Pertanika J. Sci. & Technol. 32 (1): 31 - 44 (2024)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

Towards Maximising Hardware Resources and Design Efficiency via High-Speed Implementation of HMAC based on SHA-256 Design

Shamsiah Suhaili*, Norhuzaimin Julai, Rohana Sapawi and Nordiana Rajaee

Department of Electrical and Electronic Engineering, Faculty of Engineering, Universiti Malaysia Sarawak, 94300 UNIMAS, Kota Samarahan, Sarawak, Malaysia

ABSTRACT

Some applications, such as Message Authentication Code (MAC), rely on different hashing operations. There are various hash functions, including Message-Digest 5 (MD5), RACE Integrity Primitives Evaluation Message Digest 160 (RIPEMD-160), Secure Hash Algorithm 1 (SHA-1), and Secure Hash Algorithm 256 (SHA-256), among others. The network layer is the third of seven layers of the Open Systems Interconnection (OSI) concept, also known as the Internet. It handles network addressing and physical data routing. Nowadays, enhanced internet security is necessary to safeguard networks from illegal surveillance. As a result, Internet Protocol Security (IPsec) introduces secure communication across the Internet by encrypting and/or authenticating network traffic at the IP level. IPsec is an internet-based security protocol. Encapsulating Security Payload (ESP) and Authentication Header (AH) protocols are separated into two protocols. The MAC value is stored in the authentication data files of the Authentication Header and Encapsulating Security Payload. This article analyses a fast implementation of the Hash-based Message Authentication Code (HMAC), which uses its algorithm to ensure the validity and integrity of data to optimise hardware efficiency and design efficacy using the SHA-256 algorithm. During data transfer, HMAC is critical for message authentication. It was successfully developed using Verilog Hardware Description Language (HDL) code with the implementation of a Field Programmable Gate Array (FPGA) device using the Altera Quartus II Computer-Aided Design (CAD) tool to enhance the maximum frequency of the design. The accuracy of the HMAC design, which is based on the SHA-256 design, was examined and confirmed

ARTICLE INFO

Article history:
Received: 04 November 2022
Accepted: 10 May 2023
Published: 06 November 2023

DOI: https://doi.org/10.47836/pjst.32.1.02

E-mail addresses:
sushamsiah@unimas.my (Shamsiah Suhaili)
jnorhuza@unimas.my (Norhuzaimin Julai)
srohana@unimas.my (Rohana Sapawi)
rnordiana@unimas.my (Nordiana Rajaee)
* Corresponding author

ISSN: 0128-7680
e-ISSN: 2231-8526

using ModelSim. The results indicate that the maximum frequency of the HMAC-SHA-256 design is approximately 195.16 MHz.

Keywords: Field Programmable Gate Array, hash function, Hash-based Message Authentication Code, Secure Hash Algorithm 256, Verilog Hardware Description Language

INTRODUCTION

There are seven layers, and the internet layer is the network layer in which data transfer from one terminal to another depends on the address and routing network. Traffic networks are prone to eavesdropping and illegal access without a network-integrated security element. However, selecting a suitable encryption and authentication product for the network can solve this problem. The internet community created the Security Protocol (Randall, 1999). The third network layer of the seven-layer OSI architecture employs the IPsec protocol. The seven layers are divided into application, presentation, session, transport, network, data link, and physical layers. One of the network encryption protocols is IPsec (IP Security), the most recent IP-based technology.

The IP provides network authentication and encryption to protect the network from illegal surveillance. Because of its improved capabilities, IP Security has become a fact of life in terms of network security for Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). The IPsec is divided into two protocols: Authentication Header (AH), which examines IP packet authentication and data integrity, and Encapsulating Security Payload (ESP), which encrypts and authenticates the message. Both AH and ESP are equipped with two different modes: tunnel mode and transit mode; as a whole, the IP packet is encrypted in tunnel mode, while only the transport layer is encrypted in the latter. On the other hand, HMAC-MD5, HMAC-SHA, and HMAC-RIPEMD160 are authentication and data integrity methods. These methods may be used to safeguard all distributed applications, e-mail, file transfers, and web access.

This article focuses on computing the Hash-based Message Authentication Code (HMAC) using the MAC (Message Authentication Code) algorithm. Message Authentication Code (MAC) is used to verify the validity of a message, while HMAC is a subset of MAC that uses a cryptographic hash function and a private key for verification. It accepts arbitrary input with a specified key and produces MAC output. The authentication data element in the AH header contains this MAC value. Network transmission operations are followed using the same key to obtain the same MAC at the destination. The message is valid if the MAC value received at the destination corresponds to the one broadcasted. Similar to AH, ESP enables the use of MAC with HMAC. The encryption procedure takes place before the IP layer, which is at the IPsec layer when the application sends the message across the network. A message is routed via the network to its destination using an IP address, part of an IP layer. The router will then determine

the destination address based on the sender's IP address. The decryption of the packet is required to access the sent data.

Security has recently emerged as a hot topic among researchers. Various cryptography algorithms have been developed to enhance the effectiveness of these information-protecting processes. Message digests are generated using hash function techniques during data transmission. As a result, it becomes a crucial tool for embedding security in e-mail, Internet banking, and other applications. A hash function generates a fixed-length output from an arbitrary-length message input. The one-way nature of hash functions makes converting a hash value to a message input challenging. A hash function is a cryptography technique that does not require a key, such as MD5, RIPEMD160, or SHA-1. In this study, an SHA family was constructed and tailored to meet the performance requirements for cryptographic algorithms. There are four types of hash functions in SHA-2, which are SHA-224, SHA-256, SHA-384, and SHA-512. The length of SHA determines the output length of these hash algorithms, ranging from 256 to 512 bits. This article presents the design of the SHA-256 hash function.

This study optimises hardware resources and performance by utilising the hash function of SHA-256 with a Message Authentication Code. Meanwhile, IPsec and HMAC-SHA-256 are focused on several related projects to optimise hardware size, performance, and consumption. McLoone and McCanny (2002) presented IPsec hardware on a single chip that included Rijndael and HMAC-SHA-256. The wireless design raises the maximum frequency, as shown in a previous study (Selimis et al., 2003). On the other hand, the HMAC with SHA-1/MD5 was initially presented in earlier research (Wang et al., 2004), where hardware complexity was minimised, and an efficient hash function structure was devised to share hardware. Additionally, Michail et al. (2004) demonstrated the HMAC-SHA-1 implementation on an FPGA device, whereas Yiakoumis et al. (2005) showed the execution of a small-sized, high-speed HMAC-SHA-1.

Improvement methods adopted by Khan et al. (2007) used pipelining and parallelism to create the HMAC-hash unit and combine them into a single reconfigurable unit. Even though these ideas worked well, the greatest frequencies they could reach were only a few tens of MHz, or up to 111 MHz, as presented in research by Yiakoumis et al. (2005). Meanwhile, an FPGA implementation of HMAC based on SHA-256 was designed in previous studies by Juliato and Gebotys (2011) and Rubayya and Resmi (2015). The results were obtained using a Xilinx device in both designs. Furthermore, the HMAC design in Rubayya and Resmi (2015) demonstrated significant improvement. It suggests that greater performance is needed to meet the demands of current systems.

Table 1 shows the previous design of HMAC with various types of hash functions, such as SHA-1, MD5, and RIPEMD-160. Numerous studies have been conducted on HMAC, but not all of them have focused on the frequency maximum of FPGA design implementation (Choi & Seo, 2020; Oku et al., 2018; Lin et al., 2017; Ravilla & Putta, 2015a; Ravilla &

Table 1
Previous HMAC design

No.	Author	FPGA Device	HMAC Design	Frequency Maximum (MHz)
1	McLoone and McCanny (2002)	Xilinx XCV1000E	HMAC (SHA-1)	50
2	Selimis et al. (2003)	V150bg352	HMAC (SHA-1)	82
3	Wang et al. (2004)	EP2OKIOOOEBC652-IX	HMAC (SHA-1/MD5)	22.67
4	Michail et al. (2004)	Xilinx V3200efg1156	HMAC (SHA-1)	62.0
5	Yiakoumis et al. (2005)	Xilinx VirtexE-8	HMAC (MD5) HMAC (SHA-1)	55 111
6	Khan et al. (2007)	Xilinx XC2V4000	HMAC (MD5, SHA-1, RIPEMD-160)	43.47
8	Juliato and Gebotys (2011)	Altera Apex 20K, EP20K1000EBC652	HMAC (SHA-256)	35.55
	Juliato and Gebotys (2011)	Xilinx Virtex-E, XCV1600EBG1156	HMAC (SHA-256)	48.12
	Juliato and Gebotys (2011)	Xilinx Virtex-II, XCV2V4000BF957	HMAC (SHA-256)	59.66
9	Rubayya and Resmi (2015)	Xilinx Device (no mention device name)	HMAC (SHA-256)	110.009
10	Ravilla and Putta (2015a, 2015b)	No FPGA implementation	HMAC (SHA-256)	-
11	Choi and Seo (2020)	No FPGA implementation	HMAC (SHA-256)	-
12	Chen and Yuan (2012)	No FPGA implementation	HMAC (SHA-256)	-
13	Lin et al. (2017)	No FPGA implementation	HMAC (SHA-256)	-
14	Oku et al. (2018)	No FPGA implementation	HMAC (SHA-256)	-
15	Jung and Jung (2013)	No FPGA implementation	HMAC-based RFID mutual authentication	-
16	Kieu-Do-Nguyen et al. (2022)	Virtex 4/virtex 5	HMAC-SHA-256	188
17	Pham et al. (2022)	Virtex2 XC2VP20	SHA-256	165

Putta, 2015b; Jung & Jung, 2013). The reference study by Kieu-Do-Nguyen et al. (2022) combined all SHA-2 families into one core, such as HMAC-SHA2-224/256/384/512, whereas Pham et al. (2022) designed only the combination of SHA-256/512/256d hash function. This article focuses on SHA-256 because of its wide implementation in security design implementations, such as Bitcoin, also known as cryptocurrency.

MATERIALS AND METHODS

Message authentication is the process of verifying the authenticity of messages, where two important factors must be considered in verifying the authenticity of a message, such

as a source and that the message has not been altered. The traditional encryption method authentication takes place when the sender and the receiver possess the same key throughout the transmission process (Stallings, 1996). In other words, only the genuine user has the key to decrypt the message. Figure 1 shows the block design for the complete message authentication process, which contains a secret key to generate data into the algorithm. The sender receives this message and its MAC; once the MAC has been compared to the MAC at the receiver, the receiver must determine whether it has received the same MAC using the same secret key. The message has not been changed and is deemed authentic if the output MAC matches the one transmitted.

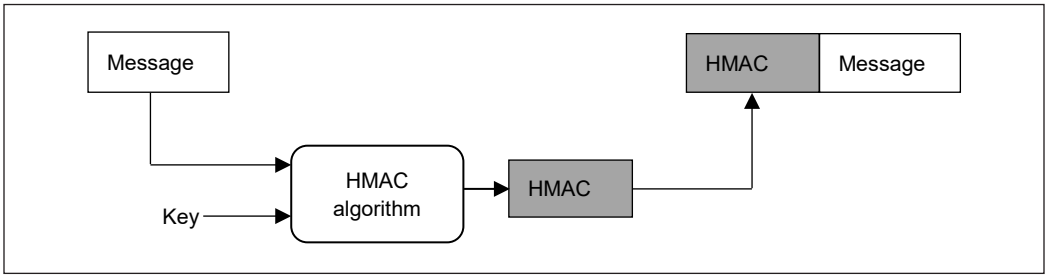

Figure 1. Message authentication using HMAC

One application that uses the hash function to verify message authentication is HMAC. An HMAC based on SHA-256 was implemented in this study. The input key is first hashed before any input text using HMAC. The input key is generated by XORing the inner pad (Ipad) with the input text. An inner pad value is 36 hexadecimal values with a 64-byte timeout, depending on the length of the key. The output of this SHA-256 hash function will be fed into the next SHA-256 to produce the HMAC design output. Before hashing the first SHA-256 output, the key inputs must be XORed with the outer pad (Opad), which has 5C values in hexadecimal (FIPS PUB 198-1, 2008; FIPS PUB 180-4, 2015). The concatenation of the 64-byte key, K_0, and Opad outputs, as well as the output of the first SHA-256, will then be hashed together to produce the HMAC output. Equation 1 shows an MAC calculated with the HMAC function over a textual representation of the data, while Figure 2 depicts the HMAC structure using SHA-256. The symbols $\oplus$ and || stand for XOR and concatenation, respectively.

$$\text{MAC(Message input)}_t = \text{HMAC(K, Message input)}_t$$

$$\text{H}((K_0 \oplus \text{Opad}) \,||\, \text{H}((K_0 \oplus \text{Ipad}) \,||\, \text{Message input}))_t \qquad [1]$$

Message Authentication Code is one of the applications of the hash function. Figure 2 depicts the high-level implementation of the HMAC design, comprising an input pad (Ipad) with a fixed 36 hexadecimal value and an output pad (Opad) with a fixed 5C hexadecimal

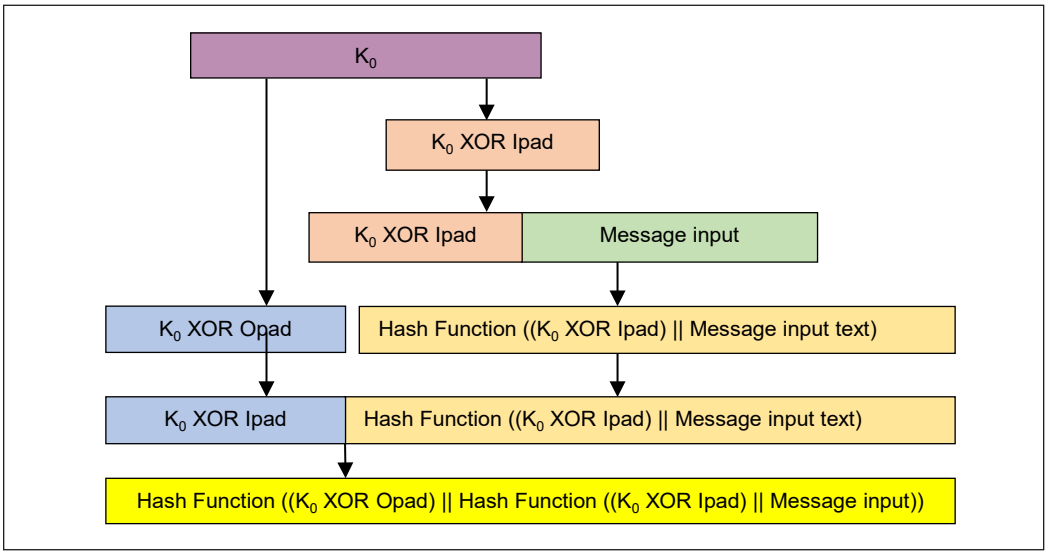

Figure 2. Illustration of HMAC construction

value. The values from the Ipad are sent into the initial SHA-256 design. The counter then causes the HMAC count to generate the output of the first SHA-256 to the second SHA-256 to obtain the overall HMAC output. HMAC is an abbreviation for Hash-based Message Authentication Code, a cryptographic authentication technique that uses a hash function and a secret key. Two identical hash function architectures are used in the HMAC design to obtain the HMAC results. Meanwhile, two hash functions must be used in the HMAC architecture to complete the HMAC structure. In this study, SHA-256 algorithms were employed in the HMAC design. Figure 3 depicts the architecture of an HMAC design with two SHA-256 hash algorithms.

Moreover, Figure 3 illustrates how the key Ipad input message was integrated with the text input. These values will be used as the SHA-256 hash function's input message. There will be two instances of 512-bit blocks if the initialisation value and message input are used. The state machine controls the first and second SHA-256 input in the proposed design. As a result, the first SHA-256 must be executed first, and the second SHA-256 must wait for the message input from the output of the first SHA-256 design, as shown in Figure 3. HMAC uses two hash functions to operate. Two SHA-256 hash functions are employed in HMAC design. The initial SHA-256 algorithm must be executed until results are achieved. These findings serve as the input for the second SHA-256 algorithm with message input. Figure 3 depicts the concatenation of a key with input from an Ipad and SHA-256 with input from a message. Therefore, the validity of the results of initial SHA-256 remains crucial in acquiring the final HMAC based on SHA-256.

The 64 states in this design have generated the sequence input for the first and second SHA-256 algorithms. Thirty-two states were executed: 16 for the first 512 bits and another

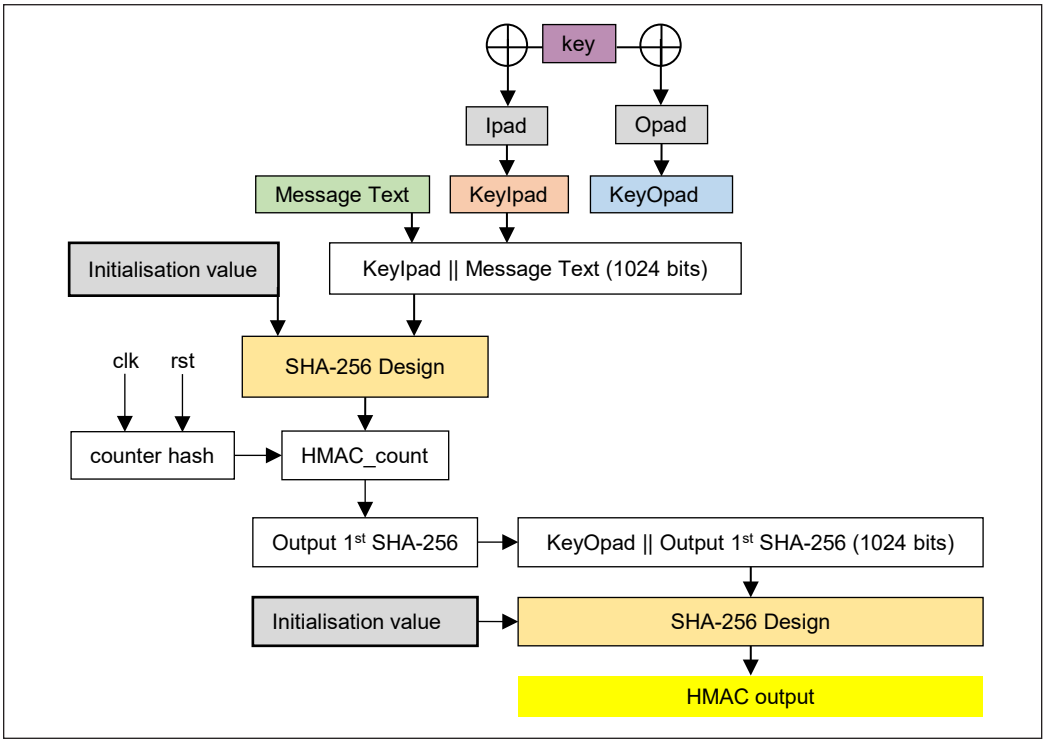

Figure 3. Implementation of proposed HMAC-SHA-256

16 for the second 512 bits SHA-256 to process the message for the first SHA-256. The message will use the output of the first 512-bit hash function to obtain the output of the second 512-bit loop of the input message. An HMAC count is used to generate the correct output SHA-256 hash function with the help of a counter. Input Ipad is the same concept as message input to the second SHA-256. The key Opad output will be combined with the first SHA-256 output. There will be two 512-bit blocks. The state begins at 33 and progresses to 48 for the first 512 bits. The remaining 512 bits will then be executed until 64 states are reached. Similar to the previous SHA-256 process, the output of the second 512-bit block of the second SHA-256 will use the output of the first 512 bits of the second hash function. Finally, this will produce the resulting output obtained in this study.

Secure Hash Functions authenticate messages; therefore, it is necessary to comply with the hash function requirement. A hash function must possess certain properties. Some of these properties include generating outputs of fixed length and the computational infeasibility of determining an x such that Hash(x) equals hash value. It is true for any given code h. It is easy to calculate the hash code, but it is impossible to recover the original message by reversing it. In addition, it is computationally impossible to determine $y \neq x$ for any given block x for which Hash(x) = Hash(y). In other words, identical hash codes are impossible to find, and all these characteristics constitute weak hash functions. Additionally,

no pair (x, y) can be found for which Hash(x) = hash value, computationally, (y). If the last property of the hash function criteria is met, the function is considered "strong." Any message less than 264 bits in length can be entered into the SHA-256 algorithm. SHA-256 processes 512-bit message inputs and 160-bit initial values to produce a 160-bit hash code output. Figure 4 depicts the 512-bit message input of the SHA-256 hash function.

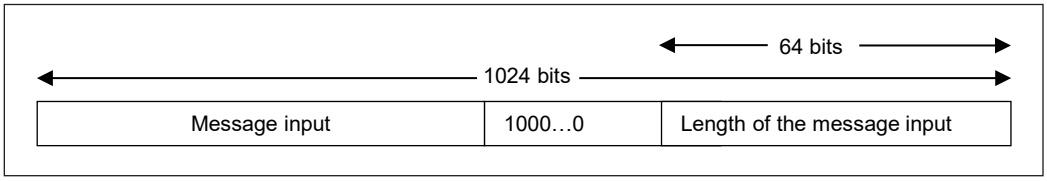

Figure 4. 512-bit message input of SHA-256

Several considerations must be made to generate output. The message must first be padded to almost exactly match 960 modulo 1024. Figure 4 shows a 1-bit input following the input message. It is then padded with 0 bits to produce the total length of the message. After the message is padded, 64-bit inputs are added to the message. Message padding consists of 512 bits and the message's total length. HMAC-based SHA-256 uses the SHA-256 approach to determine authentication in this design. In other words, the SHA-256 hash algorithm was utilised to construct HMAC. Based on the SHA-256 algorithm, the SHA-256 architecture has eight fixed inputs. Hence, the SHA-256 algorithm requires eight variables as initial input during hash computation. Table 2 displays the hexadecimal buffer initialisation of the SHA-256 hash function. Eight distinct input buffer initialisations exist; they are used during the first phase of execution, and their values are fixed for all SHA-256 hash functions. After the startup operation, the input message is processed in 1024-bit blocks of 32 bits each. Figure 5 displays the 64 highest-level steps of the SHA-256 message compression process.

Table 2
Buffer initialisation of SHA-256

Register	Buffer Initialisation (Hex)
A	32'h6a09e667
B	32'hbb67ae85
C	32'h3c6ef372
D	32'ha54ff53a
E	32'h510e527f
F	32'h9b05688c
G	32'h1f83d9ab
H	32'h5be0cd19

The input message is padded during an early step of the SHA-256 hash process. Padding the message begins once the input for the message is received, and the message is completed by appending a single one-bit. Next, n zero bits will be added, and this pattern will continue until the total number of bits in the message equals 448 modulo 512. The final 64 bits are set aside specifically for use in relation to the calculation of how long the message should be. The message input capacity is 512 bits. The message scheduler computes the message, W_t of SHA-256. For $0 \leq t \leq 15$, a message is extracted directly from the input message,

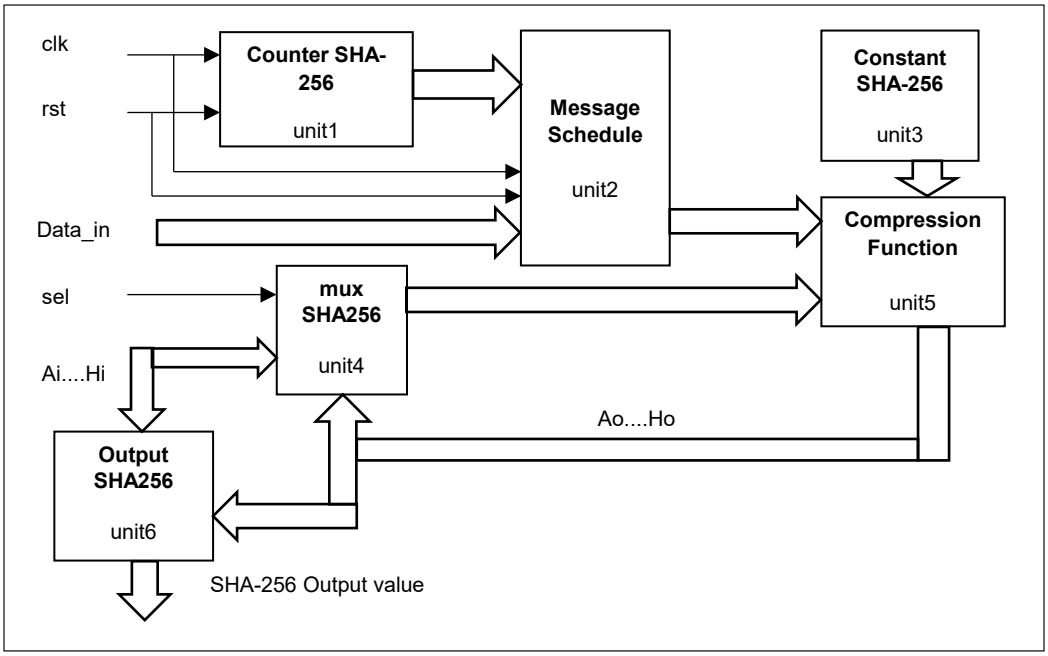

Figure 5. Top level of SHA-256 design

whereas for $16 \leq t \leq 63$, a message W_t is calculated using Equation 2. The value t denotes the number of transformation rounds. $ROTR^n(x)$ is a right rotation of x by n bits, whereas $SHR^n(x)$ is a right shift of x by n bits, as shown in Equations 3 and 4.

Message schedule SHA-256, W_t

$$W_t = \text{message input} \qquad 0 \leq t \leq 15$$

$$W_t = \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16} \qquad 16 \leq t \leq 63 \qquad [2]$$

where,

$$\sigma_0^{256}(x) = ROTR^7(x) + ROTR^{18}(x) + SHR^3(x) \qquad [3]$$

$$\sigma_1^{256}(x) = ROTR^{17}(x) + ROTR^{19}(x) + SHR^{10}(x) \qquad [4]$$

The SHA-256 compression function is made up of four functions that round from t = 0 to t = 63. The four functions are *Ch*(*x,y,z*), *Maj*(*x,y,z*), $\Sigma_0(x)$ and $\Sigma_1(x)$, as shown in Equations 5, 6, 7 and 8. The symbols $\wedge, \neg$ and $\oplus$ represent the logical AND gate, NOT gate, and XOR gate, respectively.

$$Ch(e,f,g) = (e \wedge f) \oplus (\neg e \wedge g) \qquad [5]$$

$$Maj(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \quad [6]$$

$$\sum_0 (a) = ROTR^2(a) + ROTR^{13}(a) + ROTR^{22}(a) \quad [7]$$

$$\sum_1 (e) = ROTR^6(e) + ROTR^{11}(e) + ROTR^{25}(e) \quad [8]$$

The hash computation was used to construct eight variables with initial values to evaluate the four functions of Equations 5, 6, 7 and 8. The message input, W_t, and constant K_t form the 64 iterative operations. The output of the following Equations 9, 10 and 11 is the output of hash values.

$$Temp_1 = h + \sum_1 (e) + Ch(e,f,g) + K_t + W_t \quad [9]$$

$$Temp_2 = \sum_0 (a) + Maj(a,b,c) \quad [10]$$

$$\begin{aligned} h &= g \\ g &= f \\ f &= e \\ e &= d + Temp_1 \\ d &= c \\ c &= b \\ b &= a \\ a &= Temp_1 + Temp_2 \end{aligned} \quad [11]$$

After 64 iterations, the modulo-32-bit adders calculate the hash values, H_0 to H_7. The SHA-256 hash value in its final form is generated using the Big-endian format.

$$H_0 = a + H_0, H_1 = b + H_1, H_2 = c + H_2, H_3 = d + H_3$$

$$H_4 = e + H_4, H_5 = f + H_5, H_6 = g + H_6, H_7 = h + H_7$$

Message Digest = $H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4 \,||\, H_5 \,||\, H_6 \,||\, H_7$

RESULTS AND DISCUSSION

Figure 6 shows the timing simulation waveform result of the HMAC-SHA-256 design with the message input text “Sample #1”. HashCalc validated the HMAC values to ensure output accuracy. Based on the simulation waveform results, the output of the HMAC value provides the correct result of the HMAC value without error, which is similar to the calculation from HashCalc software, as shown in Figure 7. FMax is the maximum

Figure 6. Simulation waveform of SHA-256 design

clock frequency of HMAC-SHA-256 that a digital design can operate at, and it improves greatly when a clock constraint is applied to the design. Figure 8 depicts the maximum clock frequency of HMAC-SHA-256 with SDC 5.3 clock limitations.

Table 3 displays the proposed HMAC-SHA-256 design and other publications utilising HMAC on various FPGA family devices. HMAC-SHA-256 was successfully designed using Altera Quartus II 15.0. ModelSim-Altera 10.3d was used for functional and timing simulation to validate

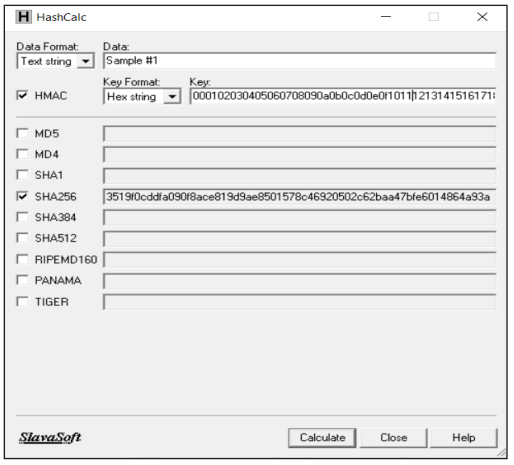

Figure 7. HMAC calculation value of SHA-256 design with specific key

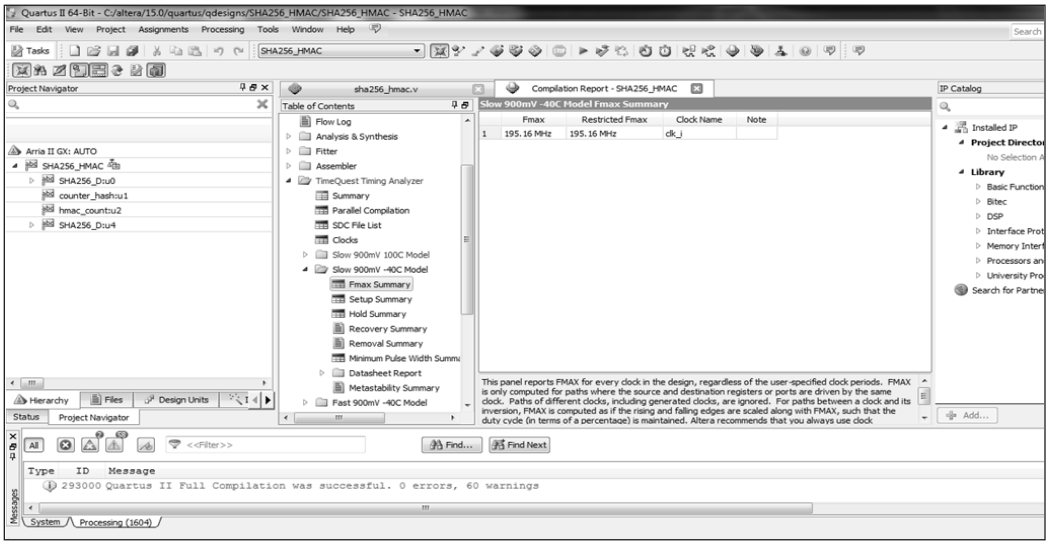

Figure 8. Maximum clock frequency (FMax) of HMAC-SHA-256

Table 3
FPGA-based implementation comparison of the previous HMAC design

Authors/year	FPGA Device	Design	Maximum Frequency (MHz)	LUT/ CLB /LE	Reg
Proposed Design	Altera Arria II GX	Proposed HMAC (SHA-256)	195.16	**3953**	2714
Kieu-Do-Nguyen et al. (2022)	Virtex 4/Virtex 5	HMAC (SHA-256)	188	**1615**	**-**
Pham et al. (2022)	Virtex2 XC2VP20	SHA-256	165	3695	-
Rubayya and Resmi (2015)	Xilinx Device (no mention device name)	HMAC (SHA-256)	110.009	6861	-
Juliato and Gebotys (2011)	Altera Apex 20K, EP20K1000EBC652	HMAC (SHA-256)	35.55	9231	-
Juliato and Gebotys (2011)	Xilinx Virtex-E, XCV1600EBG1156	HMAC (SHA-256)	48.12	3463	-
Juliato and Gebotys (2011)	Xilinx Virtex-II, XCV2V4000BF957	HMAC (SHA-256)	59.66	3608	-
Khan et al. (2007)	Xilinx XC2V4000	HMAC (MD5, SHA-1, RIPEMD-160)	43.47	7484	-
Yiakoumis et al. (2005)	Xilinx VirtexE-8	HMAC (MD5) HMAC (SHA-1)	55 111	686	-
Michail et al. (2004)	Xilinx V3200efg1156	HMAC (SHA-1)	62.0	6011	-
Wang et al. (2004)	EP2OKIOOOEBC652-IX	HMAC (SHA-1/MD5)	22.67	-	-
Selimis et al. (2003)	V150bg352	HMAC (SHA-1)	82	1018	-
McLoone and McCanny (2002)	Xilinx XCV1000E	HMAC (SHA-1)	50	7247	-

the output results. The maximum frequency of the HMAC-SHA-256 design increases dramatically on the Arria II GX with 3953 LUT and 2714 total registers, as shown in Table 3. Based on these findings, synthesis and implementation on Arria II GX give fast speed with a maximum frequency of 195.16 MHz compared to other HMAC publications utilising various hash functions on FPGA family device types. Furthermore, with the assistance of Altera Quartus II and TimeQuest Timing Analyser advisors, HMAC-SHA-256 design results are improved (https://www.altera.com/en_US/pdfs/literature/ug/ug_tq_tutorial.pdf). By applying SDC clock constraint 5.3 to the HMAC-SHA-256 design on Arria II GX, the maximum frequency of the design can be met under stable setup and hold conditions. Thus, this study proposed a high-performance and error-free HMAC-SHA-256 design with appropriate FPGA devices and clock constraints that meet the design requirement.

CONCLUSION

The design of HMAC-SHA-256 was successful through the use and development of high-speed computing, which possessed a maximum frequency of 195.16 MHz. FPGA

implementation on the Arria II GX can offer great speed. Moreover, the design can be significantly enhanced with the assistance of advisor Altera Quartus II. Furthermore, providing the design with the proper SDC clock constraints will allow the TimeQuest timing analyser to meet the time requirements.

ACKNOWLEDGEMENTS

The author thanks the Ministry of Higher Education, Malaysia, Fundamental Research Grant Scheme (FRGS/1/2020/TK0/UNIMAS/02/11), Universiti Malaysia Sarawak (F02/FRGS/2035/2020) and Osaka Gas Foundation in Cultural Exchange (OGFICE) Research Grant Scheme (IG/F02/OSAKA/01/2022**)** for supporting this work.

REFERENCES

Chen, F., & Yuan J. (2012). Enhanced key derivation function of HMAC-SHA-256 algorithm in LTE network. In 2012 Fourth International Conference on Multimedia Information Networking and Security (pp. 15-18). IEEE Publishing. https://doi.org/10.1109/MINES.2012.106

Choi, H., & Seo, S. C. (2020). Optimization of PBKDF2-HMAC-SHA256 and PBKDF2-HMAC-LSH256 in CPU environments. In I. You (Ed.), *Information Security Applications* (pp. 321-333). Springer Cham. https://doi.org/10.1007/978-3-030-65299-9_24

FIPS PUB 198-1. (2008). *Federal Information Processing Standards, The Keyed-Hash Message Authentication Code (HMAC).* Information Technology Laboratory National Institute of Standards and Technology Gaithersburg. https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.198-1.pdf

FIPS PUB 180-4. (2015). *Federal Information Processing Standards, Secure Hash Standard (SHS).* Information Technology Laboratory National Institute of Standards and Technology Gaithersburg. https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf

Juliato, M., & Gebotys, C. (2011). *FPGA Implementation of an HMAC Processor based on the SHA-2 Family of Hash Functions*. University of Waterloo Technical Report. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5043ce0a65691fd16ff7a546e6c0013d9ee190ca

Jung, S. W., & Jung, S. (2013). HRP: A HMAC-based RFID mutual authentication protocol using PUF. In *The International Conference on Information Networking 2013 (ICOIN)* (pp. 578-582). IEEE Publishing. https://doi.org/10.1109/ICOIN.2013.6496690

Khan, E., El-Kharashi, M. W., Gebali, F., & Abd-El-Barr, M. (2007). Design and performance analysis of a unified, reconfigurable HMAC-Hash unit. *IEEE Transactions on Circuits and Systems-I: Regular Papers, 54*(12), 2683-2695. https://doi.org/10.1109/TCSI.2007.910539

Kieu-Do-Nguyen, B., Hoang, T. T., Tsukamoto, A., Suzaki, K., & Pham, C. K. (2022). High-performance multi-function HMAC-SHA2 FPGA implementation. In *20th IEEE International Interregional NEWCAS Conference, NEWCAS 2022* (pp. 30-34). IEEE Publishing. https://10.1109/NEWCAS52662.2022.9842174

Lin, L., Chen, K., & Zhong, S. (2017). Enhancing the session security of zen cart based on HMAC-SHA256. *KSII Transactions on Internet and Information Systems, 11*(1), 466-483.

McLoone, M., & McCanny, J. V. (2002). A single-chip IPSec cryptographic processor. In *IEEE Workshop on Signal Processing Systems* (pp. 133-138). IEEE Publishing. https://doi.org/10.1109/SIPS.2002.1049698

Michail, H. E., Kakarountas, A. P., Milidonis, A., & Goutis, C. E. (2004). Efficient implementation of the keyed-hash message authentication code (HMAC) using the SHA-1 hash function. In *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 2004 (ICECS 2004)* (pp. 567-570). IEEE Publishing. https://doi.org/10.1109/ICECS.2004.1399744

Oku, D., Yanagisawa, M., & Togawa, N. (2018). Scan-based side-channel attack against HMAC-256 circuits based on isolating bit-transition groups using scan signatures. *IPSJ Transactions on System LSI Design Methodology, 11*, 16-28. https://doi.org/10.2197/ipsjtsldm.11.16

Pham, H. L., Tran, T. H., Duong Le, V. T., & Nakashima, Y. (2022). A high-efficiency FPGA-based multimode SHA-2 accelerator. *IEEE Access Open Access, 10*, 11830-11845. https://doi.org/10.1109/ACCESS.2022.3146148

Randall, K. N. (1999). *ISCA Guide to Cryptography*. McGraw-Hill.

Ravilla, D., & Putta, C. S. R. (2015a). Routing using trust-based system with SHA-2 authentication. *Procedia Computer Science Open Access, 46*, 1108-1115. https://doi.org/10.1016/j.procs.2015.01.023

Ravilla, D., & Putta, C. S. R. (2015b). Implementation of HMAC-SHA256 algorithm for hybrid routing protocols in MANETs. In *2015 International Conference on Electronic Design, Computer Networks & Automated Verification (EDCAV)* (pp. 154-159). IEEE Publishing. https://doi.org/10.1109/EDCAV.2015.7060558

Rubayya, R. S., & Resmi, R. (2015). Memory optimization of HMAC/SHA-2 encryption. In *2014 First International Conference on Computational Systems and Communications (ICCSC)* (pp. 282-287). IEEE Publishing. https://doi.org/10.1109/COMPSC.2014.7032663

Selimis, G., Sklavos, N., & Koufopavlou, O. (2003). VLSI implementation of the keyed-hash message authentication code for the wireless application protocol. In *10th IEEE International Conference on Electronics, Circuits and Systems, 2003 (ICECS 2003)* (Vol. 1, pp. 24-27). IEEE Publishing. https://doi.org/10.1109/ICECS.2003.1301967

Stallings, W. (1996). *Data & Computer Communications* (6th ed.). Prentice Hall.

Wang, M. Y., Su, C. P., Huang, C. T., & Wu, C. W. (2004). An HMAC processor with integrated SHA- 1 and MD5 algorithm. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753)* (pp. 456-458). IEEE Publishing. https://doi.org/10.1109/ASPDAC.2004.1337618

Yiakoumis, I., Papadonikolakis, M., Michail, H., Kakarountas, A. P., & Goutis, C. E. (2005). Efficient small-sized implementation of the Keyed-Hash message authentication code. In *EUROCON 2005-The International Conference on" Computer as a Tool"* (Vol. 2, pp. 1875-1878). IEEE Publishing. https://doi.org/10.1109/EURCON.2005.1630347