Pertanika J. Sci. & Technol. 30 (1): 581 - 603 (2022)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

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

FPGA-based Implementation of SHA-256 with Improvement of Throughput using Unfolding Transformation

Shamsiah Suhaili*and Norhuzaimin Julai

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

ABSTRACT

Security has grown in importance as a study issue in recent years. Several cryptographic algorithms have been created to increase the performance of these information-protecting methods. One of the cryptography categories is a hash function. This paper proposes the implementation of the SHA-256 (Secure Hash Algorithm-256) hash function. The unfolding transformation approach was presented in this study to enhance the throughput of the SHA-256 design. The unfolding method is employed in the hash function by producing the hash value output based on modifying the SHA-256 structure. In this unfolding method, SHA-256 decreases the number of clock cycles required for traditional architecture by a factor of two, from 64 to 34 because of the delay. To put it another way, one cycle of the SHA-256 design can generate up to four parallel inputs for the output. As a result, the throughput of the SHA-256 design can be improved by reducing the number of cycles by 16 cycles. ModelSim was used to validate the output simulations created in Verilog code. The SHA-256 hash function factor four hardware implementation was successfully tested using the Altera DE2-115 FPGA board. According to timing simulation findings, the suggested unfolding hash function with factor four provides the most significant throughput of around 4196.30 Mbps. In contrast, the suggested unfolding with factor two surpassed the classic SHA-256 design in terms of maximum frequency. As a result, the throughput of SHA-256 increases 13.7% compared to unfolding factor two and 58.1% improvement from the conventional design of SHA-256 design.

Keywords: FPGA implementation, SHA-256, throughput, unfolding technique

ARTICLE INFO

Article history:
Received: 1 July 2021
Accepted: 15 September 2021
Published: 10 January 2022

DOI: https://doi.org/10.47836/pjst.30.1.32

E-mail addresses:
sushamsiah@unimas.my (Shamsiah Suhaili)
jnorhuza@unimas.my (Norhuzaimin Julai)
*Corresponding author

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

INTRODUCTION

Cryptography is the study of encrypting messages such that only the intended recipient may read them. Cryptographic algorithms are divided into symmetric cryptography, asymmetric cryptography, and hash functions. Asymmetric cryptography employs two separate keys to encrypt and decrypt the message, whereas symmetric cryptography utilizes only one key. The SHA-256 design was the subject of this investigation with no key. Instead, the hash value of a variable-length message was converted to a fixed-length text hash value. Hash functions include various types such as SHA (Secure Hash Algorithm) family, MD5 (Message Digest 5), SHA-1 (Secure Hash Algorithm 1), RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest 160), and other types.

The hash value results from a hash function where the input depends on the user, which leads to output for a specific algorithm. The hash code is only obtained once the one-way property receives the message input. It is impossible to reverse the process of obtaining the message from the hash code. When the output hash codes M1 and M2 are identical in the second preimage, finding the message M2 is challenging. Finally, collision resistance occurs when two separate messages and two hash function messages digest. Finding the same hash code H(M1) = H(M2) with two different messages is tough.

One of the approaches for obtaining a new program that runs more than one iteration of the original program is to utilize an unfolding algorithm. The unfolding factor defines the number of iterations in the original program. This methodology is used to increase the performance of the SHA-256 design (Suhaili & Watanabe, 2017). The architecture is referred to as Register Transfer Level, and this strategy focuses on the latency of the designs (RTL). Unfolding transformation factors 2 and 4 were implemented in this paper to minimize the latencies of the SHA-256 hash function. (Parhi, 1999). Both designs were carried out in parallel. The area, on the other hand, grew dramatically. Much research has been done related to SHA-256 using both ASIC and FPGA implementation (Shahid et al., 2011; Sun et al., 2007; Sklavos & Koufopavlou, 2003; Miao et al., 2009; Mestiri et al., 2015; Chaves et al., 2006; McEvoy et al., 2006; Ahmad & Das, 2005; Padhi & Chaudri, 2017; Kahri et al., 2015; Michail et al.,2010; Michail et al., 2005; Phan et al., 2021; Kester & Henry, 2019; Bensalem et al., 2021; He et al., 2018; Zhang et al., 2019; Wu et al., 2020; Li et al., 2019; Li et al., 2020; Brazhinikov, 2020; Chen & Li, 2020).

The inner pipelining with the unfolding of SHA-256 hash functions were designed in this study. Based on Arria II GX, these algorithms were synthesized and implemented. ModelSim was used to verify the simulation results. The following is a breakdown of the structure of the paper: Section 2 presents the proposed SHA-256 design. The implementation results are detailed in Section 3, along with a comparison of alternative SHA-256 solutions. The conclusions are discussed in the final part.

RELATED WORKS

SHA-2 hash function consists of four different hash functions such as SHA-224, SHA-256, SHA-384, and SHA-512. The output length of these hash algorithms depends on the SHA-2 size ranging from 224 to 512-bit. This paper only focused on the SHA-256 algorithm because of the extension from the SHA-1 algorithm. In addition, previous SHA-256 was implemented using different types of FPGA devices.

Miao et al. (2009) designed and implemented SHA-1, SHA- 224, SHA-256, SHA-384, and SHA-512 cryptographic chip on EP2S15F484C3 of Stratix II. These designs were written using Verilog code. Modelsim 6.0c simulator tool was used to simulate and verify the design. From this result, SHA-256 provided 143.16 MHz of maximum frequency and 909.8 Mbps of throughput. In this design, Carry Save Adder (CSA) and Carry Lookahead Adder were implemented into the SHA-256 design to increase execution speed (Miao et al., 2009). Sklavos and Koufopavlou (2003) proposed SHA-256 designs with 83 MHz of maximum frequency and 326 Mbps of throughput. This design was implemented on Xilinx Virtex v200pq240. From this result, the maximum frequency of the SHA-256 algorithm slightly decreases compared with the previous design. This design can be applied efficiently to implement digital signature algorithms, keyed-hash message authentication codes, and random numbers generators architectures (Sklavos & Koufopavlou, 2003).

Ahmad and Das (2005) proposed two types of SHA-2, such as SHA-256 and SHA-512 algorithms. These algorithms were designed and tested based on the Altera Quartus II CAD tool. This paper does not mention which specific programming is used to design and implement the SHA-256 algorithms. Overall, the designs were analyzed and synthesized using Verilog HDL and VHDL, placed and routed in Altera devices of APEX II, Stratix, and Mercury family FPGAs. The maximum frequency of SHA-256 designs was 41.97 MHz (Ahmad & Das, 2005). The maximum frequency of the SHA-256 algorithm decreased significantly compared with other designs.

McEvoy et al. (2006) designed six SHA processors using VHDL and implemented them on Xilinx Virtex II xc2v2000-bf957. For SHA-256, one processor had a basic quasi-pipelined core, one had a 2x-unrolled core, and another had a 4x-unrolled core (McEvoy et al., 2006). These techniques have been proposed to speed up the calculations in the SHA core; Carry Save Adder (CSA), unrolling, quasi-pipelining, which use register break the long critical path within the SHA core, Block RAM, and parallel counter. From the results, unrolled SHA-256 provided low latency compared with the basic design. However, the basic SHA-256 design gave 133.06 MHz of the maximum frequency with 1009 Mbps of throughput.

Chaves et al. (2006) has improved the performance of SHA-2 algorithms. SHA-256 was implemented on Xilinx VIRTEX II Pro (XC2VP30-7) and provided 174 MHz of maximum frequency. This design proposed a rescheduling technique that efficiently uses

a pipelined structure without increasing area and hardware reutilization methods to enable resource-saving. The results showed that the maximum frequency of SHA-256 was 174 MHz with 1370 Mbps of throughput (Chaves et al., 2006). Sun et al. (2007) proposed SHA-2 (256, 384, 512) designs, and the algorithms were written using Verilog code. These designs were simulated and verified using ModelSim 6.0a. The target FPGA device for these designs was the same as the device in Sklavos paper, Xilinx Virtex v200pq240-6. The method of SHA-256 was based on Carry Lookahead Adder (CLA) and Carry Save Adder (CSA). This technique tried to improve the critical path of the design. However, the results showed that the maximum frequency of SHA-256 decreased significantly and only gave 74 MHz with 291 Mbps of throughput (Sun et al., 2007).

Shahid et al. (2011) introduced different hash function algorithms modeled using VHDL. Xilinx and Altera Quartus II were used to synthesize and implement these designs. The designs were divided into: basic and embedded with DSP units and block RAMs. The use of embedded FPGAs resources in the implementation of SHA-2 provided high frequency compared with basic designs. The maximum frequency of SHA-2 increased significantly on Virtex 5 and Stratix III with 218.2 MHz and 205.8 MHz, respectively (Shahid et al., 2011). Kahri et al. (2015) focused on both SHA-256 and SHA-512 designs. The designs showed the results based on a finite state machine with a padded process unit. The FSM performed five states: pad 0, pad 1, pad 2, pad 3, and pad F. The SHA-256 design gave 202.54 MHz of maximum frequency 1.58 Gbps of throughput (Kahri et al., 2015).

Padhi and Chaudhari (2017) designed the optimized pipelined architecture of the SHA-256 hash function. The design has been implemented on Xilinx Virtex-4 FPGA using Verilog HDL code. In this design, CSA was used to enhance the performance of architecture. The maximum frequency of the design was 170.75 MHz with a throughput of 1344.98Mbps (Padhi & Chaudhari, 2017). Michail et al. (2005) proposed an SHA-256 design with 64.1 MHz with 2052.1 MHz. This design was implemented on Virtex E using the pre-computation technique to improve design throughput (Michail et al., 2005). Michail et al. (2010) improved the throughput of optimized SHA-256 design using VHDL based on Virtex FPGA implementation. This design uses partially unrolled operation with pre-calculation and pre-computation. Besides, CSA also is applied in this design to increase the throughput of design (Michail et al., 2010).

He et al. (2018) proposed the SHA-256 design using a three-stage pipeline using Cyclone II FPGA implementation. The throughput obtained for the design was 655.66 MHz with a masking scheme (He et al., 2018). Next, Zhang et al. (2019) and Wu et al. (2020) proposed the SHA-256 design with ASIC implementation. Both inventions have been implemented and synthesized with 14nm technology (Zhang et al., 2019; Wu et al., 2020). Finally, Li et al. (2019) proposed an asynchronous SHA-256 implementation design in SMIC 40nm technology. The result of the design was simulated and verified using Synopsys VCS (Li et al., 2019).

Li et al. (2020) proposed an FPGA-based implementation that gave the implementation results with 100 MHz and 787 Mbps of throughput. This design used neural network and blockchain fusion-based image copyright protection and implemented DNN and SHA-256 on FPGA (Li et al., 2020). Brazhnikov (2020) proposed SHA2 with 28nm CMOS technology. This design produced different results by applying different adder types to the SHA-256 design, such as RCA, CLA, Han-Carlson, Brent-Kung, Kogge Stone, and Sklansky, in terms of delay and area (Brazhnikov, 2020). There is no information related to Maximum frequency and throughput in this paper. The performance of SHA-256 in terms of high-throughput was the main objective of this paper. Chen and Li (2020) proposed a rescheduling method to enhance the throughput. This design was implemented on Virtex-4 with a throughput of 1984 MHz with an area of 979 slices (Chen & Li, 2020).

Bensalem et al. (2021) proposed the latest implementation to improve throughput. Bensalem improved the SHA-256 design implementation on FPGA using OpenCL optimization techniques. These optimization techniques include inserting local memories, loop splitting, loop unrolling, and loop pipelining. This design obtained a throughput of 3973 Mbps (Bensalem et al., 2021). These designs were the previous implementation of SHA-256 based on ASIC and FPGA implementation. In this paper, the improvement of throughput of the SHA-256 hash function was designed and implemented on FPGA to improve the performance by using unfolding transformation techniques. Table 1 depicts a simplified version of the prior SHA-256 design and the proposed SHA-256 design.

Table 1

Previous design of SHA-256 algorithm and proposed SHA-256 design

No.	Authors / year	Device	Design Method
1	Miao et al. (2009)	Stratix II–FPGA Implementation	• Iterative CSA and CLA
2	Sklavos and Koufopavlou (2003)	Xilinx Virtex v200pq240–FPGA Implementation	• Iterative
3	Ahmad and Das (2005)	APEX II, Stratix, and Mercury family FPGAs–FPGA Implementation	• Iterative
4	McEvoy et al. (2006)	Xilinx Virtex II xc2v2000-bf957–FPGA Implementation	• CSA Quasi-pipelining
5	Chaves et al. (2006)	Xilinx VIRTEX II Pro (XC2VP30-7)–FPGA Implementation	• Pipelining

Table 1 *(Continue)*

No.	Authors / year	Device	Design Method
6	Sun et al. (2007)	Xilinx Virtex v200pq240-6–FPGA Implementation	• Iterative CSA and CLA
7	Shahid et al. (2011)	Virtex 5 and Stratix III–FPGA Implementation	• Iterative
8	Padhi and Chaudhari (2017)	Xilinx Virtex-4–FPGA Implementation	• Iterative
9	Michail et al. (2005)	Virtex E–FPGA Implementation	• Iterative (pre-computation technique)
10	Michail et al. (2010)	Virtex–FPGA Implementation	• Iterative partially unrolled operation CSA
11	He et al. (2018)	Cyclone II–FPGA Implementation	• Three-stage pipeline
12	Zhang et al. (2019)	ASIC Implementation	• Iterative
13	Wu et al. (2020)	ASIC Implementation	• Iterative
14	Li et al. (2020)	FPGA–FPGA Implementation	• Iterative neural network blockchain fusion
15	Brazhnikov (2020)	28nm CMOS technology	• Iterative RCA, CLA, Han-carlson, Brent-Kung, Kogge Stone and Sklansky
16	Chen and Li (2020)	Virtex-4–FPGA Implementation	• Iterative (rescheduling method)
17	Bensalem (2021)	FPGA Implementation ASIC Implementation	• OpenCL optimization techniques loop splitting, loop unrolling, and loop pipelining
18	Proposed SHA-256 Design	Arria II GX–FPGA Implementation	• Iterative
19	Proposed SHA-256 Unfolding Design (factor 2)	Arria II GX–FPGA Implementation	• Unfolding factor two
19	Proposed SHA-256 Unfolding Design (factor 4)	Arria II GX and Cyclone IV--FPGA Implementation	• Unfolding factor four

MATERIALS AND METHODS

These designs were developed to improve throughput performance. SHA-256 was created using Verilog code. The counter SHA-256 module and the other five modules are modules inside this design architecture. The structure of the modules inside the SHA-256 unfolding design is the distinction between two different types of SHA-256 designs. The sequence of constants and messages was identified to be altered when other inputs were used. For input, 15 blocks of 32-bit data were added. Equation 1 was used in this design for the message, W_t. SHA-256 message, Wt

$$W_t = \text{message input} \qquad 0 \le t \le 15$$
$$W_t = \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16} \qquad 16 \le t \le 63 \qquad [1]$$

Where,

$$\sigma_0^{256}(x) = ROTR^7(x) + ROTR^{18}(x) + SHR^3(x) \qquad [2]$$
$$\sigma_1^{256}(x) = ROTR^{17}(x) + ROTR^{19}(x) + SHR^{10}(x) \qquad [3]$$

Equations 2 and 3 were used to generate both functions. The rotation value for x value is shown in Equation 2 for sigma_0. While for Equation 3, sigma 1 can be obtained by rotating the message x with the value given in Equation 3. It was divided into two portions for the compression function, $Temp_1$ and $Temp_2$. Ch and Σ_1 make up $Temp_1$, while maj and Σ_0 make up $Temp_2$. The equations for summation 0 and summation 1, namely Σ_0 and Σ_1 are shown in Equations 4 and 5. The number of rotation inputs a and e will be based on the number specified in the equations, just like sigma 0 and sigma 1.

$$\sum_0(a) = ROTR^2(a) + ROTR^{13}(a) + ROTR^{22}(a) \qquad [4]$$
$$\sum_1(e) = ROTR^6(e) + ROTR^{11}(e) + ROTR^{25}(e) \qquad [5]$$

The message sequence was generated using a counter module. The final module was developed after finishing all rounds of iteration by the SHA-256 hash algorithm. Before SHA-256 began processing the message, a Multiplexer module assisted in generating eight buffer initializations. 64X32-bit ROM blocks were used to define the constant Kt. Finally, the output module was used to create the SHA-256 message digest. In this model, the final output of the SHA-256 compression function was combined with buffer initialization.

Modifications must be made to each module to improve the performance of the throughput SHA-256 design. For example, two 32-bit parallel inputs with constants were required for the factor two design. Similarly, four parallel 32-bit inputs and four parallel constants were needed in this design. As a result, all information for the following sequence cycle must be changed. Each of the inputs modules has to be changed to achieve this

method. Figure 1 shows the flowchart for this design. It starts with a compilation of RTL designs. Then, the design was evaluated using both functional and timing simulation with both design and testbench file of SHA-256 hash function algorithm before download to FPGA hardware design.

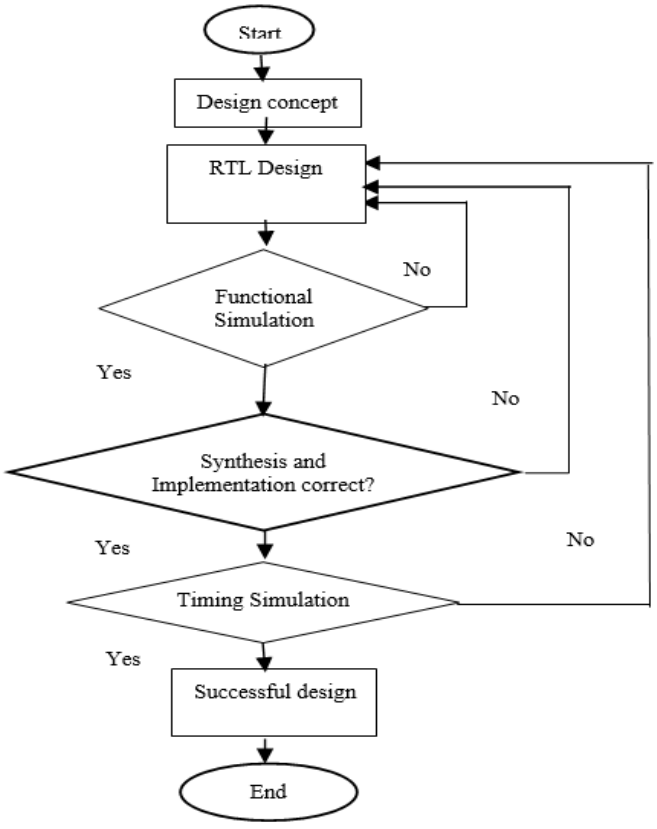

Figure 1. SHA-256 flowchart design

The SHA-256 hash function was improved as a result of these changes. The latency is shortened dependent on the factor J using an unfolding design methodology (Parhi, 1999). Furthermore, this method improves the throughput of the design. For unfolding factor two, the latency reduces to 46.4% from the traditional design, and for unfolding factor four, the percentage reduces around 45.1%. The number of latencies was calculated final results. It was decreased as the design architecture unfolded and changed in response to varied inputs.

In addition, based on modification on modules inside the design, the frequency performance of the design has increased. Compared to the usual design, the frequency of unfolding design rose dramatically with factor two. Compared to two other designs, the

modification of the unfolding method had a large area implementation. However, because of the short latencies, it allowed for high throughput.

The SHA-256 algorithm adjusted message schedule and compression function created the unfolding factor 2 and 4 architecture. This paper used the unfolding technique with factors two and four. Modifications to two modules must be considered. Therefore, it is necessary to consider the alterations to these two modules in the design. The block diagrams of $Temp_{10}$ and $Temp_{20}$ are shown in Figures 2 and 3. The following block diagrams and $\sum 1o$ show how they differed from traditional $Temp_1$ and $Temp_2$. The output of the unfolding design was remade in a different order, with different results. These equations have the SHA-256 algorithm compression function added to them. Σ_{10}, $Cho(next\_e, e, f)$, Message, W_{t_1} and Constant, K_{t_1} are found in $Temp_{1o}$, whereas $\sum 0o$ and $Majo(next\_a, a, b)$ are found in $Temp_{2o}$. A 32-bit adder was used to achieve these results. The $Temp_{1o}$ and $Temp_{2o}$ block diagram design has a different set of data inputs.

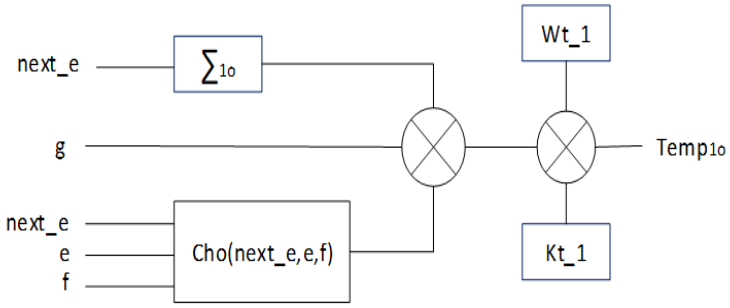

Figure 2. Architecture of $Temp_{10}$ block diagram

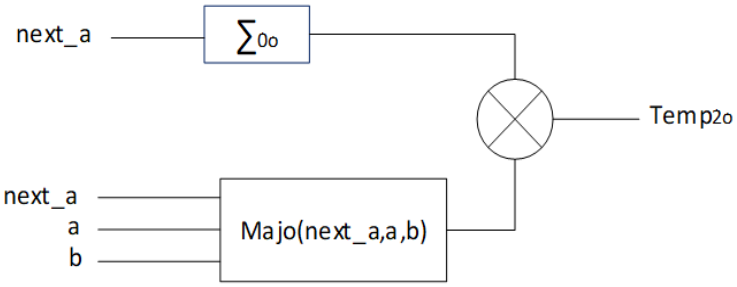

Figure 3. Architecture of $Temp_{10}$ block diagram

The two architectures inside $Temp_{10}$ and $Temp_{20}$ are shown in Figures 4 and 5. The different types of gates are used in both architectures with different location topologies. Both Figures 4 and 5 show that the data inputs differ from the standard function for Cho and Majo. From Figures 4 and 5, it is clearly shown that the new data inputs are applied.

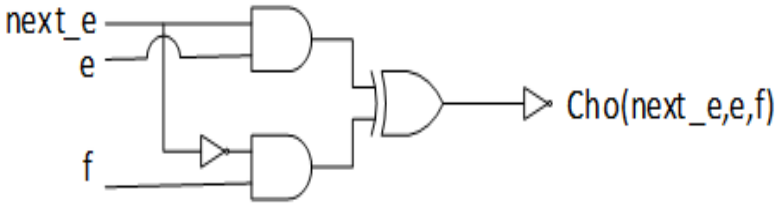

Figure 4. Architectures of Cho Function

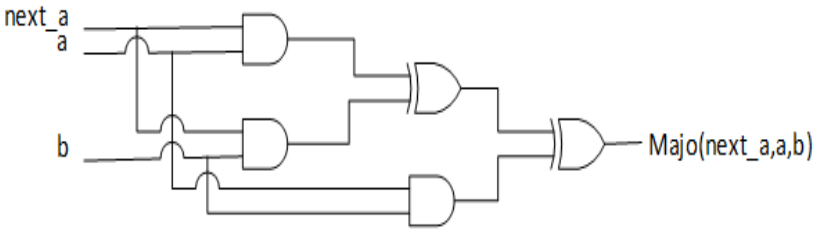

Figure 5. Architectures of Majo Function

The proposed block diagram for summation_0o and summation_1o are shown in Figures 6 and 7, respectively. The input next_a was used to represent, whereas next_e was used to represent. With a fixed number of values, all rotations in both designs followed the right direction. Finally, using an XOR gate to combine all inputs, the final outputs $\sum 0o$ and $\sum 1o$ were achieved.

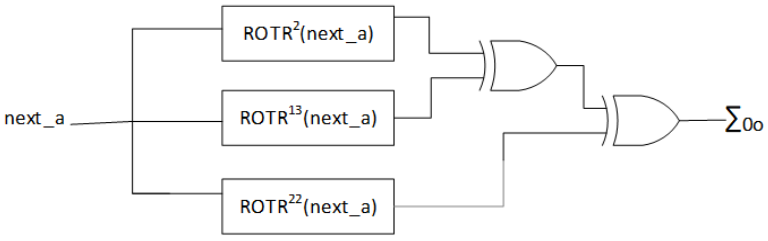

Figure 6. Architecture of $\Sigma_{1o}(next\_e)$

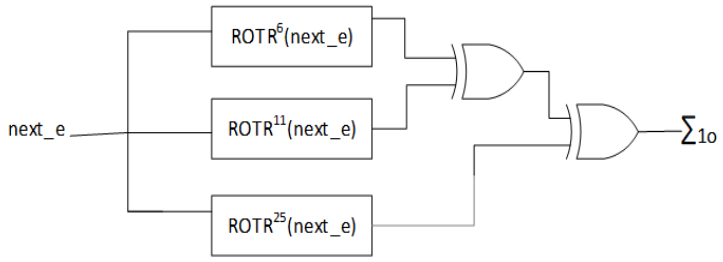

Figure 7. Architecture of $\Sigma 1_o(next\_e)$

New next_eo and next_ao were calculated using output $Temp_{1o}$ and $Temp_{2o}$. Equations 6 and 7 below show the new output value signal.

$$next\_eo = c + Temp_{1o} \quad [6]$$

$$next\_ao = Temp_{1o} + Temp_{2o} \quad [7]$$

Figures 8 and 9 illustrate the revised inner architecture for Temp11 and Temp21. These two inputs, Ch1 and Maj1 from Figures 8 and 9, are all presented with distinct signals. It is due to factor 4 of the unfolding technique. Similarly, function Σ11 and Σ01 also use new input to be applied in the new architecture of $Temp_1$ and $Temp_2$. The new equations can be derived from Equations 8 and 9 below by applying the new inputs signal to both equations.

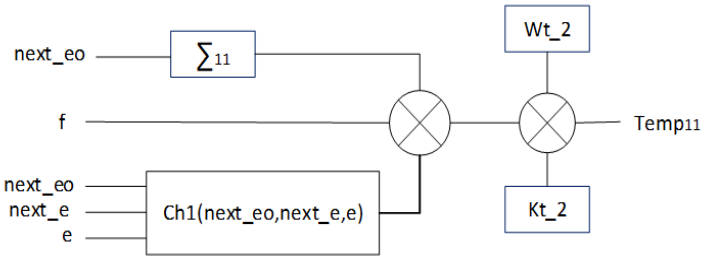

Figure 8. Architecture of $Temp_{11}$ block diagram

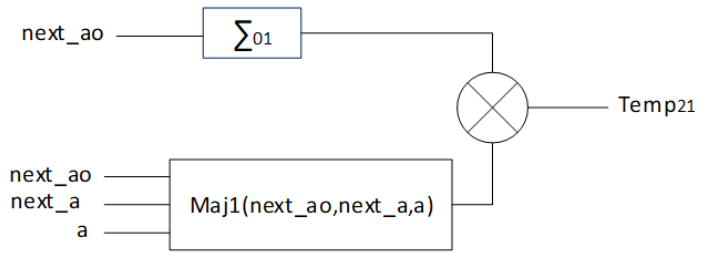

Figure 9. Architecture of $Temp_{21}$ block diagram

The unfolding factor four was calculated until $Temp_{12}$ and $Temp_{22}$ since it required four parallel executions.

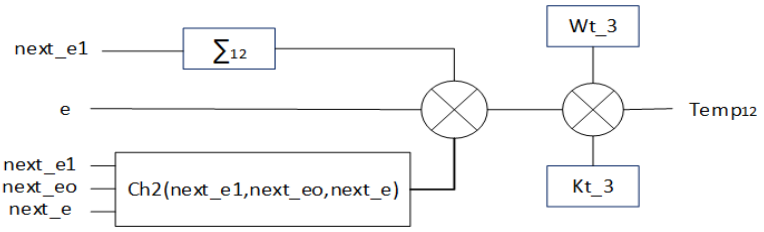

Figure 10. Architecture of $Temp_{12}$ block diagram

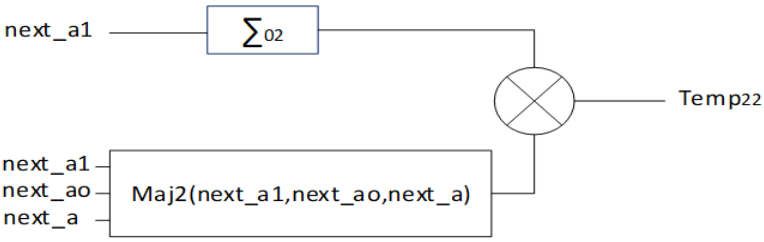

Figure 11. Architecture of $Temp_{22}$ block diagram

Figures 10 and 11 illustrate the new output for both Temp functions. In both function Σ12 and Σ02, two new signal inputs are employed. In addition, the data was relocated to the same place in the input sequence as the one before it. Both Figures 10 and 11 demonstrate these data inputs.

The new signals for Equations 10 and 11 were derived from the $Temp_{12}$ and $Temp_{22}$ datasets. The output of next_e2 and next_a2 are shown in the equation below.

$$\text{next\_e2} = a + Temp_{12} \quad [10]$$

$$\text{next\_a2} = Temp_{12} + Temp_{22} \quad [11]$$

The message schedule was modified from prior results in the same way the compression algorithm was. The modification of the previous equation for sigma0 and sigma1 was processed after receiving the signal. The start of this sequence was at wt2 and concluded at wt15.

Figures 12 and 13 depict the architectures for the σ_{0o} and σ_{1o} functions, respectively. The fundamental role of these architectures is to generate the SHA-256 message schedule. A constant quantity of value was used to rotate W_2 in the appropriate direction for σ_{0o}, whereas for σ_{1o}, new data input was used. The W_2 was right-shifted with a specific value in σ_{0o} function, and similarly with the W_{15}. It was right-shifted in σ_{1o} function with a certain value.

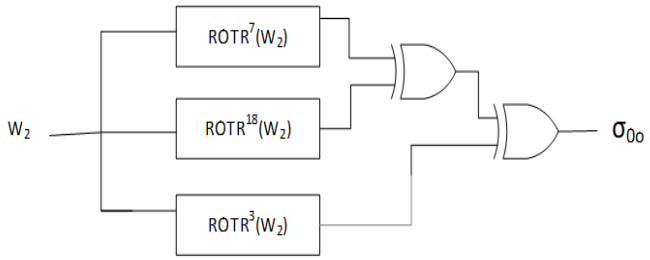

Figure 12. Architecture of σ_{0o}

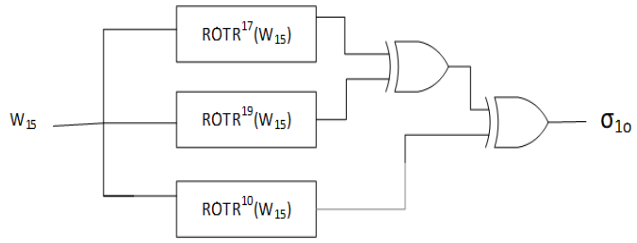

Figure 13. Architecture of σ_{1o}

With input W_3 and next wt, σ_{01} and σ_{11}were computed for unfolding factor 4. The following Equation 12 was used to calculate next_wt1.

$$\text{next\_wt1} = W_2 + \sigma_{01} + W_{11} + \sigma_{11} \quad [12]$$

Finally, input W_4 and next_wto were used to generate the input for σ_{02} and σ_{12}. Figure 13 was used to create the next_wto. Figure 14 depicts the structure of next_ wt2 and the overall design of unfolding factor 4 for the message schedule. Message input of Wo data began with W_4 and ended with W_{15}. The output sequence of next_wt used in unfolding factor four uses the similar method used in factor two.

Figure 15 illustrates an SHA-256 hash function with an FPGA implementation design. In this phase, functional simulation is used to check the results of the invention. First, Verilog code needs to be converted into gate-level based on an FPGA family device chosen in the early phase of the design. Then, the compilation and synthesis process will be executed to translate the Verilog code into a netlist to represent the actual hardware device. Logic synthesis tools play important roles in digital electronic design automation. After the synthesis process, timing simulation needs to be evaluated in terms of time setup and time hold of the output waveform. Finally, the design can be downloaded to the FPGA device.

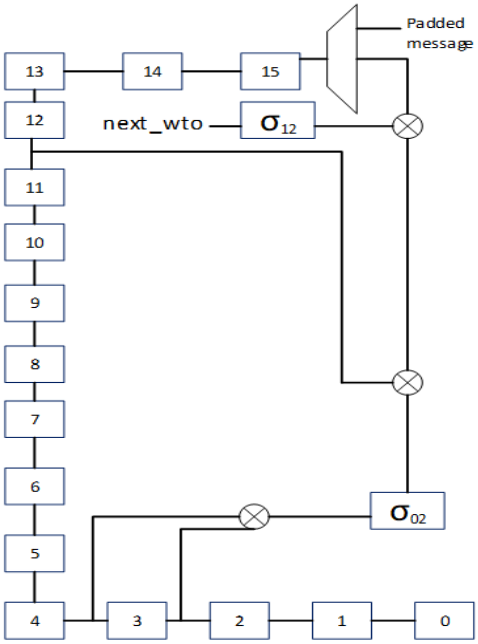

Figure 14. SHA-256 unfolding factor 4 message schedule

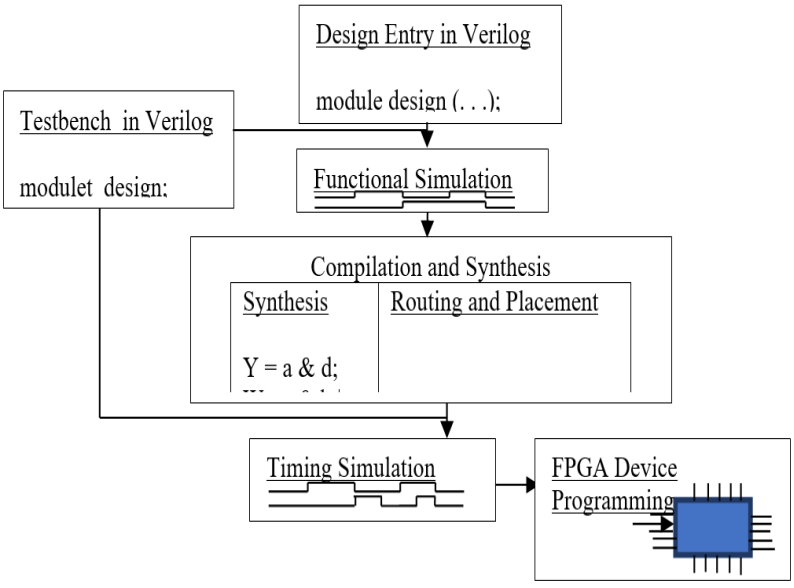

Figure 15. FPGA implementation flow of SHA-256 design

RESULTS AND DISCUSSION

The proposed SHA-256 unfolding factors two and four have been successfully developed and tested. The Altera Quartus II was used to compile all of the designs written in Verilog code. The functionality of the design was simulated and verified using ModelSim in terms of timing and functional simulation. Finally, the throughput of these designs was calculated using Equation 13.

$$\text{Throughput} = (512 \text{ X FMax}) / \text{Number of Cycle} \quad [13]$$

Table 2 shows a comparison of the proposed SHA-256 designs with previous publications. The design throughput increased significantly by using this unfolding method compared to previous design configurations. Most of the previous SHA-256 algorithms were applied the iterative and pipelining method to design the hash function. Therefore, the area implementation of the designs was smaller than the unfolding method. The unfolding method employs a parallel operation based on the number of unfolding factors. According to the data in Table 2, the unfolding SHA-256 design with factor four design had the highest throughput of all the SHA-256 designs.

Table 2 summarizes the findings of other designs. Iterative and pipelining approaches were applied to the prior SHA-256 algorithm using this table. Different adders, such as CSA and CLA, were applied to the iterative process in the preceding SHA-256 design. Unrolled pipelining was employed in the SHA-256 design of the pipelining method. As a result, three types of SHA-256 algorithms have been proposed to increase the throughput of SHA-256 design: iterative SHA-256 design, unfolding SHA-256 with factor two, and unfolding SHA-256 with factor four. When an unfolding transformation is applied to the SHA-256 algorithm, the design operation improves dramatically because the design is handled in parallel. By minimizing the number of latencies, this strategy can assist in enhancing the performance of SHA-256 designs. Different devices produced different results when the SHA-256 design was implemented. As a result, better results can be obtained by selecting an appropriate family device for implementation.

Table 2

Other SHA-256 design results in synthesis and implementation comparison

SHA-256 Hash Function	Area	Device	FMax	Throughput
Proposed SHA-256 Design	855 ALUTs	Arria II GX	228.15 MHz	1756.58 Mbps

Table 2 *(Continue)*

SHA-256 Hash Function	Area	Device	FMax	Throughput
Proposed SHA-256 unfolding design factor 2	1345 ALUTs	Arria II GX	251.07 MHz	3621.07 Mbps
Proposed SHA-256 unfolding design factor 4	2064 ALUTs	Arria II GX	159.82 MHz	4196.30 Mbps
SHA-2 (Shahid et al., 2011)	320 CLBs	Virtex 5	218.2 MHz	1719 Mbps
SHA-2 (Shahid et al., 2011)	795 ALUTs	Stratix III	205.8 MHz	1621 Mbps
SHA(256,384,512) (Sun et al., 2007)	2207 CLBs	Virtex v200pq 240-6	74 MHz	291 Mbps
SHA-256 (Sklavos & Koufopavlou, 2003)	1060 CLBs	Virtex v200pq240	83 MHz	326 Mbps
SHA-256 (Miao et al., 2009)	2150 ALUTs	Stratix II	143.164 MHz	909.816 Mbps
SHA-256 (Mestiri et al., 2015)	387 Slices	Virtex 5 XC5VFX70T	202.54 MHz	1580 Mbps
SHA-256 (Chaves et al., 2006)	755 Slices	XC2PV-7	174 MHz	1370 Mbps
SHA-256 (MeEvoy et al., 2006)	1373 Slices	Virtex-II xc2v2000-bf957	133.06 MHz	1009 Mbps
SHA-256 (Ahmad & Das, 2005)	-	-	41.97 MHz	335.9 Mbps
SHA-256 (Padhi & Chaudhari, 2017)	610 Slices	Virtex-4	170.75 MHz	1344.98 Mbps
SHA-256 (Kahri et al., 2015)	387 Slices	Virtex-5 XC5VFX70T	202.54 MHz	1580 Mbps
SHA-256 (Michail et al., 2010)	1534 CLBs	Virtex	35.1 MHz	2077 Mbps
SHA-256 (Michail et al., 2010)	1655 CLBs	Virtex E	36.4 MHz	2190 Mbps
SHA-256 (Michail et al., 2005)	-	Virtex E	64.1 MHz	2052.1 Mbps
SHA-256 (Phan et al., 2021)	1895 slice/ LUT	Virtex 5	411.3 MHz	3290.4 Mbps
SHA-256 (Bensalem et al., 2021)	67150 LU	Arria10	243 MHz	3970 Mbps

Table 2 *(Continue)*

SHA-256 Hash Function	Area	Device	FMax	Throughput
SHA-256 (He et al., 2018)	7219 Cells	Cyclone II	116.2 MHz	875.22 Mbps
SHA-256- cascade (Li et al., 2019)	-	SIMC 40nm technology (Synopsys)	227.27 MHz	3600 Mbps
SHA-256 (Li et al., 2020)	-	Alpha-Data ADMPCIE-7V3	100 MHz	787 Mbps
SHA-256 (Chen & Li, 2020)	979 Slices	Virtex 4	255.7 MHz	1984 Mbps

The proposed design used 855 ALUTs and had a maximum clock frequency of 228.15 MHz. Thus, the performance of the proposed design was improved significantly by using this technique. According to the results, the proposed design had the highest throughput, with 4196.30 Mbps and a maximum frequency of 159.82 MHz due to the internal pipelining design, which used 1159 registers. Compared to traditional architecture, this technique enhanced the design by eliminating round cycles. As a result, the clock cycle count of the SHA-256 unfolding architecture dropped from 66.5 to 19.5 cycles. The hash function design of SHA-256 with excellent throughput was accomplished using the unfolding transformation approach.

The proposed unfolding SHA-256 design can improve the performance of the hash function design. The design frequency can be considerably increased by employing the unfolding method and following the criteria for developing better HDL coding. In addition, the architecture of the FPGA device plays a vital part in the SHA-256 design. Thus, the performance of the SHA-256 design can be improved by identifying the appropriate FPGA device. The earlier implementation of the SHA-256 design is shown in Table 2. Due to budget and device constraints, it is difficult to locate the same device for the same design. The area implementation of the design increases from iterative design to unfolding design, as shown in this table. The throughput of the SHA-256 unfolding with factor four design, on the other hand, greatly increases. SHA-256 improves throughput by 13.7 percent compared to unfolding factor 2 and by 58.1 percent compared to the traditional SHA-256 design. Figure 16 shows the timing simulation results for the conventional method of SHA-256 design with 64 cycles to generate the final output result. The output of SHA-256 hash function represents by eight 32-bit signal output of {Ha0, Ha1, Ha2, Ha3, Ha4, Ha5, Ha6, Ha7} with the value output of "ba7816bf8f01cfea41410de5dae2223b0036177a9cb410ff61f20015ad".

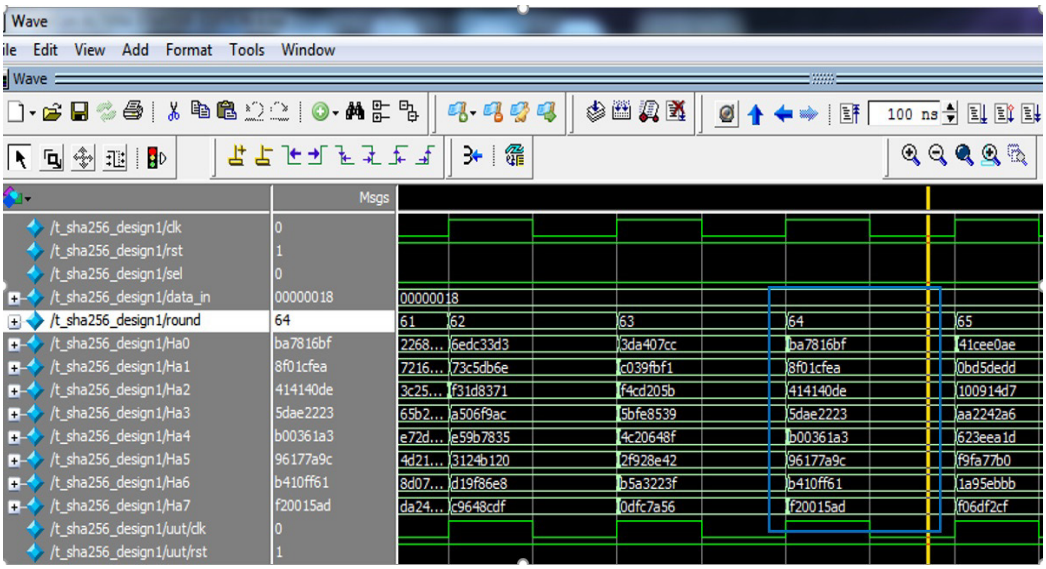

Figure 16. Timing simulation waveform of SHA-256 design

According to Equation 13, the number of cycles for unfolding SHA-256 with factor two is lowered by two, from 64 to 32, as shown in Figure 17. Due to the clock delay in the timing simulation, the number of cycles given in this simulation waveform was 34. Then there will be an increase in throughput of the SHA-256 design.

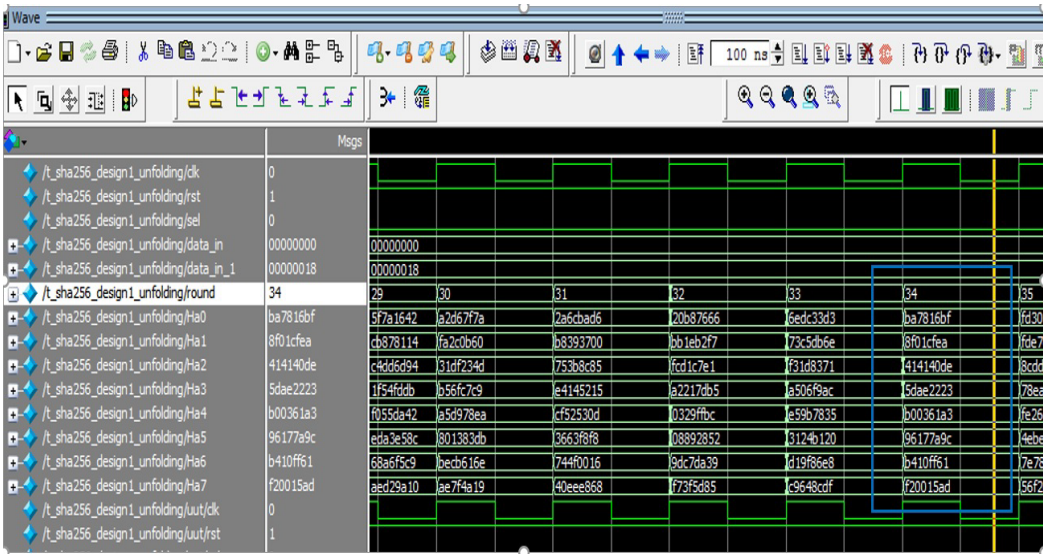

Figure 17. Timing simulation waveform of unfolding SHA-256 with factor two design

Similarly, the cycles of unfolding SHA-256 factor four will decrease by four, bringing the total number of cycles down from 64 to 18. The throughput and performance of the SHA-256 design can benefit from a minimal number of cycles. Figure 18 shows the timing simulation results for unfolding SHA-256 with factor four. Because of the concurrent processing that limits the input-output (I/O) FPGA pads, only 32-bit MSB final results are given in Figure 19(a).

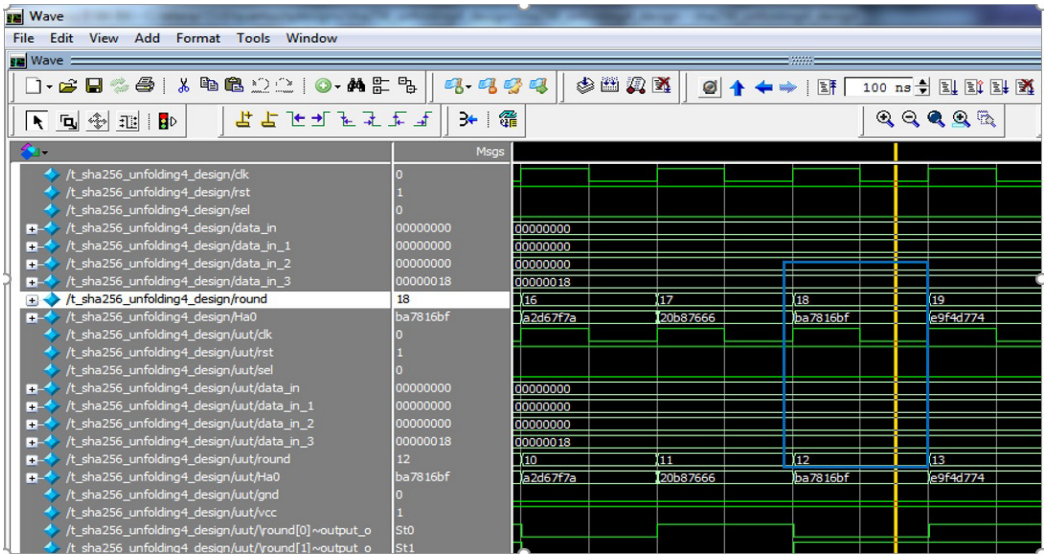

Figure 18. Timing simulation waveform of unfolding SHA-256 with factor four design

The unfolding SHA-256 with factor four design has been downloaded into Cyclone IV E: EP4CE115F29C7 FPGA family device to verify the hardware implementation. The SHA-256 input in this particular instance is the text "abc." As a result, the text output of the SHA-256 hash function for 256 bits should be "ba7816bf8f01cfea41410de5dae2223b0036177a9cb410ff61f20015ad".

On the Cyclone IV E family device, Figures 19(a) and 19(b) show the hardware implementation of this hash function. The 32-bit MSB part of the output on the target device EP4CE115F29C7 is "ba7816bf." The result was evaluated using seven segments, and three red LED bits, yielding 7 in decimal and "111" in binary. If the result of this hash function provides a correct output of 256 bits, the 3 LED outputs will appear 7 in decimal numbers, which is "111" in binary form.

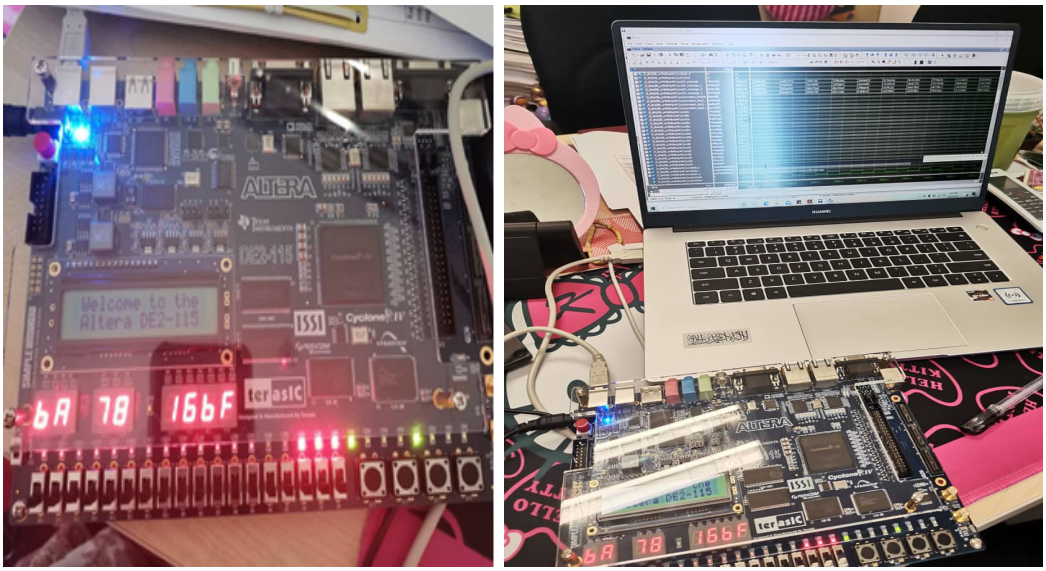

Figure 19. (a) FPGA output implementation display (b) Hardware implementation of SHA-256 hash function

CONCLUSION

The high-performance design of the hash function is essential in security design. By applying this unfolding method with factor four to the SHA-256 hash function, the throughput of the design increase significantly. It is the best solution to improve the performance of the hash function. The maximum frequency of SHA-256 design implementation illustrates the critical path of the design. In order to obtain the high-performance design, the speed needs to be considered, thus leading to the high throughput of the SHA-256 design. ModelSim was used to simulate the SHA-256 design, then tested on an FPGA device for hardware implementation to verify the hash value output using six blocks of seven-segment and 3 LEDs. The proposed SHA-256 unfolding factors two and four designs are comparable to other SHA-256 methods in the area and maximum frequency. From Table 2, based on the throughput of the previous SHA-256 design with iterative and pipelining, the proposed design with unfolding techniques produced the highest throughput of 4196.30 Mbps with factor four. It is clearly shown that by applying this unfolding method, the SHA-256 design improves significantly in terms of design throughput because the number of latencies reduces four times. For conventional design and unfolding factor two, cycles decreases from 64 to 34. The number of cycles drops from 34 to 18 when the unfolding factor increases to four. As a result, when factor four is applied to the Unfolding SHA-256 design, the throughput dramatically increases. The percentage improvement of SHA-256 unfolding factor four was 13.7% and 58.1% compared to SHA-256 unfolding with factor two and conventional design of SHA-256, respectively. Compared to iterative design, the

proposed unfolding design had the drawback of large area implementation. This design, however, can be enhanced by combining pipelining and unfolding techniques to provide the greatest effect. This design is essential for security applications and could be used in future innovations, such as other hash functions, HMAC, and security applications.

ACKNOWLEDGEMENTS

This project was supported by Universiti Malaysia Sarawak (UNIMAS) -SGS grant F02/SGS/1785/2018

REFERENCES

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

Bensalem, H., Blaquière, Y., & Savaria, Y. (2021). Acceleration of the secure hash algorithm-256 (SHA-256) on an FPGA-CPU cluster using OpenCL. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1-5). IEEE Publishing. https://doi.org/10.1109/ISCAS51556.2021.9401197

Brazhnikov, S. (2020). A hardware implementation of the SHA2 hash algorithms using CMOS 28nm technology. In *IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)* (pp. 1784-1786). IEEE Publishing. https://doi.org/10.1109/EIConRus49466.2020.9039083

Chaves, R., Kuzmanov, G., Sousa, L., & Vassiliadis, S. (2006). Improving SHA-2 hardware implementations. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 298-310). Springer. https://doi.org/10.1007/11894063_24

Chen, Y., & Li, S. (2020). A high-throughput hardware implementation of SHA-256 algorithm. In *IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1-4). IEEE Publishing. https://doi.org/10.1109/ISCAS45731.2020.9181065

He, Z., Wu, L., & Zhang, X. (2018). High-speed pipeline design for HMAC of SHA-256 with masking scheme. In *12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)* (pp. 174-178). IEEE Publishing. https://doi.org/ 10.1109/ICASID.2018.8693229

Kahri, F., Mestiri, H., Bouallegue, B., & Machhout, M. (2015). 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 Publishing. https://doi.org/ 10.1109/SSD.2015.7348105

Kester, Q. A., & Henry, B. (2019). A hybrid data logging system using cryptographic hash blocks based on SHA-256 and MD5 for water treatment plant and distribution line. In *2019 International Conference on Cyber Security and Internet of Things (ICSIoT)* (pp. 15-18). IEEE Publishing. https://doi.org/ 10.1109/ICSIoT47925.2019.00009

Li, J., He, Z., & Qin, Y. (2019). Design of asynchronous high throughput SHA-256 hardware accelerator in 40nm CMOS. In *2019 IEEE 13th International Conference on ASIC (ASICON)* (pp. 1-4). IEEE Publishing. https://doi.org/ 10.1109/ASICON47005.2019.8983530

Li, W., Zhu, Y., Tian, L., Nan, T., & Chen, X. (2020). FPGA-based hardware acceleration for image copyright protection syetem based on blockchain. In *7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)* (pp. 234-239). IEEE Publishing. https://doi.org/10.1109/CSCloud-EdgeCom49738.2020.00048

McEvoy, R. P., Crowe, F. M., Murphy, C. C., & Marnane, W. P. (2006). 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. 317-322). IEEE Publishing. https://doi.org/10.1109/ISVLSI.2006.70

Mestiri, H., Kahri, F., Bouallegue, B., & Machhout, M. (2015). Efficient FPGA hardware implementation of secure hash function SHA-2. *International Journal of Computer Network and Information Security, 7*(1), 9-15. https://doi.org/ 10.5815/ijcnis.2015.01.02

Miao, L., Jinfu, X., Xiaohui, Y., & Zhifeng, Y. (2009). Design and implementation of reconfigurable security hash algorithms based on FPGA. In *2009 WASE International Conference on Information Engineering* (pp. 381-384). IEEE Publishing. https://doi.org/ 10.1109/ICIE.2009.278

Michail, H., Athanasiou, G., Kritikakou, A., Goutis, C., Gregoriades, A., & Papadopoulou, V. (2010). 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 Publishing.

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

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

Parhi, K. K. (1999). *VLSI digital signal processing systems: Design and implementation*. John Wiley & Sons.

Phan, V. D., Pham, H. L., Tran, T. H., & Nakashima, Y. (2021). High performance multicore SHA-256 accelerator using fully parallel computation and local memory. In *2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)* (pp. 1-3). IEEE Publishing. https://doi.org/ 10.1109/COOLCHIPS52128.2021.9410349

Shahid, R., Sharif, M. U., Rogawski, M., & Gaj, K. (2011). Use of embedded FPGA resources in implementations of five round three SHA-3 candidates. In *2011 International Conference on Field-Programmable Technology* (pp. 1-9). IEEE Publishing. https://doi.org/10.1109/FPT.2011.6132680

Sklavos, N., & Koufopavlou, O. (2003). 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. 153-156). IEEE Publishing. https://doi.org/ 10.1109/ISCAS.2003.1206214

Suhaili, S., & Watanabe, T. (2017). 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 Publishing. https://doi.org/ 10.1109/ICEEI.2017.8312449

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

Wu, R., Zhang, X., Wang, M., & Wang, L. (2020). A high-performance parallel hardware architecture of SHA-256 hash in ASIC. In *2020 22nd International Conference on Advanced Communication Technology (ICACT)* (pp. 1242-1247). IEEE Publishing. https://doi.org/ 10.23919/ICACT48636.2020.9061457

Zhang, X., Wu, R., Wang, M., & Wang, L. (2019). A high-performance parallel computation hardware architecture in ASIC of SHA-256 hash. In *2019 21st International Conference on Advanced Communication Technology (ICACT)* (pp. 52-55). IEEE Publishing. https://doi.org/ 10.23919/ICACT.2019.8701906.