An FPGA based SHA-256 Processor

Kurt K. Ting, Steve C.L. Yuen, K. H. Lee, and Philip H.W. Leong

Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
New Territories, Hong Kong
{kting,clyuen,khlee,phwl}@cse.cuhk.edu.hk

Abstract. The design, implementation and system level performance of an efficient yet compact field programmable gate array (FPGA) based Secure Hash Algorithm 256 (SHA-256) processor is presented. On a Xilinx Virtex XCV300E-8 FPGA, the SHA-256 processor utilizes 1261 slices and has a throughput of 87 MB/s at 88 MHz. When measured on actual hardware operating at 66 MHz, it had a maximum measured system throughput of 53 MB/s.

1 Introduction

Field programmable gate array (FPGA) devices provide an excellent technology for the implementation of general purpose cryptographic devices. Compared with application specific integrated circuits (ASIC), FPGAs offer lower non-recurring engineering costs, shorter design time, greater flexibility and the ability to change the algorithm or design in the field. They have been used in a number of high performance cryptosystems including RSA [12], DES [13], Rijndael (AES) [6] and IDEA [10]. FPGA implementations of cryptographic algorithms have applications as coprocessors for microprocessor based systems or in high performance embedded applications.

The Secure Hash Signature Standard (SHS) was proposed by the US National Institute of Standards and Technology (NIST) in 2001 [9]. The standard describes four secure hash algorithms (SHA) and the version which outputs a 256-bit message digest is referred to as SHA-256. In this paper, only SHA-256 will be considered, although adapting the design to other digest sizes should be trivial.

Applications of the SHS include generating and verifying digital signatures, generating and verifying message authentication codes and also increasing the entropy in pseudo random number generators.

In this paper, a novel architecture for the implementation of the SHA-256 hash algorithm is presented. Making extensive use of shift registers, the design is compact yet achieves high performance. The system level performance of the SHA-256 core was tested on the Pilchard reconfigurable computing platform [11].

Although FPGA based processors for the MD5 hash algorithm have been proposed [4, 5], we are not aware of any published designs for SHA-256 processors. A NIST validated commercial SHA-1 and MD5 core is available from Tality

Corporation [1] which operates at 75MHz in $0.25\mu m$ technology and achieves 59 MB/s throughput.

SecuCore [2] is a commercial SHA-256 IPcore, featuring a maximum frequency of 166 MHz in a $0.18\mu m$ process with a throughput of 156 MB/s. Both the SecuCore processor and our FPGA-based SHA-256 processor use $0.18\mu m$ technology. The two designs require a similar number of clock cycles so the higher performance of the SecuCore implementation is due to the higher clock rate (166 MHz vs 88 MHz). One would expect an ASIC implementation to have a higher clock rate than an FPGA due its customized logic and routing.

The rest of the paper is organized as follows: in Section 2, the SHA-256 algorithm is described. Section 3 describes the architecture of the processor. Performance measurements are presented in Section 4, future work is described in Section 5, and conclusions are drawn in Section 6.

2 SHA-256 algorithm

The SHA-256 algorithm takes a message of length less than 2^{64}-bits and produces as output, a message digest 256-bits in length. The digest serves as a concise representation of the message, and has the property that any change to the message is very likely to result in a change to the corresponding digest. The SHA-256 algorithm has a security of 128-bits, meaning that a birthday attack [7] can produce a collision in $O(2^{128})$ time.

In the SHA-256 algorithm, six logical functions which operate on 32-bit values are used:

$$
\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\sim x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
\Sigma_0(x) &= \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x) \\
\Sigma_1(x) &= \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x) \\
\sigma_0(x) &= \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) \\
\sigma_1(x) &= \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x)
\end{aligned}
$$

where $\wedge$, $\sim$ and $\oplus$ are the bitwise AND, NOT and XOR operations; and ROTR and SHR are the rotate right and shift right functions respectively.

In order to hash a message M of l bits, a preprocessing step is first performed:

1. A “one” bit is appended to the end of the message, followed by k “zero” bits where k is the smallest non-negative solution to the equation $l + 1 + k \equiv 448 \bmod 512$. The binary representation of l as a 64-bit number is then appended so the length of the padded message is a multiple of 512-bits.
2. The padded message is then divided into N 512-bit blocks $M^{(1)}, M^{(2)}, \ldots M^{(N)}$.
3. The initial values of eight 32-bit words $H_j^{(0)}$ $(j = 0, 1, \ldots 7)$ are initialized to the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers.

The message blocks are then processed as follows for i = 1 to N:

1. The message schedule W_t $(t = 0 \ldots 63)$ is prepared according to the equation

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & 16 \leq t \leq 63 \end{cases} \quad (1)$$

2. Eight 32-bit working variables a, b, c, d, e, f, g, h are initialized to $H_0^{(i-1)}$, $H_1^{(i-1)}$, $H_2^{(i-1)}$, $H_3^{(i-1)}$, $H_4^{(i-1)}$, $H_5^{(i-1)}$, $H_6^{(i-1)}$, $H_7^{(i-1)}$ respectively.
3. The compression function is performed for t = 0 to 63:

$$T_1 = h + \Sigma_1(e) + Ch(e,f,g) + K_t + W_t; T_2 = \Sigma_0(a) + Maj(a,b,c)$$
$$h = g; g = f; f = e; e = d + T_1; d = c; c = b; b = a; a = T_1 + T_2$$

4. The intermediate hash $H^{(i)}$ is computed:

$$H_0^{(i)} = a + H_0^{(i-1)}; H_1^{(i)} = b + H_1^{(i-1)}; H_2^{(i)} = c + H_2^{(i-1)}; H_3^{(i)} = d + H_3^{(i-1)};$$
$$H_4^{(i)} = e + H_4^{(i-1)}; H_5^{(i)} = f + H_5^{(i-1)}; H_6^{(i)} = g + H_6^{(i-1)}; H_7^{(i)} = h + H_7^{(i-1)}$$

The final digest is formed by concatenating the final hash values

$$H_0^{(N)}, H_1^{(N)}, H_2^{(N)}, H_3^{(N)}, H_4^{(N)}, H_5^{(N)}, H_6^{(N)}, H_7^{(N)}$$

3 System Architecture

A shift register based approach was used to implement the SHA-256 algorithm which results in a fast and compact design. This architecture was inspired by NIST's descriptions of secure hash algorithms [8]. By inspecting the algorithm description in Section 2, it can be seen that the message schedule and compression function map naturally to a shift register structure.

The core contains three main components which implement the message scheduler, compression function and intermediate hash. These are controlled by a finite state machine which schedules the three blocks.

3.1 Message Scheduler

The message scheduler is implemented as a chain of sixteen 32-bit shift registers which store the intermediate message schedules W_t. Figure 1 shows the hardware architecture used to implement equation 1. It uses 16 cycles to load sixteen initial 32-bit words, ${M_t}^{(i)}$ for $t = 0$ to 15. During the 64 iterations of $t = 0$ to 63, it provides the message schedule W_t for the compression function by shifting the values in the chain from left to right. In the hardware implementation, W_t is added to the constant K_t to form W_K_t before being sent to the compression function. The rationale for moving the addition of K_t from the compression function to the message scheduler was to reduce the critical path of the compression function. This scheme results in a speedup of approximately 20%.

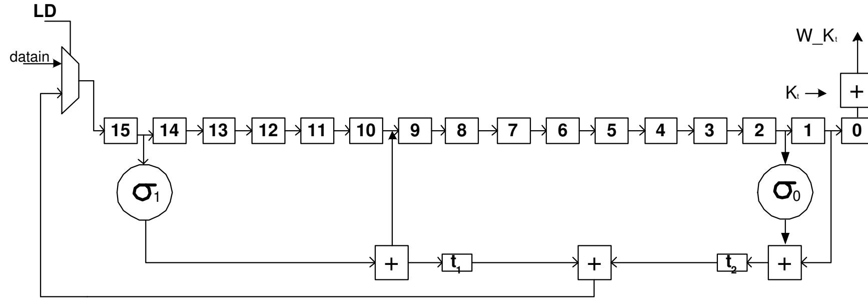

Fig. 1. Message scheduler block diagram.

3.2 Compression Function

The compression function module was implemented using shift registers, in a manner similar to the message scheduler. The 8 working variables $a, b, \ldots h$ are stored in eight 32-bit shift registers and connected according to Figure 2. The critical path in the design is the computation of $a = \sum_0 + Maj(a, b, c) + \sum_1 + Ch(e, f, g) + h + W + K_t$. The path was therefore pipelined by inserting a latch between $\sum_1 + Ch(e, f, g) + h + W_K_t$ and $a = \sum_0 + Maj(a, b, c)$ (shown as "L" in Figure 2, with functions before "L" taking inputs earlier along the chain).

By-pass logic was also added between registers d and e to allow the outputs of the compression function, $a, b, \ldots h$ to be shifted out through h. The loading of values into the message scheduler is fully overlapped with the operations of the compression function, a new message block being loaded when the previous message block is in the 48th round of the compression function.

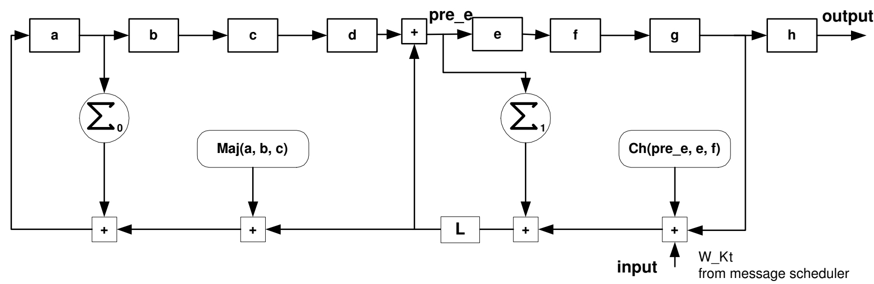

Fig. 2. Compression function block diagram.

3.3 Intermediate Hash

After 64 iterations of compression function, eight 32-bit intermediate values stored in the working variables $a, b, \ldots h$ are obtained. To compute the intermediate hash $H^{(i)}$, the working variables are added to the previous intermediate hash $H^{(i-1)}$ and written back to the registers. In the hardware implementation, $H^{(i-1)}$ is stored in another 256-bit latch and updated before the 64 iterations of the compression function begins. This is illustrated in Figure 3. The path in dashed lines is used for updating $H^{(i-1)}$ and the path in solid lines calculates the current intermediate hash $H^{(i)}$.

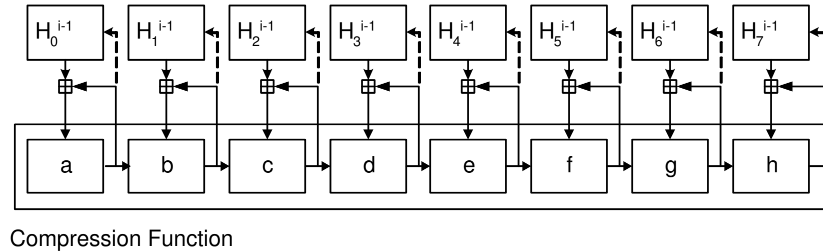

Fig. 3. Intermediate Hash from Compression Function.

3.4 PC Interface

The FPGA platform used was a Pilchard FPGA card (Figure 4) [11] populated with a Xilinx Virtex XCV300E-8 FPGA. Pilchard uses a SDRAM memory bus interface instead of the conventional PCI bus and has much improved latency and bandwidth over the standard PCI bus.

The Pilchard platform provides a 64-bit wide memory mapped bus to the FPGA. In the current configuration, PC reads and writes operate at 133 MHz, which is the clock speed of the memory bus. The SHA core operates at a lower clock rate (66 MHz). In order to interface the two, on-chip dual port Block RAM was used. As shown in Figure 5, to compute a digest, the PC writes 64-bit data at 133 MHz to the input Block RAM. The SHA core reads 32-bit data at 66 MHz from the other port of the Block RAM and writes the resulting digest value to the 32-bit output Block RAM.

Polling was used to ensure reliable communications between the host PC and the SHA core. The PC signals the SHA core to start after it has filled the input buffer and polls the core until it signals that it has finished. It then fills the buffer again.

Fig. 4. Photograph of the Pilchard board.

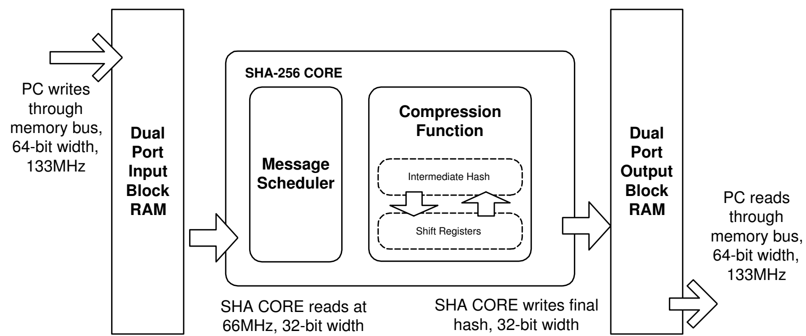

Fig. 5. PC interface block diagram.

4 Results

The design was synthesized and implemented using the Xilinx ISE 4 tools and tested on the Pilchard platform. On a Xilinx Virtex XCV300E-8 FPGA, the SHA-256 core has a maximum frequency of 88 MHz (as reported by the Xilinx tools). Each 512-bit message block requires 8 cycles to load and 65 cycles to process. This translates to a maximum throughput of $\frac{512}{8 \times 65} \times 88 \times 10^6 = 87$ MB/s for an 88 MHz clock. In the actual implementation on Pilchard, the system clock was 133 MHz and the SHA-256 core was operated using a half rate 66 MHz clock. This configuration has a maximum throughput of 65 MB/s.

4.1 Resource Usage

A summary of the resource utilization of the SHA-256 implementation (including interface logic) in shown below. According to the Xilinx tools, the design (including host interface) uses 1,261 Virtex slices and has an equivalent gate count of 167,190 gates.

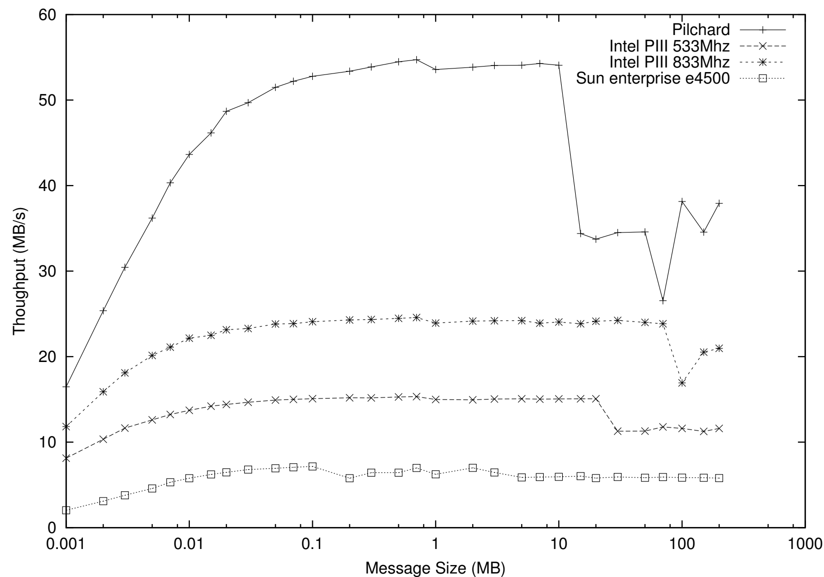

Fig. 6. Measured throughput of the Pilchard and software-only implementations as a function of the file size.

The SHA-256 processor was tested on a Pilchard card hosted on a 533 MHz Intel Pentium III machine with 128MB RAM. Files containing randomly generated numbers with sizes between 1K and 200 MB were tested and the results

verified with the mhash software library [3]. For each different input file size, the test was repeated 20 times and averaged.

The top trace of Figure 6 shows the measured system throughput in MB/s verses the input file size on a log scale. The results include all file I/O and operating system overheads as they are the times computed for computing a digest of an actual file. The throughput quickly saturates to a maximum value of 53 MB/s for file sizes greater than 200 KB. For the "uncacheable" memory type range register (MTRR) that was used, Pilchard is capable of a throughput of 132 MB/s, and the SHA-256 core 65 MB/s, thus the overall system throughput was limited by the handshaking overhead associated with the SHA-256 core.

For file sizes larger than 30 MB, throughput drops to approximately 30MB/s. A possible explanation for this strange phenomenon is that performance is greater for the smaller files due to the operating system caching the file reads.

The mhash optimized software implementation of SHA-256 [3] was also used for performance comparison purposes. Throughput measurements on Intel Pentium and Sun Enterprise machines are shown in the bottom traces of Figure 6. It is interesting to note a similar drop in throughput associated with large input file sizes in the software implementation. The hardware performance is more than double that of the 833 MHz Pentium III software implementation for file sizes up to 20 MB, and for larger file sizes, the FPGA version is about 50% faster.

The FPGA design has approximately a two times speed improvement over an 833 MHz Pentium III processor. However, this was achieved using a single chip compared with a workstation. FPGA based implementations have advantages in terms of cost, memory, energy and size over a software implementation, which may be important considerations in embedded applications.

5 Future Work

We are working on several improvements to the present design which could serve to improve the performance and flexibility of the SHA-256 system.

- Padding of the message is currently performed in software. In some applications it might be necessary to perform the message padding in hardware.
- Multiple SHA-256 processors could be instantiated in the FPGA as each only uses 40% of the resources. This would allow multiple files to be hashed simultaneously.
- With better floorplanning, routing delays could be significantly reduced and the performance of our design could be significantly improved.
- Larger buffers between the host and the SHA-256 core would serve to reduce the overheads associated with handshaking, increasing system throughput.
- It is possible to reduce the number of cycles, hence improving both latency and throughput via a parallel load facility so that several shift registers can be loaded in a single cycle.

6 Conclusion

In this paper, a shift register based architecture for implementing a SHA-256 processor was presented. This approach combines modest hardware requirements with high performance (87 MB/s). Detailed measurements of system level performance were reported, the system being about to compute SHA-256 digests with a throughput of 53 MB/s, the performance being limited by handshaking overheads. Suggestions for further improving the throughput of the system were given.

References

1. *www.tality.com.*
2. *http://www.secucore.com/products.htm.*
3. *http://mhash.sourceforge.net/.*
4. J. Arnold. Mapping the MD5 hash algorithm onto the NAPA architecture. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 267–268, 1998.
5. J. Deepakumara, H.M. Heys, and R. Venkatesan. FPGA implementation of MD5 hash algorithm. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 919–924, 2001.
6. M. McLoone and J. McCanny. High performance single-chip FPGA Rijndael algorithm implementations. In *Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES)*, pages 65–76. LNCS 2162, Springer, 2001.
7. A. Menezes, P. van Oorschoot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
8. NIST. Descriptions of sha-256, sha-384, and sha-512. *available from http://csrc.nist.gov/encryption/shs/sha256-384-512.pdf.*
9. NIST. *Secure Hash Signature Standard (FIPS PUB 180-2).* 2001.
10. O.Y.H. Cheung, K.H. Tsoi, K.H. Leung, P.H.W. Leong, and M.P. Leong. Tradeoffs in parallel and serial implementations of the international data encryption algorithm IDEA. In *Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES)*, pages 333–347. LNCS 2162, Springer, 2001.
11. P.H.W. Leong, M.P. Leong, O.Y.H. Cheung, T. Tung, C.M. Kwok, M.Y. Wong, and K.H. Lee. Pilchard – a reconfigurable computing platform with memory slot interface. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) – to appear*, 2001.
12. M. Shand and J. E. Vuillemin. Fast implementations of RSA cryptography. In E. E. Swartzlander, M. J. Irwin, and J. Jullien, editors, *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 252–259. IEEE Computer Society Press, Los Alamitos, CA, 1993.
13. S. Trimberger, R. Pang, and A. Singh. A 12Gbps DES Encryptor/Decryptor core in an FPGA. In *Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES)*, pages 156–163. Springer, 2000.