*Asian J. Exp. Sci., Vol. 31, No. 2, 2017; 17-26*

*Online available on : www.ajesjournal.com, ISSSN : 0971-5444*

# Simulation and implementation of *Hash Algorithm*-3 (*SHA-3*) on FPGA

**Yaghoub Mirchi** and **Siavash Amin Nejad**
Rasht Branch, Faculty of science and Research
Department of Electronics, Islamic Azad University, Rasht Branch, Iran- 4147654919
Email : peyman.mirchi@gmail.com
Received: March 5, 2017; Revised: June 16, 2017; Accepted: June 20, 2017.

**Abstract :** Secure hash functions refer to key component of many applications, such as digital signature projects or authentication systems. Many of these applications are used in cost-sensitive markets, and thus implementation of such components with low budget is very important. In 2012, the National Institute of Technology (NIST), after a 5-year long-term competition selected Keccak algorithm as a Hash function that must be standardized as SHA-3. During and after this competition, variety of Keccak implementations have been proposed and evaluated for hardware platforms. However, few results were released for non-standard implementations. Since to date SHA-3 algorithm has not become operational and presented only in few workshops, in this research firstly what is this algorithm and how it works are considered and then this algorithm is examined using software Matlab. After examining software via Quartus software, the program has been examined in hardware via Verilog language and finally implemented on EP2C20F484C7N board and the results have been compared with the real value which has been presented via Keccak.

**Keywords**- Keecak, SHA-3 algorithm, Software Review, hardware review, implementation.

## Introduction

Hash function which is called Hash, Hash Code, Digest, and Message. Digest is a primary function which converts the *data of arbitrary length* to fixed-*length data*. Hash can be considered as digital fingerprint of a date. With this method, it can obtain fixed-length string of a data which is encrypted with mathematical methods as "one-sided" (Bertoni *et al*., 2011). Discovering the major string from hash string (inverse operation) is almost impossible. Another point is that each data creates a distinctive hashed string. These properties convert hashing to an ideal and efficient method for storing passwords in programs. Even if a hacker (Hacker) can penetrate into the system and database and gain a part of the information (including hashed passwords), passwords cannot be retrieved from them (Chang *et al*., 2012). Hash algorithms are used to compress files and create a small data which is signature. In previous standards such as SHA-1 and SHA-2, algorithm weaknesses existed which brought down their degree of security. *Hash Algorithm*-3 (*SHA-3*) is the newest hash algorithm which was standardized to increase security of electronic signature after long competition between experts in 2014. In this system, if a bit of data file changes, the signature changes and it cannot produce a similar signature with changes in file. Coding is used to create security in data transmission and/or electronic signature to ensure the accuracy of the information (Bertoni *et al*., 2007). In 2014, in International Conference on Electronics, it was stated that a high speed hardware which had been selected for keccak has improved as a standard hash function named SHA-3 and its function was evaluated via various FPGA platforms against SHA-1 and SHA-2 circuits. The results showed that KECCAK is suitable for high speed implementation, but implementing it on new FPGA devices is difficult. SHA-3 competition was ended at the late 2012 by introducing KECCAK as a *winner algorithm*. Sravani and Pallavi (2015) gained proper results by combining three blocks of algorithm RHO, PI, CHI and converting to a block. This caused storing 16% in the entire program and reducing number of program courses and increase of the highest operational frequency (Ethan Heilman, 2012). Hash function has been suggested as a cryptographer of a wide range of RFID protocols. Since KECCAK was selected by NIST as the winner of SHA-3 competition in 2012, this question was raised that to which extent we can resolve KECCAK limitations to reduce cost in RFID. PETER PESSL and MICHAEL HUTTER by presenting a hardware implementation of KECCAK which aims to lower power enabled to resolve KECCAK limitations. Their project is smaller than the smallest SHA-1 and SHA-2 implementation. Athanasiou *et al.* (2014) proposed architecture on Xilinx vitrex-5, vitrex-6, vitrex-7 comparing existing FGAP implementation gained considerable improvements. Specifically, better operational power was gained for Vitrex-5 architecture.

A large body of works has been made on FPGA Implementations of SHA-3 Candidates; most of previous implementations are optimized for high power (output) and few implementations exist for compressed plans. The early results on compressed BLAKE implementations use the same method with proposed plan in this article, in which RAM block has been used, thus the results are not comparable. Here, all the algorithms have been distributed using RAM (Bertoni *et al*., 2014). Thus, all the required references are considered in counting pieces (Table 1).

Table-1 Results from implementation for Virtex-5 FPGAs.

| Algorithm | Slices | BRAM | MHz | MBit/s | MBit/s/Slice |
|---|---|---|---|---|---|
| Grøstl | 368 | 0 | 305 | 975 | 2.64 |
| Keccak | 393 | 0 | 159 | 864 | 2.19 |
| BLAKE | 251 | 0 | 211 | 477 | 1.90 |
| Skein | 519 | 0 | 299 | 262 | 0.50 |
| JH | 193 | 0 | 283 | 23 | 0.11 |

In this regards, the present research states new results in this contexts. They reported the results for all final candidates of SHA-3 for Vitex-6 and Spartan-6 FPGAs. They implemented all candidates for two 256 and 512-bit *digests*. Some of their plans differ from the plans presented in the present research, e.g. new Keccak plan is so faster, while other plans gain much similar function (output)( Merkle, 1987).

**Overview of the SHA-3 and its history**

National Institute of Standards and Technology is working on the process of selecting a latent hash algorithm through a public competition. New hash algorithm entitled "SHA-3" will be mentioned and hash SHA-2 algorithms which have been defined in secure hash standard, FIPS 180-3, will be completed. The selected algorithm to be suitable for use by U.S. government as well as private sector at the end of competition and available to everyone worldwide has been the main aim. Since then, this competition as referred to SHA-3 competition is mentioned in all documents (Bertoni *et al*., 2011). The competition is NIST response to recent advances in the discovery of cryptanalysis of hash algorithm, such as standard SHA-1 hash algorithm. A rush by Hongobo Yu and Yiqun Lisa Yin which was developed by other people has seriously discussed on the safety in the use of SHA-1 in digital signatures and other applications that require collision resistance. While SHA-2 family provides an essential alternative from hash algorithms; NIST expects from selected SHA-3 to provide the security which is as well as SHA-2 algorithms with considerable improvement in the period or extra features (NIST, CSD). In preparation for the competition SHA-3, NIST held workshops on 31 October to 1 November 2005 and 24 and 25 August 2006 to discuss about the status of hash algorithm and create a path towards standard development of a new hash algorithm. As a result, NIST established a public competition similar to what used to select the advanced standard encryption (ASE) (NIST Releases SHA-3). NIST released the selection requirements and SHA-3 algorithm evaluation standards in January 2007 for public comment in federal registry notice; these evaluation standards were updated based on public feedback and put in a second ad of the Federal Registration which was released on 2 November 2007, called a new hash algorithm and specified start of competition (NIST Cryptographic Algorithm Validation Program (CAVP)). Candidate requests (presentations) have been in the 31 October 2008 deadline, at which NIST received delegate (assignment) package, presented 21 candidate algorithms for ASE competition in 1998. Among 64 requests (presentations), NIST on 10 December 2008 announced accepting 51 candidates for the first round as candidates met the minimum acceptance standards, indicating the start of the first round of SHA-3 competition. The first round of candidates packages were put online in www.nist.gov/hash-competition for public visit (Bertoni *et al.,* 2012). NIST held the first SHA-3 Candidate Conference on 25- 28 February 2009 in Leuven, Belgium, which the presenters of the accepted candidates at first round were invited to present their algorithms. NIST discussed on a program to reduce the first round candidates to the more controllable numbers for future studies till summer 2009 and started the second round competition. Since then, NIST was used more than ever. As a result of studies on first round candidates, NIST announced selection of 14 algorithms in 24 July 2009 as the second round candidate so as to move towards the second round competition. In following, a time of major events on SHA-3 competition in several courses:

-1/11/2005: latent workshops, NIST, Gaithersburg, MD.

-25/8/2006 : second latent hash workshop, UCSB, CA.

-23/1/2007: Federal registry notice-announcement of new hash algorithm to review federal information processing standard 180-2, secure hash standard.

-11/2/2007: federal registry notice-the announcement for needing to candidate algorithms *delegate* for latent hash algorithm SHA-3.

31/10/2008– The latest deadline for presentation SHA-3.

-1/11/2008: first round of competition.

-10/12/2008: the first round candidates were announced. Public comment on first round candidates was started.

-28/2/2009: the first conference on candidate SHA-3, KU Leuven, Belgium.

- 24/7/2009the first round was ended and the second round was started. The second round candidate was announced. The public comment round on second round candidate was started.

**SHA-3 hash function**

SHA-3 hash algorithm which is recognized as Keccak is a latent hash function which was projected by Bertogne et al. this algorithm is based on *foam construction (absorption/pressure) which is a group of algorithms with limited internal status which takes a* input *bit stream* of *any length* and output *bit stream* of any *desired length*. This algorithm receives a 3D matrix which is called the matrix of size (length Word 5: 5) as input. With regard to the desired output, this algorithm uses two parameters for sponge construction. The two input parameters include bitrate (bit rate) r and c capacity. During absorption stage (major hash calculation), the rate of bit transmission is the primary status with the first input part XOR. The result as a

transmission rate of new bit which forms a new status with capacity of primary status matrix is used. This status is fed in the main algorithm process. Then the new obtained status is used as the new primary status and this process continues for *interaction*s. In following, during compression stage, it is assumed that the output is half of the capacity size, thus there are not further calculations. Output widths of potential 224, 256, 384, 512 bits SHA-3 are produced with bit transmission rate of 576, 832, 1024, 1088, 1152 bits. Conversion stage (adsorption process) of SHA-3 includes five separate functions, iterated for 24 times(24 bit length of word).these functions work on a 1600 bits status matrix, a, and explained in following. Lota function uses a curve constant which is different for each curve, but it can be calculated once more before hash calculation.

$$\textbf{Theta}: a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1] \quad (1)$$

$$\textbf{Rho}: a[x][y][z] \leftarrow a[x][y]\left[z - \frac{(t+1)(t+2)}{2}\right], \quad (2)$$

$$with\ t: 0 \le t < 24\ and \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^{t} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}\ in\ GF(5)^{2\times 2}$$

$$or\ t = -1\ if\ x = y = 0$$

$$\textbf{Pi}: a[x][y] \leftarrow a[x'], with \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} \quad (3)$$

$$\textbf{Chi}: a[x] \leftarrow a[x] + (a[x+1] + 1)a[x+2] \quad (4)$$

$$\textbf{Iota}: a \leftarrow a + RC[i_r] \quad (5)$$

## The proposed SHA-3 architecture

To clarify the concept, the proposed SHA-3 architecture is presented. In beginning, SHA-3 communication core has been displayed.

SHA-3 communication hash core

The introduced SHA-3 communication hash core has been displayed in Fig. 1. This core consists of the conversion curve.

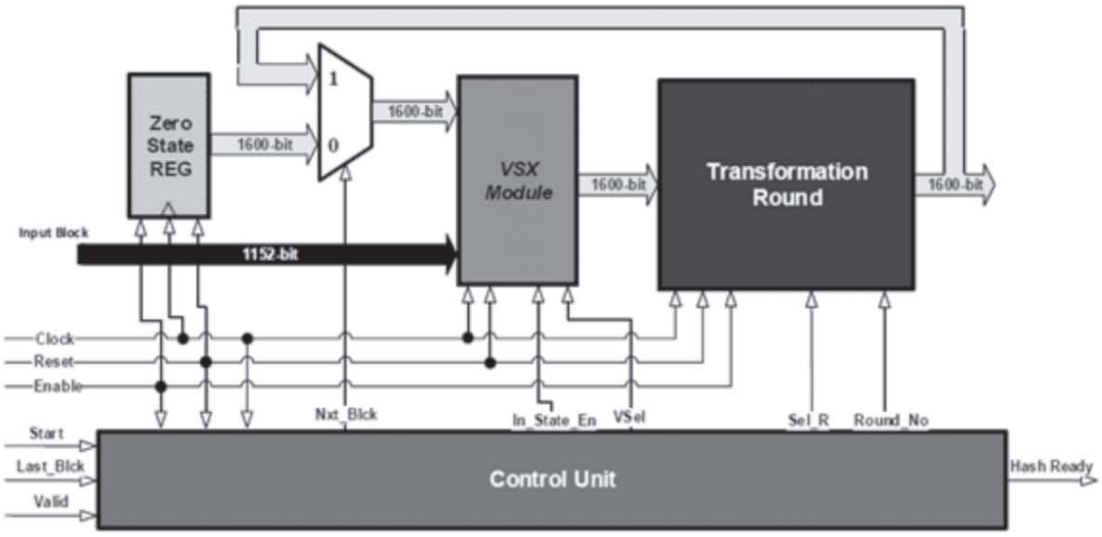


Fig 1. Pipelined SHA-3 architecture.

As displayed via algorithm, a primary zero status is used during the early iteration. Zero status is hold in REGs of the zero status which includes simple counters. One multiplexer of 2 to 1 exists next to REGs of zero status which is responsible to realize feedback action when a multi-block message is processed. In this state, hash output of the status of current block is fed in VSX so as to become the new input status of XOR. VSX modules are displayed in Fig. 2. These modules include a 1152 bit XOR for primary storing and five additional components which each one is responsible to develop a suitable status at each algorithm version. Then, a 5 to 1 multiplexer is used to pass suitable status to the conversion curve which is based on version selection. In contrast to current SHA-3/Keccak architectures, version selection falls after primary XOR. This selection of plan is made so as to implement this modulus efficiently as a component in FPGA which results in low expenses of routings and improvement in delay and surface standards.

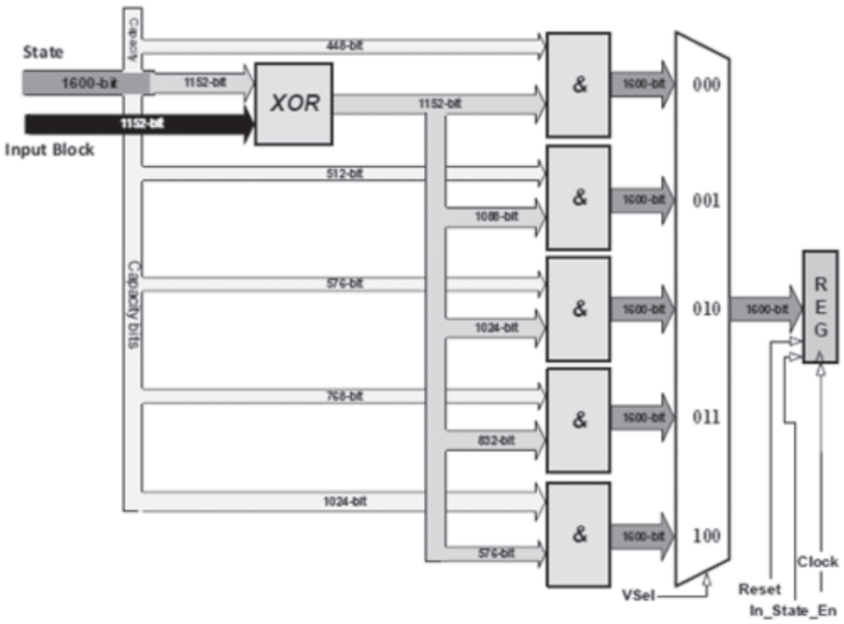


Fig. 2. Selection of VSX modulus and version.

Further, there is a counter after multiplexer. This counter avoids VSX logic to increase the critical path which is inside the conversion curve. However this counter imposes an additional hour cycle inside each information block, its effect on total operational power for large multi-block messages can be little. SHA-3 core is controlled by control unit which is realized using FSM. FSM includes 5 statues which is called idle status; status SI for version selection and primary XOR, statues S2 and S3 for major calculations and statues S4 and S5 for calculating the last hash block. Further, FSM includes a counter which counts to 48 which is correspond to 24 two-stage communication conversion curve iterations. Due to two-stage communication, the proposed core can be fed with two input blocks during the first and second cycles of operation hours. This makes balance in doubling the required hour cycles for processing a block.

**SHA-3 communication conversion curve**

The proposed core conversion curve has been displayed in Figure 3. As shown in this algorithm, this curve includes five modules which realize tasks of five algorithms, i.e. *Theta*، *Rho*، *Pi*، *Chi* and *Iota.* In start of curve, 2 to 1 multiplexer exists for curve feedback.

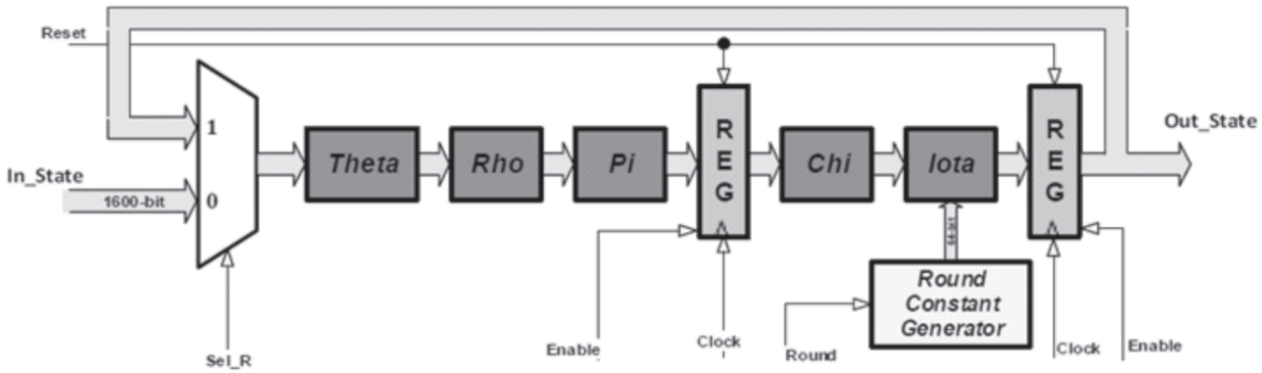


Fig. 3. Round of SHA-3 pipelined conversion.

Suitable Circular constants that are used in Iota function are produced by circular constant productive. There are three other project methods for this modulus: 1-production in flight of Circular constants with an appropriate circuit which performs linear feedback shift counting (LFSR); 2-precalculation of 24 constants and use of 24 counters to store them and a MUX tree to feed these values in *Iota*; 3-use of a circular middle memory which contains pre-calculated constants and feeds the correspond value in each iteration to Iota. We have developed and implemented these alternatives in Xilinx Viretx-5 technology and concluded that the most efficient alternative for operational power has been the first method. To use communication method, two counters are put in the curve. The first counter is put between *Pi* and Chi so as to divide the critical path two half. The second counter at the end of curve is put before feedback branching. Thus, the critical path which is produced due to feedback is almost cut at half. *Control signs* are two communications counting of public hour and reset (restore) and the signs which are fed enable as inputs in core. In contrast, 2 to 1 multiplexer selection symptom as well as circular fixed generator control symptom is divided into the core control unit.

**Results**

**Optimization using LUT samples**

For newer Virtex-5 and FPGAs, the optimization of project is possible with manual modeling of the primary LUT geometric shapes. In the present work, we combined and with multiplexer before . Direct modeling helps for reduced count compared to the older results presented by Jung et al. this idea works properly, because just 3 input bits are required for each output bit, works on one or zero bit in each row and multiplexer selects between and calculation or route-through. Thus, we can package calculation of four output bits in two samples LUT6_2 with four input bits. Multiplexer bit and bit with $x$=0 are assigned to a single LUT6_2 sample which calculates . On the whole, we require three LUT6_2 samples per row.

**Combined settings**

To measure our implementations, we used a systematic method same as ATHENa framework. We combined our project for $d$ {25.2k / 1 $k$ $l$. For each of settings, we used Xilinx ISE 14.5 to combine our project with different optimization settings. Then settings with the best post-place and the common results at the second step of optimization were used so as to increase the operational power with strict timing restrictions. We explained analysis by Keccak-$f$ [1600] and Keccak-$f$ [800] for platforms with less restriction. For analysis of lightweight species, we generalized analysis by Keccak-$f$ [400] and added research on Keccak-$f$ [200]. The best post-place and the common results from architecture of the proposed piece have been displayed in figures 4 and 5 which assume alternative relations for a set of various parameters. The *recognition* space for all Keccak structures under study, Keccak-$f$ [1600]، Keccak-$f$ [800]، Keccak-$f$ [400], Keccak-$f$ [200] are drawn on a logarithmic axis, indicating the space in the pieces on the function specified as Mbit/s. each proposed plan as well as the projects based on FPGA which have been introduced previously in literature review are specified separately. Further, we specify specific bounds of surface-operational power ratio which is a common standard to compare various projects in terms of their return. We present operational power of projects in simple and explicit values as shown in tables 2 & 3.

Table - 2. Results of implementation for Keccak-f [1600] and Keccak-f [800]

| | Design | | | | | | Resource | | Performance | | Metric |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Variant | State structure | Data path ($d$) [Bit] | Capacity ($c$) [Bit] | Rate ($r$) [Bit] | Architecture | Platform | Slices | BRAM/ DSP | Frequency (f) [Mhz] | Throughput ($\mathcal{T}$) [Mbit/s] | [Mbit] [s*slice] |
| KECCAK-$f$[1600] | Slice | 25 | 512 | 1088 | This paper | Virtex-5 | **140** | 0/0 | 200 | 81 | 0.58 |
| | | 50 | 512 | 1088 | This paper | Virtex-5 | 161 | 0/0 | 186 | 151 | 0.93 |
| | | 100 | 512 | 1088 | This paper | Virtex-5 | 195 | 0/0 | 177 | 287 | 1.47 |
| | | 200 | 512 | 1088 | This paper | Virtex-5 | 272 | 0/0 | 166 | 539 | 1.98 |
| | | 200 | 512 | 1088 | [12][1] | Virtex-5 | 393 | 0/0 | 159 | 864 | 2.20 |
| | | 200 | 512 | 1088 | [21] | Virtex-5 | 344 | 0/0 | - | 870 | 2.53 |
| | | 400 | 512 | 1088 | This paper | Virtex-5 | 455 | 0/0 | 158 | 1024 | 2.25 |
| | | 800 | 512 | 1088 | This paper | Virtex-5 | 854 | 0/0 | 151 | 1959 | 2.29 |
| | Lane | 64 | 512 | 1088 | [9] | Virtex-6 | 144 | 0/0 | 250 | 128 | 0.89 |
| | | 64 | 512 | 1088 | [11] | Virtex-5 | 151 | 3/0 | 520 | 501 | 3.32 |
| | | 64 | 512 | 1088 | [10] | Virtex-5 | 159 | 1/0 | 248 | 71 | 0.45 |
| | | 64 | 512 | 1088 | [10] | Virtex-5 | 275 | 0/0 | 260 | 117 | 0.43 |
| | Parallel | 1600 | 512 | 1088 | This paper | Virtex-5 | 1,215 | 0/0 | 195 | 5,054 | 4.16 |
| | | 1600 | 512 | 1088 | [7] | Virtex-5 | 1,338 | 1/0 | 248 | 11,252 | 8.41 |
| | | 1600 | 512 | 1088 | [7] | Virtex-5 | 1,369 | 0/0 | 297 | 13,452 | **9.83** |
| | | 1600 | 512 | 1088 | [8] | Virtex-5 | 1,433 | 0/0 | 205 | 8,747 | 6.10 |
| | | 1600 | 512 | 1088 | [6][1] | Virtex-5 | 2,326 | 0/201 | 306 | 3,120 | 1.34 |
| KECCAK-$f$[800] | Slice | 25 | 256 | 544 | This paper | Virtex-5 | 120 | 0/0 | 227 | 96 | 0.80 |
| | | 25 | 512 | 288 | This paper | Virtex-5 | **112** | 0/0 | 220 | 62 | 0.55 |
| | | 50 | 256 | 544 | This paper | Virtex-5 | 146 | 0/0 | 186 | 158 | 1.08 |
| | | 50 | 512 | 288 | This paper | Virtex-5 | 138 | 0/0 | 168 | 105 | 0.76 |
| | | 100 | 256 | 544 | This paper | Virtex-5 | 186 | 0/0 | 163 | 312 | 1.68 |
| | | 100 | 512 | 288 | This paper | Virtex-5 | 168 | 0/0 | 162 | 205 | 1.22 |
| | | 200 | 256 | 544 | This paper | Virtex-5 | 267 | 0/0 | 155 | 528 | 1.98 |
| | | 200 | 512 | 288 | This paper | Virtex-5 | 249 | 0/0 | 158 | 355 | 1.43 |
| | | 400 | 256 | 544 | This paper | Virtex-5 | 428 | 0/0 | 165 | 1,120 | 2.62 |
| | | 400 | 512 | 288 | This paper | Virtex-5 | 416 | 0/0 | 159 | 717 | 1.72 |
| | | 800 | 256 | 544 | This paper | Virtex-5 | 591 | 0/0 | 205 | 2,785 | **4.71** |
| | Parallel | 800 | 512 | 288 | This paper | Virtex-5 | 524 | 0/0 | 209 | 1,880 | 3.59 |

Table - 3. Results from implementation for Keccak-f [400] and Keccak-f [200] on Virtex-5

| | Design | | | | | Resource | | Performance | | Metric |
|---|---|---|---|---|---|---|---|---|---|---|
| Variant | State structure | Data path ($d$) [Bit] | Message Digest ($n$) [Bit] | Capacity ($c$) [Bit] | Rate ($r$) [Bit] | Slices | BRAM/ DSP | Frequency ($f$) [Mhz] | Throughput[1] ($\mathcal{T}$) [Mbit/s] | [Mbit] [s*slice] |
| KECCAK-$f$[400] | Slice | 25 | 128 | 256 | 144 | 106 | 0/0 | 191 | 57 | 0.54 |
| | | 25 | 128 | 128 | 272 | 108 | 0/0 | 195 | 87 | 0.81 |
| | | 25 | 160 | 160 | 240 | 103 | 0/0 | 186 | 78 | 0.75 |
| | | 25 | 160 | 320 | 80 | 106 | 0/0 | 204 | 39 | 0.37 |
| | | 25 | 224 | 224 | 176 | **102** | 0/0 | 186 | 64 | 0.63 |
| | | 25 | 256 | 256 | 144 | 106 | 0/0 | 180 | 54 | 0.51 |
| | | 50 | 128 | 256 | 144 | 111 | 0/0 | 188 | 113 | 1.02 |
| | | 50 | 128 | 128 | 272 | 134 | 0/0 | 192 | 172 | 1.28 |
| | | 50 | 160 | 160 | 240 | 124 | 0/0 | 186 | 155 | 1.25 |
| | | 50 | 160 | 320 | 80 | 106 | 0/0 | 211 | 81 | 0.76 |
| | | 50 | 224 | 224 | 176 | 122 | 0/0 | 195 | 134 | 1.10 |
| | | 50 | 256 | 256 | 144 | 115 | 0/0 | 184 | 110 | 0.96 |
| | | 100 | 128 | 256 | 144 | 165 | 0/0 | 167 | 201 | 1.22 |
| | | 100 | 128 | 128 | 272 | 176 | 0/0 | 152 | 273 | 1.55 |
| | | 100 | 160 | 160 | 240 | 172 | 0/0 | 176 | 293 | 1.70 |
| | | 100 | 160 | 320 | 80 | 148 | 0/0 | 175 | 135 | 0.91 |
| | | 100 | 224 | 224 | 176 | 160 | 0/0 | 172 | 237 | 1.48 |
| | | 100 | 256 | 256 | 144 | 157 | 0/0 | 173 | 207 | 1.32 |
| | | 200 | 128 | 256 | 144 | 261 | 0/0 | 171 | 410 | 1.57 |
| | | 200 | 128 | 128 | 272 | 261 | 0/0 | 171 | 610 | 2.34 |
| | | 200 | 160 | 160 | 240 | 249 | 0/0 | 174 | 580 | 2.33 |
| | | 200 | 160 | 320 | 80 | 208 | 0/0 | 168 | 259 | 1.24 |
| | | 200 | 224 | 224 | 176 | 239 | 0/0 | 163 | 450 | 1.88 |
| | | 200 | 256 | 256 | 144 | 241 | 0/0 | 177 | 424 | 1.76 |
| | Parallel | 400 | 128 | 256 | 144 | 289 | 0/0 | 236 | 1135 | 3.93 |
| | | 400 | 128 | 128 | 272 | 318 | 0/0 | 306 | 2194 | **6.90** |
| | | 400 | 160 | 160 | 240 | 304 | 0/0 | 304 | 2029 | 6.68 |
| | | 400 | 160 | 320 | 80 | 240 | 0/0 | 283 | 869 | 3.62 |
| | | 400 | 224 | 224 | 176 | 254 | 0/0 | 283 | 1558 | 6.13 |
| | | 400 | 256 | 256 | 144 | 262 | 0/0 | 277 | 1330 | 5.07 |
| KECCAK-$f$[200] | Slice | 25 | 128 | 128 | 72 | 87 | 0/0 | 191 | 62 | 0.71 |
| | | 25 | 160 | 160 | 40 | **80** | 0/0 | 191 | 40 | 0.50 |
| | | 50 | 128 | 128 | 72 | 116 | 0/0 | 175 | 113 | 0.97 |
| | | 50 | 160 | 160 | 40 | 103 | 0/0 | 213 | 89 | 0.86 |
| | | 100 | 128 | 128 | 72 | 144 | 0/0 | 191 | 246 | 1.71 |
| | | 100 | 160 | 160 | 40 | 137 | 0/0 | 188 | 157 | 1.14 |
| | Parallel | 200 | 128 | 128 | 72 | 159 | 0/0 | 339 | 872 | **5.48** |
| | | 200 | 160 | 160 | 40 | 146 | 0/0 | 327 | 545 | 3.73 |

**Implementations with higher security**

Implementations of projects Keccak-*f* [600] and Keccak-*f* [800] are examined in this section. Firstly implementations of the proposed Keccak-*f* [1600] project as well as another existing project are discussed in literature review; then the results from implementation of Keccak-*f* [800] project will be examined with various capabilities. Anyone can see that the projects with operational power of the architecture of the cut at the proposed axis of Keccak-*f* [1600] are scaled with increased d linearly. However, ratio of operational power- appropriate level is not scaled with d (Fig. 4). This effect is probable due to a balance in *electrical energy consumption* which is due to 1600 bit Keccak large memory and control logic which both have a source usage which is less dependent on width of *datapath* d. thus, projects with d = {200, 400, 800} $_{\text{have}}$ almost same surface-operational power return, because consumption of electricity supply is a less prominent part in this project. However, the return declines considerably for smaller projects.

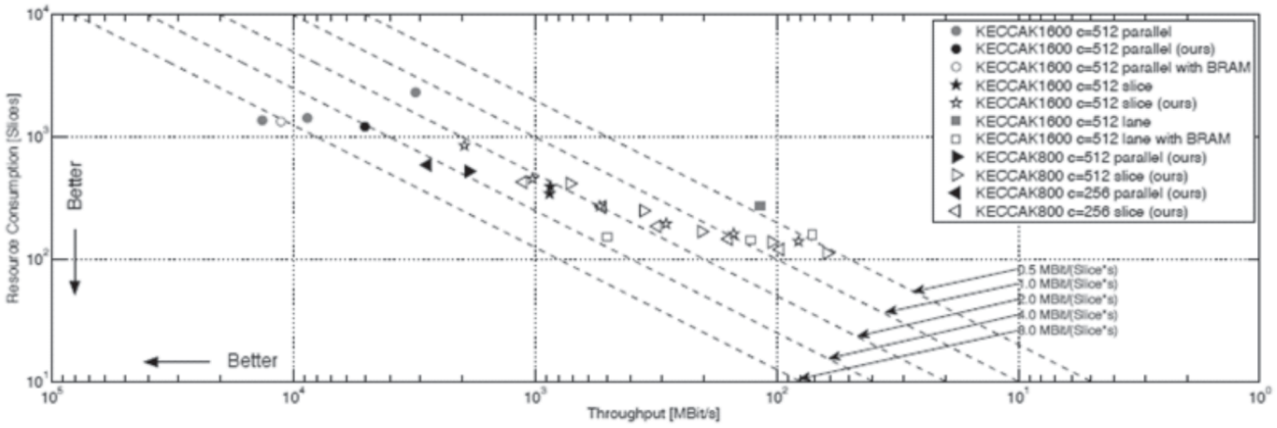


Fig. 4. Replacement Relationship of the operational power-level for different implementations Keccak-f [1600] and Keccak-f [800].

Architecture of the cut at the presented axis works better than the projects introduced in the past in terms of exploitation from sources at the width of the same *data path*, d=200. Our optimization type with d=25 is the smallest project reported to date, event compared to the projects which use BRAM. For instance, the results reported in (Bertoni *et al*., 2011) acts faster than our small project, but it uses BRAM3, which adds a considerable overhead which is not covered with the pieces. In left side of figure 4, results from implementation of parallel Keccak can be observed. Our parallel project refers to one of the slowest parallel projects due to message absorption overload. However, this project is smallest projects. For Keccak-*f* [800] implementations, we generalized analysis and put the projects with different capabilities. Capacity and capability not just affect security of Keccak algorithm, but also affects resource consumption, and thus the efficiency of Architecture. The more security at project, the implementation becomes smaller, but on the other hand operational power diminishes. Both these effects are caused by smaller rate. If the capacity increases, the rate decreases and thus the function decrease. On the other hand, if the rate decreases, the control logic decreases due to smaller counters for absorption and compression stages. Thus, the source consumption is less for high capacity type.

**Light implementations**

Light hash functions are used in RFID components and are latent tools for the goals to set credit. Large status Keccak-*f* [*b*] is a big barrier in use of it in these applications. Thus, we examined our proposed architecture features for a variety of Keccak-*f* [400] and Keccak-*f* [200], which are the lightweight versions of the winner SHA-3. Results from projects Keccak-*f* [400] and Keccak-*f* [200] are same as the implementations with further security in terms of scalability, but in general they are smaller than heavyweight types (Fig. 5).

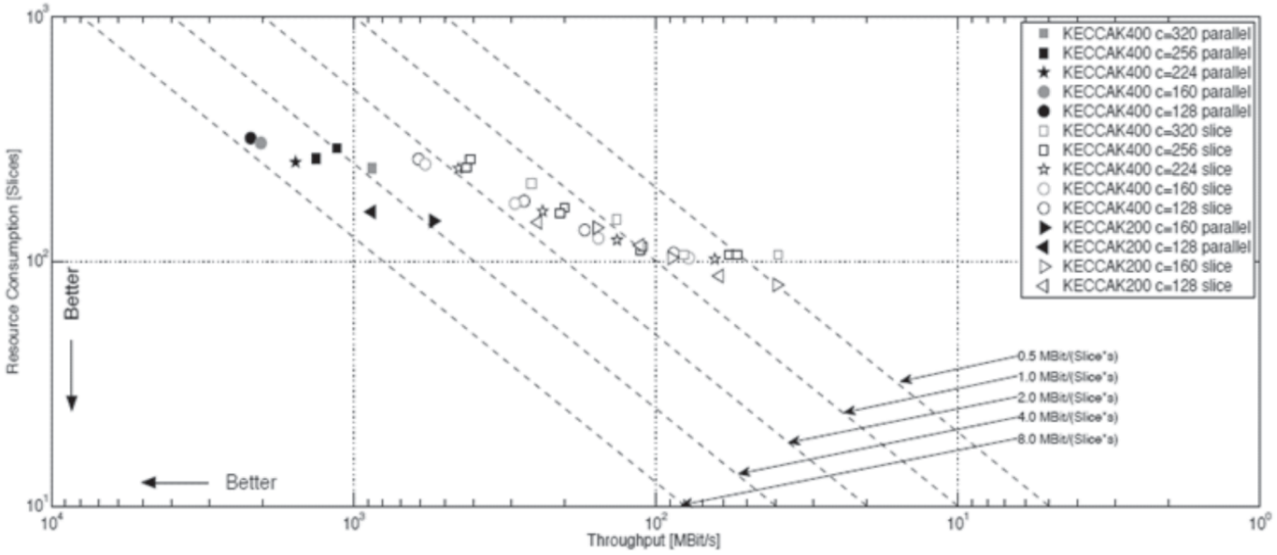


Fig. 5. level-operational power replacement for different implementations Keccak-f [400] and Keccak-f [200].

It seems that architecture of cut at the proposed axis is scalable for lightweight implementations of Keccak algorithm. Total operational power is determined via width of *datapath* (d) and rate(r). Parameter (d) determines number of clock cycles and affects Clock *frequency. However, it seems that changes in clock frequency have not statistical relation for the versions with the same d-value, because these changes due to different settings are greater for Xilinx tool chain. Rate(r) determines number of processed input bits at each hour cycle. it should consider that we have focused on long hash messages for evaluation, thus additional compression steps are required for shorter messages for projects with* $r < n$ feature.

**Serial port**

Standard serial port refers to one of the most common external communication of computers till last few years, having the ability to connect various devices such as modems, scanners, printers, etc., respectively. However, due to the need for higher speeds in current use, other ports are being replaced with this port. For example, parallel port which can transmit 8 lines of communication simultaneously and/or new serial ports (USB and USB2) which have the ability to transmit super-fast information are suitable alternatives to this port. However, use of this port has not been just canceled, but also use of this port outperforms other ports in some cases due to technical and economic reasons. Practically, connecting to the serial port is much difficult than parallel port. In most cases, a device which connects to serial port requires converting the serial transmission to parallel which this is made via UART.

**UART board**

UART is the abbreviated form of Univeral Asynchronous Reciever Transmitter which means Comprehensive asynchronous transmitter and receiver. UART implies converting the bits received from serial port to the parallel data. As known, data in serial port are received bits in a row; further, as known, a bit has no meaning for computer and Bytes are just meaningful. UART buffers input bytes from serial port and transmits them for processing when some of them reach to a byte.

**Interface circuit**

In large systems, UART is an auxiliary circuit to transmit serial data. Checking major hash of system are periodic states which retrieve and receive the words. The receiver interface circuit has two major functions: 1- the first provides a mechanism for the signal to have access to a new word and make retrieve to avoid the received word at multiple times; 2-the second provides a buffer space between receiver and major space. There are three major projects:

**Conclusion**

In this article, two-stage communication architecture of new SHA-3 algorithm was introduced. A special attempt has been made and several other design methods have been examined to deduce efficient FPGA implementations in terms of operational power and level/operational power ratio which gained considerable improvements compared to FPGA implementations. In this research, firstly SHA-3 algorithm has been implemented via software Matlab and quartus and then the results of tests were observed on Ultra board which all the responses have been found consistent with major version in Keccak site.

**References**

Athanasiou, George S., Makkas, George-Paris., Theodoridis, G. (2014): High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm. 2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP). **DOI:** 10.1109/ISCCSP.2014.6877931.

Bertoni, G., Daemen, j., Peeters, M. and Van Assche, G. (2007): "Sponge Functions," ECRYPT Hash Workshop, 1-22. http://sponge.noekeon.org/SpongeFunctions.pdf Ethan Heilman to hash-forum@nist.gov, October 5, 2012, Hash Forum.

Bertoni, G., Daemen, j., Peeters, M. and Van Assche, G. (2011): "Keccak Specifications," Submission to NIST (Round 3), 2011. Federal Information Processing Standards Publication 180-4, Secure Hash Standard (SHS), Information Technology Laboratory, National Institute of Standards and Technology, http://csrc.nist.gov/groups/ST/hash/sha- 3/Round3/submissions_rnd3.html.

Bertoni, G., Daemen, j., Peeters, M. and Van Assche, G. (2012): Keccak implementation overview version 3.2, 2012. http://keccak.noekeon.org/Keccak-implementation-3.2.pdf.

Bertoni, G., Daemen, j., Peeters, M. and Van Assche, G. (2014): "SAKURA: a flexible coding for tree hashing. ACNS 2014: Applied Cryptography and Network Security pp 217-234. doi: 10.1007/978-3-319-07536-5_14https://link.springer.com/conference/acns.

Chang, S.J., Perlner, R., Burr, W.E., Turan, M.S., Kelsey, J.M., Paul, S. and Bassham, L.E. (2012): Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition 2007. National Institute of Standards and Technology. U.S. Department of Commerce. 1-80. http://dx.doi.org/10.6028/NIST.IR.7896.

Merkle, R.C. (1987): "A digital signature based on a conventional encryption function," Advances in Cryptology -CRYPTO '87, A Conference on the Theory Applications of Cryptographic Techniques, Santa Barbara, California, USA, 369-378.

NIST Computer Security Division (CSD). "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions" (PDF). NIST (National Institute of Standards and Technology). U.S. Department of Commerce. 1-37. http://dx.doi.org/10.6028/NIST.FIPS.202.

NIST Releases SHA-3 Cryptographic Hash Standard. http://www.nist.gov/itl/csd/201508_sha3.cfm. Accessed: 2015-11-23.

NIST Cryptographic Algorithm Validation Program (CAVP), http://csrc.nist.gov/groups/STM/cavp/

Sravani, M.M. and Pallavi, C.H. (2015): Design of Compact Implementation of SHA- 3(512) on FPGA. International Research Journal of Engineering and Technology (IRJET);. **02(02**), 41-46.