DESIGN OF COMPACT IMPLEMENTATION OF SHA-3(512) ON FPGA

M.M.Sravani[1], C.H.Pallavi[2]

[1] PG Student, Department of ECE, Siddharth Institute of Engineering & Technology, A.P., India
[2] Assistant Professor, Department of ECE, Siddharth Institute of Engineering & Technology, A.P., India

---***---

Abstract - *In this work we present a compact design of newly selected Secure Hash Algorithm (SHA-3) on Xilinx Field Programmable Gate Array (FPGA) device Spartan 3E. The design is logically optimized for area efficiency by merging Rho, Pi and Chi steps of algorithm into single step. By logically merging these three steps we save 16 % logical resources for overall implementation. It in turn reduced latency and enhanced maximum operating frequency of design. It utilizes only 240 Slices and has frequency of 301.02* ***MHz's Comparing the results of our design with the*** *previously reported FPGA implementations of SHA3-512, our design shows the best throughput per slice (TPS) ratio of 30.1.*

Key Words: SHA3; Cryptography; FPGA; Xilinx; Security etc….

1. INTRODUCTION

A cryptographic hash function is a deterministic process whose input is arbitrary random block of data and produces an output of fixed size, which is known as the (Cryptographic) hash value. These functions were initially introduced to provide specific security requirements and integrity. Recent secure hash algorithms were found Susceptible to attacks including MD5, RIPEMD, SHA-0, SHA-1 and SHA-2 [1]. The long-term security of these algorithms was uncertain, which led to requirement of new cryptographic hash function. Therefore National Institute of Standards and Technology (NIST) announced Keccak algorithm as new secure hash algorithm (SHA-3) in the year 2012 [2] and announced as Federal Information Processing Standard Publication (FIPS PUB) 202 in April, 2014 [3].

FPGAs are ideal platform for the implementation of cryptographic algorithms. Modern FPGAs are equipped with enhanced embedded resources such as BRAMs and Digital Signal Processing (DSP) blocks in addition to LUTs and CLBs [4] that can be used to optimize the implementations. Different implementations of secure hash algorithm (SHA-3) on FPGA platform are reported in open literature [5]. The previous work has been categorized into low-area and high-speed implementations. The main challenge was to implement the design with most suitable available resources keeping a balance between area and throughput constraints. Different approaches have been adopted previously for the implementation of SHA-3 depending whether it is for low-area or high-speed designs. throughput compact FPGA implementation of SHA-3 that offers maximum possible throughput with better device utilization in terms of TPS.The remaining paper is organized as follows. Section 2 describes the previous work related to implementation of SHA-3 on FPGA and section 3 gives brief overview of SHA-3 algorithm. In section 4 we present the compact hardware implementation of SHA-3. we give the implementation results of our work . At the end, we provide conclusions related to our work in section 5.

2. PREVIOUS WORK

A lot of earlier work related to FPGA implementations of the SHA-3 has been reported since 2012. Most of these implementations [6], [7], [8] are optimized for high throughput and few are known about compact designs [9]. In terms of area, the design in [6] has the lowest hardware resources utilization. Many hardware designs for FPGAs have been published during the SHA-3 competition. To the best of our knowledge, only three generally different architecture types of the KECCAK algorithm have been implemented and published in the literature so far, while the KECCAK designers proposed more possibilities to process the state in hardware. Most design compute the compression function using a fully parallel data path to reduce the number of clock cycles to a minimum. These implementation focus on maximizing the throughput, reaching up to 13 GBit/s. Variants of this architecture type are pipelined or process more than one data stream in parallel.

For lightweight implementations, a lane-oriented architecture is favored in most of the proposed designs, The only alternatively implemented design strategy for implementations of KE C C A K is a slice-oriented architecture as proposed in , This implementation strategy was also followed by the recent and more extensive study in ..
For ASICs, there are also a lot of reported results. Implementations of KECCAK with the full and reduced states. The area of the full KECCAK-f[1600] was pushed further to consume even less area.

3. KECCAK

3.1Basic Block Diagram Of Keccak

It is based on the sponge construction, so Keccak can be considered as a family of sponge functions [10][11][12]

The aims of using the sponge construction are to have a provable security against generic attacks and to make the use of compression function more simple, flexible, and functional.

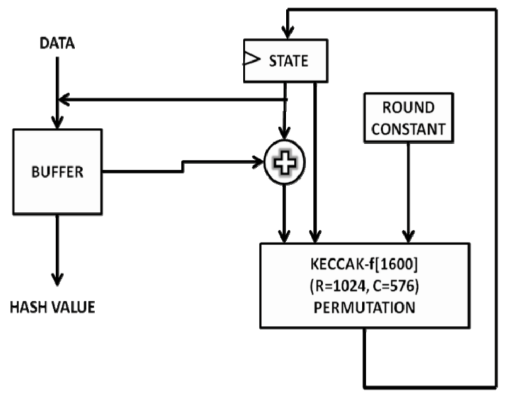

Fig -1: Block Diagram Of KECCAK Algorithm.

In sponge construction model, there are two portions for the internal stage registers. Also, there are two phases, absorbed and squeezed.

The input message will be XORed with the data stored in the first portion of the internal stage registers during the absorption stage. And then the resulted value of the XORing process will be updated along with the data stored in the second portion of the internal stage register. During the squeezing phase, the data of the first portion will be used as a part of the output. It is worth mentioning that the sponge construction can accommodate the output of any size by updating the internal stage register.

3.2Keccak Parameters

Keccak is recognized as a new Secure Hash Algorithm-3 i.e. SHA-3 [3] announced by NIST. Gilles Van Assche, Guido Bertoni, Michael Peeters and Joan Daemen designed and proposed the construction of Keccak Hash function. The Keccak-f permutation is the basic component of Keccak Hash function and supports 224 -bit, 256-bit, 384-bit and 512-bit hash variants. It consists of number of rounds and each round is the combination of logical operations and bit permutations. Keccak is generated from sponge function with Keccak [r, c] members. This means, it can be Parameterized by the state size b, the rate **r** and the capacity c. The three parameters are interdependent, i.e. b = c + **r** and thus, changing one parameter changes at least one other parameter. A forth important parameter is the size n of the message digest. The parameters b and **r** determine the performance of KE C C A K, whereas c and n are important security parameters.

The addition of r + c gives width of the Keccak function permutation and is it is further limited to values as indicated 25, 50, 100, 200, 400, 800, 1600. The Keccak team introduced the Keccak [1600] function for SHA3 proposal with different values of 'r' and 'c'. Keccak [1600] was selected because of its increased number of rounds in order to provide improved security margin. For 256-bit hash value r = 1088 and c = 512. For 512-bit hash output, the values of r and c are 576 and 1024 respectively. The 1600-bit state matrix of Keccak composed of 5x5 matrixes of 64-bit words. Initially, the message block should undergo the inversion procedure so that last byte should come first and first byte should become last.

It variant with a reasonable amount of area on modern FPGAs. However, several applications emphasis lowers costs and thus area over throughput and security. Therefore, the security requirements may be lowered for these applications to achieve less implementation cost. Therefore, we will also analyses variants of KE C C A K **-f** [b]with b $\in$ {200, 400, 800}. The variants still have reasonable security for many applications.

3.3. Security Parameters

There are major security claims presented for sponge construction and KECCAK [13][14], which can be used to derive meaningful parameter sets. The most interesting security properties of hash functions are the collisions, the preimage and the second pre image resistance. The exact claimed security bounds in terms of hash digest length n and capacity c are:

- Collision resistance: $O(\min(2^{n/2}, 2^{c/2}))$
- Preimage resistance: $O(\min(2^{n}, 2^{c/2}))$
- Second preimage resistance: $O(\min(2^{n}, 2^{c/2}))$

Note, that the security assumed for the preimage resistance is claimed to be higher by some publications, because no generic attack is known which is as good as the theoretical bound, The Security parameters used in the present evaluation are partially derived from the Photon specifica-tion adapted to KE CC A K, with the exception of the smallest Photon variant.

Additionally higher security versions are evaluated for the message digest sizes, where the state size allows a theoretically optimal reimage resistance. All evaluated parameters are presented in Table 1. Using the same Photon parameter sets has the main advantage that it is easy to replace one hash function with the other and it will be easier to compare the performance of both algorithms. Similar, but slightly different parameters are used by the Sponging hash function. For the ASIC designs, only a subset of these options is investigated for now, because of the long time needed to evaluate all possibilities.

3.4Operation of Keccak Hash Function

The operation of Keccak hash function comprised of three different stages i.e. first initialization followed by absorbing and finally squeezing stage as shown in Fig. 1, where M is the message and Z is the hash output. During initialization phase, every single bit of state matrix 'A' is initialized with zero.

In 2nd stage i.e absorbing stage, all r-bit wide block of messages is XORed with recent matrix state, in this way 24 rounds of Keccak permutation are accomplished sequentially. When absorbs every block of initial message taken as input block comes in that sequence.

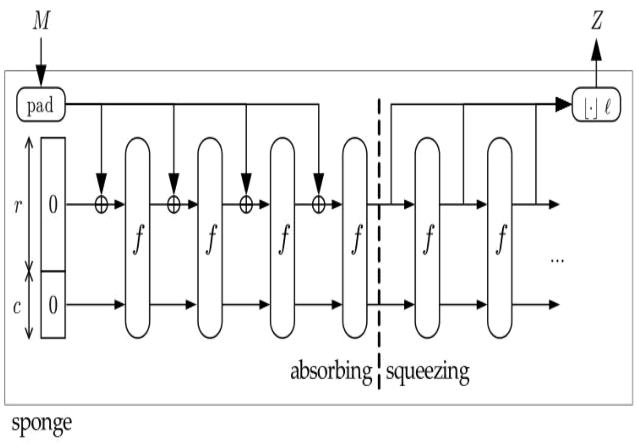

Fig-2: The Sponge Function

Finally, in squeezing stage, matrix state is basically truncated to attainable length of output hash and extra bits are removed from current state matrix to achieve desired hash length. If hash value is required more than r-bit (bit-rate), then additional Keccak permutations are actively achieved and further their results are awaited until hash width attains its desired length.

3.5 Single Compression Box

Every single compression function of Keccak composed of 24 rounds and each round is sub-divided into five steps i.e. Theta (Θ), Rho (ρ) and Pi (π), Chi (χ), Iota (i) explained in below section.

a. Theta (Θ) Step

Theta function comprises of three equations that involves simple XOR and bitwise cyclic shift operations. Equation (1) involves the XOR operation between lanes (set composed of 64-bits along the constant x and y co-ordinates) of each row of the state matrix A that results in five output lanes.

$$C[X] = A[X,0] \oplus A[X,1] \oplus A[X,2] \oplus A[X,3] \oplus A[X,4] \quad 0 \le X \le 4 \qquad (1)$$

Initially left circular shift will be applied on the five output lanes in such a way that last lane becomes first and second last lane becomes last lane in (2). After that right circular shift will be carried out on the lanes so that first lane becomes the last and second lane becomes the first lane and then left circular shift will be applied on each lane in order to change the positions of the bits within each lane

$$D[X] = C[X-1] \oplus \mathrm{ROT}(C[X+1,1]) \qquad 0 \le X \le 4 \qquad (2)$$

Equation (3) of Theta just involves XORing between the input state matrix and output lanes obtained from (2).

$$A[X, Y]' = A[X, Y] \oplus D[X] \qquad 0 \le X, Y \le 4 \qquad (3)$$

b. Rho (ρ) and Pi (π) Step

The next two steps Rho (ρ) and Pi (π) can be expressed jointly by (4) that compute an auxiliary 5 x 5 array B from the state array 'A'. The operation of Rho (ρ) and Pi (π) take each of the 25 lanes of the state array 'A', perform circular rotation on it by the fixed number of values depending upon the 'x' and 'y' co-ordinates i.e r[x, y] given in Table I [3] (called Rho (ρ) step) and then place the above rotated lanes at the different location in the new array B (called Pi (π) step). Note that all the indices are taken modulo 5.

Table-1: The Cyclic Shift Offsets "R(X,Y) for Keccak

	X=3	X=4	X=0	X=1	X=2
Y=2	25	39	3	10	43
Y=1	55	20	36	44	6
Y=0	28	27	0	1	62
Y=4	56	14	18	2	61
Y=3	21	8	41	45	15

$$B[Y, 2X + 3Y] = \mathrm{ROT}(A[X, Y], r[X, Y]) \quad 0 \le X, Y \le 4 \qquad (4)$$

c. Chi (χ) Step

The Chi (χ) step operates on the lanes, i.e. words with 64-bits and manipulates the B array obtained in the previous Rho (ρ) and Pi (π) step and replaces the result in the state array A. We can say that the Chi (χ) step takes the lane at location [x,y] and XOR it with the logical AND of the lane at address location of [x+1,y] and the complement at location [x+2,y]. Following equation is illustrating the function Chi (χ).

$$A[X,Y] = B[X,Y] \oplus ((\mathrm{NOT}\ B[X+1, Y]) \ \mathrm{AND}\ B[X+2, Y]) \quad 0 \le X, Y \le 4 \qquad (5)$$

d. Iota(i)step

The Iota step is the simplest step of Keccak algorithm. just performs the XOR operation of predefined 64-bit constant RC given in [3] with the lane at location [0,0] of the new state matrix 'A'.

$$A[0,0] = A[0,0] \oplus RC \qquad (6)$$

4. IMPLEMENTATION

In this work, we present an iterative design of SHA-3 512-bit for compact implementation as shown in Fig. 2. The architecture has 128-bit input data just to save extra input bits. The next block in proposed design is padder block which pads the required number of zeros with the input data in order to form 1600-bit state and then inversion is applied on each byte. The output from the padder block is forwarded to 2 x 1 Multiplexer (MUX) which drives the output data from padder to the compression-box of the architecture and selects the input data for the first round and feedback data for other twenty three rounds of Keccak with the help of controlling signal (Ctrl 1).

When Ctrl 1 is low, MUX select the input data and at high, MUX will select the feedback data. First padded message is directly copied to Reg A which previously initialized with all zeroes and resulting bits are forward to Compression-Box (C-Box). It is basically the implementation of compression function in SHA-3 algorithm which comprises of theta (Θ), rho (ρ), pi (π), chi (χ) and iota (i) step. For performance, we logically optimized our design by implementing rho (ρ), pi (π) and chi (χ) steps as a single step. This results in saving of hardware resources in term of 48 slices. After completing 24 iterations, final output is forwarded to Reg B for storage in order to synchronize the data-path. The last component in the architecture is Truncating component where inversion per byte is performed on the output bits and then truncated to the desired length of hash output.

4.1Implementation Of Compression-Box

The details of the implementation for each step are given as follow:

a) Theta **Step (Θ):** Theta step consists of three main steps in terms of equations that mainly require bitwise XOR operation.quation (1) involves bitwise XOR operation between the 64-bit lanes of each row where every lane of each row is independent of each other so parallel operations can be applied on these lanes. We have used conventional 64-bit XOR operator in parallel to perform XORing between the five lanes in each row of the state array 'A' and results are stored in intermediate registers. The above parallel XOR operations make our design fast and more efficient in terms of performance. Second step (2) of step theta involves one bit left circular rotation which is accompanied by simple rewiring or replacing the bit pattern of each row, then XORed with the previous output lanes. The results are stored in an intermediate registers in the form of five lanes. These lanes are again XORed with input state matrix A[x, y] to form new 5 x 5 state matrix A'[x,y]. All the operations are done on modulo 5.

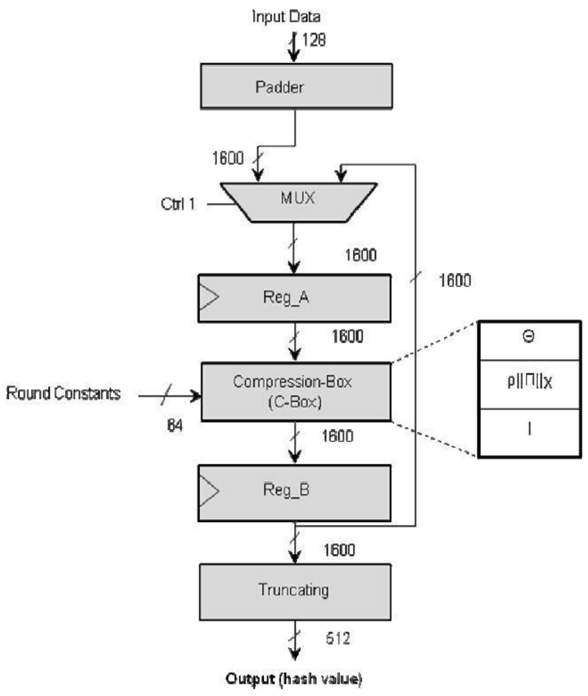

Fig -3:128-bit Keccak sequential architecture

b)Integrated Rho **(ρ), Pi (π) and Chi (χ) Step:** The rho and pi are basically permutations and each lane require cyclic shift by some fixed numbers according to the value of cyclic shift offset given in SHA-3 FIPS PUB 202. If this step is implemented separately, we would need extra 48 slices for the operation of rho and pi step. In order to save these slices we have logically optimized our design by merging the rho and pi step into the chi step. We have performed all the calculations required in rho and pi step manually and applied cyclic shift
on each lane of state matrix obtained at the output of theta step and relocated them after calculating its new position according to (4). For example, for X=1, and Y=2, (4) can be written as 1)

$$B[2,2(1) + 3(2)] = ROT(A[1,2] , r[1,2]) \qquad (7)$$

The value of r(1,2) is 10 according to cyclic shift offset table in FIPS PUB 202 and all the operations are done modulo 5, therefore (7) is transformed into (8).\

$$B[2,3] = ROT(A[1,2] ,10) \qquad (8)$$

In this way, we have performed calculations for all the possible combinations of the state matrix and placed these values directly in (5) of chi step. For X=1 and Y=3, (5) can be represented as follows shown by (9).

$$A[1,3] = B[1,3] \oplus ((\text{NOT } B[2,3]) \text{ AND } B[3,3]) \quad (9)$$

Putting the values of B[1, 3], B[2, 3] and B[3, 3] in (9).

$$A[1,3] = \text{ROT}(A[0,1], 36) \oplus ((\text{NOT ROT}(A[1,2], 10)) \text{ AND ROT}(A[2,3], 15)) \quad (10)$$

We have performed all the cyclic rotations manually and placed the results in (10) and then XOR, NOT and AND

The above logical optimization technique allow us to minimize the resources by 16 % that results in enhanced performance of our design. The diagrammatic view of integrated Rho, Pi and Chi step is shown in Fig. 3. It shows the implementation of integrated rho, pi and chi step, it can be easily seen that Rho and Pi permutations are applied directly during the implementation of Chi step by simple re-wiring.

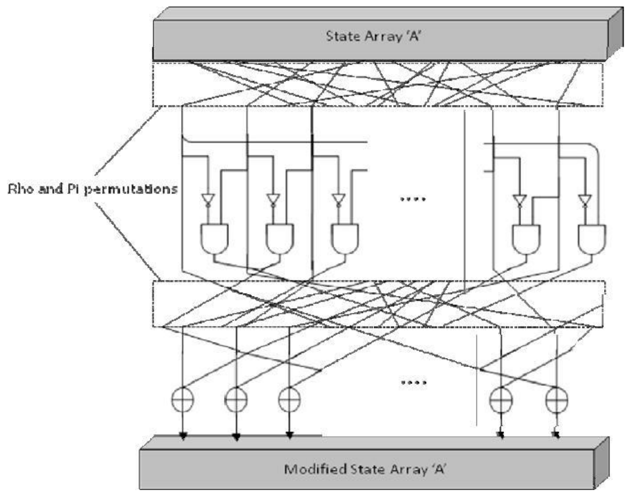

Fig-4: Integrated Rho, Pi and Chi

c) Iota (i): In Iota (i) step, we have used conventional 64 bit XOR operator to perform XORing between the least significant 64-bits of state array and round constant RC. The values of round constant are fixed and different for every round. These round constants are stored in registers.

5. CONCLUSION

In this work we have presented the design for compact hardware implementation of SHA3-512. We tried to keep balance tradeoff between area and throughput where our design presents best possible results both in term of area and throughput as compared to previous reported results. Our logical optimization by merging the three transforms i.e. rho, pi and chi in to single transform and by exploring maximum parallelism in the algorithm are contributing factor. This optimization results in overall reduced latency which significantly enhanced the system performance.

REFERENCES

[1] X. Wang, D. Feng, X. Lai, and H. Yu, “Collisions for hash functions md4, md5, haval-128 and ripemd,” IACR, August 2004.

[2] National Institute of Standards and Technology (nist), “Cryptographic hash algorithm competition,” 2007.

[3] FIPS-202, “Federal information processing standards publication fips-202, secure hash algorithm-3 (sha-3),” 2014.

[4] Xilinx, “Virtex 2.5 V field programmable gate arrays”.

[5] F.R. Henrquez, A.D. Prez, N.A. Saqib, and C.K. Koc, Cryptographic Algorithms on Reconfigurable Hardware. Signals and Communication Technology, Springer, 2007.

[6] S. Kerckhof, F. Durvaux, N.V. Charvillon, F. Regazzoni, G.M. de Dormale, and F.X. Standaert, “compact fpga implementations of the five sha-3 finalists,” Springer Berlin Heidelberg, vol. 7079, pp. 217–233, 2011.

[7] A. Akin, A. Aysu, O.C. Ulusel, E. Savas, “Efficient hardware implementations of high throughput sha-3 candidates keccak, luffa and blue mid night wish for single- and multi-message hashing,” ACM, pp. 168–177, 2010.

[8] K. Gaj, E. Homsirikamol, and M. Rogawski, “Comprehensive comparison of hardware performance of fourteen round 2 sha-3 candidates with 512-bit outputs using field programmable gate arrays,” 2nd SHA-3 Candidate Conference, pp 23-24, August 2010.

[9] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. ONeill, and W.P. Marnane, “FPGA implementations of the round two sha-3 candidates,” The second SHA-3 Candidate Conference, 2010.

[10] G. Provelengios, P. Kitsos, N. Sklavos, and C. Koulamas, “FPGA-based design approaches of keccak hash function,” 15th Euromicro Conference, pp. 648–653, 2012.

[11] K. Latif, M.M. Rao, A. Aziz, and A. Mahboob, “efficient hardware implementations and hardware performance evaluation of sha-3 finalists,” in Proceeding of 3rd SHA-3 Candidate Conference, march 2012.

[12] E. Homsirikamol, M. Rogawski, and K. Gay, “comparing hardware performance of round 3 sha-3 candidates using multiple hardware

[13] architectures in xilinx and altera fpgas,” ECRYPT II Hash Workshop, pp. 1–15, 19-20 May 2011.

[14] Bertoni, G.,Daemen, J.,Peeters, M.,Assche, G.V.: The Keccak Reference.Online Publication(2011).

[15] Bertoni, G,. Daemen, J., Peeters , M.,van Assche, G,:Cryptographic Sponge functions.

BIOGRAPHIES

M.M.Sravani was born in Chittoor, Ap,India She has obtained her B Tech degree in Electronics and Communication from Audisankara College Of Engineering in 2012. Presently she is pursuing her Masters degree in VLSI System Design of Electronics and Communication in Siddharth Institute of Engineering & Technology, Puttur, from 2013 to 2015. She is interested in Cryptography in VLSI. she is currently working on a project titled "Compact Implementation Of SHA3-512 on FPGA" as a partial fulfillment of his M.Tech degree.

C H Pallavi was born at Chittoor, AP, and India. She completed her B.Tech degree from Srikalahsti Institute Of Engineering, Srikalahasti, Ap, India in 2009, and M.Tech in 2013 from the JNTUA University in VLSI System Design as specialization He is currently working as a Assistant Professor in Department of ECE in Siddharth Institute of Engineering & Technology, Puttur, AP, India. Her research interest includes Low power design, VLSI design, and embedded design. She has been published several papers in different various conferences.