A New High Throughput and Area Efficient SHA-3 Implementation

Ming Ming Wong, Jawad Haj-Yahya, Suman Sau and Anupam Chattopadhyay
School of Computer Science and Engineering (SCSE)
Nanyang Technological University (NTU) Singapore 639798
Email: {mmwong, jawad, ssau, anupam}@ntu.edu.sg

***Abstract*—High performance and area efficient Secure Hash Algorithm (SHA-3) hardware realization is investigated and proposed in this work. In addition to the new and simplified round constant (RC) generator, the presented SHA-3 hash implementations employed architectural optimization approaches based on the concepts of unrolling, pipelining and subpipelining. This has therefore produced a total of five implementations of SHA-3 which are denoted as Cases I-V in both FPGA and ASIC. Considering the trade-offs between the performance and hardware cost, the best architecture in term of the throughput and area efficiency is identified in Case V. The architecture has the highest throughput of 16.51 Gbps and area efficiency of 11.47 Mbps/slices for the FPGA implementation. While in ASIC, our best implementation (Case V) achieves the highest throughput of 48 Gbps.**

I. Introduction

Cryptographic hash function is a deterministic procedure that take input messages of arbitrary length to produce a fixed length bit string termed as the digest message. This function is widely deployed in digital signature schemes, message authentication codes (MACs) and several other information security applications. The most essential properties of a secure cryptographic hash function are being simple in computation and yet highly non-invertibility with strong collision resistance.

In general, hardware implementations of cryptographic function are proven to be more secure than its software realizations. Furthermore, the implementations provide the flexibility to be optimized to cater the application's specific requirements such as low hardware area cost, high performance, efficient resource utilization and low power consumption. Our work aimed at achieving high throughput hardware architecture of SHA-3 Keccak without imposing unnecessary increment in the area cost.

The main contribution of this work is three-fold. First, a simplified round constant generator for SHA-3 hash is introduced, of which it requires much smaller hardware resources compared to the conventional design. Second, a new 2-stages subpipelined transformation round is employed where the register is inserted after the Theta (θ) step in the hash permutation function. Third, several architectural designs that are suitable for both single and multi-message hashings are investigated and hence resulted in five different SHA-3 architectures (represented in Case I-V).

Among all the implementations, Case V is presented as the new architecture that employed 2-stages subpipelined and unrolling by factor 2, followed by 2-stages pipeline in between the adjacent rounds. The experimental results proved that Case V offers higher throughput performance and provides better efficiency compare to the other cases. Note that the aim of the study is to introduce performance enhancement while keeping the hardware area cost to the minimal. Therefore, we did not opt for higher pipelining and unrolling factors in our designs.

II. The SHA-3 Function

The Keccak hash function [1], designed by G. Bertoni et. al., was announced by the NIST as the new Secure Hash Algorithm-3, i.e. SHA-3, in 2015. Generally, the Keccak algorithm is based on sponge construction, where the hash transformation is performed on an internal state that takes input of arbitrary length, and produce an output of the desired length. Depending on the selection of hash variants, the sponge construction will be build using appropriate bitrate (r) and capacity (c), where these parameters also serves as the main factor that determines the overall security level.

In brief, the SHA-3 hash function is composed of two phases which are as depicted in Figure 1. During the **absorbing phase**, the bitrate of the initialized state is XORed with the first part of the input. The new bitrate and together with the capacity of the initialized state matrix will form a new state that is used in f-permutation. The resulting state will serve as the new initial state for the next round and the process continues for 24 iteration rounds. Each round is divided into five separate steps, i.e. Theta (θ), Rho (ρ), Pi (π), Chi (χ) and Iota (ι) which are listed in equations (1), (2), (3) and (4).

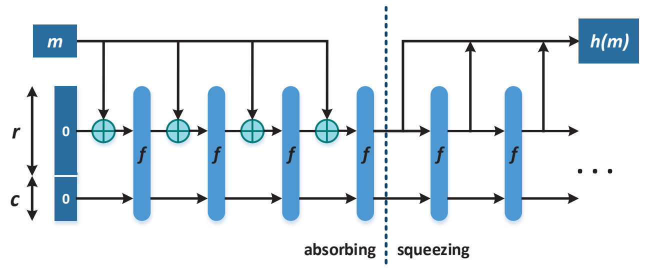

Fig. 1. Keccak sponge construction

Theta (θ) step:

$$\begin{aligned} C[x] &= A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4] \\ D[x] &= C[x-1] \oplus \mathrm{rot}(C[x+1],1) \\ A[x,y] &= A[x,y] \oplus D[x]; \\ &\quad 0 \le x,y \le 4 \end{aligned} \tag{1}$$

Rho (ρ) and Pi (π) steps:

$$B[y, 2x+3y] = \text{rot}(A[x,y], r[x,y]); \quad 0 \leq x,y \leq 4 \tag{2}$$

Chi (χ) step:

$$A[x,y] = B[x,y] \oplus ((\bar{B}[x+1,y] \wedge B[x+2,y]); \quad 0 \leq x,y \leq 4 \tag{3}$$

Iota (ι) step:

$$A[0,0] = A[0,0] \oplus RC[i]; \tag{4}$$

The element $A[x,y]$ signifies the particular word in the state array and the intermediate variables are denoted are $B[x,y]$, $C[x]$ and $D[x]$. Meanwhile, $RC[i]$ presents the round constants that corresponds to the specific round i. In addition to that, $r[x,y]$ represents the rotation offsets and the binary cyclic shift operation is indicated by $rot(w,r)$. Lastly, no further calculation is performed in **squeezing phase** and the output is contained in the first r-length of the internal state. The transformations will repeat until all the blocks come to the output of desired length.

In this study, we have chosen to work on 512-bits Keccak[1600] due to its guaranteed security margin. Therefore, the respective value of r and c are 576 and 1024. With that, the 1600-bit state matrix of Keccak is composed of 5×5 matrix of 64-bit words.

III. Hardware Architectures for Hash Function

Several research studies in the literature have presented various hardware implementations of SHA-3 hash function. The implementation's aim is directed either towards the lightweight design or architecture with high speed performance. As for the latter, various approaches such as unrolling, pipelining and parallel processing can be deployed on top of the basic SHA-3 implementation (refer Figure 2(a)) in order to achieve maximal throughput enhancement [2].

These optimization methodologies offers speed improvements in different manners. Unrolling is particularly efficient in improving the throughput for single message hashing [3]. Pipelining on the other hand brings the advantages of combined data throughput enhancement in multi-message hashing, where the function processes more than one message concurrently [3]. In other words, the blocks of different messages can be overlapped in the pipeline. SHA-3 hash function is widely utilized in vast variety of information security applications that involved single and multi-message hashings. Therefore, both of the above-mentioned approaches will be investigated and justified in this study.

Through the prior approach, the f-permutation block can be replicated and unrolled in SHA-3 hash function. As an example, with unrolling factor of 2, two transformation rounds are computed in a single clock cycle. Consequently, the total number of clock cycles is reduced by factor of two and this double-folds the overall throughput performance. Meanwhile, in the latter approach, pipeline can be employed in SHA-3 in two different manners. For instance, the straightforward approach is done by placement of pipeline registers in between the adjacent rounds within the unrolled hash architecture. In other words, the hash function is implemented with unrolling of factor 2 and followed by 2-stages pipelining.

Consequently, two different messages can be processed simultaneously in such implementation. Taking into the consideration of the combined throughput of all the available data stream, the overall hash function performance is in fact enhanced by a factor of two. Such achievement will be significantly beneficial in the case where the number of message blocks is relatively large [2].

Another plausible means of pipelining is by insertion of pipeline registers inside the hash function round. To be exact, the sub-pipelining is applied in between the interval steps within the f-permutation block. In this methodology, the attainable throughput improvement is relative to the maximum reduction in the function's critical path. Thus, the challenge comes in determining the appropriate placement of register pipeline in order to achieve the minimum critical path in SHA-3 function.

On the downside, speed enhancement offered through both unrolling and pipelining comes at the expenses of large hardware area cost, i.e., the efficiency in terms of throughput over area ratio may decreased substantially. Hence, these optimizations are only practical for applications where speed performance is of high imperative and the available hardware resource is sufficiently abundant.

In addition to the above-mentioned designs, a new architecture is presented in this study. In our proposed SHA-3 function, 2-stages subpipelining is applied within the f-permutation block and the overall hash function is unrolled by factor of 2 and followed by 2 pipelines insertion in between the round. This 2-stages subpipelining precisely divides in between the Theta (θ) step and the remaining four steps (ρ, π, χ, ι). To our best knowledge, such SHA-3 implementation design has yet to be reported in the previous studies. Detailed explanation of our proposed architecture, alongside with further algorithmic optimizations are discussed in Section IV.

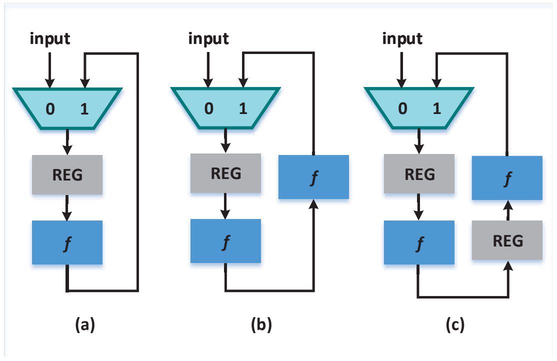

Fig. 2. Architecture of SHA-3 Hash Function (a) Basic iterative (b) Unrolling (k=2) (c) Unrolling (k=2) with pipelining (n=2).

IV. The New and Optimized SHA-3 Implementation

The new architecture design for SHA-3 hardware realization is presented in the following.

A. *Simplified Round Constant (RC) Generator*

To date, there are three plausible implementations for SHA-3 hash round constant generator. First, a circuit constructed using LFSR can be employed to perform on-the-fly generation of the round constant values [4]. Second, all of the 24 pre-calculated round constants of 64-bits length are stored in memory forms (such as registers or circular buffers) and transferred to the Iota module via multiplexer [5]. Third, combinatorial circuit using series of XOR gates can be realized to compute the round constant value upon every iteration.

In this work, our round constant generator also requires memory storage but in a much smaller size, where the length of the 24 pre-calculated constants value is now reduced to a byte size. This is achieved by storing only the non-zero bits in each of the round constant value. Based on the SHA-3 specification, there are only maximum number of 8 non-zero bits observed in each round constant value. The simplified 8-bits round constant values are tabulated in Table I. Besides, this also simplifies the computation in Iota (4), where number of logical XOR needed is reduced from 64 to 8. The bitwise XOR operation will be performed on bit $0, 1, 3, 7, 15, 31$ and 63 of $A[0, 0]$ respectively.

TABLE I
THE SIMPLIFIED ROUND CONSTANTS RC[I]

RC[0]	0x01	RC[1]	0x32	RC[2]	0xBA
RC[3]	0xE0	RC[4]	0x3b	RC[5]	0x41
RC[6]	0xF1	RC[7]	0xA9	RC[8]	0x1A
RC[9]	0x18	RC[10]	0x69	RC[11]	0x4A
RC[12]	0x7B	RC[13]	0x9B	RC[14]	0xB9
RC[15]	0xA3	RC[16]	0xA2	RC[17]	0x90
RC[18]	0x2A	RC[19]	0xCA	RC[20]	0xF1
RC[21]	0xB0	RC[22]	0x41	RC[23]	0xE8

B. New Subpipelined Transformation Round

The recent works on subpipelined SHA-3 hash were also reported in [5], [6] where in both of the works, the pipeline is inserted in between Pi (π) and Chi (χ) steps. On the other hand, in our design, the pipeline registers are inserted after the Theta (θ) operation (refer Figure 3). As a result, the longest delay in the first half of the computation is constituted of 5 XORs while the second part which includes Pi (π) to Iota (ι) covers the longest delay of 2 XORs, 1 AND and 1 XOR. The aim is to reduce the critical path by approximately half of which this shall lead to achieving maximal clock frequency.

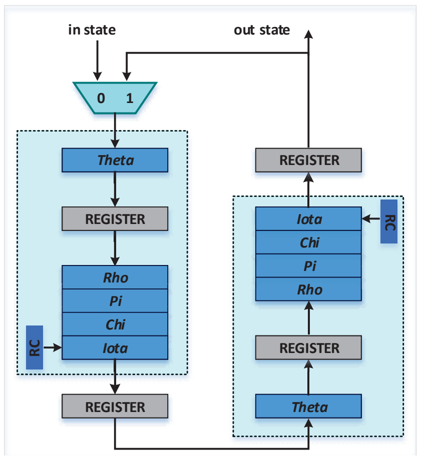

Fig. 3. New SHA-3 hash implementation in Case V.

C. Proposed SHA-3 Hash Architecture Design

A total of five different SHA-3 hash implementations were investigated in this work;

- Case I: Basic iterative with simplified round constant generator.
- Case II: Case I with new 2-stages subpipelining within transformation round.
- Case III: Case I with unrolling factor of 2.
- Case IV: Case III and followed by 2-stages pipelining in between adjacent rounds.
- Case V: Case II with unrolling factor of 2 and followed by 2-stages pipelining in between adjacent rounds.

Implementation in Case I represents the basic SHA-3 hash architecture with the simplified round constant generator (refer Section IV-A) and meanwhile, architecture in Case II further employed the new subpipelining approach (refer Section IV-B). Case III and Case IV are implemented using the architectures discussed in Section III and these will be also be used for benchmarking purposes. Eventually, the new and improved SHA-3 hash architecture presented as Case V (as depicted in Figure 3), employed all the methodologies in Cases I-IV.

V. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

The SHA-3 hardware architectures (for Cases I, II, III, IV and V) were implemented using VHDL language, simulated by ModelSim and synthesized with Xilinx ISE Design Suite. The targeted FPGA platform was XC6VLX75T from Xilinx Virtex-6 family. For ASIC, the designs were synthesized and mapped to TSMC 65nm standard cell library by using Synopsys Design Compiler version J-2014.09.

The FPGA implementation results are investigated in terms of the achievable frequency (maximum), area, throughput and efficiency and these metrics are tabulated in Table II. In general, the throughput for multi-message hashing is computed as follows.

$$\text{Throughput} \quad = \quad \frac{\#blocksize \times F_{max}}{\#clockcycle} \times \#N_{msg} \qquad (5)$$

where $\#blocksize$ refers to the number of processed bits, $\#clockcycle$ corresponds to the total clock cycles between successive messages to generate each digest message and F_{max} is the highest attainable frequency in the implementation. $\#N_{msg}$ is the number of messages that can be simultaneously hashed at a given time.

TABLE II
SHA-3 HASH IMPLEMENTATION RESULTS FOR CASES I-V ON XILINX VIRTEX 6 XC6VLX75T FPGA IMPLEMENTATIONS.

Case	Approach (Pipeline/Unroll)	Fmax (MHz)	Area (Slices)	Throughput (Gbps)	Efficiency (Mbps/Slices)
I	Basic	153	871	3.68	4.22
II	Subpipeline n=2	335	1393	8.04	5.77
III	Unrolled k=2	45	2145	2.16	1.00
IV	Unrolled k=2 Pipeline n=2	85	1416	4.08	2.88
V	Unrolled k=2 Pipeline n=2 Subpipeline n=2	344	1406	**16.51**	**11.47**

Hardware synthesis results deduced from Cases I to IV precisely reflected the characteristic and the performance of

TABLE III

EXPERIMENTAL RESULTS FROM THE RECENT HIGH THROUGHPUT SHA-3 HASH IMPLEMENTATIONS ON VIRTEX XILINX FPGA. THE NOTATION S, P AND U INDICATE THE APPROACHES USING SUBPIPELINING, PIPELINING AND UNROLLING RESPECTIVELY.

Work	Devices (Xilinx)	Approach (S/P/U)	Fmax (MHz)	Area (Slices)	Throughput (Gbps)	Efficiency (Mbps/Slices)
Athanasiou et al. (2014) [5]	Virtex 5	S [n=2]	389	1,702	9.35	5.49
Athanasiou et al (2014) [5]	Virtex 6	S [n=2]	397	1,649	9.55	5.80
Ioannou et al. (2015) [7]	Virtex 5	U [k=2] P [n=2]	352	2,652	16.90	6.37
Ioannou et al. (2015) [7]	Virtex 6	U [k=2] P [n=2]	391	2,296	18.77	8.17
Michail et al. (2015) [8]	Virtex 5	U [k=3] P [n=3]	352	3,197	25.34	7.93
Michail et al. (2015) [8]	Virtex 6	U [k=3] P [n=3]	391	3,965	28.15	7.10
Michail et al. (2015) [8]	Virtex 5	U [k=4] P [n=4]	357	4,632	34.27	7.40
Michail et al. (2015) [8]	Virtex 6	U [k=4] P [n=4]	392	4,117	37.63	9.14
Mestiri et al. (2016) [6]	Virtex 5	S [n=2]	317	4,793	6.34	1.32

9

different architecture designs for SHA-3 hash. Through subpipelining (refer to Case II) where a multi-message hashing is permissible, the total execution cycles also increases linearly. This is due to the iterative computation in SHA-3 hash function. Such approach however, is efficient in critical path reduction where the maximum attainable frequency can be greatly improved. Meanwhile, unrolling promotes throughput enhancement via hardware replication as the total execution cycles is reduced. This advantage is rather insignificant for SHA-3 as the critical path is further extended with the replication of combinational circuit (refer Case III). A much favorable way is shown in Case IV where register pipelines are inserted upon every unrolled module. With that, the simultaneous message processing is enabled without imposing additional delays in the existing critical path.

Based on these investigations, subpipelining is observed effective in critical path reduction while unrolling with pipelining enables simultaneous processing in SHA-3 hash function. Generally, both of the attributes bring positive effect on the throughput performance. In addition to that, it is evident that hardware unrolling has much adverse effect on the hardware area size compared to the insertion of pipeline registers. These factors have therefore driven the motivation of this work to explore a new configuration (Case V) where inner round subpipelining, unrolling and round pipelining are integrated to push the throughput performance to the next level, while maintaining the overall area-performance efficiency. From the hardware synthesis results, SHA-3 implementation in Case V is proven to has fastest throughput (16.51 Gbps) and highest efficiency (11.47 Mbps/Slices) compared to the other cases.

FPGA implementation comparison with the recent works as reflected in Table III, the highest throughput performance was presented in the work by [7], [8] where the authors employed higher unrolling factor in the SHA-3 hash implementations. However, this is traded off with much larger amount of slices needed. Therefore, considering the throughput/area factor, our Case V has the highest implementation efficiency compared to the existing works.

Selection of an efficient implementation does not lie solely on the architecture design but also the depending on the specific FPGA platform. For instance, both the throughput and efficiency of the same design would vary in Xilinx Virtex 5 and Virtex 6 implementations. This is evident from the results reported in [5], [7], [8]. Based on the observations in Table III, implementations on Virtex 6 have outperformed the implementations using Virtex 5. Similarly for our Case V, the architecture is lower in both throughput (13 Gbps) and efficiency factor (7.40 Mbps/Slices) on Xilinx Virtex 5 XC5VLX20T FPGA.

Furthermore, it is also worth noting that there is a non-linear relation between the depth of the employed pipelining and unrolling and the occupied area in FPGA. This is attributed to the nature of the device in the sense that each slice contained one or more LUTs, multiplexers, and flip-flops to implement the required logic. As the number of pipeline stages increases, the unused resource from the already employed slices can be utilized for accommodate the additional logic. This results in the non-linear increment in terms of hardware area cost.

TABLE IV

EXPERIMENTAL RESULTS FROM THE RECENT HIGH THROUGHPUT SHA-3 HASH IMPLEMENTATIONS ON ASIC.

Work	GE (K)	Fmax (MHz)	Throughput[1] (Gbps)	Efficiency (Mbps/GE)
Our work (best), Case V	105	1,000	48.00	0.4571
(Bertoni et al. [9]) basic	48	526	25.20	0.525
(Bertoni et al. [9]) unroll 2	67	333	32.00	0.4776
(Bertoni et al. [9]) unroll 3	86	244	35.12	0.4084
(Bertoni et al. [9]) unroll 4	105	192	36.92	0.3516
(Bertoni et al. [9]) unroll 6	143	135	38.9	0.272

To our best knowledge, the only related work that presented ASIC implementation of 512-bits Keccak[1600] was reported by Bertoni et al. [9]. As their work is synthesized on $130nm$ general purpose ST technology library, the attained frequency needs to be scaled accordingly in order to perform fair comparison. A scaling factor of 1.43 is required when moving from $130nm$ down to $90nm$ and followed by another round of 1.43 scaling when moving further from $90nm$ to $65nm$ [10]. With that, the maximum frequency is approximately doubled once porting from $130nm$ to $65nm$. The ASIC implementation results of our best design, Case V and the *scaled* results of different implementations reported by Bertoni et al. are summarized in Table IV. The results showed that the architecture in Case V exhibits the highest throughput even though it has slightly lower area efficiency compared to the first two cases (the basic and the unroll 2).

VI. CONCLUSION

In this paper, we proposed a simplified round constant generator which is hardware cost effective for SHA-3 hash function and a new inner f-permutation subpipelining approach was demonstrated. Incorporated with 2-stages subpipelined and unrolling by factor 2, followed by 2-stages pipeline in between the adjacent rounds, a new SHA-3 hardware realization was presented in Case V. Based on experimental results, the new SHA-3 implementation was proven high in throughput performance and outperformed the existing works in terms of throughput/area efficiency.

As of future work, we aim to study our implementation performance in integration with actual application, where the optimized SHA-3 block will be used as part of memory integrity unit of a secured processor.

[1]Scaled throughput for Bertoni et al. to the 65nm process technology.

References

[1] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Keccak*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 313–314.

[2] E. Homsirikamol, M. Rogawski, K. Gaj, and G. Mason, "Comparing hardware performance of round 3 SHA-3 candidates using multiple hardware architectures in Xilinx and Altera FPGAs," in *Ecrypt II Hash Workshop 2011*, 2011. [Online]. Available: http://www.ecrypt.eu.org/hash2011/proceedings/hash201107.pdf

[3] A. Akin, A. Aysu, O. C. Ulusel, and E. Savaş, "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*, ser. SIN '10. New York, NY, USA: ACM, 2010, pp. 168–177. [Online]. Available: http://doi.acm.org/10.1145/1854099.1854135

[4] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. P. Marnane, "FPGA implementations of the round two SHA-3 candidates," in *2010 International Conference on Field Programmable Logic and Applications*, Aug 2010, pp. 400–407.

[5] G. S. Athanasiou, G. P. Makkas, and G. Theodoridis, "High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm," in *2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, May 2014, pp. 538–541.

[6] H. Mestiri, F. Kahri, M. Bedoui, B. Bouallegue, and M. Machhout, "High throughput pipelined hardware implementation of the KECCAK hash function," in *2016 International Symposium on Signal, Image, Video and Communications (ISIVC)*, Nov 2016, pp. 282–286.

[7] L. Ioannou, H. E. Michail, and A. G. Voyiatzis, "High performance pipelined FPGA implementation of the SHA-3 hash algorithm," in *2015 4th Mediterranean Conference on Embedded Computing (MECO)*, June 2015, pp. 68–71.

[8] H. E. Michail, L. Ioannou, and A. G. Voyiatzis, "Pipelined SHA-3 implementations on FPGA: Architecture and performance analysis," in *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, ser. CS2 '15. New York, NY, USA: ACM, 2015, pp. 13:13–13:18. [Online]. Available: http://doi.acm.org/10.1145/2694805.2694808

[9] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, "Keccak implementation overview," *URL: http://keccak. neokeon. org/Keccak-implementation-3.2. pdf*, 2012.

[10] S. Borkar, "Design challenges of technology scaling," *IEEE micro*, vol. 19, no. 4, pp. 23–29, 1999.