Hardware implementation of a multi-mode hash architecture for MD5, SHA-1 and SHA-2

Sylvain Ducloyer, Romain Vaslin, Guy Gogniat, Eduardo Wanderley
LESTER CNRS FRE 2734
Lorient, FRANCE
name@univ-ubs.fr

***Abstract*— In this paper, we propose a unified architecture adapted to the field of embedded systems which combines commonly used hash algorithms in a single design in order to reduce area requirements and optimize the maximum frequency. We present an implementation of three hash functions: MD5 [8], SHA-1 [9] and SHA-2 [10]. Many similarities exist between these algorithms which help us to move towards a unique architecture. The design was implemented on an Altera Stratix II device. It only requires 3311 ALUTs and operates at a frequency of 93 MHz which provides a throughput of 567 Mbps for MD5/SHA-2 and a 476 Mbps throughput for SHA-1 (including padding operations).**

I. Introduction

Cryptography field has been strongly active these last years, essentially in consequence of an increasing need for security issues. For example, most of the digital communications in mobile or network applications require some protections against potential threats (virus, worms and hackers).

Hash functions are well known cryptographic solutions used to guarantee message integrity (see Figure 1). Several hash algorithms exist, however MD5 and SHA-1 are currently the most used. Many processor-based solutions have already been proposed but they are not adapted to the specific constraints of embedded systems (efficiency, area, power consumption).

There are different ways to implement those kinds of protection in a system. The first one is using a coprocessor for each algorithm. In this case the area overhead will be significant. The second solution would be to use a reconfigurable hardware device like FPGA and to dynamically reconfigure the FPGA for switching from one algorithm to another. It means that the system will need extra memory to store the different configurations for the FPGA.

The last alternative is a configurable coprocessor which provides a few different configurations for those algorithms. This solution provides a trade-off with the two previous solutions presented. Due to the common features of MD5, SHA-1 and SHA-2, it is possible to minimize the area required to implement these algorithms, to limit the storage (one FPGA context instead of three) and to provide flexibility. For SHA-2 we focus our work on SHA-224/256 version because they are based on a 32 bits datapath.

In this paper, we present the first hardware architecture for integrity checking based on three hash functions MD5, SHA-1 and SHA-2. The paper is organised as follows: in section II, hash functions are presented with hash algorithm descriptions. The proposed architecture is detailed in section III. Section IV focuses on throughput, area results and maximum frequency. Finally, related work is presented in order to compare our architecture with previous works in section V.

II. Hash functions

Hash functions are designed to obtain a signature depending on an initial message. In addition, the functions are one-way-function in order to generate a unique fixed-length bit vector as an output. This signature is then sent with the message. The addressee will check the message integrity by comparing the received signature with the one he will compute based on the received message (Figure 1).

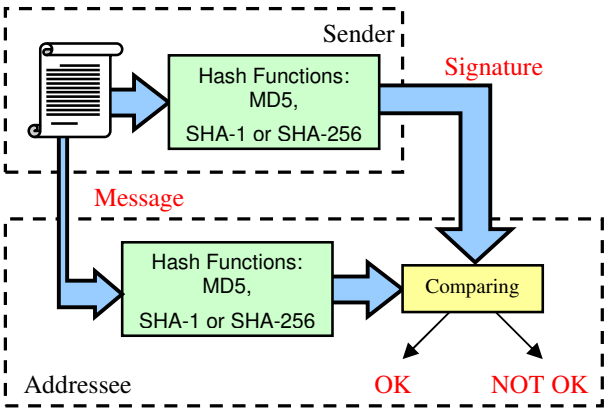

Fig.1. Hash Function

In our case, we combine three hash functions MD5, SHA-1 and SHA-224/256 in a configurable global architecture. In order to obtain a signature corresponding to a message, we perform several steps: a pre-processing of the message and then the hash computation. The 3 selected algorithms rely on the same way to obtain the signature. The pre-processing phase is the same for all of them. The hash computation differs for each algorithm.

A. *Pre-processing of the message*

The first step is padding. The goal of padding is to obtain a message that will have a size modulo 512 bits. It needs be modulo 512 bits because it is the size of the input of these algorithms. The last 64 bits of the padded message are reserved to store the size of the initial message. Figure 2 shows an example of a padded message. To fill the gap between the end of the original message and the last 64 bits, a bit set to 1 is inserted with a succession of bits set to 0.

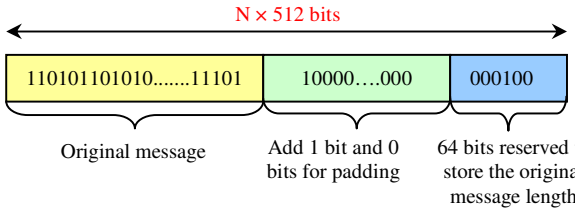

Fig.2 Padding

The second step is to fill the last 64 bits with the size of the message. In some cases, there are not enough bits left to store the length (an example is a message of 1008 bits). Then a new 512 bits word will be created. This word will be filled with some 0 until the space reserved to store the size is reached.

As soon as the pre-processing ends, the message is ready to be hashed. As it is said previously, the pre-processing phase is the same for MD5, SHA-1 and SHA-2.

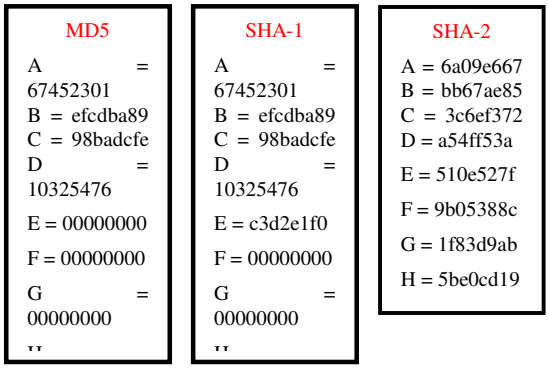

Fig.3 Initialization of buffers values

B. *Hashing of the message*

It is the heart of a hash algorithm. During this step, the computation part of the algorithm will be done. This module consists of four processing rounds of 16 steps each for MD5 and SHA-2; 20 steps each for SHA-1. In this section, we detail the different ways the signatures are computed. It will help to extract the similarities between the 3 algorithms in order to move towards a unified hardware architecture.

Each 512 bits word composing the message will pass through the algorithm. At the beginning some buffers are initialized (see Figure 3 for values). When the first 512 bits word is hashed, the second word is hashed but the buffers are not reset to the initial values. The result from the previous 512 bits word is used. The process is iterated until all the 512 bits words have been hashed. In all the following text the letter W will mean a 32 bits word from a 512 bits word. If an index appears, it means for example the second word of the 512 bits word (W_2). K is a different constant for every step t. The values of K can be found in the official algorithm RFC [8] [9] or FIPS [10].

Concerning the SHA-2, as said previously we have focused our work only on the SHA-224/256 version in order to simplify the architecture. Indeed, the SHA-512 will require a 64 bits architecture.

1) *MD5*

MD5 (Message Digest) was developed by Rivest in 1991 [8]. It is a hash function which produces a digital signature of 128 bits for an arbitrary-length message. Each round the operations of Equation 1 are performed. Table 1 summarizes all the values of the parameters.

$$T = B + ([A + Fmd5(B, C, D) + W_t + K_t] << S)$$
$$A = D$$
$$B = T$$
$$C = B$$
$$D = C$$

Eq.1 Hash operations of MD5

0 < t < 15 → Round 1
Fmd5 (B,C,D) = (B and C) or ((not B) and D)
S = 7, 12, 17, 22
16 < t < 31→ Round 2
Fmd5 (B,C,D) = (B and D) or (C and (not D))
S = 5, 9, 14, 20
32 < t < 47 → Round 3
Fmd5 (B,C,D) = B xor C xor D
S = 4, 11, 16, 23
48 < t < 63 → Round 4
Fmd5 (B,C,D) = C xor (B or (not D))
S = 6, 10, 15, 21

Tab.1: Parameters for MD5

2) SHA-1

SHA-1 (Secure Hash Algorithm) was developed by NSA (National Security Agency) in 1993 [9]. It is a hash function which produces a digital signature of 160 bits for an arbitrary-length message.

$$T = (A << 5) + Fsha1(B,C,D) + E + W_t + K_t$$
$$A = T$$
$$B = A$$
$$C = \text{left rotation of 30 bits on } B$$
$$D = C$$
$$E = D$$

Eq.2: Hash operations of SHA-1

0 < t < 19 → Round 1
$Fsha1(B,C,D) = (B \text{ and } C) \text{ xor } ((\text{not } B) \text{ and } D)$

20 < t < 39 → Round 2
$Fsha1(B,C,D) = B \text{ xor } C \text{ xor } D$

40 < t < 59 → Round 3
$Fsha1(B,C,D) = (B \text{ and } C) \text{ xor } (B \text{ and } D) \text{ xor } (C \text{ and } D)$

60 < t < 79 → Round 4
$Fsha1(B,C,D) = B \text{ xor } C \text{ xor } D$

Tab. 2: Parameters for SHA-1

W_t is a 32-bits word of the message. However if $t < 16$, W_t is the t^{th} 32-bits word of the message block, else if $t \geq 16$, W_t is calculated according to Equation 3.

$$Wt = (W_{t-3} \text{ xor } W_{t-8} \text{ xor } W_{t-14} \text{ xor } W_{t-16}) << 1$$

Eq.3: Computation of Wt for t > 16

3) SHA-2

SHA-2 (Secure Hash Algorithm) was developed by NSA in 2000 [10] because collisions have been found in SHA-1 which means that the algorithm is not secured enough. The SHA-224/256 version provides a digital signature of 224/256 bits for an arbitrary-length message. SHA-2 uses different shifts and constants, but its structure is almost identical to SHA-1.

$$T1 = H + \sum_1(E) + Ch(E, F, G) + W_t + K_t$$
$$T2 = \sum_0(A) + Maj(A, B, C)$$
$$A = T1 + T2$$
$$B = A$$
$$C = B$$
$$D = C$$
$$E = D + T1$$
$$F = E$$
$$G = F$$
$$H = G$$

Eq.4: Hash operations of SHA-2

$$Ch(E, F, G) = (E \text{ and } F) \text{ xor } ((\text{not } E) \text{ and } G)$$
$$Maj(A,B,C) = (A \text{ and } B) \text{ xor } (A \text{ and } C) \text{ xor } (B \text{ and } C)$$
$$\sum_0(A) = (A >> 2) \text{ xor } (A >> 13) \text{ xor } (A >> 22)$$
$$\sum_1(E) = (E >> 6) \text{ xor } (E >> 11) \text{ xor } (E >> 25)$$

Eq.5: Logical operations of SHA-2

Like in SHA-1, there are some operations to obtain W_t. If $t < 16$, W_t is the t^{th} 32 bits word of the message block. Else if $t \geq 16$, W_t is computed according to Equation 6.

$$W_t = \sigma1(W_{t-2}) + W_{t-7} + \sigma0(W_{t-15}) + W_{t-16}$$
$$\sigma0(W_t) = (W >> 17) \text{ xor } (W >> 19) \text{ xor } SHR^{10}(W)$$
$$\sigma1(W_t) = (W >> 7) \text{ xor } (W >> 18) \text{ xor } SHR^{3}(W)$$

Eq.6: Computation of Wt for t > 16 *SHR stands for shift right*

III. HARDWARE ARCHITECTURE

There are many algorithmic similarities between MD5, SHA-1 and SHA-2. We have attempted to study the algorithms in order to implement them in a single design while minimizing the required resources and optimizing the throughput.

Figure 4 shows the unified architecture we propose. The architecture is divided in 6 blocks. The PADDING and FIFO blocks are not configurable because for all the algorithms the execution will be the same (see II-1). Concerning the BUFFER SCHEDULER, MESSAGE SCHEDULER and MACROFUNCTION, those blocks can be configured depending on the selected algorithm. The ROM stores all the constants used by the algorithms. With our solution we should limit the logic amount necessary to implement the 3 algorithms due to the architectural recovery between these algorithms.

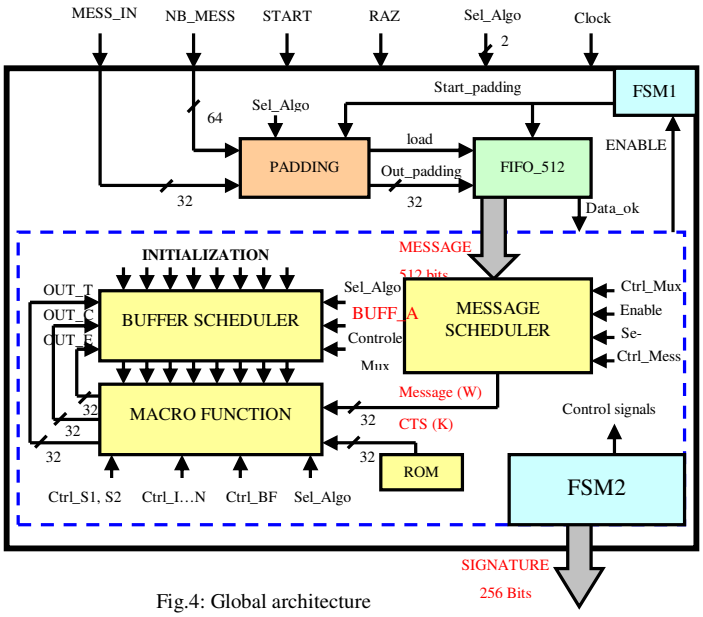

Fig.4: Global architecture

A. *Padding and storage*

Padding enables to prepare the message for the hash as presented in section II – 1. After the pre-processing step, the words are stored in a 512 bits FIFO. The architecture of padding and storage is designed according to Figure 5.

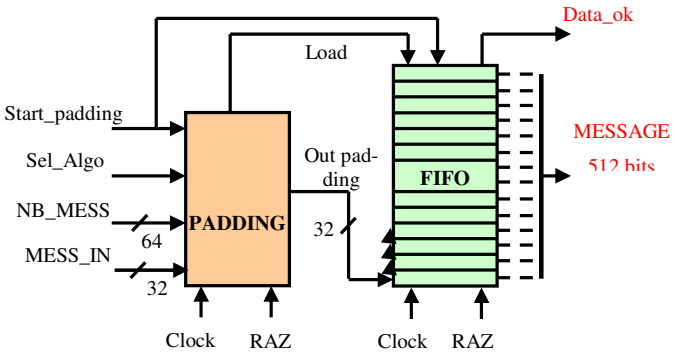

Fig.5: Details about PADDING and FIFO blocks

B. *Message scheduler*

This structure enables to select a 32 bits word in the word of 512-bits according to hash algorithms and the current executed round.

As presented in section II – 2, for MD5 no computation is required to get the 32 bits message to send for hashing. For SHA-1 and SHA-2, Equations 3 and 6 are implemented in the MESSAGE COMPUTATION block to obtain the computed value of the message. Figure 6 gives an overview of the MESSAGE SCHEDULER architecture. There is a large need of logic to control the system inputs. It helps to manage the configuration depending on the current algorithm running and the current round being executed.

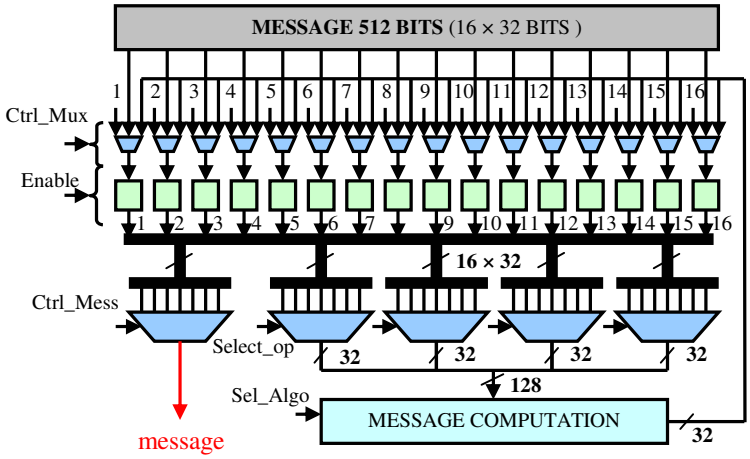

Fig.6. Message Scheduler

C. *Buffer scheduler*

This function only enables a simple shift according to the algorithm selected. These buffers are used in all the computation process of the algorithm. At the beginning they will be initialized by the values presented in Figure 3.

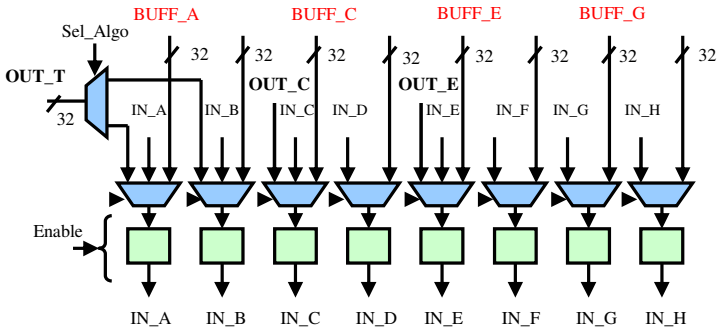

Fig.7. Shift registers for the BUFFER SCHEDULER

D. *Macro function*

It is the heart of the architecture; indeed all calculations according to the algorithms (MD5, SHA-1 and SHA-2) are done by this architectural block. In this part we have extended the proposition of [6]. As the authors had limited their work to MD5 and SHA-1. Our proposition gives us the possibility to choose between three algorithms. The first step is to extract the computation similarities. In section II – 2, the logic equations to be performed with the different buffers (Table 1 and 2, Equation 5) are detailed.

	MD5	SHA-1	SHA-2
X xor Y xor Z	x	x	
(X and Y) xor ((not X) and Z)	x	x	x
(X and Z) xor (Y and not(Z))	x		
Y xor (X or not(Z))	x		
(X and Y) xor (X and Z) xor (Y and Z)		x	x

Tab.3: Logical equations recovery between algorithms

Based on Table 3, we can extract the combinational function Fcomb (Equation 7) which contains all the logical equations required to be able to perform the computation for the 3 algorithms.

$$Fcomb(X,Y,Z) = \begin{cases} (X \text{ and } Z) \text{ or } (Y \text{ and } (\text{not } Z)) \\ X \text{ xor } Y \text{ xor } Z \\ Y \text{ xor } (X \text{ or } (\text{not } Z)) \\ (X \text{ and } Y) \text{ xor } ((\text{not } X) \text{ and } Z) \\ (X \text{ and } Y) \text{ xor } (X \text{ and } Z) \text{ xor } (Y \text{ and } Z) \end{cases}$$

Eq.7: combinational function

The second step toward a unified architecture is to propose a macro function (Tcomb) based on the Fcomb. Equation 8 shows the macro function. Table 4 summarizes the values of the function parameters depending on the algorithm running.

$$Tcomb = I + [\ ((\ J << S1\ ) + Fcomb(X, Y, Z) + \sum_1 (N) + W + K\ ) << S2\ ]$$

Eq.8: Macro function

For SHA-2, we need to have a second dedicated combinational function (Equation 9) in order to have a result for T2 in Equation 4.

$$Tcomb2 = \sum_0 (A) + Maj(A, B, C\ )$$

Eq.9: Dedicated SHA-2 function

	MD5	SHA-1	SHA-2
I	B	E	H
J	A	A	0
S1	0	5	0
Fcomb (X,Y,Z)	(B,C,D)	(B,C,D)	(E,F,G)
Σ_1(N)	0	0	E
W	W_t	W_t	W_t
K	K_t	K_t	K_t
S2	S	0	0

Tab.4: Parameters values depending on the algorithm

Figure 8 gives an overview of the macro function architecture. The main output is OUT_T which is the Tcomb value. OUT_C is the result for BUFFER C with SHA-1. OUT_E provides the result of Tcomb2.

E. Unified architecture features

All architectural blocks described previously need to be associated with state machines to manage all the internal signals. FSM2 (see Figure 4) manages the signal for the hash part. FSM1 manages the activation of the PADDING and FIFO_512 blocks. Thus, in this top-level architecture, we can realize three hash functions MD5, SHA-1 and SHA-2 thanks to the Sel_Algo signal. Moreover, the proposed architecture allows a pipelined implementation. Indeed, during hash computation, the system can achieve padding for the next 512 bits word. The latency of the pre-processing is saved for each 512 bits word. Note that, this padding penalty will always be present at the beginning of the message hashing.

The total latency to hash a 512 bits word depends on the chosen algorithm. 66 cycles are required for MD5 and SHA-2. With these 2 algorithms, the architecture performs one round per clock cycle. It is the same with SHA-1 but it requires 82 cycles as there are about 80 rounds to be performed. 18 more cycles are necessary to do the pre-processing on a 512 bits word before starting to compute the hash. As said before, this latency can be overlapped by the hash computation in the case of a message with a size bigger than 512 bits.

The last point to notice concerns the potential weakness of our proposition. Due to the specificity of our proposition which mainly relies on the recovery between the 3 algorithms; we might expect to have a longer critical path. Indeed, the solution implements 3 algorithms. It means that if we compare the critical path with a dedicated application, the critical path should be longer and the design may operate at a lower frequency. It may limit the throughput of the architecture. The next section presents further details concerning this last point.

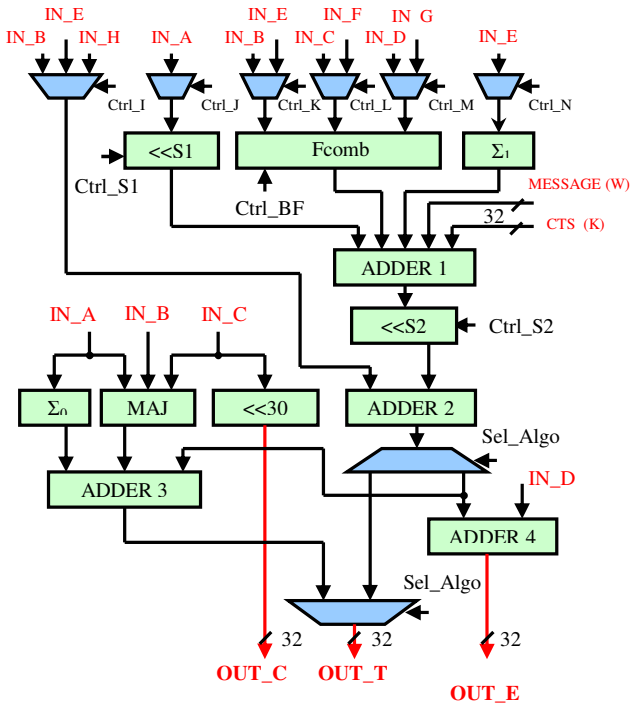

Fig.8: Architecture of the macro function

IV. RESULTS

The proposed architecture has been synthesized and implemented on an Altera Stratix II - EP2S60F672C3 device. We obtained a maximum achieved operating frequency of 93 MHz for our architecture. The throughputs are 567 Mbps for MD5 and SHA-2, and 476 Mbps for SHA-1. The throughput is calculated with Equation 10. All the throughput values presented in Table 5, have been obtained for 512 bits messages. It means that the time to perform padding and so on is included in the Number of clock cycles per block value. So the throughput value would be higher if the values were calculated for a 1024 bits word. In this case due to the way we have designed our architecture the pre-processing of the second 512 bits word would have been overlapped by the hashing operations of the first 512 bits word.

Target Device	Hash Functions	Area	Max Freq (MHz)	Throughput (Mbps)	Efficiency Mbps/area unit
Dedicated MD5 core	**MD5**	**676**	**90,6**	**702.83**	**1.03**
Dedicated SHA-1 core	**SHA-1**	**1034**	**111.33**	**695.13**	**0.662**
Dedicated SHA-2 core	**SHA-2**	**1380**	**99.53**	**772.11**	**0.516**
Non optimized Three hash core	**MD5, SHA-1, SHA-2**	**3090**	**90.6**	**702 MD5, SHA-2, 565 SHA-1**	**0.22/0.22/0.18**
Optimized Multi-hash core	**MD5,SHA-1, SHA-2**	**1662**	**93.04**	**721 MD5, SHA-2, 580 SHA-1**	**0.43/0.43/0.34**
Multi-hash cores:					
Virtex-II 2V2000-6 [6]	MD5, SHA-1	1882	77	602 MD5, 485 SHA-1	0.31/0.25
Virtex-II XC2V400 [11]	SHA-224/256	1260	69	276	0.21
Dedicated cores:					
EP1K100QC208 [3]	SHA-1	1622	43.08	268.99	0.165
Virtex-II XC2V4000-6 [4]	MD5	647	75.5	586	0.9
Virtex-II 2V3000 [5]	MD5	1369	60.2	467.3	0.34
Virtex-II 2V3000 [5]	SHA-1	1550	144.1	899.8	0.58
ASIC 0.18 µm [7]	MD5	16.000 gates	145	1140	-
ASIC 0.13 µm [7]	MD5	10.332 gates	133.3	1004	-

Tab.5: Relevant figures about our architecture and other existing solution (For xilinx area unit is slice and ALUT for Altera)

$$\text{Throughput} = \frac{\text{Block size} \times \text{Max frequency}}{\text{Number of clock cycles per block}}$$

Eq.10: Equation to obtain the throughput of architecture

The first part of Table 5 gives a summary of relevant figures about our work. The optimized multi-hash core is the architecture which was presented in the previous sections. It clearly appears that the work done on the recovery of the algorithm gives very good results for the area of the design. The area is about 2 times less important than the non optimized architecture with 3 hash cores. Furthermore, we also provide figures of dedicated cores we developed in order to compare the gain of our proposition. The MD5 core is a quite small core compared to the SHA-1 and SHA-2 ones. The overhead added by the possibility to manage 3 algorithms with our architecture seems to be negligible as the size of a SHA-2 core is around 1380 slices.

As discussed in section III – 5, due to the longer critical path, the frequency results are not always in favour of our proposition. There are some losses compare to SHA-1 and SHA-2 dedicated cores. But this loss is balanced by the gain in area. Some differences may also come from the technological difference between FPGA (ALTERA, Xilinx).

Moreover, the loss concerning the throughput is very negligible. In the worst case (SHA-1), the throughput goes from 695 Mbps to 565Mbps (-20%). This loss was expected because the throughput is linked to the frequency of the system.

Indeed, the number of cycles required to obtain the result is still 66 cycles but as the frequency of the optimised core is lower than the dedicated core, the throughput decreases. Concerning the throughput with MD5 and SHA-2 the values are noticeably the same. In addition, the performances of our solution are better than the non optimised version. This difference comes from the fact that the maximum frequency of the non optimised design is limited by the critical path of MD5.

V. RELATED WOTK

In [6], the authors propose an architecture for MD5 and SHA-1 based on the idea of the study recovery between the two algorithms. The extensions we propose to their work improve the results they have obtained. Our solution provides an efficiency of 0.43 and 0.34 compared to the 0.31 and 0.25 of [6]. In addition our solution is able to manage the SHA-224/256.

Another paper references a multi-hash core. In [11], authors implemented all the SHA-2 versions (224, 256, 384 and 512). Again our proposition provides a better efficiency even if we consume a little more area space. This difference mainly comes from the logic required for MD5 and SHA-1. Indeed, MD5 and SHA-1 architectures are very close and the amount of logic necessary to add SHA-2 to those 2 algorithms is important (especially T2 in equation 9 which does not have any recovery with other algorithms).

If we have a look at the proposition done in [3], [4] and [5], it appears that the efficiency of our solution is similar or better than those solutions. The fact that we are able to manage 3 algorithms makes up for the little area overhead or the throughput loss.

In [7], an ASIC solution is proposed in 0.18µm and 0.13µm. Thus, these architectures propose the best performances. However the price of such architecture is really high and does not provide any flexibility.

The last comparison to do is with an architecture which uses dynamic reconfiguration. An advantage of such solution is the flexibility and the reduce needs for logic. Throughout this paper, we show how we provide flexibility for the algorithm. Furthermore, in Table 5, it is shown that the area required to gather the 3 algorithms is not important. It means that our solution can be considered to be a very good alternative to the solution with dynamic reconfiguration. Last important point, our solution does not need more memory. Indeed, with the dynamic reconfiguration, extra memory is necessary to store all the bitstreams for the 3 algorithms.

VI. Conclusions

In this paper, we have presented an architecture which gathers three hash functions MD5, SHA-1 and SHA-2. The large similarities between them lead us to study the algorithms in order to implement them in a single design.

The architecture which was implemented on an Altera Stratix II device only requires 1662 slices and operates at 93 MHz. Moreover, we obtained throughput of 721 Mbps for MD5 and SHA-2 and 580 Mbps for SHA-1.

The comparison with previous works shows that our implementation is a good compromise for flexibility, area and throughput. It also confirms that the recovery between algorithms might lead to an optimized architecture. This point is essential due to the specific field of embedded systems with limited resources.

References

[1] R.L. Rivest. The MD5 Message-Digest Algorithm. RFC1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[2] Federal Information Processing Standards. Secure Hash Standard. FIPS PUB 180-2, August 1, 2002.

[3] Dai Zibin, Zhou Ning. FPGA Implementation of SHA-1 Algorithm. Inst. of Electron. Technol., Inf. Eng. Univ., Zhengzhou, China, October 2003.

[4] K. Järvinen, M. Tommiska, and J. Skyttä. Hardware Implementation Analysis of the MD5 Hash Algorithm. Proceedings the 38th Annual Hawai'i International Conference on System Sciences, HICSS'38, Big Island, Hawaii, USA, page 298 (abstract), January 3 – 6, 2005.

[5] J.M. Diez, S. Bojani´c, Lj. Stanimirovi´c, C. Carreras, and O. Nieto-Taladriz. Hash Algorithms for Cryptographic Protocols: FPGA Implementations. Proceedings of the 10th Telecommunications Forum TELFOR'2002, Belgrade, Yugoslavia, November 26 – 28, 2002.

[6] Kimmo Järvinen. A Compact MD5 and SHA-1 Co Implementation Utilizing Algorithm Similarities. Helsinki University of Technology Signal Processing Laboratory Otakaari 5 A, FIN-02150, Finland 2005.

[7] Akashi Satoh, Tadanobu Inoue IBM Research, ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS, Tokyo Research Laboratorya April 2005.

[8] MD5 rfc1321 http://www.faqs.org/rfcs/rfc1321.html

[9] SHA-1 rfc3174
http://www.faqs.org/rfcs/rfc3174.html

[10] SHA-2 fips 180-2
http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[11] Ryan Glabbn Laurent Imbert, Graham Jullien, Arnaud Tisserand and Nicolas Veryat-Charvillon, Multi-mode operator for SHA-2 hash functions, Journal of Systems Architecture, 2007

.