the output length selector and the table (number) of padding rules will be applied to generate a padded data with r bits (r = 1152).

After generating a padded data, the mapping part is applied to generate preprocessed data that can be ready for the Keccak-*f* [1600] core (24 rounds of 5 sub-functions).

This mapping part took the initial value of the bitrate 'r', XOR it with the padded data, and concatenated the result with the initial value of the capacity 'c'. Then a transformation of generated data is needed to represent it in three-dimensional arrays 5 × 5 × 64 bits: state, equation (1) was applied as mentioned in Ref. [16].

$$\mathrm{State}(x,\ y,\ z) = \mathrm{String}\ (z + 64 * (5 * x + y)) \tag{1}$$

where String is an intermediate result such that String = (Padded data XOR r) ||c, while (x, y, z) are the state dimensions such that $0 \leq x \leq 4$, then $0 \leq y \leq 4$, and $0 \leq z \leq 63$.

After this mapping part, input selector signal selects the initial state to be processed by the permutation rounds block, as mentioned in Fig. 1.

During this permutation rounds block part, 24 iterations of five sub-functions (Theta, Rho, Pi, Chi and Iota) must be applied on the initial state. Which means that there is at least a need of 24 clock cycles to generate the hash value of the selected state, and knowing that the throughput is calculated according to formula (2) below, as reported in Refs. [17–30].

$$\mathrm{Throughput} = \frac{\mathrm{Block\ size} \times \mathrm{Max\ Frequency}}{\#\mathrm{Clock\ cycles}} \tag{2}$$

where the block size is the bitrate size 'r' and #clock cycles represents the number of iterations needed of the five sub-functions to generate the hash value.

Therefore, to increase the throughput, either we increase the maximum frequency or decrease the number of clock cycles required to generate the hash value.

Table 1
The round constants (RC).

RC [0]	0000000000000001	**RC [8]**	000000000000008A	**RC [16]**	8000000000008002
RC [1]	0000000000008082	**RC [9]**	0000000000000088	**RC [17]**	8000000000000080
RC [2]	800000000000808A	**RC [10]**	0000000080008009	**RC [18]**	000000000000800A
RC [3]	8000000080008000	**RC [11]**	000000008000000A	**RC [19]**	800000008000000A
RC [4]	000000000000808B	**RC [12]**	000000008000808B	**RC [20]**	8000000080008081
RC [5]	0000000080000001	**RC [13]**	800000000000008B	**RC [21]**	8000000000008080
RC [6]	8000000080008081	**RC [14]**	8000000000008089	**RC [22]**	0000000080000001
RC [7]	8000000000008009	**RC [15]**	8000000000008003	**RC [23]**	8000000080008008

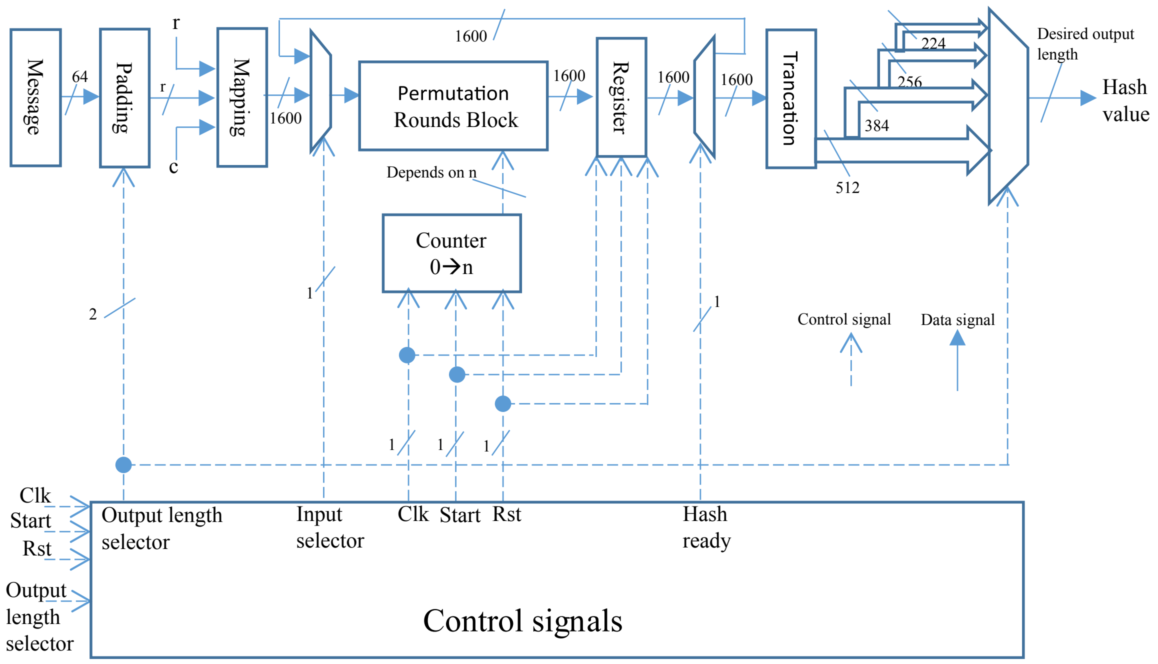

Fig. 2. Proposed design overview.

Table 3
Padding rule.

r	c	Desired output length	Description
1152	448	224	The input is concatenated with 100 … 0 until a multiple of 8, then concatenated with 8 bit representation of 28, then 8 bit representation of r/8, then 100 … 0 until a multiple of r.
1088	512	256	The input is concatenated with 100 … 0 until a multiple of 8, then concatenated with 8 bit representation of 32, then 8 bit representation of r/8, then 100 … 0 until a multiple of r.
832	768	384	The input is concatenated with 100 … 0 until a multiple of 8, then concatenated with 8 bit representation of 48, then 8 bit representation of r/8, then 100 … 0 until a multiple of r.
576	1024	512	The input is concatenated with 100 … 0 until a multiple of 8, then concatenated with 8 bit representation of 64, then 8 bit representation of r/8, then 100 … 0 until a multiple of r.

In this work, we have focused on decreasing the number of clock cycles by using a replication technique. Fig. 4 shows the internal design of permutation rounds block with the counter $(0 \rightarrow 24)$ which means without minimisation of clock cycles.

Therefore, as shown in Fig. 4, to decrease the number of clock cycles, divide the number of the counter into the half with taking care of round constant values. Fig. 5 shows this part of our approach with only 12 clock cycles.

In this work, we have evaluated all possible number of clock cycles {24, 12, 8, 6, 4, 3, 2}, and we have generated the throughput that corresponds to these values.

Besides increasing the throughput, we have also taken into account all different output lengths (128, 256, 384 and 512) to build a complete design of Keccak hash algorithm. Therefore, to produce a hash value with 512 bits length, the padding 4 will be selected as mentioned in Fig. 3, then, after truncating the state to 512 bits by following equation (1), a selector signal will select the signal of 512 bits and assign it to the hash value which represents our output design.

5. Results and comparison

High-performance implementation of Keccak hash algorithm is always demanded, especially in terms of speed (Throughput) which represents the essential metric in data processing.

For this reason, we have focused on providing a new design of Keccak hash algorithm using FPGA platform and VHDL language. In addition, to compare fairly this work with related works in the literature, we chose Virtex5 (XC5VLX110-3FF1760) and Virtex6 (XC6VHX380T-3FF1924) family due to its popularity in this field.

In this work, the synthesizer of Xilinx ISE design suite has been used. After place & route phase, the synthesizer reported a complete description of our proposed design in terms of achieved maximum frequency and area consumption. While, the throughput (TP) was computed by

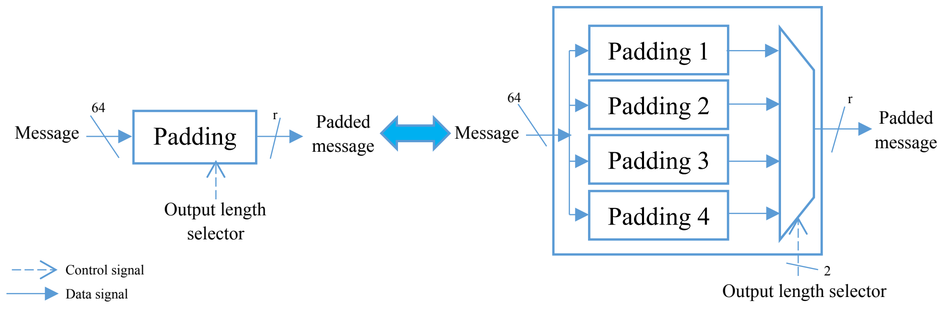

Fig. 3. Padding part mechanism.

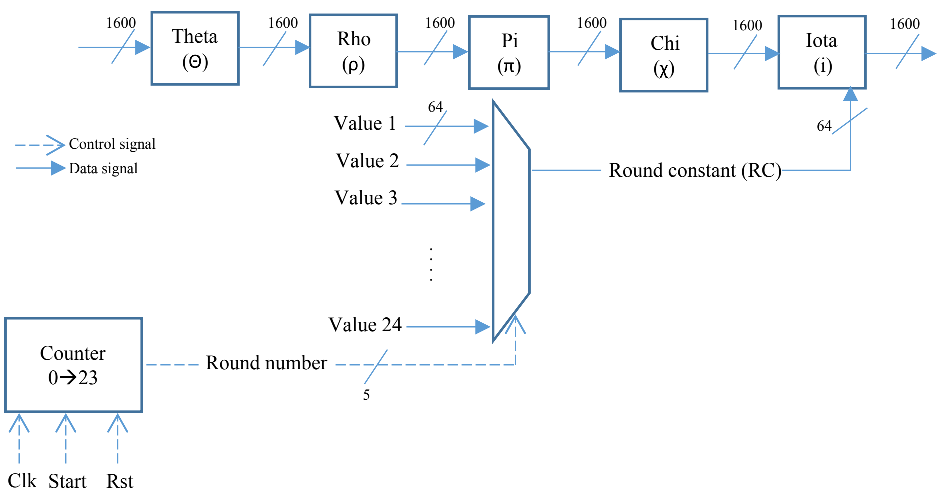

Fig. 4. Permutation rounds block.

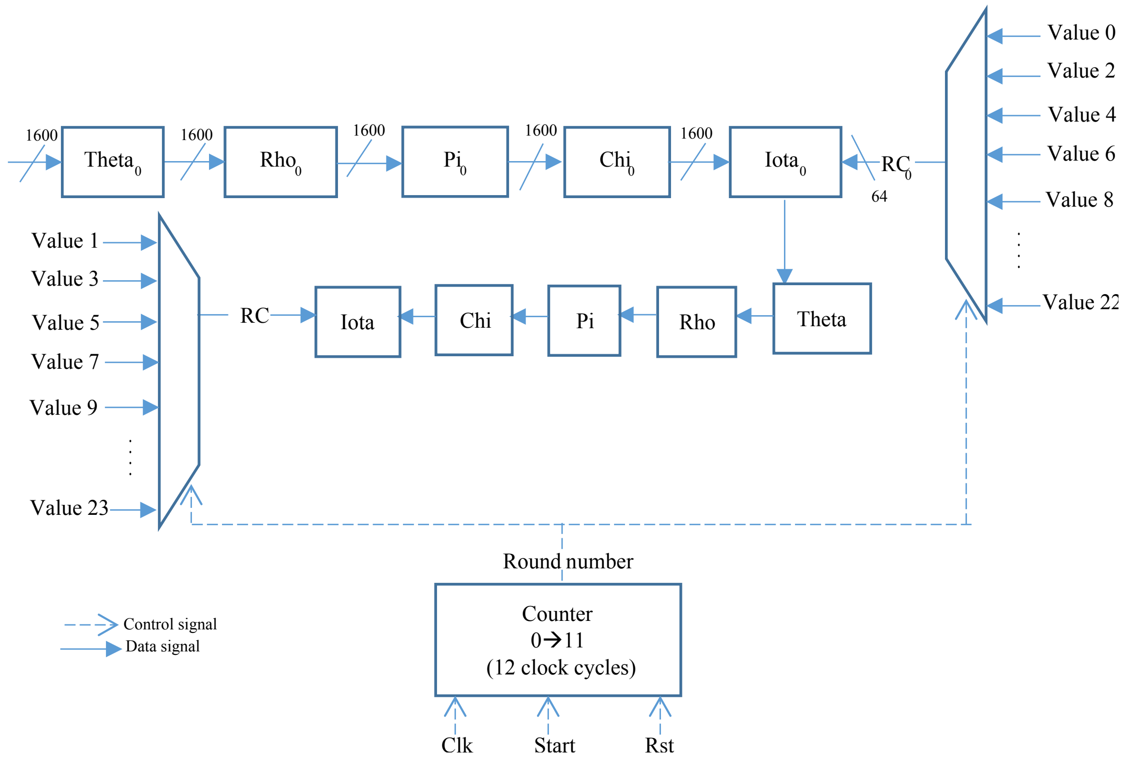

Fig. 5. Our permutation rounds block with 12 clock cycles.

following equation (2).

As it is shown in equation (2), the decrease of the #clock cycles allows an increase in the throughput, which was our main purpose. Therefore, our approach focused on decreasing the number of iterations needed to produce a hash value.

The proposed design could achieve 33.35 Gbps as a maximum throughput for 224 output length, and 16.67 Gbps for 512 output length. Table 4 summarises the achieved implementation results of each possible iteration number in terms of area consumption, maximum frequency, and throughput of all output lengths. In addition, it evaluates our results by giving a fair comparison of our high achievements to the recently published works reported in the literature.

Furthermore, we have calculated the efficiency of our design by dividing the throughput (in Mbps) by the area consumption (number of slices). Table 5 summarises the achieved efficiency of all output lengths and for each number of iterations needed to produce a hash value.

6. Discussion

The throughput is an important metric in data processing especially in information security field, because, it describes the efficiency of the algorithm and its resistance against cryptanalysis attacks that focuses on the hardware weaknesses.

It is clearly mentioned in Table 4 that our results could achieve a high throughput which was our main goal, especially in the case of maximum repetition of the five sub-functions that decreases the number of clock cycles. That throughput could exceed achieved related works throughput.

However, in the same case, our results are weak compared to recent authors' works in terms of area consumption and maximum frequency. Due to the relation between area consumption and efficiency, this significant increase on area consumption produces undesirable side effects on efficiency results. Table 5 shows explicitly these undesired results,

Table 4
Implementation results and comparison.

Our design	FPGA device	#clock cycles	Area (slices)	Fmax (MHz)	TP-224 (r = 1152)	TP- 256 (r = 1088)	TP- 384 (r = 832)	TP-512 (r = 576)
	Virtex5	24	1365	326.38	15.66	14.79	11.31	7.83
		12	2144	192.25	18.45	17.43	13.32	9.228
		8	3595	139.97	20.15	19.03	14.55	10.07
		6	4586	111.57	21.42	20.23	15.47	10.71
		4	7224	78.13	22.50	21.25	16.25	11.25
		3	8693	61.50	23.61	22.30	17.05	11.80
		2	12487	41.64	**23.98**	**22.65**	**17.32**	**11.99**
	Virtex6	24	1432	413.77	19.86	18.75	14.34	9.93
		12	3557	232.45	22.31	21.07	16.11	11.15
		8	3646	165.48	23.82	22.50	17.21	11.91
		6	5257	139.38	26.76	25.27	19.32	13.38
		4	10120	95.70	27.56	26.03	19.90	13.78
		3	7220	80.72	30.99	29.27	22.38	15.49
		2	15579	57.91	**33.35**	**31.50**	**24.09**	**16.67**
[22]	Virtex6	12	1406	344	–	–	–	16.51
[23]	Virtex6	24	1048	194.78	–	8.830	–	–
[24]	Virtex5	48	1163	273	–	–	–	7.8
[25]	Virtex5	25	1291	377.86	–	17.132	–	–
[26]	Virtex5	24	1217	277	–	12.56	–	–

Table 5
Efficiency results.

Our design	FPGA device	# Clock cycles	Efficiency 224	Efficiency 256	Efficiency 384	Efficiency 512
	Virtex5	24	11.47	10.83	8.28	5.73
		12	8.60	8.13	6.21	4.30
		8	5.60	5.29	4.04	2.80
		6	4.67	4.41	3.37	2.33
		4	3.11	2.94	2.24	1.55
		3	2.71	2.56	1.96	1.35
		2	1.92	1.81	1.38	0.96
	Virtex6	24	13.87	13.10	10.02	6.93
		12	6.27	5.93	4.53	3.14
		8	6.54	6.17	4.72	3.27
		6	5.09	4.81	3.68	2.55
		4	2.72	2.57	1.97	1.36
		3	4.29	4.05	3.10	2.15
		2	2.14	2.02	1.55	1.07
[22]	Virtex6	12	–	–	–	11.74
[23]	Virtex6	24	–	8.42	–	–
[24]	Virtex5	48	–	–	–	6.70
[25]	Virtex5	25	–	13.27	–	–
[26]	Virtex5	24	–	10.32	–	–

especially compared with approaches [22,25]. For the rest of the competitors ([23,24,26]), our efficiency results are higher.

Our expected achievements was reaching a high throughput implementation, by repeating the circuit of the five sub-functions of Keccak as possible as we can in a combinational mode, in order to minimise the number of clock cycles needed to produce the hash value. As clearly reported in Table 4, we could increase the throughput by decreasing the number of iterations. However, the achieved throughput was not quite enough due to the decrease of maximum frequency that represents an essential element to increase the throughput as shown in equation (2). Our results show that when we decrease the number of clock cycles, the maximum frequency decreases.

Fig. 6 describes the decrease of maximum frequency (Fmax) on Virtex5, when we decrease the number of iterations, namely when we increase the circuit repetition in combinational mode.

It is clearly shown in Fig. 6 that when we decrease the number of iterations, the frequency decreases exponentially.

In order to determine the relationship between the maximum frequency (Fmax), the throughput (TP) and the number of repetitions (n) of the five sub-functions (Theta, Rho, Pi, Chi, and Iota), we started by measuring our performance indicators (Fmax, TP) for different n values in case of 512 output length. As mentioned in Table 6.

According to Table 6, there is a relation of TP such as: $TP_n = f(F_{max}, n)$ Therefore, based on equation (2), we have:

$$TP_1 = \frac{\tau F_{max1}}{\#clock_1}$$

where ‘r’ is the treated block size and $\#clock_1$ is the number of iterations needed to generate the hash value in the case of n = 1, which means without any repetition of the circuit ($\#clock_1 = 24$);

$$TP_2 = \frac{\tau F_{max2}}{\#clock_2}$$

where $\#clock_2$ represents the number of iterations needed to generate the hash value in the case of n = 2, which means with one repetition of the circuit (Theta, Rho, Pi, Chi and Iota). Hence

$$\#clock = 12 = \frac{\#clock_1}{2}$$

Then,

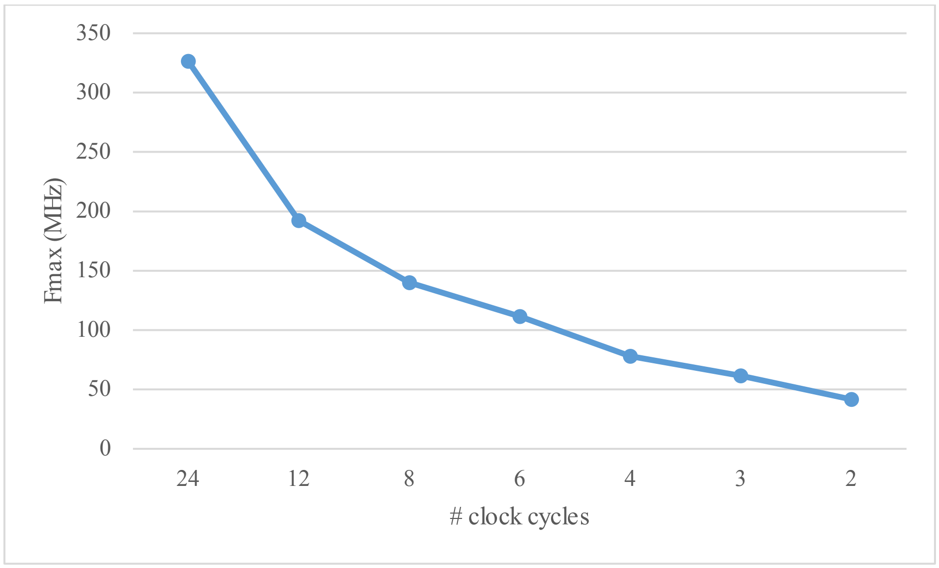

Fig. 6. The evolution of maximum frequency (Fmax).

Table 6
Our implementation results with 512 output length.

# clock cycles	# repetition of the circuit (n)	F_{max} (MHz)	TP-512 (MHps)	1/n	TP_n/TP_1
24	1	326.38	7.83	1	1
12	2	192.25	9.22	0.5	1.18
8	3	139.97	10.07	0.33	1.29
6	4	111.57	10.71	0.25	1.37
4	6	78.13	11.25	0.17	1.44
3	8	61.50	11.80	0.12	1.51
2	12	41.64	11.99	0.08	1.53

$$TP_2 = \frac{\tau F_{max2}}{\#clock_2} = \frac{2rF_{max1}}{\#clock_1}$$

After multiplying and dividing by F_{max1}, we get:
Which gives

$$TP_2 = \frac{2F_{max2}TP_1}{F_{max1}}$$

As a result,

$$Tp_n(F_{max}, n) = \frac{nF_{maxn}TP_1}{F_{max1}}$$

where n ∈ {1, 2, 3, 4, 6, 8, 12} which is a submultiple of 24. equation (3) can be expressed as follows:

$$F_{maxn} = \frac{F_{max1}TP_n}{nTP_1} \tag{3}$$

Therefore, as mentioned in Table 6, the ratio TP_n/TP_1 is substantially constant. In addition, F_{max1} is a constant value. Consequently, the responsible element for Fmax diminution is 'n'. In other words, this decrease of F_{max} is due to the repetition of the circuits in combinational mode that broaden the design and makes its critical path longer, hence, the minimum period increases, and consequently, the maximum frequency decreases exponentially.

Despite of these weaknesses of our proposed design, there are some published techniques to be applied on our design, such as the adopted technique in Ref. [41], or what we call: folding technique as used in Ref. [24]. These two methods aim to decrease the area consumption and provide a compact design of the algorithm. Whereas, the pipelining technique [20,21] aims to decrease the critical path of the design, which allows a decrease in terms of the minimum period, and consequently, an increase on maximum frequency (Fmax).

7. Conclusion

Cryptographic hash algorithm has an important role in many sensitive domains, including military, health and e-commerce. It allows generating a random number, storing passwords and producing digital signatures.

Keccak hash algorithm is the securest one until now, due to its proven success during the NIST contest. It prove a strong resistance against cryptanalysis attacks as well as a high performance in terms of hardware description.

In this paper, we focused on hardware side of Keccak hash algorithm using FPGA platform, in which we have proposed an approach that allows achieving high throughput by decreasing the number of clock cycles needed to produce a hash value.

The proposed design has generated all possible output lengths of Keccak (224,256,384 and 512) by processing a single block of message completely and producing its hash value through different steps: padding, mapping, permutations round and truncating step.

We could achieve 33.35 Gbps as a high throughput on Virtex6 family without using DSPs as FPGA resources or any software helper as used in Ref. [21]. However, our achievements in terms of area consumption and maximum frequency were two drawbacks that prevented our work to achieve a high performance of Keccak hash algorithm. Therefore, our future focus will be on these two metrics by applying some techniques to minimise area consumption and increase maximum frequency.

Acknowledgement

This work was financially supported by National Center of Scientific and Technical Research (CNRST, Morocco) under the grant reference K007/015. In addition, the authors of this work are thankful to Professor Dorel Lucanu and Professor Vlad Radulescu from Computer Science Faculty, Alexandru Ioan Cuza University, Iasi (Romania), for sharing with us their scientific writing experiences.

References

[1] Xiong Li, Maged Hamada Ibrahim, Saru Kumari, et al., Anonymous mutual authentication and key agreement scheme for wearable sensors in wireless body area networks, Comput. Network. 129 (2017) 429–443.
[2] Lifei Wei, Haojin Zhu, Zhenfu Cao, et al., Security and privacy for storage and computation in cloud computing, Inf. Sci. 258 (2014) 371–386.

[3] Kun-Hee Han, Woo-Sik et Bae, Proposing and verifying a security protocol for hash function-based IoT communication system, Clust. Comput. 19 (1) (2016) 497–504.
[4] Dinusha Vatsalan, Ziad Sehili, Peter Christen, et al., Privacy-preserving record linkage for big data: current approaches and research challenges, in: Handbook of Big Data Technologies, Springer, Cham, 2017, pp. 851–895.
[5] Yu et Yinhui, Lei Zhang, Research on a provable security RFID authentication protocol based on Hash function, J. China Univ. Posts Telecommun. 23 (2) (2016) 31–37.
[6] Jennifer Yick, Biswanath Mukherjee, et Ghosal Dipak, Wireless sensor network survey, Comput. Network. 52 (12) (2008) 2292–2330.
[7] Ron Rivest, The MD5 Message-Digest Algorithm. http://tools.ietf.org/rfc/rfc1321, 1991.
[8] National Institute of Standards and Technology, FIPS PUB 180. http://security.isu.edu/pdf/fips180.pdf, 1993.
[9] National Institute of Standards and Technology, FIPS PUB 180-1. http://www.itl.nist.gov/fipspubs/fip180-1.htm, 1995.
[10] Xiaoyun Wang, et Yu Hongbo, How to break MD5 and other hash functions, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, Berlin, Heidelberg, 2005, pp. 19–35.
[11] Xiaoyun Wang, Yiqun Lisa Yin, et Yu Hongbo, Finding collisions in the full SHA-1, in: Annual International Cryptology Conference, Springer, Berlin, Heidelberg, 2005, pp. 17–36.
[12] Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, et al., Preimages for step-reduced SHA-2, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, Berlin, Heidelberg, 2009, pp. 578–597.
[13] Richard F. Kayser, Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family, Fed. Regist. 72 (212) (2007) 62.
[14] Turan, Meltem Sönmez, Ray Perlner, Lawrence E. Bassham, et al., Status report on the second round of the SHA-3 cryptographic hash algorithm competition, NIST Interagency Report (2011) 7764.
[15] Chad Boutin, NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition, vol. 2, Press release., 2012. October.
[16] M.J. Dworkin, SHA-3 standard: permutation-based hash and extendable-output functions, Federal Inf. Process (2015), https://doi.org/10.6028/NIST.FIPS.202. Stds.(NIST FIPS)-Report Number 202.
[17] Abdulkadir Akin, Aydin Aysu, Onur Can Ulusel, et al., Efficient hardware implementations of high throughput SHA-3 candidates keccak, luffa and blue midnight wish for single-and multi-message hashing, in: Proceedings of the 3rd International Conference on Security of Information and Networks, ACM, 2010, pp. 168–177.
[18] Muzaffar Rao, Thomas Newe, et Grout Ian, Efficient high speed implementation of secure hash algorithm-3 on Virtex-5 FPGA, in: Digital System Design (DSD), 2014 17th Euromicro Conference on, IEEE, 2014, pp. 643–646.
[19] Jarosław Sugier, Low cost FPGA devices in high speed implementations of K eccak-f hash algorithm, in: Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland, Springer, Cham, 2014, pp. 433–441.
[20] George S. Athanasiou, George-Paris Makkas, Georgios et Theodoridis, High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm, in: Communications, Control and Signal Processing (ISCCSP), 2014 6th International Symposium on, IEEE, 2014, pp. 538–541.
[21] Harris E. Michail, Lenos Ioannou, Artemios G. et Voyiatzis, Pipelined SHA-3 implementations on FPGA: architecture and performance analysis, in: Proceedings of the Second Workshop on Cryptography and Security in Computing Systems, ACM, 2015, p. 13.
[22] Ming Ming Wong, Jawad Haj-Yahya, Sau Suman, et al., A new high throughput and area efficient SHA-3 implementation, in: Circuits and Systems (ISCAS), 2018 IEEE International Symposium on, IEEE, 2018, pp. 1–5.
[23] Thomas Newe, Muzaffar Rao, Daniel Toal, et al., Efficient and high speed fpga bump in the wire implementation for data integrity and confidentiality services in the iot, in: Sensors for Everyday Life, Springer, Cham, 2017, pp. 259–285.
[24] Magnus Sundal, Ricardo et Chaves, Efficient FPGA implementation of the SHA-3 hash function, in: 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), IEEE, 2017, pp. 86–91.
[25] Muzaffar Rao, Thomas Newe, Ian Grout, et al., High speed implementation of a SHA-3 core on virtex-5 and virtex-6 FPGAs, J. Circuits Syst. Comput. 25 (07) (2016) 1650069.
[26] Aziz, Arshad, Latif, Kashif, et al. Resource efficient implementation of keccak, Skein & JH algorithms on reconfigurable platform. Cankaya University Journal of Science and Engineering, vol. 13, no 1.
[27] Muzaffar Rao, Thomas Newe, Ian Grout, et al., An FPGA-based reconfigurable IPSec AH core with efficient implementation of SHA-3 for high speed IoT applications, Secur. Commun. Netw. 9 (16) (2016) 3282–3295.
[28] Fatma Kahri, Hassen Mestiri, Belgacem Bouallegue, et al., High speed FPGA implementation of cryptographic KECCAK hash function crypto-processor, J. Circuits Syst. Comput. 25 (04) (2016) 1650026.
[29] Ali Alzahrani, Fayez et Gebali, Multi-core dataflow design and implementation of secure hash algorithm-3, IEEE Access 6 (2018) 6092–6102.
[30] Debjyoti Bhattacharjee, Vikramkumar Pudi, Anupam et Chattopadhyay, SHA-3 implementation using ReRAM based in-memory computing architecture, in: Quality Electronic Design (ISQED), 2017 18th International Symposium on, IEEE, 2017, pp. 325–330.
[31] Guido Bertoni, Joan Daemen, Michaël Peeters, et al., The keccak sha-3 submission, Submission to NIST (Round 3) 6 (7) (2011) 16.
[32] Bertoni Guido, et al., Cryptographic Sponge Functions, 2011, pp. 1–93.
[33] Guido Bertoni, Joan Daemen, Michael Peeters, et al., On the indifferentiability of the sponge construction, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, Berlin, Heidelberg, 2008, pp. 181–197.
[34] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, et al., Merkle-Damgård revisited: how to construct a hash function, in: Annual International Cryptology Conference, Springer, Berlin, Heidelberg, 2005, pp. 430–448.
[35] Brian Baldwin, Andrew Byrne, Liang Lu, et al., FPGA implementations of the round two SHA-3 candidates, in: Field Programmable Logic and Applications (FPL), 2010 International Conference on, IEEE, 2010, pp. 400–407.
[36] Francisco Rodríguez-Henríquez, Nazar Abbas Saqib, Arturo Díaz Pérez, et al., Cryptographic Algorithms on Reconfigurable Hardware, Springer Science & Business Media, 2007.
[37] National instruments. Introduction to FPGA technology; top five benefits. http;//zone.ni.com/devzone/cda/tut/p/id/6984, Dec 2010
[38] George Provelengios, Paris Kitsos, Nicolas Sklavos, et al., FPGA-based design approaches of keccak hash function, in: Digital System Design (DSD), 2012 15th Euromicro Conference on, IEEE, 2012, pp. 648–653.
[39] Harris E. Michail, Lenos Ioannou, Artemios G. et Voyiatzis, Pipelined SHA-3 implementations on FPGA: architecture and performance analysis, in: Proceedings of the Second Workshop on Cryptography and Security in Computing Systems, ACM, 2015, p. 13.
[40] Bernhard Jungk, Jurgen et Apfelbeck, Area-efficient FPGA implementations of the SHA-3 finalists, in: Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on, IEEE, 2011, pp. 235–241.
[41] Alia Arshad, Arshad Aziz, et al., Compact implementation of SHA3-512 on FPGA, in: Information Assurance and Cyber Security (CIACS), 2014 Conference on, IEEE, 2014, pp. 29–33.