Merged Computation for Whirlpool Hashing

Ricardo Chaves[1,2], Georgi Kuzmanov[2], Leonel Sousa[1], and Stamatis Vassiliadis[2]
[1] Instituto Superior Técnico/INESC-ID. Portugal. http://sips.inesc-id.pt/
[2] Computer Engineering Lab, TUDelft. The Netherlands. http://ce.et.tudelft.nl/

Abstract

This paper presents an improved hardware structure for the computation of the Whirlpool hash function. By merging the round key computation with the data compression and by using embedded memories to perform part of the Galois Field (2^8) multiplication, a core can be implemented in just 43% of the area of the best current related art while achieving a 12% higher throughput. The proposed core improves the Throughput per Slice compared to the state of the art by 160%, achieving a throughput of 5.47 Gbit/s with 2110 slices and 32 BRAMs on a VIRTEX II Pro FPGA. Results for a real application are also presented by considering a polymorphic computational approach.

1 Introduction

Currently, the need for digital authentication and digital documents validation is of vital importance in most data transmission systems and secure environments. One class of algorithms used in such applications is designated by hash functions. In the near past, the most common hash functions were the MD5 and SHA-1; however, collision attacks have been reported for both. While for SHA-1 these attacks still require massive computations and are considered practically unfeasible for the time being [10], the MD5 hash function is now considered unsafe since collision attacks can be made on a standard desktop computer [5]. To overcome the weaknesses of SHA-1, a 256 to 512-bit hash function, SHA-2, is becoming more generally used. However, the SHA-2 is computational similar to SHA-1 and future attacks can be expected. In 2003, the Whirlpool hash function was introduced in the New European Schemes for Signatures, Integrity, and Encryption (NESSIE). This new hash function has a similar security level to SHA-2 (with 512 nits), while at the same time suggesting a better performance [6].

In this paper a novel, compact, hardware implementation of the Whirlpool hash function is presented. The computational structure is improved by:

- Merging the S-Boxes with the Galois Field (2^8) multiplication in embedded memories;
- Merging the round key computation with the data hashing.

In order to validate and test the structure, the proposed core has been implemented on a XILINX VIRTEX II Pro FPGA. Implementation results suggest a throughput of 5.47 Gbit/s, using 2110 slices and 32 BRAMs, resulting on a Throughput/Slice metric of 2.59 Mbit/s per Slice.

Comparative analysis with current state of the art suggests an efficiency improvement in Throughput/Slice, compared to related art of 162%. That is, a 12% higher throughput, achieved with only 43% of the reconfigurable logic used by the best related works.

The remainder of the paper is organized as follows. Section 2 describes the Whirlpool hash function and the main characteristics of the algorithm. Section 3 presents the proposed structure, where part of the computation is performed in lookup tables and the key computation and the data compression are merged. The obtained experimental results are used in section 4 for a comparative evaluation with the related art. Section 5 concludes the paper with some final remarks.

2 Whirlpool Hash Function

Whirlpool is a collision-resistant 512-bit hash function operating on messages up to 2^{256} bit long. It is based on a Miyaguchi-Preneel hash compression function and on the Rijndael algorithm [3].

The core computation of the Whirlpool hash function consists of 3 computation layers and one key addition, as depicted in Figure 1. These calculations are repeated for 10 rounds. The 512 bit input value is treated as an 8 by 8 byte matrix. The first computation layer is the non-linear layer (γ), where a byte substitution is performed for each byte of the 512 bit input [8]. In the second computation layer (π), the replaced bytes of layer γ are rotated. Since these permutations are fixed, they can be hardwired in a hardware implementation. The third computational layer is the diffusion layer θ. This layer consists of a modular multiplication applied to each row of the input matrix. The calculation of the first output byte (b_{00}) from θ is presented in (1) where

a_{0i} are the output bytes from the cyclical permutation layer.

$$b_{00} = a_{00} \oplus [a_{01} \otimes 09] \oplus [a_{02} \otimes 02] \oplus [a_{03} \otimes 05] \oplus [a_{04} \otimes 08] \oplus a_{05} \oplus [a_{06} \otimes 04] \oplus a_{07} \quad (1)$$

Finally the expanded key is added, in GF(2^8), corresponding to the XOR bitwise logical operation. These operations are depicted in Figure 1.

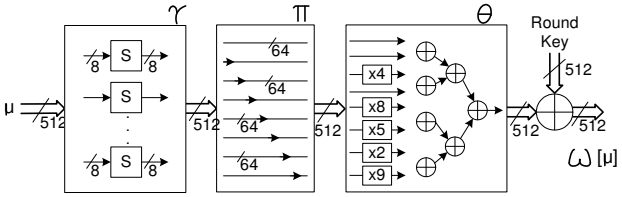

Figure 1: $w[\,]$ Whirlpool operations.

The output of one round is used as the input of the next round. After 10 rounds the output value is added, to the partial Digest Message (DM). This generates the new partial DM (H_i), has presented in (2). D_i is the i-th 512-bit block of the input message. The final DM is given by the last partial digest message (H_t) after all input blocks have been processed.

$$\begin{aligned} H_i &= W[D_i \oplus H_{i-1}] \oplus D_i \oplus H_{i-1} \; ; \\ DM &= H_t \; . \end{aligned} \quad (2)$$

Key schedule expansion: As depicted in Figure 1, each round of the digest message calculation requires a round key. The key schedule expands the initial 512-bit key, also designated by Initialization Vector (IV), into a sequence of round keys K_i^j where j is the round and i corresponds to the data block being computed. This key expansion is performed by applying the Whirlpool core computation to the key, as depicted in (3). The input value for each round is the value of the partial digest message H_i calculated in (2). For the first round, the IV is used.

$$\begin{aligned} K_i^0 &= H_{i-1} && \text{for } i \geq 1 \; ; \\ K_i^j &= W[K_i^{j-1}] && \text{for } j \geq 1 \; . \end{aligned} \quad (3)$$

For the round keys computation, a predefined round dependent constant is used [8]. For the computation of each Whirlpool round, two $w[\,]$ operations have to be performed.

3 Proposed Whirlpool Design

In this section, a hardware structure for the Whirlpool hash function implementations is presented. By exploring some of the computational and mathematical characteristics of this algorithm a faster and more efficient hardware implementation is achieved. A lookup table based Whirlpool implementation is described as well as the computational merging of the key computation with the data compression.

Lookup table implementation of Whirlpool: Identically to the AES algorithm, where a more efficient hardware can be achieved by using T-Boxes [2], in the Whirlpool function the non-linear layer γ and the diffusion layer θ can also be merged. This is possible, since the operations are performed at the byte level and the cyclical permutation π layer can be performed before the non-linear layer γ. With this modification, the main computational part of the Whirlpool calculation, which resides in the substitution operation of layer γ and part of the $GF(2^8)$ polynomial multiplication of layer θ, can be computed by lookup tables, as depicted in Figure 2. For the 512-bit message, 64 lookup tables of 8-bit input and 64-bit output are required.

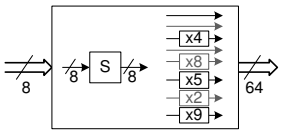

Figure 2: Whirlpool Lookup table (pW).

The output of the lookup table is the individual input bytes multiplied by the required coefficients. To conclude the computation of layer θ, the corresponding bytes of the polynomial multiplication are added. To complete the Whirlpool core operation, only the round key has to be added.

In order to reduce the size of the lookup tables by half, a less granular computational structure can be used. 32 bit output lookup tables can be employed, by analyzing and reducing the polynomial expression outputted by the lookup table. The 8 coefficients of the multiplied polynomial are:

$$(01, 01, 04, 01, 08, 05, 02, 09). \quad (4)$$

The first and trivial reduction, lays in the removal of the computational redundancy of the three multiplications by 01. The second reduction lays in the computation of some of the coefficients outside the lookup table. This external computation reduces the size of the lookup table but has to be properly chosen in order not to significantly affect the critical path of the overall computation. Analyzing the coefficients, it becomes clear that the most efficient option is the computation of multiplication by 02 from the computed multiplication by 01. Identically, the multiplication by 08 can be computed from the multiplication by 04. With this computational structure, a good compromise between a coarse grain and a fine grain implementation of the algorithm is achieved.

Merging computational blocks: The core computation of Whirlpool, described in Figure 1, is used in both the

data and the round key computation. This additional computation is advantageous for the safety of the algorithm, also making its implementation more resistant to Differential Power Analysis (DPA) attacks [8]. However, as a consequence, this additional computational cost represents an increase in the required silicon area, for a hardware implementation. This however can be minimized. Since the computation performed to obtain the round key is exactly the same as the computation for the data compression. A significant gain in terms of area can be achieved, by merging this computation into a single hardware structure. To allow both computations to coexist and maintain approximately the same performance, a pipeline structure can be used. By merging the computation some additional multiplexing logic is required, as depicted in Figure 3. With this

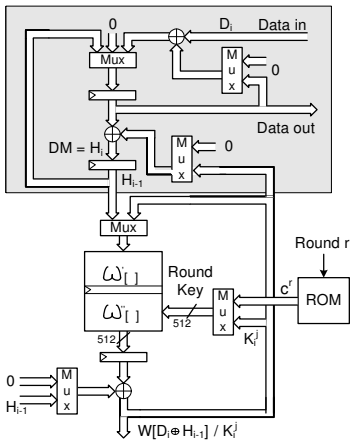

Figure 3: Whirlpool proposed core.

one level pipeline structure, the 10 computational rounds of the Whirlpool algorithm require 20 clocks cycles. Given that the round key computation uses the partial digest message (H_{i-1}) of the previous round, one additional clock cycle is required. Due to the fact that the next data round key computation can only start when the computation of the previous partial digest message (H_{i-1}) is concluded, the computation of each data block of 512 bits requires 21 clock cycles in total. Note that a pipelining structure would not be efficient if the computational merging were not explored.

In order to simplify the data input/output and to reduce the amount of required hardware, the input registers described in Figure 3 within the grey area, also work like shift registers. These shift registers are used to load the 512-bit data block, in a more efficient way, with 64-bit data input/output blocks. The $D_i \oplus H_{i-1}$ addition is performed as data are loaded, thus only one 64-bit XOR gate is required.

4 Performance analysis and related work

In this section, experimental results on a Xilinx FPGA realization of the proposed core are presented and compared with current art. A polymorphic implementation of the Whirlpool core is also realized.

In order to test and evaluate the proposed design, the Whirlpool core has been realized in a Xilinx VIRTEX II Pro (XC2VP30-7) FPGA using the Xilinx ISE (6.3) tool. Additional realizations for a VIRTEX-E and a VIRTEX-4 have also been obtained in order to properly compare with existing Whirlpool art. For the polymorphic implementations, the MOLEN processor prototype [9] was used. All the presented values were obtained after Place and Route, or after actual measurements on the prototype hardware.

Whirlpool FPGA Realization: By using the embedded block memories (BRAMs) of the VIRTEX FPGAs, a more compact implementation can be achieved. These memories can be directly used to implement the lookup tables described in section 3. For the VIRTEX II Pro and VIRTEX-4 FPGAs, BRAMs can be used with outputs up to 32 bits per BRAM. For the VIRTEX-E family, which only has BRAMs with outputs up to 16 bits, two block memories have to be used for each lookup table. The results for the FPGA implementations are depicted in Table 1. This table also presents data for the current state of the art.

When compared with the structure proposed in [4], implemented on a VIRTEX-E FPGA, an improvement of merely 39% is achieved, however the values presented in the referenced paper are from synthesis only, while ours are after Place&Route. Furthermore the values presented with a star (*) in Table 1 are incoherent when compared with other papers [6, 7] and our results. For a very fine grain structure, the number of slices is too small. In addition the maximum frequency reported in this paper [4] is extremely high especially when compared with the proposed Whirlpool core in this paper, which with a better usage of the embedded components of the FPGA and with a pipeline is only able to achieve 67 MHz while, the reported frequency in [4] is 75MHz. These erroneous results might be due to the inaccuracy of the synthesis estimator.

In [7], a compact implementation is presented for a VIRTEX II Pro FPGA. The proposed Whirlpool structure requires 45% more slices and 32 BRAM, not used in [7]; however, it is capable of achieving a throughput more than 13 times higher. In terms of Throughput per Slice, our core is 10 times better than the one proposed by Prams [7]. The Throughput per Slice metric does not take into account the BRAM usage, since these are available resources, independently if they are used or not. As BRAMs are hardwire cores in the considered FPGA technology, the silicon area they occupy on the chip is much smaller than the area of the slices. Therefore, even if some metric is used to take into account the BRAM area, into a different throughput per silicon area metric, it is not expected to introduce significant changes in the figures reported in the last row of Table 1.

Regarding the most recent art [6], implemented on a

Table 1: Whirlpool performance comparison

Architecture	Kitsos [4]	Our	Prams [7]	Our	McLo [6]	Our
Device	XCV1000-8	XCV1000-8	XC2VP40-7	XC2VP30-7	XC4V	XC4V-11
Slices	5585*	2138	1456	2110	4956	2118
BRAMS	0	64	0	32	68	32
Freq. (MHz)	75*	67	131	224	94	220
TP.(Mbit/s)	4.48*	2.38	0.38	5.47	4.79	5.38
TP/Slice	**0.8**	**1.11**	**0.26**	**2.59**	**0.97**	**2.54**

VIRTEX-4, the comparison results suggest that the proposed core is capable of a throughput increase of 12%, while reducing by 57% the required slices and 50% the required BRAMs. In terms of global efficiency estimated by the Throughput per Slice metric, our core is 162% better.

Polymorphic Whirlpool Implementation: In order to easily integrate the proposed core in software applications and to easily validate and test it in real hardware, it was integrated into the MOLEN polymorphic processor [1, 9]. In this processor, the core is integrated as a CCU that directly accesses the main memory and communicates with the General Purpose Processor (GPP) via a set of exchange registers. The core is evoked as an equivalent software function call, requiring minimal modification of the pure software implementation. In order to use the proposed Whirlpool core as a CCU unit for the MOLEN processor, some additional wrapping logic is required. This additional wrapping logic does not penalize the critical path of the core. The CCU for the proposed Whirlpool core requires 2245 slices (16%) and 32 BRAMs, corresponding to an additional 127 slices, regarding the standalone core. In this functional test, the Whirlpool CCU is running with the same clock frequency as the main data memory, operating at 100MHz. At this frequency, throughputs of 2.4Gbit/s are achieved.

5 Conclusions

In this paper, a performance and hardware efficient structure for the most recent hash function, the Whirlpool, is presented. In the proposed implementation, the S-Boxes and part of the GF(2^8) multiplications are merged into a single operation, performed by a lookup table. Since the most complex part of the GF(2^8) multiplication is computed by a lookup table, a faster and more compact structure is achieved. Additionally, the merging of the round key computation and the data compression hardware is also proposed. Implementation results suggest an overall improvement of the Throughput/Slice to related art of about 160%; having a 12% faster throughput with only 43% of the reconfigurable area of the previous best proposals. On a VIRTEX II Pro FPGA, a throughput of 5.47 Gbit/s is achieved, with 2110 slices and 32 BRAMs, resulting in a Throughput per Slice of 2.59. A real application has been tested on a polymorphic prototype running at 100MHz, achieving throughputs of 2.4 Gbit/s.

Acknowledgments

This work was partially supported by the Dutch Technology Foundation STW, applied science division of NWO (project DCS.7533); the Portuguese Foundation for Technology; and the European Network of Excellence on High-Performance Embedded Architecture and Compilation.

References

[1] R. Chaves, G. Kuzmanov, L. A. Sousa, and S. Vassiliadis. Improving SHA-2 hardware implementations. In *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006*, October 2006.

[2] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. A. Sousa. Reconfigurable memory based AES co-processor. In *Proceedings of the 13th Reconfigurable Architectures Workshop (RAW 2006)*, page 192, April 2006.

[3] J. DAEMEN and RIJMEN. The design of Rijndael. AES-The Advanced Encryption Standard. *Springer-Verlag*, 2002.

[4] P. Kitsos and O. Koufopavlou. Efficient architecture and hardware implementation of the Whirlpool hash function. *IEEE Transactions on Consumer Electronics*, 50:208 – 213, February 2004.

[5] V. Klima. Finding MD5 collisions a toy for a notebook. Cryptology ePrint Archive, Report 2005/075, 2005.

[6] M. McLoone, C. McIvor, and A. Savage. High-Speed Hardware Architectures of the Whirlpool Hash Function. In G. J. Brebner, S. Chakraborty, and W.-F. Wong, editors, *FPT*, pages 147–162. IEEE, 2005.

[7] N. Pramstaller, C. Rechberger, and V. Rijmen. A compact FPGA implementation of the hash function Whirlpool. In S. J. E. Wilton and A. DeHon, editors, *FPGA*, pages 159–166. ACM, 2006.

[8] V. Rijmen and P. S. L. M. Barreto. The WHIRLPOOL hash function. World-Wide Web document, 2001.

[9] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte. The Molen polymorphic processor. *IEEE Transactions on Computers*, pages 1363–1375, November 2004.

[10] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.