Literature Survey of SHA-256 Hardware implementations for ECSE 682 project

Shabbir Hussain
Dept. of Computer Engieering, McGill University
Montreal, Quebec
Email: shabbir.hussain@mail.mcgill.ca

***Abstract*—SHA-256 is a cryptographic one way hashing algorithm used to verify the authenticity of files and messages. It is a NIST standard used in many protocols such as SSH, PGP, IPSec and HMAC. Hardware implementations of the SHA-256 algorithm have been proposed in the literature to reduce the power and performance footprint of cryptographic calculations on General Purpose Processors (GPP). These implementations use techniques such as Loop unrolling, pipelining, retiming to speed up hashing and can achieve performance boosts up to 150x, with throughput of 11GBps and power as low as 46mW.**

I. Introduction

Cryptography is often known as a practice to send secret messages. While encrypting and decrypting messages are an important part of keeping secrets, so too is the method of verifying that message content is unaltered from sender to receiver. One of the methods employed to make sure that a message remains unmodified in transit is called a Message Authentication Code (MAC). These codes are computed based on the message contents and a shared secret key between sender and receiver. A MAC provides message integrity because an imposter can not feasibly create a false message and MAC pair or modify the contents of an existing message that will trick the receiver into believing that the messages were authored by the true sender [1]. This is because the generation and verification of a MAC involves using a cryptographically secure hash function such as the SHA-256 algorithm. The SHA-256 algorithm is a NIST standard that is part of a family hash functions called the SHA2 family. These functions take as input an arbitrary length string of bits and compute fixed length string of bits called the Digest Message (DM). These functions are one way functions with a low chance of collisions. Changing 1 bit of the input dramatically changes the output making it very difficult to guess the input based on the output. This property is what gives it its relevance in the field of cryptography. The only way to compute the input a DM is to guess and as of writing this paper there no known attacks on the SHA-256 to speed up this process. Given a 256 bit DM it would take a modern computer several years to find the corresponding input. This is because cryptographic functions are slow in general.
Computing a MAC for every message adds a performance and power penalty. IPSecs performance bottleneck is the HMAC mechanism which is responsible for authenticating the transmitted data [2]. VLSI is literature is rich in solutions that offload the has computation to hardware to quickly compute hashes with a low power and performance foot print. This paper surveys three recent designs that implement the SHA-256 algorithm in hardware.

The paper will be divided as follows. Section V will present the sha256 algorithm. Section III will discuss the design by Sklavos et al. Section IV will explain the design by Chaves et al. Section VI will explain the design by Michail et al. Finally, section VII will have concluding thoughts about all three implementations and an outlook into the future work that can be done.

II. SHA-256 algorithm

The SHA-256 algorithm takes in a bit string of a maximum size of 2^{64} bits and outputs a 256-bit output called the Digest Message (DM) [3]. The SHA-256 algorithm can be divided into three steps.

1) Padding
2) Data Initialisation
3) Round Hashing

During the Padding step, the input message is subdivided into 512 bit blocks. The last block is padded to a multiple of 512 bits. At the end of the contents of the last block a 1 bit is placed followed by zeroes. The last 64 bits of that block are used to store the message length. Figure 1 show an example of a padded input.

The next step is the data Initialisation. In this step many variables are initialized to values specified by the National Institute of Standards and Technology [3]. The DM takes its initial value, the constants take their values, and the message block is loaded into a vector called the message schedule. The initialization of the message schedule is show in figure 2.

The round hashing step happens once for every message block. During this round, the initial digest message is broken into 8 32-bit components and manipulated by a set of rotations and combinational logic 64 times in an inner loop. This inner loop can be seen in Figure 3. The functions in the inner loop and data initialization (sigma, MAJ, etc.) are combinational logic functions defined by NIST [3]. Finally, the after all the rounds are completed the digest message is recombined into a 512-bit format which is the result of the hash.

$$\underbrace{01100001}_{\text{“a”}}\ \ \underbrace{01100010}_{\text{“b”}}\ \ \underbrace{01100011}_{\text{“c”}}\ \ 1\ \ \overbrace{00\ldots00}^{871}\ \ \underbrace{\overbrace{00\ldots011000}^{128}}_{\ell=24}$$

Fig. 1. An example of the SHA-256 padding

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

Fig. 2. The generation of the Message Schedule vector (W) based on the input Message (M)

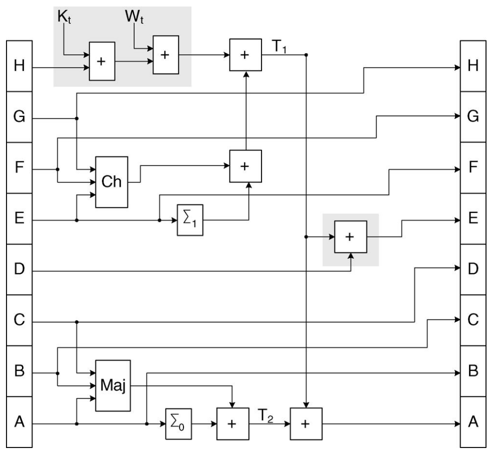


Fig. 3. One round of the inner loop of the SHA-256 algorithm

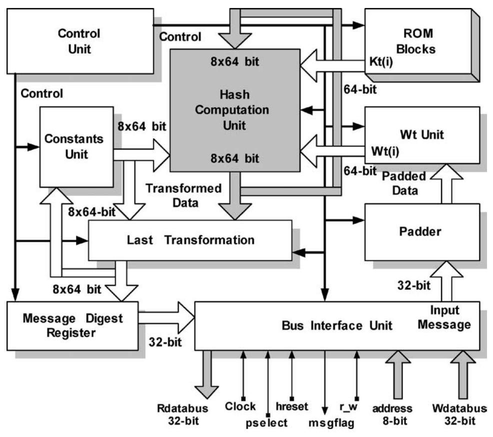


Fig. 4. System overview of the Sklavos et al. architecture

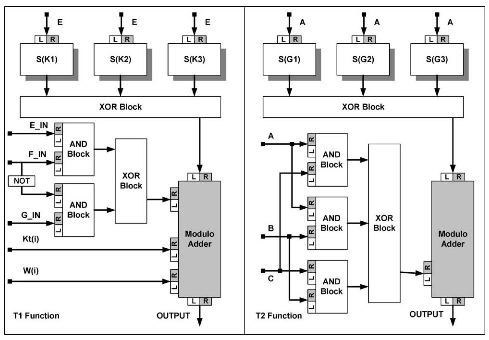


Fig. 5. The T1 and T2 computation

III. Implementation of the SHA-2 Hash Family Standard Using FPGAs

The proposed architecture by Sklavos et al. [4] supports three hash functions (256, 384 and 512) of the SHA-2 standard. Their design is aimed at reducing processing delays for networked devices such as cellphones. The proposed design boasts higher operation frequency, needs less silicon area resources than previous solutions and has a throughput 417% faster than research at the time. What makes this system fast is their:

- inner loop computation
- Data initialization block
- ROM Stored constants

Their solution reuses hardware cleverly because performs 32-bit operations and 64-bit operations on the same hardware. To do so they used a 64-bit buffer which they mask with zeroes during 32-bit computations. A High level overview of their solution can be seen in figure 4.

A. Inner loop

The inner loop computation is simple. However, the key is that they save on hardware resources by reusing the same elements for three different algorithms. All three algorithms are similar in their inner loop or round computation but differ in the constants used for each algorithm. To get around this, they have made the sigma calculations parametrized. Figure 5 shows how each function S(n) takes in a parameter K or G depending on the algorithm.

B. Data Initialization

The data initialization block is made faster due to pipelining and flexible due to parametrization. Figure 6 shows this data initialization block. When calculating W_t values for the SHA-256 algorithm, the first 16 values are straight from the message and the rest are computed from previous W_t values making this step ideal for pipelining. Pipelining is used to compute the sigma values in previous clock cycles. Also, the S(n) and R(n) functions are parameterized for variables J and L depending on the algorithm, thus saving on hardware. Moreover, to speed up the lookup for constants, they've stored the constants for all three algorithms in ROM which speeds up their lookup.

C. Results

The authors have taken their design and implemented on XILINX Virtex v200pq240 FPGA. These results can be seen in figure 7. The FPGA implementation used 2384 Configurable Logic Blocks (CLB) and achieved a throughput of 291 Mbps, 350 Mbps and 467 Mbps for SHA-256, SHA-384 and SHA-512 respectively. It is interesting to note that the throughput increases for each algorithm, this is because each algorithm is processing more data in parallel. Moreover, the architecture by Sklavos et al. achieves a higher data rate than other

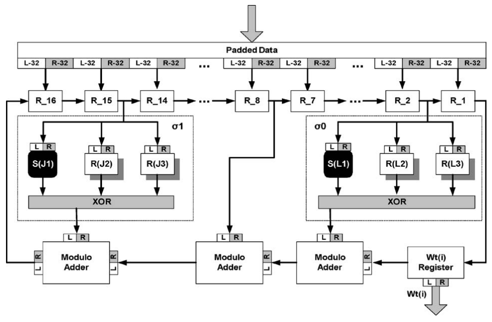
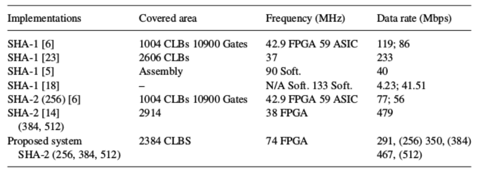


Fig. 6. The data expansion block



Implementations	Covered area	Frequency (MHz)	Data rate (Mbps)
SHA-1 [6]	1004 CLBs 10900 Gates	42.9 FPGA 59 ASIC	119; 86
SHA-1 [23]	2606 CLBs	37	233
SHA-1 [5]	Assembly	90 Soft.	40
SHA-1 [18]	–	N/A Soft. 133 Soft.	4.23; 41.51
SHA-2 (256) [6]	1004 CLBs 10900 Gates	42.9 FPGA 59 ASIC	77; 56
SHA-2 [14] (384, 512)	2914	38 FPGA	479
Proposed system SHA-2 (256, 384, 512)	2384 CLBS	74 FPGA	291, (256) 350, (384) 467, (512)

Fig. 7. Results comparison in the Sklavos et al. implementation

implementations of SHA-256. Simulations showed that the power consumption can be as low as 46mW.

D. Comments

This research implementation demonstrates that high data rates are possible for the SHA2 hash algorithms family. However, there are some drawbacks to their work. First, the comparison made between their SHA-256 implementation and the one in [5] is not completely fair. The previous work they compare to, also computes other hash algorithms from the MD4 family. It is understandable that it is slower given that it is more flexible. Second, their introduction motivates creating faster hardware for cryptography in phones, however, their solution remains at the FPGA implementation. It would have been interesting to know if this design will scale down to the power and area requirements for the mobile applications.

IV. Cost-Efficient SHA Hardware Accelerators

The research by Chaves [6] improves the SHA256 computation throughput and uses less area when compared to the previous implementations in the literature and in use by the industry and 150 times speed up against pure software implementations [6]. Implementation results on several FPGA technologies of the proposed SHA, show that a throughput of 1.4 Gbit/s is achievable for the SHA-256 hash functions **??**. To evaluate both these parameters they focus on the metric throughput per slice which is a ratio of the input bits processing speed versus the area used. Their improvements required them to carefully examined the algorithm for hotspots that can be improved and came up with three techniques to speed up the operations:

- operation rescheduling
- hardware reuse
- memory-based block expansion structures.

The full design by Chaves et al. can be seen in figure **??**.

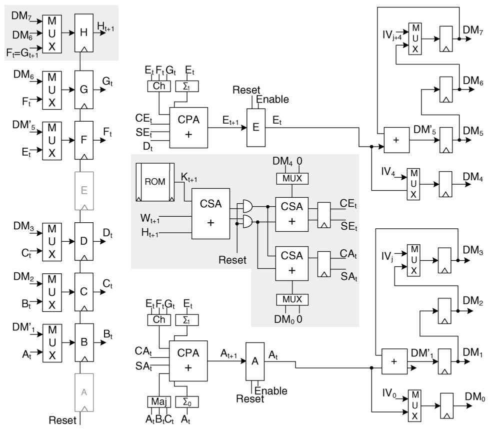


Fig. 8. Full design of Chaves et al.

$$\begin{aligned}
E_{t+1} &= D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + H_t + K_t + W_t \\
A_{t+1} &= \Sigma_0(A_t) + Maj(B_t, C_t, D_t) + \Sigma_1(E_t) \\
&\quad + Ch(E_t, F_t, G_t) + H_t + K_t + W_t. \qquad (10)
\end{aligned}$$

Fig. 9. Computation of A and H

$$\begin{aligned}
\delta_t &= H_t + K_t + W_t = G_{t-1} + K_t + W_t \\
E_{t+1} &= D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \delta_t \\
A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) + \Sigma_1(E_t) \\
&\quad + Ch(E_t, F_t, G_t) + \delta_t
\end{aligned}$$

Fig. 10. Computation of A and H with delta

V. SHA-256 algorithm

A. Operation rescheduling

The technique of operation rescheduling reduces the critical path in the computation of the inner loop of the SHA-256 algorithm. The computation of A and E in the inner loop of SHA-256, as seen in figure 3, require the most amount of time whereas the other buffers only require a shifting operation. The computations of calculating A and E are described in 9. To reduce the critical path of one round, the additions of H, K_t and W_t will happen in a previous round. This computation is described in figure 10. This results E having a critical path of 4 additions to 3 additions.

B. Hardware Reuse

Knowing that the calculation of the new DM at the end of a block uses delayed values of A and E, it is possible to reduce the number of adders needed to calculate the output of the new DM. The relationship between the buffer variables are given in figure 11. Thus the DM can be computed from delayed values of A and E as seen in figure 12. Which means using only four time steps and 2 adders we can compute the values of DM as seen in figure 13. This technique makes the design more area efficient.

C. Data Block Initialization

To efficiently compute the values of W_t a datablock initialization unit is proposed and can be seen in figure 14. This also

$$H_t = G_{t-1} = F_{t-2} = E_{t-3}$$
$$D_t = C_{t-1} = B_{t-2} = A_{t-3}.$$

Fig. 11. Relation between buffer variables

$$\begin{aligned}
\mathrm{DM7}_i &= E_{t-3} + \mathrm{DM7}_{i-1}\\
\mathrm{DM6}_i &= E_{t-2} + \mathrm{DM6}_{i-1}\\
\mathrm{DM5}_i &= E_{t-1} + \mathrm{DM5}_{i-1}\\
\mathrm{DM3}_i &= A_{t-3} + \mathrm{DM3}_{i-1}\\
\mathrm{DM2}_i &= A_{t-2} + \mathrm{DM2}_{i-1}\\
\mathrm{DM1}_i &= A_{t-1} + \mathrm{DM1}_{i-1}
\end{aligned}$$

Fig. 12. Relation between DM and A,E

$$\begin{aligned}
\mathrm{DM}[j+4]_i &= E_{t-j} + \mathrm{DM}[j+4]_{i-1}; \quad 1 \le j \le 3\\
\mathrm{DM}[j]_i &= A_{t-j} + \mathrm{DM}[j]_{i-1}; \quad 1 \le j \le 3.
\end{aligned}$$

Fig. 13. Only two adders are needed

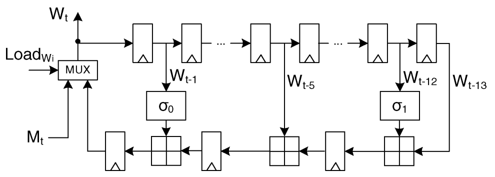


Fig. 14. Only two adders are needed

Architecture	Sklav [22]	Our	McEv. [13]	Our	Helion [23]	Our
Device	XCV	XCV	XC2V	XC2V	XC2PV-7	XC2PV-7
IV	cst	yes	cst	yes	cst	yes
Slices	1060	764	1373	797	815	755
BRAMS	≥1	1	≥1	1	1	1
Freq.	83	82	133	150	126	174
Cycles	n.a.	65	68	65	n.a.	65
Throughput	326	646	1009	1184	977	1370
TP/Slice	**0.31**	**0.84**	**0.74**	**1.49**	**1.2**	**1.83**

Fig. 15. Only two adders are needed

has the effect of reducing the critical path since additions are computed over several past iterations.

D. Results

To analyze the performance of this design, the authors have implemented their design on an VIRTEX II Pro FPGA. The comparison between other implementations can be seen in figure 15. This implementation out performs both the academic implementations as well as the industry implementation. The Chaves implementation consistently has a higher throughput and lower area.

E. Comments

This paper introduces a new implementation to perform the SHA-256 algorithm. However, there are still some questions left unanswered.
First, there was no comparison of Data block expansion structures. The paper proposed three structures to perform Data Block expansion and ruled out 2 due to the fact that some components were not widely available on all FPGAs. Although

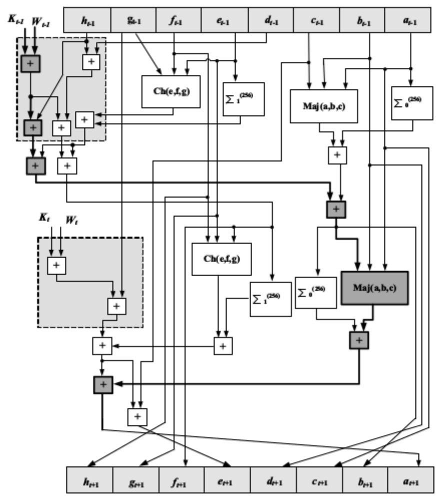


Fig. 16. Only two adders are needed

that may be true, it would have been useful to know what the trade offs are for using the different structures in terms of throughput for designers with access to the right FPGA.
Second, the authors show only a performance analysis of the entire system. How much did each technique contribute to speed up the algorithm or save on area? A breakdown would illustrate the impact each technique and the trade offs of including it or not.
Third, a comparison with the Sklav implementation is not really a fair comparison. This is because the Sklav model is a flexible design that can perform three different types of SHA hashes. The throughput and area considerations for the Sklav design are impacted because of its flexibility.
Lastly, the authors did not explain what benchmark they used to obtain their throughput results. It would have been interesting to know how to reproduce the results.

VI. On the Exploitation of a High-Throughput SHA-256 FPGA Design for HMAC

The solution my Michail et al. [2] is the fastest in the literature. They survey the past literature notice that techniques such as retiming, pre-computation, loop unrolling etc. are used separately to improve the algorithm efficiency. Their results are achieved by using all of the previously mentioned techniques together and then some of their own. The resulting implementation has a tremendous throughput of 11Gbps after place and route on a Xilinx Virtex 6 board. This section will discuss only the unique aspects of this design.

A. Loop Unrolling

In order to speed up computation the authors of [2] used loop unrolling. Figure 16 depicts the unrolling of the inner loop. The inner loop is unrolled twice to obtain a mega block. A choice of two unrolling was chosen based on a previous study that found two unrolling resulted in the best overall throughput and area performance.

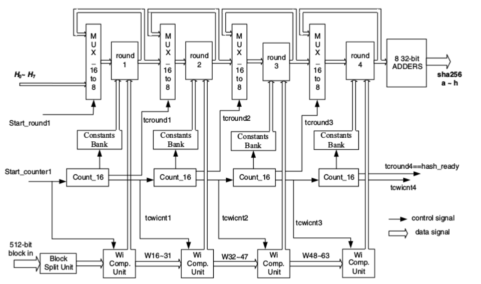


Fig. 17. Four stage pipelining of the inner loop

B. Pipelining

In this study a four stage pipeline was used in order to maximize throughput. By pipelining the round computation, the authors of [2] were able to compute four different hash values at once. This has the effect of almost quadrupling throughput. Again, 4 stages were chosen because it was the optimal throughput versus area optimization.

C. Comments

This research is a great study in the space of VLSI not only because of the amazing results in throughput but also because of the traceability of their work. Why did they choose 4 stage pipelining and not five? Why did they unroll the loop twice? Each technique had a reference to their past work justifying their techniques. However, there were no estimates of how much power it would consume. This is justified because their target application was for servers and routers which are not necessarily limited by power.

VII. Conclusion

This survey is a study of three major VLSI implementations of the SHA-256 algorithm. Each adaptation improved from their predecessors to provide higher throughput over a lower area. They did so using VLSI techniques such as retiming, pre-computation, loop unrolling and pipelining paying careful attention to the parts of the algorithm that these methods were best suited to. The highest throughput achieved was by Michail et al. with a throughput of 11Gbps. Although these implementations were fast, none of these implementations were suited to the embedded computing market. Only one design discussed the estimated power consumption but neither were specifically optimizing for a low power design suited for embedded systems. As technology moves to the internet of things, it is likely that future designs will optimize for power as well.

References

[1] B. Schneier, “Applied cryptography–protocols, algorithms, and...” 1994.

[2] H. E. Michail, G. S. Athanasiou, V. Kelefouras, G. Theodoridis, and C. E. Goutis, “On the exploitation of a high-throughput sha-256 fpga design for hmac,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 1, p. 2, 2012.

[3] P. FIPS, “180-4. secure hash standard,” *National Institute of Standards and Technology*, vol. 17, 2015.

[4] N. Sklavos and O. Koufopavlou, “Implementation of the sha-2 hash family standard using fpgas,” *The Journal of Supercomputing*, vol. 31, no. 3, pp. 227–248, 2005.

[5] S. Dominikus, “A hardware implementation of md4-family hash algorithms,” in *Electronics, Circuits and Systems, 2002. 9th International Conference on*, vol. 3. IEEE, 2002, pp. 1143–1146.

[6] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, “Cost-efficient sha hardware accelerators,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 8, pp. 999–1008, 2008.