

On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm *

M. Knežević⁽¹⁾, K. Sakiyama⁽¹⁾, Y. K. Lee⁽²⁾ and I. Verbauwhede^{(1),(2)}

⁽¹⁾Katholieke Universiteit Leuven, ESAT/COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{mknezevi,ksakiyam,iverbauw}@esat.kuleuven.be

⁽²⁾University of California, Los Angeles, Electrical Engineering
420 Westwood Plaza, Los Angeles, CA 90095-1594 USA
jfirst@ee.ucla.edu

Abstract

In this paper we present two new architectures of the RIPEMD-160 hash algorithm for high throughput implementations. The first architecture achieves the iteration bound of RIPEMD-160, i.e. it achieves a theoretical upper bound on throughput at the micro-architecture level. The second architecture is designed by performing a gate level optimization and achieves a better performance than the first one at the cost of a larger gate area. Throughputs of 3.122 Gbps and 624 Mbps are achieved, with and without pipelining, respectively.

Keywords: RIPEMD-160, hash algorithm, efficient implementation, FPGA, throughput optimal design, retiming, DFG.

1 Introduction

Recent attacks on the most popular hash algorithms such as MD4, MD5, SHA-0 and SHA-1 [13, 11, 14, 12] necessarily demand the use of other hash algorithms. While the second generation of the SHA family is still young and there are only a few security evaluations of this algorithm, a solution can be found in the well known RIPEMD-160 hash algorithm. This algorithm was designed by Dobbertin, Bosselaers and Preneel [1] in 1996 and it is still resistant to the attacks that work against the rest of the MD family [7].

At the same time the need for high throughput optimized implementations of the crypto primitives is getting more essential in almost every networking application. Message

*This work is funded partially by IBBT, Katholieke Universiteit Leuven (OT/06/40) and FWO projects (G.0300.07 and G.0450.04). This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the EU IST FP6 projects (ECRYPT) and by the IBBT-QoE project of the IBBT.

authentication, digital signature scheme and key derivation are just some of the examples where the hash algorithms are used as a building block. Due to the recursive mode of the operations, an efficient implementation of the hash algorithms has always been a challenge.

In this paper we propose two new high-throughput architectures for the FPGA implementation of the RIPEMD-160 hash algorithm and compare them with previous work. Using the Xilinx Virtex2Pro FPGA board we achieve throughputs of 3.122 Gbps and 624 Mbps with and without pipelining, respectively.

The remainder of this paper is structured as follows. In Sect. 2 we give some background information about the RIPEMD-160 hash algorithm. Section 3 describes the theoretically throughput optimal architecture in the micro-architecture level. In Sect. 4 and 5 we show further optimization in the gate level. Implementation results and comparison with previous work are given in Sect. 6. Section 7 concludes the paper and gives some guidelines for future work.

2 RIPEMD-160 Algorithm

RIPEMD-160 shown in Alg. 1¹ is a hash algorithm that takes an input of arbitrary length (less than 2^{64} bits) and produces an output of 160-bit length after performing five independent rounds. Each round is composed of 16 iterations resulting in 80 iterations in total. RIPEMD-160 operates on 512-bit message blocks which are composed of sixteen 32-bit words. The compression function consists of two parallel datapaths as shown in Fig. 1. F_i and F'_i are non-linear functions and K_i and K'_i are fixed constants. Temporary variables A, B, C, D and E for the left and A', B', C', D' and E' for the right datapath, are initialized with

¹With \oplus we denote a modular addition.

Algorithm 1 RIPEMD-160 algorithm.

$$\begin{aligned}
 T &= \text{rol}_s(A \oplus F_i(B, C, D) \oplus X_i \oplus K_i) \oplus E \\
 E &= D \\
 D &= \text{rol}_{10}(C) \\
 C &= B \\
 B &= T \\
 A &= E \\
 T' &= \text{rol}_{s'}(A' \oplus F'_i(B', C', D') \oplus X'_i \oplus K'_i) \oplus E' \\
 E' &= D' \\
 D' &= \text{rol}_{10}(C') \\
 C' &= B' \\
 B' &= T' \\
 A' &= E'
 \end{aligned}$$

the five 32-bit chaining variables, h_0, h_1, h_2, h_3 and h_4 respectively. Chaining variables are either initialized with the fixed values to hash the first 512-bit message block or updated with the intermediate hash values for the following message blocks. Each step of the algorithm uses a different message word X_i for the left and X'_i for the right datapath. All the 16 message words are reused for each round but in a different order. For a detailed description of the algorithm please refer to [1].

3 Optimization in Micro-Architecture Level

The MD family hash algorithms can be considered as an example of digital signal processing (DSP) systems. Block diagrams are most frequently used to graphically represent a DSP system. Data flow graph (DFG) is an example of

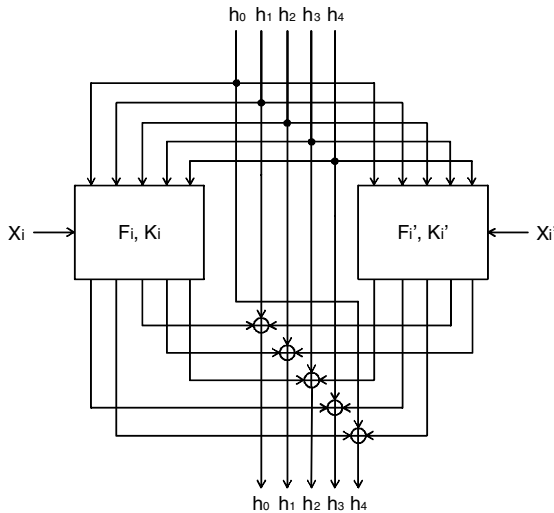


Figure 1. Compression function of RIPEMD-160 algorithm.

a block diagram, where the nodes represent computations (or functions) and directed edges represent datapaths. Each edge has a nonnegative number of delays associated with it. These unit-delay elements (often called algorithmic delays) can also be treated as functional blocks as they are implemented using registers [9]. As an example of a DFG we can look at the Fig. 2, where the functional nodes are represented with the light gray circles and registers are represented with the black squares. To each edge, that is at the input of the register, one unit-delay T_D is associated.

The *iteration bound* of the circuit is defined as

$$T_\infty = \max_{l \in L} \left\{ \frac{t_l}{w_l} \right\} \quad (1)$$

where t_l is the loop calculation time, w_l is the number of *algorithmic* delays (marked with T_D in Fig. 2) in the l -th loop, and L is the set of the all possible loops [9]. A DFG of RIPEMD-160, which is shown in Fig. 2, is derived from Alg. 1 and contains five different loops. The iteration bound is determined by the loop $B \rightarrow F \rightarrow \oplus \rightarrow \text{rol}(s) \rightarrow \oplus \rightarrow B$ and is equal to

$$T_\infty = 2 \times \text{Delay}(\oplus) + \text{Delay}(F) + \text{Delay}(\text{rol}) \quad (2)$$

Since the DSP systems are mostly implemented using sequential circuits, the *critical path* is defined as the longest path between any two storage elements [9]. The critical path of the DFG in Fig. 2 is the path marked with the bold lines ($4 \times \text{Delay}(\oplus) + \text{Delay}(\text{rol})$) and it is larger than the iteration bound. Therefore, to achieve a throughput optimal design, we need to apply some transformations on the given DFG.

In [5] the authors apply some DSP techniques on SHA-2 family hash algorithms, such as the iteration bound analysis and the *retiming* and *unfolding* transformations. By applying these techniques, an architecture whose critical path is equal to the iteration bound can be derived. In this optimization, the functional operations used in a hash algorithm,

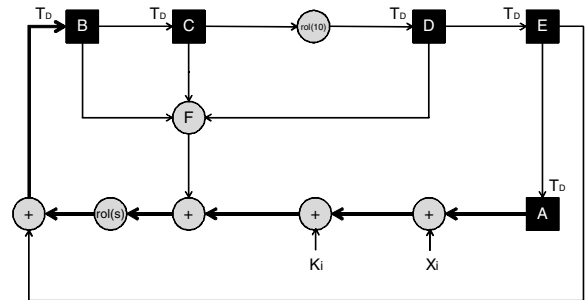


Figure 2. Data flow graph for compression function of the RIPEMD-160 algorithm.

e.g. non-linear functions and additions, are assumed to be *atomic*, i.e. a functional operation cannot be split or merged into some other functional operations. In other words, the optimization is limited to the micro-architecture level.

Retiming [6] is a widely used technique in design of the DSP systems. It is a transformation technique that changes the locations of unit-delay elements in a circuit without affecting the input/output characteristic of the circuit [9].

By applying only the retiming transformation, we can obtain a DFG of RIPEMD-160 whose critical path delay is reduced to the iteration bound. Figure 3 shows the DFG after retiming transformation and the critical path is again marked with the bold lines. Now the critical path delay is equal to the iteration bound, which means that the DFG given in Fig. 3 represents a throughput optimal architecture that achieves a theoretical upper bound in the micro-architecture level.

Due to the retiming transformations, two adders are placed between registers E and A1 now. This causes A1 to be initialized with $h_0 \oplus X_0 \oplus K_0$ instead only with h_0 . Making the values of X_i and K_i equal to zero at the last iteration, A1 becomes equal to A.

In the next section we show a gate level optimization by merging a few functional nodes, which results in an architecture with an even higher throughput.

4 Optimization in Gate Level

Observing Fig. 3 we notice that a set of functional nodes consists of four modular adders, one non-linear function and two cyclic shifts. As the critical path is in the loop, the only way of optimizing the DFG is to optimize the loop. Unfortunately, the variable cyclic shift is placed between two modular adders and prevents us from using a carry save adder (CSA) instead. However, in this section we show how another approach can be used for further optimization of the loop.

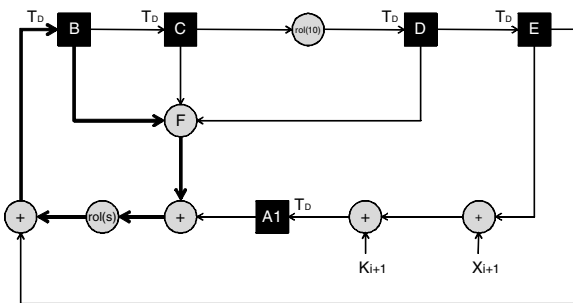


Figure 3. Throughput optimized DFG of the RIPEMD-160 algorithm.

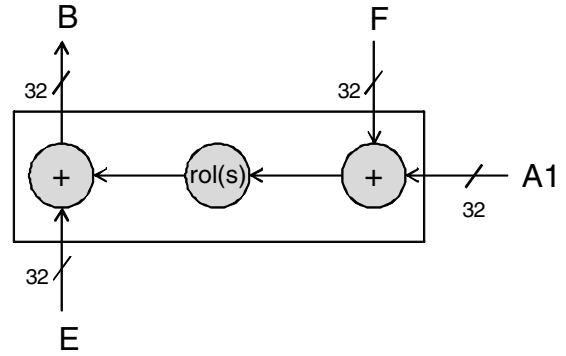


Figure 4. ADD+ROT part of the loop in the RIPEMD-160 algorithm.

Let us consider a simple example shown in Fig. 4 where the part of the loop with three 32-bit inputs, A1, E and F, and one output, B, is shown. To simplify discussion we omit the input s which represents the number of bits in the cyclic shift. The functionality of this block is given in Fig. 5. After adding two operands A1 and F, the cyclic shift is applied, and the operand E is finally added resulting in the output B. We define *carry_1* as the carry bit that may occur in the result of adding $(32 - s)$ LSB's of A1 and F. This bit will be propagated to the s MSB's of the sum. Another carry bit (*carry_2*) may occur in the result of the whole addition and will be discarded before the rotation starts (due to the modular addition).

In order to optimize the given block, we rotate operands A1 and F before adding them together. Doing this we have to take care of the carry bits *carry_1* and *carry_2*. The

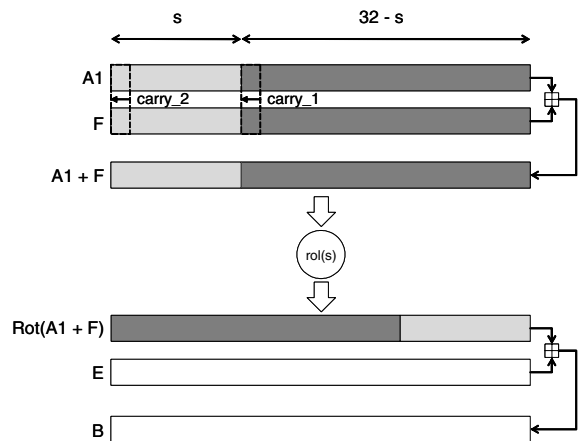


Figure 5. Functionality of the original ADD+ROT part in the RIPEMD-160 algorithm.

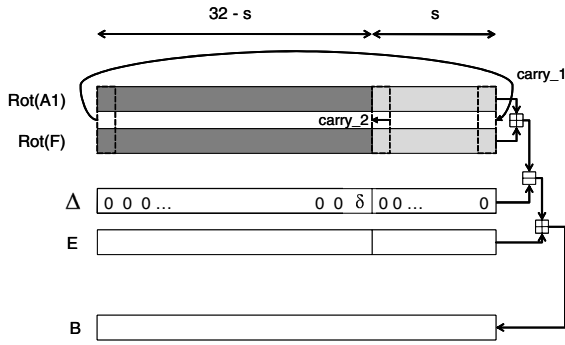


Figure 6. Functionality of the ADD+ROT part after the first transformation.

latter carry must not be propagated after the rotation of the two operands. To prevent this carry propagation we can subtract vector Δ from the sum of $Rot(A1)$ and $Rot(F)$ as it is shown in Fig. 6. The bit value δ is equal to 1 if $carry_2 = 1$, otherwise $\delta = 0$. Beside the $carry_2$, we also need to take care of the $carry_1$ bit. This carry must be added to the rotated operands $Rot(A1)$ and $Rot(F)$ (see Fig. 6). Depending on the $carry_1$ and $carry_2$ we have four different possibilities. In order to reduce the critical path, we compute all four possibilities in parallel.

The only drawback of this architecture is that subtraction of Δ and addition of $carry_1$ is still executed within the loop which does not decrease the critical path. As the addition is an associative operation we can subtract vector Δ from the operand E before entering the loop. This logic that decreases the critical path is shown in Fig. 7.

Using the similar design criteria, we add $carry_1$ after adding $Rot(A1)$, $Rot(F)$ and E. Instead of using one additional adder for this operation, we use a CSA and the fact that $carry_form$ of CSA is always shifted to the left for one position before the final addition. In this way we just need

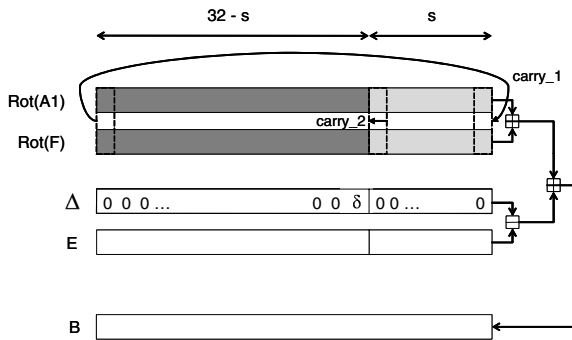


Figure 7. Functionality of the ADD+ROT part after optimization.

Table 1. Selecting the appropriate input of the MUX.

MUX	$carry_2$	$carry_1$
0	0	0
1	0	1
2	1	0
3	1	1

to change the LSB bit of the shifted $carry_form$ depending on the $carry_1$.

The architecture describing this whole logic is shown in Fig. 8. To achieve a high-throughput of the algorithm we prepare all four possibilities in parallel. The values of $carry_1$ and $carry_2$ determine which input of the MUX will be propagated to the output value B as it is shown in Table 1. Note here that input E1 is obtained as $E1 = D - \Delta$ where Δ is chosen such that $\delta = 1$. Input E represents the case where $\delta = 0$.

5 Final High-Throughput Architecture

Following the design principles described in the previous section and using additional retiming transformation we can obtain the final high-throughput architecture for RIPEMD-160 algorithm. The first step is to use the throughput optimized part of the loop instead of the original one (see Fig. 8). A DFG that shows this architecture is given in Fig. 9. The optimized part of the loop is denoted as a black box (ADD+ROT).

The fact that we use the optimized part of the loop moves now the critical path between registers E and A1. This

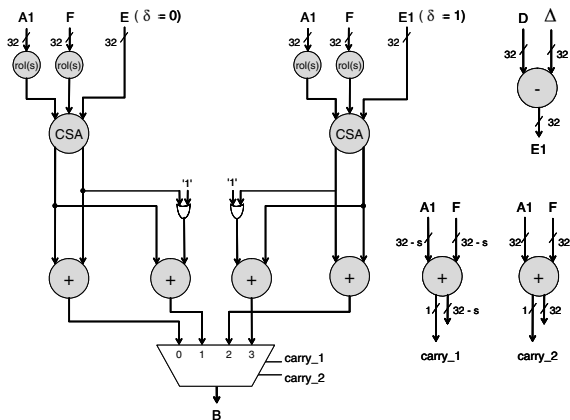


Figure 8. Throughput optimized ADD+ROT part of the loop in the RIPEMD-160 algorithm.

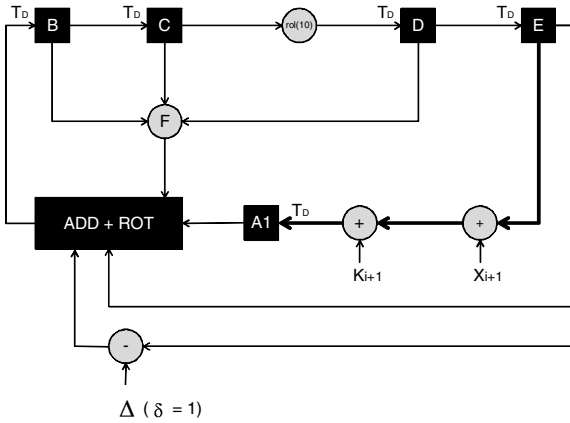


Figure 9. DFG of the RIPEMD-160 architecture with optimized ADD+ROT part of the loop.

problem can easily be solved by using CSA instead of two adders. Figure 10 shows this solution.

As the critical path occurs between the output of E and the input of B (bold line) now, we need to introduce one more register E1 and move subtractor at its input as it is shown in Fig. 11. In this way the critical path is placed within the loop again and the high-throughput architecture of RIPEMD-160 is finally obtained.

6 Implementation Results and Comparison With Previous Work

In order to verify the speed of our architectures we have implemented the proposed solutions using GEZEL, a design environment for exploration, simulation and implementa-

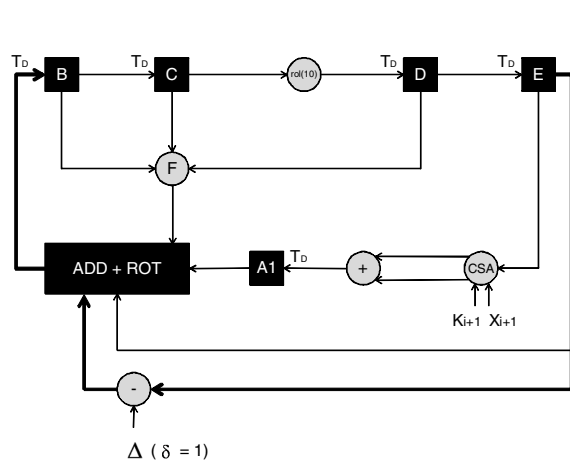


Figure 10. Using CSA instead of two adders changes the critical path.

tion of domain specific architecture [3]. Both implementations were verified on Xilinx Virtex2Pro FPGA board. Our results and comparison with previous work are given in Table 2.

In [10] the authors propose a RIPEMD processor that performs both RIPEMD-128 and RIPEMD-160 hash algorithms and they separately implement RIPEMD-160 processor for the comparison purpose. In order to achieve high throughput of 2.1 Gbps they use a pipelining technique. However, due to the recursive mode of operation of the RIPEMD-160 algorithm, using pipelining is possible only for hashing independent messages. Our architecture with the optimized loop can easily be pipelined and for this special case we could achieve a throughput of 3.122 Gbps. Pipelining is done by replicating a DFG of the RIPEMD-160 algorithm five times, one for each of the five different nonlinear functions. However, as it comes at a high price of the occupied area and can be useful only in limited numbers of application, we do not provide more details about the pipelined implementation.

Here, we can also notice that the speed of the architecture with optimized ADD+ROT part is 10 % faster than the version without. On the other hand the size is 65 % larger due to the parallel processing shown in Fig. 8 and using one additional register (see Fig. 11).

7 Conclusion

We showed how the iteration bound analysis can be used for the high-throughput implementation of the RIPEMD-160 hash algorithm. Since the iteration bound is a theoretical minimum of the critical path, there is no further through-

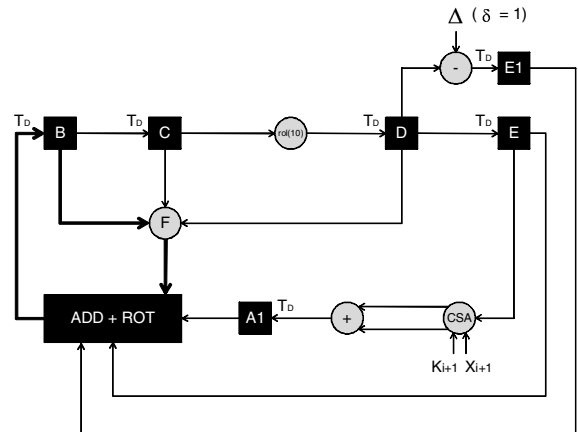


Figure 11. Throughput optimized DFG of the RIPEMD-160 algorithm with optimized ADD+ROT part.

Table 2. Implementation results and comparison with previous work.

Design	FPGA chip	Frequency [MHz]	Cycle/Block	Throughput [Mbps]	Size
[8] ⁽¹⁾	EPF10K50	26.66	162	84	1964 LC
[2] ⁽²⁾	XCV300E	42.90	337	65	2008 LUT
[4] ⁽³⁾	XC2V4000	43.47	162	137.4	14,911 LUT
[10] ⁽⁴⁾	XC2V250	73	82	455	2014 CLB ⁽⁵⁾
This work					
without optimized loop	XC2VP30	90.97	82	568	2721 LUT
with optimized loop	XC2VP30	100.05	82	624	4410 LUT

⁽¹⁾ This is a unified architecture of MD5 and RIPEMD-160 hash algorithms.

⁽²⁾ This is a unified architecture of MD5, RIPEMD-160, SHA-1 and SHA-256 hash algorithms.

⁽³⁾ This is a unified architecture of MD5, RIPEMD-160 and SHA-1 hash algorithms.

⁽⁴⁾ In the original paper throughput of 2.1 Gbps is shown for hashing 5 independent messages (pipelining).

To make a fair comparison we consider throughput for a single message only.

⁽⁵⁾ In the original paper the use of 2014 CLBs, 4006 FGs and 1600 DFFs is reported. One CLB in Virtex2 FPGA family contains 8 LUTs [15].

put optimization in the micro-architecture level. Thus, we further optimized our architecture in the gate level, achieving the final high-throughput implementation of the RIPEMD-160 algorithm. This approach can be a guideline for a high-throughput implementation of other popular hash algorithms.

References

- [1] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In *Fast Software Encryption, Lecture Notes in Computer Science, vol. 1039*, pages 71–82. Springer-Verlag, Berlin, 1996.
- [2] S. Dominikus. A hardware implementation of md-4 family hash algorithms. In *Proc. ICECS '02, Vol. III*, pages 1143–1146, 2002.
- [3] http://rijndael.ece.vt.edu/gezel2/index.php/Main_Page. Gezel Development Environment.
- [4] E. Khan, M. Watheq El-Kharashi, F. Gebali, and M. Abd-El-Barr. Design and performance analysis of a unified, reconfigurable hmac-hash unit. In *IEEE Transaction on Circuits and Systems*, pages 2683–2695, 2007.
- [5] Y. K. Lee, H. Chan, and I. Verbauwhede. Iteration Bound Analysis and Throughput Optimum Architecture of SHA-256 (384,512) for Hardware Implementations. In *Information Security Applications, 8th International Workshop, WISA 2007, LNCS 4867, S. Kim, H. Lee, and M. Yung (eds.), Springer-Verlag*, pages 102–114, 2007.
- [6] C. Leiserson, F. Rose, and J. Saxe. Optimizing synchronous circuitry by retiming. In *Third Caltech Conference on VLSI*, pages 87–116, 1983.
- [7] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. On the collision resistance of ripemd-160. In *Information Security*, 2006.
- [8] C. Ng, T. Ng, and K. Yip. A Unified Architecture of MD5 and RIPEMD-160 Hash Algorithms. In *Proceedings of the 2004 International Symposium on Circuits and Systems (ISCAS'04)*, pages 889–892, 2004.
- [9] Keshab K. Parhi. *VLSI Digital Signal Processing Systems - Design and Implementation*.
- [10] N. Sklavos and O. Koufopavlou. On the Hardware Implementation of RIPEMD processor: Networking High Speed Hashing, up to 2 Gbps. In *Computers and Electrical Engineering*, pages 361–379, 2005.
- [11] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions md4 and ripemd. In *Advances in Cryptology, EUROCRYPT 2005*, 2005.
- [12] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full sha-1. In *Advances in Cryptology, CRYPTO 2005*, 2005.
- [13] X. Wang and H. Yu. How to break md5 and other hash functions. In *Advances in Cryptology, EUROCRYPT 2005*, 2005.
- [14] X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on sha-0. In *Advances in Cryptology, CRYPTO 2005*, 2005.
- [15] Xilinx. Virtex-II Platform FPGA User Guide.