Efficient FPGA Implementation of the SHA-3 Hash Function

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

Abstract

This thesis presents the proposal of improved structures, supported on FPGAs, for the new cryptographic hash function standard, SHA-3, with a special focus on efficiency. Cryptographic hash functions are an essential part of modern cryptography and are used extensively in applications requiring message and user authentication and digital signatures. SHA-3 represents the next generation of cryptographic hash functions and is forecast to assume the position of its predecessor, SHA-2, which is one of the most prevalent standards today. Hash functions are well suited to be implemented in hardware as co-processors, performing large-scale hashing of messages such as Ethernet packages.

The goal of this project has been to devise structures and implementation approaches which are able to improve the existing state-of-the-art with respect to efficiency, where efficiency is defined as the achievable throughput per required area.

While several structures and optimizations have already been proposed in the existing state-of-the-art, these tend to be somewhat inconsistent regarding the applications and expected messages for the hash function and the targeted metrics. Herein, several solutions are proposed considering existing and novel optimization techniques, while also proposing an evaluatoin model to better evaluate the possible structural options and to define clear upper bounds to the achievable results.

1. Introduction

Hash functions are an essential part of modern cryptography in integrity and authentication applications. In 2007, The National Institute of Security and Technology (NIST) concluded that it was in due time to determine a successor to the SHA-2 hash function standard [1]. This decision was based on general life-expectancy as well as recently published papers proving a reduction in its strength. A public competition which was initiated in 2007 and ended in 2012 after multiple elimination rounds, determined that a subset of the Keccak sponge function family was the optimal candidate for the new standard titled SHA-3.

Efficiency is key in the implementation of cryptographic algorithms and since 2007, various implementations in both software [2] and hardware [16] have emerged in a range of performances for the SHA-3 hash function, exceeding its predecessors. Hardware co-processors such as FPGAs and ASICs are advantageous for non-general tasks such as in cryptography and offer high parallel processing power compared to general purpose CPUs. Another advantage of hardware implementations when comparing with software is decreased accessibility, which is a great asset concerning security. While software has a short development path it usually runs in shared memory space on top of an operating system, ensuing much room for vulnerabilities.

The scope of this paper covers efficient FPGA implementations of SHA-3. It is imperative with a thorough understanding of the existing literature to be able to advance in the technical field. An argument for the choice of technology has been that the majority of the state-of-the-art falls within the area of FPGA implementations. For comparison reasons, ASIC implementations can be less convenient because of the many variables caused by a diversity in technologies and approaches. The challenge of FPGA implementations is to achieve a high frequency. As the size of a design increases, the delay caused by routing can force the system clock to operate at a much lower frequency than what is supported by the FPGA model. A modern FPGA consists mainly of slices containing Configurable Logic Blocks (CLBs) as well as additional Digital Signal Processing (DSP) slices, Block RAM (BRAM), various clock resources and input and output ports (IOs). Slices are conventionally the main unit for measuring the consumption of area on the FPGA, but the utilization of the additional resources should also be included. As with most other relevant literature, the performance objective is optimal efficiency which is a measure of throughput over area.

The main objective of this work is to improve the state-of-the-art of efficient hardware implementations of the SHA-3 hash function standard. A special focus is made on FPGA implementations and the goal is to arrive at a proposed solution which exceeds the existing literature in efficiency. For the proposed solution to be efficient, it must achieve optimal throughput for rapid processing of messages, while requiring minimal area for implementation. A secondary goal is to define the upper bound efficiency of a SHA-3 FPGA implementation through theoretical models based on considerations of realistic conditions.

The paper is organized as follows. In Section 2, the SHA-3 hash function and the underlying Keccak algorithm will be briefly presented [3]. Subsequently in Section 3, the state-of-the-art is presented and analyzed with regard

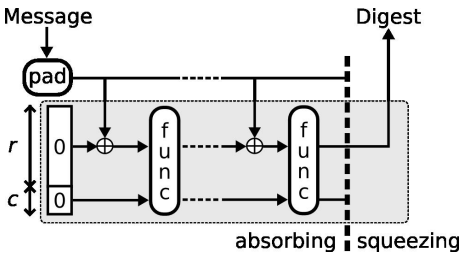

Figure 1: SHA-3 sponge construction.

to the most relevant implementations. The proposed solutions are presented in Section 4, including the description of the research and analysis beyond the existing state-of-the-art which have shaped the resulting structures. Section 5 contains the implementation details of the several imposed SHA-3 structures and the specific factors which have been considered for each individual case. The experimental evaluation of the presented structures are given in Section 6, and compared with both the existing state-of-the-art and the theoretical models and considerations discussed earlier in the paper. Finally, a conclusion is given along with the future work.

2. The SHA-3 Algorithm

The algorithm is a family of sponge functions called Keccak which in turn is based on the sponge construction, as depicted in Figure 1. The sponge construction provides a generalized security proof and involves the iteration of an underlying sponge function along with injecting a padded input message with XORs and truncation of the output digest. The data block in which the sponge function acts is called the state and is divided into an outer state - where data is both injected and extracted after processing - and an inner state which is reset to zeros at each new message. The functionality of the sponge therefore depends on the length of the input message. The iteration takes place in the two phases of the sponge, the absorbing and the squeezing phase respectively - if the input message and the output digest is larger than the outer state. Otherwise, the underlying sponge function is only processed once.

The state is presented as a 3-dimensional block for the benefit of easy apprehension of the sponge function operation, as illustrated in Figure 2. Words constituting the padded input message fills the state lane-wise starting at the center. The inner state is therefore located at the highest coordinates, i.e. from 4,4 to 4,3 and downwards. The size of the outer state where the message is injected is conventionally referred to as the block size. The size of the inner state is the main parameter influencing the security proof of the sub-versions of SHA-3 and the digest is roughly half this size. There are currently four sub-versions of SHA-3 supported by NIST, as seen in Table 1. They differ in the size-ratio between the inner state and block size, but the total state is always 1600 bits (5x5x64).

The underlying function consists of 24 rounds of a five-step sequence of transformations and permutations called the round function. These are largely based on XORs, rotations as well as a few NOT and AND operators:

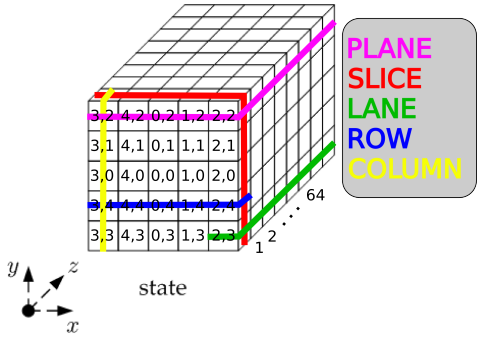

Figure 2: The state.

Version	Block size	Inner state	Digest
SHA3-224	1152	448	224
SHA3-256	1088	512	256
SHA3-384	832	768	384
SHA3-512	576	1024	512

Table 1: The four sub-versions of SHA-3.

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)

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

Code 1: Pseudo-code of the five steps of the round function.

Theta provides diffusion on to two adjacent columns. Rho permutates each lane internally by a rotation offset given by a 5x5 matrix r. Pi permutates the lanes with respect to each other in the x and y positions. Chi provides non-linearity, acting on each row and Iota differentiates each round by XORing the center lane with round constants. The round constants can either be generated by a 7-bit linear-feedback-shift-registers (LFSRs) or be pre-generated and stored as an array.

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 0x80 is appended to the last byte of the block. The NIST API specifies the byte-ordering as little-endian and

bit-ordering as big-endian so that the MSB of each byte is located at the lower address.

3. State of the art

The existing hardware implementations can be grouped into two classes: unfolded high-speed structures and folded compact structures. The former contains simple unfolded architectures with the highest performance related with efficiency. These designs have maximum internal data path with registers usually based on flip flops. The latter contains the more complex designs with minimal footprint. Block or distributed RAM is usually utilized in these implementations and the width of the internal data path is minimized. The compact designs are usually more complex as they inhibit elegant instruction cycles and control units in the pursuit of maximum concurrency. Still, they tend to be too conservative in their area utilization to obtain a decent throughput and are for that reason inferior in efficiency compared to the high-speed designs. A brief chronological introduction of these designs are given below, starting with the high-speed implementations.

Strömbergson [4] provided valuable feedback in 2008 during the early stages of the SHA-3 competition, reviewing the proposed VHDL code for FPGA implementations by the Keccak team (Bertoni *et al.*) [17]. Specifically, he discovered early problems and potential improvements in the early design which have since been corrected. Baldwin *et al.* [5] compared in 2010 the round-two candidates with a general purpose wrapper and included all four subversions of Keccak. The wrapper is the only component which separates this implementation from the previous literature and contains a hardware padding unit. A limited 32-bit input port is the bottleneck of all but the SHA3-512 version. Homsirikamol *et al.* [**?**] presented at the same time a similar comparison of the 256 bit versions of the same candidates and the following year, the 512 bit versions with and without pipelining. Additionally, they explored the potential for unrolled and unfolded implementations and concludes that this is not relevant for Keccak. They obtain a higher frequency than previous implementations and for that also a higher throughput with the non-pipelined design. The pipelined implementations do not perform well as the area increases more than the frequency. The most critical path remains through the round function as it does not contain any pipeline registers. Input and output buffers are implemented with FIFOs in BRAM, thus saving slices. The 2010 paper by Akin *et al.* [6] explores internal pipelining and with that obtains close to maximum frequency on a Virtex-4 FPGA and record high throughput. However, the area consumed is too high to achieve a good efficiency, even with optimal FPGA frequency. Another comparison was performed by Jararweh *et al.* [7] in 2012 before the end of the competition, but without any radical new achievements. Athanasiou *et al.* [8] introduced in 2014 a general SHA-3 design supporting all sub-versions and a pipeline register in the round function. They obtain a high frequency and a relatively low consumption of slices. Ayuzawa *et al.* [9] explores the utilization of DSP slices, specifically on the pipelined design by Akin *et al.* as well as variations of this. They find that certain pipelined designs benefit from utilizing the DSP slices as no delay is added to the most critical path. The following year, Ioannou *et al.* [10] presents an implementation without additional pipeline registers, a smaller area than Athanasiou *et al.* but with a comparable frequency. Additionally, they present a pipelined 2-factor unrolled design which performs even better with respect to efficiency than the straight-forward design.

As with the High-speed design, the initial compact design was first suggested by Bertoni *et al.* with the low-area coprocessor. Instead of processing the full width of the state, this design has a lane-wise architecture so that the internal data path is 64 bits. A very high latency results in a poor throughput for this design. Each round contains 55 bubbles which are cycles where the state is not accessed. Kerckhof *et al.* [11] compares the compact design of the five SHA-3 finalists in 2012. Their implementation of Keccak is an improvement of the low-area coprocessor by Bertoni *et al.*. The internal data path is the same, but the state as well as intermediate values are stored in BRAM where two lanes can be read or written each cycle. The area is reduced by more than 50 % compared to its predecessor. Jungk & Apfelbeck [12] presents a slice-wise architecture where the state is split into 8 pieces of 8 slices, resulting in an internal data path of 200 bits and latency of 200 cycles. The state is stored in distributed RAM, which in the Virtex-5 FPGA can be read asynchronously. The round function in this design is re-scheduled so that the Rho and Pi steps are the last steps in all but the first and last round. This results in 3 different rounds, but the dependency problem of the permutation steps is avoided. San & At [13] improves on the lane-wise architecture by introducing a fine-tuned instruction sequence which allows for high concurrency along a serialized round function. The latency is further reduced by 50 % and along with an optimal frequency this implementation yields a high throughput and efficiency compared to the other compact, folded designs.

3.1. Summary

Compact structures use folding and the most efficient cases among the state-of-the-art regarding compact implementations are folded slice-wise. Folded structures use RAM to implement both the state register and additional data such as the round constants. The most efficient structures are unfolded and use pipeline registers internally in the round function. The highest reported efficiency is seen in a structure which is both unfolded, pipelined, and unrolled so that the implementation includes multiple instances of the round function. The solutions presented in the existing literature are seemingly lacking in the potential for working as stand-alone entities as many are missing a fully functioning wrapping component. The basis for the reported results are also not the same with many obtaining results from synthesis and only a few from Place and Route which are more reliable.

4. Proposed solution

This chapter presents the collective analyses and considerations which have been carried out in relation with this thesis. The existing state-of-the-art is already mature with multiple papers having advanced this scientific field with a variety of approaches. Still, some considerations and approaches are left unexplored. While not all of the concepts here are completely unique, they are found worthwhile presenting as they contribute to give a clear overview of this broad technical field. An exploration of the successful aspects found in the state-of-the-art and the analyses herein performed shaped the results achieved and presented in this chapter.

4.1. Folding

Most folded designs incorporate either a lane-wise or a slice-wise folding scheme. At first glance there seem to be several possibilities of partitioning the state. However, additional logic and clock cycles are required if each fold fails to contain all the necessary bits required to produce the output of a step mapping. These bits are referred to as dependencies and they are a bi-product of critical security features of a cryptographic hash function. The dependencies of each step mapping are rather simple, but many structural constraints arise when they are considered combined. Ultimately, as also concluded in previous literature [14], the optimal folding scheme with respect to the combined dependencies of the round function is a slice-wise architecture. Therefore, other folded configurations are not considered in the further analyses.

There is a clear difference between the performances of the compact folded designs and the straight-forward unfolded designs. It is therefore trivial to observe that while there is a trade-off between the utilization of resources and the latency, the relationship between them is not symmetrical. There is a certainty that the latency will increase by at least a factor equal to the folding-factor, e.g. a minimum latency of 48 an 96 is achieved with a folding-factor of 2 and 4. The resource utilization, however, is more complicated to determine and while round function logic is reduced, additional components such as multiplexers and larger counters are introduced when a folding-scheme is employed.

The formula for calculating the throughput, T, is given in Equation 1, where r is the block-size, which is the size of the message block. The frequency, f, is the number of clock cycles the design can perform in a second and is determined by the critical path between registers. The latency, L, is the amount of clock cycles required for the processing of one message block. The efficiency, E, is obtained by dividing the throughput by the area, A, which is the number of slices consumed by the design, as depicted in Equation 2.

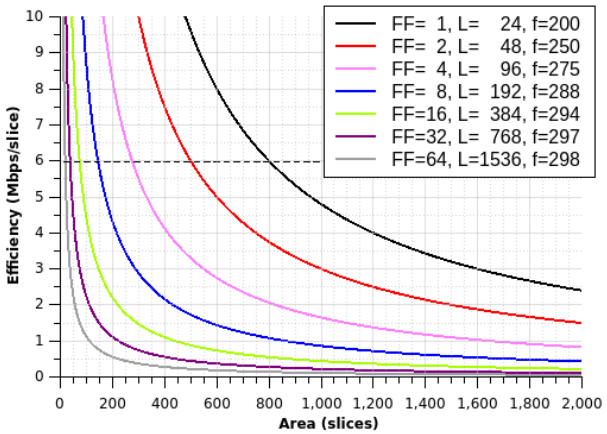

Figure 3: Efficiency model as a function of area for seven different folding factors for SHA3-512.

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

$$E = \frac{r \cdot f}{L \cdot A} \tag{2}$$

When plotting the efficiency as a function of area for various folding schemes, the cost of the increased latency becomes apparent. This plot is illustrated in Figure 3, where 6 different folding approaches are plotted in addition to the unfolded scenario with 24 cycles latency. The frequency is fixed between 200 and 300 MHz with proportionally higher value as the folding factor increases. This is based on intermediate implementation results as well as the reported performances from the existing literature [15]. The block size is set to 576 bits as this corresponds to the most secure and prevalent SHA3-512 version. The dotted line represents roughly the maximum efficiency achieved so far by the existing literature. Evidently, high folding factors impose great constraints in terms of slice utilization. It also suggests that an efficiency comparable to unfolded structures (with FF=1) seem to be unattainable. This is a simplified model which does not take into account possible optimizations such as pipelining and the actual achievable frequencies, which are not easily modeled. Nevertheless, this model helps to roughly illustrate the maximum footprint of a given folding scheme in order to achieve a given efficiency. For example, the obtained results suggests that an unfolded structure should be kept below 1000 slices for an efficiency metric (of about 5 Mbps/slice).

4.2. Pipelining

For pipelining to be an efficient technique, all the registers of the pipeline should be kept as full as possible, minimizing bubbles. At start-up and conclusion, bubbles are inevitable as the pipeline is filled and cleared. Because of the combination of the step-mappings, most SHA-3 structures will have inter-round dependencies so that each round will contain cycles where the pipeline is emptied. Each fold must wait to be processed until all previous folds have updated the state. This causes the efficiency of pipelining to

be drastically reduced as it sets a high requirement for the increase in frequency to compensate the increased latency and area. In order to approximate the cost of pipelining, the relationship between the number of pipeline registers and the throughput and are requirements must be approximated. Equation 3 gives the total area requirement of an optimized structure as a function of the folding factor and the number of internal pipelines. An approximation regarding the area consumption is made so that the total area is divided equally between the three main components: the wrapper, the state and the round function. It is also considered that 10% of the basic structure occupies the same area, independently of the folding and number of pipeline registers (PL). The default for PL is one, as the state register is included in the factor.

$$A = \left(\frac{1}{10} + \frac{9}{10 \cdot FF}\right) \cdot A_0 + \frac{(PL-1) \cdot 9 \cdot A_0}{10 \cdot FF \cdot 3} \qquad (3)$$

Similarly, the latency is increased as a function of FF and PL and the relationship is given by Equation 4.

$$L = FF \cdot L_0 + (PL-1) \qquad (4)$$

The Efficiency Equation 2 shows how the frequency must compensate for the increased latency and area. With the approximation of the area consumption ratio, it is possible to plot the necessary compensating frequency for the various cases of FF and PL. The cost of pipelining is approximated so that each increase in PL adds $3/10$ of the area for $FF = 1$, but this is naturally decreasing as FF is increasing. These cost approximations are based on intermediate results obtained in this work and numbers reported by the existing state-of-the-art. Figure 4 depicts the relationship between the efficiency and the frequency for structures with various combinations of pipelining and folding factors. The right side shows the more efficient structures with no bubbles in the pipeline. Latency is then given by Equation 4. The horizontal green line illustrates the top performance achieved by the state-of-the-art while the vertical green line the maximum frequency which is achievable with a V-5 fpga. The right-side graph suggests that despite inter-round dependencies being met, the quantity of pipeline registers should be kept at a minimum. These theoretical models can possibly fail in accuracy where slices are initially utilized inefficiently. Pipelines can be purely or partly fitted into the already occupied slices if they contain unused flip flops. A more efficient slice utilization is then achieved and the implementation of the first pipeline registers will improve the efficiency significantly.

The general issue with pipelining for unfolded structures is the sponge construction characteristic stating that multi-block-messages must be processed by the sponge function before the next block of the message is merged with the result. For Keccak, the function involves 24 rounds of Keccak-*f*. Gaj *et al.* [16] presents a statistic of the cumulative distribution of packet size for a typical Ethernet node which is a common application for hash functions. The numbers they present show that roughly 50% of the packages processed are below the block size of SHA-3 versions.

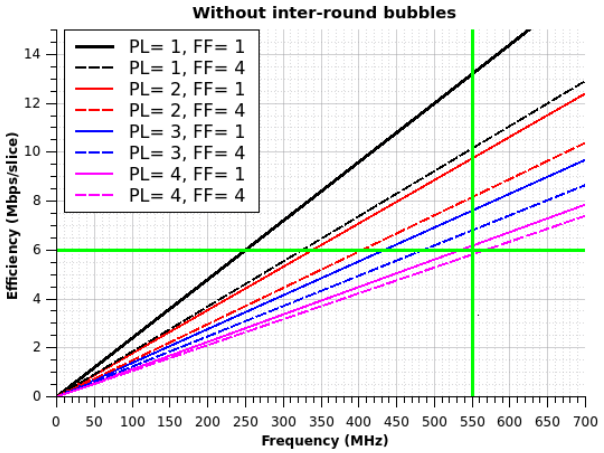

Figure 4: Graph with the efficiency as a function of frequency for various combinations of pipelines and folding factors.

Furthermore, multiple packages are usually available in a queue for processing. From this it can be concluded that both multi-block-messages and small message hashing are realistic scenarios and should be considered, i.e. a SHA-3 design should support multiple small messages as well as larger ones causing an iterated absorbing phase.

4.3. Unrolling

Unrolling a structure can in certain cases increase the efficiency, but is dependent on whether or not pipelining is utilized and how it potentially is implemented. As was concluded by Bertoni *et al.* [17], unrolling without any added pipelines results in a decrease in efficiency. The latency is reduced proportionally with the unrolling factor, however, the area and the critical path is increased. Hence, unrolling is equally irrelevant as pipelining if multi-block-messages are considered and the wrapper is kept a relatively simple component. As Ioannou *et al.* [10] reports, if small messages are considered, then unrolling with external pipelines should attain good results. The area increases less than proportionally with the unrolling factor (UF) as the wrapper is kept in the design while the rest of the logic is multiplied. The structure presented by Ioannou *et al.* is unnecessarily in-efficient. Since each state is processed in the first instance of the round function for the first twelve rounds and the last instance for the rest, the scheduling ensues a start-up and end phase where only half of the round functions are utilized. This structure also requires additional multiplexers between the two round function instances. A conventional unrolled structure where each state is automatically processed in each round function during ever other clock cycle should be more efficient.

4.4. Proposal summary

Folding is the main technique for reducing area, but an increasing folding factor seemingly comes with a cost in throughput and potentially in efficiency. It is important to note the inherent trade-off with folding so that the utiliza-

tion of this technique is motivated by a goal of minimal footprint at a cost in efficiency. Pipeline registers can be incorporated in the round function if the folding is done slice-wise and with a folding factor larger than 8, but analysis suggests that this is not enough to compensate for the increased latency which is caused by the folding.

Regarding unfolded structures, pipelining is the main technique for improving the timing and therefore the throughput. The efficiency can be distinctively improved if the number of pipelines, PL is kept reasonably low and the pipelining is implemented properly. The inherent issue with pipelining is that the relevancy of the adoption of this technique either depends on the size of the messages and therefore the specific application of the hash function, or a new type of complex wrapper design which is capable of handling both multi-block-messages and small messages.

The ability to exceed the state-of-the-art with respect to efficiency seem to be dependent on the unrolling factor, UF. With a proportional amount of pipelines, the graph suggests that an increasing UF attains a good efficiency, though the plot is an asymptote with a clear upper bound. The analysis herein therefore points to the limitation of the efficiency of a SHA-3 implementation if all realistic applications for the hash function are considered. Hashing of multi-block-messages constrains the adoption of the explored optimization techniques, or the complexity of the structure including the interface. The throughput is in this case upper bound by the critical path through the round function and the efficiency is upper bound by the minimum required resources for an unfolded structure.

The models which have been used in these analyses have their limitations and it remains to be seen how accurate they are by comparison with the actual performance of the implementations. Timing results and the achievable frequency is a metric which is especially uncertain.

5. Implementation

The Existing literature is by now plentiful and with varying degrees of novelty and factors considered. In one aspect, the performances are diverging heavily compared to how similar many of the structures seem. On the other hand, reported results are based on different estimations. This makes comparing the existing literature a challenging task and drawing solid conclusions from the results is impractical. All the designs found in the later literature also utilize some sort of optimization technique such as pipeline registers. Only the earlier literature have presented designs and results based on a basic, straight forward structure. Many of the existing designs have also not included a wrapper in the assessment of the performance. Therefore, the initial task has been to arrive at a basic design which performance and structure can serve as a standard. Further modified structures and their performance can therefore be normalized to this basic structure for a clear and coherent analysis. Additionally, it has been desired to examine the feasibility of matching or surpassing the claimed performance of the existing literature. As already discussed, the basis for the results vary among the literature as well as the extent at which NP-hard problems such as Place and Route have been prioritized. While area consumption is less challenging to estimate, the most critical path is not.

The basic structure has been developed first as it serves as a foundation for further optimizations. This is the structure with the highest achievable efficiency when hashing of multi-block-messages are considered, thus representing what is suggested as the structural limit of a SHA-3 FPGA implementation. Using the basic structure as a foundation, various implementations are derived where different tradeoffs are considered and hence, various optimization techniques are adopted. A generic SHA3-512 structure is developed in order to explore the feasibility of this functionality. Only one previous paper by Athanasiou *et al.* [8] has presented a generic SHA-3 implementation which makes this an interesting structure to explore. A pipelined structure with and without unrolling has also been developed for hashing of multiple small messages. This is done both in order to further assess the existing structures which are relevant for this limited scenario and to improve the state-of-the-art. Especially the unrolled structure is interesting as it has only been seriously explored by Ioannou *et al.* [10], albeit with an inefficient architecture and without internal pipelining. Only these two structures have been developed with a disregard for multi-block-messages.

A folded structure has been developed which belongs to the mid-range class of SHA-3 implementations with FF=4. This structure is noteworthy more complex than the ones previously introduced. This is mainly caused by the dependencies of the ρ and θ step-mappings and the re-scheduling of the round function. The two earlier papers by Jungk *et al.* and Jungk & Apfelbeck [15, 15] have presented a mid-range structure with a similar folding factor. These structures have not included a complete wrapper in the assessment of the performance and the interface that is used is the Xilinx Fast Simplex Link IP core. Only one solution for the intra-round dependency of θ is presented in one of the papers and no clear solution is given with regard to how the ρ dependency is solved. The folded structure presented here incorporates a new solution for the θ intra-round dependency and is assessed as a stand-alone entity. The wrapper is compatible with the standard interfacing where lanes are transmitted sequentially.

5.1. Basic structure

A basic structure here is informally defined as having one round of Keccak-*f* implemented in combinatorial logic, registers are made up of flip-flops and standard syntax for a hardware description language such as VHDL is used to infer all components. The basic structure supports one of the sub-versions of SHA-3 and is configured for SHA3-512 as this is the most prevalent version with the highest security claim. For comparison with other existing implementations, results are adjusted to the SHA3-512 version with the smallest block size if other versions are presented. As depicted in Figure 5, the top level of the design contains three RTL blocks: statereg, roundfunc and wrapper. In this way, the three components correspond respectively

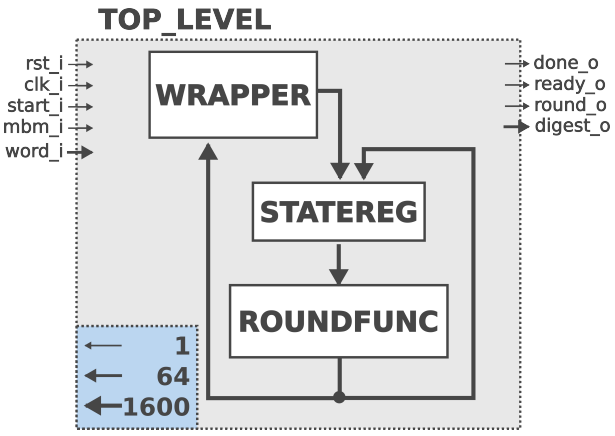

Figure 5: Simplified top level schematic of the basic structure.

to their theoretical modules: the state, Keccak-f and the sponge construction. The roundfunc is passive and the input is always connected to the state in statereg. The complex logic is therefore kept in the wrapper so that any structural modifications to the basic design involve alterations of the wrapper while the two other components remain mostly the same. Both the state, partitions of it such as a plane which is used during the θ step-mapping and the IO-buffer are realized as two-dimensional arrays of $x * 64$ bits. The IO-ports have been set to 64 bits for this structure, however, as have been pointed out regarding unrolling, a larger port size is required if UF and the block message size is sufficiently large.

A generic structure which supports all four sub-versions of SHA-3 is also implemented and differ from the basic structure only with respect to the wrapper component which is slightly more complex and with a larger IO-buffer in order to contain the largest message block of SHA3-224.

5.2. Pipelined and Unrolled structure

Using the basic structure as a foundation, one can proceed with implementing the explored optimization techniques. For the unfolded design, this means optimizing for efficiency. If the folding factor and small latency is to be maintained, then the number of relevant techniques are limited. As discussed, the potential for increasing the efficiency relies on the possibility of utilizing pipelines. This again depends on the consideration of messages and support of the wrapper. Therefore, multi-block-messages are not considered here and only small messages are supported for hashing.

A structure with one internal pipeline has been considered which corresponds to PL=2. The manner in which it is incorporated and how the round function is partitioned is loosely based on the analysis performed by Pereira *et al.* [18]. While the number of pipelines incorporated in their literature is found to be excessive, the study of propagation time of each step-mapping is still relevant and applicable. The top level of the pipelined structure is identical to the basic structure and the main modifications distinguishing these two are found in the implementation of the state register and the IO-buffer in distributed RAM and increased round counter of the wrapper. In the basic structure and in most designs in general, a portion of the occupied slices have unused FF. This means that for the incorporation of the first pipeline registers, the area does not necessarily increase proportionally. The round function can therefore be split into multiple combinational paths and a more efficient slice utilization ensues. A precondition for this is that the slices are not saturated with control signals as this inhibits further utilization.

The unrolled structure is adapted from the pipelined structure, but with a further modification of the schedule with UF and PL=2. Four messages are here injected into the IO-buffer within the clock cycles of the 24 rounds. A second state register is implemented between instance one and two of the round function and the internal pipelining scheme is identical to the pipelined structure with PL=2.

The ι step-mapping is more complicated here than in the pipelined structure as four messages must be processed by the same round constants during different clock cycles. This is solved by storing the precomputed round constants as two 7x47 ROM with each 7-bit word replicated four times and divided between the roundfunc components. Each value of the round counter corresponds to a given round constant.

5.3. Folded structure

The folding factor affects all of the components of the design and the resulting complexity is noticeably higher than in the previously presented structures. With FF=4, the round function is reduced by 75 % of its width and the IO-buffer and the state registers are reduced equally with the utilization of the depth of the RAM. In addition to solving the inter-round dependencies of ρ and the intra-round dependencies of θ, the implementation must consider the re-scheduled round function, the interfacing between the slice-wise structure and the standard of SHA-3 and the dependency of ι with the provision of the round constants.

The intra-round dependencies caused by θ must be solved, either with extra logic or an extra clock cycle. Additionally, the memory mapping of the state must be done as efficiently as possible so that the least amount of slices are occupied. Each fold of round x depends on processed bits from each of the sub-rounds of round x-1.

The state is mapped into memory so that a memory block is assigned to each lane of the state. This is believed to be the best alternative for a slice-wise structure with a low folding-factor. The state is stored in distributed RAM and the slice utilization is therefore directly related with the folding-scheme. The utilized depth of the memory is equal to the FF and the width is equal to the total size of the state divided by FF. The lane-oriented memory mapping requires two instances of the state which are switched so that state A and B are read and written to every other round. This solution is easily applied to both distributed RAM and BRAM. A large extra register is necessary if the bits are not packed together. This works in a roll-around

manner so that for subsequent folds, the addressing of each memory block is simply incremented, as depicted in Table 2. Four different write addresses and one read address are used for the 24 RAM blocks depending on the sub-round and the state instance.

Cycle	0	1	2	3	4	5	6	7
Write	4	5	6	7	0	1	2	3
	5	6	7	4	1	2	3	0
	6	7	4	5	2	3	0	1
	7	4	5	6	3	0	1	2
Read	0	1	2	3	4	5	6	7

Table 2: Addressing of the state memory.

5.4. Implementation summary

Five different structures are presented here where one is folded and belong to the mid-range class of SHA-3 implementations while the other four are unfolded. The folded structure is the implementation with the highest complexity, but economize in area requirements at a cost in throughput. This structure is similar to one solution proposed by Jungk *et a.* [12, 15], but is unique in that it functions as a stand-alone entity as the wrapper incorporates an IO-buffer and converts the standard SHA-3 interfacing with the slice-wise folding-scheme. Additionally, a new solution for solving the intra-round dependency of θ is used. All structures incorporate a wrapper which handles the interface and control logic of the structure. This component is easily adaptable for modifications of the structure and is optimized for minimal latency and overall high efficiency. Combinations of optimization techniques beyond what is presented here are expected to increase the complexity of the wrapper as a multitude of circumstances and conditions must be considered. This is especially true with respect to the generic structure. The theoretical models of Chapter four suggests that the folded structure obtains a lower throughput and overall efficiency than the unfolded structures. Furthermore, of the latter, highest efficiency should be achieved by the structures which have adopted pipelines. The unrolled structure is expected to obtain the best efficiency. While only a version with an internal pipeline is implemented, the analysis suggests that this can be disregarded without much reduction in efficiency.

6. Experimental evaluation

All structures have been implemented on the xc5vlx50t Virtex-5 FPGA model and results are based on this if not mentioned otherwise. Each of the different structures implemented has been validated by both behavioral and post-Place and Route simulation and the generated digest compared with Known-Answer-Tests provided by the Keccak team [19]. The attained performance is presented here and assessed with respect to both the theoretical models conceived in relation with this work and the existing implementations.

The achieved performances of the structures developed in this project are listed in Table 3 along with the existing implementations comprising the relevant state-of-the-art. Structures in bold constitutes the contributions of this thesis. The structures are sorted by efficiency normalized for the SHA3-512 version. Because of the differences in the block size, the throughput and efficiency of a SHA3-256 structure is approximately 50% of an identical SHA3-512 structure. The presented structures in the state-of-the-art are implemented on Virtex-5 except those noted otherwise. UF(1-9) indicates the unrolling factor, FF(1-9) the folding factor, and PL(1-9)X/PL(1-9) indicates the number of pipeline stages and whether these are internal or e(x)ternal. A structure noted with PL2 incorporates one internal pipeline register in addition to the main state register. B (buffer) and NB (no buffer) implies whether an IO-buffer has been included in the assessment.

The top performance is achieved by the unrolled structure. The frequency is still inferior to both structures by Ioannou *et al.* [10] and comparable to Gaj *et al.* and Jararweh *et al.* [7] which are structures without any internal pipelines. In one way, this emphasizes the highly deviation timing performances of each structure, despite the utilization of solid optimization techniques such as pipelining. It is however, important to consider the basis of the results. The performance of the unrolled structure matches the unrolling model to a decent extent and the area is larger than the basic structure by an expected amount. The frequency of the unrolled and the pipelined structures are similar which is expected as the critical path should be identical.

The area consumption of the folded structure is quite comparable to the results attained by Jungk *et al.* [12, 15] with their similarly folded structure. They do not include an IO-buffer and the deviation in the required area is likely caused by excessive amount of registers incorporated in their proposed solution. The timing performance is also comparable. A source of deviation is also the different solutions adopted for solving the intra-fold dependencies of θ.

7. Conclusion

The main goal of this work has been to improve the existing state-of-the-art with respect to efficient hardware implementations of the SHA-3 hash function. While a special focus has been made on FPGAs, as the prototyping technology, the aim has been to keep the presented solutions as general as possible with respect to the supporting technology. Several solutions are herein proposed in the form of SHA-3 hardware structures with distinct performance objectives and considerations which are found to exceed the performance of the state-of-the-art. The contributions of this work consists of a selection of proposed solutions evaluated for high-performance and theoretical models which through approximations demonstrate the upper bound efficiency of SHA-3 hardware implementations with respect to relevant optimization techniques. Additionally, The necessary preconditions and considerations for the utilization of the relevant optimization techniques are demonstrated.

Five structures were proposed in this thesis. A basic

Paper	Latency (clk cycles)	f (MHz)	A (slices)	T (Gbps)	T/A (adjusted) (Mbps/slice)	Note
Unrolled	12	287	1967	13.78	7.00	UF2.PL2.B
Ioannou [10]	12	352 ††	2652	16.90	6.37	UF2.PL2X.NB
Pipelined	48	273	1163	7.80	6.06	PL2.B
Ioannou [10]	24	382 ††	1581	9.17	5.79	NB
Athanasiou [8]	48	389	1702	18.70	5.48	PL2.NB
Gaj [16]	24	283 †	1272	12.82	5.34	B
Basic	24	223	1192	5.35	4.49	B
Baldwin [5]	25	189	1117	8.50	4.06	NB
Generic	24	201	1268	4.83	3.80	B
Jungk *et al.* [15]	24	195	1305	8.49	3.59	NB
Pereira [18]	100	452 †	3117	7.70	3.34	PL4.B
Akin* [6]	121	509 †	4356	22.33	2.69	PL5.B
Baldwin [5]	25	189	1971	8.50	2.38	B
Jararweh [7]	24	271 †	2828	12.28	2.29	B
Akin* [6]	25	143 †	2024	6.07	1.63	B
Folded	96	200	460	1.20	2.61	FF4.B
San & At [13]	1062	520 ††	151	0.25	1.66	FF25.BRAM.B
Jungk *et al.* [15]	50	144	914	3.13	1.81	FF2.NB
Jungk *et al.* [15]	100	150	489	1.63	1.76	FF4.NB
Jungk *et al.* [15]	200	166	301	0.90	1.59	FF8.NB
Jungk & Ap [12]	200	159	393	0.46	1.17	FF8.NB
Jungk & St[20]	1665	257	90		0.99	FF64.B
Winderickx [21]	1730	248 ††	134	0.25	0.66	FF64.B
Jungk *et al.* [15]	1600	206	164	0.14	0.45	FF64.NB
Kerckhof** [11]	2154	250 ††	144	0.07	0.47	FF25.B
Bertoni [17]	5160	265	448	0.05	0.12	FF25.B

Table 3: Existing SHA-3 implementations sorted by efficiency adjusted for SHA3-512. Unfolded structures on top and folded structures below. FF(1-9)=folding factor, UF(1-9)=unrolling factor, PL(1-9)X/PL(1-9)= external/internal pipeline registers, NB/B=no buffer/buffer. *Virtex-4 implementation. **Virtex-6 implementation. † Results obtained from synthesis. ††Unknown source of results.

structure, representing the most straight forward implementation of the SHA-3 algorithm, is used as a basis of comparison for the existing state-of-the-art and the structures herein proposed; a generic structure, supporting all four sub-versions of SHA-3 which is a functionality that has been scarcely covered by the existing literature; an unrolled structure which contains two instances of the round function; a purely pipelined structure with an extra state register implemented in the round function; a folded structure which reduces the area requirements by processing parts of the state per clock cycle.

The unrolled structure allows for achieving a throughput of 13.8 Gbps with a cost of 1967 slices, on a Virtex-5 FPGA. This yields an efficiency of 7 Mbps/slice, being 9% better that be best state-of-the-art to date. This is achieved by multiplying all components of the structure apart from the wrapper containin the control logic. This results in the latency being reduced by 50% while the area is increased by less than 100%. It is important to increase the number of state registers proportionally so that the frequency is not decreased. A combination with internal pipelining is proven to be effective.

Both the unrolled and the purely pipelined structure have internal pipeline stages in the round function. This is a key factor for the reduction of the critical path and increase of the efficiency. As such, pipelining and the combination of pipelining and unrolling are effective techniques for improving the efficiency. However, as is argued here, the relevancy of the adoption of these techniques is limited by the messages which are considered for hashing. A precondition for the success of pipelining is that bubbles are prevented so that all pipeline stages are full during each clock cycle. This will not be the case for messages which are larger than the block size. The sponge function nature of SHA-3 specifies that each block is processed by the round function for 24 rounds before being merged with the subsequent block of the same message. Therefore, only blocks of unrelated messages are able to fill consecutive pipeline stages. As unrolling is inherently inefficient without pipelining, these considerations are relevant for both optimization techniques. Distinctly higher efficiencies than the basic structure are only found to be achieved by structures which implement pipelining. Where hashing of arbitrarily sized messages are considered, the basic structure represents roughly the upper bound efficiency of SHA-3 hardware implementations.

The other proposed structure that significantly improves the state-of-the-art is the folded structure. This structure considers a folding factor of 4, meaning that 25% of the state is processed in each clock cycle. This is the only structure presented which economizes in area requirements. As herein explored, the ensuing trade-off between area requirements and throughput is distinctively asymmetric. This means that a compact structure will inevitably obtain a much lower throughput and efficiency than a fully unfolded one. This folded structure allows for achieving a hashing throughput of 1.2 Gbps at a cost of 460 slices, on a Virtex-5 FPGA. This yields an efficiency of 2.6 Mbps/slice. Clearly, this achieved efficiency is significantly lower than the unrolled structure herein proposed. Nevertheless,

it is the folded structure which presents better efficiency metrics when compared with the related folded state-of-the-art, being 28 % more efficient than the best slice-wise folded structure of the state-of-the-art. This improved efficiency is achieved by an efficient structure which only contains the absolute necessary registers for a proper functionality. A memory mapping solution herein suggested, is adopted, which limits the required RAM units to the bare minimum of what is required with a small extra register. The intra-round dependencies caused by θ are solved by a novel solution which allows for slice 0 to be stored in RAM in contrast with the existing solution.

From this, it can be concluded that the use of the folding technique should be motivated by the particular objective of reducing area resources. However, the higher the folding factor, the lower the overall efficiency that can be reached.

8. Future work

Interesting future research directions involve the further exploration of combinations of the discussed optimization techniques for unfolded structures. Also, a highly unrolled structure with a generic SHA-3 sub-version support can be very useful. For optimal stand-alone functionality, a padder component could be incorporated in the wrapper. The developed wrapper component also has room for improvement. The control logic can be implemented more efficiently with a faster counter and the IO-buffer can be reduced by one lane as this can be fed directly from the input port to the state register. Structures with other folding factors are also worth evaluating. Specifically, a structure with FF=8 which could allow for an internal pipeline to be implemented in the round function. This may allow for improved performances.

Acknowledgments

This work was supported by the ARTEMIS Joint Undertaking under grant agreement n. 621429 and 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. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[2] D. J. Bernstein & T. Lange. "eBACS: ECRYPT Benchmarking of Cryptographic Systems", 2012. https://bench.cr.yp.to/results-sha3.html.

[3] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche and R. V. Keer. "Keccak sponge function family main document ", 2008. http://keccak.noekeon.org/Keccak-main-1.0.pdf.

[4] J. Stromgbergson. Implementation of the Keccak Hash Function in FPGA Devices. December 2008.

[5] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill and W. P. Marnane. "FPGA Implementations of the Round Two SHA-3 Candidates". 2010. http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/BALDWIN_FPGA_SHA3.pdf.

[6] 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.

[7] H. Tawalbeh Y. Jararweh, L. Tawalbeh and A. Moh'd. Hardware Performance Evaluation of SHA-3 Candidate Algorithms. *Journal of Information Security*, pages 69–76, April 2012.

[8] G. P. Makkas G. S. Athanasiou and G. Theodoridis. High Throughput Pipelined FPGA Implementation of the New SHA-3 Cryptographic Hash Algorithm. *IEEE*, 2014.

[9] Y. Ayuzawa, N. Fujieda, and S. Ichikawa. "Design Tradeoffs in SHA-3 Multi-Message Hashing on FPGAs", 2014.

[10] H. E. Michail L. Ioannou and A. G. Voyiatzis. High Performance Pipelined FPGA Implementation of the SHA-3 Hash Algorithm. *MECO*, pages 1–4, 2015.

[11] S. Kerckhof, F. Durvaux, N. Veyrat-Charvillon, F. Regazzoni, G. M. de Dormale, and F. X. Standaert. Compact FPGA Implementations of the Five SHA-3 Finalists. *Researchgate*, January 2011.

[12] B. Jungk and J. Apfelbeck. Area-efficient FPGA Implementations of the SHA-3 Finalists. *IEEE*, 2011.

[13] 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.

[14] B. Jungk. "FPGA-based Evaluation of Cryptographic Algorithms - PhD Dissertation", 2016.

[15] B. Jungk, M. Stöttinger, and M. Harter. Among Slow Dwarfs and Fast Giants: A Systematic Design Space Exploration of KECCAK. 2013. jungkshrink.

[16] M. Rogawski E. Homsirikamol and K. Gaj. Comparing Hardware Performance of Round 3 SHA-3 Candidates using Multiple Hardware Architectures in Xilinx and Altera FPGAs. *Researchgate*, 2011.

[17] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche and R. V. Keer. "Keccak implementation overview Version 3.3", 2012. http://keccak.noekeon.org/Keccak-implementation-3.2.pdf.

[18] F. Pereira, E. Ordonez, I. Sakai and A. Souza. "Exploiting Parallelism on Keccak: FPGA and GPU Comparison", 2013.

[19] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. "Known-Answer-Tests and Monte Carlo test results". http://keccak.noekeon.org/KeccakKAT-3.zip.

[20] UNKNOWN. Hobbit - Smaller But Faster Than A Dwarf: Revisiting Lightweight SHA-3 FPGA Implementations. *ReConFig 2016*, 2016.

[21] 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.