A High Speed KECCAK Coprocessor for Partitioned NSP Architecture on FPGA Platform

TRS No. : XXXXX

Abstract:
The messages in the latest security protocols such as IPSec, TLS and SSL must be handled by high-speed crypto systems. Current computationally extensive cryptographic implementations on different platforms such as software, Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) without adequate optimization achieve lesser throughput than should be possible. In the paper we consider a cryptographic hashing algorithm KECCAK and its implementations. To achieve better throughput, the proposed implementations of KECCAK explores FPGA design spaces. In this paper three different architectures for KECCAK coprocessor are implemented in Artix-7 (XC7A100T, CSG324) FPGA platform. The Processing Element (PE) handles all communication interfaces, data paths and control signals hazards of Network Security Processor (NSP). A partitioned area in the system ensures that the processor data path is completely isolated from secret key memory. The memory to KECCAK core communication is done by Direct Memory Access Controller (DMA). The performances of the implemented KECCAK are better in terms of throughput and resource usage than the existing work reported in the literature.

Rourab Paul, Sandeep K Shukla
Dept Name : Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur-208016, India

1 Introduction

Hash functions are used for data integrity and authentication purposes. As there are various vulnerability in widely-used MD-5, SHA-1 and SHA-2 hash algorithms, the US NIST had initiated the SHA-3 contest in order to select a suitable replacement. In 2009, KECCAK was selected as the winner of the contest and in august 2015, NIST announced that KECCAK became a new hashing standard. In the coming few years, it is expected that KECCAK will become mandatory cryptographic hash algorithm for all mainstream and future network security protocols along with existing standards such as TLS, SSL, SET, IPSec, and PKI. The tremendous growth of high-speed communication technology demands high throughput crypto hardwares. Hence, many initiatives are taken to optimize the throughput of KECCAK crypto core.

Article [1] proposed a KECCAK hardware in Virtex-5 FPGA device which includes the usage of DSP blocks to minimize the consumption of traditional user logic such as look-up tables (LUTs). Authors claimed that incorporating Xilinx DSP48E block in their pipelined design increased the timing performance of the design but the result shows that they achieved only 5.71 Gbps throughput. The DSP blocks have many functions such as SIMD based multiply-accumulator, multiply-adder, and a one- or n-step counter etc, which makes the DSP48E more expensive than normal LUT based Slices. Hence, the achievement of few Gbps throughput with expensive DSP configured AND, XOR, MOD, NOT operation in KECCAK algorithm cannot be a good solution.
The Virtex-5 and Virtex-6 FPGA based KECCAK hardware in article [2] consumed 88 clock cycle for one KECCAK round. The total 24 rounds required 88×24=2112 clock cycles. The 5 sequential functions of KECCAK rounds such as θ, ρ, π, χ and ι are very primitive in nature, nevertheless this implementation folded each function and split their logic load into several clock cycles which increases total latency unnecessarily.
Articles [3] and [4] are the 1 clock 1 round folded KECCAK architectures where θ, ρ, π, χ and ι functions are placed into a single clock cycle. Article [4] incorporated a 3 stages pipeline concept based on multi-message hashing hardware where new input arrives at every clock cycle. In the 1^{st} pipeline stage θ is computed. The ρ and π step are implemented in the 2^{nd} stage. The final stage combines χ and ι steps. At each clock cycle, the result of the cyclic shift operation is stored in a register.
Articles [5] and [6] stated KECCAK might be an alternative of AES for some specific applications. Article [5] implemented KECCAK in Virtex-4 FPGA for securing Precision Time Protocol (PTP) standardized by IEC 61850 family for substation communication network applications. They unroll KECCAK by different unrolling factor and attempted to show without compromising with security and latency, KECCAK may become a substitute of AES-128 for HMAC type applications. Article [6] implemented different cipher mode KECCAK in Virtex-7 FPGA Board and showed it is more efficient in terms of resource throughput than AES.
Article [7] explored the KECCAK implementation in different FPGAs of Xilinx and Altera. They achieved 8.023 Gbps maximum throughput for KECCAK-256 in Kintex 7 Xilinx board. They claimed loop unrolled technique cannot increase KECCAK throughput, but we find that the loop unrolled factor by two can double the throughput of normal folded KECCAK architecture.
Article [8] implemented hardware/software co-design based KECCAK for HMAC and PRNG security functions in ZYNQ SoC FPGA platform where, the capacity parameter of KECCAK is configurable.
Articles [9] and [10] implemented all candidates of SHA-3 finalist in Xilinx and Altera FPGA platform and tried to find out the most efficient SHA-3 in terms of slices and throughput. [9] claimed only three candidates, KECCAK, Luffa, and CubeHash, have the better throughput to area ratio than the current standard SHA-256. Of the three algorithms, Keccak and Luffa have achieved very high throughputs, while CubeHash outperformed other candidates in terms of minimal resource usage. Article [10] achieved only 0.843 Gbps throughput which implies very less throughput/slice ratio of their architecture.
Article [11] is a very light weight FPGA implementation using only Shift Register and Lookup Tables. The implementation consumed very fewer silicon spaces but compromises with very less throughput.
Article [12] implemented fastest KECCAK in FPGA. They changed the number of pipeline stages and studied various circuit issues of FPGA hardware. The presented implementations in three Virtex FPGA families performed better with respect to throughput by 250% than previous publications.
The contribution of the paper is stated below

- We implemented 3 architectures of KECCAK in FPGA platform such as (i) Folded (FL), (ii) Loop Unrolled Folded (LU2) and (iii) Pipelined Loop unrolled Folded (PPL2-LU2)carchitecture. To the best of our knowledge, the PPL2-LU2 achieved the highest throughput in existing lit-

erature.

- We proposes three partitions in NSP architecture such as a processor area, a crypto area and a confidential area. The KECCAK is implemented as a coprocessor placed in the crypto area and the master secret keys are stored in a different confidential portion of the FPGA to prevent all software related attacks from processor area.

- We propose a complete hardware flow of security protocols where crypto applications are implemented as coprocessor. The parallel execution of master Processing Element (PE) and coprocessor along with DMA make the system more efficient in NSP environment.

The organization of the paper is as follows. In Section 2, a details discussion about three KECCAK architecture is presented concerning the parallelism issues of the design. The system architecture and system flow are stated in Sections 3 and 4, respectively. Implementation of our design and analysis of its results are briefed in Section 5 . The paper is concluded in Section 6.

2 KECCAK Architecture

The implemented KECCAK[r, c, d] has bit rate $r = 1088$, capacity c=512 and data word size d=64 [13] which iterates 24 rounds to produce 256 bits digest. For each round one *Round Count* RC_i is generated to address a *Round Constant* $RCon_i$. For each round five consecutive functions such as θ, ρ, π, χ and ι modify a value RI_i which comes from 1600 bit state register SR_i. The RC_i starts from 0 and ends in 23. It is to be noted that the detail algorithmic description of KECCAK is avoided intentionally due to hard page restriction. In our proposal KECCAK architecture is implemented in 3 different approaches, (i) Folded (FL), (ii) Loop Unrolled Folded (LU2) and (iii) Pipelined Loop unrolled Folded (PPL2-LU2)architecture. The FL, LU2 and PPL2-LU2 iterate 24 rounds in 24, 12 and 6 clocks respectively. The hardware architectures of KECCAK is described in next two sections.

2.1 Hardware Components

The KECCAK mainly has three hardware component named as *Round Constant ROM*, *Round Map Block* and *Count Generator*.

2.1.1 Count Generator

The *Count Generator* generates round counts RC_i from 0 to 23 for Round Constant ROM block. In the folded architecture the RC_i of *Count Generator* is incremented by one in rising edge of each clock. 24 clocks are required for 24 iterations. In the loop unrolled architecture two round counts RC_i and RC_{i+1} count from 0 to 23 for 24 KECCAK rounds in 12 clocks cycles. In the Pipe-lined Loop unrolled architecture two LU2 *Count Generator*s is used to generate 4 RC_is. One *Count Generator* counts 0 to 11, while other one counts 12 to 23 using 6 clocks cucles. Hence, parallel counts of RC_i and RC_{i+1} in 2 LU2 *Count Generator* complete 24 iterations in 6 clocks.

2.1.2 Round Constant ROM

The round *Round Constant ROM* intakes *round value* RC_i and produces a 64 bit *Round Constant* $RCon_i$. The mapping between RC_i and $RCon_i$ is based on computational look up table as stated in KECCAK data sheet [13]. One RC_i of the folded architecture can address only one $RCon_i$ in one clock cycle, whereas two *round counts* RC_i and RC_{i+1} of the Loop Unrolled architecture can address two *Round constants* $RCon_i$ and $RCon_{i+1}$ in a single clock. In the Pipe lined architecture, four *Round constants* $RCon_i$, $RCon_{i+1}$, $RCon_{i+12}$ and $RCon_{i+13}$ can be addressed by four *round counts* RC_i, RC_{i+1}, RC_{i+12} and RC_{i+13} in a single clock.

2.1.3 Round Map Block

The *Round Map Block* executes 5 sequential functions θ, ρ, π, χ and ι. The 32 bit width DMA master port S_AXIS_TDATA sends 34 *data word* into 1600 bit width *State Register* SR_i. In each round the SR_i is updated by 5 consecutive operations θ, ρ, π, χ and ι. After 24 rounds the 256 bit hash value is derived from SR_{i+23}. In the folded architecture only one RO_i can be calculated from Ri_i in single clock whereas for the LU2 architecture two *Round Outs* RO_i and RO_{i+1} can be generated from initial *Round In* RI_{i-1}. In the Pipelined architecture two *Round Map Block*s are used to generate four *Round Outs* RO_i, RO_{i+1}, RO_{i+12} and RO_{i+13}.

2.2 Architectures

In these proposal 3 hardware architectures are proposed for KECCAK hash algorithm.

2.2.1 Folded Architecture

The Folded KECCAK is a straight forward design where logic load of one round is placed in a single clock of *Round Map Block*. At a rising edge of clock *Count Generator* block generates a new RC_i to address a $RCon_i$ from *Round Constant ROM*. The updated $RCon_i$ and a RI_i register derived from state register SR_i generates ι_i followed by four consecutive processes θ, ρ, π, χ. The output port of *Round Map Block* RO_i is shorted with ι_i which updates state register from SR_i to SR_{i+1}. The equ. 1 derives RO_i from RI_i.

$$RO_i = Round\ Map\ Block[RI_i, RCon_i] \tag{1}$$

The internal functions of *Round Map Block* are θ, ρ, π, χ and ι which are calculated by equ. 2 and 3.

$$\iota in_i = \iota_i(\pi_i(\rho_i(\theta_i(RI_i)))) \tag{2}$$

$$RO_i = \iota_i(\iota in_i, RCon_i) \tag{3}$$

Folded architecture requires 24 clock cycles to iterate 24 rounds. The data words in *Round State Register* SR_i is collected from DMA slave port S_AXIS_TDATA. The hardware architecture of folded design is shown in Fig. 1.

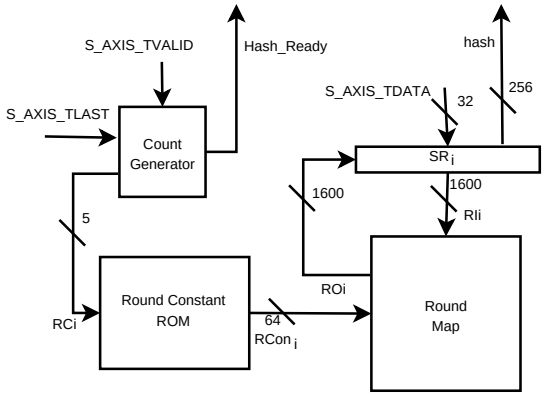

Figure 1: Folded Architecture of KECCAK

2.2.2 Loop Unrolled Architecture (LU2)

In the Loop Unrolled architecture two loops are unrolled to calculate SR_{i+1} directly from SR_{i-1}. In the previous architecture, one RC_i was needed to address one $RCon_i$, whereas in Loop Unrolled approach two RC_is are required to address a pair of $RCon_i$s. These two RC_is are incremented by two in each clock. In 1^{st} clock RC_i and RC_{i+1} are initialized by 0 and 1 respectively. In 2^{nd} clock RC_i becomes 2 and RC_{i+1} becomes 3, and so on at a 12^{th} clock they both reach to 22 and 23. Hence, for the i^{th} loop RC_{i+1} can be derived from RC_{i-1} using equ. 4 and 5.

$$RC_i = RC_{i-1} + 1 \tag{4}$$

$$RC_{i+1} = RC_{i-1} + 2 \tag{5}$$

As shown in Fig. 2, the $RCon_i$ and $RCon_{i+1}$ are addressed by RC_i and RC_{i+1} respectively are used in *Round Map Block* to compute RO_{i+1}directly from RI_i which is stated in equ. 6.

$$RO_{i+1} = Round\ Map\ Block[RI_i, RCon_i, RCon_{i+1}] \tag{6}$$

Inside *Round Map Block* ρ_i, χ_i and RO_i are calculated using equs.2 and 3. The RO_{i+1} is generated using equs. 7 and 8

$$\iota in_{i+1} = \iota_{i+1}(\pi_{i+1}(\rho_{i+1}(\theta_i(RI_{i+1})))) \tag{7}$$

$$RO_{i+1} = \iota_{i+1}(\chi_{i+1}, RCon_{i+1}) \tag{8}$$

Here it may to be noted that in Loop unrolled architecture we double the logic load of each clock to reduce the required number of clock from 24 to 12, which increases the critical path of the architecture.

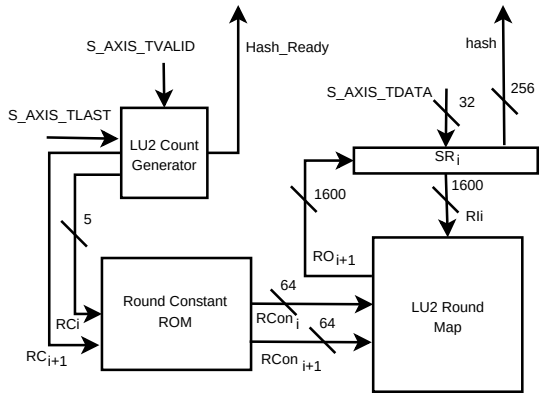

Figure 2: LU2 Architecture of KECCAK

2.2.3 Pipelined Loop Unrolled Architecture (PPL2-LU2)

The LU2 architecture stated in the previous section can be adopted in pipeline design. In this PPL2-LU2 architecture 2 pipeline stages are used to compute 24 KECCAK rounds. In the 1^{st} pipeline stage one LU2 *Round Map Block* executes 0^{th} round to 11^{th} round in 6 clock cycles and in 2^{nd} pipeline stages another LU2 *Round Map Block* executes 12^{th} round to 23^{rd} round in the next 6 clock cycles.

1^{st} **Pipeline Stage:** At the 1^{st} clock of the 1^{st} Pipeline stage RC_i and RC_{i+1} are initialized

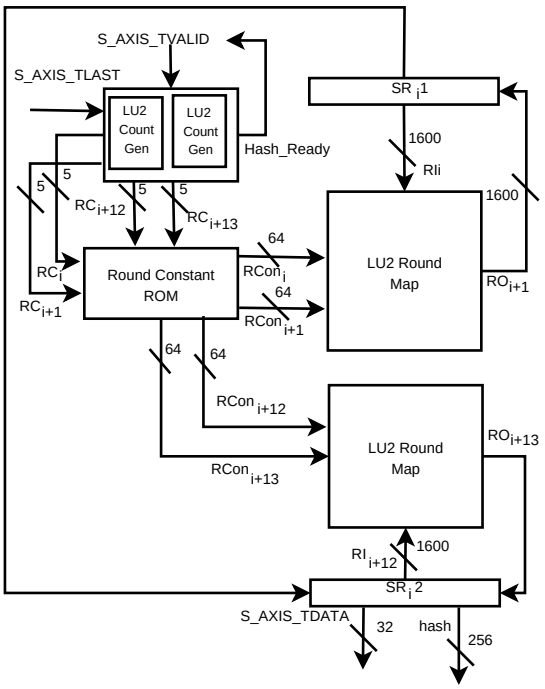

Figure 3: PPL2-LU2 Architecture of KECCAK

by 0 and 1 respectively. After passing the 1^{st} 6 clocks in *Count Generator* RC_i and RC_{i+1} reach to 10 and 11, respectively and LU2 *Round Map Block* -1 transforms the state of register $SR-1$ from SR_{i-1} to SR_{i+11}. Then $SR-1$ sends his state SR_{i+11} to $SR-2$.

2^{nd} **Pipeline Stage:** For the next 6 clocks 2 operations are executed parallely. In the 7^{th} clock of 1^{st} operation *Count Generator* re-initializes RC_i and RC_{i+1} to 0 and 1 respectively and starts same counting sequence as stated above. The LU2 *Round Map Block* -1 grabs a new data word in $SR-1$ and execute pipeline stage-1.

At the 7^{th} clock of 2^{nd} operation RC_{i+12} and RC_{i+13} start their count from 12 and 13 respectively. By increasing 1 in both RC_{i+12} and RC_{i+13} counts in each clock cycle they reach to 22 and 23 at 12^{th} clock. The state register $SR-2$ in LU2 *Round Map Block* -2 simultaneously changes state from SR_{i+11} to SR_{i+23} and produces a 256 bit has value at the 23^{rd} clock. The PPL2-LU2 design allows a first block at output after 12 clock cycles and it can process a new input block in every six clock cycles. The pictorial presentation of (PPL2-LU2 is shown in Fig. 3.

Table 1: Comparison Table

Name	throughput Gbps	platform	# clock	clock MHz	# Slices
Proposed Folded	16.492	Artix 7	24	390.53	4188
Proposed LU2	19.99	Artix 7	12	234.97	7139
Proposed PPL2LU2	39.98	Artix 7	6	234.97	8459
[1]	5.70	Virtex 5	25	258	2573
DSP V1 [1]	1.16	Virtex 5	25	58	3176
DSP V2 [1]	0.43	Virtex 5	384	334	3009
Pipelined [1]	3.12	Virtex 5	49	306	2326
Sync [1]	5.56	Virtex 5	25	278	2517
[13]	0.07	Virtex 5	5160	265	444
[2]	0.08	Virtex 6	2112	285	188
[3]	10.25	Virtex 6	24	231	1043
[4]	6.07	Virtex 4	25	143	2024
[5]	10.54	Virtex 5	20	242.77	482
[7] 512	2.473	Spartan 3	33	95.45	3362
[7] 512	4.795	Virtex 4	33	185.05	3622
[7] 512	3.550	Virtex 5	33	137.01	1647
[7] 512	6.522	Virtex 6	33	251.70	1181
[7] 512	8.023	Kintex 7	33	309.69	1416
[6]	9.370	Virtex 7	-	206.70	2495
[6]	0.125	Virtex 7	-	152.23	264
[8] DCC	1.636	Zed Board	- -	110.74	2891
[8] DCCL	0.25	Zed Board	-	104	2923
[3]	13.67	Virtex 6	24	301.57	915
[3]	12.49	Virtex 5	24	275.56	133
[9]	11.84	Virtex 6	-	-	1165
[9]	12.77	Virtex 5	-	-	1395
[10]	0.843	Virtex 6	-	159	393
[11]	0.152	Virtex 5	-	248	134
[12]	27.07	Virtex 4	6	282	12870
[12]	34.272	Virtex 5	6	357	4632
[12]	37.632	Virtex 6	6	392	4117

3 System Architecture

The processors offer flexibility, but it weakens the implementation level security of crypto algorithms. In order to avoid side-channel attacks [14], the secret keys must be manipulated regularly using a key management protocol. When a processor accesses and manipulates confidential secret keys, the keys comes in processor registers or in the cache memory, which exposed confidential data for software attacks. We propose an NSP architecture which is partitioned [15] in 3 different areas such as a processor area, a crypto area and a confidential area as shown in Fig. 4. The processor area consists PE, local memory, bus, bus interconnect, DMA, bus streamer between DMA and coprocessor, Data Path Controller (DPC) to set data path connections of coprocessors and other memories. The crypto area consists a Random Number Generator (RNG), Encryption Block, SHA-3 Block and a shared memory shared between PE and coprocessor to store data (plain text, cipher text and hash value), public key and pre-master key. The confidential memory only consists the master key memory.

4 System Flow

Let us assume that for an instant, we have a combination of RSA Key Exchange, AES encryption and SHA-3 hash algorithm. A typical security protocol uses several steps for key exchange, encryption

and hashing processes.

1. Alice generates a random number named as *client hello* (32 bytes).
2. Alice sends *client hello* to Bob.
3. Bob sends his certificate, public Key and a random number named *server hello* (32 bytes).
4. Alice generates another random number from *RNG* block placed in the crypto area.
5. Alice encrypts the random number using Bob's public key using RSA algorithm mounted in encryption block of the crypto area. The encrypted random number named *pre master key* is sent to Bob.
6. Bob receives *pre master key*. The KECCAK HMAC algorithm in the crypto area generates a master key (48 bytes) using *pre master key*, *client hello* and *server hello*. Four master keys are generated in this process, such as $client\_write\_MAC\_key$, $server\_write\_MAC\_key$, $client\_write\_key$ and $server\_write\_key$. Here the MAC keys are for the authentication and integrity, write keys are for the symmetric encryption. Two more keys $client\_write\_IV$ and $server\_write\_IV$ are only generated for implicit *nonce* in Authenticated Encryption with Associated Data (AEAD) cipher.

The four master keys are stored in the four consecutive addresses of *Master Key Memory* (MKM). The MKM is an isolated memory space placed inside the confidential area of NSP. The PE has an access in the address line of MKM but the proposed partitions ensure that no physical data line is established between the processor area and the confidential area, which enforces a prevention from all processor related attacks. DPC is a custom IP connected as slave component of PE through AXI bus. The PE has an access into 8-bit slave register of DPC named as $DPC\_SREG(7:0)$. The least three significant bit of $DPC\_SREG(2:0)$ are connected with Master Key Memory (MKM). The MKM(2) is 1 for memory writing operation and 0 for memory reading operation. MKM(1:0) decides the address of four keys such as 00 for $client\_write\_MAC\_key$, 01 for $client\_write\_key$, 10 for $server\_write\_MAC\_key$, and 11 for $server\_write\_key$. $DPC\_SREG(3)$, $DPC\_SREG(4)$ and $DPC\_SREG(5)$ are connected to the enable line of RSA block, SHA block and AES block respectively. $DPC\_SREG(6)$ connected with the selecting input of MUX_hash to decide whether the hash value will be stored into master key memory for *master key gen process* or into shared memory for *Hash* process. The $DPC\_SREG(7)$ is connected with the selecting input of MUX_en to decide whether data input of encryption block comes from RNG block for *pre masterkey gen process* or from shared memory for *encryption* process. In Table 2 the DMA addresses and the value of $DPC\_SREG(7:0)$ are shown for 4 different operations such as *Pre master key Generation*, *Master Key Generation*, *Encryption* and *Hash*.

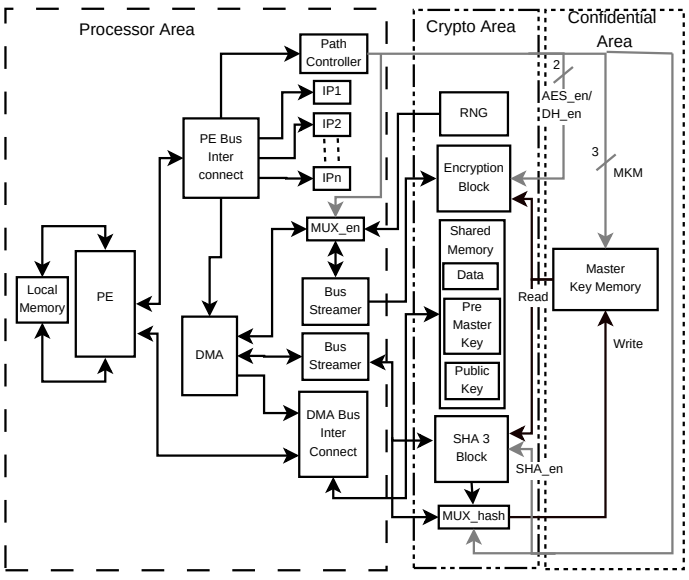

Figure 4: Partition Architecture

Table 2: DMA Addresses & Control Signals

Proc Name	DMA TxAddr	DMA RxAddr	MUX_en	MUX_hash	RSA_en	SHA_en	AES_en	MKM $w\bar{r}$:ad(1:0)
Pre Master Key Gen	×	Pre Master	1	0	1	0	0	× × ×
Master Key Gen ($cl\_wr\_mac\_key$, $ser\_wr\_mac\_key$, cl_wr_key ser_write_key)	Pre Master	×	×	1	1	0	0	1 0 0
	Pre Master	×	×	1	1	0	0	1 0 1
	Pre Master	×	×	1	1	0	0	1 1 0
	Pre Master	×	×	1	1	0	0	1 1 1
Encryption	Plain Text	Cipher Text	0	0	0	0	1	0 0 0
Hash	Plain Text	Hash Text	0	0	1	0	0	0 1 0

5 Implementation & Results

The proposed KECCEAK architectures are implemented in Artix-7 (XC7A100T, CSG324) FPGA platform using Vivado 2015.2 design Suite. The folded architecture achieves 16.49 Gbps throughput which becomes almost double for LU2 architecture. The Folded and LU2 KECCAK achieves better throughput than [1], [13], [2], [3], [4], [5], [7], [6], [8], [9], [10] and [11]. Table 1 shows, the two stages pipeline adopted in PP2-LU2 architecture achieved better throughput than fastest KECCAK till date proposed in article [12]. The timing diagram for KECCAK core is shown in Fig. 5 where DMA sends 34 word through S_AXIS_TDATA keeping the S_AXIS_TVALID high for 34 clock. The S_AXIS_TLAST indicates 34^{th} word and starts KECCAK core. The Folded KECCAK takes 24 clock cycles and produces 256 bit hash data which passed to DMA while master port M_AXIS_TVALID keeps high. As shown in Fig. 5, the whole computation process are divided into 3 finite states. Memory reading by DMA takes 34 clocks, hashing process takes 24 clock cycles and memory writing takes 8 clock cycles. The timing diagram of LU2 and PP2-LU2 is selfsame with fig. 5, except the hashing time. LU2 and PP2-LU2 consumes 12 and 6 clock cycles respectively. The power consumption of Folded, LU2 and PPL2-LU2 architecture are 166.959 watt, 701.979 watt and 1754.825 watt respectively.

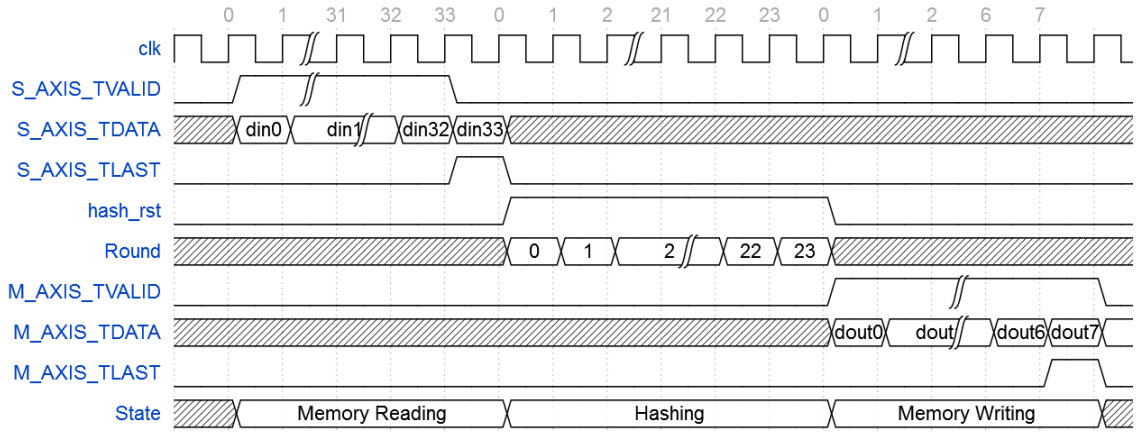

Figure 5: Timing Diagram of Folded KECCAK Architecture

6 Conclusions

This paper proposes 3 high speed architectures for KECCAK hash algorithm. The loop unrolled and pipelined approaches create a major acceleration in PPL2-LU2 architecture. The high speed KECCAK finds applications mainly in NSP based platform. The security protocol running in proposed partitioned NSP, uses KECCAK for hashing and master Key generation purposes. The isolated key memory prevents leakage of key information. DMA provides a smart data transfer between main memory to crypto coprocessor keeping the processor free for control signal generation, data interface hazards, and data path setup tasks.

References

[1] G. Provelengios, P. Kitsos, N. Sklavos, and C. Koulamas. Fpga-based design approaches of keccak hash function. In *2012 15th Euromicro Conference on Digital System Design*, pages 648–653, Sept 2012.

[2] Stéphanie Kerckhof, François Durvaux, Nicolas Veyrat-Charvillon, Francesco Regazzoni, Guerric Meurice de Dormale, and François-Xavier Standaert. *Compact FPGA Implementations of the Five SHA-3 Finalists*, pages 217–233. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[3] Kashif Latif, M. Muzaffar Rao, Athar Mahboob, and Arshad Aziz. *Novel Arithmetic Architecture for High Performance Implementation of SHA-3 Finalist Keccak on FPGA Platforms*, pages 372–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[4] Abdulkadir Akin, Aydin Aysu, Onur Can Ulusel, and Erkay Savaş. Efficient hardware implementations of high throughput sha-3 candidates keccak, luffa and blue midnight wish for single- and multi-message hashing. In *Proceedings of the 3rd International Conference on Security of Information and Networks*, SIN '10, pages 168–177, New York, NY, USA, 2010. ACM.

[5] N. Moreira, A. Astarloa, U. Kretzschmar, J. Lázaro, and E. Molina. Securing ieee 1588 messages with message authentication codes based on the keccak cryptographic algorithm implemented in fpgas. In *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, pages 1899–1904, June 2014.

[6] P. Yalla, E. Homsirikamol, and J. P. Kaps. Comparison of multi-purpose cores of keccak and aes. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 585–588, March 2015.

[7] T. Honda, H. Guntur, and A. Satoh. Fpga implementation of new standard hash function keccak. In *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, pages 275–279, Oct 2014.

[8] K. E. Ahmed and M. M. Farag. Hardware/software co-design of a dynamically configurable sha-3 system-on-chip (soc). In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 617–620, Dec 2015.

[9] Kris Gaj, Ekawat Homsirikamol, Marcin Rogawski, Rabia Shahid, and Malik Umar Sharif. Comprehensive evaluation of high-speed and medium-speed implementations of five SHA-3 finalists using xilinx and altera fpgas. *IACR Cryptology ePrint Archive*, 2012:368, 2012.

[10] B. Jungk and J. Apfelbeck. Area-efficient fpga implementations of the sha-3 finalists. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 235–241, Nov 2011.

[11] J. Winderickx, J. Daemen, and N. Mentens. Exploring the use of shift register lookup tables for keccak implementations on xilinx fpgas. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, Aug 2016.

[12] Harris E. Michail, Lenos Ioannou, and Artemios G. Voyiatzis. Pipelined sha-3 implementations on fpga: Architecture and performance analysis. In *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, CS2 '15, pages 13:13–13:18, New York, NY, USA, 2015. ACM.

[13] Michaël Peeters Guido Bertoni, Joan Daemen and Gilles Van Assche. Keccak sponge function family main document. page available on line at http://keccak.noekeon.org, April 2009.

[14] François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, David Samyde, and Jean-Jacques Quisquater. *Power Analysis of FPGAs: How Practical Is the Attack?*, pages 701–710. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[15] Lubos Gaspar, Viktor Fischer, Lilian Bossuet, and Robert Fouquet. Secure extension of fpga general purpose processors for symmetric key cryptography with partial reconfiguration capabilities. *ACM Trans. Reconfigurable Technol. Syst.*, 5(3):16:1–16:13, October 2012.