Global Journal of Engineering and Technology Review

Journal homepage: www.gjetr.org

Global J. Eng. Tec. Review 4 (4) 73 – 81 (2019)

High-Throughput of SHA-256 Hash Function with Unfolding Transformation

Shamsiah binti Suhaili [1*], Takahiro Watanabe[2]

[1] Faculty of Engineering, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia
[2] Graduate School of Information, production and Systems, Waseda University, 2-7 Hibikino, Wakamatsu-ku, Kitakyushu-shi, Fukuoka 808-0135, Japan

ABSTRACT

Hash Function in cryptography algorithms is used to encrypt the message by giving the appropriate output based on the structure of the hash function itself. This algorithm is important for security application such as Keyed-Hash Message Authentication Code (HMAC), digital signature and others. There are different types of hash function such as MD5, SHA-1, RIPEMD-160, SHA-256, SHA-224, SHA-384, SHA-512 and others. In this paper, the unfolding transformation method was proposed to improve the throughput of the SHA-256 hash function. Three types of SHA-256 hash function were designed namely SHA-256 design, SHA-256 design inner pipelining with unfolding factor 2 and SHA-256 design inner pipelining with unfolding factor 4. The designs were written in Verilog code and the output simulations were verified using ModelSim. The simulation results showed that the proposed SHA-256 inner pipelining unfolding with factor 4 provided the highest throughput which is 4196.30 Mbps, and with factor 2 was superior in terms of maximum frequency and was better than the conventional SHA-256 design.

Type of Paper: other.

Keywords: Cryptography algorithm; FPGA; SHA-256 Hash Function; Unfolding transformation, Verilog

1. Introduction

Cryptography is the science of writing secret codes; to ensure none can read an encrypted message except the intended user. There are three different types of cryptographic algorithms namely symmetric cryptography, asymmetric cryptography and hash function. While symmetric cryptography uses only a key to encrypt and decrypt the message, asymmetric cryptography uses two different keys and the hash function requires no key. This paper focused on the SHA-256 hash function. It transformed a variable message input into a fixed size string hash value [1].

*Paper Info: Revised: October 11, 2019
Accepted: December 03, 2019

* Corresponding author: Shamsiah binti Suhaili
E-mail: sushamsiah@unimas.my
Affiliation: Faculty of Engineering, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia

ISSN 0128-2905.

The hash function output is called hash code, also referred to as hash value or messages digest. The hash function input can be of any length while the output has a fixed length based on the structure of the hash function. It is a one-way function where it is not feasible to find an input message. In this paper, the unfolding transformation method was proposed to improve the throughput of the SHA-256 hash function. Nowadays, security on the network is a major issue in data transmission. A network layer needs to be secure enough with cryptographic algorithms so that it can be used to accommodate encryption and authentication processes. Therefore, high performance of cryptographic hash function algorithm is one of the important aspects of security algorithm. Hence, designing a cryptographic hash function algorithm on reconfigurable hardware with high speed, low power and small area implementation needs to be considered.

The unfolding transformation method is used to improve the throughput of the SHA-256 hash function, which is the purpose of this study. This method focused on the latency of the designs and the architecture refers to Register Transfer Level (RTL). In this paper, unfolding transformation factor 2 and 4 were used to reduce the latencies of the SHA-256 hash function. There was parallel execution for both designs. The combination of inner pipelining and unfolding resulted in further power reduction because the power supply voltage was reduced aggressively, and the frequency of operation was also reduced with small area implementation.

2. Methodology – Unfolding Design

The motivation of using unfolding method was to improve the performance in terms of throughput. Verilog code was used to design SHA-256. The SHA-256 architecture consists of 6 top-level modules such as counter SHA-256, message schedule, constant SHA-256, multiplexer, compression function and output SHA-256. The difference between the conventional SHA-256 design and SHA-256 unfolding designs are the input of the inner structure of SHA-256 unfolding design. By using different inputs, the sequence of constant and message was found to be also different. The input data was 15 blocks input of padded 32-bit data. Equation (1) was used to obtain the input message, W_t for $16 \le t \le 63$. Message schedule SHA-256, Wt

$$\begin{aligned} W_t &= \text{message input} && 0 \le t \le 15 \\ W_t &= \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16} && 16 \le t \le 63 \end{aligned} \tag{1}$$

Where,

$$\sigma_0^{256}(x) = ROTR^7(x) + ROTR^{18}(x) + SHR^3(x) \tag{2}$$

$$\sigma_1^{256}(x) = ROTR^{17}(x) + ROTR^{19}(x) + SHR^{10}(x) \tag{3}$$

σ_0 and σ_1 represented sigma 0 and sigma 1. Both functions were obtained from Equation (2) and Equation (3). Equation (2) meant that the message x was rotated right by 7 bits, the result was then added with a right rotation of x by 18 bits and finally, the result was added with three shifts to the right. As for Equation (3), the message x was rotated by 17 bits, the result was then rotated 19 bits to the right and finally shifted right by 10 bits.

A counter module was used to generate the sequence the message. The final hash code was obtained after the 64 rounds of iteration of the compression function used by the SHA-256 hash function. A Multiplexer module helped to generate eight buffer initialisation of SHA-256, before SHA-256 started processing the message. The constant Kt was defined using 64X32-bit ROM blocks. Finally, the SHA-256 message digest was produced using the output module. The last output of the compression function of SHA-256 was added with buffer initialisation in this model. Modification must be done for each module in order to improve the performance of SHA-256 in terms of throughput. Two parallel inputs of 32 bits and two parallel constants were needed in order to obtain unfolding factor 2. Similarly, for unfolding factor 4, four parallel inputs of 32 bits and four parallel constants were needed to design the unfolding transformation.

Therefore, all inputs for the next sequence of cycle need to be changed based on the movement of input which was already applied in parallel form. Each module needed to be modified in terms of inputs to obtain inner pipelining and unfolding with factors two and four. These modifications provided the improvements for the SHA-256 hash function. The number of cycles is reduced based on the number of J factors using an unfolding design technique [3]. Moreover, the throughput of the SHA-256 algorithm is also increased using this technique. The number of cycles was reduced from 66.5 to 35.5 for unfolding factor 2 and for unfolding factor 4 it was reduced from 35.5 to 19.5. These cycle numbers were obtained from waveform simulation results of the design. It reduced due to the structures of SHA-256 design unfolding design change based on different inputs.

By reducing the number of cycles with unfolding transformation techniques, the throughput of the design improved significantly. Furthermore, the performance of the SHA-256 hash function has improved in terms of frequency due to the inner pipelining method. The frequency of SHA-256 unfolding with factor 2 increased significantly compared to conventional design. Even though the latencies were reduced by factor 2, modification for SHA-256 unfolding with factor 4 increase the area implementation compared with two other SHA-256 designs. However, it provided high throughput because of low latencies.

The unfolding factor 2 and 4 architectures were produced by the modified message schedule and compression function of the SHA-256 algorithm. The unfolding technique with factor 2 and 4 had been implemented in this paper. Each block in the message schedule and compression function must be considered for its modifications. The modifications of each of the block in the message schedule and compression function must be considered. Figure 1 and Figure 2 show the block diagrams of Temp_{1o} and Temp_{2o}. The following block diagrams of Temp_{1o} and Temp_{2o}were the modifications of conventional Temp1 and Temp2. The input sequence of SHA-256 unfolding design remakes the output gave different results. The compression function of the SHA-256 algorithm was added with these equations. Temp_{1o} consists of $\sum 1o$, $\mathrm{Cho(next\_e, e, f)}$, Message,$\mathrm{W}_{t\_1}$ and Constant,K_{t_1}; while Temp_{2o}contains $\sum 0$ o and $\mathrm{Majo(next\_a, a, b)}$. These results were obtained using a 32-bit adder. All data inputs were different for each of the blocks in Temp_{1o} and Temp_{2o} block diagram architectures.

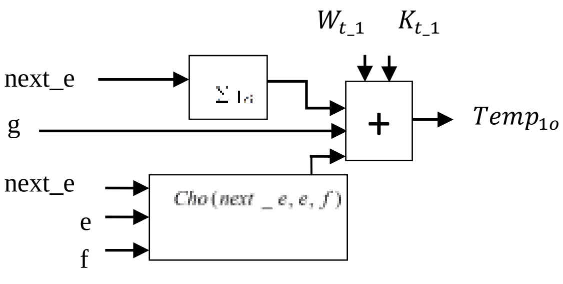


Figure 1. $Temp_{1o}$ Block Diagram Architecture

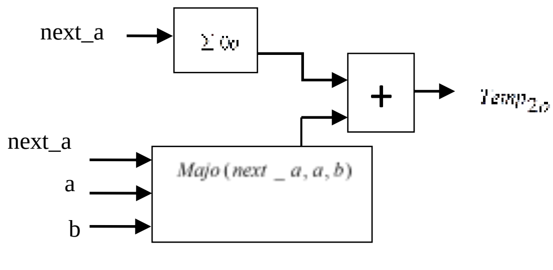


Figure 2. $Temp_{2o}$ Block Diagram Architecture

The Figures 3 and Figure 4 show the two architectures for Cho(next_e,e,f) and Majo(next_a,a,b). The AND, NOT and XOR gates are all components of both the architectures, with different structures of implementation. The data inputs for both the architectures were different from conventional equation for function Ch and Maj. The compression function of the SHA-256 algorithm that was used to obtain data input next_e and next_a.

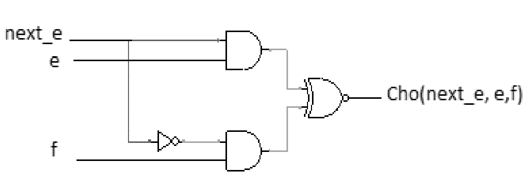


Figure 3. Cho (next_e,e,f) Function

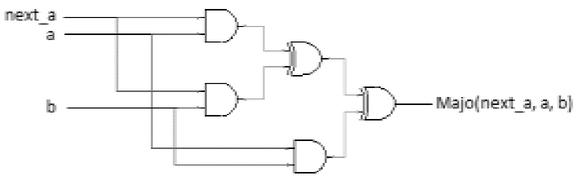


Figure 4. Majo(next_a,a,b) Function

Figures 5 and Figure 6 illustrate the proposed architectures for $\sum 0\,o$ and $\sum 1o$, respectively. $\sum 0\,o$ was depicted by input next_a while for $\sum 1\,o$ the input is next_e. Both architectures had all rotations follow in the right direction with a fixed amount of values. Finally, the final outputs of $\sum 0\,o$ and $\sum 1o$ was obtained by combining all inputs using an XOR gate.

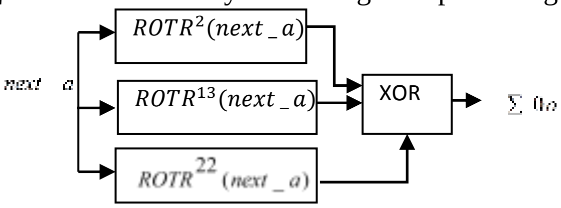


Figure 5 . $\sum_{0o}(next_a)$ Architecture

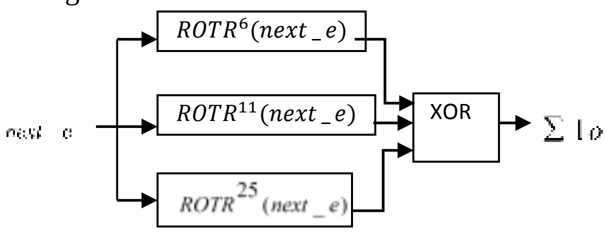


Figure 6. $\sum_{1o}(next\_e)$ Architecture

Output $Temp_{1o}$ and $Temp_{2o}$ was used to calculate new next_eo and next_ao. The following equations show the output for next_eo and next_ao.

next_eo = c + $Temp_{1o}$ (4)

next_ao = $Temp_{1o}$ + $Temp_{2o}$ (5)

Figures 7 and 8 show the block diagram architecture for $Temp_{11}$ and $Temp_{21}$. For $Temp_{11}$, the inputs for Ch1 were next_eo, next_e, and e while for $\sum 11$, the input was next_eo. For $Temp_{21}$, the input for Maj1 were next_ao, next_a and a while for $\sum 01$, the input was next_ao.

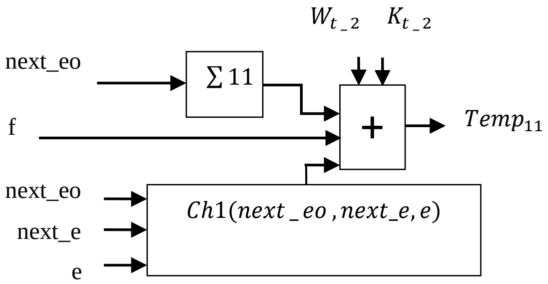


Figure 7. $Temp_{11}$ Block Diagram

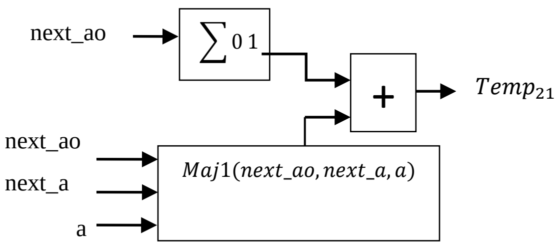


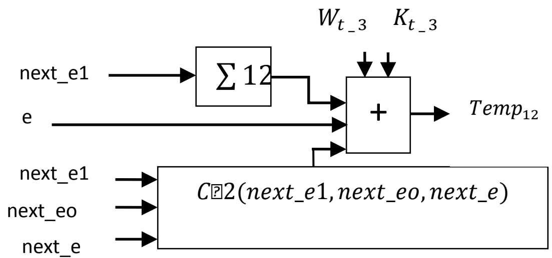
Figure 8. $Temp_{21}$ Block Diagram

Then, $Temp_{11}$ and $Temp_{21}$ were calculated in order to obtain next_e1 and next_a1. The outputs for both equations are illustrated in the following equation.

next_e1 = b + $Temp_{11}$ (6)

next_a1 = $Temp_{11}$ + $Temp_{21}$ (7)

Since unfolding factor 4 needed four parallel executions, it was calculated until $Temp_{12}$ and $Temp_{22}$.



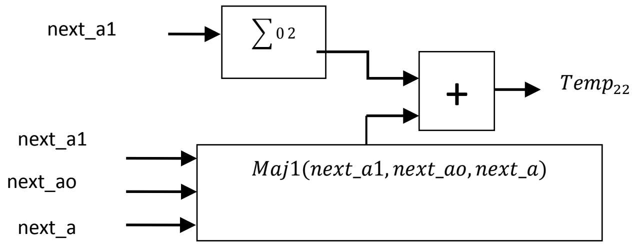


Figure 9. $Temp_{12}$ Block Diagram

Figure 10. $Temp_{22}$ Block Diagram

Figures 9 and Figure 10 show the output for Temp12 and Temp22. The input for $\sum 12$ was next_e1 while for $\sum 02$ the input was next_a1. The sequence of the input was similar to the previous but the data was moved one place. For example, the inputs for Ch2 were next_e1, next_eo and next_e. For Maj2, the inputs are next_a1, next_a0 and next_a. The data for next_e2 and next_a2

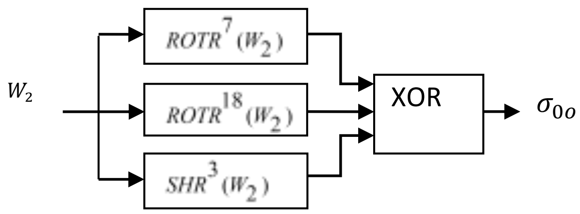
was calculated based on data from Temp12 and Temp22. The following equation shows the next_e2 and next_a2 output.

next_e2 = a + $Temp_{12}$ (8)

next_a2 = $Temp_{12}$ + $Temp_{22}$ (9)

Similar to the compression function, the message schedule was modified from previous results. Once the first message schedule was received, the new equation for sigma0o and sigma1o and next_wto was processed. In order to obtain next_wt and next_wt1, the input was moved one place. The input message for wt0 was from message wt2. This sequence started from wt2 until wt15. After that, the input followed the sequence of next_wt starting from next_wt and next_wto. So, we needed to add another W_message as input such as W_message and W_message1.

The architectures for both σ_{0o} and σ_{1o} functions are shown by Figures 11 and 12. Generating a message schedule for SHA-256 is the main function of these architectures. W_2 was rotated in the right direction with a fixed amount of value for σ_{0o} while for σ_1 , a different value was used rotate it. The W_2 was right shifted by 3 in σ_{0o} architecture, and for σ_{1o} architecture, the W15 was right shifted by 10.



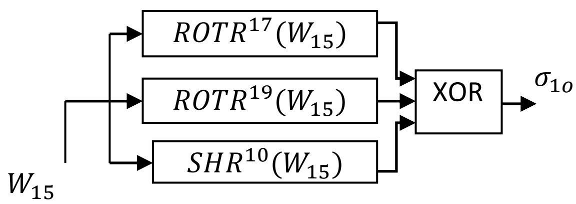


Figure 11. σ_{0_o} Architecture

Figure 12. σ_{1_o} Architecture

For unfolding factor 4, σ_{01} and σ_{11} were calculated with input W3 and next_wt respectively. Then, next_wt1 was obtained by the following Equation (10). Finally, the input for σ_{02} and σ_{12} were obtained from input W4 and next_wto respectively. The next_wto was derived from Figure 13. The structure of next_ wt2 is shown in Figure 13. This figure shows the final structure of the message schedule for unfolding factor 4. The input data for message W0 started with W4 until W15. The output sequence of next_wt used the similar method used in unfolding factor 2. The sequence of next wt was next_wt, nextk_wto, next_wt1 and next_wt2.

$$next\_wt1 = W_2 + \sigma_{01} + W_{11} + \sigma_{11} \quad (10)$$

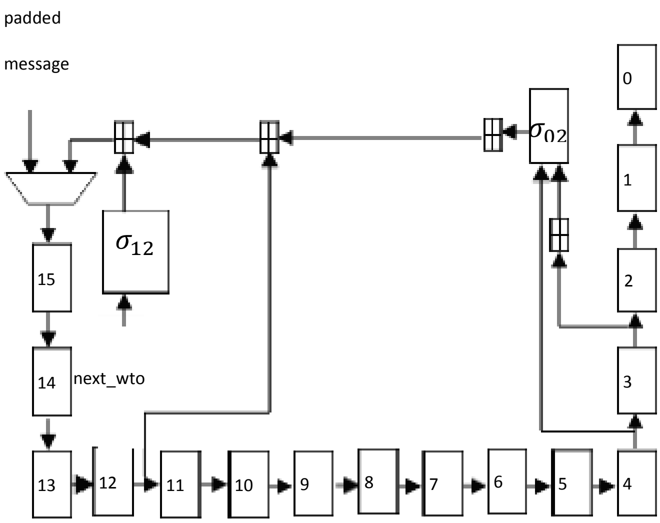


Figure 13. Message Schedule of SHA-256 Unfolding factor 4

3. Result and Discussion

The SHA-256 unfolding factor 2 and 4, the proposed SHA-256, were successfully designed and tested. All designs were written in Verilog code and compilation process was done using Altera Quartus II. ModelSim was used to simulate and verify for both functional and timing simulation of the design. Equation (11) was used to calculate the throughput of these designs.

$$\text{Throughput} = (512 \text{ X } \text{FMax}) / \text{Number of Cycle} \quad (11)$$

The results showed the throughput of SHA-256 with unfolding factor 4 increased significantly compared to other SHA-256 designs. The proposed SHA-256 designs were compared with previous publications as shown in Table 1. Table 1 shows the synthesis and implementation comparison results of other SHA-256 designs in terms of FPGA implementation. By using unfolding transformation, the performance of SHA-256 design in terms of throughput increase significantly. This is because of the inner structure of SHA-256 design was processed in parallel form. This method can improve the performance of SHA-256 design where the number of latencies of the design was reduced. Implementation of SHA-256 design on different devices provided different results. Thus, by choosing appropriate family device for implementation, the better results can be obtained.

Table 1. Synthesis and Implementation Results of Other SHA-256 Designs

Design	Device	ALUTs/ CLBs	Freq (MHz)	Throughput (Mbps)
SHA-256 Design	Arria II GX	855 ALUTs	228.15	1756.58
SHA-256 Unfolding Design (factor 2)	Arria II GX	1345 ALUTs	251.07	3621.07
SHA-256 Unfolding Design (factor 4)	**Arria II GX**	**2064 ALUTs**	**159.82**	**4196.30**
SHA-2 [4]	Virtex 5	320 CLBs	218.2	1719
SHA-2 [4]	Stratix III	795 ALUTs	205.8	1621

SHA(256,384,512) [5]	Virtex v200pq 240-6	2207 CLBs	74	291
SHA-256 [6]	Virtex v200pq240	1060 CLBs	83	326
SHA-256 [7]	Stratix II	2150 ALUTs	143.164	909.816
SHA-256 [8]	Virtex 5 XC5VFX70T	387 Slices	202.54	1580
SHA-256[9]	XC2PV-7	755 Slices	174	1370
SHA-256 [10]	Virtex-II xc2v2000-bf957	1373 Slices	133.06	1009
SHA-256 [11]	-	-	41.97	335.9
SHA-256 [12]	Virtex-4	610 Slices	170.75	1344.98
SHA-256 [13]	Virtex-5 XC5VFX70T	387 Slices	202.54	1.58
SHA-256 [13]	Virtex	1534 CLBs	35.1	2077
SHA-256 [14]	Virtex E	1655 CLBs	36.4	2190
SHA-256 [15]	Virtex E	-	64.1	2052.1
SHA-256 [16]	Virtex II	1708 slices	52.1	3100.9
SHA-256 [17]	Cyclone II	7219 cells	116.24	875.22

Lee Y.K. et al. [17] discovered by using ASIC the implementation of the SHA-256 design provided large area implementation in terms of gates which is 22,025. Even though the throughput of SHA-256 design was high but area implementation is too large. It is better to have balance between area and throughput of SHA-256 design. This paper proposed small area implementation and high throughput of SHA-256 design by using unfolding transformation factor 4. 2064 ALUTs was used by the proposed SHA-256 unfolding factor 4 and 159.82 MHz was the maximum clock frequency of this design. The throughput of this design was improved by increasing the number of unfolding factor. From the table, the result shows that the SHA-256 unfolding design with factor 4 give the highest throughput in terms of FPGA implementation which is 4196.30 Mbps with 159.82 MHz maximum frequency. Consequently, the proposed SHA-256 unfolding factor 2 produced better results in terms of maximum frequency due to the inner pipelining design where 1159 registers were used. The novelty of this paper is the design of SHA-256 using the unfolding transformation with factor 4 can improve the throughput of SHA-256 design. The throughput of the SHA-256 design was improved by using this method due to the small number of latencies compared to the conventional design. The number of clock cycles of SHA-256 unfolding factor 4 design decreased from 66.5 cycles of conventional design to 19.5 cycles of unfolding design. Thus, the high throughput of SHA-256 design was obtained by using the unfolding transformation method.

4. Conclusion

The study successfully completed and tested SHA-256 unfolding factor 2 and 4 designs which were the proposed SHA-256. The area and maximum frequency are comparable to other SHA-256 designs. The proposed SHA-256 unfolding with factor 4 design gave the highest throughput with 4196.30 Mbps based on the comparison with other SHA-256 designs. In conclusion, implementation of unfolding transformation can improve the performance of SHA-256 hash function by reducing the number of cycles where the data generate in parallel transformation. This leads to high throughput of SHA-256 design. The throughput of SHA-256 design also increases significantly by using this methodology. In the future, the Keyed-hash Message Authentication Codes (HMAC) can utilize the proposed SHA-256 design in order to enhance the performance of security design.

Acknowledgements

The authors would like to thank Universiti Malaysia Sarawak (UNIMAS) and Waseda University for providing opportunity and facilities to support this project.

References

(NIST), N. I. (August 2015). Secure Hash Function . Federal Information Processing Standards (FIPS) Publication 180-4.

(1999). In K. Parhi, VLSI Digital Signal Processing Systems : Design and Implementation (pp. 119 - 140). John Wiley & Sons, Inc.

Kahri, F., Mestiri, H., Bouallegue, B. and Machhout, M., 2015, March. Efficient FPGA hardware implementation of secure hash function SHA-256/Blake-256. In 2015 IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15) (pp. 1-5). IEEE. 10.1109/SSD.2015.7348105

IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15), (pp. 1-5).

H. Mestiri, F. K. (2015). Efficient FPGA Hardware Implementation of Secure Hash Function SHA-2. IJCNIS, Vol. 7(No. 1), 9-15.

Michail, H., Kakarountas, A., Milidonis, A. and Goutis, C., 2008. A top-down design methodology for ultrahigh-performance hashing cores. IEEE Transactions on Dependable and Secure computing, 6(4), pp.255-268. DOI: 10.1109/TDSC.2008.15

Michail, H., Milidonis, A., Kakarountas, A., & Goutis, C. (2005, December). Novel high throughput implementation of SHA-256 hash function through pre-computation technique. In 2005 12th IEEE International Conference on Electronics, Circuits and Systems (pp. 1-4). IEEE. 10.1109/ICECS.2005.4633433

Michail, H., Athanasiou, G., Kritikakou, A., Goutis, C., Gregoriades, A., & Papadopoulou, V. (2010, July). Ultra high speed SHA-256 hashing cryptographic module for ipsec hardware/software codesign. In 2010 International Conference on Security and Cryptography (SECRYPT) (pp. 1-5). IEEE.

Ahmad, I. and Das, A.S., 2005. Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs. Computers & Electrical Engineering, 31(6), pp.345-360. https://doi.org/10.1016/j.compeleceng.2005.07.001

Li, M., Xu, J., Yang, X. and Yang, Z., 2009, July. Design and implementation of reconfigurable security Hash algorithms based on FPGA. In 2009 WASE International Conference on Information Engineering (Vol. 2, pp. 381-384). IEEE. DOI**:** 10.1109/ICIE.2009.278

Lee, Y.K., Chan, H. and Verbauwhede, I., 2007, August. Iteration bound analysis and throughput optimum architecture of SHA-256 (384,512) for hardware implementations. In International Workshop on Information Security Applications (pp. 102-114). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-77535-5_8

Padhi, M. and Chaudhari, R., 2017, December. An optimized pipelined architecture of SHA-256 hash function. In 2017 7th International Symposium on Embedded Computing and System Design (ISED) (pp. 1-4). IEEE. 10.1109/ISED.2017.8303943

Shahid, R., Sharif, M.U., Rogawski, M. and Gaj, K., 2011, December. Use of embedded FPGA resources in implementations of 14 round 2 SHA-3 candidates. In 2011 International Conference on Field-Programmable Technology (pp. 1-9). IEEE. 10.1109/FPT.2011.6132680

Sklavos, N. and Koufopavlou, O., 2003, May. On the hardware implementations of the SHA-2 (256, 384, 512) hash functions. In Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03. (Vol. 5, pp. V-V). IEEE. DOI: 10.1109/ISCAS.2003.1206214

Chaves, R., Kuzmanov, G., Sousa, L. and Vassiliadis, S., 2006, October. Improving SHA-2 hardware implementations. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 298-310). Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/11894063_24

McEvoy, R.P., Crowe, F.M., Murphy, C.C. and Marnane, W.P., 2006, March. Optimisation of the SHA-2 family of hash functions on FPGAs. In IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06) (pp. 6-pp). IEEE. DOI: 10.1109/ISVLSI.2006.70

binti Suhaili, S. and Watanabe, T., 2017, November. Design of high-throughput SHA-256 hash function based on FPGA. In 2017 6th International Conference on Electrical Engineering and Informatics (ICEEI) (pp. 1-6). IEEE.

Sun, W., Guo, H., He, H. and Dai, Z., 2007, October. Design and optimized implementation of the SHA-2 (256, 384, 512) hash algorithms. In 2007 7th International Conference on ASIC (pp. 858-861). IEEE. DOI: 10.1109/ICASIC.2007.4415766

He, Z., Wu, L. and Zhang, X., 2018, November. High-speed Pipeline Design for HMAC of SHA-256 with Masking Scheme. In 2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID) (pp. 174-178). IEEE.