Novel Control Unit Design for a High-Speed SHA-3 Architecture

Pranav Gangwar
Department of Electronics and Communication Engineering
Delhi Technological University
New Delhi, India
pranavgangwar@gmail.com

Neeta Pandey
Department of Electronics and Communication Engineering
Delhi Technological University
New Delhi, India
neetapandey@dce.ac.in

Rajeshwari Pandey
Department of Electronics and Communication Engineering
Delhi Technological University
New Delhi, India
rpandey@dce.ac.in

***Abstract*— This paper presents a control unit design for improving the throughput of SHA3-256 architecture. The Iterative data path design is employed to process multi-block messages, which makes the design functional for any input size. A higher throughput is achieved by reducing the number of control signals in the critical path, and optimizing the counter which generates the memory address and affects the maximum operating frequency. The synchronization problems caused by this approach between the generated Hash and its corresponding Valid signal, are overcome by the Synchronizer. The proposal exhibits 920 Mbps higher throughput, i.e. a 7% improvement over state of the art.**

Keywords—SHA-3, hash, keccak, high-speed, high throughput, multi-block message, control unit, FPGA

I. Introduction

Cryptographic Hash function is used in applications like digital signature generation and verification [1], password verification [2], proof-of-work [3], file or data identifiers [4], key derivation [5], message authentication codes [6], and pseudorandom bit generation [7]. It processes any arbitrarily long message, and produces a digest of a fixed length. Secure Hash Algorithm (SHA) is class of Hash functions that are computationally infeasible to invert.

The National Institute of Standards and Technology (NIST) devised a cryptographic Hash standard called SHA-0 in 1993 [8], which was later modified to SHA-1 in 1995 [9]. With increasing security standards NIST announced four variants of SHA-1 in 2002 and named this family SHA-2 [10]. The theoretical proof of possible attacks on SHA-1 in 2005 raised security alarms as both SHA-1 and SHA-2 were based on the same underlying math, which led to the development of SHA-3 through a public competition [11]. In 2012, NIST officially declared SHA-3 family of standard based on Keccak algorithm. The SHA-3 family consists of SHA3-224, SHA3-256, SHA3-384, and SHA3-512.

Existing literature records three ways of implementing the SHA-3 architecture: iterative [12], folded [13], and pipelined [14]. The folded structure consumes the minimum area while the pipelined structure achieves the highest throughput. The iterative structure strikes a balance between high speed and low area. The pipelined structure cannot be used to process a sequence of multi-block messages because the digest of first message block is XORed with the second message block before advancing it for processing. Hence, multi-block messages cannot be processed in a pipeline. Therefore, the iterative structure is suitable for all real time security applications.

The throughput of these architectures could not be increased by inserting Flip Flops and reducing the critical path, because the increased clock latency offsets the gain of the reduced critical path. This paper improves the throughput by optimizing the control logic in the critical path, a method not yet explored in the literature.

The paper is organized as follows. Section II gives a brief description of the Keccak algorithm used in SHA-3 hashing standard. Section III describes the proposed design, Section IV presents results, and Section V concludes the paper.

II. Preliminaries

The SHA-3 belongs to a cryptographic primitive family Keccak [12], and is characterized by rate (r) and capacity (c). Figure 1 depicts block diagram of the Sponge function [15] which forms basis of the Keccak algorithm. The width of the state processed by the Sponge function is b, where b=r+c, which is initially set to zero. The f block in Fig. 1 can represent any random transformation. The Sponge function can be divided into two phases: Absorbing phase, and Squeezing phase. In the Absorbing phase, a XOR operation is performed between r-bit message blocks $M_{(1)}$, $M_{(2)}$, …., $M_{(n)}$, and r bits of the internal state, or the r-bit f output. In the Squeezing phase, the Keccak-f permutation is repeatedly applied on the message blocks to generate an output of desired length by concatenating $H_{(1)}$, $H_{(2)}$, …, $H_{(m)}$.

The message block M is padded with zeros so that the length of M is divisible by r. The padded block is divided into n r-bit blocks: $M_{(1)}$, $M_{(2)}$, …, $M_{(n)}$. The length of the message block decides the number of iterations in the Absorbing phase. The Squeezing phase is performed only once, as the length of the digest is smaller than the size of the states in Keccak-f.

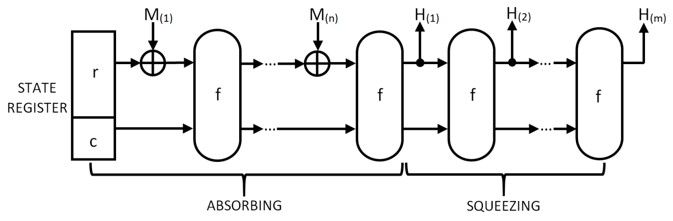

Fig. 1. Sponge Function [15]

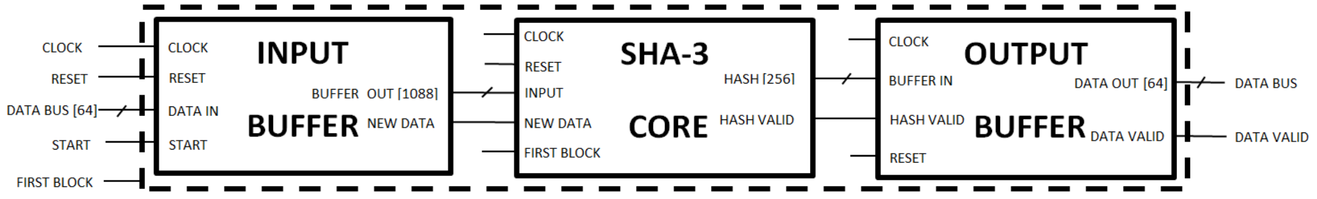

Fig. 2. Block Diagram representation of the Proposed Design

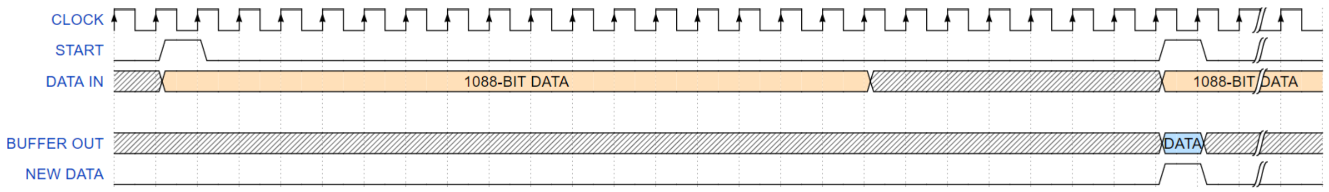

Fig. 3. Operation of Input Buffer

The state width b in Keccak-f[b] family of functions determines the length of the internal state, and the permissible values are: 25, 50, 100, 200, 400, 800 and 1600. For the state width of 1600 bits, the value of r is taken as 1152, 1088, 832 and 576 bits for SHA3-224, SHA3-256, SHA3-384 and SHA3-512 respectively. This paper implements SHA3-256 hashing standard which is based on Keccak-f[1600].

The processing of the Keccak starts by the conversion of the internal state of 1600-bits into a 3-D structure of 5*5*64 bits according to (1).

$$A[x,y,z]=S[w(5y+x)+z] \tag{1}$$

$$0 \le x,y < 5 \qquad 0 \le z < w \qquad b = 5 * 5 * w$$

The Keccak-f[1600] function requires twenty four processing rounds, where each round consists of five transformations [14]:

A. θ Step

$$B[x,z] = A[x,0,z] \oplus A[x,1,z] \oplus A[x,2,z] \oplus A[x,3,z] \oplus A[x,4,z] \tag{2}$$

$$C[x,z] = B[(x-1)\%5, z] \tag{3}$$

$$D[x,z] = B[(x+1)\%5, (z-1)\%64] \tag{4}$$

$$E[x,z] = C[x,z] \oplus D[x,z] \tag{5}$$

$$\theta[x,y,z] = A[x,y,z] \oplus E[x,z] \tag{6}$$

B. ρ Step

$$\rho[x,y,z] = \theta[x,y,\{z + r(x)(y)\}\%64] \tag{7}$$

C. π Step

$$\pi[y,\{(2x+3y)\%5\},z] = \rho[x,y,z] \tag{8}$$

D. χ Step

$$F[x,y,z] = \sim(\rho\pi[x,y,z]) \tag{9}$$

$$G[x,y,z] = F[x,y,z] \,\&\, \rho\pi[(x+1)\%5,y,z] \tag{10}$$

$$\chi[x,y,z] = \rho\pi[x,y,z] \oplus G[(x+1)\%5,y,z] \tag{11}$$

E. ι Step

$$\iota[0,0,z] = \chi[0,0,z] \oplus RC[z] \tag{12}$$

The value rotation offset (r) used in (7), and the round constant (RC) of (12) are mentioned in [14].

III. PROPOSED ARCHITECTURE

Figure 2 shows the proposed system consisting of an Input Buffer, an SHA-3 core, and an Output Buffer. The I/O Buffers are necessary due to the limited number of I/O pins available in the FPGA, and to simplify interfacing. A 64-bit Data Bus is used to eliminate idle time for the SHA-3 Core, and minimize the number of wait cycles for the Input Buffer.

The proposed system has a separate control unit for Input Buffer, SHA-3 Core, and Output Buffer. The control unit of Input Buffer waits for the assertion of START signal, after which it stores the 64-bit data from DATA BUS in the internal register for the next 17 clock cycles. At the 24th clock cycle, it enables the NEW DATA signal as depicted in Fig. 3.

The Output Buffer's control unit stores the calculated HASH in the internal register upon assertion of the HASH VALID signal. It is transmitted through the 64-bit DATA BUS in the next four clock cycles as shown in Fig. 4.

The complete architecture of the SHA-3 Core is shown in Fig. 5. It comprises of four major blocks namely: Transformation, Memory, Input Processing, and Control. The Transformation Block receives 1600-bit input from the State Register, and a 64-bit Round Constant from the Memory to apply the transformations from (2) to (12).

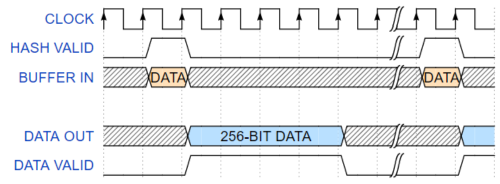

Fig. 4. Operation of Output Buffer

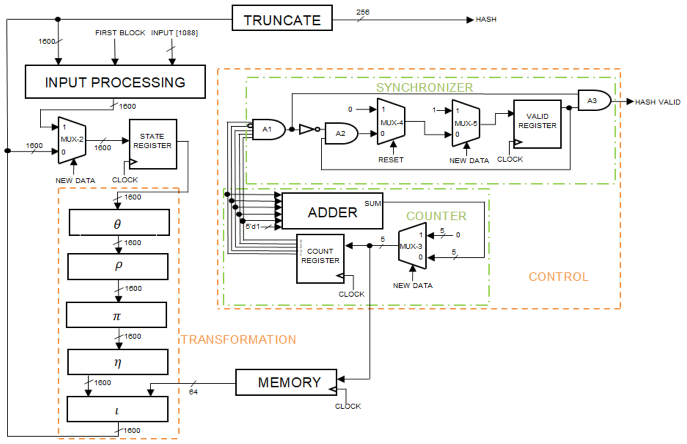

Fig. 5. Proposed architecture of SHA-3 Core

The Input Processing receives 1088-bit INPUT from the Input Buffer, 1600-bit Transformation's output, and the FIRST BLOCK signal. The FIRST BLOCK signal is crucial for processing multi-block messages, as shown in Fig. 6.

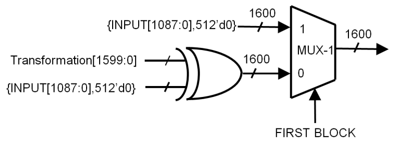

Fig. 6. Input Processing Block

Memory stores the 64-bit Round Constants required by the ι transformation. The synchronous memory receives the address from MUX-3, which is same as the value being stored in Count Register.

The Control Unit of the SHA-3 Core selects the data to be stored in the State Register, provides the appropriate address to the Memory, and handles the synchronization of HASH output with the HASH VALID signal. MUX-2 is a part of the critical path, which selects the State Register's input. Hence, the NEW DATA signal is toggled with the 1088-bit BUFFER OUT to ensure it is stable much before the Input Processing's output.

The Counter greatly affects the operating frequency of the architecture, because it generates the memory address of 64-bit Round Constant required by the ι transformation. Therefore, the counter is reset only by the NEW DATA signal to shorten the critical path, and the synchronization problems arising due to the absence of the RESET signal are tackled by the Synchronizer.

The Synchronizer is a part of the Control Unit which generates the HASH VALID signal at the 24th clock cycle after assertion of the NEW DATA signal, as depicted in Fig. 7. The block diagram of the Synchronizer in Fig.5 shows that the AND gate A1 goes high whenever the Count Register reaches $(23)_{10}$, which propagates zero to the Valid Register in the next clock cycle if there is no new message block to be processed (determined by the low value of the NEW DATA signal). This enables the HASH VALID signal in the current clock cycle, but prevents the free-flowing Counter from asserting it on reaching $(23)_{10}$ in the absence of any message block. The Valid Register stores zero whenever the RESET signal goes high, which ensures that the Counter does not enable the HASH VALID signal before receiving the first message block.

Therefore, the controllability was shifted from the Counter to the Synchronizer in order to minimize the critical path, and achieve a higher operating frequency.

IV. Results

The proposed architecture is implemented using Verilog, and simulated on Xilinx ISE Design Suite 14.7 tool. The maximum operating frequency, and the area occupied by the design are obtained after performing Place and Route on Xilinx Virtex-6 XC6VLX75TFF784-3 FPGA.

The available Keccak-f[1600] based SHA3-256 architectures which are implemented on FPGA, and capable of processing multi-block messages are compared on the basis of Throughput, Area, and Throughput per unit Area in Table I. For a fair comparison among the designs, the Latency only takes into account the clock cycles required for processing the Hash of a single message block. It does not include the clock cycles required for loading the message block, and transmitting the calculated hash.

The throughput of all the designs is calculated by (13), where the Block Size is 1088 bits and the Latency is taken from their respective references.

$$\text{Throughput} = \frac{\text{Frequency} * \text{Block Size}}{\text{Latency}} \qquad (13)$$

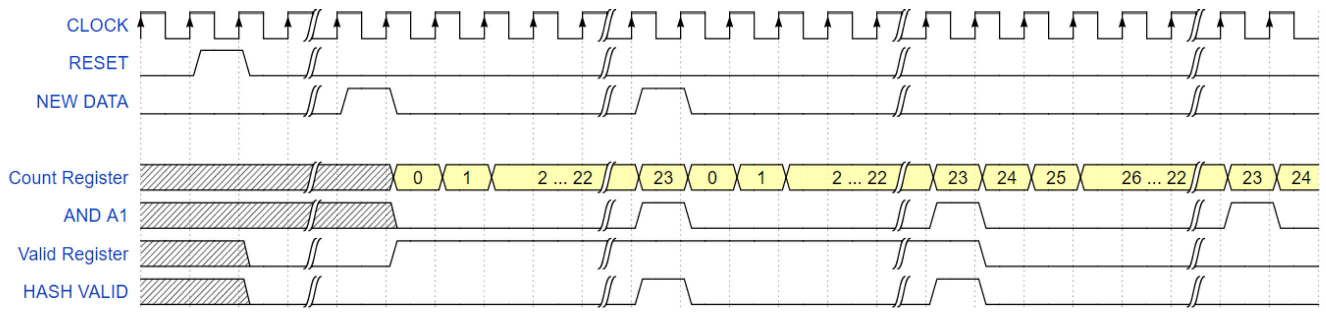

Fig. 7. Timing Diagram of the Synchronizer

TABLE I. COMPARISON TABLE

Ref.	FPGA	Freq. (MHz)	Area (Slices)	Latency (Clock Cycles)	Throughput	Throughput/ Area (Mbps/Slice)
[16]	Virtex-6	271	154	3696	79.77 Mbps	0.52
[17]	Virtex-6	311	91	1665	203.22 Mbps	2.23
[18]	Virtex-6	197	397	200	1.07 Gbps	2.69
[19]	Virtex-5	195.73	1971	25	8.52 Gbps	4.32
[20]	Virtex-6	251.7	1181	24	11.41 Gbps	9.66
[21]	Virtex-6	286.21	1207	24	12.97 Gbps	10.75
[22]	Virtex-6	301.57	915	25	13.12 Gbps	14.34
This work	Virtex-6	309.6	1249	24	14.04 Gbps	11.24

Thus, the throughput of the proposed design is higher by 920 Mbps than state of the art designs.

The designs presented in [16], [18] and [19] contain a padding unit in their architecture which increases their reported area, whereas the rest of the designs assume the message block to be 1088 bits long.

V. CONCLUSION

A new approach of optimizing the Control Unit to improve the throughput of SHA3-256 architecture is presented in this paper. It focuses on reducing the critical path delay of the design by using minimal control logic. The design of the counter is improved with the help of a Synchronizer, which generates the memory address and the HASH VALID signal to increase the throughput of the design. The post place and route results of the proposal obtained on Virtex-6 XC6VLX75TFF784-3 FPGA using Xilinx ISE Design Suite 14.7 confirm a higher throughput than state of the art. A nominal increase in area is observed due to the use of a convoluted synchronizer in the design.

This design approach is not specific to the SHA3-256 architecture, but could be employed in any SHA-3 design. The SHA3-256 was taken as a reference to justify the design methodology which enhances the throughput, due to the presence of a large number of designs for comparison.

REFERENCES

[1] R. C. Merkle, "A digital signature based on a conventional encryption function", Advances in Cryptology Crypto'87, vol. 293, pp. 369-378.

[2] Milton M. Anderson, "Re-initialization of an iterated hash function secure password system over an insecure network connection", *US Patent*, May 1998.

[3] M. Jakobsson, A. Juels, "Proofs of work and breadpudding protocols", Proc. Commun. Multimedia Security, pp. 258-272, 1999-Sep.

[4] G. H. Moulton, S. B. Whitehill, "Hash file system and method for use in a commonality factoring system", *US Patent*, March 2004.

[5] H. Krawczyk, "Cryptographic Extraction and Key Derivation: The HKDF Scheme", Proc. 30th Ann. Conf. Advances in Cryptology (CRYPTO '10), 2010.

[6] Liehua Xie, G. R. Arce and R. F. Graveman, "Approximate image message authentication codes," in *IEEE Transactions on Multimedia*, vol. 3, no. 2, pp. 242-252, June 2001.

[7] L. C. Noll, R. G. Mende, S. Sisodiya, "Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system", *US Patent*, March 1998.

[8] National Institute of Standards and Technology (NIST), document Secure Hash Standard, FIPS PUB 180, U.S. Department of Commerce, May 1993, [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/NIST.FIPS.180.pdf.

[9] National Institute of Standards and Technology (NIST), document Secure Hash Standard, FIPS PUB 180-1, U.S. Department of Commerce, Apr. 1995, [Online]. Available: https://archive.org/stream/federalinformati1801nati#mode/2up.

[10] National Institute of Standards and Technology (NIST), document Secure Hash Standard, FIPS PUB 180-2, U.S. Department of Commerce, Aug. 2002, [Online]. Available: https://csrc.nist.gov/CSRC/media/Publications/fips/180/2/archive/2002-08-01/documents/fips180-2.pdf.

[11] M. J. Dworkin, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, Aug. 2015, [Online] Available: http://dx.doi.org/10.6028/NIST.FIPS.202.

[12] E. Homsirikamol, M. Rogawski, K. Gaj, "Comparing Hardware Performance of Round 3 SHA-3 Candidates using Multiple Hardware Architectures in Xilinx and Altera FPGAs", Cryptology ePrint Archive: Report 2012/368, 2012.

[13] M. Sundal and R. Chaves, "Efficient FPGA Implementation of the SHA-3 Hash Function," 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Northrhine Westfalia, Germany, 2017, pp. 86-91.

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

[15] B. Guido, Cryptographic sponge functions, pp. 1-93, 2011.

[16] J.-P. Kaps, P. Yalla, K. K. Surapathi, B. Habib, S. Vadlamudi, S. Gurung, and J. Pham, "Lightweight Implementations of SHA-3 Candidates on FPGAs, " in Progress in Cryptology INDOCRYPT 2011, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 7107, pp. 270-289.

[17] B. Jungk and M. Stöttinger, "Hobbit — Smaller but faster than a dwarf: Revisiting lightweight SHA-3 FPGA implementations," *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, 2016, pp. 1-7.

[18] B. Jungk, "Evaluation Of Compact FPGA Implementations For All SHA-3 Finalists", Third SHA-3 Candidate Conference, March, 2012.

[19] B. Baldwin *et al.*, "FPGA Implementations of the Round Two SHA-3 Candidates," *2010 International Conference on Field Programmable Logic and Applications*, Milano, 2010, pp. 400-407.

[20] T. Honda, H. Guntur and A. Satoh, "FPGA implementation of new standard hash function Keccak," *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, Tokyo, 2014, pp. 275-279.

[21] E. Homsirikamol, M. Rogawski, K. Gaj, "Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs", Cryptology ePrint Archive, Report 2010/445, Dec. 2010.

[22] K. Latif, M. Rao, A. Aziz, A. Mahboob, "Efficient hardware implementations and hardware performance evaluation of SHA-3 finalists", Proc. Conf. SHA-3 Candidate, pp. 1-14, Mar. 2012.