Multi-mode Operator for SHA-2 Hash Functions

Ryan Glabb, Laurent Imbert, Graham Jullien, Arnaud Tisserand, Nicolas Veyrat-Charvillon

Multi-mode Operator for SHA-2 Hash Functions

Ryan Glabb*, Laurent Imbert*†§, Graham Jullien*, Arnaud Tisserand§ and Nicolas Veyrat-Charvillon‡
* ATIPS Laboratories, Dept. of Electrical and Computer Engineering
† CISaC, Dept. of Mathematics and Statistics
University of Calgary, Calgary, Alberta Canada T2N 1N4
‡ LIP, École Normale Supérieure de Lyon
46 allée d'Italie, F-69364 Lyon, France
§ LIRMM, CNRS–Univ. Montpellier 2, UMR, 5506
161 rue Ada. F-34392 Montpellier, France
glabb@atips.ca nicolas.veyrat-charvillon@ens-lyon.fr

***Abstract*— We propose an improved implementation of the SHA-2 hash family to include a multi-mode of operation with minimal latency and hardware requirements over the entire operator.**
The multi-mode architecture presented is able to perform either a SHA-384 or SHA-512 hash or to behave as two independent SHA-224 or SHA-256 operators. We also demonstrate that our architecture achieves performance comparable to separate implementations while requiring much less hardware. This could be useful for a server running multiple streams or in parallel PRNG generation.

I. Introduction

Cryptographic hash functions [1] are a fundamental tool in modern cryptography, used mainly to ensure data integrity when transmitting information over insecure channels. Hash functions are also used for the implementation of digital signature algorithms, keyed-hash message authentication codes and in random number generator architectures.

In 2004, an algorithm was discovered [6] that decreased the resistance to collisions of SHA-1 (Secure Hash Algorithm) [7], the most popular hash function so far, reducing the number of necessary computations from 2^{80} to 2^{69} and putting it below the accepted security threshold for high-security operations. Since then, the SHA-2 family of hash functions [8], developed by the National Institute of Standards and Technology (NIST), has become the new standard.

Due to their complexity and limited lifespan, cryptographic primitives are generally implemented in software on general purpose processors. However, many secure cryptographic algorithms such as AES (Advanced Encryption Standard) and SHA-1 were designed to be implemented in hardware, and are drastically less efficient in terms of speed when coded in software [1]. In terms of hardware implementations, the two principal technologies are Application-Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGAs). Due to their ease of use and lower cost, we have targeted FPGAs from the Virtex and Spartan3 families for the prototyping phase and for the synthesis results reported in this paper.

The aim of this work is to show the advantages of using reconfigurable hardware operators to include a multi-mode of operation with the SHA-2 hash family, using shared resources on a single chip-set.

II. SHA-2 hash standard

Throughout this paper, we will follow the definitions and notations used in the SHA-2 specification [8]. This specification details all steps of the hash algorithms and constants used in the computation. We will only report on the relevant parts useful for the understanding of implementation and optimization issues that are considered in this paper.

The SHA-2 hash standard specifies four secure hash algorithms, SHA-224, SHA-256, SHA-384, and SHA-512. All four of the algorithms are iterative, one-way hash functions that can process a message to produce a hashed representation called a *message digest*. Each algorithm can be described in two stages: preprocessing and hash computation. Preprocessing involves preparing the message through padding, parsing the padded message into m-bit blocks, and setting any initialization values to be used in the hash generation. The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations, to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest.

A message M of length l to be hashed is processed by blocks of m bits. Each block is divided in 16 w-bit words for computation, the word-size w depending on the algorithm.

The most important difference between the four algorithms is the size of the message digest. Additionally, the algorithms differ in terms of the size of the blocks and words of data that are used during hashing (table I).

Algorithm	Word	Message size	Block	Digest	Security
SHA-224	32	$< 2^{64}$	512	224	112
SHA-256	32	$< 2^{64}$	512	256	128
SHA-384	64	$< 2^{128}$	1024	384	192
SHA-512	64	$< 2^{128}$	1024	512	256

TABLE I
Secure hash algorithm properties

A. Preprocessing

This process consists of three steps: 1) padding the message M; 2) cutting the padded message into blocks; and 3) setting the initial hash value, $H^{(0)}$. The purpose of padding is to ensure that the padded message is a multiple of 512 or 1024 bits.

B. Hash computation

1) Hash Computation of SHA-256 and SHA-512: We will describe SHA-256 and SHA-512 together, in order to stress their numerous similarities.
For SHA-256, $w = 32$ and $t_{max} = 63$, and for SHA-512, $w = 64$ and $t_{max} = 79$.

Both algorithms use:

- a message schedule of $t_{max} + 1$ w-bit words
- eight working variables of w bits each
- a hash value of eight w-bit words.

After preprocessing is completed, each message block, $M^{(1)}, M^{(2)}, \cdots, M^{(N)}$, is processed in order:
(Additions (+) are all performed modulo 2^w)
For $i = 1$ to N {

- Prepare the message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{Alg\}}(W_{t-2}) + W_{t-7} & \\ +\sigma_0^{\{Alg\}}(W_{t-15}) + W_{t-16} & 16 \le t \le t_{max} \end{cases}$$

- Initialize the eight working variables, a, b, c, d, e, f, g, and h, with the $(i-1)$st hash value:

$$a||b||c||d||e||f||g||h = H^{(i-1)}$$

- For $t = 0$ to t_{max} rounds, perform {

$$T_1 = h + \Sigma_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \Sigma_0^{\{256\}}(a) + Maj(a, b, c)$$

$$a||b||\cdots||h = (T_1 + T_2)||a||b||c||(d + T_1)||e||f||g$$

}
- Compute the ith intermediate hash value $H^{(i)}$:

$$H^{(i)} = (a + H_0^{(i-1)})||(b + H_1^{(i-1)})||\cdots||(h + H_7^{(i-1)})$$

}

After repeating those steps a total of N times (after processing $M^{(N)}$), the resulting message digest of M is

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}||H_7^{(N)}$$

2) SHA-224 and SHA-384: The SHA-224 algorithm is identical to SHA-256, with the exception of using a different initial hash value and truncating the final hash value to the left-most 224 bits. Similarly, the SHA-384 algorithm is identical to SHA-512, except for the different initial hash value and truncating of the final hash value to the left-most 384 bits.

III. Merging of the SHA-2 family

Merging the SHA-2 family of functions into a single architecture is much more efficient than implementing separate operators for each hash algorithm. For example, in [9], SHA-256, SHA-384 and SHA-512 were each implemented using a separate computational unit. During the computation of SHA-256, the left half of the 64-bit datapath is unused and held to zero in that implementation.

Our multi-mode SHA-2 operator has been designed to optimize the hardware efficiency. It is able to run either a hash function working on $w = 64$-bit words (SHA-384 or 512), or two $w = 32$-bit functions (SHA-224 or 256) running concurrently. When running in split mode, the operator can be considered as two separate operators each running a $w = 32$-bit hash.

A. Sharing the datapath

1) Comparison between the hash functions: The hash functions of the SHA-2 family share many similarities. We can classify them into two categories: the $w = 32$ bit functions, SHA-224 and SHA-256, and the $w = 64$ bit functions, SHA-384 and SHA-512. Given their respective word sizes, a large part of the datapaths is identical, and other parts can be shared efficiently:

- The padding is identical with regard to the respective word sizes. A message of length l is processed by blocks of 16 words, and a "1" is appended at the end, followed by as many zeros (k) as necessary in order to have $l + 1 + k \equiv (14 \cdot w) \bmod (16 \cdot w)$. A 2-word binary representation of l is then appended.
- The message scheduler is identical for all hash functions, except for the σ functions which are different depending on the word size.
- The definition of the initial hash value $H^{(0)}$ allows its implementation to be shared between the algorithms. That is, the left halves of the SHA-512 words of $H^{(0)}$ are the words of $H^{(0)}$ for SHA-256. Similarly for SHA-384, the right halves of the words of $H^{(0)}$ are the words for $H^{(0)}$ for SHA-224. For example for $H_0^{(0)}$:

  SHA-224 $H_0^{(0)} =$ **c1059ed8**
  SHA-256 $H_0^{(0)} =$ **6a09e667**
  SHA-384 $H_0^{(0)} =$ cbbb9d5d**c1059ed8**
  SHA-512 $H_0^{(0)} =$ **6a09e667**f3bbc908
- In the functions defined by the SHA-2 standard, only Ch and Maj are identical for all algorithms. The σ and Σ operations are different, although they are based on the same idea, that is a bitwise XOR of three different rotations/shifts of the input value, but the rotate/shift values differ and thus cannot be shared. Since there are only two different sets of functions (one for $w = 32$ and another for $w = 64$), they are both hard-wired with selection between the two using a MUX, which is a lower hardware cost solution than the use of a generic structure (barrel rotate/shifter).

- The round constants are the same for equal word sizes, and the value of K_t for $w = 32$ is identical to the left half of the corresponding $w = 64$ constant. For example:
  SHA-224/256 $K_0 =$ `428a2f98`
  SHA-384/512 $K_0 =$ **428a2f98**`d728ae22`
- The round computation and the intermediate value definitions are the same for all SHA-2 algorithms, although the number of rounds differs depending on the word size. Only 64 rounds are performed for $w = 32$-bit hashes, and 80 for $w = 64$-bit hashes.

B. Physical sharing of the hardware

Our multi-mode architecture fits a hash function of two 32-bit words into the same datapath as that used for a single 64-bit hash. We note that α and β are the two 32-bit word hash functions using the left and right parts of the datapath, respectively, used to compute the 64-bit hash γ.
The physical sharing is accomplished by considering all operations on words for γ as two separate 32-bit operations on α and β, where the dependancies between the two halves are inhibited depending on the running mode. For example, each register of the message scheduler can be seen either as a 64-bit register or as 2 independant 32-bit registers running in parallel (Figure 1). This involves no hardware overhead since the left and right halves are independant regardless of the operator mode.
When an addition modulo 2^w is performed, there is a carry propagation between the right and left parts for γ that must be inhibited when computing two adjacent modulo 2^{32} additions α and β. Beside the small logic overhead, control parts are duplicated in order to allow α and β to run concurrently as well as in parallel.

Figure 1 shows the modifications required to the standard carry propagate adder, available on the FPGA, that allow either one modulo 2^{64} addition or two concurrent modulo 2^{32} additions to be performed.

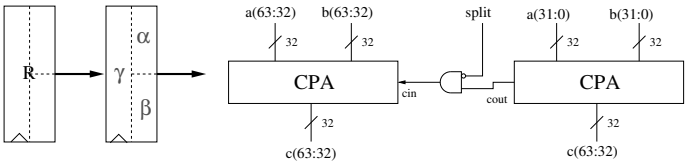

Fig. 1. The registers can either be considered as one 64-bit or as two concurrent 32-bit registers (left). We also introduce a 64-bit / 2×32-bit selectable modular adder (right)

1) Padder: In the multi-mode version of the padder, the word counter has been modified in order for it to be used as either two separate 64-bit counters, or as a single 128-bit counter. This implies a rather complicated management of the carry since the last 4-bits (resp. 5-bits) of each message length for a $w = 32$-bit (resp. $w = 64 - bit$) hash are given by the input to our system *Bit_valid*. If the operator works in split mode, one carry used in the word counter must be discarded and *Bit_valid* used for the lower bits in the message length of α.

2) Message Scheduler: The message scheduling for SHA-256 and SHA-512 is the same except for: the word size which is doubled; the σ_0 and σ_1 functions which consist of wiring; and the number of rounds that does not affect the logical structure of the scheduler. The figure below illustrates the multi-mode computation of W_t. MUXes select data paths for each mode, and the previously introduced split adders are used to perform the modulo 2^w additions.

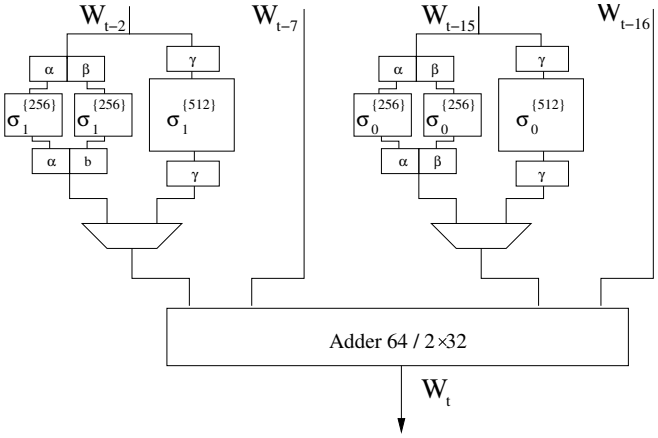

Fig. 2. Multi-mode implementation of the W_t computation

3) Round constants unit: Since a dual-port 32-bit RAM block is used to compute the SHA-384/512 64-bit round constants, it can also be used, at the cost of some overhead logic in the address input, to provide two different 32-bit round constants as well as two concurrent SHA-244/256 hashes.
In order to ensure the same latency properties as in the separate architectures, some logic has to be added to ensure the correct initialization of the computation when the mode is changed, since the constant unit must output either $K_{0\gamma}$ or $K_{0\alpha}||K_{0\beta}$ depending on the new mode.

4) Round computation unit and $a, b, \cdots, h$ variables: The equations for computing the new values of variables $a, b, \cdots, h$ are the same for all hash functions of the SHA-2 family, with appropriate changes relating to the relative word sizes and with the exception of the Σ functions. The only modification of the round computation unit for the multi-mode version therefore consists in using the split adders and implementing both Σ^{512} and Σ^{256} operators for each Σ function, as was done for the message scheduler.

5) Intermediate hash: The initial hash value, $H^{(0)}$, is selected through additional logic that takes advantage of the similarities between the values of all algorithms, providing for every modes. The computation of a new intermediate hash is performed using the split adders.

6) Analysis: The multi-mode architecture shares the same properties as separate architectures for the operators in terms of latency and speed. As discussed earlier, it is possible to improve the overall throughput by segmenting the critical path.

Reference Architecture	Slices	Freq (MHz)	Cycles per block	Throughput (Mb/s)	Throughput/Area (Mb/s/slice)
[9] SHA-256	*2120	83	81	262	0.123
[9] SHA-384	*3932	74	97	293	0.075
[9] SHA-512	*4474	75	97	396	0.089
[10] SHA-384/512	*5828	38	pipelined	479	0.082
[9] SHA-256/384/512	*4768	74	81/97	233.9/390.6	0.049/0.082
Proposed Architectures					
SHA-224	1297	77	64(+2)	269.5	0.208
SHA-256	1306	77	64(+2)	308	0.236
SHA-224/256	1260	69	64(+2)	276	0.219
SHA-384	2581	69	80(+2)	331	0.128
SHA-512	2545	69	80(+2)	442	0.174
SHA-384/512	2573	66	80(+2)	422	0.164
****Multi-mode SHA-2**	2951	50	64/80(+2)	2×200/320	0.136/0.108

TABLE II

*1 CLB=2 SLICES FOR VIRTEX, TARGET:VIRTEX 200/400XCV, **MAX MULTI-MODE THROUGHPUT IS 400/640MBPS=2×200/320MBPS

IV. IMPLEMENTATION RESULTS

This section summarizes our implementation results using Synplify Pro as a synthesis tool. The criteria considered are FPGA resources (slices), maximum throughput (Mb/s) and their ratio in Mb/s/slice.

Every hash function of the SHA-2 family was synthesized as a stand-alone operator (224,256,384 and 512), or merged by word operating size (224/256 or 384/512), and we also give our results for the multi-mode architecture which is capable of all modes or running two independent 32-bit(SHA-224/256) operators simultaneously.

A. Comparison with published implementations

We now compare our architectures with previously published stand-alone and multi-mode SHA-2 implementations [9], [10]. (See Table II)

The focus of [9] was to implement SHA-256, 384 and 512 in a single operator using the Virtex XCV200 as a target. Reference [10] discusses a pipelined approach to a single chip SHA-384/512 architecture. In both of these designs, 16 to 32 clock cycles are required for the padder to process an input message block before computation begins. This is avoided in our system thanks to an 'on-the-fly' padder that allows a throughput increase of up to 25 percent compared to [10].

Additionally, due to some pre-computation techniques, we are able to achieve clock frequencies significantly higher than [10] and slightly less than [9] while at the same time significantly reducing the hardware costs.

Our multi-mode operator, in particular, uses considerably fewer resources compared to the multi-mode 256/384/512 implementation [9] with 2951 slices compared to 4768 slices and has a much better throughput to area ratio.

V. CONCLUSION

In this paper, we have introduced a concurrent SHA-2 operator which optimizes the datapath when a 64-bit SHA-2 hash mode is supported and removes all unnecessary latencies. The proposed multi-mode architecture is able to perform a single SHA-384 or SHA-512 hash function or to behave as two independant computations of SHA-224 or SHA-256 hash functions with minimal hardware overhead. We demonstrated the benefit of integrating a concurrent 32-bit mode when a 64-bit hash is to be supported.

Additionally, the new architecture achieves a performance comparable to previously published separate implementations of these functions while requiring much less hardware. Most importantly, all of the new implementations presented in this paper are more efficient than previously published implementations when considering the throughput-to-area ratio.

VI. ACKNOWLEDGEMENTS

This work was financially supported through iCORE(Informatics Circle of Research Excellence), NSERC (Natural Sciences and Engineering Research Council of Canada),CMC and an ACI grant from the French ministry of Research and Education.

REFERENCES

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[2] R. L. Rivest, "The MD5 message-digest algorithm," Internet informational RFC 1321, Apr. 1992.

[3] Dobbertin, Bosselaers, and Preneel, "RIPEMD-160: A strengthened version of RIPEMD," in *IWFSE: International Workshop on Fast Software Encryption, LNCS*, 1996.

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

[5] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD," Cryptology ePrint Archive, Report 2004/199, p. 4, 2004.

[6] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," Shandong University, Shandong, China, Technical report, June 2005.

[7] National Institute of Standards and Technology, *FIPS PUB 180-1: Secure Hash Standard*. Gaithersburg, MD, USA: NIST, Apr. 1995. [Online]. Available: http://www.itl.nist.gov/fipspubs/fip180-1.htm

[8] ——, *FIPS PUB 180-2: Secure Hash Standard*. Gaithersburg, MD, USA: NIST, Aug. 2002. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[9] N. Sklavos and O. Koufopavlou, "Implementation of the SHA-2 hash family standard using FPGAs," *The Journal of Supercomputing*, vol. 31, no. 3, pp. 227–248, Mar. 2005.

[10] M. McLoone and J. McCanny, "Efficient single-chip implementation of sha-384 & sha-512," *IEEE Proc., International Conerence on Field-Programmable Technology (FTP)*, pp. 311–314, 2002.