An ASIC Design for a High Speed Implementation of the Hash Function SHA-256 (384, 512)

Luigi Dadda
Politecnico di Milano
Milan, Italy
ALaRI-USI
Lugano, Switzerland
dadda@alari.ch

Marco Macchetti
Politecnico di Milano
Milan, Italy
macchett@elet.polimi.it

Jeff Owen
STMicroelectronics NV
Manno, Switzerland
jefferson.owen@st.com

ABSTRACT

An implementation of the hash functions SHA-256, 384 and 512 is presented, obtaining a high clock rate through a reduction of the critical path length, both in the Expander and in the Compressor of the hash scheme. The critical path is shown to be the smallest achievable. Synthesis results show that the new scheme can reach a clock rate well exceeding 1 GHz using a 0.13μm technology.

Categories and Subject Descriptors

B.7.1 [**Types and Design Styles**]: Algorithms implemented in hardware; B.5.1 [**Design**]: Styles (pipeline).

General Terms

Algorithms, Design, Performance.

Keywords

Hash function, Secure Hash Standard, data authentication.

1. INTRODUCTION

The integrity of a message can be controlled by means of another message usually, but not necessarily, shorter than the message to be controlled; the latter is obtained from the former by means of a special function called *hash*. We assume here that the reader has a basic knowledge of hash functions.

A number of algorithms have been proposed so far. We mention here the SHA (Secure Hash Algorithm) developed by the National Institute of Standards and Technology who published in 2002 new versions called SHA-256, SHA-384, SHA-512, (the SHA-2 family), where the numbers represent the hash length in bits [1]. The hash algorithms have been implemented in software or in hardware, the latter offering a far higher speed [2].

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

A direct translation of the Hash algorithms into a hardware ASIC implementation leads to a scheme whose critical path is rather long, allowing a clock rate not sufficiently high for many applications. In a scenario where the encryption-decryption-hashing of messages and in general of multimedia documents is applied systematically for reason of security, it becomes necessary to implement the corresponding algorithms in an hardware form fast enough to obtain that those operations are transparent to the user, i.e. the corresponding delays are not noticeable.

An additional reason is given by the need to match, as far as possible, the increasing transmission speed obtained in fiber-optic links, today reaching 40 Gbit/sec with the road open, through WDM technology, to much higher speed. This justifies the search of hashing schemes with the smallest possible critical path delay.

We are showing in this paper a solution characterized by the fastest clock rate up to now obtained (for a given technology) for a hardware implementation of the SHA-2 family in custom silicon. The new scheme is obtained by applying suitable transformations to a most simple basic, or *canonical*, scheme implementing directly the given hash algorithm. The transformations are called *delay balancing* and *quasi-pipelining*. This work differs substantially from what has been done in [3]: instead of optimizing only the compressor part of the hash function, here we have applied transformations on both the compressor and the expander part, as described in the following.

This paper is organized as follows: in Sect. 2 we describe the SHA-2 algorithm and give a canonical hardware scheme; Sect. 3 presents the optimization of the Expander circuit; Sect. 4 presents a new scheme for the Compressor circuit; in Sect. 5 we give synthesis results on an ASIC technology library; Sect. 6 concludes the paper.

2. THE SHA-2 BASIC SCHEME

The hash algorithm to be implemented is described in [1]. Due to lack of space, we do not discuss explicitly the details of the algorithm; instead, a basic scheme implementing it is shown in Fig.1. The scheme is composed by two parts: the Expander and the Compressor. The first receives as input the 16 words (of 32 bits each in SHA-256, 64 bits each in SHA-384 and SHA-512) composing a message block, delivers them to the Compressor and at the same time inserts them in a shift register. At the end of the message block and for the following 48 steps, no more words are input to the

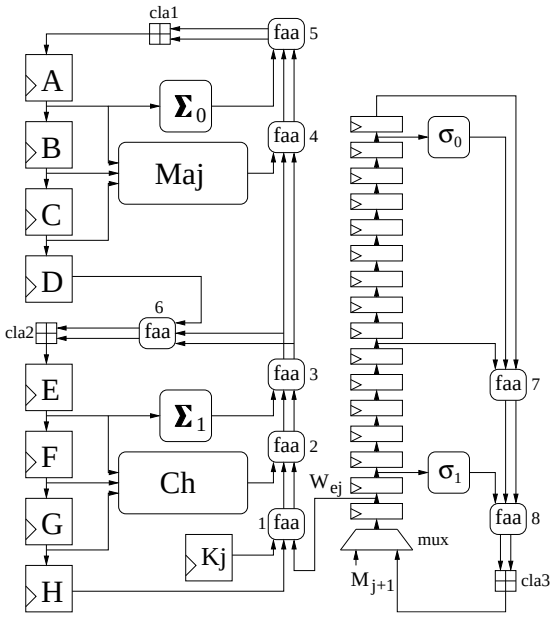

Figure 1: Canonical scheme for the SHA256 algorithm

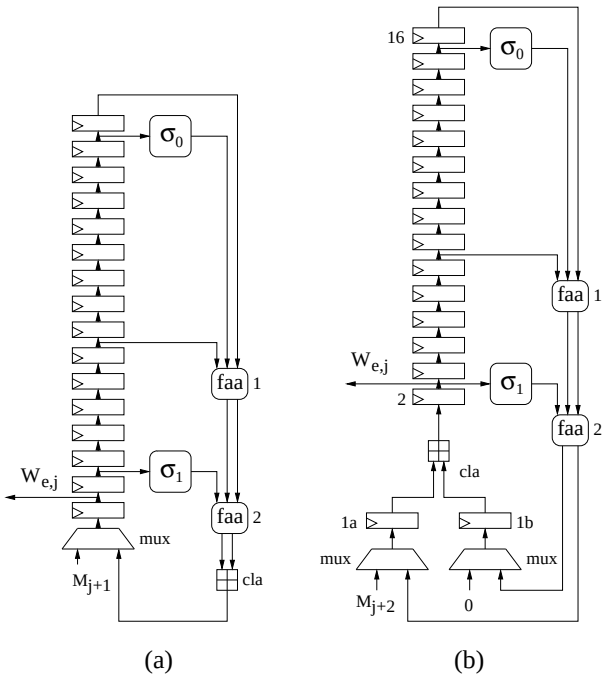

Figure 2: Implementation options for the Expander circuit

Expander, which generates 48 expansion words and sends them to the Compressor.

In the following we will consider specifically the SHA-256 case. It is straightforward to obtain the schemes for SHA-384 and SHA-512, since only the width of the connection buses is changed, from 32 bits to 64 bits.

The whole algorithm includes the preliminary operation of message partitioning and padding, that we assume as done at the system level by software. We assume also that the intermediate hashes are accumulated in a bank of registers that is not shown in the Figures, focusing our consideration on the inner part of the algorithm.

The most critical part of the algorithm is the chain of additions (modulo 2^{32}) which is necessary to obtain two main functions called α and ϵ. In order to reduce the complexity and to reach a higher clock rate, a number of adders modulo 2^{32} are replaced in Fig.1 with a sequence of carry-free adders, implemented with full adder arrays (FAA's); this solution has been applied to the SHA-1 algorithm in [2], and to the SHA-512 algorithm in [4]. However, these two solutions, together with [5] and [6], are targeted to reconfigurable devices and consequently optimize the implementation according to the peculiarities of the target FPGA platform. We instead focus our discussion on the case of ASIC implementations.

We refer to the scheme of Fig.1 as the canonical implementation.

3. INCREASING THE EXPANDER CLOCK RATE

The first step is to obtain a circuit for an Expander with a higher clock rate. This circuit is fed:

- For the first 16 clock cycles ($0 \leq j < 16$) by the 32-bits words $W_j = M_j$ which compose a block of the message.
- For the following 48 clock cycles ($16 \leq j < 64$) by the function:

$$W_j = \sigma_1(W_{j-2}) \boxplus W_{j-7} \boxplus \sigma_0(W_{j-15}) \boxplus W_{j-16}$$

where $\boxplus$ represents modulo 2^{32} (or modulo 2^{64}) addition.

Figure 1 and Fig.2a represent a scheme in which:

- The implicit chain of three modular adders has been replaced by two full-adder-arrays and a single carry propagating modular adder, implemented, for speed reason, in the form of a Carry-Look-Ahead adder (CLA). Note that a FAA accepts three 32-bits words and produces an output of two, 32-bits words: the carry from the leftmost position is discarded in order to obtain a modulo 2^{32} addition.
- The output $W_{e,j}$ (note the e subscript) to be input to the Compressor of the hash circuit is taken from the output of the first register of the shift-register. This is done in order to break the long chain of adders that begins in the Expander and ends in the Compressor part, and to define precisely the computational paths starting with $W_{e,j}$. Having labeled this with the temporal index j, the input to the multiplexer must be labeled with the timing index $j+1$, as shown with M_{j+1}. Note that the equation for $W_{e,j}$ should be re-indexed accordingly (it is unnecessary to write the corresponding equation).

In Fig.2b we have operated a transformation of the preceding scheme, based on the delay balancing method. This, as discussed in [3], consists in displacing some functional units from the critical path to a shorter path, so that the delay of the critical path is decreased and the delay of the chosen secondary path is correspondingly increased. The ideal case

would be to make the two delays identical: this, in practice is not possible, but in any case the two delays will be made closer. We realize this by displacing the CLA from the input of the first register and putting it in the connection between the first and the second register of the shift-register; in the preceding scheme this connection is just a *short-circuit*, connecting bit-wise the 32 (or 64) adjacent bit registers, but it can be used for hosting the CLA.

This modification implies that the original first register is replaced with two registers $1a$ and $1b$ in order to store the two outputs from $faa2$, whose outputs will feed cla. The output of the latter will feed register number 2. We will also need a new multiplexer, connecting two input couples to one output couple, also shown in Fig.2b. For the first 16 words indicated with M_{j+2}, only one of the corresponding inputs will be used (the other one is fed with a vector of zeros) while the other input couple will be used during the following 48 clock cycles, where the two outputs from $faa2$ will feed cla.

3.1 The critical path in the Expander

It can easily be seen by inspection that among the three paths starting from the shift register in Fig.2b the longest appears to be:

$$\sigma_0 \Rightarrow faa1 \Rightarrow faa2 \Rightarrow mux$$

The corresponding delay is approximately 3*delay(FAA), since the delays of σ_0 and that of a FAA are comparable and the delay of the multiplexer can be neglected. The only component in the paths between registers $1a$, $1b$ and register 2 is cla.

The ratio delay(CLA) versus delay(FAA) depends on the technology being used: for the $0.13\mu m$ technology library being used, a value very close to 3 has been found. In such a case the two paths are quite well balanced and, correspondingly, the maximum clock rate for Fig.2b scheme is close to twice the value possible for Fig.2a, if we neglect the setup and hold times for the registers. As an additional remark, if the output for the Compressor is taken from the register 2 of the shift-register, and if it is called $W_{e,j}$, the timing index at the multiplexer input is $j+2$: the scheme exhibits a latency of 2 clock cycles.

4. A NEW COMPRESSOR SCHEME

The scheme of the new Compressor circuit is given in Fig. 5. Its critical path is composed by: a CLA, a FAA and the one of the following units having the largest delay: Ch, Σ_1, Maj, Σ_0. Since these units have a comparable delay, we can assume that the critical path delay is given by: delay(CLA) + 2*delay(FAA). The critical path delay just obtained for the Expander is: delay(CLA) or 3*delay(FAA), whichever is the largest (in the $0.13\mu m$ technology library adopted for the hardware synthesis - see Sect.5 - they are roughly equal).

We can then conclude that the Expander maximum clock rate will be higher that the maximum clock rate obtainable in the Compressor.

4.1 Obtaining the new Compressor scheme

In order to obtain the final scheme of Fig.5 from the canonical scheme of Fig.1 we first introduce the functionally equivalent schemes shown in Fig.3 and in Fig.4.

The scheme in Fig.3 is a preliminary step toward the definition of a pipeline structure. It differs from a straight-

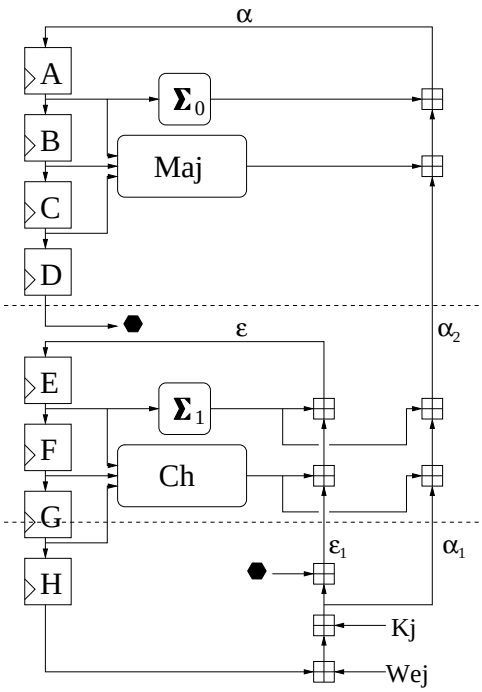

Figure 3: Initial modification of the Compressor block

forward implementation of the algorithm in the following points:

- The circuits generating ϵ and α functions are split in the bottom section of the circuit (see the dashed lines in the Figure), where the sub-functions ϵ_1 and α_1 are generated:

$$\alpha_1 = W_{e,j} \boxplus K_j \boxplus H$$

$$\epsilon_1 = \alpha_1 \boxplus D$$

- In the middle section, function ϵ and the sub-function α_2 are computed:

$$\epsilon = \epsilon_1 \boxplus Ch(E,F,G) \boxplus \Sigma_1(E)$$

$$\alpha_2 = \alpha_1 \boxplus Ch(E,F,G) \boxplus \Sigma_1(E)$$

- In the top section α is computed:

$$\alpha = \alpha_2 \boxplus Maj(A,B,C)) \boxplus \Sigma_0(A)$$

Note that in Fig.3 scheme we are certainly requiring more silicon area: it is what we accept to pay for obtaining more parallelism, i.e. a smaller critical path delay in the following schemes.

The sections of the circuit, which are delimited with dashed lines in the Figure, form an ideal partition that is the basis for building the pipelined implementation. A further transformation is shown in Fig.4 scheme, where the pipeline sections have actually been built. In this scheme:

- Registers M1 and M2 (storing ϵ_1 and α_1 respectively) have the function to separate sections τ and $\tau - 1$.
- Register L (storing α_2) serves as a separator for sections $\tau - 1$ and $\tau - 2$.

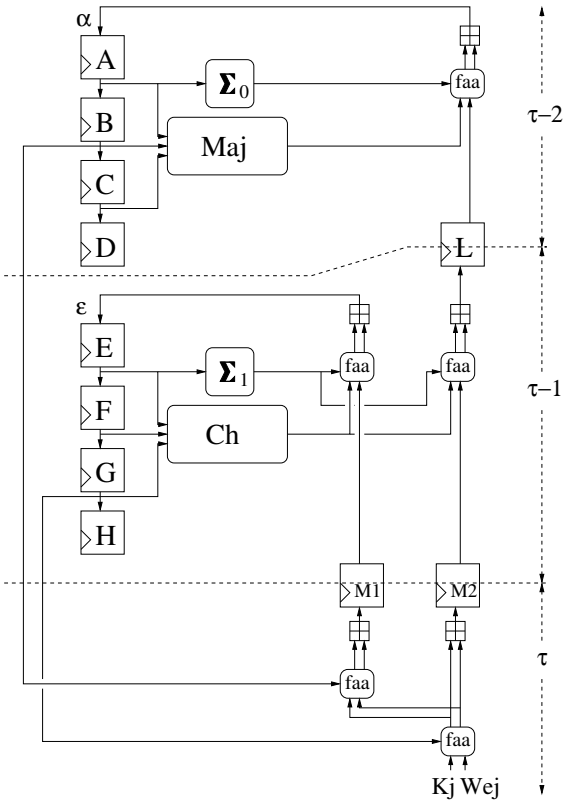

Figure 4: The modified scheme for the Compressor circuit

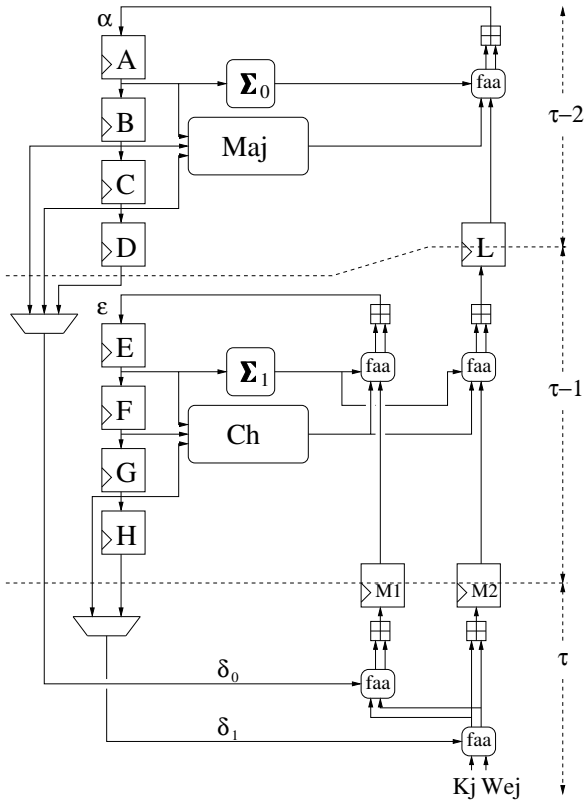

Figure 5: The Compressor scheme, valid for $0 \leq j \leq 64$

As a consequence:

- Section τ is affected by variables $W_{e,j}$ and K_j
- Section $\tau - 1$ is affected, indirectly through section τ, by variables $W_{e,j-1}$ and K_{j-1}
- Section $\tau - 2$ is affected, indirectly through τ and $\tau - 1$ sections, by variables $W_{e,j-2}$ and K_{j-2}

Moreover:

- The output from register H, crossing the border $\tau \div (\tau - 1)$, must be replaced by its *future* value, i.e. the output from G. The underlying motivation is that in Fig.1 all events are clocked simultaneously over the Compressor circuit. Instead in Fig. 4, due to the presence of registers M1 and M2, the events in section τ are ahead by one clock period with respect to the events in section $\tau - 1$. Since register H is inserted in a shift register, its *future* value can be taken from the preceding register in the shift chain, i.e. register G.
- For the same reason, the output value of register D must be replaced by the output value of B, since going from section $\tau - 2$ to section τ the connection has to cross two section borders.
- The adders (modulo 2^{32}) in all sections are replaced by Full-Adder-Arrays (in which the carry from the most significant stage is neglected) obtaining the redundant, two output words sum of three input words. The functions $\epsilon_1, \alpha_1, \epsilon, \alpha_2, \alpha$ are obtained by carry propagating (modulo 2^{32}) adders, usually for speed reason of the CLA type.

It can be verified in Fig. 4 scheme that all paths, in all sections, have a value at most equal to delay(CLA) + 2*delay(FAA), assuming that the delays in Ch, Σ_1, Maj, Σ_0 are approximately equal to the delay of a CLA.

However, the scheme of Fig.4 is valid only if the pipeline is full. At the beginning of the computation the pipeline is empty (i.e. registers M1, M2 and L are cleared and registers $A, \ldots, H$ are assumed to contain their initial values). In the next Section we will examine step by step the evolution of their content in the first three clock cycles: during those cycles the pipe will be filled. We are going to find that during the first cycle the values of D and H will be used (as in a non-pipelined system such as the scheme of Fig.3), but at the end of the first (second, and third) cycle the values transmitted from sections $\tau - 1$ (δ_1) and $\tau - 2$ (δ_0) to section τ have to change, as shown in the following detailed analysis. This requires two suitable multiplexers, as shown in Fig. 5.

4.2 The operation of Fig. 5 scheme

In the following, we report the detailed events occurring in the first three clock cycles for the scheme of Fig.5 (the value of j acts like a counter of the clock periods).

- $j = 0$. We assume that registers $A, \ldots, H$ are already loaded with the initial hash values: A(0), B(0), ... , H(0) (these are the standard constants or the *hash* of the sub-message composed by the preceding processed message block(s)). $W_{e,0}$ and K_0, which are the first

input values coming from the Expander and the table of constants, are used in the bottom section of the pipeline. δ_1 = H(0); δ_0 = D(0). The registers M1 and M2 are the only registers to be clocked and as a consequence they take their first new values after the reset. The register L is still in the reset state.

- $j = 1$. $A, \ldots, H$ are unchanged, but this time $E, \ldots, H$ are clocked while $A, \ldots, D$ still remain inactive. $W_{e,1}$ and K_1, which are the second input values coming from the Expander and the table of constants, are used in the bottom section of the pipeline. δ_1 = G(0); δ_0 = C(0). The registers M1 and M2 take their second new values and register L takes its first new value after the reset.

- $j = 2$. $A, \ldots, D$ remain unchanged; $E, \ldots, H$ get their first new values. $W_{e,2}$ and K_2, which are the third input values coming from the Expander and the table of constants, are used in the bottom section of the pipeline. δ_1 = G(1) (= F(0)); δ_0 = B(0) (still an initial value). The registers M1 and M2 take their third new values and register L takes its second new value.

The states of the two multiplexers remain unchanged until the final hash value has been computed. After the last couple $W_{e,63}$, K_{63} has been input, three more clock cycles are needed to complete the sequence of operations. The first for transferring the corresponding new values into M1 and M2 respectively; the second for generating the final values of $E, \ldots, H$, the third for the final values of , $A \ldots, D$. During this last clock cycle , $E \ldots, H$ must not be clocked.

The latency of the circuit is therefore equal to 68 clock periods, considering also the delays inside the Expander.

The final content of $A, \ldots, H$ represents the hash of the message block just processed. It will be added (word-wise, modulo 2^{32}) to the intermediate hash value stored in an hash register (8*32 bits) to obtain the final (or the next intermediate) hash value. The latter will also become the initial value of $A, \ldots, H$ registers for the processing of the following message block.

5. SYNTHESIS RESULTS

The described circuits have been implemented in VHDL using Mentor HDL Designer Pro and synthesized using Synopsys Design Compiler on a Sun workstation. The technology library being used is the STMicroelectronics HCMOS9 library, featuring 0.13μm process and 1.32 V core voltage.

For privacy reasons, related to the use of the technology library, we do not report the absolute performance values of the circuit, but we observed that the speed of the Fig. 5 scheme is about 36% higher than that of Fig.1, reaching operating frequencies well above 1 GHz.

The improvement is important, and it is obtained by pipelining the circuit without actually increasing the total number of clock cycles by a significant amount: we pass from the 65 clock cycles needed for a straightforward implementation of the algorithm to the 69 cycles needed by the proposed implementation (considering also the accumulation of the result into the intermediate hash registers). It was possible to perform these optimizations because the SHA-256 algorithm has some precise characteristics, such as the shift registers which are implicitly defined by the A,B,C,D and the E,F,G,H variables in the algorithm description.

We effectively pushed the pipelining approach to the limit, in the sense that it is not possible to create more pipeline sections and increase the total amount of clock cycles only by a small further factor. This can be understood by considering the presence of the Maj and Ch functions, which access, at the same time, the values stored in three different positions in the corresponding shift registers, and whose output value is immediately inserted back into the shift register.

Of course, the results could change when the target platform is an FPGA; in such cases, the fastest possible implementation is perhaps obtained only with a careful exploitation of the basic cells of the particular FPGA device that is being used.

In our case, the increase in speed is partially balanced by an increase in area requirements: the circuit of Fig.5 occupies 24% more space on silicon than the basic scheme.

There are some commercial implementations of the SHA-256 algorithm, see [7] for instance. Beside the fact that the implementation details are not revealed, we note that the operating frequency is much lower than the presented design (about 200 MHz in a 0.18μn process) and that pipelining is not introduced (the circuits require 65 clock cycles to produce the hash value).

6. CONCLUSIONS

In this paper we presented a high speed ASIC implementation of the SHA-256 hash algorithm. Starting from a basic canonical scheme, we applied optimizations on both the Expander and the Compressor blocks. The proposed circuit is capable of running at clock frequencies well above 1 GHz, when it is synthesized in a 0.13μm silicon process; this is obtained with a small increase in the area requirements.

7. ADDITIONAL AUTHORS

This paper is co-authored by Samantak Chakrabarti, ALaRI-USI, Lugano, Switzerland.

8. REFERENCES

[1] NIST. Announcing the secure hash standard. *Federal Information Processing Satndards Publication 180-2*, August 2002.

[2] Y. K. Kang, D. W. Kim, T. W. Kwon, and J. R. Choi. An efficient implementation of hash function processor for ipsec. *Proceedings of the 3rd Asia-Pacific Conference on ASICs*, August 2002.

[3] L. Dadda, M. Macchetti, and J. Owen. The design of a high speed asic unit for the hash function sha-256 (384,512). *Proc. of DATE 2004 Designer's Forum*, pages 70–76, 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. *Proc. of ISC 2002*, pages 75–89, October 2002.

[5] N. Sklavos and O. Koufopavlou. On the hardware implementations of the sha-2(256,384,512) hash functions. *Proc. of ISCAS 2003*, May 2003.

[6] K. Ting, S. Yuen, K. Lee, and P. Leong. An fpga based sha-256 processor. *Proceedings of the FPL 2002 Conference*, pages 577–585, September 2002.

[7] http://www.heliontech.com/core6.htm.