Throughput Optimized SHA-1 Architecture Using Unfolding Transformation

Yong Ki Lee[1], Herwin Chan[1] and Ingrid Verbauwhede[1,2]
[1] *University of California, Los Angeles* [2] *Katholieke Universiteit Leuven*
{jfirst, herwin, ingrid} @ ee.ucla.edu

Abstract

In this paper, we analyze the theoretical delay bound of the SHA-1 algorithm and propose architectures to achieve high throughput hardware implementations which approach this bound. According to the results of FPGA implementations, 3,541 Mbps with a pipeline and 893 Mbps without a pipeline were achieved. Moreover, synthesis results using 0.18μm CMOS technology showed that 10.4 Gbps with a pipeline and 3.1 Gbps without a pipeline can be achieved. These results are much faster than previously published results. The high throughputs are due to the unfolding transformation, which reduces the number of required cycles for one block hash. We reduced the required number of cycles to 12 cycles for a 512 bit block and showed that 12 cycles is the optimal in our design.

1. Introduction

Hash functions are primitive components in many cryptographic systems. The key features of hash functions are the one way function property, collision resistance and their high speed. Hash functions are commonly used in Digital Signature Algorithm [9] and message authentications. Considering that data sizes and communication speeds are dramatically increasing every year, low throughput of hash functions can be a bottle neck in the digital and/or communications systems.

In this paper, we analyze the delay bound of SHA-1 using the iteration bound [8], which defines the minimum delay in which an algorithm may run independent of implementation architectures. We propose architectures which are close to the bound. Our architectures achieve 3,541 Mbps with a four-step pipeline and 893 Mbps without a pipeline in FPGA implementations. The pipeline technique [3] is used to increase throughput and improve hardware reuse. The synthesis results using 0.18μm CMOS technology show that 10.4 Gbps with a pipeline and 3.1 Gbps without a pipeline can be achieved. We designed the architectures to achieve high throughput using the unfolding transformation, i.e. the reduction of the required number of cycles for a 512 bit message block. The highest throughput is achieved with the 12 cycle version in both FPGA implementations and CMOS synthesis. This optimum point is verified by mathematical analysis. This paper also shows the performances of design trade-off for how the reductions in the number of cycles affect the performance in SHA-1 implementations. Moreover, this theoretical analysis method can be used to estimate other hash algorithms.

The remainder of the paper is structured as follows. In section 2, the background of SHA-1 is explained, and related works are presented in section 3. The theoretical delay bound of SHA-1 is analyzed in section 4, and our design architectures and their analyses are presented in section 5. The performance results and a comparison with some conventional implementations are given in section 6, and we conclude in section 7.

2. SHA-1 Algorithm

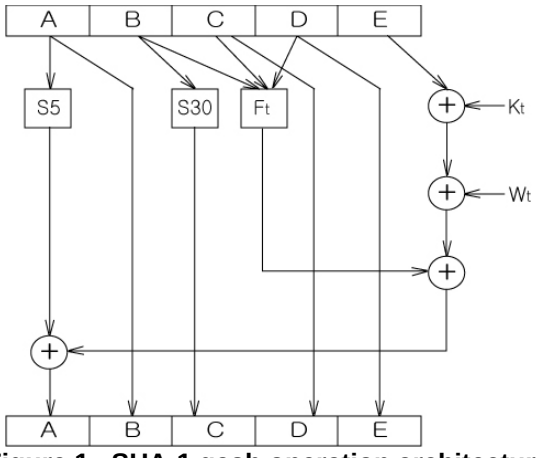

Figure 1. SHA-1 gash operation architecture

SHA-1 [1], which is one of the most popular hash algorithms, was issued by the National Institute of Standards and Technology (NIST) in 1995. SHA-1 takes input data of length less than 2^{64} bits and gives output of 160 bit length. After packing and padding an arbitrary length message into one or multiple 512 bit blocks, each block is processed separately. Each block message requires 4 rounds of hash operations, where each round is composed of 20 hash operations. The differences among the rounds are on a scrambling constant, K_t, and a nonlinear operation, F_t, where t represent the t-th hash operation. The architecture of one step hash operation is illustrated in Figure 1 and the mathematical expression is described in Figure 2.

$$TEMP_t = S5(A_t) + F_t(B_t, C_t, D_t) + E_t + W_t + K_t;$$
$$E_{t+1} = D_t;\ D_{t+1} = C_t;\ C_{t+1} = S30(B_t);$$
$$B_{t+1} = A_t;\ A_{t+1} = TEMP_t;$$

Figure 2. Equation set for SHA-1 hash operation

In Figure 1 and Figure 2, $S5$ and $S30$ represent 5 and 30 circular left shifts respectively and W_t is a 32 bit register value. The number of W_t registers can be either 80 or 16. If 80 registers are to be used, the W_t values can be calculated before the hash operations, and if 16 registers are to be used, the W_t values are dynamically updated during the hash operations. In this paper, all the implementations and syntheses are done using 16 registers.

3. Related Works

In [2] and [3], a four-step pipeline technique is used for high throughputs and hardware reuse. Since a scrambling constant and a nonlinear function are changed each round, the operation blocks of each round can be reused by a pipeline. In [3], a high throughput architecture is proposed which requires 40 cycles for 80 hash operations. By reducing the required number of cycles by half, better throughput is achieved while lowering the energy consumption. In [2] and [3], the architectures are implemented in FPGA, and in [4], [5] and [6], the architectures are implemented in ASIC.

Even though there are many published papers for high throughput SHA-1 implementations including the above references, there has been no publication saying about the delay bound of SHA-1 and how to achieve the bound.

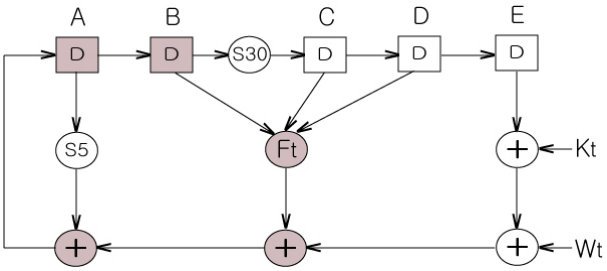

Figure 3. Data-flow graph of SHA-1

4. Iteration Bound of SHA-1

The maximum attainable speed of the SHA-1 algorithm is determined by calculating the iteration bound [8]. The data-flow graph (DFG) representation is depicted on Figure 3, where D represents a delay node and the other nodes represent the individual functions in a SHA-1 hash operation. Since the order of the four adders in SHA-1 does not make a difference on the mathematical calculation, there are a few different ways to represent a SHA-1 DFG. Figure 3 is the DFG which has the smallest bound.

A loop is a path that begins and ends at the same node. For example, $D(A) \rightarrow S5 \rightarrow + \rightarrow D(A)$ is a loop, where $D(A)$ is the delay node named A. If t_l is the loop calculation time and w_l is the number of delay nodes in the l-th loop, the l-th loop bound is defined as t_l / w_l. The iteration bound is defined as follows.

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

where L is the set of all the possible loops.

Since $Delay(S5), Delay(S30) << Delay(F_t) < Delay(+)$, the loop which corresponds the gray nodes in Figure 3 is the one having the maximum loop bound. Therefore, the iteration bound of SHA-1,

$$T_\infty = Delay(+) + \frac{Delay(F_t)}{2}.$$

This means that a delay of one SHA-1 hash operation can not be less than this bound.

5. Analysis of Unfolded SHA-1 Architectures

In order to approach the bound given in section 4, we performed the unfolding transformation [8]. The unfolding transformation combines the operations of several iterations into a single cycle. By unfolding, the hidden concurrencies can be parallelized. In this section, we increase the unfolding factor in powers of 2, and show how the delay approaches the iteration bound.

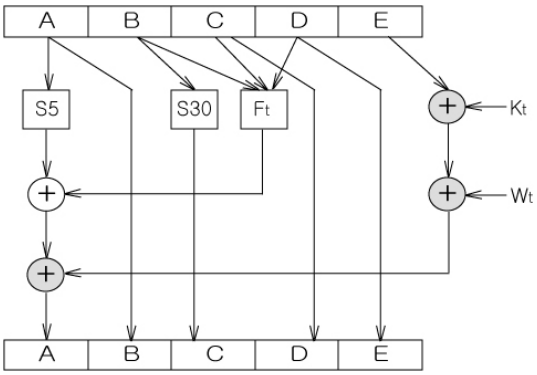

Figure 4. One hash operation per cycle

5.1. One hash operation per cycle

We start with an architecture without the unfolding transformation. Since the additions dominate the delay in SHA-1 hash operations, the number of additions in the critical path must be minimized before other functions are considered.

Among the four additions in a hash operation, only one addition can be parallelized. Therefore, the critical path has a delay of three additions as shown on Figure 4. The shaded nodes represent the critical path.

5.2. Two hash operations per cycle

By unfolding two hash operations, i.e. with unfolding factor two, four additions out of eight additions can be parallelized. Therefore, the critical path has a delay of four additions. Figure 5 shows the architecture which has the minimum critical path delay for two hash operations per cycle. Therefore, the critical path is composed of four additions and one circular shift. A similar approach of unfolding two hash operations is done at [3].

Since the delay of circular shifts is negligible in hardware implementations, we will count only additions, +, and non-linear functions, F_t, in the delay analysis. Therefore, the normalized loop delay with unfolding factor two, $\hat{T}_2$, is as follows.

$$\hat{T}_2 = \frac{4 \times Delay(+)}{2} = 2 \times Delay(+)$$

where we divide by 2 to normalize by the unfolding factor.

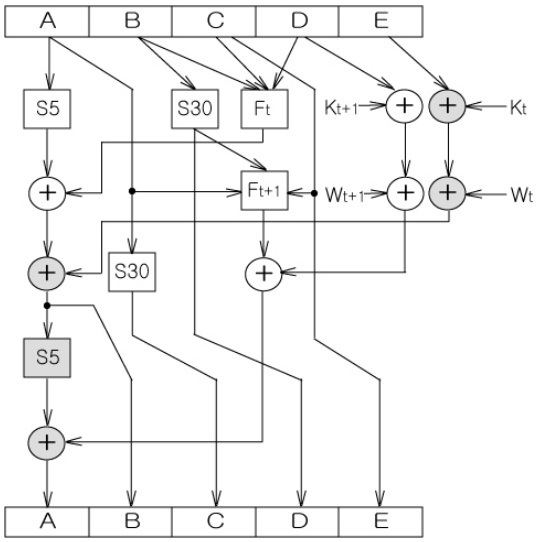

Figure 5. Two hash operations per cycle

By unfolding by the unfolding factor 2, we have a gain of one addition delay comparing with the architecture of one hash operation per cycle. However, it has still much larger delay than the iteration bound.

5.3. More than two operations per cycle

In order to approach the iteration bound, we increase the unfolding factor. When $n > 2$, each addition of 2 hash operations to a cycle, i.e. increasing unfolding factor by two, causes a delay increase by $2 \times Delay(+) + Delay(F_t)$. This fact is explained in sub-section 5.4. Note that we ignore the delay of the circular shifts. Therefore, the normalized loop delay with unfolding factor n is as follows.

$$\hat{T}_n = \frac{4 \times Delay(+) + (n-2) \times \{Delay(+) + Delay(F_t)/2\}}{n}$$

$$= Delay(+) + \frac{Delay(F_t)}{2} + \frac{2 \times Delay(+) - Delay(F_t)}{n}$$

where n is a even number ≥ 2, and the last term is obviously a positive value since $Delay(+) > Delay(F_t)$.

So, if we have the limit of n,

$$\lim_{n \to \infty} \hat{T}_n = Delay(+) + \frac{Delay(F_t)}{2}$$

This means our architectures approach the iteration bound, which is calculated using the DFG in section 4.

Table 1. The total numbers of additions and non-linear functions in critical paths for a block hash

n	1	2	4	5	8	10	16	20	40	80
N_{add}	246	168	132	126	120	120	126	132	168	246
N_{non}	0	0	22	27	36	40	49	54	76	117

However, n cannot be arbitrarily increased.

For the unfolding factor n, $80/n$ cycles are required for 80 hash operations. We allocate two extra cycles, one is for getting the input and initializing registers and the other is for making and giving the output. Therefore, the total required number of cycles per block hash is $80/n+2$, and the total delay with unfolding factor n, $\hat{T}_n^*$, is as follows.

$$\hat{T}_n^* = \left(\frac{80}{n}+2\right)\times \hat{T}_n \times n$$
$$= \left(\frac{80}{n}+2\right)\times\left((n+2)\times Delay(+)+\frac{n-2}{2}\times Delay(F_t)\right)$$

Therefore, the total number of addition delays, N_{add}, and the total number of non-linear function delays, N_{non}, in the critical path for one block hash can be formulated as follows.

$$N_{add} = \left(\frac{80}{n}+2\right)\times(n+2) = 2n+\frac{160}{n}+84$$
$$N_{non} = \left(\frac{80}{n}+2\right)\times\left(\frac{n-2}{2}\right) = n-\frac{80}{n}+38$$

In order to distribute 80 operations equally over each cycle, the possible values of n are divisors of 80, i.e. 1, 2, 4, 5, 8, 10, 16, 20, 40 and 80. The optimal value of n can be shown by calculating every possible case. Table 1 shows the comparison of N_{add} and N_{non} values for all the possible n. Assuming that $Delay(F_t) << Delay(+)$, $n=8$ is the optimal point.

5.4. Eight operations per cycle

In this sub-section, we show the architecture for the unfolding factor eight, which achieves the highest throughput. Using equations in Figure 2 iteratively, A_{t+8}, B_{t+8}, C_{t+8}, D_{t+8} and E_{t+8} can be expressed in terms of A_t, B_t, C_t, D_t and E_t. The expanded equations are given in Figure 6. Having the values of $K_t \sim K_{t+7}$ and $F_t \sim F_{t+7}$ is straightforward since the values only depend on the time indexes and the given parameters. However, $W_t \sim W_{t+7}$ values must be used a little bit carefully due to their dynamic changes during 2~4 round operations. Nevertheless, having the values of $W_t \sim W_{t+7}$ is also straightforward.

In Figure 6, A and F represent an addition delay and a non-linear function delay respectively, and the values of right side of equations are the assignment delays for each equation. The assignment delays can be calculated as follows.

$$Delay(\alpha+\beta) = \max\{Delay(\alpha), Delay(\beta)\} + A$$
$$Delay(F_i(\alpha,\beta,\gamma)) = \max\{Delay(\alpha), Delay(\beta), Delay(\gamma)\} + F$$

The variables, α, β and γ, are either register values or functions whose delays can be similarly calculated. As shown in Figure 6, if $i \geq t+2$ in $TEMP_i$, a increase in unfolding factor by two causes a delay increase by $2\times Delay(+)+Delay(F_t)$ as stated in the sub-section 5.3.

Assignment Delay

$$TEMP_t = \underbrace{[S5(A_t)+F_t(B_t,C_t,D_t)]}_{A+F} + \underbrace{[E_t+K_t+W_t]}_{2A} \quad \Rightarrow 3A$$

$$TEMP_{t+1} = \underbrace{S5(TEMP_t)}_{3A} + \underbrace{[\{D_t+K_{t+1}\}+\{W_{t+1}+F_{t+1}(A_t,S30(B_t),C_t)\}]}_{2A+F} \quad \Rightarrow 4A$$

$$TEMP_{t+2} = \underbrace{S5(TEMP_{t+1})}_{4A} + \underbrace{[\{C_t+K_{t+2}+W_{t+2}\}+F_{t+2}(TEMP_t,S30(A_t),S30(B_t))]}_{4A+F} \quad \Rightarrow 5A+F$$

$$TEMP_{t+3} = \underbrace{S5(TEMP_{t+2})}_{5A+F} + \underbrace{[\{S30(B_t)+K_{t+3}+W_{t+3}\}+F_{t+3}(TEMP_{t+1},S30(TEMP_t),S30(A_t))]}_{5A+F} \quad \Rightarrow 6A+F$$

$$TEMP_{t+4} = \underbrace{S5(TEMP_{t+3})}_{6A+F} + \underbrace{[\{S30(A_t)+K_{t+4}+W_{t+4}\}+F_{t+4}(TEMP_{t+2},S30(TEMP_{t+1}),S30(TEMP_t))]}_{6A+2F} \quad \Rightarrow 7A+2F$$

$$TEMP_{t+5} = \underbrace{S5(TEMP_{t+4})}_{7A+2F} + \underbrace{[\{S30(TEMP_t)+K_{t+5}+W_{t+5}\}+F_{t+5}(TEMP_{t+3},S30(TEMP_{t+2}),S30(TEMP_{t+1}))]}_{7A+2F} \quad \Rightarrow 8A+2F$$

$$TEMP_{t+6} = \underbrace{S5(TEMP_{t+5})}_{8A+2F} + \underbrace{[\{S30(TEMP_{t+1})+K_{t+6}+W_{t+6}\}+F_{t+6}(TEMP_{t+4},S30(TEMP_{t+3}),S30(TEMP_{t+2}))]}_{8A+3F} \quad \Rightarrow 9A+3F$$

$$TEMP_{t+7} = \underbrace{S5(TEMP_{t+6})}_{9A+3F} + \underbrace{[\{S30(TEMP_{t+2})+K_{t+7}+W_{t+7}\}+F_{t+7}(TEMP_{t+5},S30(TEMP_{t+4}),S30(TEMP_{t+3}))]}_{9A+3F} \quad \Rightarrow 10A+3F$$

$$E_{t+8} = S30(TEMP_{t+3}),\ D_{t+8} = S30(TEMP_{t+4}),\ C_{t+8} = S30(TEMP_{t+5}),\ B_{t+8} = TEMP_{t+6},\ A_{t+8} = TEMP_{t+7}$$

Figure 6. Equation Set for Delay Optimized Eight Operations per cycle

Table 2. Results of FPGA Implementations for Critical Path Optimization

Required cycles for a 512 bit block	82 Cycles	42 Cycles	22 Cycles	12 Cycles	24 Cycle (pipeline)
Area in slices (including RAM)	1,446	1,575	1,742	2,394	4,258
Critical Path Delay	12.8	17.6	27.2	47.8	24.1
Frequency (MHz)	78.1	56.8	36.8	20.9	41.5
Throughput (Mbps)	488	693	856	893	3,541
Energy per block	1.16mJ	0.93mJ	0.96mJ	1.30mJ	0.81mJ

Table 3. Results of CMOS Synthesis for Critical Path Optimization

Required cycles for a 512 bit block	82 Cycles	42 Cycles	22 Cycles	12 Cycles	24 Cycle (pipeline)
Area in gates (including RAM)	26,939	30,797	39,580	54,133	124,643
Critical Path Delay	3.99	5.32	8.56	13.75	8.19
Frequency (MHz)	250.6	187.9	116.8	72.7	122.1
Throughput (Mbps)	1,564	2,291	2,718	3,103	10,419

6. Implementations in FPGA and Synthesis for ASIC

Our designs were implemented using GEZEL [7], a design environment for exploration, simulation and implementation of domain specific architecture. GEZEL designs and their associated test-benches can be automatically translated to synthesizable VHDL codes. Using this environment, many different designs can be quickly described, simulated and compared.

We implemented the SHA-1 algorithm with unfolding factors $n =$ 1, 2, 4 and 8 to verify our theoretical results. The results for the FPGA implementation (Virtex2 XC2V1000) are given in Table 2, and the synthesized results for an ASIC using TSMC 0.18µm standard cell library are given Table 3. The throughputs are calculated using the following equation:

$$Througput = \frac{Frequency}{\#\ of\ Cycles} \times (512\ bits)$$

For the four-step pipeline, we used the 22 cycle version since the 12 cycle version cannot be equally divided in four parts. Note that only 10 cycles out of 12 cycles are devoted to the hash operations. Since each round of the pipeline version requires 6 cycles (5 cycles for hash operations and one for either initialization or finalization), the total number of cycles of the pipeline version is 24 cycles.

As a result, in FPGA implementations, the 12 cycle version achieved 893 Mbps and the pipeline version achieved 3,541 Mbps. In CMOS synthesis, the 12 cycle version achieved 3.1 Gbps and the pipeline version achieved 10.4 Gbps. We also measured the energy consumptions of FPGA implementations. According to the results, the pipeline version consumes the least energy per 512 bit message block, and among the non-pipeline versions, the 42 cycle version consumes the least energy. Even though the 12 cycle version achieved the most throughputs in both cases, some other versions could be preferred due to the costs of area and power.

The total delay of a SHA-1 calculation is the product of the number of cycles and the one cycle delay. It is compared with the total number of addition delays, N_{add}, in the Figure 7. Even though N_{add} does not account for the delays of non-linear functions, (the delay of additions dominates the total delay) the results show their linear dependency on each other. The

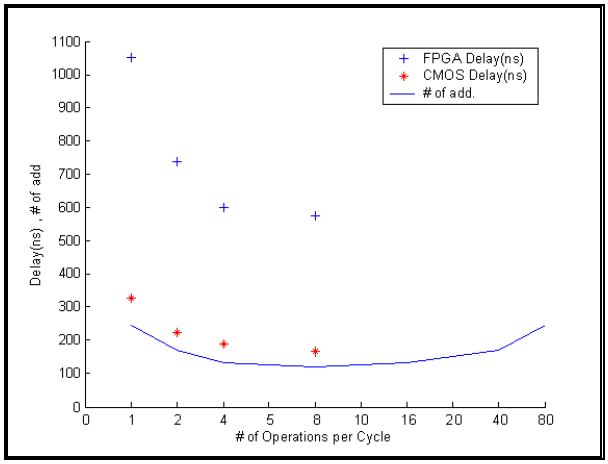

Figure 7. Comparison the total number of critical path additions and total delay for one block hash

Table 4. The comparison between our results and others

	[2]	[3]	[4]	[5]	[6]	Our Proposal			
Technology	Xilinx FPGA V100ecs144	Xilinx FPGA V150bg352	0.25u ASIC	0.18u ASIC	0.13u ASIC	Virtex 2 FPGA XC2V1000		0.18u ASIC	
Area	1,578 CLBS	N/A	20,536 Gates	70,170 Gates(1)	9,859 Gates	2,394 Slices	4,258 Slices	54,133 Gates	124,643 Gates
Frequency (MHz)	72	55	143	116	333.3	20.9	41.5	72.7	122.1
Cycles	84	40	82	80	85	12	24	12	24
Pipeline	4 steps	4 steps	None	None	None	None	4 steps	None	4 steps
Throughput (Mbps)	1,700	2,816	893	824.9	2,006	893	3,541	3,103	10,419

(1) This is a unified solution for MD5, SHA1 and RIPEMD160, and this area includes RAM

* All our proposals include RAM areas.

results of the other unfolding factors can be easily predicted due to this linearity. Therefore, we can conclude $n = 8$ is the optimum unfolding factor based on both mathematical analysis and implementation results.

All the programming is done at register transfer level and we have mostly concentrated on optimizing micro-architecture rather than focusing lower-level optimization. We believe that more careful implementations can achieve a better throughput and/or a less area. Nevertheless, the throughputs are the fastest results among ever published results. The comparison with other implementations is described in Table 4.

7. Conclusion

We analyzed the iteration bound of SHA-1 and proposed throughput optimized SHA-1 architectures which approaches the bound. The implementations in FPGA achieved 893 Mbps without a pipeline and 3,541 Mbps with a pipeline, and syntheses using 0.18μm CMOS technology achieved 3.1 Gbps without a pipeline and 10.4 Gbps with a pipeline. The high throughputs were possible by the unfolding transformation. Moreover, we showed that 12 cycle versions are optimal in our design approach by mathematical analysis.

Our analysis method and the architecture can be applied to other hash algorithms. Moreover, since we showed how the reductions of cycles affect the performance, our results can be used to predict performance changes for some other implementations when our idea is applied.

8. Acknowledgement

This research has been supported by UC Micro and NSF (Grant SRC-2003-HJ-1116).

9. References

[1] Secure Hash Standard, National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-2, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[2] N. Sklavos, E. Alexopoulos and O. Koufopavlou, "Networking Data Integrity: High Speed Architectures and Hardware Implementations," *The International Arab Journal of Information Technology*, Vol. 1, No, 0, July 2003.

[3] H. Michail, A.P. Kakarountas, O. Koufopavlou and C.E. Goutis, "A Low-Power and High-Throughput Implementation of the SHA-1 Hash Function," *IEEE International Symposium on Circuits and Systems (ISCAS)*, Kobe, Japan, 23-26 May, 2005.

[4] Y. Ming-yan, Z. Tong, W. Jin-xiang and Y. Yi-zheng, "An Efficient ASIC Implementation of SHA-1 Engine for TPM," *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems*, December 6-9, 2004.

[5] G. T S and T S B Sudarshan, "ASIC Implementation of a Unified Hardware Architecture for Non-Key Based Cryptographic Hash Primitives," *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, 2005.

[6] A. Satoh and T. Inoue, "ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS," *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, 2005

[7] Gezel development environment, http://rijndael.ece.vt.edu/gezel2/index.php/Main_Page.

[8] K.K. Parhi, VLSI Digital Signal Processing Systems: Design and Implementation, Weley, 1999,pp. 43~61 and 119~140.

[9] Digital Signature Standard, National Institute of Standards and Technology, Federal Information Processing Standards Publication 186-2, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf