A 1 Gbit/s Partially Unrolled Architecture of Hash Functions SHA-1 and SHA-512

Abstract. Hash functions are among the most widespread cryptographic primitives, and are currently used in multiple cryptographic schemes and security protocols, such as IPSec and SSL. In this paper, we investigate a new hardware architecture for a family of dedicated hash functions, including American standards, SHA-1 and SHA-512. Our architecture is based on unrolling several message digest steps and executing them in one clock cycle. This modification permits implementing majority of dedicated hash functions with the throughput exceeding 1 Gbit/s using medium-size Xilinx Virtex FPGAs. In particular, our new architecture has enabled us to speed up the implementation of SHA-1 compared to the basic iterative architecture from 544 Mbit/s to 1.1 Gbit/s using Xilinx XCV1000. The implementation of SHA-512 has been sped up from 717 to 929 Mbit/s for Virtex FPGAs, and exceeded 1 Gbit/s for Virtex-E Xilinx FPGAs. In this paper, we present the details of our designs for SHA-1 and SHA-512, our design methodology, results of static timing analysis and results of experimental testing based on the SLAAC-1V PCI FPGA board.

Keywords: Hash Functions, SHA-1, SHA-512, Unrolled Architecture, Basic Iterative Architecture, IPSec.

1. Introduction

Hash functions are very common and important cryptographic primitives. Their primary application is their use for message authentication, integrity, and non-repudiation as a part of the Message Authentication Codes (MACs) and digital signatures [1-3].

The current American federal standard, FIPS 180-2, recommends the use of one of the four hash functions developed by National Security Agency (NSA) and approved by NIST [4, 5]. By far the most widely accepted of these four functions is SHA-1 (Secure Hash Algorithm-1), a revised version of the standard algorithm introduced in 1993. The best attack against this algorithm is in the range of 2^{80} operations, which makes its security equivalent to the security of Skipjack [6] and the Digital Signature Standard (DSS) [7].

After introducing a new secret-key encryption standard, AES (Advanced Encryption Standard) [8], with three key sizes, 128, 192, and 256 bits, the security of SHA-1 did not any longer match the security guaranteed by the encryption standard. Therefore, an effort was initiated by NSA to develop three new hash functions, with the security equivalent to the security of AES with 128, 192, and 256 bit key respectively. This effort resulted in the development and standardization of three new hash functions referred to as SHA-256, SHA-384, and SHA-512 [4, 5].

All four standardized algorithms have a similar internal structure and operation. All of them are based on sequential processing of consecutive blocks of data, and therefore cannot be easily sped up by using pipelining or parallel processing (at least when only one stream of data is being processed).

The majority of reported implementations of SHA-1 based on the current generation of FPGA devices, such as Virtex [9], can only reach the throughputs up to 500 Mbit/s [10-16]. The higher speeds can only be accomplished by using more expensive FPGA devices, such as Virtex-E or Virtex II (see Table 1). Similarly, the FPGA implementations of SHA-512 based on the medium cost Virtex devices reach the speeds in the range of 700 Mbit/s [11, 12].

Significantly higher speeds might be required for applications such as High Definition Television (HDTV), videoconferencing, Virtual Private Networks, etc. [17, 18]. Our goal was to propose, implement, and verify a new architecture of standard hash functions that would allow them to be executed with the throughputs in the range of 1 Gbit/s using medium cost FPGA devices, such as Xilinx Virtex 1000.

2. Partially Unrolled vs. Basic Iterative Architecture of Hash Functions

A general block diagram common for all four SHA standards and many other dedicated hash functions is shown in Fig. 1. An input message passes first through the preprocessing unit which performs padding and forms message blocks of the fixed length, 512 or 1024 bits, depending on the hash function. The preprocessing unit passes message blocks to the message scheduler unit. Message scheduler unit generates message dependent words, W_t, for each step of the message digest. The message digest unit performs actual hashing. In each step, it processes a new word generated by the message scheduler unit. The message digest is the most critical part of the implementation, as it determines both the speed and area of the circuit.

The most straightforward implementation of the message digest, most often used in practice is shown in Fig. 2a. It is called the basic iterative architecture (or just *basic architecture*) . In this architecture, registers R and H are first both

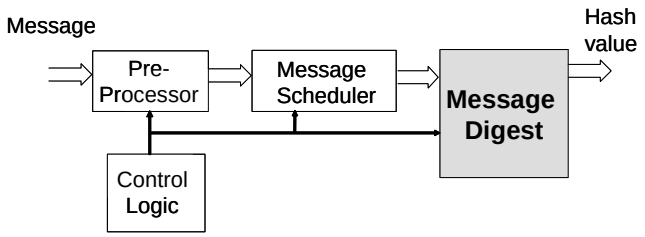


Fig. 1. General block diagram of the hardware implementation of a dedicated hash function, such as SHA-1 and SHA-512

initialized with a value of the constant initialization vector, IV. Subsequently, the architecture executes one step of the message digest per one clock period. In each step t, the message digest accepts a different message dependent word, W_t, and a different step dependent constant, K_t. After executing all steps, the result of the last step stored in the register R is added to the previous value of the register H. Then, the processing of the message digest resumes for a new set of the message dependent words, W_t, corresponding to the new block of the message.

Two straightforward ways of speeding up hardware implementations of hash functions (and any other logic functions) are parallel processing using multiple instantiations of the basic iterative architecture, and pipelining. Out of these two methods, pipelining is more attractive, because of the smaller area penalty. Nevertheless, both of these architectures are able to improve an average circuit throughput only under the assumption that multiple independent streams of data are processed simultaneously. If a single long message needs to be hashed, none of these architectures offers *any* improvement in terms of an execution time.

A new architecture of the dedicated hash functions investigated in this paper is shown in Fig. 2b. It is called *partially unrolled architecture*. In this architecture, k steps have been "unrolled" and are executed in the same clock cycle. As a result, the total number of clock cycles necessary to compute one iteration of the message digest has been reduced by a factor of k. At the same time, the critical path through k steps is likely to be significantly shorter than k times the path through a single step. This is because in hash functions, the critical path through a step of the message digest is different for each word of the step input (see Fig. 5ab).

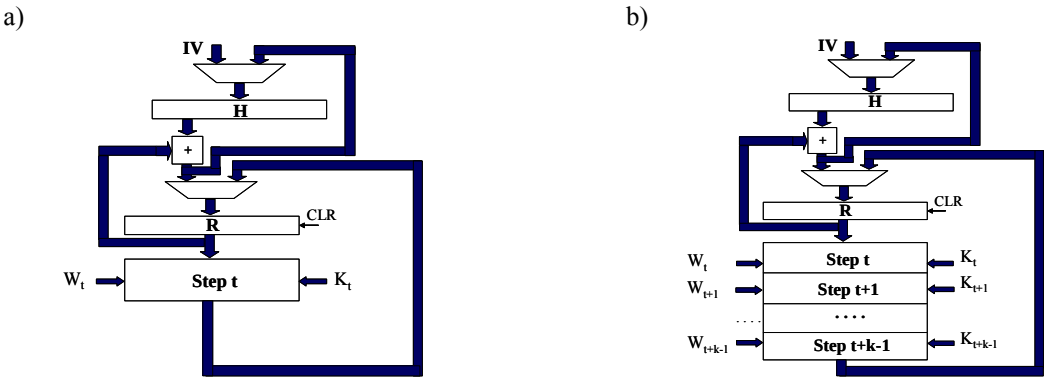


Fig. 2. General diagrams of the message digest units for a) basic architecture, b) partially unrolled architecture with k steps unrolled

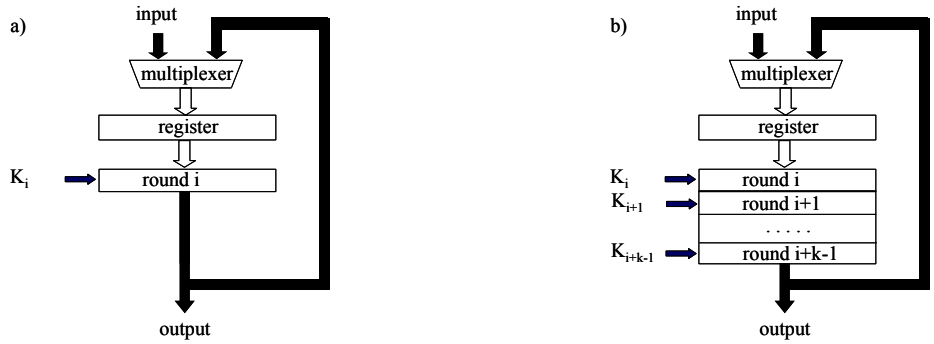


Fig. 3. Architectures of the encryption/decryption units of secret-key ciphers a) basic iterative architectures, b) partially unrolled architecture

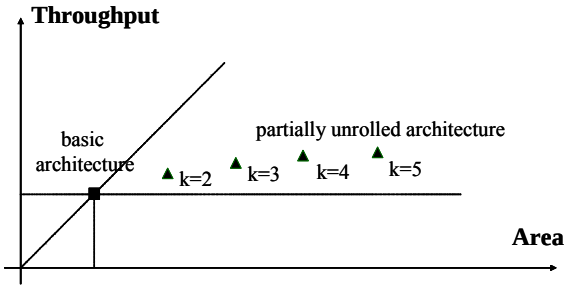


Fig. 4. Typical dependence between throughput and area of the partially unrolled architecture for secret-key block ciphers as a function of the number of rounds unrolled, k

Our research in this area has been inspired by a widely known partially unrolled architecture of secret-key block ciphers, shown in Fig. 3b [19, 20]. This architecture might be used as an alternative to the most popular basic iterative architecture of secret-key block ciphers shown in Fig. 3a. In the feedback block cipher modes, such as CBC and CFB, this architecture is the only architectural-level way of increasing encryption throughput of a single stream of data. Nevertheless, for majority of secret-key block ciphers with identical rounds, the throughput increases very slowly as a function of the number of unrolled rounds, k, while the circuit area grows almost proportionally to k. This dependence is shown in Fig. 4. We believe that this dependence does not apply to all dedicated hash functions because of the non-uniform way of processing data within message digest steps.

3. Previous work

Fully and partially unrolled architectures of dedicated hash functions have been investigated by several authors in the past, but no definite conclusions have been made. In [21] a fully unrolled architecture of MD5 has been compared with a basic iterative architecture. Unrolling of all 64 rounds resulted in a throughput increase by a factor of 2.1, while at the same time the circuit area increased by a factor of 5.4. In [22] a partially unrolled architecture of SHA-1, with the number of rounds unrolled k=5, has been investigated. A high level architecture presented in this paper was very similar to the one proposed in this paper. Nevertheless, the reported results were rather discouraging, with only 11% gain in the circuit throughput and a 43% penalty in the circuit area for the partially unrolled architecture over the basic iterative architecture.

All other hardware implementations of dedicated hash functions reported in the literature [23, 24] or available as commercial IP cores [10-16] have followed the basic iterative architecture with only one step of hash function executed in each clock cycle.

4. Details of the Hardware Architectures

4.1 Internal structure of the message digests of SHA-1 and SHA-512

Internal structures of the message digests for SHA-1 and SHA-512 are shown in Fig. 5ab. In both functions, input registers are initialized with the constant initialization vector, and are updated with the new value in each round. In SHA-1, four out of five words (A, B, C, and D) remain almost unchanged by a single round. These words are only shifted by one position down. The last word, E, undergoes a complicated transformation equivalent to multioperand addition modulo 2^{32}, with five 32-bit operands dependent on all input words, the round-dependent constant K_t, and the message dependent word W_t. The internal structure of the message digest of SHA-512 is similar. The primary differences are as follows: The number of words processed by each round is 8, each word is 64 bits long, and the longest path is equivalent to addition of seven 64-bit operands modulo 2^{64}. These operands depend on seven out of eight input words (all except D), the round-dependent constant K_t, and a message dependent word W_t. Six out of eight input words remain unchanged by a single round.

4.2 Basic architecture of SHA-1

From Fig. 5a, the critical path of a single SHA-1 round involves the calculation of the chaining variable A at the moment t+1, given by the following formula:

$$A_{t+1} = A_t<<<5 + f_t(B_t, C_t, D_t) + E_t + K_t + W_t + HA'_t$$

where X_t is a value of the variable X in the step t, and $HA'_t = HA$ when t=79, otherwise 0. HA is a word A of the register H in Fig. 2a.

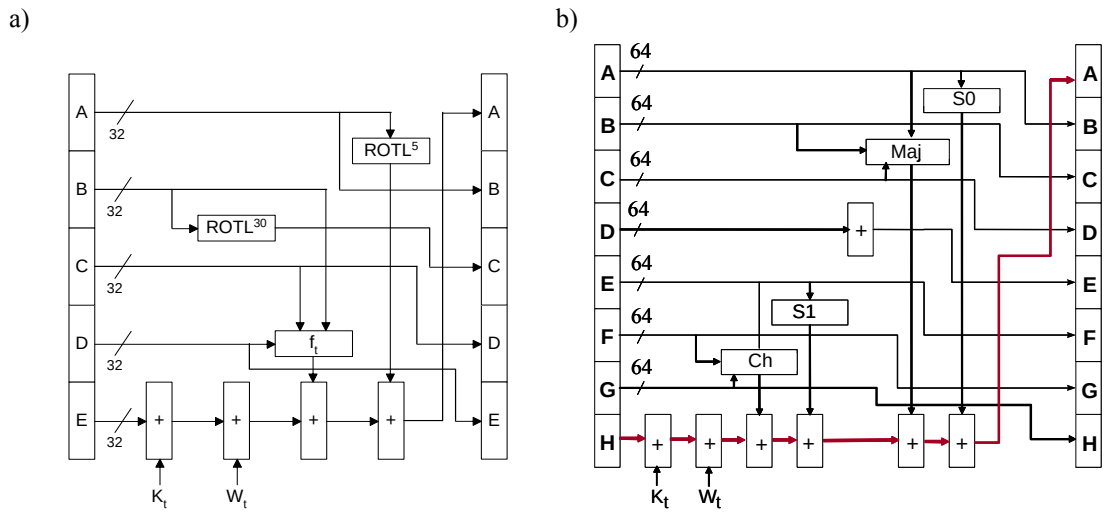


Fig. 5. Functional block diagrams of the message digest units
a) SHA-1 in the basic iterative architecture
b) SHA-512 in the basic iterative architecture

Additionally, we know that

$$B_t = A_{t-1}, \quad C_t = B_{t-1}<<<30, \quad D_t = C_{t-1}.$$

None of these operations involve any logic, consequently, the expression

$$f_t(B_t, C_t, D_t) = f_t(A_{t-1}, B_{t-1}<<<30, C_{t-1})$$

can be precomputed in the previous clock cycle, t-1, and will not contribute to the critical path. Similarly, the sum

$$\sum_{HA' K_t W_t} = K_t + W_t + HA'_t$$

can be precomputed by the message scheduler unit, because all values are known already in the previous clock cycle.
As a result, the critical path reduces to the addition of four operands

$$A_{t+1} = A_t<<<5 + E_t + \sum_{HA' K_t W_t} + f_t(A_{t-1}, B_{t-1}<<<30, C_{t-1}).$$

All aforementioned optimizations lead to the schematic of the basic architecture of SHA-1 shown in Fig. 6a. The lowest level multiplexers choose initialization vectors IV_0 to IV_4 only in the first clock cycle of computations for any new message. The variables HB'.. HE' are equal to HB..HE only in the last step of the message digest computations for a given message block, i.e., only when t=79; otherwise, they are equal to zero.

4.3 Partially unrolled architecture of SHA-1

The optimization of the unrolled message digest is straightforward. The general technique employed is to precalculate sums at the earliest possible stage, using either regular carry propagate adders (CPAs) or carry save adders (CSAs) (see Fig. 7). The calculations in the critical path follow a sequence of computations described by the equations below:

$$A_{t+1} = A_t<<<5 + f_t(B_t, C_t, D_t) + E_t + K_t + W_t = A_t<<<5 + f_t(B_t, C_t, D_t) + E_t + \sum K_t W_t$$

$$A_{t+2} = A_{t+1}<<<5 + f_{t+1}(B_{t+1}, C_{t+1}, D_{t+1}) + E_{t+1} + K_{t+1} + W_{t+1} =$$
$$= A_{t+1}<<<5 + [\, f_{t+1}(A_t, B_t<<<30, C_t) + D_t + \sum K_{t+1} W_{t+1} \,]$$

$$A_{t+3} = A_{t+2}<<<5 + [\, f_{t+2}(A_{t+1}, A_t<<<30, B_t<<<30) + [C_t + \sum K_{t+2} W_{t+2} \,]]$$

$$A_{t+4} = A_{t+3}<<<5 + [\, f_{t+3}(A_{t+2}, A_{t+1}<<<30, A_t<<<30) + [B_t<<<30 + \sum K_{t+3} W_{t+3} \,]]$$

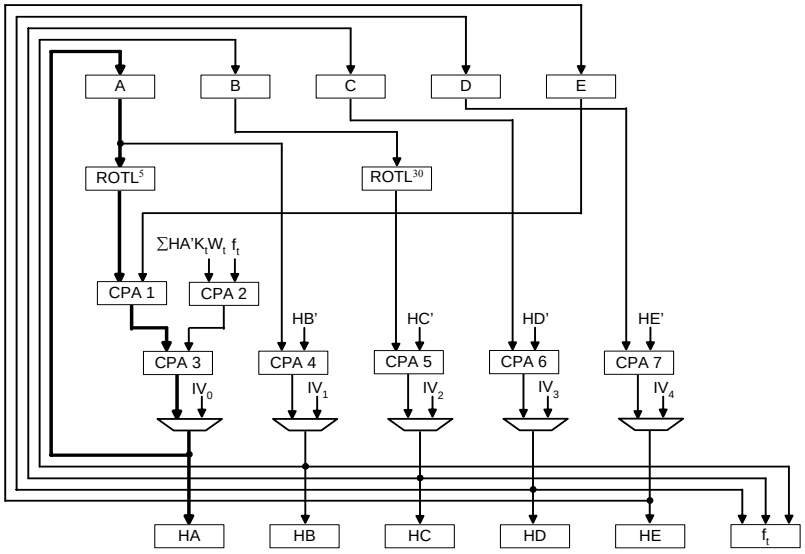
$$A_{t+5} = A_{t+4}<<<5 + [\, f_{t+4}(A_{t+3}, A_{t+2}<<<30, A_{t+1}<<<30) + [A_t<<<30 + \sum K_{t+4} W_{t+4} + HA'_{t+4}]]$$

At each stage two paths are critical. One is a calculation of the new value of A_{t+i} (i=1..5), which involves rotation by five positions and a single addition. The second is the precalculation of the value of $[f_{t+i} + [E_{t+i} + \sum K_{t+i} W_{t+i}]]$ to be used in the next stage. This precalculation involves the calculation of f_{t+i} and a single addition of a precalculated value $[E_{t+i} + \sum K_{t+i} W_{t+i}]$.

In the first stage of computations (computing A_{t+1}), precalculated values do not exist, so the computations must be performed from scratch. In every second stage starting from stage two, the precomputation of the sum $[f_{t+i} + [E_{t+i} + \sum K_{t+i} W_{t+i}]]$ is the most time consuming operation. Finally, in every second stage starting from stage three, the only contribution to the critical path is a single addition.

The total combinational delay in the critical path is 5 carry propagate adders, and 6 slices of logic. Compared to the implementation of the basic iterative architecture, this is an increase by 3 carry propagate adders and 5 slices of logic. On average this corresponds to less than carry propagate adder, and one slice of logic per message digest step. Furthermore, unrolling 5 times reduces the combinational path by four setup times and four delays of the registers, as well as it is likely to minimize the total sum of interconnect delays.

a)



b)

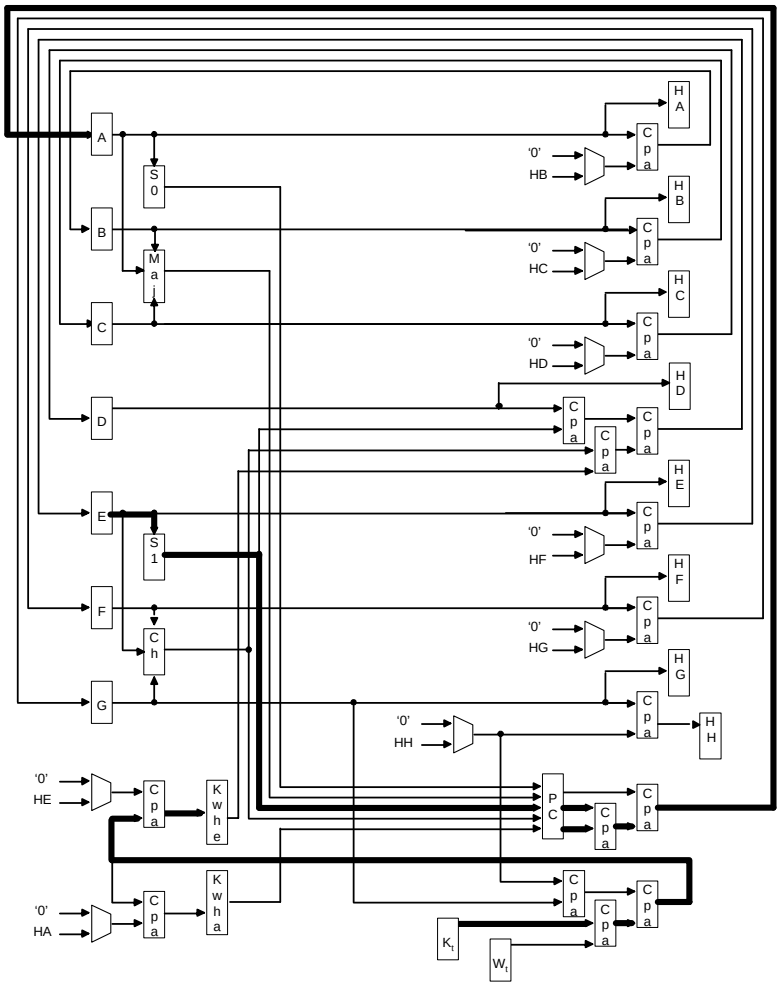


Fig. 6. Our implementations of the message digest units in the basic iterative architecture a) SHA-1, b) SHA-512

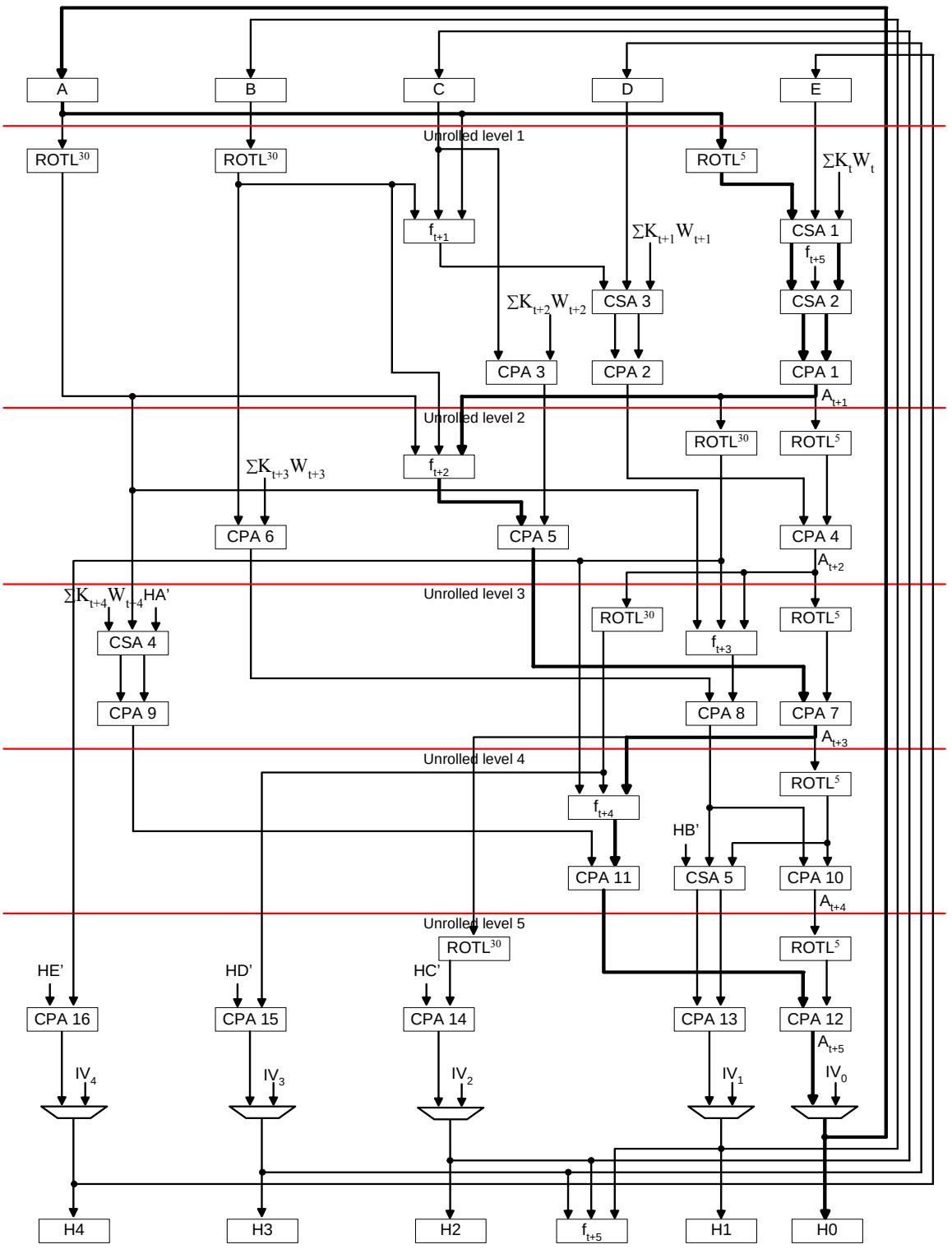


Fig. 7. Our implementation of the message digest unit of SHA-1 in the partially unrolled architecture with 5 steps unrolled

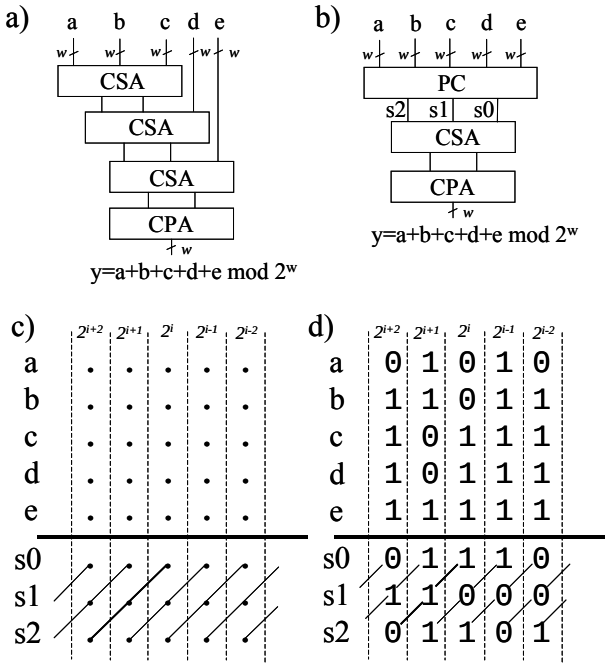


Fig. 8. Using a 5-to-3 Parallel Counter. a) adding five *w*-bit numbers using a tree of 3-to-2 carry save adders, b) adding five *w*-bit numbers using 5-to-3 parallel counter followed by a 3-to-2 carry save adder, c) operation of the 5-to-3 parallel counter in the dot notation, d) example of the operation of the 5-to-3 parallel counter

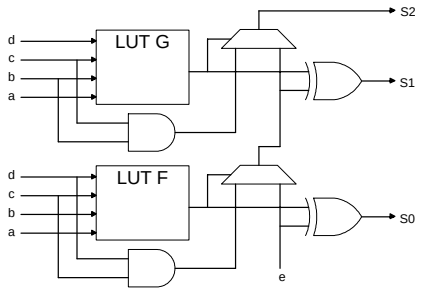


Fig. 9. Using internal structure of a single CLB slice of the Xilinx Virtex FPGA device to implement a bit-slice of a 5-to-3 Parallel Counter (PC)

4.4 Basic architecture of SHA-512

From Fig. 5b, the critical path of a single SHA-512 round involves the calculation of the chaining variable A at the moment t+1, given by the following formula:

$$A_{t+1} = S0(A_t) + Maj(A_t, B_t, C_t) + S1(E_t) + Ch(E_t, F_t, G_t) + K_t + W_t + H_t + HA_t'$$

where X_t is a value of the variable X in the step t; S0, Maj, S1, Ch are the logic functions defined in the SHA-512 standard, and $HA'_t = HA$ when t=79, otherwise 0.

Additionally, we know that

$$H_t = G_{t-1}.$$

The functions S0 and Maj execute in parallel in approximately the same amount of time. The same holds true for functions S1 and Ch.

The sum

$$KWHA_t = K_t + W_t + G_{t-1} + HA'_t$$

can be precomputed in the previous clock cycle, t-1.

As a result, the critical path reduces to the addition of five operands

$$A_{t+1} = S0(A_t) + Maj(A_t, B_t, C_t) + S1(E_t) + Ch(E_t, F_t, G_t) + KWHA_t.$$

The straightforward use of carry save adders in case of five operand addition would lead to three levels of 3-to-2 carry save adders, followed by a carry propagate adder as shown in Fig. 8a. Instead, we have decided to use a 5-to-3 parallel counter (see Fig. 8b) [25], which reduces the number of binary digits at each position in the sum of five operands from 5 to 3,

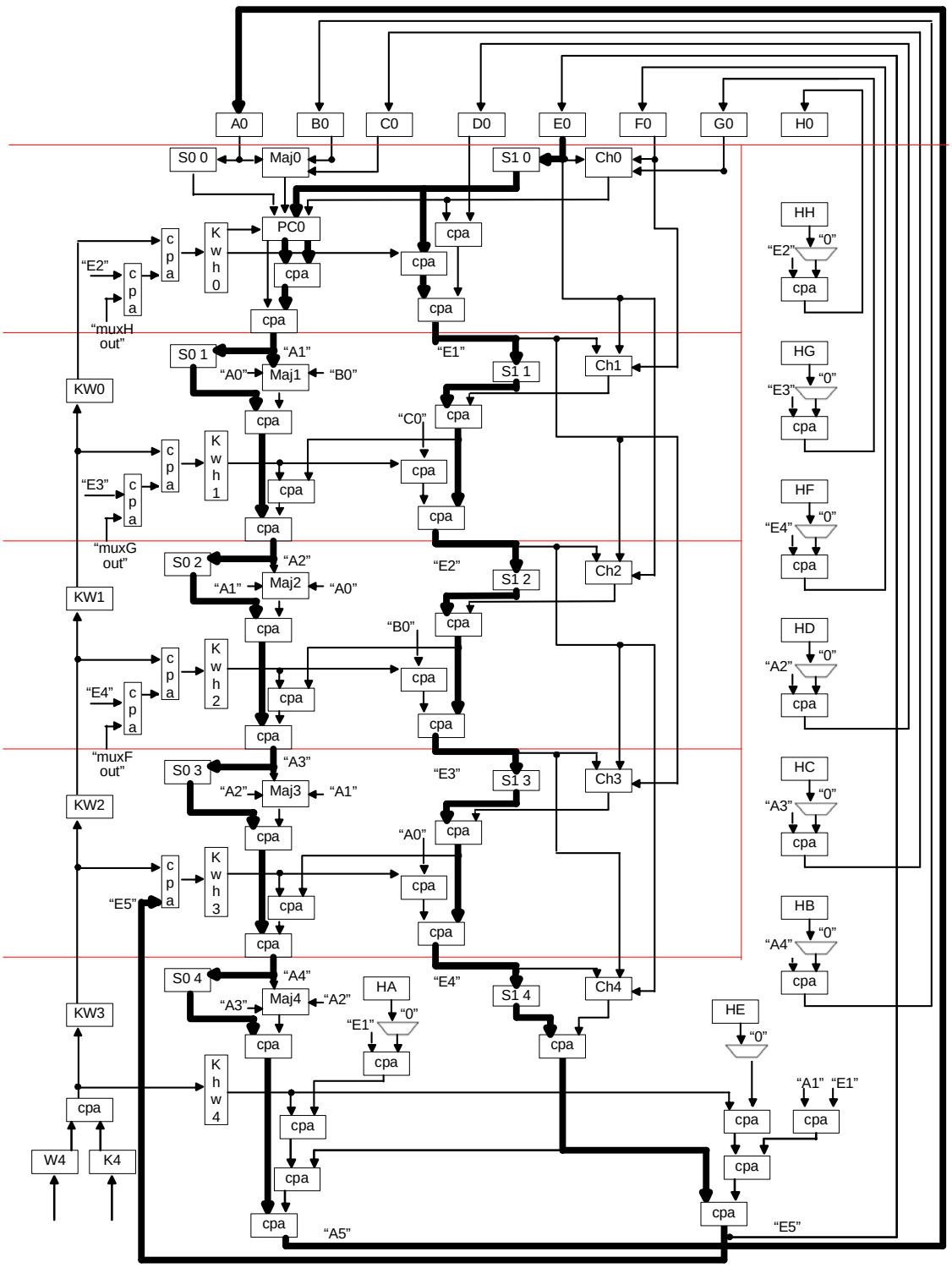


Fig. 10. Our implementation of the message digest unit of SHA-512 in the partially unrolled architecture with 5 steps unrolled

and has approximately the same delay as a 3-to-2 carry save adder. The operation of the 5-to-3 parallel counter is shown in Fig. 8c, using the dot notation. In this notation, each dot represents a binary digit, 0 or 1 [25]. The 5-to-3 parallel counter adds five binary digits with the same weight, 2^i, and represents the result using three binary digits with three subsequent weights, 2^i, 2^{i+1}, and 2^{i+2}. An example of the operation of this counter is shown in Fig. 8d. The speed-up comes from the fact that the operation of the parallel counter can be realized in Virtex FPGAs using resources of a single CLB slice as shown in Fig. 9.

All aforementioned optimizations lead to the schematic of the basic architecture of SHA-1 shown in Fig. 6b. The registers HA-HH are set to the initialization vectors IV_0 to IV_7 only in the first clock cycle of computations for any new message. The multiplexers selecting between HB and '0', HC and '0', etc. choose non-zero values only in the last step of the message digest computations for a given message block, i.e., only when t=79.

4.5 Unrolled architecture of SHA-512

The unrolled architecture of SHA-512 is shown in Fig.10. Because of the dependence of E_{t+1} on E_t, and A_{t+1} on A_t and E_t (see Figs. 5b), three major critical paths (A0 to A0, E0 to A0 and E0 to E0) exist in the circuit. These paths are marked in Fig. 10 with thicker lines. Values of variables A_{t+i}, and E_{t+i} are denoted as "Ai" and "Ei" respectively, e.g., "E2" denotes E_{t+2}. Precomputations in the previous clock cycle are used to reduce the number of operands in the first four stages of the unrolled architecture. Recall that in the basic architecture, the $KWHA_t$ sum is computed based on the equation $H_t = G_{t-1}$. In the unrolled architecture with k=5, t changes by 5 every clock cycle. As a result,

$$H_t = G_{t-1} = F_{t-2} = E_{t-3} = E_{t+2-5} = \text{"E2" in the previous clock cycle.}$$

On the far left side of Fig. 10, "E2" is used to precompute KWH0 (notation for $KWHA_{t+0}$) for the next clock cycle.

$$KWH0 = KWHA_t = K_t + W_t + H_t + HA'_t$$

This method is repeated in stages two to four in order to compute $KWHA_{t+i}$.(denoted in Fig. 10 as KWHi, i=1..3) In stage 5, $H_{t+4} = E_{t+1} =$ "E1" , so this value is computed in the same clock cycle, and as a result is not included in the earlier precomputed $KWH4 = KWHA_{t+4}$, which reduces to $KWHA_{t+4} = K_{t+4} + W_{t+4}$. Please, note that in Fig. 10, the sum $K_{t+i} + W_{t+i}$ is denoted as KWi.

Further reductions in critical paths were accomplished in each stage by adding values of logic functions S1 and Ch as early as possible, reusing values of S1 + Ch, and by selective routing to balance the number of slices in various critical paths. In Table 1, we give the lengths of each of the five most critical paths expressed in the number of CLB slices, as well as absolute delays of each path with the division between logic and routing. All of these paths were well-balanced in slice count, total propagation time, and logic to routing ratio. For comparison, the basic architecture required 4 levels of CLB slices, which would result in 20 levels of CLB slices if unrolled directly.

Table 1 Delays of the five most critical paths in the unrolled architecture of SHA-512

Output to Input	Slices in Path	Delay	Logic	Route	% Logic	% Route
E0 to A0	16	62.25	29.56	32.69	47.5%	52.5%
E0 to KWH3	16	61.86	30.22	31.63	48.9%	51.1%
F0 to A0	16	61.06	30.48	30.57	49.9%	50.1%
A0 to A0	16	60.83	30.17	30.66	49.6%	50.4%
E0 to E0	15	60.17	28.46	31.71	47.3%	52.7%

5. Design Methodology

Our target FPGA device was the Xilinx Virtex XCV-1000-6. This device is composed of 12,288 basic logic cells referred to as CLB (Configurable Logic Block) slices, includes 32 4-kbit blocks of synchronous dual-ported RAM, and can achieve synchronous system clock rates up to 200 MHz [9]. This device was chosen because of the availability of a general purpose PCI board, SLAAC-1V, based on three FPGA devices of this type [18]. Additionally, a new family of Virtex-E Xilinx devices was targeted as well.

All hardware architectures were first described in VHDL, and their operation verified through functional simulation using Active HDL, from Aldec, Inc. Test vectors and intermediate results from the reference software implementations based on the Crypto++ library [26] were used for debugging and verification of VHDL codes. The revised VHDL code became an input to logic synthesis performed using FPGA Compiler II from Synopsys. Tools from Xilinx ISE 4.2 were used for mapping, placing, and routing. These tools generated reports describing area and speed of implementation, a netlist used for timing simulation, and a bitstream used to configure an actual FPGA device. All designs were fully verified through behavioral, post-synthesis, and timing simulations, and experimentally tested using procedure described in Section 6.

6. Testing Procedure

The experimental testing of our cryptographic modules was performed using the SLAAC-1V hardware accelerator board, including three Virtex 1000 FPGAs as the primary processing elements [9]. Only one of the three FPGA devices was used to implement hash core.

Test program written in C used the SLAAC-1V APIs and the SLAAC-1V driver to communicate with the board. Our testing procedure is composed of three groups of tests. The first group verifies the circuit functionality at a single clock frequency. The goal of the second group is to determine the maximum clock frequency at which the circuit operates correctly. Finally, the purpose of the third group is to determine the limit on the maximum encryption and decryption throughput, taking into account the limitations of the PCI interface.

Our first group of tests is based on the NIST recommendations provided in [27]. These recommendations describe the comprehensive suite of three functional tests for SHA-1. The second test is aimed at determining the maximum clock frequency of the hash function modules. Three megabytes of pseudorandomly generated data are sent to the board for hashing, the result is transferred back to the host and compared with the corresponding output obtained using software implementation of the given hash function based on the Crypto++ library [26]. This procedure is repeated 30 times using the same clock frequency to minimize the effect of input data values on the results of analysis. The next clock frequency is chosen based on the rules of the binary search, i.e., in the middle between two closest earlier identified frequencies giving different test results. The test is repeated until the difference between these two frequencies is smaller than the required accuracy of the measurement (< 0.1 MHz in our tests). The highest investigated clock frequency at which no single processing error is detected is considered the maximum clock frequency. In our experiments, this test was automatically repeated 10 times with consistent results in all iterations.

The third group of tests is an extension of the second group. After determining the maximum clock frequency, we measure multiple times and average the amount of time necessary to process 3 MB of data, taking into account the delay contribution and the bandwidth limit of the 32 bit/33 MHz PCI interface. The experimentally confirmed limit of this interface was about 1 Gbit/s.

7. Results

In Fig. 11, the minimum clock periods of SHA-1 and SHA-512 obtained using static timing analysis and the experiment are given. For the unrolled architecture, the effective clock period is the minimum time necessary for the data signals to pass the critical path. Since in both our unrolled designs, the data signal is traveling through the critical path over multiple clock periods, the effective clock period is a multiple of the actual clock period. In case of the unrolled architecture for SHA-1 the multiplication factor is 2, in case of the SHA-512 architecture, the multiplication factor is 5.

Based on the knowledge of the minimum clock period, the maximum data throughput has been computed according to the equation:

$$Throughput = Message\_block\_size / (Effective\_clock\_period * Number\_of\_rounds/k)$$

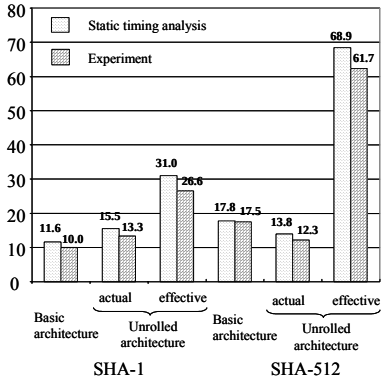


Fig. 11. Minimum clock periods of SHA-1 and SHA-512 in the basic iterative architecture and partially unrolled architecture obtained using static timing analysis and determined experimentally. For the unrolled architecture, an effective clock period is given describing a multicycle critical path through the message digest.

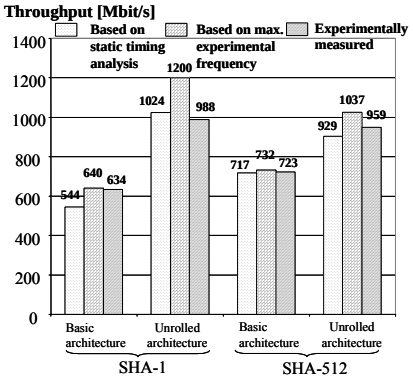


Fig. 12. Maximum throughputs of SHA-1 and SHA-512 in the basic iterative architecture and partially unrolled architecture: a) obtained using static timing analysis, b) calculated based on the maximum experimentally measured clock frequency, c) experimentally measured, including the contributions of the PCI interface.

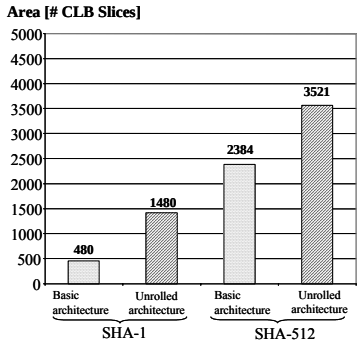


Fig. 13. Number of CLB slices used by the implementations of SHA-1 and SHA-512 in the basic iterative architecture and partially unrolled architecture

The maximum throughput values calculated based on the minimum clock periods obtained using static timing analysis and experiment are shown in Fig. 12. In the same figure, these results are compared with the experimentally measured data throughputs that take into account the delay contributions and the bandwidth limit of the PCI interface. This comparison demonstrates that the PCI interface is capable of operating with a constant uninterrupted data flow up to about 960-990 Mbit/s, and has a negligible influence on the data throughput below this communication rate.

The number of CLB Slices used by both implementations of SHA-1 and SHA-512 are shown in Fig. 13. The difference in the number of CLB slices is primarily caused by the difference in the size of input and output registers in the message digest units of both functions (512 bits vs. 160 bits), and the width of the multioperand adders in the critical path of these units (64 bits vs. 32 bits). In SHA-512, four 4 kbit block RAMs are used to store 80 64-bit constants K_t.

Out of the two analyzed hash standards, SHA-1 offers much better potential for loop unrolling. As a result of loop unrolling, the throughput of SHA-1 increased by a factor of almost two (1.9 times), while at the same time its area grew only by a factor of three. SHA-512 is much less suitable for loop unrolling, as its observed speed-up was only 30%, and the area increase 48%.

8. Comparison with other hash cores

There exist multiple commercial IP cores implementing SHA-1 [10-16]. In Table 2, we present the comparison of our designs for SHA-1 with the most representative IP cores with equivalent functionality. For the Xilinx Virtex family of FPGA devices, our core for SHA-1 in the basic iterative architecture outperforms the second best core (from Helion Technology Ltd) by 13%, using 30% less CLB slices. Our core for the partially unrolled architecture of SHA-1 with 5 rounds unrolled, outperforms all reported Virtex cores by a factor of at least two in terms of throughput, and uses about two times more area. The similar advantages exist for the implementations using Virtex-E devices, where our core for the unrolled architecture approaches the throughput of 1.2 Gbit/s.

At this point, there are relatively few cores available for the new standard, SHA-512 (see Table 3) [11, 12]. Our implementation of the basic iterative architecture slightly outperforms the equivalent core from ALMA Technologies in terms of throughput, using a smaller amount of FPGA resources. Our partially unrolled architecture is the fastest core for the Virtex family of FPGA devices outperforming the second best core by 30% at the cost of only 31% increase in the circuit area. For the Virtex-E family of FPGA devices our core is the only currently available SHA-512 core that exceeds the throughput of 1 Gbit/s.

9. Comparison with software implementations

Efficient software implementations of hash functions have been extensively studied in the literature [28-31]. In [28], basic recommendations on developing an efficient and portable implementation of SHA-1 in C have been formulated. In [29], a close to optimum implementations of dedicated hash functions using Pentium's superscalar architecture have been presented. In [30], software parallelism of all major dedicated hash functions have been studied. Finally, in [31], optimizations targeting Pentium III have been investigated. These optimizations made use of MMX registers and instructions available in Pentium III.

In this paper, we used for comparison, software implementations of SHA-1 and SHA-512, available as a part of the Crypto++ library [26]. Although Crypto++ is not the fastest of the reported software implementations, the reason for using this library was its portability, availability in public domain, and wide practical deployment.

A PC with 2.2 GHz clock, 1 GByte RAM, and cache size 512KB, running Windows XP was used in our measurements. The Crypto++ implementation of hash functions written in C++ was compiled using MS Visual Studio with Service Pack 5.

The obtained throughput was 40.5 Mbit/s for SHA-1 and 30.4 Mbit/s for SHA-512. These throughputs were respectively 25 times and 31 times smaller than the throughputs of our partially unrolled hardware implementations of SHA-1 and SHA-512 for Xilinx Virtex 1000-6 FPGAs.

Table 2. Comparison of our designs for SHA-1 with the representative commercial IP cores with equivalent functionality

Source	Clock frequency [MHz]	Throughput [Mbit/s]	Area [CLB Slices]
Xilinx Virtex			
Our, basic	**85**	**544**	**480**
Our, unrolled (k=5)	**64**[1]	**1024**	**1480**
ALMA Technologies	70	442	686
Helion Technology Limited	76	480	689
Ocean Logic Pty Ltd	56	352	612
Xilinx Virtex-E			
Our, basic	**103**	**659**	**484**
Our, unrolled (k=5)	**72.5**	**1160**	**1484**
ALMA Technologies	87	549	686
Bisquare Systems Private Limited	66	422	579
Helion Technology Limited	95	600	689
Intron, Ltd.	71	449	716
Ocean Logic Pty Ltd	71.5	452	612
Xilinx Virtex-II			
ALMA Technologies	102	644	686
Amphion Semiconductor	99	626	854
Helion Technology Limited	103.5	654	569
Ocean Logic Pty Ltd	79	498	612

[1] multi-cycle clock used in the critical path, critical path $\leq 2\ T_{CLK} = 2/f_{CLK}$, 5 steps executed in 2 clock cycles

Table 3. Comparison of our designs for SHA-512 with the representative commercial IP cores with equivalent functionality

Source	Clock frequency [MHz]	Throughput [Mbit/s]	Area
Xilinx Virtex			
Our, basic	**56**	**717**	**2384 Slices 4 Block RAMs**
Our, unrolled (k=5)	**67**[1]	**929**	**3521 Slices 4 Block RAMs**
ALMA Technologies	56	707	2690 Slices 4 Block RAMs
Xilinx Virtex-E			
Our, unrolled (k=5)	**72**[1]	**1034**	**3517 Slices 4 Block RAMs**
ALMA Technologies	68	859	2690 Slices 4 Block RAMs
Xilinx Virtex-II			
ALMA Technologies	72	910	2507 Slices 4 Block RAMs
Amphion Semiconductor	50	626	2403 Slices 4 Block RAMs

[1] multi-cycle clock used in the critical path, critical path $\leq 5\ T_{CLK} = 5/f_{CLK}$, 5 steps executed in 5 clock cycles

10. Summary

A new *partially unrolled* architecture has been proposed for a family of dedicated hash functions, including four American standard algorithms SHA-1, SHA-256, SHA-384, and SHA-512. This architecture has been inspired by a similar architecture used in implementations of secret-key block ciphers. The unrolled architecture has been designed, optimized, and experimentally verified for the most widely used hash algorithm, SHA-1, and one of the new hash standard algorithms SHA-512. For the purpose of comparison, the basic iterative architecture has been implemented for both functions as well.

The new architecture appeared to be particularly suitable for the implementation of SHA-1. For the number of rounds unrolled equal to k=5, it allowed to almost double the throughput of SHA-1 compared to the basic iterative architecture, at the cost of increasing circuit area by a factor of three. The similar design for SHA-512 appeared to have much less benefit; the increase in the circuit throughput was only 30%, and the area of the circuit increased by 48%.

This different behavior of two hash algorithms could be easily explained by analyzing the structure of both algorithms. In the unrolled architecture of SHA-1, many message digest steps could be substantially sped up by preprocessing partial results of a given step in the previous steps. The same optimization was not possible in SHA-512 due to sequential dependencies present in the algorithm.

Our partially unrolled implementation of SHA-1 reached the target throughput of 1 Gbit/s in Virtex XCV 1000, and outperformed all known to the authors commercial IP cores with equivalent functionality by at least a factor of two. Our implementation of SHA-512 also compared favorably with commercial IP cores, and reached a target throughput of 1 Gbit/s using Virtex-E family of Xilinx FPGAs. To our best knowledge, our implementations of SHA-1 and SHA-512 are the only FPGA implementations of these hash functions available to date that can sustain a throughput over 1 Gbit/s for a single stream of data.

References

1. Stallings, W.: Cryptography and Network Security, 1999 Prentice-Hall, Inc., Upper Saddle River, New Jersey. 2nd Edition
2. Menezes, A. J., van Oorschot P. C., and Vanstone S. A.: Handbook of Applied Cryptography, CRC Press, Inc., Boca Raton, 1996
3. FIPS 198, HMAC - Keyed-Hash Message Authentication Code, available at http://csrc.nist.gov/encryption/tkmac.html
4. NIST Cryptographic Toolkit, Secure Hashing, available at http://csrc.nist.gov/encryption/tkhash.html
5. FIPS 180-2, Secure Hash Standard (SHS), August 2002, available at http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
6. FIPS 185, Escrowed Encryption Standard (EES), February 1994.
7. FIPS 186-2, Digital Signature Standard (DSS), February 2000, available at http://csrc.nist.gov/encryption/tkdigsigs.html
8. NIST, FIPS Publication 197, Specification for the Advanced Encryption Standard (AES), November 26, 2001, available at http://csrc.nist.gov/encryption/aes/
9. Xilinx, Inc.: Virtex 2.5 V Field Programmable Gate Arrays, available at www.xilinx.com
10. SHS Validation List, available at http://csrc.nist.gov/cryptval/shs/shaval.htm
11. ALMA Technologies web page, available at http://www.alma-tech.com
12. Amphion Semiconductor web page, available at http://www.amphion.com
13. Bisquare Systems Private Limited web page, available at http://www.bisquare.com
14. Helion Technology Limited web page, available at http://www.heliontech.com
15. Intron, Ltd. Web page, available at http://www.intron.lviv.ua
16. Ocean Logic Pty Ltd web page, available at http://www.ocean-logic.com
17. IP Security Protocol (ipsec) Charter - Latest RFCs and Internet Drafts for IPSec, http://ietf.org/html.charters/ipsec-charter.html
18. GRIP (Gigabit Rate IP Security) project page, available at http://www.east.isi.edu/projects/GRIP/
19. Elbirt, A. J., Yip, W., Chetwynd, B., Paar, C.: An FPGA implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. Proc. 3rd Advanced Encryption Standard (AES) Candidate Conference, New York, April 13-14, 2000
20. Gaj, K., and Chodowiec, P.: Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays, Proc. RSA Security Conference - Cryptographer's Track, April 2001
21. Deepakumara J., Heys H.M., and Venkatesan R.: FPGA Implementation of MD5 Hash Algorithm, Proc. IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2001), Toronto, Ontario, May 2001, available at http://www.engr.mun.ca/~howard/PAPERS/ccece_2001.pdf.
22. Hoare R., Menon P., and Ramos M..: 427 Mbits/sec Hardware Implementation of the SHA-1 Algorithm in an FPGA, International Association of Science and Technology for Development (IASTED) Journal 2002, 381-142.

23. Ting K.K., Yuen S.C.L., Lee K.H., and Leong P.H.W.: An FPGA Based SHA-256 Processor, Proc. 12th International Conference, FPL 2002, Montpellier, France September 2-4, 2002.
24. Kang K.Y., Kim D.W., Kwon T.W., and Choi J.R.: Hash Function Processor Using Resource Sharing for IPSec, Proc. 2002 International Technical Conference On Circuit/Systems, Computers and Communications, available at http://www.kmutt.ac.th/itc2002/CD/pdf/18_07_45/TA2_OE/5.pdf
25. Parhami, B.: Computer Arithmetic: Algorithms and Hardware Design, Oxford University Press, 2000
26. Crypto++, free C++ class library of cryptographic schemes, available at http://www.eskimo.com/~weidai/cryptlib.html
27. Digital Signature Standard Validation System (DSSVS) User's Guide available at http://csrc.nist.gov/cryptval/shs.html
28. McCurley K.S.: A Fast Portable Implementation of the Secure Hash Algorithm, Sandia National Laboratories Technical Report SAND93-2591.
29. Bosselaers A., Govaerts R. and Vandewalle J.: Fast Hashing on the Pentium, in N. Koblitz (Ed.): Advances in Cryptology - CRYPT0 '96, LNCS 1109, Springer-Verlag Berlin Heidelberg 1996, 298-312.
30. Bosselaers A., Govaerts R. and Vandewalle J.: SHA: A Design for Parallel Architectures?, in W. Fumy (Ed.): Advances in Cryptology - EUROCRYPT '97, LNCS 1233, Spnnger-Verlag Berlin Heidelberg 1997, 348-362.
31. Nakajima J. and Matsui M.: Performance Analysis and Parallel Implementation of Dedicated Hash Functions, in L.R. Knudsen (Ed.): EUROCRYPT 2002, LNCS 2332, Springer- Berlin Heidelberg 2002, 165–180.