On Optimized FPGA Implementations of the SHA-3 Candidate Grøstl[⋆]

Bernhard Jungk, Steffen Reith, and Jürgen Apfelbeck

Fachhochschule Wiesbaden University of Applied Sciences
{jungk|reith}@informatik.fh-wiesbaden.de
apfelbeck@ite.fh-wiesbaden.de

Abstract. Actual and future developments of the automotive market (e.g. the AUTOSAR project or car2car communication systems) will increase the need for a suitable cryptographic infrastructure in modern vehicles. A core component for such a generic cryptographic core is a secure cryptographic hash function, because these functions are the base for a lot of applications like challenge-reponse authentication systems and digital signature schemes. In the present paper we evaluate the SHA-3 candidate Grøstl with respect to area requirements, which play a very important role for cost-sensitive markets.
The National Institute of Standards and Technology (NIST) has started a competition for a new secure hash standard. In this context third party implementations of all proposed hash functions are regarded as an important part of the competition. We chose to implement the Grøstl hash function for FPGAs, for its resemblance to AES. More precisely we developed two optimized versions, one optimized for throughput, the other one for area. Both implementations improve the results and estimates presented in the original submission to the competition. The performance of both implementations may be improved further, thus Grøstl seems to be a good candidate for implementations on medium sized FPGAs. Besides that, it is shown that Grøstl needs a significant amount of resources, which will hinder its use for automotive applications.

Key words: Cryptography, hashfunction, FPGA, automotive, car2car

1 Introduction

The National Institute of Standards and Technology (NIST) has started a competition for a completely new hash function, very similar to the past AES competition [1]. Again this competition requires third party implementations of all proposed candidates and both software and hardware implementations are necessary to evaluate the overall performance of the proposed hash functions.

[⋆] Research supported in part by BMBF grant 17N1308.

In our work we focus on FPGA implementations of the candidate Grøstl [2]. The Grøstl hash function borrows many ideas from the Rijndael/AES algorithm [3], which is one of the reasons, we chose this particular SHA-3 candidate. FPGA implementations are important for automotive applications, because they can offer better performance at a lower cost in many areas, where software implementation would be too slow and a custom ASIC chip would be too costly.

For this work two implementations were developed and evaluated. The first implementation is a high-throughput implementation of Grøstl, whereas the second one is an area optimized version. Our results show a slight improvement (cf. Tab. 1) of the reported and estimated results in [2].

2 Previous work

The Grøstl algorithm is described in detail in [2], where a reference software implementation and some optimized versions can be found, which will not be considered further in this paper. Additionally, some results and estimates on FPGA implementations are given there, which are comparable to our implementations. Beside these early results, we are not aware of any other published implementation results, especially with respect to area constraints.

One benefit of the Grøstl hash function is its similarity to AES, because extensive research has been done to improve the throughput and reduce the required area of AES (e.g. [4,5,6]). This existing work can be used to improve Grøstl, as we did in our area optimized version, which is based on ideas presented in [5]. Further work on AES and Grøstl may result in optimizations for both algorithms.

Many more optimizations are known for AES and some of them can be expected to be applicable to Grøstl. Especially the AES S-Box optimizations may be applied unchanged [4]. Some architectural optimizations are probably not directly applicable to Grøstl. For example, pipelined high-throughput implementations of AES often use the less secure ECB mode, which does not depend on the output of the procession of the previous message block [7]. In contrast Grøstl is similar to the CBC mode, which feeds the output back into the processing of the next message block, thus limiting the usefulness of pipelining for Grøstl.

3 The Grøstl hash function

The Grøstl hash function uses a compression function f, which is executed for each message block m_i. Each message block m_i is one part of the original input message m and has l-bits. The value of l depends on the desired hash length. The submission paper of Grøstl fixes l for 224 and 256 bit-sized hashes to $l = 512$ and for 386 and 512 bit hashes to $l = 1024$.

The compression function f uses two transformations P and Q to compute $f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h$, where h is either an initial value, if it's the first message block or the output of the previous computation of f.

Both transformations P and Q are composed of four different sub-transformations *AddRoundConstant*, *SubBytes*, *ShiftBytes* and *MixBytes*. Each of the transformations P and Q perform these four sub-transformations sequentially for n rounds, where $n = 10$ for the 224 and 256 bit hashes and $n = 13$ for 384 and 512 bits.

It is important to note, that the sub-transformations map each message block to a matrix representation with eight rows and eight or sixteen columns, depending on l. Each entry of this matrix is one byte of the message block.

The *AddRoundConstant* transformation adds the current round to this matrix (add means bitwise XOR). The P-instance of this transformation adds the round to the very first byte, whereas the Q-instance adds $0xFF \oplus$ *round* to the first byte in the eighth row. The *SubBytes* transformation is a non-linear transformation, which uses the exact same s-box as AES does. The *ShiftBytes* transformation performs a cyclic left shift for each row. The first to the seventh row are shifted 0 to 6 bytes to the left, whereas the eighth row is shifted 7 bytes for $l = 512$, or 11 bytes for $l = 1024$. The final *MixBytes* transformation performs a matrix multiplication

$$A \leftarrow B \times A,$$

where A is the message block mapped to the matrix representation and B is a circulant matrix $B = circ(02, 02, 03, 04, 05, 03, 05, 07)$. *circ* uses the given row to fill the matrix B, by rotating the round to the right each further row, beginning at the first row of B with the original argument to *circ*.

After all rounds are completed and all message blocks are processed, the compression function outputs the compressed result x. This result is used by the output transformation Ω, which first computes $P(x) \oplus x$ and then truncates the result to the desired hash size, which results in the final hash value.

Another important function of Grøstl is the padding function which takes a message m of arbitrary length as input and outputs a padded message m' which is a multiple of l.

4 Implementations

We have developed two implementations of the Grøstl hash function. The first implementation is optimized for throughput, whereas the second is an area optimized version. Currently, both implementations are implemented with only 512 bits internal state, thus they are only able to compute 224 or 256 bit sized hashes. Our choice matches perfectly to applications in the automotive environment, because of strong resource restrictions in this market.

The most fundamental difference between both implementations is the computation of each Grøstl round. The high-throughput implementation computes the sub-transformations of P and Q completely in parallel, thus requiring only one clock cycle per round. The area-optimized divides the internal state in eight smaller sub-states. For each sub-state the sub-transformations of P and Q need only one clock cycle, thus the number of clock cycles for one Grøstl round is increased by a factor of eight.

Both implementations share a common design (Fig. 1). At a first glance, the design is a straight forward implementation of the Grøstl algorithm, but there are some important changes, which reduce the required area in both implementations.

The padding function receives all message blocks and passes them to the compression function f, padding the message blocks as necessary at

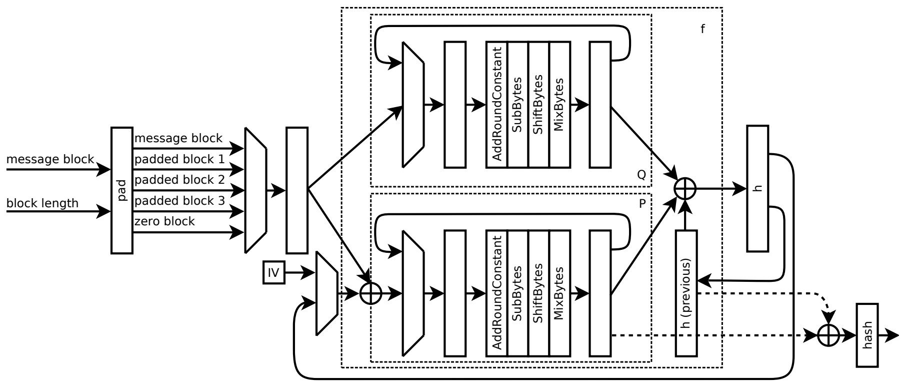

Fig. 1. The overall design of both implementations.

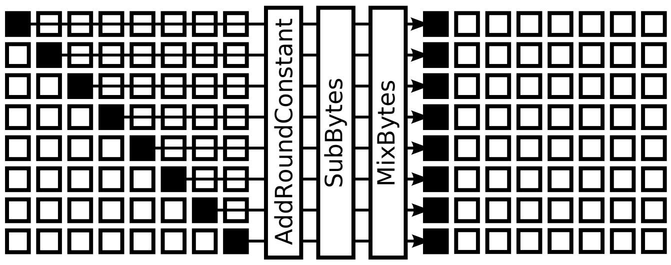

Fig. 2. The I/O register implementation for the Grøstl round.

first. The compression function takes each message block and applies the P and Q transformations in parallel. Depending on the current round, the sub-transformations are applied to either the input message block or the output of the previous round. When the last round is complete, a new value for h is computed and fed back into P combined with the next message block. After the last message block is transformed, Ω is computed and the hash value is placed in an output register.

The design includes some optimizations, which were discovered by examination of the Grøstl hash function. One idea is to reuse the P-instance used for the compression function for the single invocation of P in Ω. The necessary changes are a result of the different input to P. Ω computes $P(h) \oplus h$. This obviously differs from the usual computation of $P(h \oplus m) \oplus Q(m) \oplus h$.

Reuse will be simplified greatly, if we arrange the input message block m, so that the computation of $h \oplus m$ results in h. Thus, the input message block m has to be all-zero for this last invocation of P. We achieve this by changing the padding function to output an all-zero message block at the right point in time. Now, reusing the logic used to compute one Grøstl round for the computation of P in Ω is easy, because we don't have to change anything else.

We could additionally change Q to output an all-zero output, too, but this change does not reduce the area compared to our approach, using 256 additional XORs, because both approaches require the same number of LUTs [8].

The high-throughput implementation takes a lot of chip area. For many mass market applications the size of the chip is more important than the maximum throughput. The obvious strategy to reduce the area requirement is to reduce the amount of parallelization, which in turn reduces the required copies of the S-Box in *SubBytes* and the matrix multiplication in *MixBytes*.

Our approach is similar to the compact AES implementation proposed in [5]. The implementation splits a Grøstl round in eight sub-rounds, each computed in one clock cycle, thus one Grøstl round now takes about eight times longer than before. One of the ideas presented in [5] is an optimization of the input and output registers needed for the Grøstl round.

They describe two ideas, the first is based on distributed RAM resources and the other uses shift registers. We adapted the approach with distributed RAM resources. The main idea is to use the distributed RAM as 8 bits wide and 8 bits deep dual-ported RAM. Eight of these RAMs together have the necessary size to hold the internal state of the Grøstl compression function. Two of these memory banks are required, one implements the input register, the other one the output register. The input and output registers are swapped after each round.

The *ShiftBytes* transformation makes it necessary to tap the bytes out of the input memory. This is achieved by using a counter for the sub-rounds and eight additions depending on the memory the input is read from. The output is written into each individual memories at the same position, thus the explicit *ShiftBytes* transformation is unnecessary (Fig. 2).

A similar idea may be adapted for the storage of h, but the different access pattern on h makes it unnecessary to use two identical memories. Instead we use only one memory bank. The computation of the output of the compression function $P(h \oplus m) \oplus Q(m) \oplus h$ reads the old value of h and writes the new value to the RAM at the same time. The dual-port RAM allows us to achieve an read-modify-write cycle in only one clock cycle using a pipelined RAM [9], thus the processing is not slowed down.

5 Evaluation

To our best knowledge we are not aware of other FPGA implementation results for Grøstl beside those already mentioned in [2]. The presented results are similar to our high-throughput implementation. Our area optimized result is the first effort of an area optimized implementation of Grøstl with actual published results, because in [2] only estimates are mentioned, which are based on the results of implementing the Whirlpool hash function [10].

To achieve a fair comparison to the previous results, we used the same Xilinx Spartan 3 FPGA. Table 1 shows the various implementation results for an Xilinx Spartan 3 FPGA. Our high-throughput implementation slightly beats the original result, but overall our implementation seems to be rather similar. The area optimized versions targeting the Spartan 3 and Virtex 2P are each more compact than the estimate. The area optimized version fits on an fairly small Spartan 3 (XC3S400) and it is probably possible to further optimize the implementation to fit on the next smaller Spartan 3. This situation is similar for the Virtex 2P implementation.

Variant	**FPGA**	**Slices**	**Frequency (MHz)**	**Throughput (MBit/s)**
throughput (ours)	Spartan 3	6136	88.3	4520
throughput [2]	Spartan 3	6582	86.7	4439
area (ours)	Spartan 3	2486	63.2	404
area (ours)	Virtex 2P	2754	81.5	512
area estimate [2]	Virtex 2P	3000-4000	75-125	400

Tab. 1. Comparison of implementation results for 224 and 256 bit hashes using an Xilinx Virtex 2P (xc2vp100-5ff1704) and an Xilinx Spartan 3 (xc3s5000-5fg676).

6 Conclusion and Further Work

Our current work focuses on a FPGA implementation of the proposed Grøstl hash algorithm. The high-speed implementation described in this paper computes each round of the compression function in parallel, which results in a fast, but area consuming FPGA implementation. The size-optimized implementation needs less than the half area, but in turn it is eight times slower. Overall the Grøstl hash function fits on medium sized FPGAs like the Spartan 3 XC3S400 and it's likely that this can be improved further. However, our implementation does not fit on any of the older and smaller Spartan 2 FPGAs and it seems unlikely, that Grøstl may be efficiently implemented on any, but the largest Spartan 2 versions.

Both implementations may be improved in many ways. For example the implementation of the S-Box may be improved [4], thus reducing the required area of both implementations. Pipelining of the high-throughput version may increase the throughput by some degree, but the benefit would be rather small, due to inherent data dependencies. Furthermore, it is possible, that other ideas from AES may apply to Grøstl, too (e.g. [11,12]). Thus one possible direction of further work, would be to review successful optimizations of AES for their applicability to Grøstl. At the moment the area requirements of both implementations are significant with respect to automotive applications. Hence our further work will focus on area optimization, rather than improvements of the throughput.

References

1. Kayser, R.F.: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. In: Federal Register. Volume 72. National Institute of Standards and Technology (November 2007) 62212–62220
2. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST (2008)
3. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. Submission to NIST (1999)
4. Canright, D.: A Very Compact S-Box for AES. In: Proceedings of 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Springer-Verlag (2005) 441–455
5. Chodowiec, P., Gaj, K.: Very compact FPGA implementation of the AES algorithm. In: Proceedings of 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Springer-Verlag (2003) 319–333
6. Pramstaller, N., Mangard, S., Dominikus, S., Wolkerstorfer, J.: Efficient AES Implementations on ASICs and FPGAs. In: Advanced Encryption Standard – AES. Springer-Verlag (2005) 98–112
7. McLoone, M., McCanny, J.: High Performance Single-Chip FPGA Rijndael Algorithm Implementations. In: Proceedings of 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES), London, UK, Springer-Verlag (2001) 65–76
8. Xilinx: Spartan-3 Generation FPGA User Guide. (2009)
9. Alfke, P.: Creative Uses of Block RAM. Xilinx. (2008)
10. Pramstaller, N., Rechberger, C., Rijmen, V.: A compact FPGA implementation of the hash function Whirlpool. In: FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays, ACM (2006) 159–166
11. Mentens, N., Batina, L., Preneel, B., Verbauwhede, I.: A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-Box. In: Topics in Cryptology - CT-RSA 2005. Volume 3376 of Lecture Notes in Computer Science., Springer-Verlag (2005) 232–333
12. Nikova, S., Rijmen, V., Schläffer, M.: Using Normal Bases for Compact Hardware Implementations of the AES S-Box. In: Security and Cryptography for Networks. Volume 5229 of Lecture Notes in Computer Science., Springer-Verlag (2008) 236–245