Efficient FPGA Implementation of the SHA-3 Hash Function

Magnus Sundal and Ricardo Chaves
INESC-ID, Instituto Superior Técnico, Universidade Lisboa
Email: mvsundal@outlook.com, Ricardo.Chaves@inesc-id.pt

***Abstract*—In this paper, three different approaches are considered for FPGA based implementations of the SHA-3 hash functions. While the performance of proposed unfolded and pipelined structures just match the state of the art, the dependencies of the structures which are folded slice-wise allow to further improve the efficiency of the existing state of the art. By solving the intra-round dependencies caused by the θ step-mapping with the pre-computation of values and by improving the memory mapping, it is possible to reduce the required area resources and obtain shorter datapath. This allows to achieve an efficiency improvement of at least 50% in regard to the state of art. This work also provides an overview of the achievable performance and cost for different folding/unrolling options.**

I. Introduction

The SHA-3 standard was specified in 2012, with the conclusion of the public SHA-3 competition from the National Institute of Security and Technology (NIST) [1], where a selected sub-set of the Keccak sponge function was integrated. This established the potentially more secure hashing standard, given both the general life-expectancy of its predecessor, SHA-2, and existing attacks [2].

Other than security, efficiency is major requirement when implementing cryptographic algorithms. SHA-3 is seen to exceed the performance of its predecessor [3] and multiple solutions with increasing performance have been proposed, each with its own particularities. While software allows for a high flexibility and much shorter time to market, hardware implementations allow achieving significantly higher throughputs and overall better performance metrics. One alternative for exploiting the benefits of both of these technologies are FPGAs. These are fine-grained re-programmable logic devices with a high number of logic cells and additional resources which provide high parallel processing power. While ASICs allow to better performances results and lower mass production costs, FPGAs allow for a lower time to marked. FPGAs also allow for the possibility to update the hardware implementation and to achieve a better performance than software implementations. An additional advantage with FPGAs is the ease of prototyping and evaluation. Given this, FPGAs are herein chosen as the implementation technology, in particular the Virtex-5 family from Xilinx, being the most used FPGA family used in the state-of-the-art.

The existing literature is classified by the adoption of the structural optimization techniques: folding, pipelining and unrolling. Solutions with various approaches to folding represent the most compact SHA-3 structures while larger existing solutions are unfolded and represent structures with a high throughput. Solutions with further increases in throughput utilize pipelining and the number of required clock cycles (latency) is reduced in one example by the use of unrolling.

Herein three structures are proposed and developed considering a straight-forward solution, designated as basic, a pipelined structure, and a compact structure which is folded 4 times. The basic structure is presented mostly as a reference point for the comparison with the remaining proposed structures and the existing state-of-the-art. However, the main contribution is in a proposed compact structure, derived from the basic structure by adopting the technique of folding. An additional pipelined structure is also considered as a parallel processing option.

In regard to folded structures, the novel approach proposed allows to solve or mitigate the data dependency issues, by pre-processing the dependent state values in the θ computation. The considered approach allows to improve the Throughput per Area (T/A) efficiency in regard to the best existing state-of-the-art by 50%. As expected, the folded structures allow for more compact designs at a cost of lower throughputs but also relatively lower efficiency, when compared with the straight-forward unfolded structures. A wrapper component with IO-buffering is also included and discussed, which in most of the existing literature is not considered or not included in the presented assessments.

The paper is organized as follows. Section 2 gives a brief presentation of the underlying Keccak algorithm of the SHA-3 hash function. In section 3, the state-of-the-art is presented and analyzed with regard to the most relevant existing solutions. Section 4 describes the main contributions of this work while the experimental evaluation of the presented structures is presented and compared with the existing state-of-the-art in Section 5. The paper is concluded with some final remarks in Section 6.

II. The SHA-3 Algorithm

The algorithm is a family of sponge functions called Keccak which is based on the sponge construction [4], as depicted in Figure 1. The sponge construction provides a generalized security proof and involves the iteration of an underlying sponge function along with the absorption of blocks, constituting a padded input message, and truncation of the output digest.

The state, which is processed by the sponge function, is composed of the outer state, with r bits, and the inner

2159-3477/17 $31.00 © 2017 IEEE
DOI 10.1109/ISVLSI.2017.24

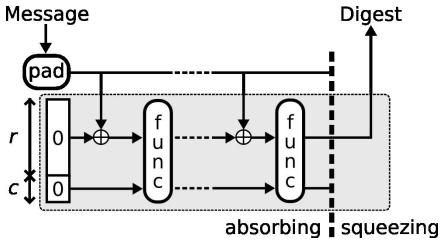

Fig. 1: SHA-3 sponge construction.

state, with c bits, initiated to zero. The outer state is where the message is absorbed and the digest is extracted after processing. Its size is referred to as the block size. The inner state is empty at the start of processing and its size is the main parameter related with the security proof of SHA-3. When the input message and the output digest are smaller than the outer state, with r bits, the sponge function only needs to be processed once. The state (composed of the outer and inner state) is represented as a 3-dimensional block, as depicted in Figure 2.

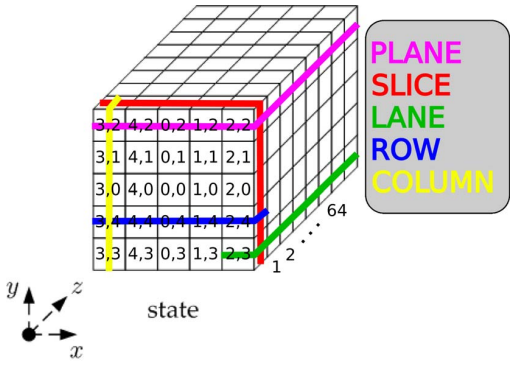

Fig. 2: The state in conventional 3-dimensional representation.

There are currently four sub-versions of SHA-3 supported by NIST: SHA3-224, -256, -384 and -512, where the numbers indicate the length of the digest. They differ in the size-ratio between the inner and outer state (block size), but the complete state is always 1600 bits (5x5x64). The sponge function consists of 24 rounds where the state is processed and updated. According to the sponge function specification, the absorbing phase is iterated if the message is large, however, the squeezing is only performed once as all of the four SHA-3 sub-versions produce digests smaller than the block size/outer state.

The logic providing the processing of each of the 24 rounds is referred to as the round function and is made up of a 5-step sequence of transformations and permutations. These are further referred to as step-mappings and are largely based on XORs, rotations as well as a few NOT and AND operators, as depicted in Figure 3.

Theta (θ) provides diffusion on to two adjacent columns. Rho (ρ) permutates each lane internally by a rotation offset given by a 5x5 matrix r. Pi (π) permutates the lanes with

θ-Theta (A)

$$B[x,z] = A[x,0,z] \oplus A[x,1,z] \oplus \ldots \oplus A[x,4,z]$$
$$C[x,z] = B[x-1,z] \oplus B[x+1,z-1]$$
$$D[x,y,z] = A[x,y,z] \oplus C[x,z]$$

ρ-Rho (D,r)

$$E[x,y,z+r(x,y)] = D[x,y,z]$$

π-Pi (E)

$$F[y,2x+3y,z] = E[x,y,z]$$

χ-Chi (F)

$$G[x,y,z] = F[x,y,z] \oplus ((\text{NOT } F[x+1,y,z]) \text{ AND } F[x+2,y,z])$$

ι-Iota (G,RC)

$$H[0,0,z] = G[0,0,z] \oplus RC[z]$$

Fig. 3: The five steps of the round function

respect to each other in the x and y positions, changing rows into columns. Chi (χ) provides non-linearity, acting on each row and Iota (ι) XORs the center lane with round-specific constants.

The input blocks are XORed with the state lane-wise, starting at the center. A message block fills up the 3-dimensional state in the following sequence with [x,y] coordinates: [1-4,0],[1-4,1], etc. The inner state ends at the highest coordinate, i.e. from [4,4], and fills up the lower coordinates till the end of the outer state. The padding of messages is such that a '1' bit is appended after the LSB of the last byte of the message and finally a '1' bit is appended to the MSB of the last byte of the block. The NIST API specifies a byte-wise big-endian bit-order while the internal Keccak ordering is little-endian, thus a bit-reordering is required [5].

III. State-of-the-art

Several structures for the computation of SHA-3 have already been proposed in the existing literature. These structure vary mainly in the level of folding and the number of pipeline stages. While unfolded structures allows to obtain higher throughputs, folded structures require less resources at a cost of lower throughputs. It should be noted that due to the dependencies within the round caused by the step-mappings, the complexity is increased as the folding technique is adopted. For example, folding a state along the lanes will split up the ρ step-mapping which rotates each lane. As such, additional logic and possibly additional clock cycles must be used to solve the dependencies. This results in lower T/A efficiency metrics can complicate further optimization, e.g. by the use of pipelines. Folded structures are distinguished by the orientation of the folding and the degree in which they are folded, i.e. the folding factor (FF). The relevant existing

solutions are either folded lane-wise or slice-wise and the maximum folding factor is respectively 64 and 25, denoting the total number of lanes and slices of the state.

The first folded structure was proposed by Bertoni *et al.* in the Keccak Implementation Overview [5], with FF=25 and the state stored on an embedded memory. The inconvenience of the lane-wise folding is demonstrated by the high latency of this solution with 215 clock cycles per round. The θ step-mapping requires that 11 lanes are read from memory before one lane is processed. Improvements to this structure is proposed by Kerckhof *et al.* [6] where the latency is reduced to 88 clock cycles per round by more efficient control logic which improves the instruction parallelism. San & At [7] propose further improvements to the latency and frequency by pipeline registers so that two rounds are completed in 88 clock cycles.

Jungk & Apfelbeck [8] propose the first slice-wise solution, with FF=8 so that 8 slices are processed in parallel. A slice-wise folding allows for the complete round function processing of a fold in one clock cycle, but requires a re-scheduling of the round function. Therefore, the latency is a multiple of the folding factor and thus lower than for lane-wise folding. The ρ and θ step-mappings still require additional logic for provision of the necessary input bits. As proposed in [9], [10], the re-scheduling of the round function allows for incorporating the ρ step-mapping into addressing of the state in memory. This implies the increase of the number of rounds, from 24 to 25, as the step-mapping sequence is altered. This folding approach is a highly flexible solution with respect to the folding factor, i.e. the number of slices being processed in parallel can easily be modified with minor changes to the control logic.

Winderickx *et al.* [11] propose a slice-wise structure with FF=64, so that 1 slice of 25 bits is processed in parallel. In this approach, the state is stored with shift register lookup tables (SRLs) for easy processing of the ρ step-mapping.

Akin *et al.* [12] present the first pipelined unfolded structure with 5 pipeline registers incorporated in the round function. This optimization achieves a very high frequency, but at a cost of high area requirements, resulting in a low T/A efficiency. Additional variations of pipelined structures have also been explored [13], [14], [15] and results suggest that one or two pipeline stages in the round function is the optimal trade-off between increased throughput and area.

As is pointed out by Ioannou *et al.* [16], pipelining is not relevant for applications where single larger messages need to be processed. The specifications of the Keccak sponge function dictates that the state containing a block must be processed by the 24 rounds before absorbing the next block of the same message. Two blocks of the same message will therefore not fill a pipeline. Regarding multiple small messages, Ioannou *et al.* propose an unrolled structure which incorporates an external pipeline.

IV. Proposed solution

As described above, several design options can be considered when implementing the SHA-3 algorithm in hardware. Herein, 3 different implementations are presented considering distinct structural options, namely; a straight-forward implementation, designated as a basic structure; a pipelined structure computing two message blocks in parallel; and a folded structure, the most novel structure with a 4 level folding, round re-scheduling and dependency resolution by pre-computing partial θ values.

A complete SHA-3 structure can be divided into 3 main components: the wrapper, the state and the round function. The wrapper component includes the interface with the SHA-3 core, the byte reordering according to the sequence needed by the scheduling of operations, the control logic and the IO-buffering. The state contains the registers for storing the 1600 bits of the state and the round function contains the 5 step-mappings needed to compute SHA-3. Input message padding is assumed to be already performed, for example in software, as considered in most of the existing state-of-the-art.

The following sections describe the 3 proposed SHA-3 structures and a few of the implementation decisions involved.

A. Unfolded structures

The basic structure contains one instance of the 3 aforementioned components, computing one round per clock cycle. A multiplexer controlled by a round counter determines whether the state is updated with the result of the round function or from the input block provided by the wrapper during the absorption phase. The separate IO-buffer allows to output the resulting digest and absorbing a new message in parallel with the processing of a message, as considered in [5]. The resulting structure is identical to the straight-forward ones considered in [3], [16].

The main goal of this structure is to serve as a reference point for the relative evaluation with the state-of-the-art.

The pipelined structure is modified from the basic structure with an internal pipeline register which separates the θ step-mapping from the rest of the round function. This is realized by implementing a synchronized input to the ρ step-mapping. Two messages must be prepared by the wrapper in parallel with two messages being processed in the round function. The latency is increased to 48 clock cycles, however, the clock cycles per block remains 24.

B. Folded structure

Towards a more compact SHA-3 structure, folding of the round computation can be considered. In this case each round is computed over multiple clock cycles, depending on the folding factor (FF). Targeting a throughput of about 1GBs and a moderate folding factor, a FF=4 is proposed. Given the results and approaches considered in the state-of-the-art, a slice-wise folding structure is considered, resulting in the processing of 16 slices in each iteration. However, special care must be taken regarding data dependencies in the θ and ρ step-mappings, in order to provide the necessary input values for the computation of the slices on each iteration.

The ρ step-mapping dependencies can be solved by re-scheduling the round computation in a similar manner as proposed in [8]. With this, the re-scheduled computation

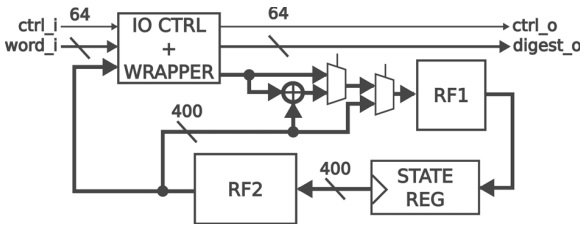

Fig. 4: Simplified schematic of the folded structure.

becomes $R_{resc} = \theta \circ \iota \circ \chi \circ \pi \circ \rho$. Thus, it is possible to split the round function into RF1, containing θ, and RF2, containing π,χ and ι. The ρ step-mapping can be embedded in the state memory. The selection between the input block, feed in the first round to RF1, and the output of RF2, feed in the following iterations, is done using a multiplexer. The resulting structure is illustrated in Figure 4.

When larger messages composed by multiple input blocks are considered, the output of RF2 of round 24 must be XORed with the new messages block before entering RF1. When processing a new message, the input block is directly fed to RF1. This is accomplished by a multiplexer selecting the input block or the XOR of this data with the output of RF2, as illustrated in the center of Figure 4. With this approach, the processing of each block requires 96 clock cycles.

Distributed RAM can be used to store both the IO-buffer (in the wrapper component) and the state. FPGA LUTs are cascaded and treated as 16x64 bit memory blocks. The optimal approach to mapping the state into memory blocks depends on the folding factor and orientation. In the proposed solution, both the state and the IO-buffer are organized in such a way that each memory block contains one lane. Different depths of the memory contain different folds, i.e. the 16 first bits of each lane (z=0-15) are located at address 0, bits z=16-31 at address 1 etc.

The ρ step-mapping is solved by addressing. There are two instances of the state located at different addresses of the memory blocks, i.e. instance one is located at address 0-3 and instance two at address 4-7. The read address corresponds to the sub-round and is therefore incremented by a sub-round counter from 0 to 7. The rotation is performed by adding an offset to the write address of the different memory blocks, however, a mismatching occurs as the rotation of each lane is not aligned with the four folds. Addressing is therefore determined by the future location of the majority of the bits of each lane. With FF=4, < 8 bits will be located at the adjacent fold for each lane. These mismatched bits are packed into the other memory blocks (lanes) with the same addressing. In this way, they can be accessed during the correct future sub-round. Without this compression, additional registers are necessary in order to provide all the dependency bits of each fold.

The processing of a slice in the θ step-mapping depends on the equivalent input slice and the slice with the lower z-coordinate, i.e. θ output fold 1 slice 0 (F1S0') depends on F1S0 and F0S15. This and the equivalent cases with the slices of the subsequent folds are easily solved as the intermediate

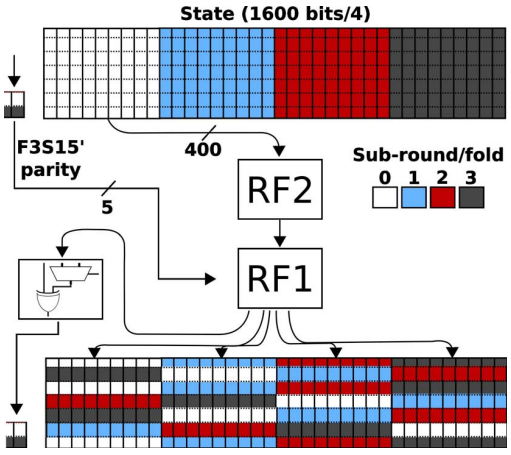

Fig. 5: θ step-mapping dependency solution: F0S0 pre-processing.

FXS15 values are simply stored in a temporary register until the next sub-round. However, F0S0, can not be calculated without access to F3S15, which is not available until the last sub-round. The existing literature [8], [10] has solved this by storing the intermediate value of F0S0 and completing this processing during the last sub-round.

An alternative solution to this is herein proposed, by performing the *F0S0 pre-processing*. In this solution the future value of the F3S15 slice, belonging to round x+1.3, is collected from the output of the round function during the sub-rounds of round x. The above description considers round x and $x+1$ as the xth and x+1th rounds of SHA-3, respectively, and .3 as the 3rd iteration of a round, given that with $FF = 4$ each round requires 4 loop iterations to be computed. With this approach, Slice F3S15 is therefore available in the θ step-mapping in RF1 during processing of F0S0 in the initial sub-round iteration, i.e. round $x + 1.0$. This pre-computation is depicted in Figure 5.

C. Implementation

While the considered structures and proposed solution are independent from the implementation technology, prototypes were implemented in order to properly test and compare with the existing state-of-the-art. For this the Xilinx FPGA Virtex-5 family was considered.

The IO-buffer in the wrapper component is implemented with flip flops as a FIFO for the basic structure. For the pipelined structure, two messages are managed by the wrapper in the IO-buffer and so the depth of distributed RAM can be used. The situation is similar for the folded structure where only parts of the message are accessed in each clock cycle. Regarding memory usage and considering SHA3-512, 9 16x16 single port RAM blocks were deployed, using Distributed RAM Blocks produced by the XIlinx IPCore generator. Multiplexers are used to assign the respective inputs to the correct memory block and to select the correct state bits to the digest value output.

Paper	FPGA	Buffer	PL	UF	FF	Latency (cycles)	f (MHz)	A (slices)	T (Gbps)	T/A (M/s)	T/A (scaled) (M/s)
Ioannou[16]*	V-5	no	1	1	1	24	382	1581	9.17	5.79	5.79
Gaj[3]*	V-5	yes	1	1	1	24	283	1272	12.82	5.37	5.37
Basic	V-5	yes	1	1	1	24	223	**1192**	5.35	**4.49**	4.49
Baldwin[17]	V-5	yes	1	1	1	25	196	1971	8.52	4.32	2.40
Ioannou[16]*	V-5	no	2	2	1	24	**352**	**2652**	16.90	**6.37**	6.37
Pipelined	V-5	yes	2	1	1	48	**273**	**1163**	7.80	**6.06**	6.06
Athan.[15]	V-5	no	2	1	1	48	389	1702	18.70	10.98	5.49
Pereira[13]*	V-5	yes	4	1	1	100	452	3117	7.70		3.34
Akin[12]*	V-4	yes	5	1	1	121	509	4356	22.33	5.13	2.69
Folded	V-5	yes	1	1	4	96	200	**476**	1.20	**2.52**	**2.52**
San & At[7]*	V-5	yes	1	1	25	1062	520	151	0.25	**1.66**	1.66
Jungk[10]	V-5	no	1	1	2	50	144	914	3.13	2.04	**1.33**
Jungk[10]	V-5	no	1	1	4	100	150	**489**	1.63	1.99	**1.30**
Jungk[8]	V-5	no	1	1	8	200	159	393	0.86	2.19	1.17
Jungk[18]	V-5	yes	8	1	64	1665	257	90	167	1.85	0.98
Winder.[11]*	V-5	yes	1	1	64	1730	248	134	0.25	1.16	0.62
Kerckhof[6]*	V-6	yes	1	1	25	2154	250	144	0.07	0.46	0.46

TABLE I: SHA3-512 implementations results. Nomenclature: FF=folding factor, UF=unrolling factor, PL= pipeline stages. M/s=Mbps/slice.
*Source of results not mentioned or from synthesis results.

Distributed RAM is also used for the state register of the folded structure where 25 16x16 simple dual port (SDP) memory blocks store the 4 25x16 bit folds.

The round constants were precomputed and hard-coded in the design and are provided by the wrapper using multiplexers controlled by the round-counter (resulting in a more compact circuitry than computing them on-th-fly with a LFSR). The pre-computed θ intra-round values, used to solve the data dependencies, are provided to RF1 either from the IO-buffer, during round 0, or from the round logic, during the remaining rounds. This selection is performed by another multiplexer controlled by the round counter.

Particular components from the FPGA technology may be used, such as LUT-based FIFOs, BRAMs, or DSPs. However, to present less technology biased results and to better compare with the existing state-of-the-art, these were not considered in the implemented structures.

V. Result analysis

All proposed structures have been developed considering a Xilinx xc5vlx50t Virtex-5 FPGA. The presented results were obtained after Place and Route using the Xilinx ISE 14.7 tool. LUT combining has been used to optimize the area requirements.

These results obtained for the developed structures and the results presented in the related literature are listed in Table I, sorted by design structures and efficiency.

Since the various results in the state-of-the-art are presented for different SHA-3 sub-versions (i.e. SHA3-256, SHA3-...), results for normalized/scaled efficiency values are also presented in the last column of Table I, (*T/A (scaled)*). Depending on the SHA-3 sub-versions, the input message is inputted in different block sizes. For example, in SHA3-256 the message is inputted in 1088-bit blocks while for SHA3-512 the message is inputted in 576-bit blocks. Since in all cases the state is always 1600-bit and processed over 24 round, the size of the input block, i.e. the SHA-3 version, directly impacts the throughput and respective T/A efficiency. As such, this efficiency scaling is based on (1), where the block size, r, is adjusted to 576 bits. f is the frequency, L is the latency denoting the required number of clock cycles for processing of a block, and A is the area.

$$E = \frac{T}{A} = \frac{r \cdot f}{L \cdot A} \tag{1}$$

As such the scaling is done by dividing the original T/A value by the size of the input block (which for SHA3-256 is 1088 bits) and multiplying by 576, the size of SHA3-512 input block, resulting in:

$$Scaled\ E = E \times \frac{SHA3\ Block\ Size}{576} \tag{2}$$

Other than this, the other differences between SHA-3 sub-version implementations are related with the size of the IO-buffer and the amount of logic used in XORing the input blocks with the state. Since this will always be advantageous for the structures supporting SHA3-512, as is the case of the ones herein presented, it not considered in the presented normalization/scaling.

When comparing with the basic unfolded structures [17], [12], [3], [16] with the **basic** herein proposed, while some differences exist they are not too significant, since they are structurally identical. It is the authors' belief that the existing differences are mainly due to the way the results were obtained and the existence or not of a wrapper interface. Some of the results presented in the state-of-the-art were obtained after synthesis only, presenting better results or using the ATHENa tool as in the case of Homsirikamol *et al.* [3]. It is unclear if

the latter results are for synthesis or post place and route.

The **basic** structure is mostly presented as a base of comparison. It should be noted that [17] includes a padding component and [12] is implemented on a Virtex-4, which should yield a lower efficiency. The cost of the wrapper with IO-buffer for the proposed basic structure is about 160 slices, including the control logic, storing 9x64 bits.

The obtained results for the **pipelined** structure are identical to the ones in the state of art. The structure proposed in [16] presents better results, but does not consider the input and wrapper component. When compared with identical folding structure proposed in [15], with no unfolding and one level of pipeline the obtained efficiency is 10% better, while including the wrapper logic.

As already noted, pipelined structures are only relevant in situations where hashing messages are equal to or smaller than the block size. For hashing of larger messages, solutions similar to the basic structure represent roughly the highest obtainable efficiency for SHA-3.

The **folded** structure considers a folding factor of 4, meaning that 25% of the state is processed in each clock cycle, presenting a trade-off between lower area requirements and throughput. When compared with the best state-of-the-art considering slice-wise folding, proposed in [10], the obtained results suggest that for the same folding level faster designs can be devised with identical area costs, resulting in an improved efficiency, up to 89%. This improvement is achieved by a more efficient structure which only contains the absolute necessary registers for a proper functionality and a memory mapping that reduces the required RAM. Nevertheless, the main improvement is achieved by the proposed pre-computation of θ, solving the intra-round dependencies more efficiently.

When compared with the lane-wise solution folding proposed in [7], the results suggest an efficiency improvement of 51%. This suggests that lane-folding may not be the best design approach, even when using BRAM, as is the case in [7]. It is not clear how the inter-round dependencies are solved in [7] nor how the authors were able to achieve such a high operating frequency.

To allow and facilitate the comparison with the proposed solutions, the implemented structures (using VHDL) are made available at: *http://sips.inesc-id.pt/∼rjfc/cores/SHA3/* .

VI. Conclusion

In this paper FPGA based SHA-3 solutions and techniques are considered and 3 structures proposed and evaluated. This work considers a straightforward, a pipelined, and a folded structure. While the first two do not present particular challenges or contributions, the folded structure allowed to present some innovations towards improved computational efficiency. The achieved efficiency improvements, in the order 50% to 90%, were achieved by considering a slice-wise folding approach with an adequate memory mapping and θ dependency pre-computation. This resulted in smaller design and lower critical path, and consequently in a higher efficiency.

Acknowledgments

"This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013."

References

[1] National Institute of Standards and Technology. FIPS PUB 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015.

[2] Henri Gilbert and Helena Handschuh. Security analysis of SHA-256 and sisters. In *International Workshop on Selected Areas in Cryptography*, pages 175–193. Springer, 2003.

[3] Ekawat Homsirikamol, Marcin Rogawski, and Kris Gaj. Comparing hardware performance of round 3 SHA-3 candidates using multiple hardware architectures in Xilinx and Altera FPGAs. In *Ecrypt II Hash Workshop*, pages 19–20, 2011.

[4] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 3:30, 2009.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keccak implementation overview. *Report, STMicroelectronics, Antwerp, Belgium*, 2012.

[6] 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. In *International Conference on Smart Card Research and Advanced Applications*, pages 217–233. Springer, 2011.

[7] I. San and N. At. Compact Keccak Hardware Architecture for Data Integrity and Authenticaion on FPGAs. *Information Security Journal: A Global Perspective, 21*, pages 231–242, August 2012.

[8] Bernhard Jungk and Jurgen Apfelbeck. Area-efficient FPGA implementations of the SHA-3 finalists. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 235–241. IEEE, 2011.

[9] Bernhard Jungk. *FPGA-based evaluation of cryptographic algorithms*. PhD thesis, Johann Wolfgang Goethe-Universität, February 2016.

[10] Bernhard Jungk and Marc Stöttinger. Among slow dwarfs and fast giants: A systematic design space exploration of KECCAK. In *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*, pages 1–8. IEEE, 2013.

[11] J. Winderickx, J. Daemen, and N. Mentens. "Exploring the Use of Shift Register Lookup Tables for Keccak Implementations on Xilinx FPGAs". *26th International Conference on Field-Programmable Logic and Applications, Lausanne*, 2016.

[12] O. C. Ulusel A. Akin, A. Aysu and E. Savas. Efficient Hardware Implementations of High Throughput SHA-3 Candidates Keccak, Luffa, Blue Midnight WIsh for Single- and Multi-Message Hashing. *SINCONF, Taganrog, Russia*, pages 168–177, September 2010.

[13] Fábio Dacêncio Pereira, Edward David Moreno Ordonez, Ivan Daun Sakai, and A Mariano de Souza. Exploiting parallelism on keccak: Fpga and gpu comparison. *Parallel & Cloud Computing*, 2(1):1–6, 2013.

[14] Yusuke Ayuzawa, Naoki Fujieda, and Shuichi Ichikawa. Design trade-offs in SHA-3 multi-message hashing on FPGAs. In *TENCON 2014-2014 IEEE Region 10 Conference*, pages 1–5. IEEE, 2014.

[15] George S Athanasiou, George-Paris Makkas, and Georgios Theodoridis. High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm. In *Communications, Control and Signal Processing (ISCCSP), 2014 6th International Symposium on*, pages 538–541. IEEE, 2014.

[16] Lenos Ioannou, Harris E Michail, and Artemios G Voyiatzis. High performance pipelined FPGA implementation of the SHA-3 hash algorithm. In *2015 4th Mediterranean Conference on Embedded Computing (MECO)*, pages 68–71. IEEE, 2015.

[17] Brian Baldwin, Andrew Byrne, Liang Lu, Mark Hamilton, Neil Hanley, Maire O'Neill, and William P Marnane. FPGA implementations of the round two SHA-3 candidates. In *2010 International Conference on Field Programmable Logic and Applications*, pages 400–407. IEEE, 2010.

[18] Bernhard Jungk and Marc Stöttinger. Hobbit - Smaller but faster than a dwarf: Revisiting lightweight SHA-3 FPGA implementations. In *ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on*, pages 1–7. IEEE, 2016.