Improving SHA-2 Hardware Implementations

Abstract. This paper proposes a set of new techniques to improve the implementation of the SHA-2 hashing algorithm. These techniques consist mostly in operation rescheduling and hardware reutilization, allowing a significant reduction of the critical path while the required area also decreases. Both SHA256 and SHA512 hash functions have been implemented and tested in the VIRTEX II Pro prototyping technology. Experimental results suggest improvements to related SHA256 art above 50% when compared with commercial cores and 100% to academia art, and above 70% for the SHA512 hash function. The resulting cores are capable of achieving the same throughput as the fastest unrolled architectures with 25% less area occupation than the smallest proposed architectures. The proposed cores achieve a throughput of 1.4 Gbit/s and 1.8 Gbit/s with a slice requirement of 755 and 1667 for SHA256 and SHA512 respectively, on a XC2VP30-7 FPGA.

Keywords: Cryptography, Hash functions, SHA-2 (256, 512), FPGA

1 Introduction

Cryptography is becoming an essential part of most electronic equipments that require data storing or manipulation. However, the algorithms used to enforce this security are too demanding to be implemented in software for the current required processing speeds. To achieve the require processing capability hardware components have to be used. These hardware cores are usually implemented either in dedicated ASIC cores [1–3] or in reconfigurable devices [4–7]. In this paper we propose a new hardware implementation of the SHA-2 algorithm, used in authentication systems and in the validity check of data. Several techniques have been proposed to improve the implementation of the SHA-2 algorithm. The most relevant are:

- the usage of parallel counters or well balanced Carry save Adders (CSA), in order to improve the partial additions. In technologies, like reconfigurable devices that have dedicated data paths for improving addition, this technique is not always beneficial;
- unrolling techniques that optimize the data dependency. This technique allows for an improvement in the throughput, however, it usually significantly increases the required circuit area [2, 6, 8];

- delay balancing and the usage of improved addition units, since in this algorithm this is the critical operation;
- the usage of embedded memories to store the required constant values (K_t);
- use of pipelining techniques, to achieve higher working frequencies. Due to highly dependent data computation the resulting throughput is usually not improved and more complex control logic is required [2,9].

However, the performance of the SHA-2 algorithm can be further improved with other techniques. To achieve this goal, this paper proposes operation rescheduling, that allows for an efficient use of a pipelined structure without an increase in area, and hardware reutilization techniques that allow for resource saving.

Both implementations of the SHA256 and SHA512 hash functions suggest:

- throughput per Slice efficiency metric improvement of 53% compared to commercial SHA256 cores, and more than 100% to current SHA256 academia art, and 77% for SHA512 implementations;
- a throughput of 1.4 Gbit/s for SHA256 and 1.8 Gbit/s for SHA512, with 755 and 1667 slices, on a XC2VP30-7 FPGA, respectively;
- 150 times speedup with respect to the software implementation.

The paper is organized as follows, Section 2 presents the SHA-2 algorithms. Section 3 describes the proposed design. The characteristics of FPGA implementations are presented in section 4. Section 5 presents the obtained results and compares them to related art. Section 6, concludes the paper with some final remarks.

2 SHA-2 Hash algorithm

In 1993 the Secure Hash Standard (SHA) was first published by the NIST. In 1995 this algorithm was reviewed in order to eliminate some of the initial weakness, and in 2001 new Hashing algorithms were proposed. This new family of hashing algorithms known as SHA-2, use larger digest messages, making them more resistent to possible attacks and allowing them to be used with larger blocks of data, up to 2^{128} bits, e.g. in the case of SHA512. The SHA-2 hashing algorithm is the same for the SHA256, SHA384, and SHA512 hashing functions, differing only in the size of the operands, the initialization vectors, and the size of the final digest message.

The following describes the SHA-2 algorithm applied to the SHA256 hash function, followed by the description of the SHA512 hash function, which differs mostly in the size of the operands, using 64-bit words instead of 32-bit.

SHA256 Hash function: The SHA256 Hash function produces a final digest message of 256 bits, that is dependent of the input message, composed by multiple blocks of 512 bits each. This input block is expanded and fed to the 64 cycles of the SHA256 function in words of 32 bits each (denoted by W_t). In each cycle or round of the SHA-2 algorithm the introduced data is mixed with the current state. This data scrambling is preformed by additions and logical operations, such as bitwise logical operations and bitwise rotations. The computational structure of each round of this algorithm is depicted in Figure 1. The several functions presented in this figure are described in Appendix I. The value W_t is the 32-bit data word, for the t round, and the K_t value represents the 32-bit constant that also depends on the round.

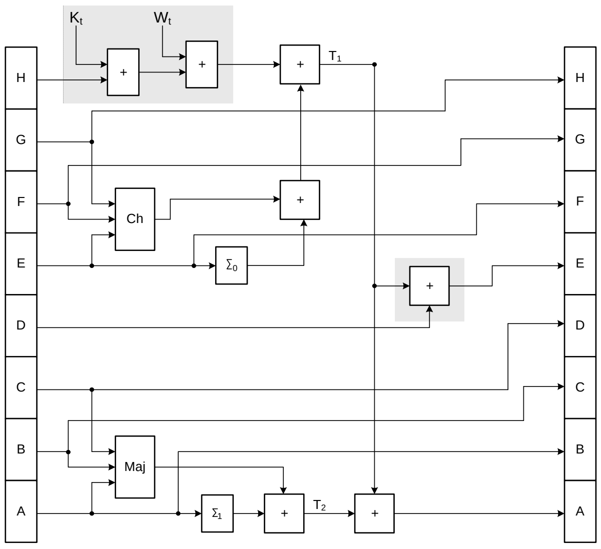

Fig. 1: SHA-2 round calculation.

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 initial values of these variables is given by the 256-bit constant value specified in [10], this value is only set for the first data block. The consecutive data blocks use the partial digest message, computed for the previous data block. Each 512 data block is processed for 64 rounds, after which the values of the variables A to H are added to the previous digest message, in order to obtain partial digest message. To better illustrate this algorithm a pseudo code representation is depicted in Figure 2. The final Digest Message (DM) for a given data stream, is given by the result of the last data block.

```
for each data_block i do
    $W = expand(\text{data\_block})$
    A = $DM_0$ ; B = $DM_1$ ; C = $DM_2$ ; D = $DM_3$
    E = $DM_4$ ; F = $DM_5$ ; G = $DM_6$ ; H = $DM_7$
    for t= 0, t≤ 63 {79}, t=t+1 do
        $T_1 = H + \sum_1(E) + Ch(E,F,G) + K_t + W_t$
        $T_2 = \sum_0(A) + Maj(A,B,C)$
        $H = G$ ; $G = F$ ; $F = E$ ;
        $E = D + T_1$
        $D = C$ ; $C = B$ ; $B = A$
        $A = T_1 + T_2$
    end for
    $DM_0$ = A + $DM_0$ ; $DM_1$ = B + $DM_1$
    $DM_2$ = C + $DM_2$ ; $DM_3$ = D + $DM_3$
    $DM_4$ = E + $DM_4$ ; $DM_5$ = C + $DM_5$
    $DM_6$ = D + $DM_6$ ; $DM_7$ = E + $DM_7$
end for
```

Fig. 2: Pseudo Code for SHA-2 algorithm.

In some higher level applications like the keyed-Hash Message Authentication Code (HMAC) [11] or when a message is fragmented, the initial hash value (IV) may differ from the constant specified in [10]. In these cases, the variables A to H are initialized by a variable Initialization Vector (IV).

SHA512 Hash function: The SHA512 hash function computation is identical to that of the SHA256 hash function, differing in the size of the operands, that are of 64 bits and not 32 bits as for the SHA256, the size of the Digest Message, that has twice the size being composed by 512 bits, and in the Σ functions described in Appendix I. This Appendix also describes the functions σ used in the data block expansion. The value W_t and K_t are of 64 bits and the each data block is composed by 16 64-bit words, having in total 1024 bits.

Data block expansion: In the SHA-2 algorithm the computation described in Figure 1 is performed for 64 rounds for the SHA256 (80 rounds for the SHA512), in each round a 32-bit word (or 64-bit for SHA512) obtained from the current data block is used. However each data block only has 16 32-bits words for SHA256 or 16 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 (1), where $M_t^{(i)}$ denotes the first 16 words of the i-th data block.

$$W_t = \begin{cases} M_t^{(i)} & , 0 \leq \mathrm{t} \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}, & 16 \leq \mathrm{t} \leq 63 \ \{\text{or } 79\} \end{cases} \tag{1}$$

Message padding: In order to assure that the input message in a multiple of 512 bits, as required by the SHA256 hash function, or 1024 for the SHA512 hash function, it is necessary to pad the original message. This message padding also comprises the inclusion of the original message dimension to the padded message. This operation can be efficiently implemented in software with a minimal cost.

3 Proposed design

In the SHA-2 algorithm, the operations that have to be performed are simple, however the data dependency of this algorithm does not allow for much parallelization. Each round of the algorithm can only be computed after the the values A to H of the previous round have been calculated (see figure 2), imposing a sequentiality to the computation. It should be noticed that in each round the computation is only required to calculate the values of A and E, since the remaining values are obtained directly from the values of the previous round, as depicted in the pseudo code of Figure 2.

In this paper, we propose a new operation rescheduling technique, a new form to initialize the algorithm, and a more efficient hardware reutilization scheme.

Operation rescheduling: In our proposal, we identified the part of the computation of a given round t that can be computed ahead in the previous round $t-1$. Only the values that do not depend on the values computed in the previous round can be computed ahead. Unlike the rescheduling technique proposed in [12] for the SHA1 algorithm, where the inter round data dependency is low, in the SHA-2 algorithm the data dependency is more complex, as depicted in Figure 1. 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 of A and E require computation and depend on all the values. In other words, the values A and E for round t can not be computed until the values for the same variables have been computed in the previous round have, as shown in (2).

$$\begin{aligned} E_{t+1} =& D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + H_t + K_t + W_t \\ A_{t+1} =& \Sigma_0(A_t) + Maj(B_t, C_t, D_t) + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + H_t + K_t + W_t \end{aligned} \tag{2}$$

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 value of K_t and W_t can be precalculated and are simply used in each round, (2) can be rewritten as:

$$\begin{aligned} \delta_t =& H_t + K_t + W_t = G_{t-1} + K_t + W_t; \\ E_{t+1} =& D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \delta_t; \\ A_{t+1} =& \Sigma_0(A_t) + Maj(B_t, C_t, D_t) + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \delta_t, \end{aligned} \tag{3}$$

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 Carry and the Save 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, depicted by the grey area in Figure 3. 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 will now require an additional clock cycle, to perform all the rounds. In the case of the SHA256 hash function 65 clock cycles are necessary to calculate the 64 rounds. As specified in the SHA-2 algorithm and depicted in Figure 2, after all rounds have been computed, the internal variables (A to H) have to be added to the previous Digest Message.

Hash value addition and initialization: As mentioned after all the rounds have been computed for a given data block, the internal variables have to be added to the current Digest Message. If this addition were to be implemented in a straightforward manner, 8 adders would be required, one for each internal variable, of 32 or 64 bits depending if SHA256 or SHA512 is being implemented. However, some hardware reuse can be achieved, by analyzing the data dependency and the fact that most of the internal variables do not require any computation, since their value is given directly by the previous values of the other variables. Taking into account that:

$$H_t = G_{t-1} = F_{t-2} = E_{t-3}; \tag{4}$$
$$D_t = C_{t-1} = B_{t-2} = A_{t-3}, \tag{5}$$

the computation of the Digest Message for the data block i can be calculated from the internal variables A and E, as:

$$\begin{aligned} DM7_i &= E_{t-3} + DM7_{i-1} \quad ; \quad DM3_i = A_{t-3} + DM3_{i-1};\\ DM6_i &= E_{t-2} + DM6_{i-1} \quad ; \quad DM2_i = A_{t-2} + DM2_{i-1};\\ DM5_i &= E_{t-1} + DM5_{i-1} \quad ; \quad DM1_i = A_{t-1} + DM1_{i-1}. \end{aligned} \tag{6}$$

Thus the calculation can be performed by only 2 addition units, as:

$$\begin{aligned} DM(j+4)_i =& E_{t-3+j} + DM(j+4)_{i-1} && ;\ 1 \leq \text{j} \leq 3\\ DM(j)_i =& A_{t-3+j} + DM(j)_{i-1} && ;\ 1 \leq \text{j} \leq 3. \end{aligned} \tag{7}$$

The selection of the corresponding part of the Digest Message (DMj), could be performed by a multiplexer. However, taking into account the sequentiality in which the values of DMj are used, a shifting buffer can be used, as depicted in the right most part of Figure 3. Since the values A_t and E_t require computation and the final value is only 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 one full adder, after the calculation of the final value of A and E, the Digest Message (DM) is added during the calculation of their final values, by a Carry Save Adder (CSA). Since the value of the previous Digest Message is known, the value can be added during the first stage of the pipeline, not being on the critical path, located in the second stage of the pipeline, where the full adders are used. In the last round the value of A and E is not calculated, being directly calculated the value of the Digest Message. During the normal round calculation only the values A_t and E_t can be computed, in these cases the input of the used CSA is put to zero, as depicted in Figure 3.

After each data block has been computed, the internal values A to H have to be re-initialized with the newly calculated Digest Message. This is performed by a multiplexer that selects either the new value of the variable of the Digest Message, as depicted in the left most side of Figure 3. Once more the values A and E are the exception. Since the final value computed for the two variables is already the Digest Message, the values are already loaded in the registers. To maintain these values during the re-initialization of the other values an enable signal is used in the A and E registers.

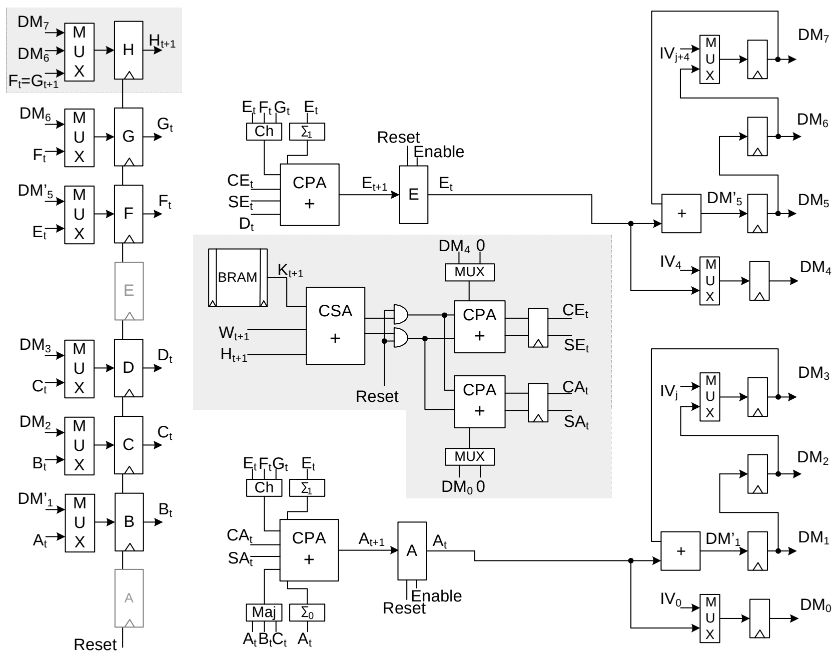

Fig. 3: SHA-2 round architecture.

In the first round the values of A to H also have to be initialized. All variables, except A and E, are simply loaded with the values in the DM registers, depicted in the leftmost part of Figure 3. For the A and E variables the value is fed through the round logic. In this case the, all the variables are set to zero (Reset) except the DM_0 and DM_4 inputs. Thus the resulting value for the A and E registers will be the initialization values of the DM registers.

In the standard for the SHA-2 algorithm the initial value of the Digest Message (loaded to the A to H variables) 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 Digest Message is no longer a constant value. I these cases the initial value is given by an Initialization Vector (IV) that has to be loaded. This loading can be performed by multiplexes at the input of the Digest Message registers. In order to optimized the architecture the calculation structure for the Digest Message can be used to load the IV, not being directly loaded into all the registers. The value of the $A1$ and $E1$ 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 to more efficiently read the final Digest Message, in a structure with an interface with smaller output ports, since less multiplexes have to be used, the values are simply shifted.

4 SHA-2 FPGA Implementation

In order to evaluate the proposed design, the resulting SHA256 and SHA512 hash functions cores have been implemented in a Xilinx VIRTEX II Pro (XC2VP30-7) FPGA using the Xilinx ISE (6.3) and SimplifyPro (8.4) tools. All the values presented our cores were obtained after Place and Route. A Custom Computing Unit (CCU) using these SHA-2 cores, has also been designed for the Molen polymorphic computational model [13], in order to fully test the cores.

In order to fully exploit the capabilities of the reconfigurable device, some design adaptation can be made. The main one lays in the use of fast carry chains for Carry Propagate Adders (CPA) instead of CSA in both the first and in the second pipeline stage, since they are able to achieve the performance with less area resources. For ASIC technologies, the structure depicted in Figure 3 is more suitable. When implementing the SHA256 hash function, a single BRAM can be used, since the 64 32-bits fit in a single 32-bit port embedded memory. However, in the SHA512 hash function the operands have 64 bits, including the constant K_t. Since the existing BRAMs do not have 64-bit ports, more than one would be required. However, they have a dual output ports of 32 bits each. Thus the 80 64-bit constants can be mapped as two 32-bit words. One port addresses the low part of the memory with the lower 32 bits of the constant and the other the high part of the memory with the higher 32 bits of the same constant. With this, only one BRAM is used to generate the 64 bit constant.

For the data block expansion in the FPGA technology considered, CPA are also used instead of CSA. The structure of the data expansion component is depicted in Figure 4.

These cores have also been integrated as a CCU for the MOLEN processor [13]. The MOLEN computational paradigm enables the SHA-2 core to be embedded in a reconfigurable co-processor, tightly coupled with the core General Purpose Processor (GPP). This, allows for a fast integration in existing software at a small cost in terms of additional area. This polymorphic architecture uses the FPGAs embedded PowerPC

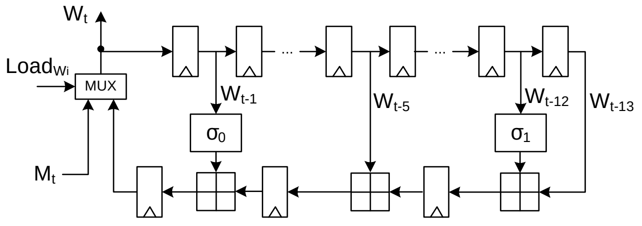

Fig. 4: SHA-2 data expansion module.

running at 300 MHz as a core GPP, with a main data memory running at 100 MHz. The implementation is identical to the one described in [12].

5 Performance analysis and related work

Even though the SHA-2 cores have been developed with a VIRTEX II Pro FPGA (XC2P30-7) as the target technology, they have also been implemented on a VIRTEX (XCV400-6) and a VIRTEX II (XC2V2000-6), in order to compare with the related art.

SHA 256 hash function core: The proposed SHA256 hash function core has been compared with the most recent and most efficient related art, for both the cores proposed in the academia and the best commercial core currently available, known by the authors. The obtained comparison figures are presented in Table 1. When compared with the most recent academic work [8, 14] the results show higher throughputs, from 17% up to 98%, while achieving a reduction in area above 25% up to 42%. These results suggest a significant improvement to the Throughput per Slice (TP/Slice) metric in the range of 100% to 170%. When compared with the commercial SHA256 core from Helion [15], 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 Throughput per Slice metric. Note that from the analyzed cores, ours is the only one capable of loading the Initialization Vector (*IV*). In the proposed FPGA implementation the logic required for the *IV* loading is located between registers as depicted in Figure 3. If the *IV* loading mechanism were not present the reconfigurable logic located in the CLB of the final register would be unused. Thus one can say that the *IV* loading mechanism is implemented at zero cost. Since this loading is performed with only an additional multiplexer located between registers, it does not influence the critical path of the circuit, as confirmed by the implementation results. The structure proposed by McEvoy [8] also has message padding

Table 1: SHA256 core performance comparison

Architecture	Sklav [14]	Our	McEv. [8]	Our	Helion [15]	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**

hardware. This message padding is performed once to the end of the message, and has no significant cost when implemented in software. Thus the majority of the proposed cores and commercial core do not include the hardware for this operation. McEvoy does not give figures for the individual cost of this extra hardware. All the SHA256 cores have the data expansion hardware.

SHA 512 hash function core: Table 2 presents the implementation results for our core and the most significant related art. The figures presented also suggest a significant reduction to the required reconfigurable area, from 25% up to 60%, while achieving a speedup to the hashing throughout. When compared with [14], the core that requires less area from those compared, the proposed core requires 25% less reconfigurable logic while a throughput increase of 85% is achieved, resulting in a Throughput per Slice metric improvement of 165%. From the known proposed SHA512 cores, the unrolled core proposed by Lien in [16] is the only one capable of achieving a higher throughput. However, this throughput is only 4% higher, while requiring twice as much area (100% more), as the one proposed in this paper. It should also be noticed that the results presented by Lien in [16], do not include the data expansion module, that would most likely influence the final throughput rate, not to mention the required area. Even in this case the proposed core indicates a Throughput per Slice metric 77% higher. All other analyzed cores have even lower values for this efficiency metric. Table 2 also presents the values for the VIRTEX II Pro implementation, for which the core was originally developed.

Polymorphic implementation of the SHA-2 cores: In order to integrate the proposed core in the existing software applications and to easily test the cores, they were integrated into the MOLEN polymorphic

[1] These values do not include the expansion data block, that in our architecture has a cost of 224 slices.

Table 2: SHA512 core performance comparison

Architecture	Sklav [14]	Lien [16]	Lien [16]	Our	McEv. [8]	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[1]	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**[1]	**0.26**[1]	**0.53**	**0.49**	**0.92**	**1.01**

processor [13]. In this processor the cores are integrated has a CCU, that can directly access the main memory and communicates with the GPP via a set of exchange registers. The core is evoked as the equivalent software function call. In order to use the proposed cores as CCU units for the MOLEN processor, some additional logic is required. The CCU for the SHA256 core requires 994 Slices using in total 7% of the available resources of the XC2VP30 FPGA. The CCU for the the SHA512 core requires 1806 Slices using in total 13% of the available resources. In this functional test the CCU is running with same clock frequency as the main data memory, operating at 100MHz. Table 3 presents the speedup achieved with the use of this hardware core, when compared with the pure software algorithm. The values presented are for the SHA256 kernel function. The values suggest a speedup up to 153 times for the SHA256

Table 3: SHA256 polymorphic performances

	Hardware		Software		
Bits	Cycles	(Mbps) ThrPut	Cycles	(Mbps) ThrPut	Kernel SpeedUp
512	354	434	30402	5.05	85
1024	552	556	60546	5.07	109
128k	50088	785	7718646	5.09	153

hash function, this speed up is achieved when the total size of the data is sufficiently large to compensate the initialization of the core, achieving a throughput of 785 Mbit/s. When only one data block is hashed the initialization time is still relevant, reducing the speedup to 85x. When at least two data block are sent the initialization becomes less significant, allowing already a speedup of 109%. The SHA512 CCU is capable of achieving a maximum throughput of 1.2 Gbit/s.

6 Conclusions

The proposed hardware rescheduling and reutilization for the SHA-2 algorithm implementations, allow for an improvement of both performance and area resources. With the operation rescheduling, we were able to reduce the critical path in a similar manner as in the loop unrolling, without duplicating the required hardware nor more complex data expansion schemes. This rescheduling also allows the usage of a well balanced pipeline structure that does not need additional control logic, and where both stages are always being used. The required reconfigurable resources are also significantly reduced due to the way the Digest Message is added to the intermediate values, requiring less multiplexors and addition units. By adding and loading the variables A and E through the round hardware, area can also be saved and one less computational cycle is required to add the Digest Message. Experimental results shown a significant gain compared to the existing commercial cores and related academia art. For the SHA256 hash function, the proposed core is capable of achieving a 17% higher throughput with an area reduction of 42%. When compared with the Helion commercial core a 40% higher throughput is achieved while reducing the required area by 7%. As an efficiency measure, the Throughput per Slice metric has been improved by 53% for the considered commercial core and more than 100% when compared with the related academic art. The SHA512 hash function implementation suggest identical results, requiring 25% less reconfigurable resources than the smallest related art while achieving a 85% higher throughput. Even when compared with the unrolled architectures, the proposed core is capable of achieving identical throughputs, only 4% slower than the fastest proposal, which uses loop unrolling, for a 50% area reduction. These values indicate an improvement to the Throughput per Slice metric of at least 77% and up to 165%. On a VIRTEX II Pro FPGA, the proposed cores are capable of a throughput of 1.37 Gbit/s for the SHA256 and 1.78 Gbit/s for the SHA512, with only 755 and 1667 slices usage receptively.

Appendix I - SHA-2 operations

In this appendix the several operations for the SAH2 algorithm are described. In Table 4 the logical operations Ch, Maj, Σ_i, and σ_i are presented, where $\oplus$ represents the bitwise XOR operation, $\wedge$ the bitwise AND operation, $ROTR^n(x)$ the right rotation operation by n bits, and $SHR^n(x)$ the the right shift operation by n bits.

Table 4: SHA256 and SHA512 functions

Designation	Function
Maj(x,y,z)	$(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
Ch(x,y,z)	$(x \wedge y) \oplus (\overline{x} \wedge z)$
$\sum_0^{\{256\}}(x)$	$ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$
$\sum_1^{\{256\}}(x)$	$ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$
$\sigma_0^{\{256\}}(x)$	$ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$
$\sigma_1^{\{256\}}(x)$	$ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$
$\sum_0^{\{512\}}(x)$	$ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$
$\sum_1^{\{512\}}(x)$	$ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$
$\sigma_0^{\{512\}}(x)$	$ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$
$\sigma_1^{\{512\}}(x)$	$ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$

References

1. L. Dadda, M. Macchetti, and J. Owen, "The Design of a High Speed ASIC Unit for the Hash Function SHA-256 (384, 512).," in *DATE*, pp. 70–75, IEEE Computer Society, 2004.
2. M. Macchetti and L. Dadda, "Quasi-pipelined hash circuits.," in *IEEE Symposium on Computer Arithmetic*, pp. 222–229, IEEE Computer Society, 2005.
3. L. Dadda, M. Macchetti, and J. Owen, "An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512).," in *ACM Great Lakes Symposium on VLSI* (D. Garrett, J. Lach, and C. A. Zukowski, eds.), pp. 421–425, ACM, 2004.
4. 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.), vol. 2433 of *Lecture Notes in Computer Science*, pp. 75–89, Springer, 2002.
5. M. McLoone and J. V. McCanny, "Efficient single-chip implementation of SHA-384 & SHA-512," *proc. of IEEE International Conference on Field-Programmable Technology*, pp. 311–314, 2002.
6. N. Sklavos and O. Koufopavlou, "Implementation of the SHA-2 hash family standard using FPGAs," *The Journal of Supercomputing*, vol. 31, p. 227248, 2005.
7. 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. Zipf, and M. Renovell, eds.), vol. 2438 of *Lecture Notes in Computer Science*, pp. 577–585, Springer, 2002.
8. R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, pp. 317–322, 2006.
9. 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.), vol. 3728 of *Lecture Notes in Computer Science*, pp. 591–600, Springer, 2005.
10. NIST, "Announcing the standard for secure hash standard, FIPS 180-1," tech. rep., National Institute of Standards and Technology, April 1995.
11. NIST, "The keyed-hash message authentication code (HMAC), FIPS 198," tech. rep., National Institute of Standards and Technology, March 2002.
12. R. Chaves, G. Kuzmanov, L. A. Sousa, and S. Vassiliadis, "Rescheduling for optimized SHA-1 calculation," in *SAMOS Workshop on Computer Systems Architectures Modelling and Simulation*, pp. 425–434, July 2006.
13. S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The Molen polymorphic processor," *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.
14. N. Sklavos and O. Koufopavlou, "On the hardware implementation of the SHA-2 (256,384,512) hash functions," *proc. of IEEE International symposium on Circuits and systems (ISCAS 2003)*, pp. 25–28, 2003.
15. HELION, "Fast SHA-2 (256) hash core for xilinx FPGA." http://www.heliontech.com/, 2005.
16. R. Lien, T. Grembowski, and K. Gaj, "A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512," in *CT-RSA*, pp. 324–338, 2004.