Rescheduling for Optimized SHA-1 Calculation

Ricardo Chaves[1,2], Georgi Kuzmanov[2], Leonel Sousa[1], and Stamatis Vassiliadis[2]

[1] Instituto Superior Técnico/INESC-ID. Rua Alves Redol 9, 1000-029 Lisbon, Portugal. http://sips.inesc-id.pt/
[2] Computer Engineering Lab, TUDelft. Postbus 5031, 2600 GA Delft, The Netherlands. http://ce.et.tudelft.nl/

Abstract. This paper proposes the rescheduling of the SHA-1 hash function operations on hardware implementations. The proposal is mapped on the Xilinx Virtex II Pro technology. The proposed rescheduling allows for a manipulation of the critical path in the SHA-1 function computation, facilitating the implementation of a more parallelized structure without an increase on the required hardware resources. Two cores have been developed, one that uses a constant initialization vector and a second one that allows for different Initialization Vectors (IV), in order to be used in HMAC and in the processing of fragmented messages. A hybrid implementation is also proposed. Experimental results indicate a throughput of 1.4 Gbits/s requiring only 533 slices for a constant IV and 596 for an imputable IV. Comparisons to SHA-1 related art suggest improvements of the throughput/slice metric of 29% against the most recent commercial cores and 59% to the current academia proposals.

1 Introduction

In current days, cryptography systems are the support for many innovations in both the industrial and the private sectors, being used from high security demanding applications, such as in banking transactions, to low security applications, like television. Three major classes of encryption algorithms exist: public key algorithms, symmetric key algorithms, and hash functions. While the first two are used to encrypt and decrypt data, the hash functions are unidirectional and do not allow the processed data to be retrieved. They are however extremely useful in data authentication and message integrity checks. Currently the most common hash functions are the MD5 and the SHA-1.Collision attacks have been found for both hash functions, however, while for MD5 they are computationally feasible on a standard desktop computer [1], the current SHA-1 attacks still require a massive computational power [2] (around 2^{69} hash operations), making it unfeasible in practical attacks for the time being.

Hash functions have the particularity of generating a small fixed length output value, the digest message or hash value, that is highly correlated with the input data, which can be significantly larger (up to 2^{64} bits). The most important characteristics of these functions is the fact that virtually no information about the input data can be obtained from the outputted hash value.An adequate hash function has a very low probability of two different input data streams generating the same hash value. The Secure Hash Algorithm 1 (SHA-1) was approved by the NIST in 1995 as an improvement to the SHA-0, and is currently used in the main security applications, such as SSH, PGP, and IPSec.

As shown in the next section, the SHA-1 computational structure is quite strait forward and with a big data dependency, not allowing for efficient pipelining. Some works improve the SHA-1 computational throughput by unrolling the calculation structure, causing a significant increase on the required hardware [3, 4]. The fully rolled architecture proposed in this paper, achieves a high throughput of the SHA-1 calculation via the rescheduling of some operations, with a minimal area increase. The proposed SHA-1 core has been implemented within the reconfigurable co-processor of a Xilinx Virtex II Pro MOLEN prototype [5].Implementation results of the proposed SHA-1 core indicate:

- A throughput of 1.4 Gbits/s with 533 Slices (2.7 Mbps per slice);
- An efficiency improvement to related art by 29% to 59%.

The hybrid implementation results indicate:

- 150x speedup with respect to the software implementation;
- 670% improvement to related art;

The paper is organized as follows: Section 2 presents an overview on the SHA-1 hash function and its computational characteristics. Section 3 describes the proposed architecture and the computational rescheduling of the SHA-1 core and the block expansion. Section 4 presents the obtained experimental results and compares them to other state-of-the-art SHA-1 implementations, both from academia and commercial companies. Section 5 concludes this paper with some final remarks.

2 SHA-1 Hash function

In 1993 the Secure Hash Standard (SHA) was first publishes by the NIST, however some weakness were found and in 1995 a revised algorithm was published [6].This revised algorithm is usually referenced as SHA-1. The SHA-1 produces a single output message digest (the output hash value) of 160-bit from an input message. The input message is composed by multiple blocks of 512 bits each. Afterwards, the input block is expanded into 80 32-bit words (denoted as W_t), one 32-bit word for each round of the SHA-1 processing. Each round computation comprises additions and logical operations, such as bitwise logical operations (in f_t) and bitwise rotations to the left (denoted by $RotL^i$), as depicted in Figure 1.

The function (f_t) calculation depends on the round being executed, as well as

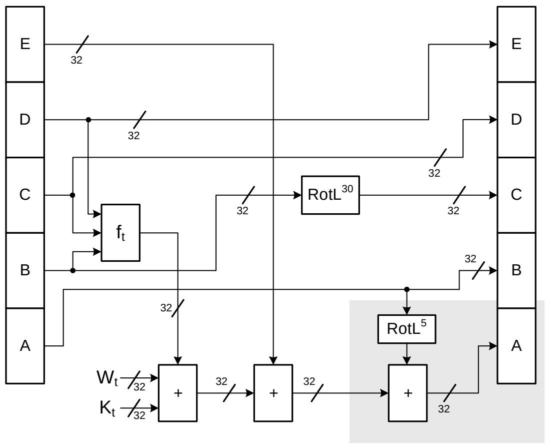

Fig. 1. SHA-1 Round calculation

the value of the constant K_t; the SHA-1 80 rounds are divided into four groups of 20 rounds each. Table 1 presents the values of K_t and the logical function executed, according to the round. In this Table, $\wedge$ represents the bitwise AND operation and $\oplus$ represents the bitwise XOR operation.

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, H_0 to H_4. After the 80 rounds have been computed, the A to E 32-bit values are added to the current Hash values. The Initialization Vector (IV) of the hash value for the first block is a predefined constant value. The output digest message is the final hash value, after all the data blocks have been computed. To better illustrate the algorithm a pseudo code representation is depicted in Figure 2. In some higher level applications such as the keyed-Hash Message Authentication Code (HMAC) [7] or when a message is fragmented, the initial hash value (IV) may differ from the constant specified in [6].

Table 1. SHA-1 functions and constants

Rounds	Function	K_t
0 to 19	$(B \wedge C) \oplus (\overline{B} \wedge D)$	0x5A827999
20 to 39	$B \oplus C \oplus D$	0x6ED9EBA1
40 to 59	$(B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)$	0x8F1BBCDC
60 to 79	$B \oplus C \oplus D$	0xCA62C1D6

Data block expansion: In the SHA-1 algorithm the computation described in Figure 1 is performed 80 times (rounds), in each round an 32-bit word obtained from the current data block is used. However, each data block only has 16 32-bits words, resulting in the need to expand the initial data block to obtain the remaining 64 32-bit words. This expansion is performed by computing (1), where $M_t^{(i)}$ denotes the first 16 32-bit words of the i-th data block.

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

```
for for each data_block do
    $W_t = expand$(data_block)
    a = $H_0$ ; b = $H_1$ ; c = $H_2$ ; d = $H_3$ ; e = $H_4$
    for t= 0, t≤79, t=t+1 do
        Temp = $RotL^5$(a) + $f_t$(b,c,d) + e + $K_t$ + $W_t$
        e = d
        d = e
        c = $RotL^{30}$(b)
        b = a
        a = Temp
    end for
    $H_0$ = a + $H_0$ ; $H_1$ = b + $H_1$ ; $H_2$ = c + $H_2$
    $H_3$ = d + $H_3$ ; $H_4$ = e + $H_4$
end for
```

Fig. 2. Pseudo Code for SHA-1 function.

In order to assure that the input message is a multiple of 512 bits, as required by the SHA-1 algorithm, it is necessary to pad the original message. This message padding also comprises the inclusion of the original message dimension to the padded message, which can be used to validate the size of the original message.

3 SHA-1 implementation

As depicted in Figure 1, the computational structure of the SHA-1 algorithm is rather strait forward. However, in order to compute the values of one round the values from the previous round are required. This data dependency imposes a sequentiality in the processing, preventing parallel computation between rounds. The only parallelism that can be efficiently explored is in the operations of each round. Some approaches [3] attempt to speedup the processing by unrolling the computation. With this technique, a speedup can be achieved, since the computation is performed as soon as the data becomes available. However, this approach carries with it a mandatory increase in the required circuit area. Other approaches, like [4], even try to increase the throughput via the usage of a pipelined structure. This however, makes the core unusable in real applications, since one data block can only be processed when the previous one has been concluded, due to the data dependency of the algorithm.

In this paper, we propose a functional rescheduling of the SHA-1 hardware units, in order to obtain the throughput increase of the unrolled architectures, while maintaining the hardware requirements identical to the fully folded ones.

Operations rescheduling: From Figures 1 and 2 it can be seen that the bulk of the SHA-1 round computation is oriented for the A value calculation. The remaining values do not require any computation, apart from the rotation of B, there values are given by the previous value of the variables A to D.

Given that the value of A is calculated with the addition of the previous value of A along with other values, no parallelism can be exploited due to the data dependency, as depicted in (2).

$$A_{t+1} = RotL^5(A_t) + [f(B_t, C_t, D_t) + E_t + K_t + W_t] \tag{2}$$

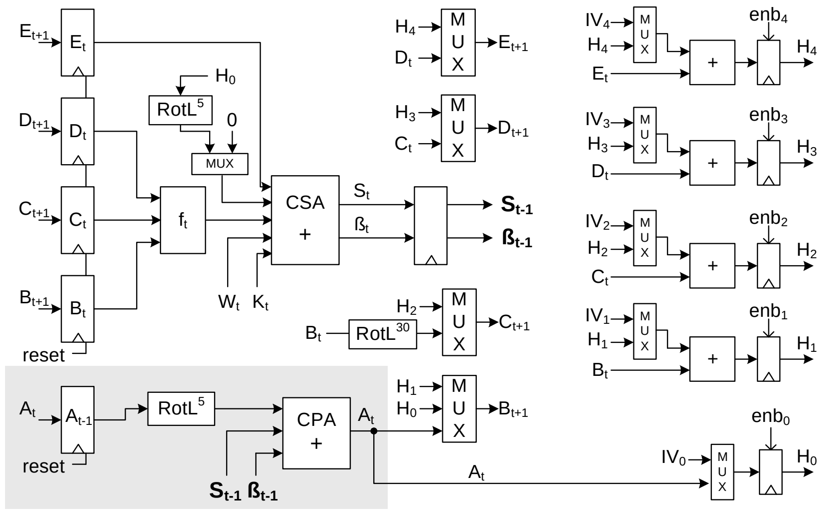

Fig. 3. SHA-1 rescheduling and internal structure

Nevertheless, since only the parcel $RotL^5(A_t)$ of (2) depends on the variable A_t, and all remaining parcels depend on variables that require no computation and do not depend on the value of A_t, some pre-computation can be performed. In 3 the parcel of (2) that does not depend of the value A is pre-computed, producing the carry (β_t) and save (S_t) vectors of the partial addition. The following holds:

$$S_t + \beta_t = f(B_t, C_t, D_t) + E_t + K_t + W_t \tag{3}$$

The calculation of the value of A_t, when part of its value is pre-computed on the previous computational cycle, as described in the following:

$$\begin{aligned} A_t &= RotL^5(A_{t-1}) + (S_{t-1} + \beta_{t-1}) \\ S_t + \beta_t &= f(B_t, C_t, D_t) + E_t + K_t + W_t \end{aligned} \tag{4}$$

By splitting the value A computation and rescheduling it to different computational cycles, the critical path of the SHA-1 algorithm is 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 3 input adder and some additional selection logic, as depicted in Figure 3. With this rescheduling an additional clock cycles is required since in the first clock cycle the value A is not calculated, since A_{-1} is not used) and in the last additional cycle the values $B_{81}, C_{81}, D_{81}, E_{81}$ are also not used. This extra additional cycle however, will be masked in the calculation of the value of the hash of each data block, as explained below.

After the 80 rounds of the SHA-1 algorithm for each data block, the final value of the internal variables (A to E) is added to the current hash value H, which remains unchanged until the end of each data block calculation, as depicted in

Figure 2. This final addition is performed by one adder for each 32 bits of the 160-bit hash value. The addition of the value H_0, however, is performed directly in the round calculation, in the CSA adder. With this option, an extra full adder is saved and the H_0 value calculation, that depends on the value A, is performed with less one clock cycle. Thus the calculation of all the hash value is concluded in the same cycle and the additional clock cycle caused by the value A calculation rescheduling is masked.

Hash value initialization: For the first data block the internal hash value has to be initialized, this is performed by adding the Initialization Vector (IV) with zero. This zero value is generated by resetting the internal values registers. This value is afterwards loaded to the internal values (B to E), through a multiplexer. Once more the value A initialization is performed in a distinct form in order to maintain the critical path as small as possible. In this case the value of H_0 is not set to the register A, instead the value A is set to zero and the value of H_0 directly introduced into the value A calculation, as described in (5).

$$\begin{aligned} S_0+\beta_0 &= f(B_{H_1},C_{H_2},D_{H_3})+E_{H_4}+K_0+W_0+RotL^5(H_0) \\ A_1 &= RotL^5(A_0)+(S_0+\beta_0) = RotL^5\ (\ 0\ )+(S_0+\beta_0) \end{aligned} \tag{5}$$

The IV can be the constant value defined in [6] or application dependent, e.g. the HMAC or in hashing fragmented messages. In this first case the multiplexer that does the selection between the IV and the current hash value, can be removed and the constant value set with the set/reset signals of the hash value registers.

In order to minimize the power consumption of the this SHA-1 core the internal registers are disabled when the core is not being used, thus reducing the amount of internal switching.

Data block expansion: As previously mentioned, the 512 bits of each data block has to be expanded in order for the 80 32-bit words (W_t) to exist. Since this expansion has to be performed for each data block,(1), it becomes more efficient to perform this operation in hardware. The implementation of the data block expansion described in (1), can be summarized to: delays, implemented by registers, and XOR operations. Finally the output value W_t is selected between the original data block, for the first 16 words, and the computed values, for the remaining values. Figure 4 depicts the implemented structure. It should be noticed that 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

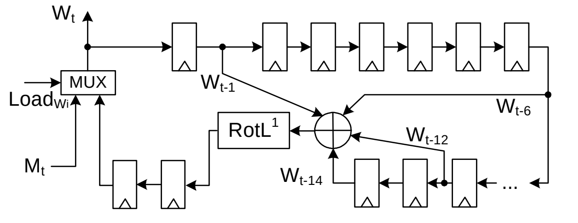

Fig. 4. Register based SHA-1 block expansion

is connected directly to the the SHA-1 core. The 4-bit XOR computation is a well suited operation for the 4-bit LUT, present in most CLBs of the Xilinx FPGAs. The one bit left rotate operation can be implemented directly in the routing process, not requiring additional hardware.

SHA-1 polymorphic processor: To create a practical platform to use and test the developed SHA-1 core, a wrapping interface has been added in order to integrate this units in the MOLEN polymorphic processor. The MOLEN paradigm [5] is based on the co-processor architectural paradigm, allowing the usage of reconfigurable custom designed hardware units. In this computational approach, the non critical part of the software code is executed on a General Purpose Processor (GPP) while the critical part, in this case the SHA-1 computation, is executed on the Custom Computing Unit (CCU).Since the hardware implemented function is called as a standard software function, the software development costs are minimal. Like in a software function, the code for the parameters passing though the XREG is included by the compiler [5].

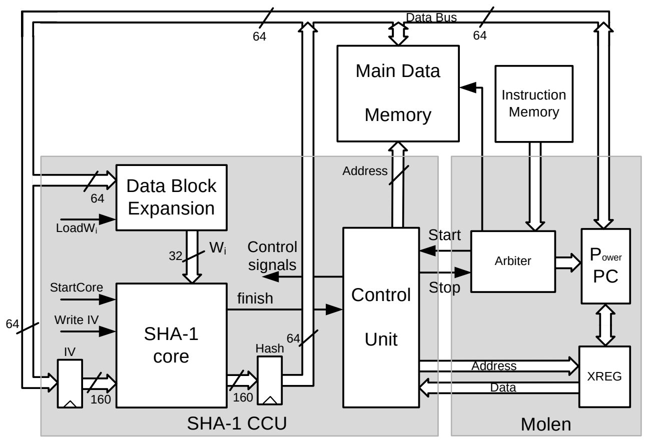

Fig. 5. SHA-1 polymorphic implementation

4 Performance analysis and related work

In order to compare the architectural gain of this operation rescheduling with the current related art, the resulting core has been implemented in a Xilinx VIRTEX II Pro (XC2VP30-7) using the ISE (6.3) Xilinx tools. A CCU using this SHA-1 core has also been designed for the MOLEN polymorphic processor [5]. This polymorphic architecture uses the FPGAs embedded PowerPC running at 300 MHz, with a main data memory running at 100 MHz.

SHA-1 core: The SHA-1 core has also been implemented on a VIRTEX-E (XCV400e-8) device (Our-Exp.), in order to compare the proposed core with the folded and the unfolded design proposed in [3]. The presented results in Table 2 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

Table 2. SHA-1 core performance comparisons

Design	Lien [3]	Lien [3]	Our-Exp.	CAST [8]	Helion [9]	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	596
Freq. (MHz)	103	73	135	127	194	230	227
TrPut.(Mbps)	659	1160	840	802	1211	1435	1420
TrPut/Slice	**1.4**	**0.8**	**2.2**	**1.4**	**2.1**	**2.7**	**2.4**

folded SHA-1 core proposed in [3], a clear advantage can be observed in both area and throughput. Experimentations suggest 20% less reconfigurable hardware occupation and 27% higher throughput, resulting in a 57% improvement on the throughput/slice metric, by adopting the proposed SHA-1 core. When compared with the unfolded architecture, the proposed core has a 28% lower throughput, however the unrolled core proposed in [3] requires 280% more hardware, resulting in a low throughput/slice, 2.75 times smaller than the core proposed in this paper.

Table 2 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 a variable IV initialization (Our+IV) are presented. These results also include the data block expansion block. The results are compared in Table 2 with the related art, including the most recent and efficient commercial SHA-1 cores known by the authors.

When compared with the leading market SHA-1 core from Helion [9], the proposed architecture requires 6% less slices while achieving throughput 18% higher. These two results originate a gain on the throughput/slice metric of about 29%.

For the SHA-1 core capable of receiving a IV other than the constant specified in [6], 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 to 2.4 caused by the additional hardware for the IV loading is counterbalanced by the capability of this SHA-1 core (Our+IV) to be used in Message Authentication applications, like the HMAC, and in the processing of fragmented messages.

Polymorphic SHA-1 implementation: For this Polymorphic implementation of the SHA-1 hash function, the core (Our+IV) with Initial Vector loading has been used.Implementations results of the SHA-1 CCU indicate a device occupation of 813 slices (see Table 4). After receiving the start signal, the SHA-1 CCU starts by reading from the exchange register the location in the main data memory of the initialization vector (IV) and after this the value of IV is read from the memory. While reading the IV from the memory, the control units also reads from the exchange register the begin and end addresses of the data to be hashed.

Table 3. SHA-1 polymorphic performances

	Hardware		Software		
Bits	Cycles	(Mbps) ThrPut	Cycles	(Mbps) ThrPut	Kernel SpeedUp
512	396	389	38280	4.01	97
1024	642	479	76308	4.03	119
128k	63126	623	9766128	4.03	155

Once the SHA-1 CCU has been initialized it goes into a loop where, it reads a 512 bit block from the main memory and computes the hash function. This loop is repeated until the current data address becomes equal to the data end address read from the exchange register. Upon conclusion, the 160 bits of the digest message are written to memory. Since the SHA-1 CCU is working at the main data memory maximum frequency, which is approximately half of the SHA-1 maximum frequency. Table 3 presents the comparison between the purely software implementation of the SHA-1 hash function and the MOLEN polymorphic approach.

Table 4. Hybrid SHA-1

Design	Lu [10]	Our +IV
Device	XCV2P100	XCV2P30-7
Slices	3441[1]	813
Freq. (MHz)	145	100
TrPut.(Mbps)	304	624
TrPut/Slice	**0.1**	**0.77**

Even though the SHA-1 algorithm can be efficiently implemented in software, achieving a throughput above 4 Mbit/s, the usage of this hybrid approach allows for a speedup up to 150 times. Note that for data streams with only a few data blocks, a lower speedup is obtained, this is due to the initial overhead required for the SHA-1 CCU initialization. Even so, a speedup of approximately 100 times is still achieved for the worst case usage. For data streams with several data blocks, the achieved speedup tends to 150 times. If throughputs above 623 Mbit/s are required, the SHA-1 core can operate at a different frequency than the main data memory. Since the SHA-1 only reads from the memory 20% of the time, a buffer can be used in order to compensate the lower frequency of the memory. This technique requires a more complex hardware structure and additional hardware resources.

This hybrid computational approach is compared with the related art in [10]. As depicted in Table 4, the proposed implementation is able to achieve a 100% higher throughput with significantly less hardware resources, thus a 670% better throughput/slice metric is obtained.

5 Conclusion

The proposed rescheduling in the SHA-1 function operations allows the computation of each round of the algorithm in two distinct clock cycles. This rescheduling permits the exploration of parallelization technics, without increasing the required hardware. With the merging of the calculation of the final value of the

[1] Synthesis results for the SHA-1 core only. An estimated value for the slice utilization has been used, for a ratio of 0.58 Slices per LUT, obtained in our SHA-1 core.

lower bits of the digest message (H_0) with the round computation of the value A, the extra cycle created by the reschedule is concealed, thus not affecting the average throughput. Two SHA-1 cores have been developed, one that uses a constant IV and a second one that allows for different initialization vectors, in order to be used in HMAC and in the processing of fragmented messages. The core with the IV loading requires some additional hardware for the registers initializations, this however does not influence the throughput since it is not located in the critical path. A polymorphic SHA-1 processor has also been proposed, capable of speeding up the hash function computation by 150%, when compared to a fully software implementation running on a PowerPC at 300MHz, at a cost of 5% occupation of a VIRTEX II Pro 30 (833 slices). When compared to a four loop unfolded architectures, the proposed core is only 28% slower, compensated by the fact that it requires 74% less logic, thus having a throughput/slice metric 172% higher. To our best knowledge the proposed core is 18% faster that any commercial SHA-1 core and academia folded art, while achieving a reduction on the required hardware. These two factors result in an improvement of the throughput/slice metric of 29% when compared with commercial products and 59% to the current academia art. The proposed core achieves a throughput of 1.4Gbits/s with 4% occupation of the used device (533 slices).
Evaluation prototype: An evaluation prototype of the hybrid SHA-1 processor is available for download at: http://ce.et.tudelft.nl/MOLEN/aplications/SHA/

References

1. Klima, V.: Finding MD5 collisions a toy for a notebook. Cryptology ePrint Archive, Report 2005/075 (2005)
2. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full sha-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
3. Lien, R., Grembowski, T., Gaj, K.: A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512. In: CT-RSA. (2004) 324–338
4. Sklavos, N., Alexopoulos, E., Koufopavlou, O.G.: Networking data integrity: High speed architectures and hardware implementations. Int. Arab J. Inf. Technol. **1**(0) (2003)
5. Vassiliadis, S., Wong, S., Gaydadjiev, G.N., Bertels, K., Kuzmanov, G., Panainte, E.M.: The Molen Polymorphic Processor. IEEE Transactions on Computers **53**(11) (2004) 1363–1375
6. NIST: Announcing the standard for secure hash standard, FIPS 180-1. Technical report, National Institute of Standards and Technology (1995)
7. NIST: The keyed-hash message authentication code (HMAC), FIPS 198. Technical report, National Institute of Standards and Technology (2002)
8. CAST: SHA-1 Secure Hash Algorithm Cryptoprocessor Core. http://http://www.cast-inc.com/ (2005)
9. HELION: Fast SHA-1 Hash Core for Xilinx FPGA. http://www.heliontech.com/ (2005)
10. Lu, J., Lockwood, J.: IPSec Implementation on Xilinx Virtex-II Pro FPGA and Its Application. In: Proceedings. 19th IEEE International Parallel and Distributed Processing Symposium. (2005) 158b – 158b