Computers and Electrical Engineering 55 (2016) 138–152

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

A low-power SHA-3 designs using embedded digital signal processing slice on FPGA[☆]

Dur-e-Shahwar Kundi[*], Arshad Aziz

Department of Electrical Engineering (PNEC), National University of Sciences & Technology (NUST), H-12, Islamabad, Pakistan

ARTICLE INFO

Article history:
Received 17 April 2015
Revised 23 March 2016
Accepted 5 April 2016
Available online 20 April 2016

Keywords:
Cryptography
Secure hash algorithm-3 (SHA-3)
Field programmable gate array (FPGA)
Keccak
DSP48E slice
Logical cascade structure (LCS)

ABSTRACT

This work presents two low-power Secure Hash Algorithm-3 (SHA-3) designs on Field Programmable Gate Array (FPGA) using embedded Digital Signal Processing (DSP48E) slice, one for area constrained environments and the other for high-speed applications. The seven equations of SHA-3 are logically optimized to three and four stage pipelined organizations for our compact and high-speed designs, respectively. The maximum parallelism between all the bitwise operations of different stages of SHA-3 is explored with respect to the 48-bit structure of DSP slice. Further Logical Cascade Structure (LCS) design strategy is proposed in accordance with the DSP slice organization. These optimizations result in saving of resources and at the same time achieve low-power with high performance. Our compact design results in saving of 79.10% DSP slices and consumes only 1/7th of power while 1600-bit DSP design provides 23.57 Gbps throughput and consumes only 1/5th of power as compared to the conventional SHA-3 designs.

1. Introduction

Secure Hash Algorithm-3 (SHA-3) was announced as a new cryptographic hash standard by National Institute of Standard & Technology (NIST) in 2012 [1] and the same was standardized in August 2015 as FIPS-PUB-202 [2]. It was designed to provide more security strength as compared to the previous hash functions that are nowadays vulnerable to pre-image and collision attacks [3,4]. Hardware implementations of SHA-3 on Field Programmable Gate Array (FPGA) are gaining more importance in order to meet real-time constraints such as high-speed (data rates) and low-power consumption of emerging systems. On communication platform like Internet of Thing (IoT) and in IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN), the power consumption of SHA-3 core is a critical issue while in wired and wireless data networks, high data rates are also important as well. Similarly the same attributes are the major implementation requirements for all modern applications. Therefore there is a need for an efficient FPGA based SHA-3 core that has low-power consumption and also provides high data rates in order to meet the demanding requirements of modern applications.

All modern FPGA devices, apart from standard Slices & Look-Up-Table (LUT) primitives, are now also equipped with dedicated embedded resources like Block RAM (BRAM) and Digital Signal Processing (DSP) slices. These embedded resources can result in high performance and low-power designs, if efficiently explored and utilized. The basic structure of DSP slice i.e. DSP48 is almost same in all the series of Xilinx FPGAs with some enhancements and additional features along with Multiply

[☆] Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. R. Cumplido.
[*] Corresponding author.
E-mail address: shahwar@pnec.nust.edu.pk (D.-e. Kundi).

http://dx.doi.org/10.1016/j.compeleceng.2016.04.004
0045-7906/© 2016 Elsevier Ltd. All rights reserved.

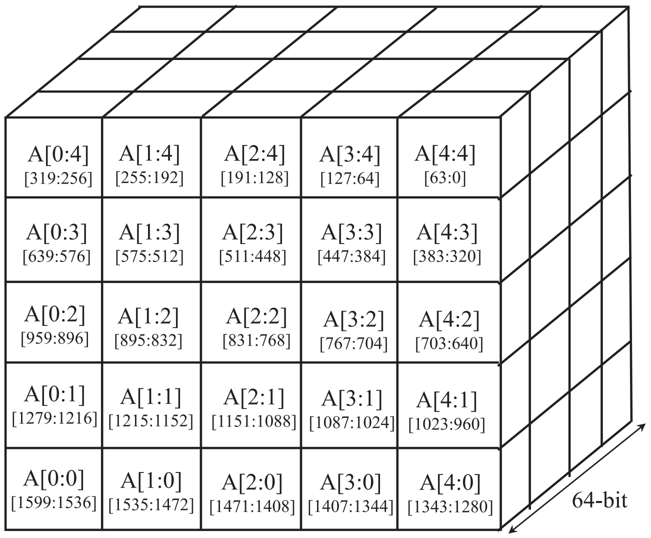

Fig. 1. SHA-3 1600-bit state matrix.

Accumulate (MAC) stage. Apart from MAC operation, these enhanced features can also be utilized to perform different kinds of logical as well as arithmetic functions [5].

Earlier, the number of DSP slices on Xilinx FPGA fabric were very less. However, with the advancement in VLSI technology their numbers are increasing and are now available in abundant quantity in all the newer generations of FPGA families such as Xilinx 7 Series [6], UltraScale & UltraScale+ Series [7]. There are many application areas in which we require very less MAC functionality but we have lot of logic intensive computations to implement on same FPGA. In those particular scenarios these DSP resources remain unutilized during the design process, although they are physically present on FPGA chip and at the same time, the logical resources of FPGA i.e. LUT primitives & Slices are entailed for the implementation of other modules of complete system. Cryptographic accelerator card is an example of such system in which we require encryption core like Advanced Encryption Standard (AES), hash core like SHA-3 as well as complete communication system on the same fabric.

Therefore, we require different design methodologies to port these logic intensive computations from conventional LUT based platform to other embedded FPGA components like DSP slices. Our proposed method in this work enabled us to shift the logical intensive functions like XOR, AND & NOT of SHA-3 algorithm from LUT primitive to these unused DSP slices and provide feasibility to save more logical resources for the implementation of other parts of system on the same chip. In addition to this, these DSP slices are low-power components as compared to their logical counterpart and have reduced set-up and clock-to-out timing characteristics. These are available in the form of DSP titles and have dedicated low-power interconnect between two cascading DSP slices that can be utilized for low-power consumption. Hence, in this work we provide low-power and resource-efficient SHA-3 cores; one for the area constrained environments and the other for the high-speed applications thereby effectively utilizing every feature of Xilinx DSP slices.

The rest of the paper is organized as follows, Section 2 gives a brief description of SHA-3 compression function whereas Section 3 focusses on the architecture of DSP48 slice of Xilinx FPGA. Section 4 presents a literature survey on SHA-3 implementations while in Section 5, we provide the details of our proposed DSP based SHA-3 designs. Implementation results of our proposed designs and their comparison with previously reported SHA-3 implementations are given in Section 6. Finally, Section 7 concludes our work.

2. SHA-3

Keccak-*f*[1600] variant of SHA-3 consists of 1600-bit of input state array that is arranged in the form of 5 × 5 matrix of 64-bit words each, as shown in Fig. 1. *A[0,0]* denotes number of bits from 1599 to 1536 (64-bit word), *A[1,0]* is from 1535 to 1472 (64-bit word) and so on as per definition of its standard [2].

The permutation function of Keccak-*f*[1600] uses a round function that comprises of five step mappings with seven computing equations as follows.

θ-Step: $(0 \le x, y \le 4, 0 \le z \le 63)$

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

$$D[x, z] = C[(x-1), z] \oplus ROT(C[(x+1), 1]); \tag{2}$$

$$A^{'}[x, y, z] = A[x, y, z] \oplus D[x, z]; \tag{3}$$

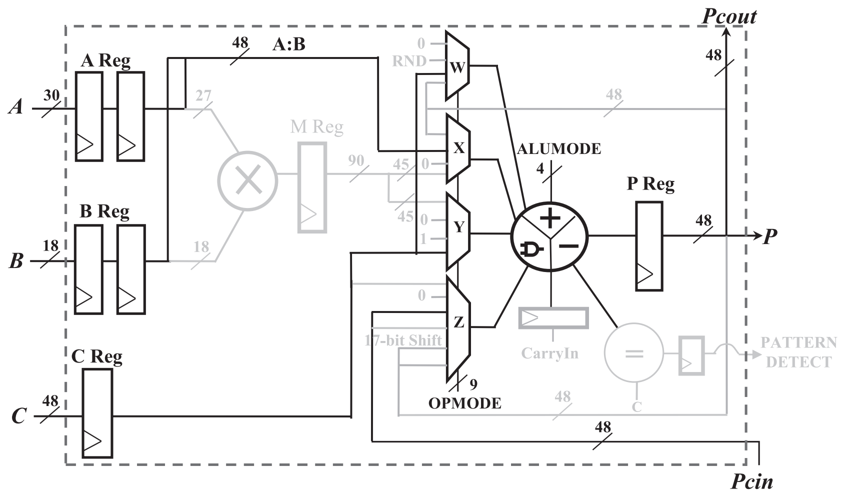

Fig. 2. DSP48E2 architecture.

ρ-*Step*: $(0 \le x,\ y \le 4)$

$$A[x, y, z] = ROT(A^{'}[x, y, z], r[x, y]); \tag{4}$$

π-*Step*: $(0 \le x,\ y \le 4,\ 0 \le z \le 63)$

$$B[y, (2x + 3y), z] = A[x, y, z]; \tag{5}$$

χ-*Step*: $(0 \le x,\ y \le 4,\ 0 \le z \le 63)$

$$A^{'}[x, y, z] = B[x, y, z] \oplus (NOT(B[(x + 1), y, z])AND(B[(x + 2), y, z])); \tag{6}$$

ι-*Step*: $(0 \le z < 63)$

$$A^{'}[0, 0, z] = A[0, 0, z] \oplus RC[z]; \tag{7}$$

In above five steps; parameters *x* and *y* signify the operations within the range of modulo 5 while *z* is of modulo 64. The *A[x, y, z]* represents a particular 64-bit lane of state while *B[x, y], C[x]* and *D[x]* are the intermediate results. The five steps consist of different bitwise operations such as *XOR* (⊕), *NOT* and *AND* logical operations. In addition to these operations, SHA-3 also comprises of bitwise cyclic shift operator (i.e. *ROT*) and *XORing* of 64-bit Round Constant (*RC*) with first lane (*A[0,0,z]*) only. The *ROT* rotates the number of bits within a particular lane (64-bit) by a constant value of *r[x, y]* while 24 rounds of SHA-3 have different values of *RC*.

3. Xilinx DSP48 slice

A DSP48 slice is available in all the modern Xilinx FPGAs that can be used to perform different kinds of logical and arithmetic operations. The logical operation capability of these slices makes them more suitable for the design of efficient cryptographic hash cores in which all the operations are logical. A DSP48 slice is a built-in embedded component of Xilinx FPGA device families, such as DSP48E primitive is available in Virtex-5 [5], DSP48E1 is available in Virtex-6 & all 7 Series [8] while DSP48E2 is available in the newest UltraScale & UltraScale+ Series [9] of Xilinx FPGA.

The number of DSP48 slices increase with the device size and family and are organized in the form of DSP tiles that are stacked vertically in columns. A DSP48 slice of Xilinx FPGA supports many absolute functions including multiply, multiply accumulate (MACC), three input addition, barrel shifter, bit-wise logic operations and many other mathematical functions. Multiple DSP48 slices can be cascaded to implement complex arithmetic and extensive mathematical functions. DSP48 slice offers upgraded proficiency of applications, improved utilization and decreased overall power consumption.

The architecture of DSP48E2 slice in UltraScale FPGA is shown in Fig. 2. It has three direct inputs *A*, *B* and *C* of 30-bit, 18-bit and 48-bit respectively and one cascaded 48-bit input *Pcin*. Inputs *A* and *B* can be concatenated as [*A:B*], if design needs to implement 48-bit wide logic operation whereas *A[29:0]* forms the MSB while *B[17:0]* forms the LSB. These three inputs in DSP48E2 slice can be selected with the help of an appropriate OPMODE setting; *A:B* input via *X* multiplexer, *C* input via *W* or *Y* multiplexer, and *Pcin* via *Z* multiplexer while in DSP48E1 we can only select two inputs *A:B* input via *X* multiplexer and *C* or *Pcin* via *Z* multiplexer. Similarly, it has 48-bit output that is available at both the *P* and *Pcout* ports, where output from *Pcout* port can only be provided to *Pcin* port of the next DSP slice.

The logical functionality is defined in adder/subtracter/logic unit of DSP48 slice by using ALUMODE settings that enable to implement 48-bit *XOR, NOR, AND, NOT, NAND* and combination of these logic operations. Furthermore, this DSP48 slice also contains internal Input/Output registers as shown in Fig. 2, that can be used for pipelining of SHA-3 design to further improve the performance with no additional hardware cost.

4. Literature review

There are numerous implementations of SHA-3 available in open literature both on software and hardware platforms. Software implementations of SHA-3 are generally slow to be used for real time applications while hardware implementations on Application Specific Integrated Circuits (ASICs) [10–13] and FPGA [14–24] are more efficient ones in order to meet the real time constrains of emerging systems such as area and power. The FPGA based designs provide both time and cost effective solutions as compared to the ASICs and at the same time have the ability to be re-programmed on the fly.

Most of the FPGA based SHA-3 designs [14–19], are based on the utilization of conventional logical resources (LUT and Slices) of FPGA. In [14], Latif et al. proposed a high speed implementation of SHA-3 on FPGA using LUTs primitives and reports the highest Throughput (Tp) of 13.67 Gbps and hardware efficiency i.e Throughput per Area (TPA) of 14.94 by utilizing 913 Slices on Virtex-6. For high speed applications, the Tp of SHA-3 core is further increased at the cost of increased Slices by using pipelining techniques [15,16] and multiple hardware architectures [17]. In [15] Akin et al. report 22.33 Gbps with 4356 Slices, in [16] Athanasiou et al. report 19.1 Gbps with 1649 Slices while Michail et al. in [17] report 37.632 Gbps with 4117 Slices. Increasing Tp of LUT based SHA-3 designs will also consume more power.

In [18] Rao et al. present LUT based SHA-3 design for IoT applications. On Virtex-6, they reported 1048 Slices with the power consumption of 2026 mW at 8.83 Gbps while on Kintex-7, their core consumed 612 mW power at 9.36 Gbps with 1185 Slices. In [19], Honda et al. have also reported power consumption of 1000 mW/Gbps and 800 mW/Gbps for their LUT based SHA-3 core on Virtex-6 and Kintex-7 FPGA respectively. Therefore, limitation of all these conventional LUT based SHA-3 designs is that their Tp is directly proportional to the power consumption. This huge amount of power consumption is not suitable for the power constrained environments. Therefore low-power SHA-3 designs based on dedicated FPGA resources like DSP slices and BRAMs are presented in [20,21] to overcome this drawback.

In [20] & [21] authors provide the BRAM based lightweight implementation of SHA-3 for low-area applications. Both the designs used BRAM to store round constants and state. In [20], Kaps et al. utilized 630 Slices and 1 BRAM with Tp and TPA of 35 Mbps and 0.05 respectively. Their design results in power consumption of 24 mW (dynamic power) on Spartan-3 FPGA. The design in [21] has further reduced the number of logical resources thereby utilized 3 BRAMs and 151 Slices using coprocessor approach. Furthermore their architecture is capable of processing multiple instructions that improved Tp to 501 Mbps and TPA to 3.32 but they have not reported the power consumption.

In addition to above discussed embedded compact designs, Sharif et al. in [22] have presented two architectures; basic and embedded SHA-3 (Keccak) designs for high-speed applications. The basic architecture was implemented using FPGA logical resources while in embedded, they have used dedicated BRAM in addition to the Slices. The relative enhancement between the two architectures was not very noteworthy as they have stored only round constants of SHA-3 using BRAM. They saved 1% of Slices at the cost of 1 BRAM with 16% reduction in Tp. Their embedded design resulted in utilization of 1338 Slices and Tp of 11.252 Gbps.

Apart from BRAM based SHA-3 designs, there exists two SHA-3 designs in open literature that have explored other dedicated resource of FPGA i.e. DSP slice. In [23] Provelengios et al. investigate SHA-3 implementation using both logical as well as DSP resources of FPGA. In their DSP based design they used 201 DSP slices and instead of saving logical resources they used additional logical 3176 Slices with increased circuit complexity. They fragmented the 64-bit operations of Keccak*f*[1600] variant of SHA-3 into two 32-bit operations to fit into one 48-bit DSP slice. This results in the increased number of DSP slices as well as its inefficient utilization and their DSP based SHA-3 design in pipelined architecture result in very reduced Tp of 0.43 Gbps and TPA of 0.31.

In more recent DSP based implementation [24], the author evaluates the performance of SHA-3 pipelined architecture by using different optimization techniques such as retiming and organization of pipeline stages in order to improve the TPA. They have implemented only Eqs. (2) and (3) of SHA-3 using these DSP slices instead of whole SHA-3 algorithm due to shortage of DSP slices on selective FPGA device. They reported overall hardware improvement 52% in terms of TPA on Virtex-5 from previously reported pipelined architecture of [15].

Nowadays nearly all the series and sub-series of FPGA devices have these embedded BRAM and DSP slices and in most of the conventional SHA-3 designs, they remain unutilized. Furthermore, the SHA-3 implementations based on these embedded resources [20–24] are unoptimized and have room for the improvement to make these SHA-3 implementations more suitable for power constrained applications. So far internal capabilities of these resources have not been explored completely in order to design an efficient SHA-3 core. These embedded resources (BRAM and DSP) of FPGA have also internal dedicated I/O registers that, if properly utilized are the appropriate choice to design pipelined SHA-3 architecture for high-speed applications.

5. Our design implementation

Main objective of this work is to present low-power SHA-3 cores by effectively utilizing the internal capabilities of embedded DSP slices of Xilinx UltraScale FPGA. The first DSP based compact SHA-3 core is designed for area constrained environments while second SHA-3 core is provided for high speed applications which is based on five times replication of the basic construct with some modifications. Both SHA-3 cores are implemented in Logical Cascade Structure (LCS) having parallel pipelined architectures with following design considerations.

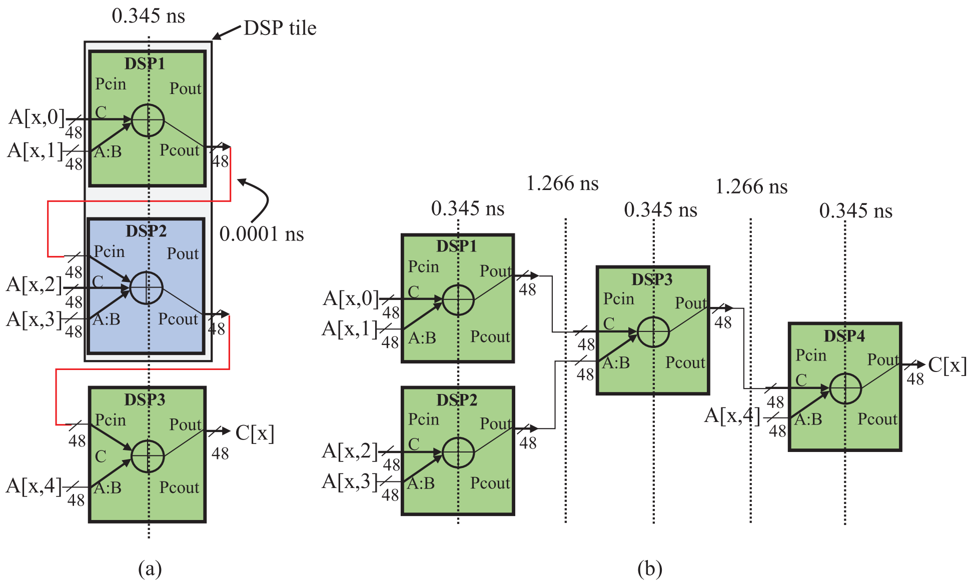

Fig. 3. (a) Logical cascade structure (b) Logical tree structure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

First, Logical optimization techniques are applied to SHA-3 algorithm, in which the seven set of equations are integrated within each other to reduce the latency as well as the hardware requirements. We used three stage organization for compact SHA-3 core while four stage organization for 1600-bit SHA-3 core that saved additional 29% DSP slices as compared to the hardware of five times replication. Both the designs are pipelined by using internal pipelined registers of DSP48 slice at free of cost that not only enhanced the performance of SHA-3 design but also effectively utilized the DSP slices of FPGA. Further, retiming and proper organization of these pipelined registers in 1600-bit architecture enabled us to hash multiple messages at the same time that, in turn increased the Tp with no additional hardware overhead.

Secondly, keeping in view the internal data path of DSP48 slice i.e. 48-bits, maximum achievable parallelism is explored between the logical operations of SHA-3 algorithm to implement all the bitwise logical operations in full 48-bit parallel fashion. Additionally, a 48-bit triple input *XOR* logic operation is implemented by a single DSP48E2 slice otherwise it will require an extra DSP slice. This is done by the appropriate OPMODE & ALUMODE setting of the DSP48E2 slice which enabled us to select three operands; *A:B* input via *X* multiplexer, *C* input via *Y* multiplexer, and *Pcin* via *Z* multiplexer. Further, different sets of equations within one step are executed using parallel hardware in order to get enhanced performance.

Lastly, the logic of SHA-3 is implemented in the form of LCS design strategy to get maximal performance from DSP48 slice instead of Logical Tree Structure (LTS). In Xilinx FPGAs, DSP slices are available in the form of pairs in DSP tiles with dedicated low-power cascading interconnect i.e. cascading output (*Pcout*) of upper DSP slice is connected to the cascading input (*Pcin*) of lower DSP slice. This dedicated interconnect within DSP tiles not only connects two DSP slices internally but also provides a connection to the other DSP slice within the same column [9] of FPGA. This dedicated cascade capability of DSP slices is highly efficient to implement low-power and high-speed pipelined designs in LCS design strategy. The use of this dedicated interconnect between the DSP48 slices wherever possible, not only enabled us to restrict maximum logic of the SHA-3 design to one column of Xilinx FPGA but also minimized the number of DSP slices.

Therefore, for the comparison, the Eq. (1) of SHA-3 is implemented in both structures as shown in Fig. 3. The Fig. 3(a) represents the implementation of Eq. (1) using LCS design strategy in which dedicated interconnect paths (as shown by red color) are fully utilized to route the logic while in Fig. 3(b), the logic of Eq. (1) is implemented in LTS design strategy in which DSP slices are now connected through external routing paths. Both design strategies are then analyzed for logical and routing delays as taken from Xilinx timing analysis report. From report, logical delay of one DSP slice is about 0.345 ns and routing delay of dedicated interconnect path from *Pcout* to *Pcin*, as shown in Fig. 3(a), is almost zero i.e. 0.0001 ns while that of external path from *Pout* to either *A:B* or *C* as shown in Fig. 3(b) is about 1.266 ns. The total critical path delays in LCS implementation strategy come out to be 1.0353 ns ($3 \times 0.345 + 2 \times 0.0001$), while in LTS are 2.878 ns ($3 \times 0.345 + 2 \times 1.266$) with additional usage of one more DSP slice.

Hence, it is clear that proper implementation of SHA-3 logic (that is in LCS design strategy) in accordance with the DSP48 slice organization in Xilinx FPGA results in both reduced hardware as well as critical path delays.

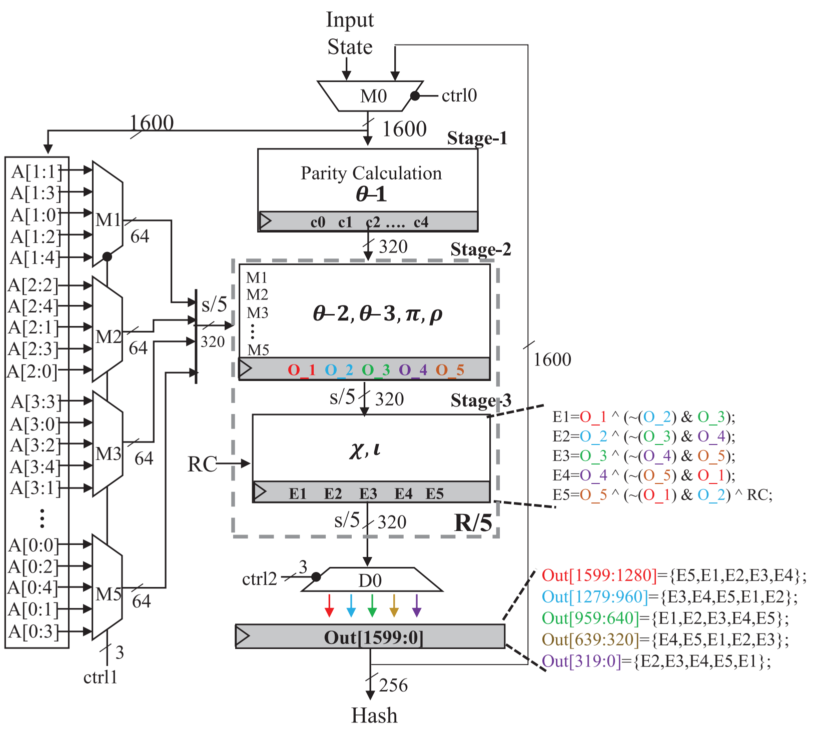
Input State
M0
ctrl0
1600
Stage-1
Parity Calculation
θ-1
c0 c1 c2 .... c4
320
Stage-2
M1 M2 M3 ⋮ M5
θ-2, θ-3, π, ρ
O_1 O_2 O_3 O_4 O_5
s/5 320
Stage-3
χ, ι
E1 E2 E3 E4 E5
RC
R/5
E1=O_1 ^ (~(O_2) & O_3);
E2=O_2 ^ (~(O_3) & O_4);
E3=O_3 ^ (~(O_4) & O_5);
E4=O_4 ^ (~(O_5) & O_1);
E5=O_5 ^ (~(O_1) & O_2) ^ RC;
ctrl2 3
D0
Out[1599:0]
Out[1599:1280]={E5,E1,E2,E3,E4};
Out[1279:960]={E3,E4,E5,E1,E2};
Out[959:640]={E1,E2,E3,E4,E5};
Out[639:320]={E4,E5,E1,E2,E3};
Out[319:0]={E2,E3,E4,E5,E1};
256
Hash
A[1:1] A[1:3] A[1:0] A[1:2] A[1:4]
A[2:2] A[2:4] A[2:1] A[2:3] A[2:0]
A[3:3] A[3:0] A[3:2] A[3:4] A[3:1]
A[0:0] A[0:2] A[0:4] A[0:1] A[0:3]
M1 M2 M3 M5
64
ctrl1 3

Fig. 4. Block diagram of compact SHA-3 design.

5.1. DSP based compact SHA-3 design

After implementing Eq. (1), the SHA-3 algorithm is further analyzed for reduced number of operations and its dependency among different steps in order to design a compact SHA-3 architecture. Therefore data path for remaining steps of this design is vertical folded by factor of 5 (i.e. R/5), in which 320-bit of SHA-3 state is processed at each clock cycle to produce final round output of 1600-bit in total of five clock cycles.

The block diagram of our DSP based compact SHA-3 design is shown in Fig. 4. The compression function of SHA-3 algorithm is computed in three stages and each stage is pipelined as indicated by the grey color. Stage-1 is implementation of Eq. (1) of θ step i.e. Parity Calculation. In Stage-2, Eqs. (2) and (3) of θ are logically combined and implemented together with ρ & π steps while in the last Stage-3, χ & ι steps of SHA-3 are realized together into a single hardware.

In this architecture the data path width is divided by a factor of 5 (i.e. s/5) for Stage-2 and Stage-3 in order to have symmetry with the output of Stage-1. The 1600-bit input state is provided to Stage-1 and at the same time distributed among the five multiplexers (*M1* to *M5*) according to π step of SHA-3 as shown in Fig. 4. In Stage-1, the 64-bit parity of each column of state is calculated in parallel to produce five 64-bit column parities; *c0* to *c4*. This 320-bit result of Stage-1 is then provided to Stage-2 where it is processed per clock cycle with only five selected lanes of the state i.e. 320-bit, out of 25 lanes. The multiplexers *M1– M5* select the five particular input lanes as required to calculate the output of particular row of state by using a 3-bit controlling signal *ctrl1*.

At *ctrl1* signal '*000*', the five lanes *A[1:1], A[2:2], A[3:3], A[4:4]*, and *A[0:0]* are selected via multiplexers; *M1, M2, M3, M4*, and *M5*, respectively for the first row. These five selected lanes are then *XORed* with two selected column parities of the previous stage by a single hardware to produce five 64-bit outputs; *O_1* to *O_5*. The output of Stage-2 is then provided to the Stage-3 in a particular rotated pattern with respect to the ρ step of SHA-3. In Stage-3, these five lanes are *XOR, NOT* & *AND* by a five set of equations in parallel to produce five 64-bit outputs (*E1– E5*) as shown in Fig. 4. The resulted five 64-bit outputs are then concatenated in a particular sequence to produce final 320-bit output. De-multiplexer *D0* saves this first 320-bit output in *Out[1599:1280]*, the second output in *Out[1279:960]* and so on. This takes total of five iterations to produce the final 1600-bit round output. After fifth iteration, the round output is provided back to the Stage-1 through feedback multiplexer *M0*. Detailed internal architecture of all stages of this compact SHA-3 design is shown in Fig. 5.

5.1.1. Stage-1

For the simplicity of implementation, the 5 × 5 state matrix of SHA-3 of Fig. 1 is first re-arranged in the form of a 5 × 7 state matrix of 48-bit lanes instead of 64-bit lanes as one DSP is capable of performing 48-bit operation. In 5 × 7 state matrix, *M[0:0]* represents 48-bits from 1599 to 1552, *M[1:0]* 1535 to 1488 while *M[2:0]* represents bits from 1471 to 1424. The remaining 16-bit each from first three lanes; (1551 to 1536), (1487 to 1472),(1423 to 1408) are concatenated to form

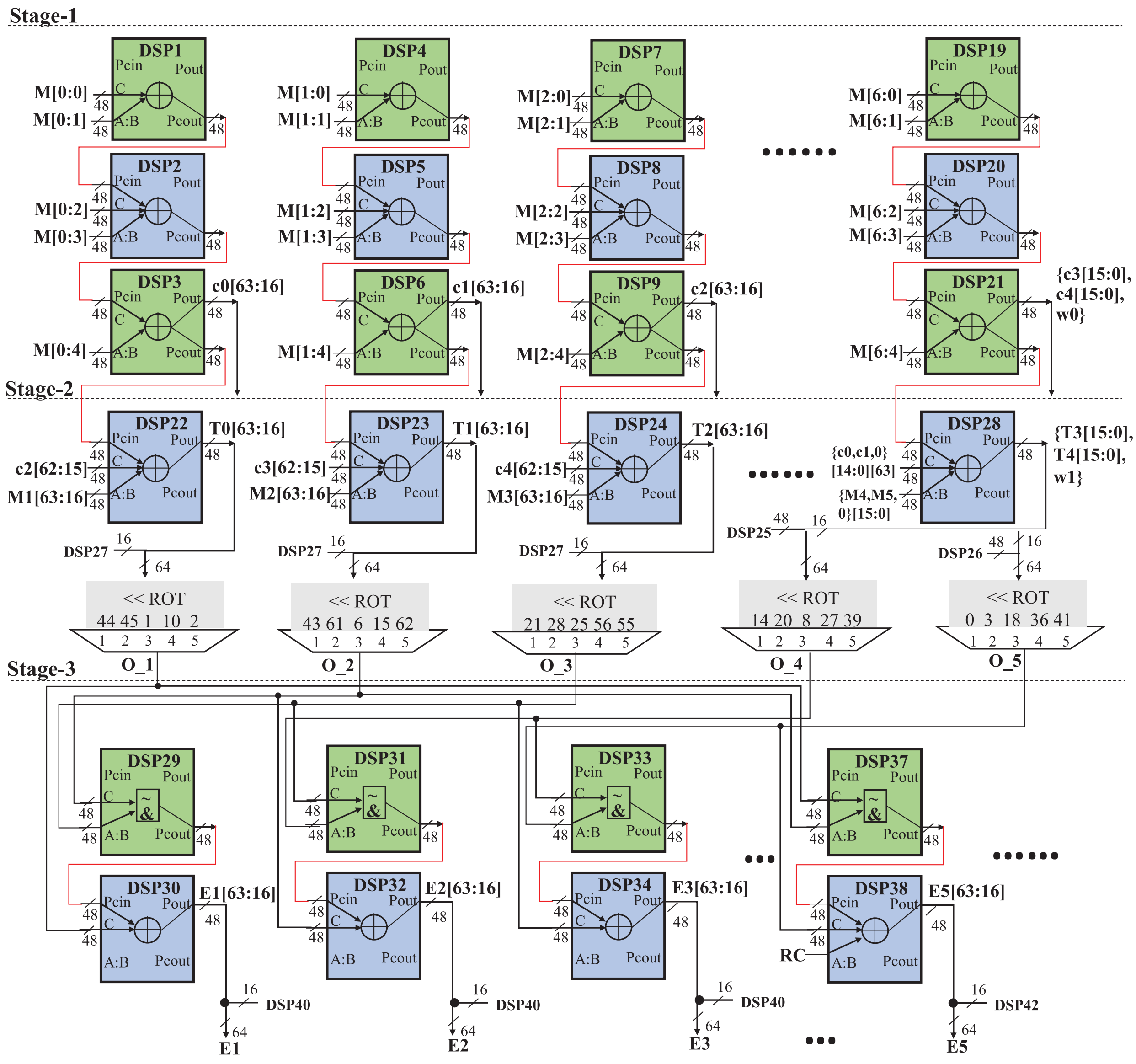

Fig. 5. Internal architecture of compact SHA-3 design.

the new 6th lane of the state i.e. *M[5:0]*. Similarly the 16-bit each from *A[3:0]* and *A[4:0]* lanes are concatenated with 16-bit zeros to form the 7th 48-bit lane i.e. *M[6:0]* of the state matrix. Now we have seven columns, each comprising of five 48-bit lanes as shown in Fig. 6.

In order to calculate parity for the first column, we need to *XOR* five 48-bit lanes (*M[0:0], M[0:1], M[0:2], M[0:3]*, and *M[0:4]*). One DSP48E2 slice, at maximum *XOR* three operands each of 48-bit by setting the appropriate values of *OPMODE* and *ALUMODE* but input at the *Pcin* port can only be provided through *Pcout*; the internal dedicated connection. So first two lanes *M[0:0]* & *M[0:1]* are *XORed* by first DSP Slice (DSP1) provided at its *C* and *A:B* port respectively as shown in Fig. 5. The output of DSP1 is taken from its *Pcout* port and provided at the *Pcin* port of DSP2 where it is further *XORed* with next two lanes *M[0:2]* & *M[0:3]* of 1st column. The output of DSP2 is also provided at the *Pcin* port of DSP3 in order to *XOR* it with last lane *M[0:4]* to produce the final result of first 48-bit MSB of *c0* i.e. *c0[63:16]*. In this way the five input lanes are provided in LCS strategy as DSP tiles stack vertically in the form of a column. The output of DSP3 is taken from its both ports; *Pcout* and *Pout*.

The 48-bit MSB of other columns (*c1, c2, c3* & *c4*) are also calculated in a similar manner, by utilizing 15 DSP48E2 slices from DSP1 to DSP15 as shown in Fig. 5. The parity calculation of last two column of 5 × 7 state matrix produces the remaining 16-bit LSB each of *c0, c1, c2, c3* & *c4* and further utilizes 6 DSP slices (from DSP16 to DSP21). The DSP21 produces 48-bit output in which the MSB 16-bit (i.e. *[47:32]*) is of *c3[15:0]*, the next 16-bit (*[31:16]*) is of *c4[15:0]* and the LSB 16-bit

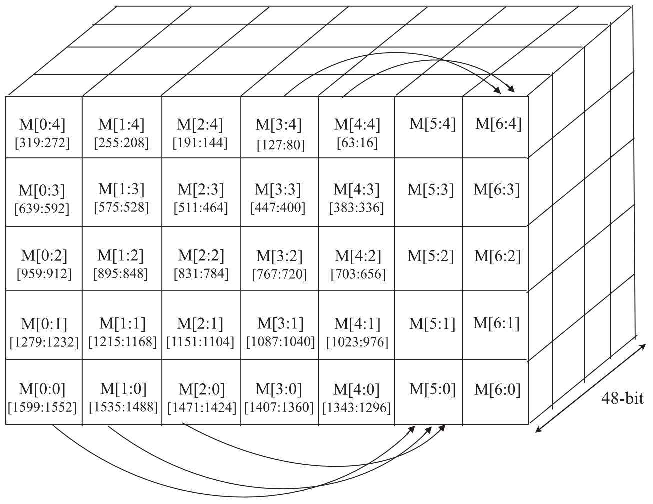

Fig. 6. Modified SHA-3 state matrix.

are zeros. So total of 21 DSP slices are used by this step in order to produce the five 64-bit column parities in full parallel fashion.

5.1.2. Stage-2

Here in this stage, we combined Eqs. (2) and (3) of θ step with π step into a single equation i.e. Eq. (8). The *A[x,y,z]* represents the input lanes of SHA-3, while *C[x-1]* & *C[x+1]* represent the column parities (*c0, c1, c2, c3* & *c4*) of previous step. In order to calculate the output of particular row, the input lanes are selected by multiplexers (*M1– M5*) in a defined order according to the π step as shown in Fig. 4. That's why in Eq. (8) '*x*' is replaced by '*(x+3y)*' while '*y*' is with '*x*' in order to incorporate the π step. For example to calculate output of *B[1:0]* lane, we require input lane *A[1:1]* at *M1* and output of column parities *c0(C[0])* and *c2(C[2])* by directly putting $x = 1$ & $y = 0$ in Eq. (8). This logical optimization technique enables us to reduce down the DSP Slice requirements to half and at the same time results in enhance performance.

$$\begin{aligned} B[x,y,z] &= A[x,y,z] \oplus D[x,y]; \\ &= A[x,y,z] \oplus C[x-1] \oplus ROT(C[x+1],1) \\ &= A[(x+3y),x,z] \oplus C[(x+3y)-1] \oplus ROT(C[(x+3y)+1],1) \end{aligned} \tag{8}$$

For the implementation of this step, we have used 7 DSP slices (DSP22– DSP28) as shown in Fig. 5 in which one DSP Slice *XOR* three operands simultaneously. In order to calculate the MSB 48-bit output *T0[63:16]* to *T4[63:16]*, MSB 48-bit of five input lanes selected by the five multiplexers (*M1–M5*) are provided at the *A:B* port of DSP22 to DSP26 respectively. Each input lane is then *XOR* with the two column parities as given by Eq. (8). The first 48-bit column parity from previous step is taken from internal cascaded path *Pcout* port while the second column parity is circular shifted by 1 is taken from the *Pout* port of Stage-1 DSP slices (DSP3, DSP6, DSP9, etc.) and are provided at the *Pcin* and *C* port of DSP22 to DSP26 respectively. The DSP27 and DSP28 produces the remaining LSB 16-bit each of *T0[15:0]* to *T4[15:0]*.

The five 64-bit outputs (*T0–T1*) are now rotated ($\ll$*ROT*) according to the ρ step of SHA-3 by using five multiplexers as shown in Fig. 5. The bits of each output are provided to the next stage in pre-defined order through these multiplexers. For first row of SHA-3 state, the five outputs are provided in 44, 43, 21, 14, 0 rotated order respectively to Stage-3 as *O_1* to *O_5*. Similarly for next row they will be provided in 45, 61, 28, 20, 3 rotated order respectively and so on.

5.1.3. Stage-3

In Stage-3, χ & ι steps of SHA-3 are implemented together. The χ comprises of *XOR, AND* & *NOT* operations between three selected 64-bit lanes from Stage-2 while ι is *XORing* of 64-bit Round Constant (*RC*) with only the *A[0:0]* lane of state. The two logical function of χ (*NOT* & *AND*) are directly implemented by using a single DSP Slice, with appropriate settings of *OPMODE* and *ALUMODE* as defined in [9] while *XOR* function with third 64-bit lane and also with *RC*, is implemented by the next cascaded DSP Slice as shown in Fig. 5.

This stage consists of five set of equations. The first pair of DSPs (DSP29 & DSP30) produce the MSB 48-bit output of 1st equation i.e. $E1 = O\_1\hat{}(\sim O\_2\&O\_3)$. In which MSB 48-bit of *O_2* provided at the *C* port is first *NOT* and then *AND* with *O_3* provided at the *A:B* port of DSP29. The output of this DSP Slice is provided to the DSP30 through its internal dedicated path *Pcout* to *XORed* finally with the MSB 48-bit of *O_1* as shown in Fig. 5.

The remaining four set of equations are implemented in a similar manner while in 5th equation i.e. $E1 = O\_1\hat{}(\sim O\_2\&O\_3)\hat{}RC$; *XORing* with *RC* is incorporated in the same DSP Slice (DSP38) by providing it at the *A:B* port as shown

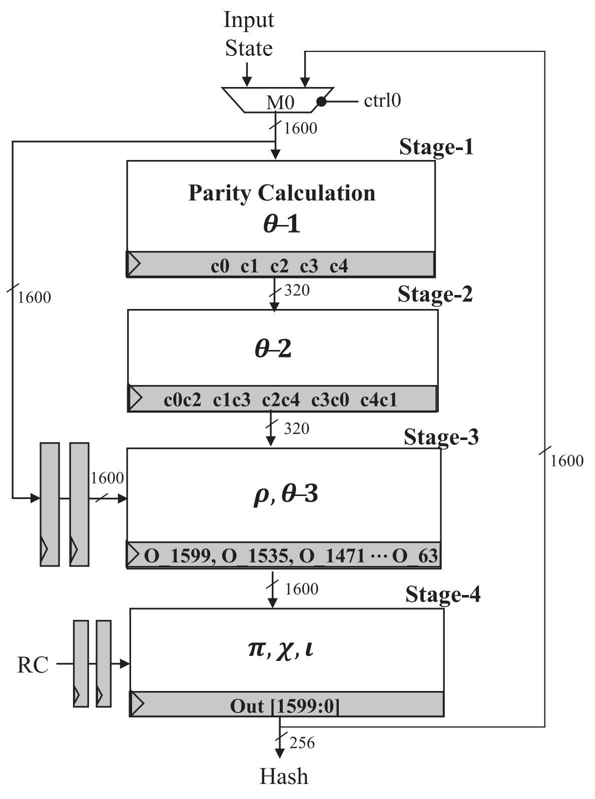

Fig. 7. Block diagram of 1600-bit SHA-3 design.

in Fig. 5, instead of using extra DSP Slice for this purpose. The complete implementation of these five equations therefore utilized 14 DSP slices from DSP29 to DSP42. The DSP slices from DSP29 to DSP38 produced MSB bits of the output while the remaining four DSP slices produced LSB bits of each output. Finally these five 64-bit outputs (*E1* to *E5*) are concatenated to form a single 320-bit output.

Thus in this way a total of 42 DSP slices are used by our compact SHA-3 core in order to process 320-bit in each clock cycle. This compact SHA-3 core is now replicated five times for implementation of full 1600-bit SHA-3 design with some modifications.

5.2. DSP based 1600-bit SHA-3 design

The design of our DSP based 1600-bit SHA-3 is shown in Fig. 7. In this architecture the permutation function of SHA-3 are logically optimized into four stages instead of three stages in order to suitably map this onto further reduced number of DSP slices. Stage-1 is the Parity Calculation (Eq. (1) of θ step), Stage-2 is the implementation of Eq. (2) of θ step, Stage-3 is the integration of ρ & Eq. (3) of θ while Stage-4 is the combined implementation of π, χ & ι.

Initial 1600-bit input state is directly provided to the Stage-1 & Stage-2 of the design through multiplexer *M0*. In Stage-1 parity sum is calculated for each column of state in parallel and are stored in the five 64-bit output registers of DSP Slice. These parity sums (*c0* to *c4*) are then provided to the Stage-2 where they are again *XORed* together to produce five possible sets of output i.e. *c0c2, c1c3, c2c4, c3c0* & *c4c1* as shown in Fig. 7. In Stage-3, the 25 64-bit input lanes are provided according to the ρ step and are *XORed* to the output of Stage-2 with respect to Eq. (3) of θ step. The resulted 25 64-bit lanes are then delivered to the next stage i.e. Stage-4 in a sequence according to the π step of SHA-3 which eliminates the need of re-arrangement circuitry. These 25 lanes in Stage-4 are finally *XOR, NOT* & *AND* together according to the χ step. Moreover *XORing* with *RC* is also integrated within the same stage to save extra DSP slices.

The DSP based 1600-bit SHA-3 architecture is fully pipelined and consists of four pipelined stages as indicated by the grey color. These pipelined registers allow us to process 4 blocks of input data in a pipelined fashion without utilizing any logical resources and as a result improve both the design's frequency and the throughput. These registers are fully realized using internal Input and Output registers of the DSP Slice at appropriate input and output ports of the DSP Slice. Input port *A:B* of DSP48 Slice provides two stage pipelined registers while input port *C* and output ports *Pout*/*Pcout* have one pipelined register.

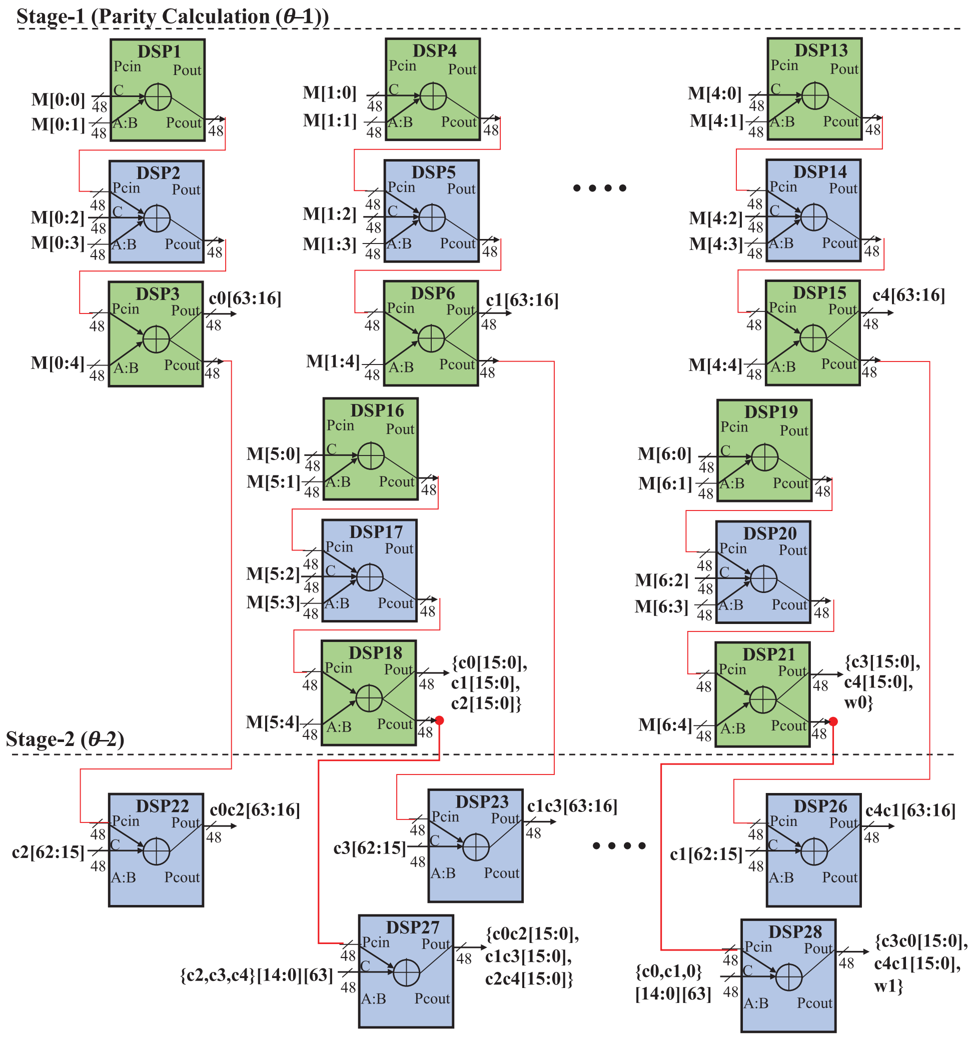

Fig. 8. Internal architecture of stage-1 & stage-2 of 1600-bit SHA-3 design.

5.2.1. Stage-1 & Stage-2

Detailed internal architecture of the Stage-1 (Parity Calculation (θ-1)) and Stage-2 (θ-2) are shown in Fig. 8. The implementation of Stage-1 is similar to the Stage-1 of compact SHA-3 design and has already been discussed in detail in Section 5.1.1. DSP1 to DSP15, produce the MSB 48-bit each of *c0, c1, c2, c3* & *c4* while for calculation of remaining LSB 16-bit, DSP16 to DSP21 are used. Therefore total of 21 DSP slices are utilized for the implementation of Stage-1.

In Stage-2, we have only implemented Eq. (2) of θ step, which is the addition of above 64-bit column parities (*c0, c1, c2, c3* & *c4*) in five possible groups that are *c0c2, c1c3, c2c4, c3c0* & *c4c1*. In all of these sets, the first column parity is provided at the *Pcin* port of each DSP Slice through internal cascaded path *Pcout* port while the second column parity (i.e. circular shifted by 1) is provided at the *C* port of each DSP Slice as taken from *Pout* port of required Stage-1 DSP Slice as shown in Fig. 8. For circular shifting no hardware resources are used and the input bits are provided in a required pattern. The implementation of this whole stage utilizes only 7 DSP slices, out of which 5 DSP slices from DSP22 to DSP26 produce the MSB 48-bit of each *c0c2, c1c3, c2c4, c3c0* & *c4c1* while DSP27 & DSP28 produce the remaining LSB 16-bit of each.

In compact DSP based SHA-3 design, the Eqs. (2) and (3) of θ step were combined together in single step that saved seven DSP slices. But here in this 1600-bit design, we have not combined these two equations into a single step because if we combine these two equations it will utilize 68 DSP slices otherwise if implemented separately will utilize only 41 DSP slices i.e. 7 DSP slices for θ-2 and 34 DSP slices for θ-3.

5.2.2. Stage-3 & Stage-4

The detailed internal architecture of both Stage-3 (ρ & θ-3) & Stage-4 (π, χ & ι) are provided in Fig. 9. The Stage-3 is the combined implementation of Eq. (4) of ρ step & Eq. (3) of θ step, in which both steps are logically optimized into a

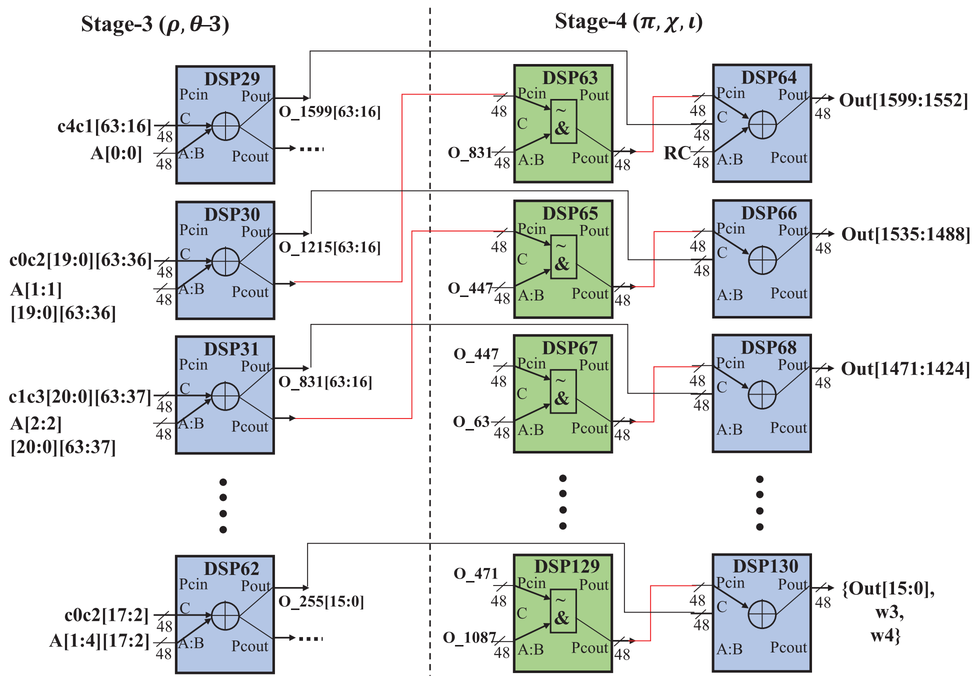

Fig. 9. Internal architecture of Stage-3 & Stage-4 of 1600-bit SHA-3 design.

single equation i.e. Eq. (9).

$$B[x, y, z] = ROT((A[x, y, z] \oplus C[x-1]C[x+1]), r[x, y]); \tag{9}$$

ρ step is circular shift of each 64-bit lanes by a different rotational values (*r[x,y]*) provided in [2] and is indicated by *ROT*. For the simplicity of hardware, we reschedule this step before θ-3 step and provided the inputs to this stage (i.e. 25 lanes and 5 parity sums) in a pre-calculated bit pattern with help of Eq. (9). This allows us to minimize both the routing delays and DSP slices. Because if we implement the circular shift functionality in hardware, it will utilize two DSP slices in order to implement a 18-bit cyclic barrel shifter. This barrel shifter is then required for each of 24 lanes, that will utilize total of 48 DSP slices but these 48 DSP slices are saved by providing pre-calculated bit pattern.

The complete implementation of Stage-3 is shown in Fig. 9. It utilized total of 34 (25+9) DSP slices ranging from DSP29 to DSP62. The first 25 DSP slices produce the MSB 48-bit of each 25 outputs i.e. *O_1599* for *A[0:0]* lane, *O_1215* for *A[1,1]* lane, *O_831* for *A[2,2]* lane and so on. The remaining 9 DSP slices produce the LSB 16-bit each of them. The inputs to DSP29 are provided as it is without circular rotation, the inputs to DSP30 are provided in pre-calculated rotated pattern of 44 while inputs to DSP31 in pre-calculated rotated pattern of 43 and so on.

The Stage-4 is integration of three steps (π, χ & ι) into another single equation i.e. Eq. (10):

$$Out[x, y, z] = A[x+3y, x, z] \oplus (NOT(A[(x+1)+3y, (x+1), z])AND(A[(x+2)+3y, (x+2), z])) \tag{10}$$

The output of Stage-3 is now provided to the Stage-4 in a sequence of π step according to the Eq. (10). With the help of this equation, the DSP slices of Stage-3 are arranged in a pre-defined order to get the output of Stage-4. For example to calculate the MSB 48-bit of *Out[0:0]* i.e. ranging from 1599 to 1552, we require *A[0,0], A[1,1]* & *A[2,2]* from the previous stage and *XORing* of *RC* with *A[0,0]*. For *Out[1,0]* which is the next MSB 48-bit ranging from 1535 to 1488, we need *A[1,1], A[2,2]* & *A[3,3]* and so on. This optimization eliminates the need of re-arrangement circuitry and also allows us to cascade the output of DSP slices through *Pcout* port which reduced critical delays between the two DSP slices. The implementation of Stage-4 results in the utilization of 68 DSP slices from DSP63 to DSP130.

Therefore, the complete 1600-bit SHA-3 design utilizes total of 130 DSP slices (21 + 7 + 34 + 68) in addition to the few logical Slices. The five set of equations along with Stage-1 are implemented in LCS design strategy having parallel pipelined hardware units. Every DSP Slice is configured to implement a 48-bit wide logical operations of 3 operands and internal I/O registers are used for pipelining. Furthermore the logic of SHA-3 is aligned in such a way to use the dedicated internal cascading connection between the two DSP slices.

6. Implementation results and comparisons

Two proposed DSP based SHA-3 designs were implemented and tested using Xilinx Vivado 2014.4 targeting DSP48 slices of Xilinx UltraScale FPGA. The implementation results of our proposed designs are shown in Table 1. For comparison with

Table 1
Implementation results of our DSP based SHA-3 designs.

Our design	Device	DSP primitive	Slices + DSP	Frequency (MHz)	Power (mW)
DSP based Compact SHA-3	Virtex-6	DSP48E1	208 + 58	451.26	130
	Kintex-7	DSP48E1	205 + 58	463.177	88
	Virtex-7	DSP48E1	297 + 58	463	147
	Virtex UltraScale	DSP48E2	142 + 42	680	87
DSP based 1600-bit SHA-3	Virtex UltraScale	DSP48E2	353 + 130	526	120

Table 2
Results comparison with FPGA based SHA-3 designs.

Design	Device	Slices, BRAM, DSP	Frequency (MHz)	Throughput* (Gbps)	Power (mW)
[22]	V5(Basic)	1352, 0, 0	298.6	13.536	–
	V5 (Embedded)	1338, 1, 0	248.2	11.252	–
[23]	V5 (Kec_Pip)	2326, 0, 0	306	6.794*	–
	V5 (Kec_DSP2)	3009, 0, 201	334	0.946*	–
[14]	Virtex-6	915, 0, 0	301.57	13.67	2026*
[16]	Virtex-6	1649, 0, 0	397	19.1	2026*
[17]	Virtex-6	1115, 0, 0	412	9.888	2026*
[18]	Kintex-7	1185, 0, 0	213.17	9.66	629
	Virtex-6	1048, 0, 0	194.78	8.83	2026
[19]	Kintex-7	1416, 0, 0	309.69	8.22	800
	Virtex-6	1181, 0, 0	251.7	6.68	1000
Our compact DSP design	**Virtex UltraScale**	**142, 0, 42**	**680**	**6.165**	**80**
	Kintex-7	**205, 0, 58**	**463.177**	**4.199**	**88**
	Virtex-6	**208, 0, 58**	**451.26**	**4.091**	**130**
Our 1600-bit DSP design	**Virtex UltraScale**	**353, 0, 130**	**526**	**23.84**	**120**

*Approximate calculation of throughput and power for SHA3-256.

the previously reported results, we have also provided the implementation results of only compact SHA-3 core on Virtex-6 and 7 Series FPGA as no results are reported using the UltraScale platforms.

Our compact SHA-3 core uses only 42 DSP48E2 slices on Virtex UltraScale with the minimal overhead of logical Slices, while 58 DSP48E1 slices on Virtex-6 and 7 Series FPGA are shown in Table 1. It gives equally good results on all the modern FPGA platforms with the added advantage of 27% further reduction of DSP slices on UltraScale FPGA. This is due to the fact that we have taken full advantage of *Pcin* port of DSP48E2 slices to process the third operand and sto perform triple 48-bit logical function without the need of any additional DSP Slice. On Virtex UltraScale, this compact SHA-3 design is operating at 680 MHz while on Virtex-6 and 7 Series it results in a clock speed of 451.26 MHz and 463 MHz, respectively. The total latency of our proposed compact design is 120 clock cycles for the processing of single message.

Furthermore in addition to this achieved frequency, our design has an added advantage of a low-power consumption by implementing SHA-3 logic in a LCS design strategy. The power consumption of our proposed compact SHA-3 core is calculated using Xilinx XPower Analysis tool [25] and it ranges from 87 mW on Virtex UltraScale to maximum of 147 mW on Virtex-7 FPGA.

Similarly 1600-bit SHA-3 core utilizes only 130 DSP slices along with 353 logical slices. The basic structure of this design is based on 5 times replica of our proposed compact SHA-3 core with few modifications that reduced down the hardware requirement of complete design from 5 $\times$ hardware resources to 3 $\times$ only, there by saving an additional 29% of DSP slices. These modification includes the deploying of four stage pipelined organization for 1600-bit SHA-3 instead of three stage (as used for compact SHA-3 design) and re-scheduling of SHA-3 transformations. Further this four stage pipelined architecture also enabled us to hash four messages at the same time. The frequency of this design is 526 MHz with a low-power consumption of 120 mW only.

The comparisons of our implementation results with previously reported FPGA based SHA-3 cores are given in Table 2. It is evident from the table that we have not only saved the logical Slices but also utilized optimal number of DSP slices as compared to all the conventional LUT based as well as embedded resources (BRAM or DSP) based SHA-3 designs.

In [22] and [23], they have used large number of logical resources in addition to the dedicated resources of FPGA. Their saving of logical resources at the cost of these dedicated resources are not very significant. In [22], Sharif et al. have provided both basic and embedded implementation of SHA-3 core and in their embedded SHA-3 design they saved only 1% of Slices by using one BRAM from their basic SHA-3 design. Furthermore there is only one fully DSP based SHA-3 design (*Kec_DSP2*) available in open literature in which Provelengios et al. utilized 201 DSP slices along with an additional 3009 logical Slices

[23]. They inefficiently utilized each DSP Slice without saving any logical Slices and perform 32-bit logical operation by using one DSP48E Slice, although it is capable of performing 48-bit wide operation. In comparison to Provelengios et al. SHA-3 design [23], our compact SHA-3 core on both Virtex-6 & Kintex-7 not only saved 71.14% of DSP slices but also saved large number of logical resources while this saving of DSP slices on Virtex UltraScale is 79.10% there by utilizing only 42 DSP slices. Similarly our high speed SHA-3 design in comparison to the same design have saved 35.32% of DSP slices in addition to the saving of 100% logical Slices.

Likewise our SHA-3 cores also consume low-power as compare to the LUT based SHA-3 designs [18,19]. Our compact SHA-3 design on Kintex-7 consumed 88 mW power which is $1/7^{th}$ and $1/9^{th}$ of [18] and [19] designs, respectively. Also on Virtex UltraScale, the power consumption of this core is only 80 mW. The use of dedicated interconnection between two DSP slices i.e *Pcout* to *Pcin*, not only reduced the routing delays but also minimized the power consumption irrespective of FPGA device.

The power consumption of our 1600-bit design is also $1/5^{th}$ of the lowest reported power i.e. 629 mW. In [18], Rao et al. report 2026 mW at 194.78 MHz with 1048 Slices. It is a known fact that power consumption increases with the increase in frequency, keeping in view this we estimated the power consumption of all LUT based SHA-3 designs whose power are not reported in literature. Therefore approximate power consumption of [14], [16], and [17] at 301.57 MHz, 397 MHz, and 412 MHz will be greater than 2026 mW. This clearly indicates that our design also perform well in terms of power consumption.

Our optimal design strategy not only provided a good balance in terms of overall FPGA utilization with the usage of these embedded resources but also improved the hardware performance. These improved performance results are because of reduced routing by proper tiling and cascading of the DSP slices in a LCS design strategy and utilization of internal pipelined registers. Therefore our high-speed SHA-3 design with 1600-bit data path achieved the maximum throughput of 23.84 Gbps as compared to all the high-speed designs [14,16,17,22] and saved large number of logical resources, entailed for implementation of other functions on the same device.

Moreover in this work, we have provided the implementation of our proposed designs on a wide variety of latest Xilinx FPGA platforms such as Virtex-6, Kintex-7, Virtex-7 and Virtex UltraScale. The proposed designs can also be implemented on Virtex-5, Atrix-7 and Kintex UltraScale and can be ported to other vendors FPGA such as Altera, Achronix, Microsimi devices subject to the availability of used features of DSP slice.

7. Conclusion

Our proposed methodology in this work provides a balanced solution for the hybrid systems that have both logical and arithmetic intensive computations and at the same time require efficiency and power saving features. Thus our two presented efficient DSP based SHA-3 designs not only result in the utilization of optimal number of DSP48E slices on Xilinx FPGA with very few logical Slices but also result in a low-power consumption. These DSP48E slices are now available in almost all the series of Xilinx FPGAs and at the same time their number are also increasing with development of new series of FPGA devices such as UltraScale & UltraScale+. Our proposed SHA-3 architectures in this work will help in utilizing the capability of these enhanced DSP48E slices of Xilinx FPGA for logic intensive computations and also save the logical resources (LUT & Slices) for the implementation of other system functionality especially in a resource constrained environment.

References

[1] National Institute of Standards and Technology (NIST). NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition. 2012. http://www.nist.gov/itl/csd/sha-100212.cfm.
[2] FIPS PUB 202. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Tech. Rep. National Institute of Standards & Technology (NIST); 2015. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
[3] Liang J, Lai X-J. Improved collision attack on hash function MD5. J Comput Sci Technol 2007;22(1):79–87. doi:10.1007/s11390-007-9010-1.
[4] Stevens M. New collision attacks on SHA-1 based on optimal joint local-collision analysis. In: Advances in cryptology EUROCRYPT 2013. vol. 7881 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2013. p. 245–61. ISBN 978-3-642-38347-2. doi:10.1007/978-3-642-38348-9_15.
[5] Xilinx Inc. Virtex-5 FPGA Xtreme DSP design considerations v3.5, User Guide; 2012. http://www.xilinx.com/support/documentation/user_guides/ug193.pdf.
[6] Xilinx Inc. 7 series FPGAs overview v1.15. Tech. Rep.; 2014. http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
[7] Xilinx Inc. UltraScale architecture and product overview v2.4. Tech. Rep.; 2015. http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf
[8] Xilinx Inc. 7 series DSP48E1 Slice v1.8, User Guide; 2014. http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.
[9] Xilinx Inc. UltraScale architecture DSP Slice v1.2, User Guide; 2015. http://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.
[10] Guo X, Srivastav M, Huang S, Ganta D, Henry MB, Nazhandali L, et al. Silicon implementation of SHA-3 finalists: BLAKE, Grostl, JH, Keccak and Skein ECRYPT II Hash Workshop 2011. Tallinn, Estonia; 2011.
[11] Kavun E, Yalcin T. A lightweight implementation of Keccak hash function for radio-frequency identification applications. In: Radio frequency identification: security and privacy issues vol. 6370 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2010. p. 258–69. ISBN 978-3-642-16821-5. doi:10.1007/978-3-642-16822-2_20.
[12] Srivastav M, Guo X, Huang S, Ganta D, Henry MB, Nazhandali L, et al. Design and benchmarking of an ASIC with five SHA-3 finalist candidates. Microproc Microsyst 2013;37(2):246–57. doi:10.1016/j.micpro.2012.09.001.
[13] Bayat-Sarmadi S, Mozaffari-Kermani M, Reyhani-Masoleh A. Efficient and concurrent reliable realization of the secure cryptographic SHA-3 algorithm. IEEE Trans Comput-Aided Des Integr Circuits Syst. 2014;33(7):1105–9. doi:10.1109/TCAD.2014.2307002.
[14] Latif K, Rao MM, Mahboob A, Aziz A. Novel arithmetic architecture for high performance implementation of SHA-3 finalist Keccak on FPGA platforms. In: Proceedings of the 8th International Conference on Reconfigurable Computing: Architectures, Tools and Applications. ARC'12. Berlin, Heidelberg: Springer-Verlag; 2012. p. 372–8. ISBN 978-3-642-28364-2. doi:10.1007/978-3-642-28365-9_34.

[15] Akin A, Aysu A, Ulusel OC, Sava E. Efficient hardware implementations of high throughput SHA-3 candidates Keccak, Luffa and Blue Midnight Wish for single- and multi-message hashing. In: NIST 2nd SHA-3 Candidate Conference. Santa Barbara; 2010.
[16] Athanasiou G, Makkas G-P, Theodoridis G. High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm. In: 6th International Symposium on Communications, Control and Signal Processing (ISCCSP-2014). Athens, Greece; 2014. p. 538–41. doi:10.1109/ISCCSP.2014.6877931.
[17] Michail HE, Ioannou L, Voyiatzis AG. Pipelined SHA-3 implementations on FPGA: architecture and performance analysis. In: Proceedings of the Second Workshop on Cryptography and Security in Computing Systems. New York, NY, USA: ACM; 2015 p. 13:13–13:18. ISBN 978-1-4503-3187-6.
[18] Rao M, Newe T, Grout I. Secure hash algorithm-3 (SHA-3) implementation on Xilinx FPGAs, suitable for IoT applications. In: Proceedings of 8th International Conference on Sensing Technology. Liverpool, UK; 2014.
[19] Honda T, Guntur H, Satoh A. FPGA implementation of new standard hash function Keccak. In: 2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE); 2014. p. 275–9. doi:10.1109/GCCE.2014.7031105.
[20] Kaps J-P, Yalla P, Surapathi K, Habib B, Vadlamudi S, Gurung S, et al. Lightweight Implementations of SHA-3 Candidates on FPGAs. In: Progress in Cryptology INDOCRYPT 2011, Vol. 7107 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2011. p. 270–89. ISBN 978-3-642-25577-9. doi:10.1007/978-3-642-25578-6_20.
[21] San I, At N. Compact Keccak hardware architecture for data integrity and authentication on FPGA. Inf Sec J: A Global Perspect 2012;21(5):231–42. doi:10.1080/19393555.2012.660678.
[22] Sharif MU, Shahid R, Rogawski M, Gaj K. Use of Embedded FPGA resources in implementations of five round three SHA-3 candidates. ECRYPT II Hash Workshop 2011. Tallinn, Estonia; 2011.
[23] Provelengios G, Kitsos P, Sklavos N, Koulamas C. FPGA-based design approaches of Keccak hash function. In: 15th Euromicro Conference on Digital System Design (DSD-2012); 2012. p. 648–53. doi:10.1109/DSD.2012.63.
[24] Ayuzawa Y, Fujieda N, Ichikawa S. Design trade-offs in SHA-3 multi-message hashing on FPGAs. In: TENCON 2014 - 2014 IEEE Region 10 Conference; 2014. p. 1–5. doi:10.1109/TENCON.2014.7022311.
[25] Xilinx Inc. Xilinx Power Estimator v2014.2. User Guide; 2014. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug440-xilinx-power-estimator.pdf

Dur-e-Shahwar Kundi is currently a Ph.D. candidate in Electrical Engineering at National University of Sciences and Technology (NUST), Karachi, Pakistan. She received her B.E. and M.S. degree in Electrical Engineering in the year 2007 and 2009, respectively from NUST, Pakistan. Her research interests include data security, cryptographic engineering and reconfigurable computing.

Arshad Aziz is an Associate Professor in Electrical Engineering department at National University of Sciences and Technology, Karachi, Pakistan. He received his Ph.D. degree in Electrical Engineering from NUST, Pakistan in 2007. His research interests include Network Security, Cryptography and FPGA based System Design.