

Cost-Efficient SHA Hardware Accelerators

Ricardo Chaves, *Student Member, IEEE*, Georgi Kuzmanov, *Member, IEEE*, Leonel Sousa, *Senior Member, IEEE*, and Stamatis Vassiliadis, *Fellow, IEEE*

Abstract—This paper presents a new set of techniques for hardware implementations of Secure Hash Algorithm (SHA) hash functions. These techniques consist mostly in operation rescheduling and hardware reutilization, therefore, significantly decreasing the critical path and required area. Throughputs from 1.3 Gbit/s to 1.8 Gbit/s were obtained for the SHA implementations on a Xilinx VIRTEX II Pro. Compared to commercial cores and previously published research, these figures correspond to an improvement in throughput/slice in the range of 29% to 59% for SHA-1 and 54% to 100% for SHA-2. Experimental results on hybrid hardware/software implementations of the SHA cores, have shown speedups up to 150 times for the proposed cores, compared to pure software implementations.

Index Terms—Cryptography, field-programmable gate array (FPGA), hardware implementation, hash functions, Secure Hash Algorithm (SHA).

I. INTRODUCTION

CRYPTOGRAPHIC algorithms can be divided into three several classes: public key algorithms, symmetric key algorithms, and hash functions. While the first two are used to encrypt and decrypt data, the hash functions are one-way functions that do not allow the processed data to be retrieved. This paper focuses on hashing algorithms. Currently, the most commonly used hash functions are the MD5 and the Secure Hash Algorithm (SHA), with 128- to 512-bit output Digest Messages (DMs), respectively. While for MD5, collision attacks are computationally feasible on a standard desktop computer [1], current SHA-1 attacks still require massive computational power [2], (around 2^{69} hash operations), making attacks unfeasible for the time being. For applications that require additional levels of security, the SHA-2 has been introduced. This algorithm outputs a DM with size from 224 to 512 bits.

The SHA-1 was approved by the National Institute of Standards and Technology (NIST) in 1995 as an improvement to the SHA-0. SHA-1 quickly found its way into all major security applications, such as SSH, PGP, and IPsec. In 2002, the SHA-2 [3] was released as an official standard, allowing the compression of inputs up to 2^{128} bits.

Manuscript received January 29, 2007; revised May 21, 2007. This work was supported in part by the Portuguese Foundation for Technology, by the Dutch Technology Foundation STW, applied science division of NWO (project DCS.7533), and by the European Network of Excellence on High-Performance Embedded Architecture and Compilation.

R. Chaves is with the Instituto Superior Técnico/INESC-ID, Lisbon 1049-001, Portugal, and also with the Computer Engineering Department, EEMCS, Delft University of Technology, 2628 CD Delft, The Netherlands.

G. Kuzmanov is with the Computer Engineering Department, EEMCS, Delft University of Technology, 2628 CD Delft, The Netherlands.

L. Sousa is with the Instituto Superior Técnico/INESC-ID, Lisbon 1049-001, Portugal.

S. Vassiliadis, deceased, was with the Computer Engineering, EEMCS, Delft University of Technology, 2628 CD Delft, The Netherlands.

Digital Object Identifier 10.1109/TVLSI.2008.2000450

To enforce the security in general purpose processors (GPPs) and to improve performance, cryptographic algorithms have to be applied at the hardware level, for example in the attestation of external memory transactions. Specialized hardware cores are typically implemented either as application-specific integrated circuit (ASIC) cores [4]–[6] or in reconfigurable devices [7]–[10]. Some work has been done to improve the SHA computational throughput by unrolling the calculation structure, but at the expense of more hardware resources [11], [12].

In this paper, we propose an efficient hardware implementation of SHA. Several techniques have been proposed to improve the hardware implementation of the SHA algorithm, using the following design techniques:

- parallel counters and balanced carry save adders (CSA), in order to improve the partial additions [4], [5], [7];
- unrolling techniques optimize the data dependency and improve the throughput [5], [9], [11], [13];
- balanced delays and improved addition units; in this algorithm, additions are the most critical operations [4], [13];
- embedded memories store the required constant values [8];
- pipelining techniques, allow higher working frequencies [5], [14].

This work extends the ideas originally proposed by the authors in [15] and [16] and presents a significant set of experimental results. Our major contributions to the improvement of the SHA functions hardware implementation can be summarized as follows:

- operation rescheduling for a more efficient pipeline usage;
- hardware reuse in the DM addition;
- a shift-based input/output (I/O) interface;
- memory-based block expansion structures.

A discussion on alternative data block expansion structure has also been introduced.

The fully implemented architectures proposed in this paper, achieve a high throughput for the SHA calculation via operation rescheduling. At the same time, the proposed hardware reuse techniques indicates an area decrease, resulting in a significant increase of the throughput per slice efficiency metric. Implementation results on several FPGA technologies of the proposed SHA, show that a throughput of 1.4 Gbit/s is achievable for both SHA-128 and SHA-256 hash functions. For SHA-512 this value increases to 1.8 Gbit/s. Moreover, a Throughput/Slice improvement up to 100% is achieved, regarding current state of the art.

The proposed SHA cores have also been implemented within the reconfigurable co-processor of a Xilinx VIRTEX II Pro MOLEN prototype [17]. The hybrid implementation results indicate a 150 times speedup against pure software implementations, and a 670% Throughput/Slice improvement regarding related art.

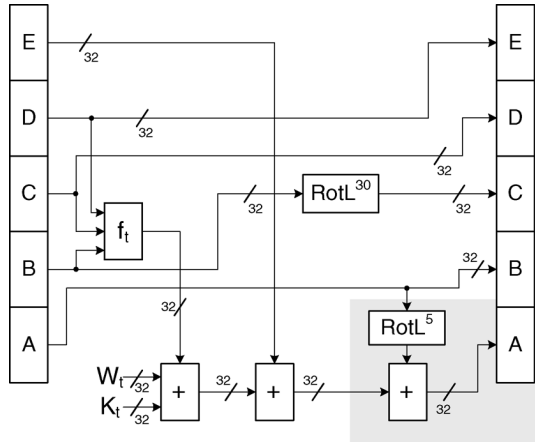


Fig. 1. SHA-1 round calculation.

The remainder of this paper is organized as follows. Section II presents the SHA-1 and SHA-2 algorithms and Section III and 4 describes the proposed design. The characteristics of the SHA function implementations on FPGAs are presented in Section V. Section VI presents the obtained results and compares them to related art. Section VII concludes this paper with some final remarks.

II. SHA-1 AND SHA-2 HASH FUNCTIONS

In 1993, the Secure Hash Standard (SHA) was first published by the NIST. In 1995, this algorithm was revised [18] in order to eliminate some of the initial weakness. The revised algorithm is usually referenced as SHA-1 (or SHA128). In 2001, the hashing algorithm, SHA-2, was proposed. It uses larger DM, making it more resistant to possible attacks and allows it to be used with larger data inputs, up to 2^{128} bits in the case of SHA512. The SHA-2 hashing algorithm is the same for the SHA224, SHA256, SHA384, and SHA512 hashing functions, differing only in the size of the operands, the initialization vectors, and the size of the final DM.

A. SHA128 Hash Function

The SHA-1 produces a single output 160-bit message digest (the output hash value) from an input message. The input message is composed of multiple blocks. The input block, of 512 bits, is split into 80×32 -bit words, denoted as W_t , one 32-bit word for each computational round of the SHA-1 algorithm, as depicted in Fig. 1. Each round comprises additions and logical operations, such as bitwise logical operations (f_t) and bitwise rotations to the left ($RotL^i$). The calculation of f_t depends on the round t being executed, as well as the value of the constant K_t . The SHA-1 80 rounds are divided into four groups of 20 rounds, each with different values for K_t and the applied logical functions (f_t) [15]. The initial values of the A to E variables in the beginning of each data block calculation correspond to the value of the current 160-bit hash value, DM_0 to DM_4 . After the 80 rounds have been computed, the A to E 32-bit values are added to the current DM. The Initialization Vector (IV) or the DM for the first block is a predefined constant value. The output

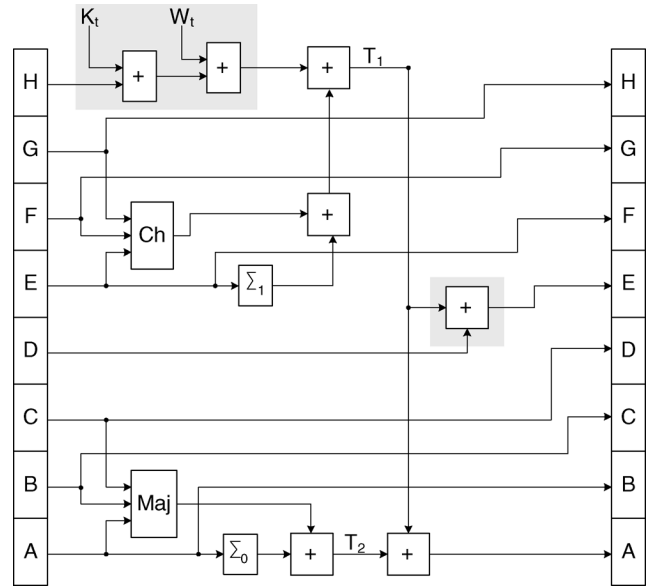


Fig. 2. SHA-2 round calculation.

value is the final DM, after all the data blocks have been computed. In some higher level applications such as the keyed-Hash Message Authentication Code (HMAC) [19], or when a message is fragmented, the initial hash value (IV) may differ from the constant specified in [18].

B. SHA256 Hash Function

In the SHA256 hash function, a final DM of 256 bits is produced. Each 512-bit input block is expanded and fed to the 64 rounds of the SHA256 function in words of 32 bits each (denoted by W_t). Like in the SHA-1, the data scrambling is performed according to the computational structure depicted in Fig. 2 by additions and logical operations, such as bitwise logical operations and bitwise rotations. The computational structure of each round, where the input data is mixed with the current state, is depicted in Fig. 2. Each W_t value is a 32-bit data word and K_t is the 32-bit round dependent constant.

The 32-bit values of the A to H variables are updated in each round and the new values are used in the following round. The IV for these variables is given by the 256-bit constant value specified in [18], being set only for the first data block. The consecutive data blocks use the partial DM computed for the previous data block. Each SHA-256 data block is processed in 64 rounds, after which the values of the variables A to H are added to the previous DM in order to obtain a new value for the DM. Comparing Figs. 1 and 2, is it noticeable a higher computational complexity of the SHA-2 algorithm in comparison to the SHA-1 algorithm.

C. SHA512 Hash Function

The SHA512 hash algorithm computation is identical to that of the SHA256 hash function, differing only in the size of the operands, 64 bits instead of 32 bits as for the SHA256. The DM has twice the width, 512 bits, and different logical functions are used [18]. The values W_t and K_t are 64 bits wide and each

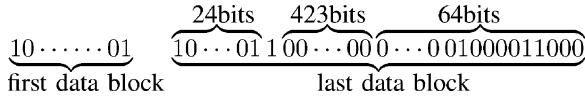


Fig. 3. Message padding for 512 bit data blocks.

data block is composed of 16×64 -bit words, having in total 1024 bits.

D. Data Block Expansion for SHA Function

The SHA-1 algorithm computation steps described in Fig. 1 are performed 80 times (rounds). Each round uses a 32-bit word obtained from the current input data block. Since each input data block only has 16×32 -bits words (512 bits), the remaining 64×32 -bit words are obtained from data expansion. This expansion is performed by computing (1), where $M_t^{(i)}$ denotes the first 16×32 -bit words of the i th data block

$$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ RotL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}), & 16 \leq t \leq 79. \end{cases} \quad (1)$$

For the SHA-2 algorithm, the computation steps shown in Fig. 2 are performed for 64 rounds (80 rounds for the SHA512). In each round, a 32-bit word (or 64-bit for SHA512) from the current data input block is used. Once again, the input data block only has 16×32 -bits words (or 64-bit words for SHA512), resulting in the need to expand the initial data block to obtain the remaining words. This expansion is performed by the computation described in (2), where $M_t^{(i)}$ denotes the first 16 words of the i th data block and the operator $+$ describes the arithmetic addition operation

$$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} \\ \quad + \sigma_0(W_{t-15}) + W_{t-16}, & 16 \leq t \leq 63 \text{ \{or 79\}}. \end{cases} \quad (2)$$

E. Message Padding

In order to assure that the input data block is a multiple of 512 bits, as required by the SHA-1 and SHA256 specification, the original message has to be padded. For the SHA512 algorithm the input data block is a multiple of 1024 bits.

The padding procedure for a 512 bit input data block is as follows: for an original message composed of n bits, the bit “1” is appended at the end of the message (the $n + 1$ bit), followed by k zero bits, where k is the smallest solution to the equation $n + 1 + k \equiv 448 \pmod{512}$. These last 64 bits are filled with the binary representation of n , the original message size. This operation is better illustrated in Fig. 3 for a message with 536 bits (010 0001 1000 in binary representation).

For the SHA512 message padding, 1024-bit data blocks are used and the last 128, not 64 bits, are reserved for the binary value of the original message. This message padding operation can be efficiently implemented in software.

III. PROPOSED DESIGN FOR SHA-1

In order to compute the values of one SHA-1 round, depicted in Fig. 1, the values from the previous round are required. This data dependency imposes sequentiality, preventing parallel computation between rounds. Only parallelism within each round can be efficiently explored. Some approaches [11] attempt to speedup the processing by unrolling each round computations. However, this approach implies an obvious increase in circuit area. Another approach [12], increases the throughput using a pipelined structure. Such an approach, however, makes the core inefficient in practical applications, since a data block can only be processed when the previous one has been completed, due to the data dependencies of the algorithm.

In this paper, we propose a functional rescheduling of the SHA-1 algorithm as described in the work [15], which allows the high throughput of an unrolled structure to be combined with a low hardware complexity.

A. Operations Rescheduling

From Fig. 1, it can be observed that the bulk of the SHA-1 round computation is oriented towards the A value calculation. The remaining values do not require any computation, aside from the rotation of B . The needed values are provided by the previous round values of the variables A to D . Given that the value of A depends on its previous value, no parallelism can be directly exploited, as depicted in (3)

$$A_{t+1} = RotL^5(A_t) + [f(B_t, C_t, D_t) + E_t + K_t + W_t]. \quad (3)$$

In (4), the term of (3) that does not depend on the value of A is precomputed, producing the carry (β_t) and save (S_t) vectors of the partial addition

$$S_t + \beta_t = f(B_t, C_t, D_t) + E_t + K_t + W_t. \quad (4)$$

The calculation of A_t , with the precomputation, is described by the following:

$$\begin{aligned} A_t &= RotL^5(A_{t-1}) + (S_{t-1} + \beta_{t-1}) \\ S_t + \beta_t &= f(B_t, C_t, D_t) + E_t + K_t + W_t. \end{aligned} \quad (5)$$

By splitting the computation of the value A and by rescheduling it to a different computational round, the critical path of the SHA-1 algorithm can be significantly reduced. Since the calculation of the function $f(B, C, D)$ and the partial addition are no longer in the critical path, the critical path of the algorithm is reduced to a three-input full adder and some additional selection logic, as depicted in Fig. 4. With this rescheduling, an additional clock cycle is required, for each data block, since in the first clock cycle the value of A is not calculated (A_{-1} is not used). Note that in the last cycle the values of B_{81}, C_{81}, D_{81} , and E_{81} are not used as well. The additional cycle, however, can be hidden in the calculation of the DM of each input data block, as explained further on.

After the 80 SHA-1 rounds have been computed, the final values of the internal variables (A to E) are added to the current DM. In turn, the DM remains unchanged until the end of each data block calculation [15]. This final addition is performed by one adder for each 32 bits portion of the 160-bit hash value.

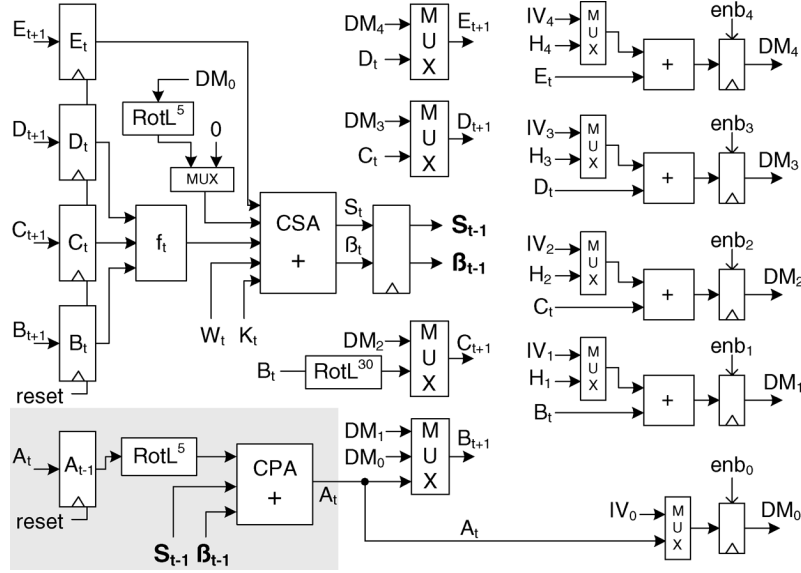


Fig. 4. SHA-1 rescheduling and internal structure.

However, the addition of the value DM_0 is directly performed by a CSA adder in the round calculation. With this option, an extra full adder is saved and the DM_0 value calculation, that depends on the value A , is performed in one less clock cycle. Thus, the calculation of all the DM_j is concluded in the same cycle.

B. Hash Value Initialization

For the first data block the internal hash value (DM_0) is initialized, by adding zero to the Initialization Vector (IV). This initial value is afterwards loaded to the internal registers (B to E), through a multiplexer. In this case, the value of DM_0 is not set to the register A . Instead, A is set to zero and DM_0 is directly introduced into the calculation of A , as described in (6)

$$\begin{aligned} S_0 + \beta_0 &= f(B_{DM_1}, C_{DM_2}, D_{DM_3}) \\ &\quad + E_{DM_4} + K_0 + W_0 + RotL^5(DM_0) \\ A_1 &= RotL^5(A_0) + (S_0 + \beta_0) \\ &= RotL^5(0) + (S_0 + \beta_0). \end{aligned} \quad (6)$$

The IV can be the constant value defined in [18] or an application dependent value, e.g., from the HMAC or from the hashing of fragmented messages. In applications, where the IV is always a constant, the selection between the IV and the current hash value can be removed and the constant value set in the DM registers. In order to minimize the power consumption, the internal registers are disabled when the core is not being used.

C. Improved Hash Value Addition

After all the rounds have been computed, for a given data block, the internal variables have to be added to the current DM. This addition can be performed with one adder per each 32 bit of the DM, as depicted in Fig. 4. In such structure, the addition of B through E with the current DM requires four additional adders. Taking into account that

$$E_t = D_{t-1} = C_{t-2} = RotL^{30}(B_{t-3}) \quad (7)$$

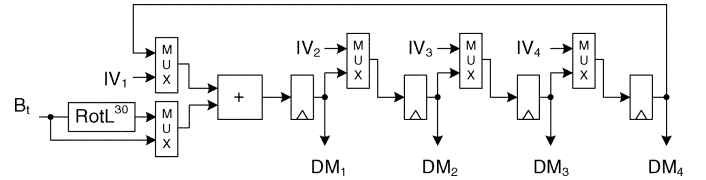


Fig. 5. Alternative SHA-1 DM addition.

the computation of the DM from the data block i can be calculated from the internal variable B , as

$$\begin{aligned} DM_{4i} &= RotL^{30}(B_{t-3}) + DM_{4i-1}; \\ DM_{3i} &= RotL^{30}(B_{t-2}) + DM_{3i-1}; \\ DM_{2i} &= RotL^{30}(B_{t-1}) + DM_{2i-1}; \\ DM_{1i} &= B_t + DM_{1i-1}. \end{aligned} \quad (8)$$

Thus, the calculation can be performed by just a single addition unit and a multiplexer unit, used to select between the value B and its bitwise rotation, $RotL^{30}$. The $rot()$ function in (9) represents the optional rotation of the input value

$$DM[j]_i = rot(B_{t-j+1}) + DM[j]_{i-1} \quad ; \quad 1 \leq j \leq 4. \quad (9)$$

The alternative hardware structure for the addition of the values B to E with the current DM is depicted in Fig. 5.

D. SHA-1 Data Block Expansion

For efficiency reasons, we expand the 512 bits of each data block in hardware. The input data block expansion described in (1), can be implemented with registers and XOR operations. Finally, the output value W_t is selected between the original data block, for the first 16 rounds, and the computed values, for the remaining rounds. Fig. 6 depicts the implemented structure. Part of the delay registers have been placed after the calculation, in order to eliminate this computation from the critical path, since the value W_t is connected directly to the the SHA-1 core. The

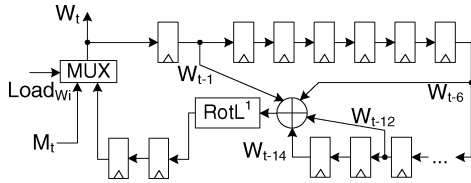


Fig. 6. Register-based SHA-1 block expansion.

one bit rotate-left operation can be implemented directly in the routing process, not requiring additional hardware.

IV. PROPOSED DESIGN FOR SHA-2

Like for the SHA-1, the functional rescheduling can also be applied to SHA-2 algorithm. However, as depicted in Fig. 2, the SHA-2 computational path is more complex and with an even higher data dependency level. In each round of the algorithm, the values A through H have to be calculated, but only the values A and E require computation. In the proposed SHA-2 computational structure [16], we identified the part of the computation of a given round t that can be computed ahead in the previous round $t - 1$.

A. Operation Rescheduling

While the variables B , C , D , F , G , and H are obtained directly from the values of the round, not requiring any computation, the values A and E for round t cannot be computed until the values for the same variables have been computed in the previous round, as shown in (10)

$$\begin{aligned} E_{t+1} &= D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + H_t + K_t + W_t \\ A_{t+1} &= \Sigma_0(A_t) + Maj(B_t, C_t, D_t) + \Sigma_1(E_t) \\ &\quad + Ch(E_t, F_t, G_t) + H_t + K_t + W_t. \end{aligned} \quad (10)$$

Taking into account that the value H_{t+1} is given directly by G_t which in its turn is given by F_{t-1} , the precalculation of H can thus be given by $H_{t+1} = F_{t-1}$. Since the values K_t and W_t can be precalculated and are directly used in each round, (10) can be rewritten as

$$\begin{aligned} \delta_t &= H_t + K_t + W_t = G_{t-1} + K_t + W_t \\ E_{t+1} &= D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \delta_t \\ A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) + \Sigma_1(E_t) \\ &\quad + Ch(E_t, F_t, G_t) + \delta_t \end{aligned} \quad (11)$$

where the value δ_t is calculated in the previous round. The value δ_{t+1} can be the result of a full addition or the two vectors from a carry save addition. With this computational separation, the calculation of the SHA-2 algorithm can be divided into two parts, allowing the calculation of δ to be rescheduled to the previous clock cycle, as depicted by the grey area in Fig. 7. Thus, the critical path of the resulting hardware implementation can be reduced. Since the computation is now divided by a pipeline stage, the computation of the SHA-2 requires an additional clock cycle to perform all the rounds.

B. Hash Value Addition and Initialization

Similar to SHA-1, the internal variables of SHA-2 also have to be added to the DM. If this addition is implemented in a straightforward manner, eight adders would be required, one for each internal variable, of 32 or 64 bits for SHA256 or SHA512, respectively. The experimental results for the SHA-1 implementation, presented in Section VI, suggest the DM addition with the shift to be more area efficient. Thus only this approach is studied in the proposed SHA-2 structure for the addition of the internal values with the DM value. Since most of the SHA-2 internal values do not require any computation, they can be directly obtained from the previous values of A and E

$$\begin{aligned} H_t &= G_{t-1} = F_{t-2} = E_{t-3} \\ D_t &= C_{t-1} = B_{t-2} = A_{t-3}. \end{aligned} \quad (12)$$

The computation of the DM for the data block i can thus be calculated from the internal variables A and E , as

$$\begin{aligned} DM7_i &= E_{t-3} + DM7_{i-1} \\ DM6_i &= E_{t-2} + DM6_{i-1} \\ DM5_i &= E_{t-1} + DM5_{i-1} \\ DM3_i &= A_{t-3} + DM3_{i-1} \\ DM2_i &= A_{t-2} + DM2_{i-1} \\ DM1_i &= A_{t-1} + DM1_{i-1} \end{aligned} \quad (13)$$

with only two addition units

$$\begin{aligned} DM[j+4]_i &= E_{t-j} + DM[j+4]_{i-1}; \quad 1 \leq j \leq 3 \\ DM[j]_i &= A_{t-j} + DM[j]_{i-1}; \quad 1 \leq j \leq 3. \end{aligned} \quad (14)$$

The selection of the corresponding part of the $DM[j]$ could be performed by a multiplexer. However, taking into account that the values of $DM[j]$ are used sequentially, a shifting buffer can be used, as depicted in the right most part of Fig. 7. Since the values A_t and E_t require computation and the final value is calculated in the last clock cycle, the calculation of the values $DM0_i$ and $DM4_i$ is performed in a different manner. Instead of using a full adder, after the calculation of the final value of A and E , the DM is added during the calculation of their final values. Since the value of DM_{i-1} is known, the value can be added during the first stage of the pipeline, using a CSA.

After each data block has been computed, the internal values A to H have to be reinitialized with the newly calculated DM. This is performed by a multiplexer that selects either the new value of the variable or the DM, as depicted in the left most side of Fig. 7. The values A and E are the exception, since the final value computed for the two variables is already the DM.

In the first round, all variables, except A and E , are initialized by loading the values of the DM registers, operation depicted in the leftmost part of Fig. 7. For the A and E variables, the value are fed through the round logic. In this case, all the variables are set to zero (Reset) except the DM_0 and DM_4 inputs. Thus, the resulting value for the registers A and E will be the initialization values of the DM registers.

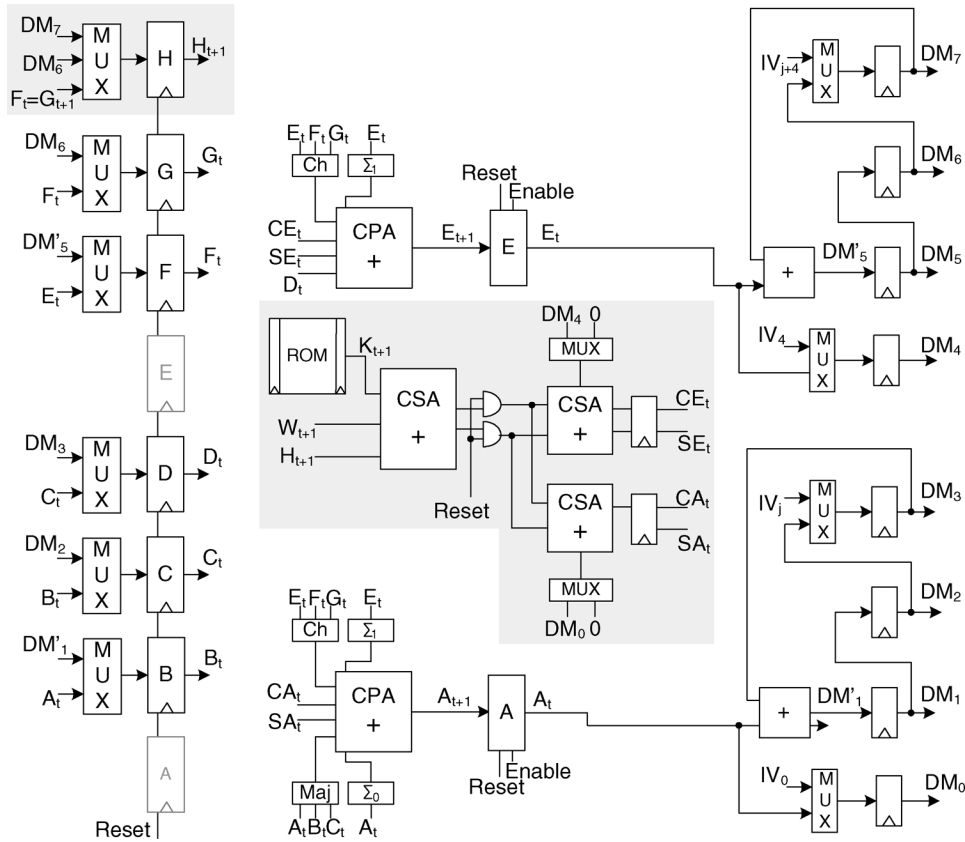


Fig. 7. SHA-2 round architecture.

In the SHA-2 algorithm standard, the initial value of the DM (loaded in A through H) is a constant value, that can be loaded by using set/reset signals in the registers. If the SHA-2 algorithm is to be used in a wider set of applications and in the computation of fragmented messages, the initial DM is no longer a constant value. In these cases, the initial value is given by an Initialization Vector (IV) that has to be loaded. In order to optimize the architecture, the calculation structure for the DM can be used to load the IV , not being directly loaded into all the registers. The value of the A and E registers is set to zero during this loading, thus the existing structure acts as a circular buffer, where the value is only loaded into one of the registers, and shifted to the others. This circular buffer can also be used for a more efficient reading of the final DM, providing an interface with smaller output ports.

C. SHA-2 Data Block Expansion

As mentioned in Section II, the W_t input value has to be expanded according to (2). This operation is performed by the data block expansion unit. The computation structure is similar to the one presented for SHA128. The arithmetic addition represented by the operator \boxplus , replaces the bitwise XOR operation (\oplus). The proposed structure is depicted in Fig. 8.

V. SHA IMPLEMENTATION

In order to evaluate the proposed SHA designs, they have been implemented as processor cores on a Xilinx VIRTEX II Pro (XC2VP30-7) FPGA. All the values presented in this paper for the proposed cores were obtained after Place and Route.

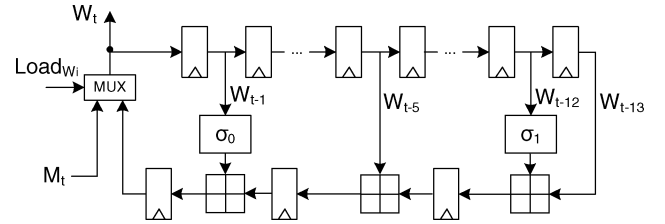


Fig. 8. SHA-2 data block expansion unit.

When implementing the ROM used to store the K_t values of SHA256 or SHA512, the FPGA embedded RAMs (BRAMs) have been efficiently employed. For the SHA256 structure, a single BRAM can be used, since the 64 32-bits fit in a single 32-bit port embedded memory block. Since BRAMs have dual output ports of 32 bits each, the 80×64 -bit SHA-512 constants can be mapped to two 32-bit memory ports; one port addresses the lower 32 bits of the constant and the other, the higher part of the same constant. Thus, only one BRAM is used to store the 64-bit K_t constants.

A. Discussion on Alternative Data Block Expansion Structures

Alternatively to the register-based structure presented in Fig. 6, other structures for the SHA-1 data block expansion can be implemented. One is based on memory blocks addressed in a circular fashion. In the presented implementation, the VIRTEX II embedded RAMs (BRAMs) are used. The other structure is based on first-inputs-first-outputs (FIFOs).

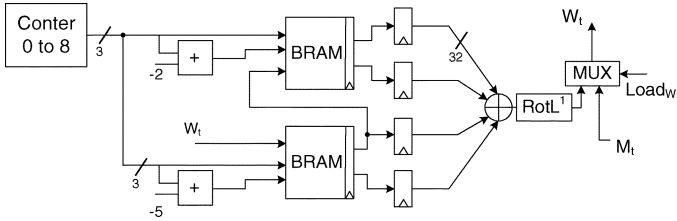


Fig. 9. BRAM-based data block expansion unit.

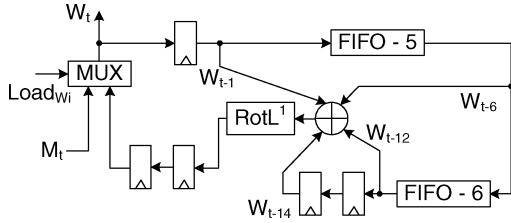


Fig. 10. FIFO-based data block expansion unit.

A 16-word memory is used to store the values with 14 (w_{t-14}) and 16 (w_{t-16}) clock cycles delay. In order to use the dual port BRAMs, the address of the new value has to be the same as the last one, thus the first and the last position of the circular buffer coincide. For this scheme to work properly, the memory must allow for write after read (WAR). This however, is only available on the VIRTEX II FPGA family. In technologies where WAR is not available, the first and last position of this circular memory can not coincide, thus an additional position in the memory is required as well as an additional port. The W_{t-14} can be addressed by using the $W_{t-16}(=W_t)$ address value subtracted by 2. Identically, the W_{t-3} address can be obtained by subtracting 5 from the W_{t-8} address. The implementation of the 16 bit position circular memory can be done by chaining two eight positions circular memories, thus requiring less memory for the entire unit, as depicted in Fig. 9.

The data block expansion can also be implemented with FIFOs. In technologies where FIFOs can be efficiently used, the registers used to create the temporal delay of W_t can be replaced by FIFOs. The FIFOs start outputting the values after n clock cycles, where n is the desired delay. The FIFOs have been automatically generated by the coregen tool from Xilinx. The resulting computational structure is depicted in Fig. 10. Circular memories can also be used. For this structure (FIFO-MEM-based), modulo 5 and modulo 6 counters have to be used, as well as memories that do not require the WAR mode.

In order to completely evaluate the proposed structures, they have been implemented on a Xilinx VIRTEX II FPGA. The obtained results are presented in Table I. From Table I, it can be concluded that when memories with WAR mode are available, the memory-based implementation is more efficient. It requires only 38 slices and two 32×8 bit memories, resulting in a slice occupation of only 30% of the register-based approach, at the expense of two BRAMs. When, only, memories without WAR mode are available, a reduction of 35% in terms of slice usage can still be achieved, at the expense of two embedded RAMs. These data block expansion structures for the SHA-1

TABLE I
SHA-1 DATA BLOCK EXPANSION UNIT COMPARISON

Design	Slices	BRAMs
Register based	144	0
Memory based	38	2
FIFO based	100	2
FIFO-MEM based	90	2

TABLE II
SHA-1 DM ADDITION COMPARISON

Design	traditional addition	shift based addition
Slices	596	565
Freq. (MHz)	227	227
TrPut.(Mbps)	1420	1420
TP/Slice	2.4	2.5

algorithm can be directly mapped to the data block expansion of the SHA-2 algorithm. For the remainder of this paper, only the register-based unit is considered, in order to obtain less technology dependent experimental results.

VI. PERFORMANCE ANALYSIS AND RELATED WORK

In order to compare the architectural gains of the proposed SHA structures with the current related art, the resulting cores have been implemented in different Xilinx devices.

A. SHA-1 Core

In order to compare the efficiency of the DM addition through shift registers proposed in Section III-C, the SHA-1 algorithm with variable IV has been implemented with both presented structures. Table II presents the obtained results for a realization on the VIRTEX II Pro FPGA. The obtained figures suggest an area reduction of 5% with no degradation on the achievable frequency, resulting in a Throughput/Slice increase from 2.4 to 2.5 Mbit/s. In technologies where full addition units are more expensive, like ASICs, even higher improvements can be expected.

The SHA-1 core has also been implemented on a VIRTEX-E (XCV400e-8) device (Column Our-Exp. in Table III), in order to compare with [11]. The presented results in Table III for the VIRTEX-E device are for the SHA-1 core with a constant initialization vector and without the data block expansion module. When compared with the folded SHA-1 core proposed in [11], a clear advantage can be observed in both area and throughput. Experimentations suggest 20% less reconfigurable hardware and 27% higher throughput, resulting in a 57% improvement on the Throughput/Slice (TP/Slice) metric. When compared with the unfolded architecture, the proposed core has a 28% lower throughput, however, the unrolled core proposed in [11] requires 280% more hardware, resulting in a TP/Slice, 2.75 times smaller than the core proposed in this paper.

Table III also presents the SHA-1 core characteristics for the VIRTEX II Pro FPGA implementation. Both the core with a constant initialization vector (Our-Cst.) and the one with a variable IV initialization (Our+IV) are presented. These results also include the data block expansion block.

When compared with the leading commercial SHA-1 core from Helion [21], the proposed architecture requires 6% less

TABLE III
SHA-1 CORE PERFORMANCE COMPARISONS

Design	Lien [11]	Lien [11]	Our-Exp.	CAST [20]	Helion [21]	Our-Cst.	Our +IV
Device	Virtex-E	Virtex-E	Virtex-E	XCV2P2-7	XCV2P-7	XCV2P30-7	XCV2P30-7
Expansion	no	no	no	yes	yes	yes	yes
IV	cst.	cst.	cst.	cst.	cst.	cst.	yes
Slices	484	1484	388	568	564	533	565
Freq. (MHz)	103	73	135	127	194	230	227
TrPut.(Mbps)	659	1160	840	802	1211	1435	1420
TP/Slice	1.4	0.8	2.2	1.4	2.1	2.7	2.5

TABLE IV
SHA256 CORE PERFORMANCE COMPARISON

Architecture	Sklav [22]	Our	McEv. [13]	Our	Helion [23]	Our
Device	XCV	XCV	XC2V	XC2V	XC2PV-7	XC2PV-7
IV	cst	yes	cst	yes	cst	yes
Slices	1060	764	1373	797	815	755
BRAMS	≥ 1	1	≥ 1	1	1	1
Freq.	83	82	133	150	126	174
Cycles	n.a.	65	68	65	n.a.	65
Throughput	326	646	1009	1184	977	1370
TP/Slice	0.31	0.84	0.74	1.49	1.2	1.83

TABLE V
SHA512 CORE PERFORMANCE COMPARISON

Architecture	Sklav [22]	Lien [11]	Lien [11]	Our	McEv. [13]	Our	Our
Device	XCV	XCV	XCV	XCV	XC2V	XC2V	XC2VP
Expansion	yes	no	no	yes	yes	yes	yes
IV	cst	cst	cst	yes	cst	yes	yes
Slices	2237	2384 ¹	3521	1680	2726	1666	1667
BRAMS	n.a.	n.a.	n.a.	2	≥ 1	1	1
Freq.	75	56	67	70	109	121	141
Cycles	n.a.	n.a.	n.a.	81	84	81	81
Throughput	480	717	929	889	1329	1534	1780
TP/Slice	0.21	0.3	0.26	0.53	0.49	0.92	1.01

slices while achieving a throughput 18% higher. These two results suggest a gain on the TP/Slice metric of about 29%.

For the SHA-1 core capable of receiving an *IV* other than the constant specified in [18], a slight increase in the required hardware occurs. This is due to the fact that the *IV* can no longer be set by the set/reset signals of the registers. This however has a minimal effect in the cores performance, since this loading mechanism is not located in the critical path. The decrease of the Throughput/Slice metric, from 2.7 to 2.5, caused by the additional hardware for the *IV* loading is counterbalanced by the capability of this SHA-1 core (Our+*IV*) to process fragmented messages.

B. SHA 256 Core

The proposed SHA256 hash function core has been also compared with the most recent and most efficient related art. The comparison figures are presented in Table IV. When compared with the most recent academic work [13], [22] the results show higher throughputs, from 17% up to 98%, while achieving a reduction in area above 25% up to 42%. These figures suggest a significant improvement to the TP/Slice metric in the

range of 100% to 170%. When compared with the commercial SHA256 core from Helion [23], the proposed core suggests an identical area value (less 7%) while achieving a 40% gain to the throughput, resulting in an improvement of 53% to the TP/Slice metric. The structure proposed by McEvoy [13] also has message padding hardware, however, no figures are given for the individual cost of this extra hardware. This message padding is performed once at the end of the message, and has no significant cost when implemented in software. Thus, the majority of the proposed cores do not include the hardware for this operation.

C. SHA 512 Core

Table V presents the implementation results for our SHA512 core and the most significant related art, to our best knowledge. When compared with [22], our core requires 25% less reconfigurable logic while a throughput increase of 85% is achieved, resulting in a TP/Slice metric improvement of 165%. From all known SHA512 cores, the unrolled core proposed by Lien in [11] is the only one capable of achieving a higher throughput. However, this throughput is only slightly higher (4%), but requires twice as much area as our proposal and indicates a 77%

higher TP/Slice metric. It should also be noticed that, the results presented by Lien in [11] do not include the data expansion module, which would increase the required area even further.

D. Integration on the MOLEN Processor

In order to create a practical platform where the SHA cores can be used and tested, a wrapping interface has been added to integrate these units in the MOLEN polymorphic processor. The MOLEN operation [17], [24] is based on the coprocessor architectural paradigm, allowing the usage of reconfigurable custom designed hardware units. The MOLEN computational paradigm enables the SHA cores to be embedded in a reconfigurable coprocessor, tightly coupled with the GPP. The considered polymorphic architecture prototype uses the FPGA with an embedded PowerPC, running at 300 MHz as the core GPP, and a main data memory running at 100 MHz. The implementation is identical to the one described in [25].

For this coprocessor implementations of the SHA hash functions, the SHA128, SHA256, and SHA512 cores, with IV loading, have been used. Implementations results of the SHA128 CCU indicate a device occupation of 813 slices, using a total of 6% of the available resources on a XC2VP30 FPGA. In this functional test the CCU is running with same clock frequency as the main data memory, operating at 100 MHz, thus achieving a maximum throughput of 623 Mbit/s. When compared with the pure software implementations, capable of achieving a maximum throughput of 4 Mbit/s and 5 Mbit/s for SHA128 and SHA256, respectively, the usage of this hybrid HW/SW approach allows for a speedup up to 150 times.

The CCUs for the SHA256 and SHA512 cores require 994 and 1806 Slices using in total 7% and 13% of the available resources, respectively. At 100 MHz, the SHA-2 CCUs are capable of achieving a maximum throughput of 785 Mbit/s for SHA-256 and 1.2 Gbit/s for the SHA-512 hash function.

VII. CONCLUSION

We have proposed hardware rescheduling and reutilization techniques to improve SHA algorithm realizations, both in speed and in area. With operation rescheduling, the critical path can be reduced in a similar manner to structures with loop unrolling, without increasing the required hardware, also leading to the usage of a well balanced pipeline structure. An efficient technique for the addition of the DM is also proposed. This technique allows for a substantial reduction on the required reconfigurable resources, while concealing the extra clock cycle delay introduced by the pipeline.

Implementation results clearly indicate significant performance and hardware gains for the proposed cores when compared to the existing commercial cores and related academia art. Experimental results for hybrid, hardware/software, implementations of the SHA algorithms suggest a speed up of 150 times for both hash computations regarding pure software implementations.

A. SHA Evaluation Prototypes

SHA evaluation prototypes are available for download at: <http://ce.et.tudelft.nl/MOLEN/applications/SHA/>.

ACKNOWLEDGMENT

The authors would like to dedicate this paper to S. Vassiliadis, a brilliant researcher, a sunny character, and a dear friend. His scientific talent will be greatly missed, his legacy—never forgotten.

REFERENCES

- [1] V. Klima, "Finding MD5 collisions—A toy for a notebook." Cryptology ePrint Archive, 2005/075, 2005.
- [2] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *CRYPTO*, V. Shoup, Ed. New York: Springer, 2005, vol. 3621, Lecture Notes in Computer Science, pp. 17–36.
- [3] National Institute of Standards and Technology (NIST), MD, "FIPS 180–2, secure hash standard (SHS)," 2002.
- [4] L. Dadda, M. Macchetti, and J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," in *Proc. DATE*, 2004, pp. 70–75.
- [5] M. Macchetti and L. Dadda, "Quasi-pipelined hash circuits," in *Proc. IEEE Symp. Comput. Arithmetic*, 2005, pp. 222–229.
- [6] L. Dadda, M. Macchetti, and J. Owen, D. Garrett, J. Lach, and C. A. Zukowski, Eds., "An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512)," in *Proc. ACM Great Lakes Symp. VLSI*, 2004, pp. 421–425.
- [7] 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," in *ISC*, A. H. Chan and V. D. Gligor, Eds. New York: Springer, 2002, vol. 2433, Lecture Notes in Computer Science, pp. 75–89.
- [8] M. McLoone and J. V. McCanny, "Efficient single-chip implementation of SHA-384 & SHA-512," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2002, pp. 311–314.
- [9] N. Sklavos and O. Koufopavlou, "Implementation of the SHA-2 hash family standard using FPGAs," *J. Supercomput.*, vol. 31, pp. 227–248, 2005.
- [10] K. K. Ting, S. C. L. Yuen, K.-H. Lee, and P. H. W. Leong, "An FPGA based SHA-256 processor," in *FPL*, M. Glesner, P. Zopf, and M. Renovell, Eds. New York: Springer, 2002, vol. 2438, Lecture Notes in Computer Science, pp. 577–585.
- [11] R. Lien, T. Grembowski, and K. Gaj, "A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512," in *Proc. CT-RSA*, 2004, pp. 324–338.
- [12] N. Sklavos, E. Alexopoulos, and O. G. Koufopavlou, "Networking data integrity: High speed architectures and hardware implementations," *Int. Arab J. Inf. Technol.*, vol. 1, pp. 54–59, 2003.
- [13] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," in *Proc. IEEE Comput. Soc. Annu. Symp. Emerging VLSI Technol. Arch. (ISVLSI)*, 2006, pp. 317–322.
- [14] H. E. Michail, A. P. Kakarountas, G. N. Selimis, and C. E. Goutis, "Optimizing SHA-1 hash function for high throughput with a partial unrolling study," in *PATMOS*, V. Paliouras, J. Vounckx, and D. Verkest, Eds. New York: Springer, 2005, vol. 3728, Lecture Notes in Computer Science, pp. 591–600.
- [15] R. Chaves, G. Kuzmanov, L. A. Sousa, and S. Vassiliadis, "Rescheduling for optimized SHA-1 calculation," in *Proc. SAMOS Workshop Comput. Syst. Arch. Model. Simulation*, Jul. 2006, pp. 425–434.
- [16] R. Chaves, G. Kuzmanov, L. A. Sousa, and S. Vassiliadis, "Improving SHA-2 hardware implementations," in *Proc. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, Oct. 2006, pp. 298–310.
- [17] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN polymorphic processor," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1363–1375, Nov. 2004.
- [18] National Institute of Standards and Technology (NIST), MD, "Announcing the standard for secure hash standard," FIPS 180, 1993.
- [19] National Institute of Standards and Technology (NIST), "The keyed-hash message authentication code (HMAC)," FIPS 198, 2002.
- [20] CAST, Woodcliff Lake, NJ, "SHA-1 secure hash algorithm cryptoprocessor core," 2005 [Online]. Available: <http://http://www.cast-inc.com/>
- [21] HELION, Cambridge, U.K., "Fast SHA-1 hash core for Xilinx FPGA," 2005 [Online]. Available: <http://www.heliontech.com/>
- [22] N. Sklavos and O. Koufopavlou, "On the hardware implementation of the SHA-2 (256,384,512) hash functions," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2003, pp. 25–28.

- [23] HELION, Cambridge, U.K., "Fast SHA-2 (256) hash core for Xilinx FPGA," 2005 [Online]. Available: <http://www.heliontech.com/>
- [24] S. Vassiliadis, S. Wong, and S. D. Cotofana, "The MOLEN $\rho\mu$ -coded processor," in *Proc. 11th Int. Conf. Field-Program. Logic Appl. (FPL)*, Aug. 2001, vol. 2147, pp. 275–285.
- [25] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. A. Sousa, "Reconfigurable memory based AES co-processor," in *Proc. 13th Reconfigurable Arch. Workshop (RAW)*, Apr. 2006, pp. 192–192.



Ricardo Chaves (S'05) received the graduate and Masters degree in electronics and computer science from the Instituto Superior Técnico (IST), Portugal, Spain, in 2001 and 2003, respectively. Currently, he is pursuing the Ph.D. degree in electrical engineering from IST and TUDelft.

He is a Researcher with R&D, Instituto de Engenharia de Sistemas e Computadores (INESC-ID). His research interests include reconfigurable computing, cryptography, computer architecture, computer arithmetic, and design of VLSI circuits.



Georgi Kuzmanov (S'95–M'05) was born in Sofia, Bulgaria, in 1974. He received the M.Sc. degree in computer systems from the Technical University of Sofia, Bulgaria, in 1998, and the Ph.D. degree in computer engineering from Delft University of Technology (TU Delft), Delft, The Netherlands, in 2004.

He is currently with the Computer Engineering Laboratory, TU Delft. Between 1998 and 2000, he was with "Info Microsystems" Ltd., Sofia, where he was involved in several reconfigurable computing and

ASIC projects as a Research and Development Engineer. His research interests include reconfigurable computing, media processing, computer arithmetic, computer architecture and organization, vector processors, and embedded systems.

Dr. Kuzmanov was a recipient of the "Veni" research grant from the Dutch Technology Foundation STW, in 2006.



Leonel Sousa (M'01–SM'03) received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 1996.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, IST, and a Senior Researcher with R&D, Instituto de Engenharia de Sistemas e Computadores. His research interests include VLSI architectures, computer architectures, computer arithmetic, parallel and distributed computing, multimedia systems. He has contributed to more than 100 papers to journals and international conferences.

Dr. Sousa is member the IEEE Computer Society and a member of ACM.



Stamatis Vassiliadis (M'86–SM'92–F'97) was born in Manolates, Samos, Greece, in 1951. Regrettably, Prof. Vassiliadis deceased in April 2007.

He was a Chair Professor in the Electrical Engineering, Mathematics, and Computer Science (EEMCS) Department, Delft University of Technology (TU Delft), Delft, The Netherlands. He previously served in the Electrical Engineering faculties of Cornell University, Ithaca, NY, and the State University of New York (S.U.N.Y.), Binghamton, NY. For a decade, he worked with IBM, where he

was involved in a number of advanced research and development projects.

Dr. Vassiliadis was a recipient of numerous awards for his work, including 24 publication awards, 15 invention awards, and an outstanding innovation award for engineering/scientific hardware design. His 72 US patents rank him as the top all-time IBM inventor. He received an honorable mention Best Paper Award at the ACM/IEEE MICRO25 in 1992, Best Paper Awards in the IEEE CAS (1998), IEEE ICCD (2001), PDCS (2002), and the Best Poster Award in the IEEE NANO (2005). He was an ACM fellow and a member of the Royal Dutch Academy of Arts and Science (KNAW).