IMPLEMENTATION OF SHA- HARDWARE ACCELERATORS

Sanjeeva M, SURESH REDDY PURINI,SRIKANTH MYDAPALLY, D.SHEKAR GOUD, A.PRADEEP KUMAR
Assistant Professor,Deptof ECE,
Ellenki college of Engineering & Technology,
sanjeevmadduluri@gmail.com

***Abstract*—This paper presents a new set of techniques for hardwareimplementations of Secure Hash Algorithm (SHA) hash functions.These techniques consist mostly in operation reschedulingand hardware reutilization, therefore, significantly decreasing thecritical path and required area. Throughputs from 1.3 Gbit/s to1.8 Gbit/s were obtained for the SHA implementations on a XilinxVIRTEX II Pro. Compared to commercial cores and previouslypublished research, these figures correspond to an improvement inthroughput/slice in the range of 29% to 59% for SHA-1 and 54%to 100% for SHA-2. Experimental results on hybrid hardware/softwareimplementations of the SHA cores, have shown speedups upto 150 times for the proposed cores, compared to pure software implementations.**

***Index Terms*—Crytography, field-programmable gate array(FPGA), hardware implementation, hash functions, Secure HashAlgorithm (SHA).**

I. INTRODUCTION

Cryptographic algorithms can be divided into threeseveral classes: public key algorithms, symmetric key algorithms,and hash functions. While the first two are used toencrypt and decrypt data, the hash functions are one-way functionsthat do not allow the processed data to be retrieved. Thispaper focuses on hashing algorithms.Currently, the most commonlyused hash functions are the MD5 and the Secure HashAlgorithm (SHA), with 128- to 512-bit output Digest Messages(DMs), respectively. While for MD5, collision attacks are computationallyfeasible on a standard desktop computer [1], currentSHA-1 attacks still require massive computational power[2], (around hash operations), making attacks unfeasible forthe time being. For applications that require additional levels ofsecurity, the SHA-2 has been introduced. This algorithm outputsa DM with size from 224 to 512 bits.The SHA-1 was approved by the National Institute of Standardsand Technology (NIST) in 1995 as an improvement to theSHA-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 compressionof inputs up to 2^{128}bits.

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

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

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-5, ISSUE-1, 2018
DOI: 10.21276/ijcesr.2018.5.1.5

• parallel counters and balanced carry save adders (CSA), inorder to improve the partial additions [4], [5], [7];
• unrolling techniques optimize the data dependency and improvethe 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 authorsin [15] and [16] and presents a significant set of experimentalresults. Our major contributions to the improvement ofthe SHA functions hardware implementation can be summarizedas 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 hasalso been introduced.

The fully implemented architectures proposed in this paper,achieve a high throughput for the SHA calculation via operationrescheduling. At the same time, the proposed hardwarereuse techniques indicates an area decrease, resulting in asignificant increase of the throughput per slice efficiencymetric. Implementation results on several FPGA technologiesof the proposed SHA, show that a throughput of 1.4 Gbit/s isachievable for both SHA-128 and SHA-256 hash functions.For SHA-512 this value increases to 1.8 Gbit/s. Moreover,a Througput/Slice improvement up to 100% is achieved, regardingcurrent state of the art.
The proposed SHA cores have also been implemented withinthe reconfigurable co-processor of a Xilinx VIRTEX II ProMOLEN prototype [17]. The hybrid implementation resultsindicate a 150 times speedup against pure software implementations,and a 670% Throughput/Slice improvement regardingrelated art.

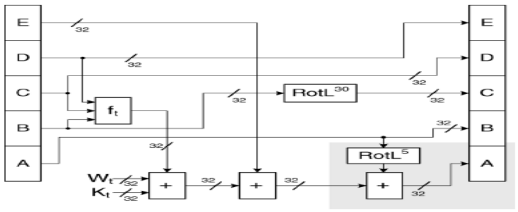

Fig. 1. SHA-1 round calculation

II. SHA-1 AND SHA-2 HASH FUNCTIONS

In 1993, the Secure Hash Standard (SHA) was first publishedby the NIST. In 1995, this algorithm was revised [18] in order toeliminate some of the initial weakness. The revised algorithm isusually referenced as SHA-1 (or SHA128). In 2001, the hashingalgorithm, SHA-2, was proposed. It uses larger DM, making itmore resistent to possible attacks and allows it to be used withlarger data inputs, up to 2^{128}bits in the case of SHA512. TheSHA-2 hashing algorithm is the same for the SHA224, SHA256,SHA384, and SHA512 hashing functions, differing only in thesize of the operands, the initialization vectors, and the size ofthe 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 messageis composed of multiple blocks. The input block, of 512bits, is split into 80× 32-bit words, denoted as W_t, one 32-bitword for each computational round of the SHA-1 algorithm, asdepicted in Fig. 1. Each round comprises additions and logicaloperations, such as bitwise logical operations (f_t) and bitwiserotations to the left ($RotL^i$). The calculation of $f_t$depends onthe round being executed, as well as the value of the constant K_t. The SHA-1 80 rounds are divided into four groups of 20rounds, each with different values for K_t and the applied logicalfunctions (f_t) [15]. The initial values of the to variables inthe beginning of each data block calculation correspond to thevalue of the current 160-bit hash value, to . After the80 rounds have been computed, the to 32-bit values areadded to the current DM. The Initialization Vector (*IV*) or theDM for the first block is a predefined constant value. The outputvalue is the final DM, after all the data blocks have been computed.In some higher level applications such as the keyed-HashMessage Authentication Code (HMAC) [19], or when a messageis

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-5, ISSUE-1, 2018
DOI: 10.21276/ijcesr.2018.5.1.5

fragmented, the initial hash value (*IV*) may differ fromthe 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 64rounds of the SHA256 function in words of 32 bits each (denotedby W_t). Like in the SHA-1, the datascrambling is performedaccording to the computational structure depicted inFig. 2 by additions and logical operations, such as bitwise logicaloperations and bitwise rotations. The computational structureof each round, where the input data is mixed with the currentstate, is depicted in Fig. 2. Each W_t value is a 32-bit data wordand $K_t$is the 32-bit round dependent constant.The 32-bit values of the to variables are updated in eachround and the newvalues are used in the following round. The *IV*for these variables is given by the 256-bit constant value specifiedin [18], being set only for the first data block. The consecutivedata blocks use the partial DM computed for the previousdata block. Each SHA-256 data block is processed in 64rounds, after which the values of the variables to are addedto the previous DM in order to obtain a new value for the DM.Comparing Figs. 1 and 2, is it noticeable a higher computationalcomplexity of the SHA-2 algorithm in comparison to the SHA-1algorithm.

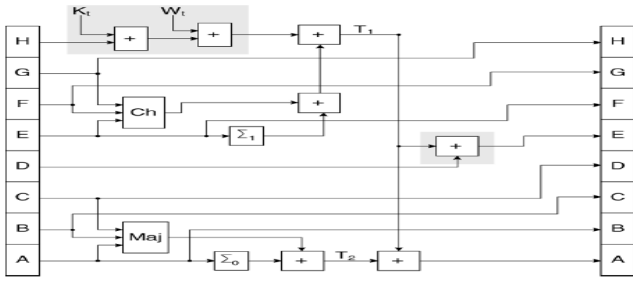

Fig. 2. SHA-2 round calculation

C. SHA512 Hash Function

The SHA512 hash algorithm computation is identical to thatof the SHA256 hash function, differing only in the size of theoperands, 64 bits instead of 32 bits as for the SHA256. The DMhas twice the width, 512 bits, and different logical functions areused [18]. The values K_t and W_t are 64 bits wide and eachdata block is composed of 16 × 64-bit words, having in total1024 bits.

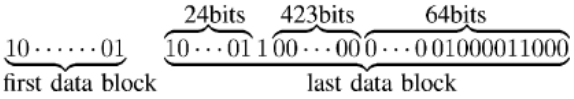

Fig. 3. Message padding for 512 bit data blocks.

D. Data Block Expansion for SHA Function

The SHA-1 algorithm computation steps described in Fig. 1are performed 80 times (rounds). Each round uses a 32-bit wordobtained from the current input data block. Since each inputdata block only has 16× 32-bits words (512 bits), the remaining64 ×32-bit words are obtained from data expansion. This expansionis performed by computing (1), where $M_t^{(i)}$ denotes thefirst 16 ×32-bit words of the th data block

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

For the SHA-2 algorithm, the computation steps shown inFig. 2 are performed for 64 rounds (80 rounds for the SHA512).In each round, a 32-bit word (or 64-bit for SHA512) from thecurrent data input block is used. Once again, the input data blockonly has 16× 32-bits words (or 64-bit words for SHA512), resultingin the need to expand the initial data block to obtain theremaining words. This expansion is performed by the computationdescribed in (2), where $M_t^{(i)}$ denotes the first 16 words ofthe th data block and the operator + describes the arithmeticaddition operation

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

E. Message Padding

In order to assure that the input data block is a multiple of 512bits, as required by the SHA-1 and SHA256specification, theoriginal message has to be padded. For the SHA512 algorithmthe input data block is a multiple of 1024 bits.

The padding procedure for a 512 bit input data block is asfollows: for an original message composed of bits, the bit “1”is appended at the end of the message (the bit), followedby zero bits, were k is the smallest solution to the equationn+1+k=448 mod 512. These last 64 bits are filled withthe binary representation of , the original message size. Thisoperation is better

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-5, ISSUE-1, 2018
DOI: 10.21276/ijcesr.2018.5.1.5

illustrated in Fig. 3 for a message with 536bits (010 0001 1000 in binary representation).

For the SHA512 message padding, 1024-bit data blocks areused and the last 128, not 64 bits, are reserved for the binaryvalue 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, depictedin Fig. 1, the values from the previous round are required.This data dependency imposes sequentiality, preventing parallelcomputation between rounds. Only parallelism withineach round can be efficiently explored. Some approaches [11]attempt to speedup the processing by unrolling each roundcomputations. However, this approach implies an obviousincrease in circuit area. Another approach [12], increases thethroughput using a pipelined structure. Such an approach, however,makes the core inefficient in practical applications, sincea data block can only be processed when the previous one hasbeen completed, due to the data dependencies of the algorithm.In this paper, we propose a functional rescheduling of theSHA-1 algorithm as described in the work [15], which allowsthe high throughput of an unrolled structure to be combined witha low hardware complexity.

A. Operations Rescheduling

From Fig. 1, it can be observed that the bulk of the SHA-1round computation is oriented towards the value calculation.The remaining values do not require any computation, asidefrom the rotation of B. The needed values are provided by theprevious round values of the variables A to D. Given that thevalue of depends on its previous value, no parallelism can bedirectly 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 ofis precomputed, producing the carry (β_t) and save (S_t) vectorsof 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 , with the precomputation, is described bythe following:

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

By splitting the computation of the value and by reschedulingit to a different computational round, the critical path of theSHA-1 algorithm can be significantly reduced. Since the calculationof the function f(B.C,D) and the partial addition areno longer in the critical path, the critical path of the algorithmis reduced to a three-input full adder and some additional selectionlogic, as depicted in Fig. 4. With this rescheduling, anadditional clock cycle is required, for each data block, since inthe first clock cycle the value of is not calculated (A_{-1} is notused). 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 datablock, as explained further on.After the 80 SHA-1 rounds have been computed, the finalvalues of the internal variables (A to E) are added to the currentDM. In turn, the DM remains unchanged until the end of eachdata block calculation [15]. This final addition is performed byone adder for each 32 bits portion of the 160-bit hash value.However, the addition of the value DM_0 is directly performedby a CSA adder in the round calculation. With this option, anextra full adder is saved and the value DM_0 calculation, thatdepends on the value A , is performed in one less clock cycle.Thus, the calculation of all the DM_j is concluded in the samecycle.

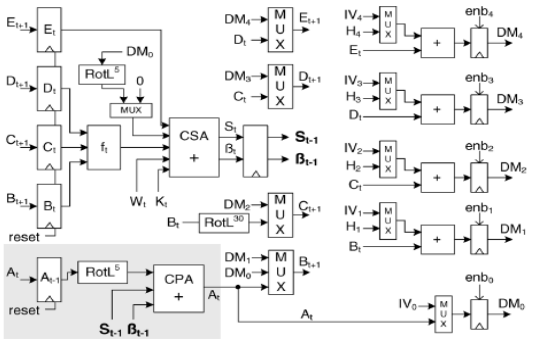

Fig. 4. SHA-1 rescheduling and internal structure.

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 toE), 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 directlyintroduced into the calculation of A, as described in (6)

$$\begin{aligned} S_0 + \beta_0 &= f(B_{\mathrm{DM}_1}, C_{\mathrm{DM}_2}, D_{\mathrm{DM}_3}) \\ &\quad + E_{\mathrm{DM}_4} + K_0 + W_0 + RotL^5(\mathbf{DM_0}) \\ A_1 &= RotL^5(A_0) + (S_0 + \beta_0) \\ &= RotL^5(0) + (S_0 + \beta_0). \end{aligned} \quad (6)$$

The *IV* can be the constant value defined in [18] or an applicationdependent value, e.g., from the HMAC or from the hashingof fragmented messages. In applications, where the *IV* is alwaysa constant, the selection between the *IV* and the current hashvalue can be removed and the constant value set in the DM registers.In order to minimize the power consumption, the internalregisters are disabled when the core is not being used.

C. Improved Hash Value Addition

After all the rounds have been computed, for a given datablock, the internal variables have to be added to the current DM.This addition can be performed with one adder per each 32 bitof the DM, as depicted in Fig. 4. In such structure, the additionof through with the current DM requires four additionaladders. Taking into account that

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

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

$$\begin{aligned} \mathrm{DM4}_i &= RotL^{30}(B_{t-3}) + \mathrm{DM4}_{i-1}; \\ \mathrm{DM3}_i &= RotL^{30}(B_{t-2}) + \mathrm{DM3}_{i-1}; \\ \mathrm{DM2}_i &= RotL^{30}(B_{t-1}) + \mathrm{DM2}_{i-1}; \\ \mathrm{DM1}_i &= B_t + \mathrm{DM1}_{i-1}. \end{aligned} \quad (8)$$

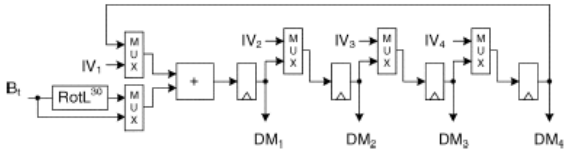

Fig. 5. Alternative SHA-1 DM addition.

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

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

The alternative hardware structure for the addition of the values Bto 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 datablock in hardware. The input data block expansion describedin (1), can be implemented with registers and XOR operations.Finally, the output value W_t is selected between the original datablock, for the first 16 rounds, and the computed values, for theremaining rounds. Fig. 6 depicts the implemented structure. Partof the delay registers have been placed after the calculation, inorder to eliminate this computation from the critical path, sincethe value $W_t$is connected directly to the the SHA-1 core. The one bit rotate-left operation can be implemented directly in therouting process, not requiring additional hardware.

IV. SHA IMPLEMENTATION

In order to evaluate the proposed SHA designs, they havebeen implemented as processor cores on a Xilinx VIRTEX IIPro (XC2VP30–7) FPGA. All the values presented in this paperfor the proposed cores were obtained after Place and Route. When implementing the ROM used to store the valuesof SHA256 or SHA512, the FPGA embedded RAMs (BRAMs)have been efficiently employed. For the SHA256 structure, asingle BRAM can be used, since the 64 32-bits fit in a single32-bit port embedded memory block. Since BRAMs have dualoutput ports of 32 bits each, the 80 ×64-bit SHA-512 constantscan be mapped to two 32-bit memory ports; one port addressesthe lower 32 bits of the constant and the other, the higher partof the same constant. Thus, only one BRAM is used to store the64-bit K_t constants.

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-5, ISSUE-1, 2018
DOI: 10.21276/ijcesr.2018.5.1.5

A. Discussion on Alternative Data Block Expansion Structures

Alternatively to the register-based structure presented inFig. 6, other structures for the SHA-1 data block expansion canbe implemented. One is based on memory blocks addressedin a circular fashion. In the presentedimplementation, theVIRTEX II embedded RAMs (BRAMs) are used. The otherstructure is based on first-inputs–first-outputs (FIFOs).A16-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 portBRAMs, the address of the new value has to be the same as thelast one, thus the first and the last position of the circular buffercoincide. For

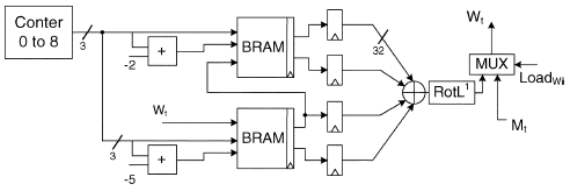

Fig. 5. BRAM-based data block expansion unit.

this scheme to work properly, the memory mustallowfor write after read (WAR). This however, is only availableon the VIRTEX II FPGA family. In technologies whereWAR isnot available, the first and last position of this circular memorycan not coincide, thus an additional position in the memory isrequired as well as an additional port. The w_{t-14} can be addressedby using the w_{t-16} (=W_t) address value subtracted by2. Identically, the w_{t-3}address can be obtained by subtracting5 from the w_{t-8} address. The implementation of the 16 bit positioncircular memory can be done by chaining two eight positionscircular memories, thus requiring less memory for theentire unit, as depicted in Fig. 5.

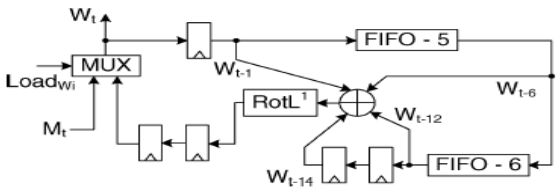

Fig. 6. FIFO-based data block expansion unit.

A16-word memory is used to

The data block expansion can also be implemented withFIFOs. In technologies where FIFOs can be efficiently used,the registers used to create the temporal delay of $W_t$can bereplaced by FIFOs. The FIFOs start outputting the valuesafter clock cycles, where is the desired delay. The FIFOshave been automatically generated by the tool fromXilinx. The resulting computational structure is depicted inFig. 10. Circular memories can also be used. For this structure(FIFO-MEM-based), modulo 5 and modulo 6 counters haveto be used, as well as memories that do not require the WARmode.In order to completely evaluate the proposed structures, theyhave been implemented on a Xilinx VIRTEX II FPGA. Theobtained results are presented in Table I. From Table I, it canbe concluded that when memories with WAR mode are available,the memory-based implementation is more efficient. It requiresonly 38 slices and two 32 ×8 bit memories, resulting ina slice occupation of only 30% of the register-based approach,at the expense of two BRAMs. When, only, memories withoutWAR mode are available, a reduction of 35% in terms of sliceusage can still be achieved, at the expense of two embeddedRAMs. These data block expansion structures for the SHA-1 algorithm can be directly mapped to the data block expansionof the SHA-2 algorithm. For the remainder of this paper, onlythe register-based unit is considered, in order to obtain less technologydependent experimental results.

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**

V. PERFORMANCE ANALYSIS AND RELATED WORK

In order to compare the architectural gains of the proposedSHA structures with the current related art, the resulting coreshave been implemented in different Xilinx devices.

A. SHA-1 Core

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-5, ISSUE-1, 2018
DOI: 10.21276/ijcesr.2018.5.1.5

In order to compare the efficiency of the DM additionthrough shift registers proposed in Section III-C, the SHA-1algorithm with variable *IV* has been implemented with bothpresented structures. Table II presents the obtained results for arealization on the VIRTEX II Pro FPGA. The obtained figuressuggest an area reduction of 5% with no degradation on theachievable frequency, resulting in a Throughput/Slice increasefrom 2.4 to 2.5 Mbit/s. In technologies where full addition unitsare more expensive, like ASICs, even higher improvements canbe expected.

The SHA-1 core has also been implemented on a VIRTEX-E(XCV400e-8) device (Column Our-Exp. in Table III), in orderto compare with [11]. The presented results in Table III for theVIRTEX-E device are for the SHA-1 core with a constant initializationvector 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 hardwareand 27% higher throughput, resulting in a 57% improvementon the Throughput/Slice (TP/Slice) metric. When comparedwith 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, 2.75times smaller than the core proposed in this paper.Table III also presents the SHA-1 corecharacteristics for theVIRTEX II Pro FPGA implementation. Both the core with aconstant initialization vector (Our-Cst.) and the one with a variable*IV* initialization (Our+IV) are presented. These results alsoinclude the data block expansion block.When compared with the leading commercial SHA-1 corefrom Helion [21], the proposed architecture requires 6% lessslices while achieving a throughput 18% higher. These two resultssuggest a gain on the TP/Slice metric of about 29%.For the SHA-1 core capable of receiving an *IV* other than theconstant specified in [18], a slight increase in the required hardwareoccurs. This is due to the fact that the *IV* can no longerbe set by the set/reset signals of the registers. This however hasa minimal effect in the cores performance, since this loadingmechanism is not located in the critical path. The decrease ofthe Throughput/Slice metric, from 2.7 to 2.5, caused by the additionalhardware for the *IV* loading is counterbalanced by thecapability of this SHA-1 core (Our+*IV*) to process fragmentedmessages.

B. SHA 256 Core

The proposed SHA256 hash function core has been also comparedwith the most recent and most efficient related art. Thecomparison figures are presented in Table IV. When comparedwith the most recent academic work [13], [22] the results showhigher throughputs, from 17% up to 98%, while achieving areduction in area above 25% up to 42%. These figures suggesta significant improvement to the TP/Slice metric in the range of 100% to 170%. When compared with the commercialSHA256 core from Helion [23], the proposed core suggests anidentical area value (less 7%) while achieving a 40% gain to thethroughput, resulting in an improvement of 53% to the TP/Slicemetric. The structure proposed by McEvoy [13] also has messagepadding hardware, however, no figures are given for theindividual cost of this extra hardware. This message padding isperformed once at the end of the message, and has no significantcost when implemented in software. Thus, the majority of theproposed cores do not include the hardware for this operation.

C. SHA 512 Core

Table V presents the implementation results for our SHA512core and the most significant related art, to out best knowledge.When compared with [22], our core requires 25% less reconfigurablelogic while a throughput increase of 85% is achieved,resulting in a TP/Slice metric improvement of 165%. From allknown 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 requirestwice as much area as our proposal and indicates a 77% higher TP/Slice metric. It should also be noticed that, the resultspresented by Lien in [11] do not include the data expansionmodule, which would increase the required area even further.

D. Integration on the MOLEN Processor

In order to create a practical platform where the SHA corescan be used and tested, a wrapping interface has been addedto integrate these units in the MOLEN polymorphic processor.The MOLEN operation [17], [24] is based on the coprocessorarchitectural paradigm, allowing the usage of reconfigurablecustom designed hardware units. The MOLEN

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-5, ISSUE-1, 2018
DOI: 10.21276/ijcesr.2018.5.1.5

computationalparadigm enables the SHA cores to be embedded in a reconfigurablecoprocessor, tightly coupled with the GPP. Theconsidered polymorphic architecture prototype uses the FPGAwith an embedded PowerPC, running at 300 MHz as the coreGPP, and a main data memory running at 100 MHz. Theimplementation is identical to the one described in [25].For this coprocessor implementations of the SHA hashfunctions, the SHA128, SHA256, and SHA512 cores, with*IV* loading, have been used. Implementations results of theSHA128 CCU indicate a device occupation of 813 slices, usinga total of 6% of the available resources on a XC2VP30 FPGA.

In this functional test the CCU is running with same clockfrequency as the main data memory, operating at 100 MHz,thus achieving a maximum throughput of 623 Mbit/s. Whencompared with the puresoftwareimplementations, capable ofachieving a maximum throughput of 4 Mbit/s and 5 Mbit/s forSHA128 and SHA256, respectively, the usage of this hybridHW/SW approach allows for a speedup up to 150 times.The CCUs for the SHA256 and SHA512 cores require 994and 1806 Slices using in total 7% and 13% of the available resources,respectively. At 100 MHz, the SHA-2 CCUs are capableof achieving a maximum throughput of 785 Mbit/s forSHA-256 and 1.2 Gbit/s for the SHA-512 hash function.

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

VI. CONCLUSION

We have proposed hardware rescheduling and reutilizationtechniques to improve SHA algorithm realizations, both inspeed and in area.With operation rescheduling, the critical pathcan be reduced in a similar manner to structures with loop unrolling,without increasing the required hardware, also leadingto the usage of a well balanced pipeline structure. An efficienttechnique for the addition of the DM is also proposed. Thistechnique allows for a substantial reduction on the requiredreconfigurable resources, while concealing the extra clockcycle delay introduced by the pipeline.Implementation results clearly indicate significant performanceand hardware gains for the proposed cores when comparedto the existing commercial cores and related academiaart. Experimental results for hybrid, hardware/software, implementationsof the SHA algorithms suggest a speed up of150 times for both hashcomputations regarding pure softwareimplementations.

REFERENCES

[1] V. Klima, “Finding MD5 collisions—A toy for a notebook.” CryptologyePrint 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, “FIPS180–2, secure hash standard (SHS),” 2002.

[4] L. Dadda, M. Macchetti, and J. Owen, “The design of a high speedASIC 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 *ProcIEEE 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 ofthe hash function SHA-256 (384, 512),” in *Proc. ACM Great LakesSymp. VLSI*, 2004, pp. 421–425.

[7] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, TLehman, and B. Schott, “Comparative analysis of the hardware implementationsof hash functions SHA-1 and SHA-512,” in *ISC*,A.H. Chanand V. D. Gligor, Eds. New York: Springer, 2002, vol. 2433, LectureNotes in Computer Science, pp. 75–89.

[8] M. McLoone and J. V. McCanny, “Efficient single-chip implementationof 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 hashfamily 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 FPGAbased SHA-256 processor,” in *FPL*, M. Glesner, P. Zipf, and M. Renovell,Eds. New York: Springer, 2002, vol. 2438, Lecture Notes inComputer Science, pp. 577–585.