ISSN: 2393-8528

Contents lists available at www.ijicse.in

International Journal of Innovative Computer Science & Engineering

Volume 2 Issue 6; November-December-2015; Page No. 11-16

Development of high Throughput Architecture of Keccak Hash Function for Cryptography on FPGA

[1] Perika Kalanwesh, [2] Thrived Dharbhashayanam

[1]Student at Sri Indu College of Engineering and Technology, Hyderabad, India.

[2] Design Engineer at Simpli5NG Semiconductor, Hyderabad, India

ARTICLE INFO

Received: 11 Oct. 2015
Accepted 05 Dec. 2015

Corresponding Author:

Perika Kalanwesh

Student at Sri Indu College of Engineering and Technology, Hyderabad, India

ABSTRACT

Security has become a very demanding parameter in today's world of speed communication. It plays an important role in the network and communication fields where cryptographic processes are involved. These processes involve hash function generation which is a one-way encryption code used for security of data. The main examples include digital signatures, MAC (message authentication codes) and in smart cards. Keccak, the SHA-3 (secure hash algorithm) has been discussed in this paper which consists of padding and permutation module. This is a one way encryption process. High level of parallelism is exhibited by this algorithm. This has been implemented on FPGA. The implementation process is very fast and effective. The algorithm aims at increasing the throughput and reducing the area.
Index Terms- Cryptography, encryption, FPGA, hash function, permutation, security.

Introduction

Need for security:

Information technology has become the backbone of modern society, which makes data security an important aspect of research. Now-a-days almost all the confidential facts about the financial status, products, research, customers or employees are stored and handled on computers, or transferred to other devices through an accessible physical medium. Information needs to be shared in such a format, which cannot be exploited in a feasible amount of time by any adversary without the knowledge of the key parameters used to encrypt the information.

Now-a-days, cryptography has been strongly corroborated in information science using the statistical information theory and number theory [1]. With the emergence of the Internet of Things (IoT) applications, research with focus on robust yet simple crypto-systems have gained importance [2]. The development of internet and multimedia contents like audio, video has caused a tremendous increase in the security requirement. To avoid easy purchase of any content via net, or to avoid problems like violation of ownership and illegal distribution, cryptography technique is used. This includes the hash function which is the key of cryptanalysis [3].

SHA-3 (Secure hash algorithm) is the hash algorithm used by NIST. Out the five hash functions competed by NIST, KECCAK was chosen and standardized as the SHA-3 algorithm by NIST. The total internal architecture of sha-3 using Keccak hash function is the 512 bit encryption is carried out. The common algorithms used for security applications were Secure Hash Algorithms; SHA-l and SHA-2.Due to some drawbacks of these algorithms, National Institute of Standards and Technology (NIST) announced a competition on 2nd November, 2007 for the creation of a new cryptographic hash function that would be entitled as SHA-3 family

The competition underwent through three rounds. NIST initially accepted sixty-four submissions by 3151 October, 2008, out of which fifty-one advanced to the initial round in December, 2008. The entries reduced to fourteen in number in the second round in July, 2009. The third round resulted in five final candidates to compete for the SHA-3 title [4]. The finalist SHA-3 candidates selected were BLAKE, Grostl, JH, Keccak, and Skein. The competition ended in October 2012 with Keccak as the winning candidate to be called as SHA-3.Keccak has been selected based on the evaluation Criterions of security, performance and flexibility [5].

High speed implementations have become a need as these algorithms are widely used. Software based implementations may not perform well as on heavily loaded servers. So the demand for high implementation exists. A cryptographic hash function must be sensitive to the smallest change caused in input data. It should result a large difference in output for that small change [6]. The input message may be given in the form of video, audio or text message [7].

Objective:-

Security has become a very demanding parameter in today's world of speed communication. It plays an important role in the network and communication fields where cryptographic processes are involved. In an effort to increase the speed and decrease the delay via pipelining it is proposed to keccak Hash Function algorithm with pipelining. These processes involve hash function generation which is a one-way encryption code used for security of data. Keccak, the SHA-3 (secure hash algorithm) consists of padding and permutation module. The implementation process is very fast and effective. The main examples include digital signatures, MAC (message authentication codes) and in smart cards. Simulation will be done to verify the functionality and synthesis will be done to get the NETLIST. The Simulation and synthesis is done using Xilinx ISE 14.7 functional simulator from Xilinx.

IMPLEMENTATION OF KECCAK HASH FUNCTION FOR CRYPTOGRAPHY

This chapter describes the brief overview of keccak cryptographic hash algorithm that has been developed with the goal to enhance its performances in terms of power consumption and area. It is become the default choice to ensure the information integrity in numerous applications. KECCAK is a family of sponge function. The *sponge function* is a generalization of the concept of cryptographic hash function with infinite output and can perform quasi all symmetric cryptographic functions, from hashing to pseudo-random number generation to authenticated encryption.

Hash Technology:-

The sponge construction is a simple iterated construction for building a function *f* with variable-length input and arbitrary output length based on a fixed-length transformation or permutation operating on a fixed number *b* of bits. Here *b* is called the width.

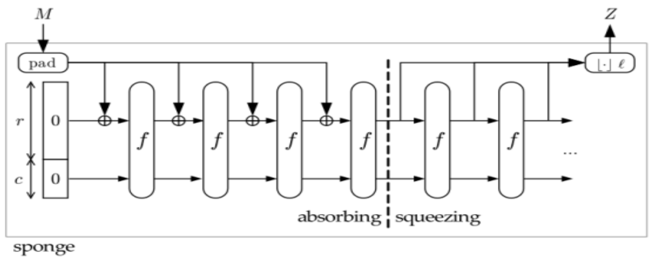


Figure 1: Sponge Function

The sponge construction builds a function SPONGE [*f* , pad, *r*] using a fixed-length transformation or permutation *f* , a sponge-compliant padding rule “pad” and a parameter bit-rate r. A finite-length output can be obtained by truncating it to its ℓ firstbits. This instance of the sponge construction is called sponge function [9].The sponge construction operates on a state of b = r + c bits. The sum r+c determine the width of the permutation used in the sponge construction and are restricted to values in {25, 50, 100, 200, 400, 800, 1600}.

The sponge construction process contains:

The Absorbing phase: The first input block of length r is xored with r bits of the state. The transform function is applied to the state results. Like the previous, the next block is addition modulo 2 with the first results. This continues until all this input is processed.

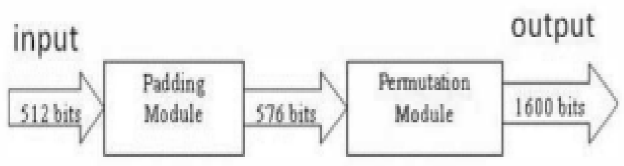
The Squeezing phase: the outputs contain the first r blocks. They are returned from this state and the transformations are continued until all the blocks make for the output length desired are obtained.

The sponge construction is applied to KECCAK-f, so we applied the padding to the message input for obtaining the KECCAK-f [r,c]. With c is capacity and r is bitrate. All the operations on the indices are done modulo 5. A signify the complete permutation state array, and A[x,y] show a particular lane in that state.

Implementation of Keccak Hash Function Algorithm:

The Keccak SHA-3 algorithm justifies the cryptanalysis importance. The variable input is converted to fixed one which the help of hash function. The output obtained is mainly referred as the hash digest. Keccak core consists of two modules i.e. padding and permutation.

Input



In padding module the input is padded to the length equivalent to i.e. the bit rate. Here, the padded output shows the result of 576 bits [10]. The input is of 512 bits, which are processed at samples of 64 bits each, for 8 times each. These 64 bits are zero padded to 576 bits output. A buffer is used by the padding module to arrange the input bits to 576. When buffer is full, the permutation module starts the calculations, the buffer is cleared and the padding module waits for the next input

Padder module:

The padder module consists of RegA, shifter, 2:1 mux, 576-bit buffer as shown in Fig3.3.From the given message, first 64 bits of data is temporarily stored in Reg_A and data is forwarded to shifter. If the control signal In_ready is high, then it indicates that the 64 input bits are ready and if the control signal In_ready is low, then it indicates that all blocks of message are consumed. The shifter will left shift the data by 64 times and then forwarded to buffer. The buffer is of 576-bit wide, the new 64 bits of data is consumed only when the buffer is not full. In the second round, the data in the buffer is left shifted by 64-bits and get concatenated with new 64 input bits. The process continues until the 576-bit buffer is full and the padder output is forwarded to permutation module. The next data blocks are padded, if the padder module receives the acknowledgement signal ackn from permutation module.512 bits

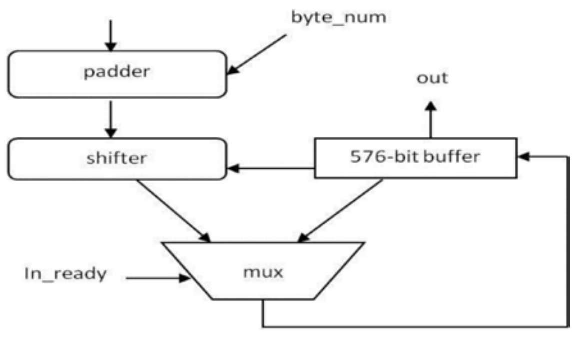


Figure 3: Padding module

Permutation module:

Permutation module as depicted in Fig. 3.4, gives an output of 1600 bits. Here, the XOR-ing is done between the output of padding module and the input given, and is further processed. This process continues until n output bits are produced. Two rounds are performed per clock cycle. Multi -rate padding is used by KECCAK. For a given input, a single 1 bit is appended and then minimum number of 0 bits are appended which are followed by a single 1 bit. This results in the output length to be a multiple of 576. Pi is the ith block of P[12] .

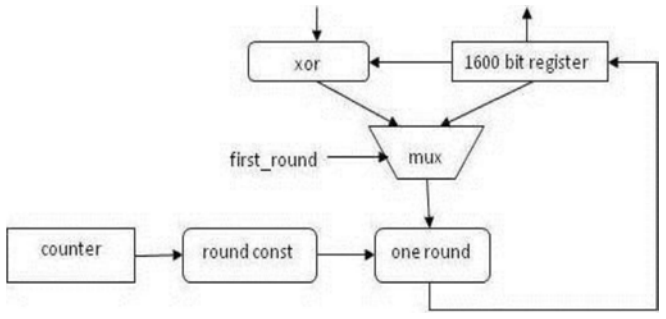


Figure 4: Permutation module

The permutation module performs 2 main functions: 1) f-permutation and 2) Truncation. For the padded 576 bits of data remaining 1024 zeroes will be added such that r+c=1600. The 1600 bits are arranged as 5x5 state arrays with 64 bit word length. If the control signal First_round is high then the padded data will be applied for transformation block and immediately the control signal First_round is disabled, such that no more padded data are allowed. As and when the padded data is consumed by permutation module an acknowledgement signal Ackn will be sent to padder module to pad next block of data.

Permutation modules. It also explains about the five operations (theta, pie, rho, chi and iota) in the permutation module.

IMPLEMENTATION OF PIPELINED KECCAKHASH-FUNCTION FOR CRYPTOGRAPHY

Introduction:

This chapter describes the brief overview of keccak cryptographic hash algorithm with pipelining that has been developed with the goal to enhance its performances in terms of frequency, speed and to decrease the delay. It also gives a brief description about pipelining working and how it will be used to decrease the delay and increase the frequency.

Implementation of Keccak hash function using pipelining:

The pipeline is a set of data processing elements connected in series, so that the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time -sliced fashion; in that case, some amount of buffer storage is often inserted between elements. A common technique for increasing the throughput of electronic circuits is that of pipelining. Pipelining consists of breaking long combinatorial paths by introducing clocked memories; this has the effect of dividing the circuit into sections, in which calculations are run independently. The output of a pipeline section becomes the input of the next one at each clock cycle; while generally preserving the global latency, pipelining increases the throughput of a circuit because several instances of the problem are injected at each clock cycle and are processed independently inside each of the sections. The clock pulse is reduced as a result of the breaking of the combinatorial path, thus allowing higher clock frequencies to be used.

In this context, the algorithm keccak has characteristics that enable the use of pipelining as a solution, since there is a data dependency only of the first level, at which data processing elements connected in series. The data dependency of the first level allow the pipeline stays full until the end of execution, eliminating the hazards of the structural type of data and control architectures commonly found in pipeline implementations. However there is the initial cost of four cycles for total completion of all stages of the pipeline and four cycles to empty the same final stages. Pipelining technique is implemented in the permutation module of the keccak hash function. For the pipeline technique, we inserted two registers in the round transformation: the first register is inserted between the Pi operation and Chi operation, so as to divide the critical path in almost the half. The second register is implemented at the end of the KECCAK round.should be noted that adding two registers between in combinational path results increasing the maximal frequency.

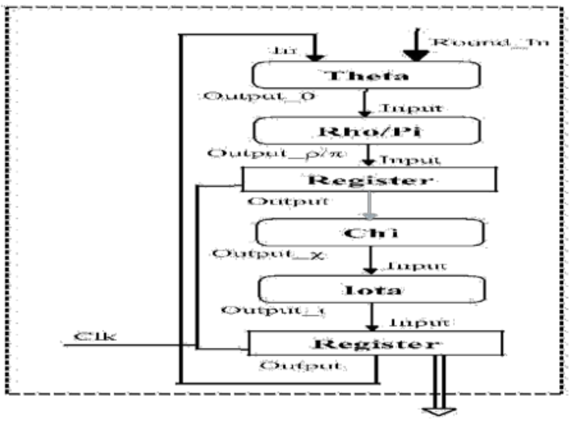


Figure 8: Proposed KECCAK round pipelined architecture

In the first round the pipeline is filled with the original blocks of data and after this round is the processed blocks are reused in the new round. At each stage of the pipeline is necessary to create a buffer capable of storing the information processed in the previous iteration. This buffer stores the state of partial array processed in the previous cycle. In the pipeline architecture are operated simultaneously 4 different data blocks. As can be seen in figure, the first data block is processed in one cycle [T| 1 | 0] this ends in the fourth round, Cycle 4 [I | 1 | 0]. The application of the proposed methodology allows increasing the data processing speed and decreasing the number of clock cycles.

	Round 0				Round 1			
Cicle s	1	2	3	4	5	6	7	8
θ	T|1|0	T|2|0	T|3|0	T|4|0	T|1|1	T|2|1	T|3|1	T|4|1
ρπ		P|1|0	P|2|0	P|3|0	P|4|0	P|1|1	P|2|1	P|3|1
χ			X|1|0	X|2|0	X|3|0	X|4|0	X|1|1	X|2|1
ι				I|1|0	I|2|0	I|3|0	I|4|0	I|1|1

Figure 10: pipeline stages demonstration

Summary:

This chapter explains about the keccak hash function with pipelining. The registers are used for implementing pipelining operation. The entire process of pipelining between the rounds in the permutation module is explained.

RESULTS

Simulation Results

The test bench is developed in order to test the modeled design. This developed test bench will automatically force the inputs and will make the operations of algorithm to perform.

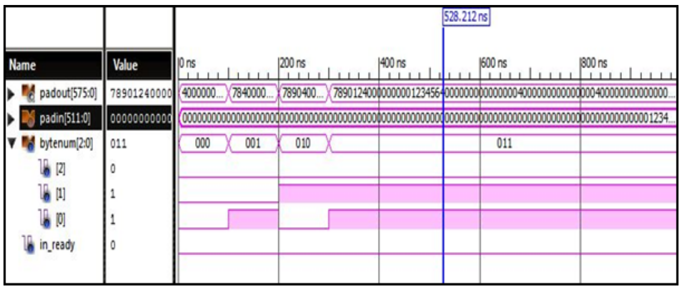


Figure11: HDL simulation result of padding module

The core of the Keccak algorithm is the permutation function which is repeatedly applied to a fixed-length state of b = r + c bits, where r and c are bit rate and capacity of the algorithm. Higher values of r improve the speed whereas higher values of c correspond to higher security level. Permutation module with initially 576 bit data is to XORed with the default value in the register and fed to the mux. The mux generates the output based on the round control pin. This pin goes low if the round value is 23. After the completion of the total 24 rounds we get 1600 bits output in the permutation module.

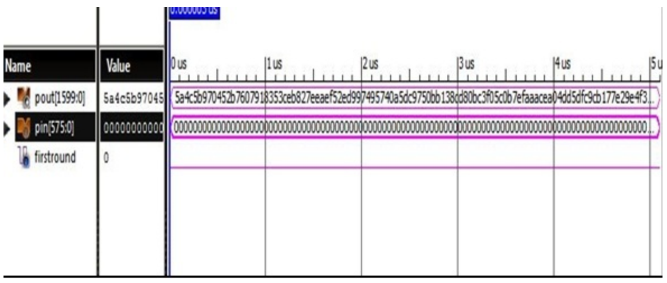


Figure 12: HDL simulation result of permutation module

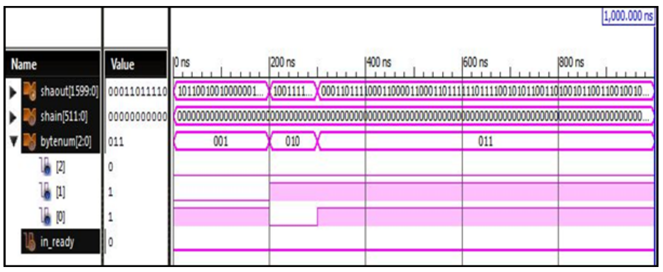


Figure 13: HDL simulation result of keccak hash function

After completion of keccak hash function the Pipelining technique is implemented in the permutation module of the keccak hash function. For the pipeline technique, we inserted two registers in the round transformation: the first register is inserted between the Pi operation and Chi operation, so as to divide the critical path in almost the half. The second register is implemented at the end of the KECCAKround.

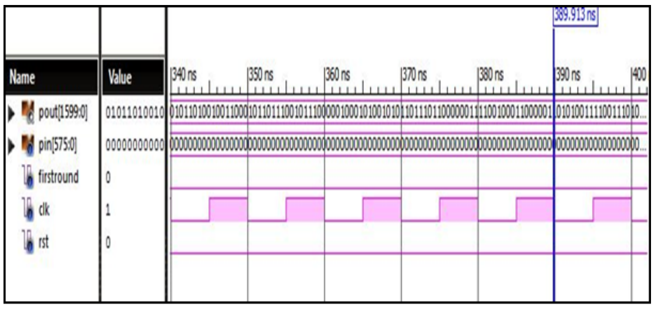


Figure 5.4: Figure14.HDL simulation result of permutation module with pipelining

In figure 5.4 simulation result of permutation module with pipelining is presented. In this input is given as 512 bits from the padder module and then remaining bits are added i.e. from the register and total five operations are performed in one round and total we have 24 rounds, after 24 rounds the output will be taken. In figure 5.5 simulation result of top module keccak hash function with pipelining is presented.

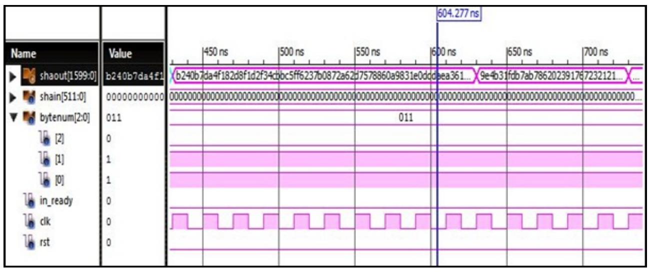


Figure 15: HDL simulation result of keccak hash function with pipelining

Synthesis results:
The developed pipelined keccak hash function is simulated and verified their functionality. Once the functional verification is done, the RTL model is taken to the synthesis process using the Xilinx ISE tool. In synthesis process, the RTL model will be converted to the gate level net list mapped to a specific technology library. The design of pipelined keccak hash function is synthesized and its results were analyzed as follows.

Table 4: Logic utilization of keccak hash function

Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	52800	712000	7%
Number of fully used LUT-FF pairs	0	52800	0%
Number of bonded IOBs	2052	1100	186%

The Device utilization summary of the keccak hash function without pipelining is shown below.
Device utilization summary:

Selected Device: 7vx1140tflg1930-2
Slice Logic Utilization:
Number of Slice LUTs: 52800 out of 712000 7%
Number used as Logic: 52800 out of 712000 7%
Slice Logic Distribution:
Number of LUT Flip Flop pairs used: 52800
Number with an unused Flip Flop: 52800 out of 52800 100%
Number with an unused LUT: 0 out of 52800 0%
Number of fully used LUT-FF pairs: 0 out of 52800 0%
IO Utilization:
Number of IOs: 2052
Number of bonded IOBs: 2052 out of 1100 186% (*)
It is the timing summary of the keccak hash function without pipelining.

<u>Timing Summary</u>:
Speed Grade: -2 Minimum period: No path found
Minimum input arrival time before clock: No path found
Maximum output required time after clock: No path found
Maximum combinational path delay: 53.559ns
The Logic Utilization of keccak hash function with pipelining is shown in the below table
Selected Device: 7vx1140tflg1930-2

Table 5: Logic utilization of keccak hash function with pipelining

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	34045	1424000	2%
Number of Slice LUTs	52609	712000	7%
Number of fully used LUT-FF pairs	0	86650	0%
Number of bonded IOBs	2054	1100	186%
Number of BUFG/BUFGCTRLs	2	128	1%

Macro Statistics
# Register: 34045
Flip-Flops: 34045
Slice Logic Distribution:
Number of LUT Flip Flop pairs used: 86976
Number with an unused Flip Flop: 52800 out of 86976 60%
Number with an unused LUT: 34176 out of 86976 39%
Number of unique control sets: 22
It is the timing summary of the keccak hash function with pipelining.

<u>Timing Summary</u>:

Speed Grade: -2

Minimum period: 5.528ns (Maximum Frequency: 180.904MHz)
Minimum input arrival time before clock: 2.089ns
Max output required time after clock: 2.783ns
Maximum combinational path delay: No path found
Summary:
In this chapter the simulations results of keccak hash function with and without pipelining are explained above. The logic utilization of both with and without pipelining are shown in the table 4 and table 5. By observing the simulation and synthesis report we conclude that delay is reduced and frequency is increased by using keccak hash function with pipelining.

CONCLUSION

Use of cryptographic algorithms like Keccak provide better security needed in today's world. The algorithm implemented in this paper also meets the less area and high throughput requirement. The merit compared with the eXlstmg architecture show increase in the throughput to area ratio. The area has been reduced to the number of 1614 registers and 8030 LUTs. The maximum frequency achieved for the hash function is 267.2 MHz and 11 clock cycles are required. The overall 512 bit encryption process requires only 10 rounds. Thus speed constraint is achieved. Thus the use of Keccak hash function proves beneficial in cryptography wherever security constraint is to be achieved.

REFERENCES

1. Siavash Bayat-Sannadi. Member, iEEE, Mehran MozaffariKermani, Member, iEEE and Arash Reyhani-Masoleh, Member, iEEE" Efficient and Concurrent Reliable Realization of the Secure Cryptographic SHA-3 Algorithm" iEEE TRANSACTiONS ON COMPUTER-AiDED DESiGN OF iNTEGRATED CiRCUITS AND SYSTEMS, VOL. 33, NO. 7, JULY 2014.
2. Santosh Ghosh and Ingrid Verbauwhede, Senior Member, iEEE" BLAKE-512-Based I 28-Bit CCA2 Secure Timing Attack Resistant McEliece Cryptoprocessor " iEEE TRANSACTIONS ON COMPUTERS, VOL. 63, NO. 5, MAY 2014.
3. Fatma Kahri, Belgacem BoualJegue, Mohsen Machhout and Rached Tourki, "An FPGA implementation of the SHA-3 : The Blake hash Function," IEEE 2013 JO'h international MultiConference on Systems, Signals & Devices (SSD) Hammamet, Tunisia, March 18-21,2013
4. Imad Fakhri Alshaikhli, Mohammad A. Alahmad, and Khanssaa Munthir, "Comparison and Analysis Study of SHA-3 Finalists," iEEE 20i 2 international Conference on Advanced Computer Applications and Technologies.
5. Saeid Nourizadeh and Mojtaba Javanmardi, "Cryptanalysis of the Reduced-Round Version of JH," iEEE 6'h international Symposium on Telecommunications (iST'2012).
6. Keccak Hash Function, NIST (National Institute of Standards and Technology),(2014, Mar.) [Online]. Available: http://csrc. nist.gov Igroups/ST Ihash/sha-3.
7. D. -J. Bernstein and T. Lange. (2012). The new SHA-3 software shootout.e-Print [Online].
8. M. Kne'zevi'c et aI., "Fair and consistent hardware evaluation of fourteenround two SHA-3 candidates," iEEE Trans. Very Large Scale integr.(VLSi) Syst., vol. 20, no. 5, pp. 827-840, May 2012.
9. K. Latif, M. Rao, A. Aziz, and A. Mahboob, "Efficient hardware implementations and hardware performance evaluation of SHA-3 finalists," in Proc. Con! SHA-3 Candidate, pp. 1-14, Mar. 2012.
10. E. M. Shakshuki, N. Kang, and T. R. Sheltami, "EAACK-A secure intrusion detection system for MANETs," iEEE Trans. ind. Electron.,vol. 60, no. 3, pp. 1089-1098, Mar. 2013.

AUTHORS DETAILS:

Perika Kalanwesh: has completed his B.Tech in Electronics and Communication Engineering from Princeton College of Engineering and Technology, J.N.T.U.H affiliated college in 2011.He is pursuing his M.Tech in VLSI System Design from Sri Indu College of Engineering and Technology , J.N.T.U.H affiliated college.

Thrived Dharbhashayanam: has been working as Designer at Simpli5NG Semiconductor since 2009, He received his bachelor degree in ECE from JNTU and Masters Degree in VLSI Design from VIT University, His research Interest is Network Security and Cryptography