Scalable FPGA Design and Performance Analysis of PHASH Hashing Function

Przemysław Zalewski
Department of Computer Engineering
Rochester Institute of Technology
Rochester, NY 14623

Marcin Łukowiak
Department of Computer Engineering
Rochester Institute of Technology
Rochester, NY 14623

Stanisław Radziszowski
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623

Abstract— **This paper presents an FPGA design and performance analysis of a recently proposed parallelizable hash function – PHASH. The main feature of PHASH is that it is able to process multiple data blocks at once making it suitable for achieving ultra high-performance. It utilizes the W cipher, as described in the Whirlpool hashing function at its core. A Virtex-4 FX60 FPGA was used in order to verify functionality of the implementation of the algorithm in hardware. To achieve high performance, state-of-the-art Virtex-5 LX330 FPGA was used as target platform. PHASH achieved a throughput over 15 Gbps using a single W cipher instance and 182 Gbps for 16 instances. For fair comparison of the performance of PHASH with widely accepted SHA-512 and Whirlpool hashing functions we have also developed their high performance implementations targeting the same FPGA platforms. SHA-512 implementation attained a throughput of 1828 Mbps, and Whirlpool attains 7687 Mbps.**

Index Terms— **scalable FPGA design; hash function; performance analysis.**

I. Introduction

Hashing functions play a fundamental role in modern cryptography [1]. Such functions process data to produce a small fixed size output referred to as a digest or hash. Typical applications of these functions include data integrity verification and message authentication schemes. The SHA (SHA-0, SHA-1, SHA-2) family of functions is one of the first widely accepted standards for hashing. SHA-512 is one of the five variants belonging to the SHA-2 family of hashing functions, which in August 2002 was announced as the US federal standard defined in the FIPS PUBS 180-2 [2]. According to currently published literature [4], the fastest SHA-512 implementation on FPGA achieves a throughput of 1550 Mbps. Hashing function Whirlpool [5] provides comparable security to SHA-512 but is able to achieve much better performance. According to currently published literature, the fastest Whirlpool implementation on FPGA achieves a throughput of 4790 Mbps [6].

Newly proposed PHASH [8] is the main function investigated in this work. For fair comparison we have also developed high performance implementations of SHA-512 and Whirlpool hashing functions targeting the same FPGA platform.

II. Overview of PHASH

PHASH [8] is a hash function designed to allow computations on data blocks to be performed in parallel. It consists of three stages: message padding, message compression and message reduction. Figure 1 shows a high level diagram of the PHASH hashing function computed sequentially. The same function can be computed in parallel as outlined in the sequel. PHASH relies on a block cipher for message compression. Any sufficiently large block cipher could be used for this purpose. For completeness, the W cipher, as presented in the Whirlpool hashing function [5], is used here. The message reduction stage is responsible for combining the intermediate values generated by the message compression stage into a single hash.

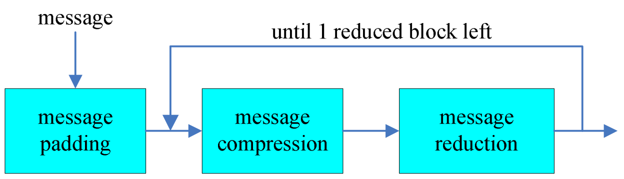

Figure 1: High level block diagram of PHASH

Figure 2 shows a detailed block diagram of the PHASH hashing function. The message compression stage takes each 896-bit block, X_i, and partitions it into two sub-blocks, Y_i and Z_i (not labeled explicitly in Figure 2), where i represents the ith block of the message to be hashed, after padding. Y_i consists of the 384 most significant bits of X_i, and Z_i consists of the remaining 512 bits. The Y_i block is concatenated with an unsigned 128 bit representation of i, producing a 512-bit block, V_i. Recall that the W cipher [5] takes a 512-bit plaintext input and a 512-bit initial key. For each Y_i block the V_i block is used as the plaintext to the W cipher, and the Z_i block is used as the key to the W cipher. The 512-bit compressed block C_i is the resulting ciphertext. The computation of the C_i's can be performed in parallel, since the computations are performed on independent blocks of data. In the message reduction stage the previously computed C_i blocks are reduced into D_i blocks. Computation of each D_i block involves using the computed C_i blocks. The first 128 C_i blocks undergo the XOR operation with each other. The next 128 C_i blocks undergo the same operation. If less than 256 C_i blocks are computed, the remaining blocks consist of all '0' bits. As a result of the above process, 256 512-bit values are reduced to two 512-bit values. These two values are then fed into the W cipher as the key and plaintext, respectively, and a single 512-bit ciphertext, D_0 is produced as a result. If more than 256 C_i blocks are computed, the next 256 C_i blocks are used in order to generate D_1 in the manner described above. The process continues until all D_i blocks have been generated. Then the same reduction process as previously described is applied to all D_i blocks until a single 512-bit value remains as a result of the reduction process. This final 512-bit value represents the hash of the original message.

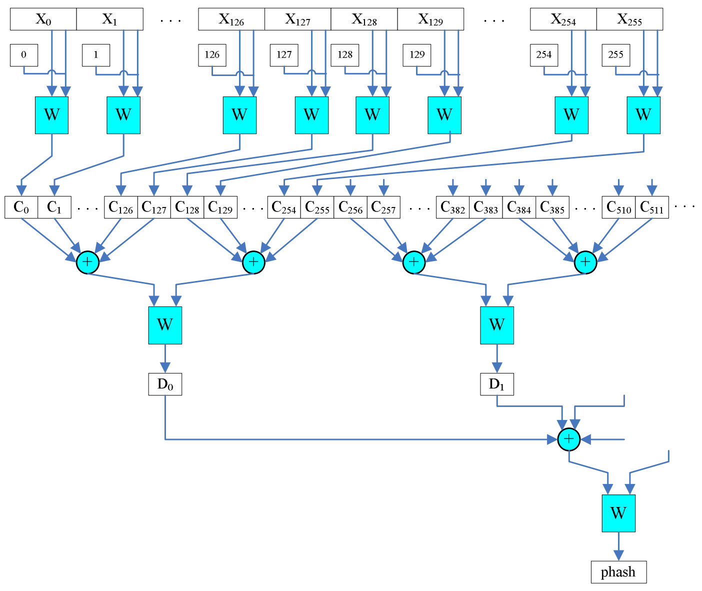

Figure 2. Detailed block diagram of PHASH

III. Previous Work on SHA-512 and Whirlpool

The SHA-512 and Whirlpool designs in this work have been derived as follows. The reference SHA-512 implementation descriptions were obtained from a combination of [3, 4, 9, 10, 11, 12 and 13]. In [4] a partially unrolled SHA-512 implementation is presented. The message compression stage is unrolled two times. In [11] a quasi-pipelined implementation of SHA-512 is introduced. The message compression stage is pipelined, as it dictates the critical delay of the algorithm. The basic message expansion stage is altered so as to decrease the delay of the individual computations. The delay balancing technique is used to achieve a decrease in the delay of this stage. To perform the required additions carry-save and carry look-ahead adders are used. In [3] yet another SHA-512 implementation is presented. The implementation draws on the collective strengths of most of the previously published work. It is quasi-pipelined and unrolled. The implementation uses carry-save and carry propagation adders, as presented in [12]. BlockRAMs are used to store the Kt constants, as presented in [13]. In [10] operation rescheduling was used. This technique also involves pre-computation in order to minimize the critical delay.

The reference Whirlpool implementation descriptions were obtained from a combination of [6 and 7]. In [6] the substitute bytes stage was implemented using both a full look-up table placed in BlockRAMs, as well as by implementing the three mini s-boxes in the distributed RAM located in CLB slices. The twice unrolled architecture that used the mini s-box approach to implement this stage was also presented. As a result three different Whirlpool implementations were obtained. The shift columns stage was implemented with no additional logic; the data paths were hardwired to perform the reordering of the data. The mix rows stage was implemented by shifting the input byte around and the modular reduction step was performed, if necessary, on the fly. The add key stage consisted of only XOR operations. In [7] Whirlpool was implemented using the architecture that duplicates the key schedule, as well as the architecture which integrates the key schedule into the design. The substitute bytes stage was implemented using both the mini s-box approach and the Boolean expression approach. As a result four different Whirlpool implementations were obtained. The shift columns stage was implemented using combinational shifters. The mix rows stage was implemented by using a look-up table containing the results of the multiplications of all possible inputs by the reduction polynomial in GF(2^8). The add key stage consisted of simple XOR operations.

IV. Design

A. SHA-512, Whirlpool and PHASH Design

A model for each of the three algorithms was created using VHDL (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language).

The SHA-512 implementation was simple and straightforward. No unrolling or operation rescheduling was used. Distributed RAM or BlockRAM were used to store the required Kt constants for the message compression stage.

The Whirlpool implementation was more involved. Three implementations of the substitute bytes stage were created. The first implementation stored the s-box entirely in BlockRAM. The second implementation implemented the s-box using mini s-boxes. These mini s-boxes were implemented as look-up tables in distributed RAM. The final, third implementation of the substitute bytes stage implemented the mini s-boxes using simple Boolean expressions. Two implementations of the mix rows stage were also created. The first implementation performed the required Galois field multiplications using look-up tables stored in distributed RAM. In the second implementation, the Galois field multipliers were realized using Boolean expressions. All combinations of the above described implementations were then synthesized. Post place and route timing information was used in order to determine the combination which provides the most efficient Whirlpool implementation. Since the W cipher is at the core of the Whirlpool hashing algorithm, it was assumed that the most efficient Whirlpool implementation also contains the most efficient W cipher implementation. This implementation was then used in PHASH. It should be noted that only W cipher implementations which do not use BlockRAM were considered as it would not be possible to implement PHASH with more than 4 W cipher instances, using BlockRAM, on most currently available FPGAs. Post place and route timing information showed that the most efficient Whirlpool configuration proved to be the one in which the substitute bytes stage was implemented using mini s-boxes;

they were implemented using look-up tables and stored in distributed RAM. The Galois field multipliers in the mix rows stage were implemented using Boolean expressions.

The model of PHASH is highly configurable with two restrictions. The first one is that the number of W cipher instances used is required to be a power of 2. The implementation in this work also restricts the maximum number of reduction levels in PHASH to three. The structure of the PHASH model is shown in Figure 3.

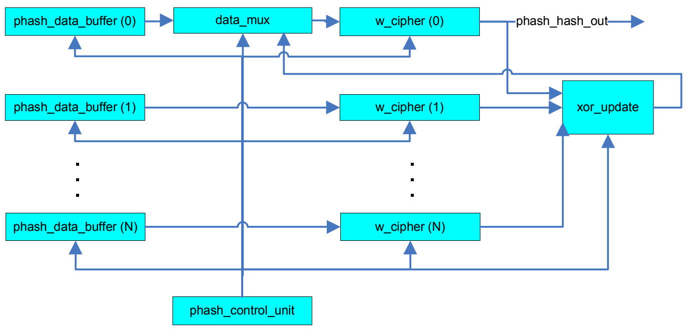

Figure 3. Block diagram of PHASH design

B. *Testing Functionality*

Throughout the implementation process many tests were performed to verify the functionality of individual components and entire systems. The VHDL testbench was responsible for reading pre-padded data to be hashed from a file, sending the data at the appropriate time, as well as comparing the obtained hash to a known value. The SHA-512 and Whirlpool test data consisted of 240 tests, starting with 20 bytes of data and incrementing by 20 bytes for each consecutive test, for a total of 4800 bytes. The PHASH test data consisted of three separate tests. The first consisted of 128 test cases. The first case ensured only a single block is hashed, and each consecutive test hashed an extra block. As a result 128 tests ranging from a single block to 128 blocks were performed. The second test consisted of hashing 256 and 257 data blocks. These block counts represent the transition between the first and second reduction levels. The final test consisted of hashing 65536 and 65537 blocks. These block counts represent the transition between the second and third reduction levels.

C. *Hardware Verification*

The available hardware platform to verify the functionality of the designs was the Xilinx ML410 development board. The ML410 is a complete development system containing a Virtex-4 FX60 FPGA as well as a multitude of additional peripherals. The Virtex-4 FX60 FPGA is a hybrid FPGA, containing not only FPGA logic, but also two PowerPC 405 cores. Several other notable features of this FPGA include 232 18-Kbit BlockRAMs, up to 20 DCMs, 25280 CLB slices, equivalent to 56880 logic cells, 128 XtremeDSP slices as well as multi-gigabit RocketIO transceivers. Additional features of the ML410 development board used in these implementations include 256 MB DDR-2 external memory, 512 MB CF card and a single UART. The features of the Virtex-4 FX60 FPGA used include CLB slices, BlockRAMs and DCMs. Xilinx ISE 9.2.04i was used in order to obtain utilization and timing information.

A total of 10 hardware systems (as described in subsection *A*) were created, two for SHA-512, six for Whirlpool and two for PHASH. The limitation for PHASH was due to the fact that the Virtex4 FX60 FPGA can accommodate PHASH with up to two W ciphers only. Table I summarizes the implementation results.

TABLE I. IMPLEMENTATION RESULTS FOR VIRTEX-4 FX60 FPGA

Algorithm	Configuration	Slices	Freq. (MHz)	Throughput (Mbps)
SHA-512	Distributed RAM	2073	106.65	1365
SHA-512	BlockRAM	1917	103.17	1321
Whirlpool	1	6605	122.09	6251
Whirlpool	2	6597	123.50	6323
Whirlpool	3	7327	105.28	5390
Whirlpool	4	7937	112.38	5754
Whirlpool	5	4833	138.96	7115
Whirlpool	6	3914	137.08	7018
PHASH 1	1 W	11010	126.71	11353
PHASH 2	2 W	16901	124.24	22264

Since all implementations achieved an operating frequency of at least 100 MHz it was possible to integrate them into a system which utilized a PowerPC processor with its communication bus operating at 100 MHz. Xilinx EDK 9.2.02 software was used in order to design and implement the final hardware systems required to test the functionality of the implementations. All the hardware verifications were successful.

D. *High Performance Design*

To exploit parallelizability of PHASH and determine its maximum possible throughput on the state-of-the-art FPGAs the VHDL model was synthesized into Virtex-5 LX330 device. For comparison purposes SHA-512 and Whirlpool were also implemented on the same platform. Table II shows the results obtained for SHA-512 and Whirlpool. It is important to note that the slice counts in Virtex-5 devices are not directly comparable to those in Virtex-4 devices. The Virtex-5 devices have less slices, however each slice contains about twice as many resources as a single Virtex-4 slice.

TABLE II. SHA-512 AND WHIRLPOOL RESULTS FOR VIRTEX-5 LX330 FPGA

Algorithm	Slices	Frequency (MHz)	Throughput (Mbps)
SHA-512	1102	142.88	1829
Whirlpool	2892	162.34	8312

It is evident that the performance optimized Whirlpool implementation greatly outperforms the SHA-512 implementation in terms of throughput. However it requires more than twice as many slices to implement. Table III shows a similar set of metrics for several PHASH implementations. For each implementation a different number of W cipher instances were included. By increasing the number of instances a significant increase in throughput was observed, at the cost of increased slice usage.

TABLE III. PHASH RESULTS FOR VIRTEX-5 LX330 FPGA

W cipher instances	Slices	Speedup Factor	Frequency (MHz)	Throughput (Mbps)
1	4469	1.00	168.07	15059
2	8031	1.92	161.29	28903
4	13537	3.69	155.04	55566
8	24427	6.70	140.85	100958
16	42362	12.13	127.39	182624

The speedup factor represents the efficiency at which the implementation scales. The throughput obtained with a single W cipher instance is used as the baseline and therefore its speedup factor is 1.00. The maximum theoretical speedup factor would be equal to the number of W cipher instances used. Looking at the results in Table II and Table III it is also evident that the PHASH implementations greatly outperform both SHA-512 and Whirlpool in terms of throughput. The PHASH implementation with a single W cipher instance requires slightly more slices than the Whirlpool implementation; however it is able to achieve nearly twice as much throughput. For a visual comparison, a bar graph of the maximum throughput of all three algorithms targeted for the Virtex-5 LX330 FPGA is shown in Figure 4. By looking at the number of slices required in order to achieve 1 Mbps of throughput both the area and operating frequency can be incorporated into a single metric. The SHA-512 implementation requires 0.60 slices per 1 Mbps of throughput, whereas the Whirlpool implementation requires only 0.35 slices per 1 Mbps of throughput.

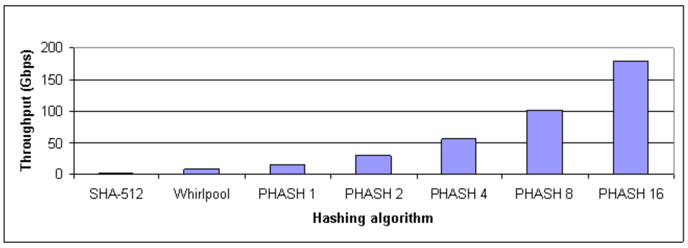

Figure 4. Maximum throughput

A PHASH implementation with a single W cipher instance requires only 0.30 slices per 1 Mbps of throughput. The remaining implementations require between 0.28 and 0.24 slices per Mbps. This shows that the PHASH implementations utilize the available slices very efficiently, even as the implementation scales.

V. CONCLUSION AND FUTURE WORK

This paper presented FPGA design and performance analysis of the SHA-512, Whirlpool and the newly proposed PHASH algorithm. The novelty of PHASH is that it is able to exploit parallelism and as a result achieve a much higher throughput than any other currently available hashing function. A PHASH implementation using only a single W cipher is able to achieve over 15 Gbps of throughput. When the number of W cipher instances is increased to 16, a throughput of over 182 Gbps is achieved. A fair comparison between SHA-512 and Whirlpool on the same FPGA implies overall better performance of Whirlpool. Several assumptions were made during the implementation of PHASH in order to facilitate the development process. Future work should include removing most, if not all, of these assumptions in order to create a less constrained implementation. The most noticeable restriction in this implementation of PHASH is that the number of W cipher instances used is required to be a power of 2. If a new implementation were to remove this restriction several more instances would be able to fit onto the Virtex-5 LX330 FPGA. Finally, the implementation in this work restricts the maximum number of reduction levels in PHASH to three. Future implementations can remove this restriction.

REFERENCES

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[2] FIPS 180-2. Secure Hash Standard, August 2002. (NIST). http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[3] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane. Optimisation of the SHA-2 family of hash functions on FPGAs. Proceedings - IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures 2006, 2006:317 – 322, 2006.

[4] F. Aisopos, K. Aisopos, D. Schinianakis, H. Michail, and A. P. Kakarountas. A novel high-throughput implementation of a partially unrolled SHA-512. Proceedings of the Mediterranean Electrotechnical Conference - MELECON, 2006, 61 – 65, 2006.

[5] P. Barreto and V. Rijmen. Whirlpool Hash Function. 2006. http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html

[6] M. McLoone, C. McIvor, and A. Savage. High-speed hardware architectures of the Whirlpool hash function. Proceedings - 2005 IEEE International Conference on Field Programmable Technology, 2005:147 – 153, 2005.

[7] P. Kitsos and O. Koufopavlou. Efficient architecture and hardware implementation of the Whirlpool hash function. IEEE Transactions on Consumer Electronics, 50(1):208 – 213, 2004.

[8] A. Kaminsky and S. Radziszowski. A case for a parallelizable hash. MILCOM 2008, San Diego, CA, USA, November 2008.

[9] I. Ahmad and A. S. Das. Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs. Computers and Electrical Engineering, 31(6):345 – 360, 2005.

[10] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Improving SHA-2 hardware implementations. Lecture Notes in Computer Science, 4249 NCS:298 – 310, 2006.

[11] L. Dadda, M. Macchetti, and J. Owen. An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512). Proceedings of the ACM Great Lakes Symposium on VLSI, pages 421 – 425, 2004.

[12] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott. Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. Information Security. 5th International Conference ISC 2002. Proceedings (Lecture Notes in Computer Science Vol.2433), pages 75 – 89, 2002.

[13] M. McLoone and J. V. McCanny. Efficient single-chip implementation of SHA-384 and SHA-512. 2002 IEEE International Conference on Field-Programmable Technology (FPT). Proceedings (Cat. No.02EX603), pages 311 – 14, 2002.

[14] N. Sklavos and O. Koufopavlou. On the hardware implementations of the SHA-2 (256, 384, 512) hash functions. Proceedings - IEEE International Symposium on Circuits and Systems, 5:153 – 156, 2003.