Journal of Optimization in Industrial Engineering
Vol.15, Issue 1, Winter & Spring 2022, 207-216
DOI:10.22094/JOIE.2021.1941138.1896

Research Article

Throughput Improvement of RIPEMD-160 Design Using Unfolding Transformation Technique

Shamsiah Binti Suhaili [a,*], Takahiro Watanabe [b], Norhuzaimin Julai [c]

[a] *Faculty of Engineering, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia,*

[b] *School of Information, Production and Systems, Waseda University, 2-7 Hibikino, Wakamatsu-ku, Fukuoka 808-0135, Japan*

[c] *Faculty of Engineering, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia*

Received 23 September 2021; Revised 09 November 2021; Accepted 18 November 2021

Abstract

RIPEMD-160 hash functions are widely used in many applications of cryptography such as digital signature, Hash Message Authentication Code (HMAC) and other data security application. There are three proposed RIPEMD-160 design namely RIPEMD-160 iterative design, RIPEMD-160 unfolding with factor two and RIPEMD-160 unfolding design with factor four. These techniques were applied to RIPEMD-160 designs to examine the inner structure of RIPEMD-160 in terms of area, maximum frequency and throughput of the design. In this project, RIPEMD-160 hash function using unfolding transformation technique with factor four provided high throughput implementation. The throughput of the RIPEMD-160 unfolding design increase significantly. The objective of this project is to enhance the performance of RIPEMD-160 in terms of throughput. By using unfolding transformation factor four technique, the throughput of RIPEMD-160 can be improved which is about 1753.50 Mbps. The percentage of performance to area ratio of RIPEMD-160 unfolding with factor four designs increase 1.51% if compared with RIPEMD-160 design. The results show performance of proposed designs give the highest value compare with other designs. The simulation results were obtained from ModelSim Altera-Quartus II to verify the correctness of the RIPEMD-160 designs in terms of functional and timing simulations.

Keywords*: FPGA; Hash Function;* RIPEMD-160; Throughput; Unfolding

1. Introduction

There are different types of hash functions such as SHA-1, MD5, RIPEMD-160, SHA-2 and others (Rodriguez-Henriquez and et al., 2006). Hash function is important for some security application such as Hash Message Authentication Codes (HMAC), digital signature and others. The RIPEMD-160 hash function can also be used in the implementation of cryptocurrency. Cryptocurrency is a digital currency that transfers the coin in blockchain where each block consists of hash of the previous block. Therefore, RIPEMD-160 hash function becomes important especially in recent peer-to-peer electronic cash system like Bitcoin.

Nowadays, the security process of money transaction becomes important aspect. No matter whether it is traditional currency or cryptocurrency transaction. There are lots of problem occur during transaction where the original data is modified by some users. Therefore, the data accuracy is very important, and it needs security design to avoid this problem. 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. Hash function can be used for some applications such as Message Authentication Code (MAC) and digital signature. Therefore, high performance of cryptographic hash function algorithm is one of the important aspects of security algorithm. Hence, designing an efficient implementation of RIPEMD-160 hash function algorithm on reconfigurable hardware needs to be considered. Thus, the implementation of efficient design of hash function needs to be applied to network security (Abu Bakar, Rosbi & Uzaki, 2017). In this project, the design focuses on RIPEMD-160 design. The scope of research for this project is to design and implement the optimized RIPEMD-160 using Verilog HDL which is based on FPGA device. The designs need to be improved to obtain the high performance RIPEMD-160 hash function. Therefore, several types of techniques are applied to this design such as iterative and unfolding transformation. The design must meet the timing requirement where setup and hold time will give the positive value. Thus, there is no violation in the slack from the timing report. This leads to the frequency maximum which is the longest path of the design. The optimized design can be obtained by giving appropriate clock constraint to the RIPEMD-160 design. The RIPEMD-160 design is simulated and tested by using ModelSim with different testbench files. The motivation of this research is to provide the improvement of RIPEMD-160 design. The inner structure of RIPEMD-160 algorithm is different from other hash function. In this design, there are two parallel design for both left and right parts. The complexity of RIPEMD-160 design is more complex in terms of shift and constant value.

*Corresponding author Email address: rai.naveed@mountsafa.com

Hash function is a one-way function which maps the arbitrary inputs and produces fixed length of the output based on the structure of the hash function itself. There is no key for the hash function structure and the output of the hash function is called hash value, hash code and message digests. Nowadays, performance of hash function needs to be taken into consideration since all the designs need to be fast enough through the application design. Therefore, it is necessary to improve the performance of hash function in terms of area, frequency and throughput of the design. In this paper, RIPEMD-160 hash functions with unfolding transformation are proposed. The advantage of unfolding design if compared to the traditional design is it can improve the throughput of RIPEMD-160 design. Besides, it reduces the number of cycles and increase the maximum frequency of the design based on factor chosen for unfolding transformation. This technique will increase the FMax which is frequency maximum of the design. This leads to high throughput of RIPEMD-160 design. Therefore, by using this technique, the improvement of throughput of optimized RIPEMD-160 design will be obtained. This is because of parallel input operation that occurs at one time. Therefore, to overcome this problem, the RIPEMD-160 design can be implemented on devices that provide huge amount of register and ALUTs.

In this design, there are two types of unfolding transformation such as factor two and factor four. Unfolding algorithm is one of the techniques that can be used by the Digital Signal Processing (DSP) application to obtain a new program that performs more than one iteration of the original program. In addition, *unfolding factor, J* describes the number of iterations from the original program. The rules of unfolding algorithm are explained as following (Parhi, 1999).

1. For each node U in the original Data flow graph (DFG), draw the J nodes $U_0, \ldots . U_{\{J-1\}}$.
2. For each edge $U \rightarrow V$ with ω delay in the original, draw the J edges $U_i \rightarrow V_{(i+\omega)\%J}$ with $\left\lceil \frac{i+\omega}{J} \right\rceil$ delays for $i = 0,1 \ldots . , J-1$.

In order to explain the structure of unfolding algorithm, one example of DSP program is shown in Equation 1. Equation (1) shows input *x(n)* and output *y(n)* with 9 delay.

$y(n) = ay(n-9) + x(n)(1)$

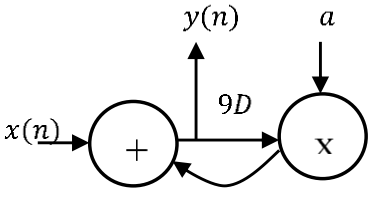

Fig. 1 The example of original DSP program(Parhi, 1999).

DFG can be constructed based on Figure 1, which is an example of the original DSP program by replacing the input and output ports with node A and B, while the addition and multiplication processes are represented by node C and node D respectively as shown in Figure 2.

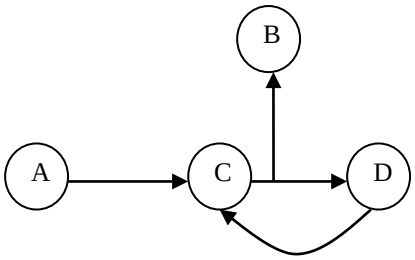

Fig. 2. A DFG corresponding to DSP program [3]

Based on the first rule of unfolding algorithm, there are 8 nodes that represent $i = 0, 1$, namely $A_0, B_0, C_0, D_0, A_1, B_1, C_1,$ and D_1. The second step of unfolding algorithm is to connect each edge $U \rightarrow V$ into the DSP program. The edge $U \rightarrow V$ with no delay is divided into two parts, $U_0 \rightarrow V_0$ and $U_1 \rightarrow V_1$. Therefore, the edge $C \rightarrow D$ with $\omega = 9$ delays becomes $C_0 \rightarrow D_{(9+0)\%2}$ with $\left\lceil \frac{9+0}{2} \right\rceil$ delays and $C_1 \rightarrow D_{(9+1)\%2}$ with $\left\lceil \frac{9+1}{2} \right\rceil$ delays. Figure 3 shows the unfolded DFG corresponding to the 2-unfolded DSP program [4]. Finally, the 2-unfolded DFG is created with $C_0 \rightarrow D_1$ with 4 delays and $C_1 \rightarrow D_0$ with 5 delays respectively.

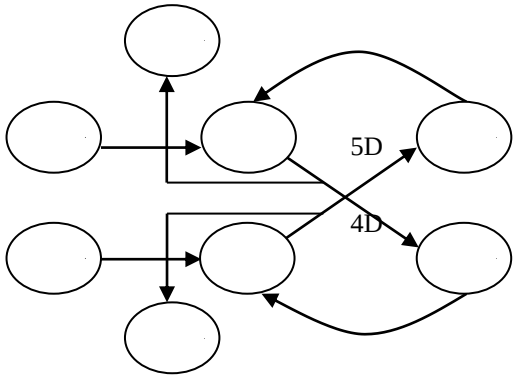

Fig. 3. The unfolded DFG corresponding to the 2-unfolded DSP program (Parhi, 1999).

The structure of RIPEMD-160 algorithm has two sides, left and right. Thus, two parallel structure will operate at the same time during the execution of the RIPEMD-160 designs. There are lots of research and investigation on RIPEMD-160 hash functions that had been done throughout the years [4 – 18]. Most of the RIPEMD-160 designs were using iterative designs. The drawback of iterative design is the performance of hash function in terms of throughput is low. However, (Michail, 2008) introduced the pipelining design of RIPEMD-160. From these two papers, the throughput of RIPEMD-160 design increase significantly. Different FPGA device was used in the implementation of RIPEMD-160 hash function. Implementation of pipelining designs normally use a lots of area implementation. In order to reduce the number of areas, unfolding technique can be applied to RIPEMD-160 hash function. Therefore, by applying unfolding transformation method, the number of cycles can be reduced. By increasing the number of unfolding factors, the performance of RIPEMD-160 will increase where the high throughput of the RIPEMD-160 design will be obtained from unfolding factor two to unfolding factor

four. It can improve the performance of the design significantly in future by combining the pipelining and unfolding techniques to. There was no unfolding RIPEMD-160 design in previous implementation. Implementation of unfolding design for RIPEMD-160 needs to be implemented for both sides of this design. The complexity of this design makes it complicated to be designed. The designers need to analyze the next input for each register. The factor of unfolding design plays an important role in designing unfolding transformation. Unfolding factor four needs more complexity of algorithm if compared to factor two because the number of cycles of the design must be reduced by four. This paper is organized as follows: In the next section, RIPEMD-160 hash function algorithm is introduced. In section 3, proposed architecture of RIPEMD-160 algorithm are presented. Section 4 is results and discussion. Finally, section 5 concludes this paper.

2. RIPEMD-160 Hash Function

2.1. RIPEMD-160 algorithm

RIPEMD-160 stands for Race Integrity Primitives Evaluation Message Digest. It consists of parallel operation which is left and right sides of the inner structures of RIPEMD-160 algorithm. The arbitrary input length for RIPEMD-160 algorithm is mapping to the algorithm to produce 160-bit of hash code. The RIPEMD-160 hash function is one-way function where the process to get the message back from hash code cannot be done inversely. Besides, collision resistance is one of the requirement characteristics of hash function. That means there are two different messages for two different message digests. The output of 160-bit of RIPEMD-160 hash function is in little-endian format. Figure 4 shows the compression function of RIPEMD-160 algorithm. From this figure, there are five inputs from the left namely A, B, C, D and E and five inputs from the right namely A', B', C', D' and E'. The output D and D' shift (rotation) to the left over 10 positions. Other shift rotations are based on the given shift value. In this compression function, there are non-linear function for input B, C and D. See Fig. 4.

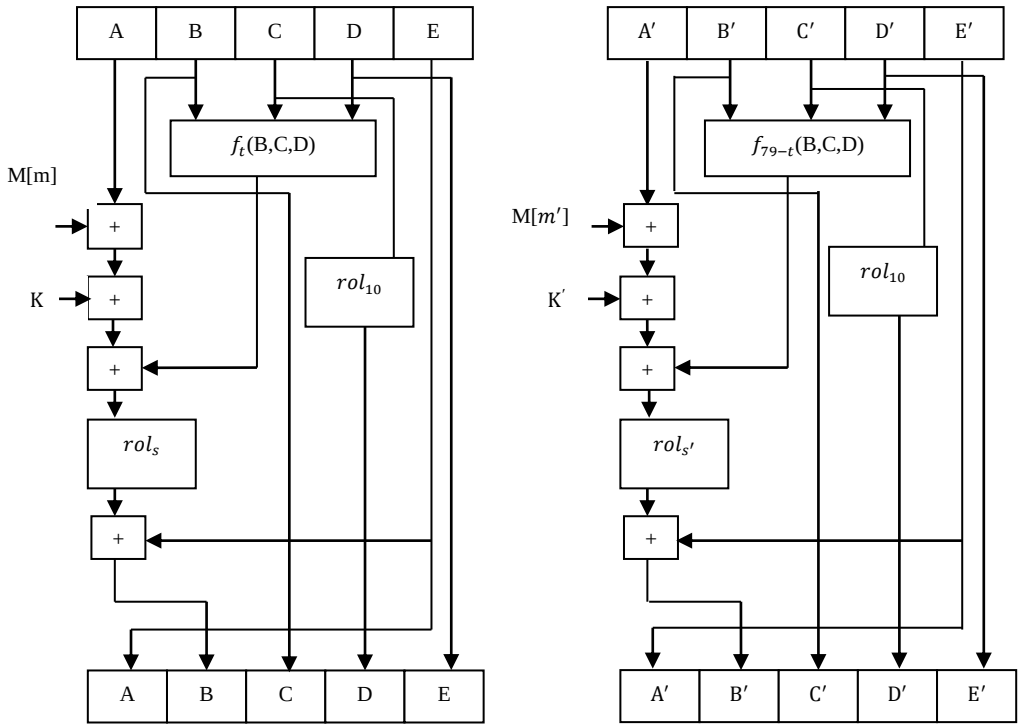

Fig. 4. Compression function of RIPEMD-160 algorithm

From Fig. 4, the output for left part, A, B, C, D, and E and right part A', B', C', D' and E' for t = 0 to 79 can be written as the following equations.

1. $T = rol_{s(t)}\big(A + f_t(B,C,D) + M[m(t)] + K(t)\big) + E$ 2. (2)
3. 4.
5. $A = E$ 6. (3)
7. $E = D$ 8. (4)
9. $D = rol_{10}(C)$ 10. (5)
11. $C = B$ 12. (6)
13. $B = T$ 14. (7)
15. $T' = rol_{s'(t)}\big(A' + f_{79-t}(B',C',D') + M[m'(t)] + K(t)\big) + E'$ 16. (8)
17. $A' = E'$ 18. (9)
19. $E' = D'$ 20. (10)
21. $D' = rol_{10}(C')$ 22. (11)
23. $C' = B'$ 24. (12)
25. $B' = T'$ 26. (13)

Equation (2) and (8) show the equation for T. These equations represent rotation, $rol_{s(t)}$ with the specific fixed shift value, s of non-linear function, $f_t(B,C,D)$ with

message input, $M[m(t)]$ and constant value, $K(t)$. Five initial inputs H0, H1, H2, H3 and H4 will provide inputs to both parts of the inner structure of RIPEMD-160 as shown in Table 1.

Table 1
Initialization value of RIPEMD-160 algorithm

Register	Left input	Right input	Initialization value
H0	A	A'	32'h67452301
H1	B	B'	32'hefcdab89
H2	C	C'	32'h98badcfe
H3	D	D'	32'h10325476
H4	E	E'	32'hc3d2e1f0

Table 2 illustrates non-linear function f(B,C,D) of RIPEMD-160 algorithm. There are five non-linear function such as f1, f2, f3, f4 and f5. In the non-linear function, the symbol $\oplus$, $\wedge$, $\vee$ and $\neg$ represent bitwise XOR, AND, OR and complement respectively. The operation for non-linear function for left and right part are different where the left part will use the normal sequence while for the right part, it will use the reverse order of non-linear function as shown in Table 3. From this table, there are 80 rounds of 32-bit of three different inputs to generate the non-linear functions.

Table 2
Non-linear functions of RIPEMD-160

Round (R)		Non-linear function	
R1	$0 \le t \le 15$	f_1(B,C,D)	B $\oplus$ C $\oplus$ D
R2	$16 \le t \le 31$	f_2(B,C,D)	(B $\wedge$ C) $\vee$ ($\neg$B $\wedge$ D)
R3	$32 \le t \le 47$	f_3(B,C,D)	(B $\vee$ $\neg$ C) $\oplus$ D
R4	$48 \le t \le 63$	f_4(B,C,D)	(B $\wedge$ D) $\vee$ (C $\wedge$ $\neg$ D)
R5	$64 \le t \le 79$	f_5(B,C,D)	B $\oplus$(C $\vee$ $\neg$ D)

Table 3
Left and right part of RIPEMD-160 rounding

Part	R1	R2	R3	R4	R5
Left	f_1(B,C,D)	f_2(B,C,D)	f_3(B,C,D)	f_4(B,C,D)	f_5(B,C,D)
Right	f_5(B,C,D)	f_4(B,C,D)	f_3(B,C,D)	f_2(B,C,D)	f_1(B,C,D)

There are ten constants are used in RIPEMD-160 algorithm which are five 32-bit constants from left and five 32-bit constants from right. Message input for both left and right parts of RIPEMD-160 represent by m and m'. The value of m and m' for both left and right of RIPEMD-160 will be executed which is based on the sequence of the round. As mentioned in the previous paragraph, there are five rounds for both left and right sides of RIPEMD-160 algorithm such as R1, R2, R3, R4, and R5. Each round consists of 16 steps and the design will complete the 80 rounds to obtain the output. There are two parts of shift value, shift from the left, s and shift from the right, s'. After processing 80 rounds of looping for this algorithm, the 160-bit of hash code can be computed. The output of RIPEMD-160 hash function can be obtained from the following equation.

$$H_0 = H_1 + C + D' \quad (14)$$

$$H_1 = H_2 + D + E' \quad (15)$$

$$H_2 = H_3 + E + A' \quad (16)$$

$$H_3 = H_4 + A + B' \quad (17)$$

$$H_4 = H_0 + B + C' \quad (18)$$

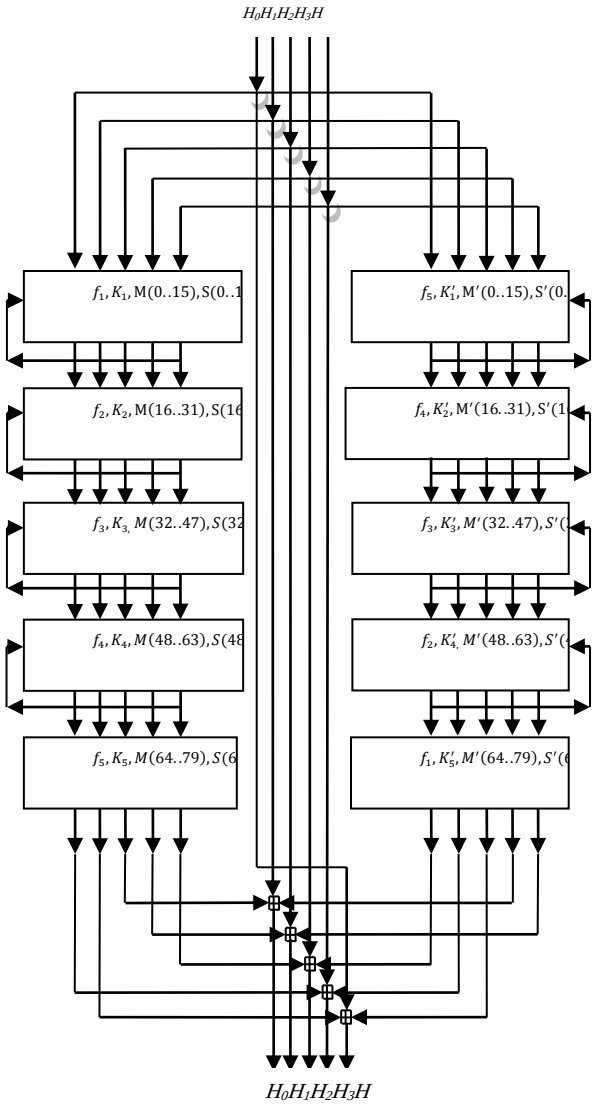

Fig. 5. Top Level of RIPEMD-160 architecture

Figure 5 illustrates the parallel data transformation for RIPEMD-160 hash function. It consists of left and right path. Finally, the output of message digest for RIPEMD-160 is equal to Message digest = $H_0 || H_1 || H_2 || H_3 || H_4$

3. Proposed Ripemd-160 Design

3.1. Iterative RIPEMD-160 Design

The proposed RIPEMD-160 architecture design is illustrated in Figure 6. Eight modules of RIPEMD-160 design are Counter, Coder, KConst, Message, Mux, Function parallel, Hash and Hash output modules.

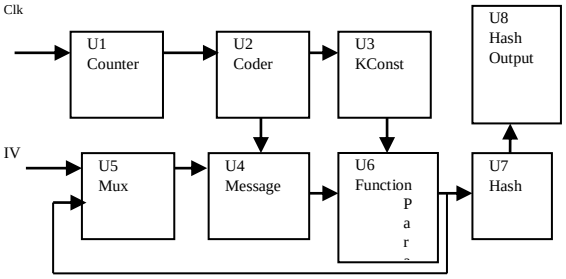

Fig. 6. Proposed RIPEMD-160 architecture design

The counter module will generate the round number of RIPEMD-160 until 80 rounds. It controls the operation of RIPEMD-160 algorithm. Then, coder module is used to divide the rounding number into five parts where each round consists of 16 steps. All the data from both modules are very important to execute the parallel function of non-linear function. These five functions will operate the operation based on the sequence from the previous table for both left and right sides of this algorithm. This module is the major process in order to execute the output of hash code. Five initial values which are the fixed module are executed first from the mux module in order to process the parallel function. The first output must be accurate because it will be feeding back to obtain the second data. In this case, mux module is used to select the appropriate input to generate data. This process is repeated until 80 rounds to complete the overall of this design. Hash module is stored the output of hash value before the output are converted into little-endian mode. Therefore, the output of this design will be generated by Hash output module which is the last output of the RIPEMD-160 design. The usage of register in terms of HDL coding style also affect the performance of the design.

3.2. RIPEMD-160 unfolding design with factor two

Figure 7 shows the top level of RIPEMD-160 unfolding design with factor two. The architecture is almost the same as iterative RIPEMD-160 design with a slightly different where this architecture has Func module. Some of these modules needs to be modified by applying the unfolding transformation technique to the design. The modules are Coder, KConst, Message and Function Parallel modules. The rounding numbers of this design need to be reduced by two. Thus, each round only consists of 8 rounds instead of 16 rounds in the iterative design. Overall, the number of looping is 40 rounds.

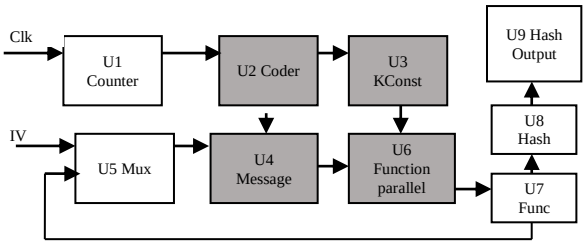

Fig. 7 Proposed RIPEMD-160 unfolding factor two design

Since this architecture considers both left and right sides of the design, all the modification needs to perform parallel data transformation. In order to generate this design in unfolding transformation factor two, the coder module is designed based on counter module. Figure 8 shows the block diagram of coder module of RIPEMD-160 unfolding factor two. It needs to reduce the number of steps in each round from 16 steps to 8 steps. In this module, there are five rounds such as Round0, Round1, Round2, Round3, and Round4. Each round consists of 8 steps. Overall, there are 40 rounds which is divided into 5 parts in coder module.

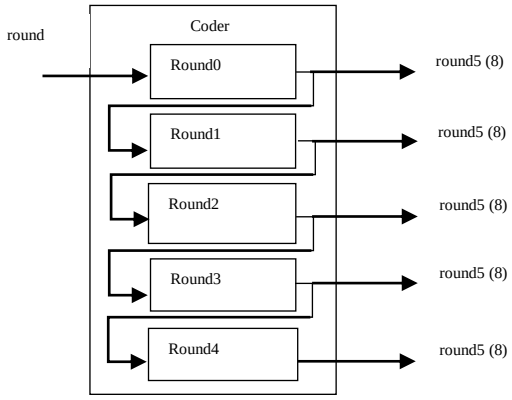

Fig. 8 Coder module of RIPEMD-160 unfolding factor two

Then, KConst module is a constant module for both left and right path of RIPEMD-160 design. It operates in parallel mode of two data from both parts. In other words, there are four data will generate for each cycle. Two from the left and two from the right path. Figure 9 shows the block diagram of KConst of RIPEMD-160 unfolding factor two. In each module consists of KConsta, KConstb from the left line and KConst1a, KConst1b from the right line. Therefore, it is divided into five because The RIPEMD-160 hash function contains five non-linear function. Each non-linear function will operate 8 steps for each round.

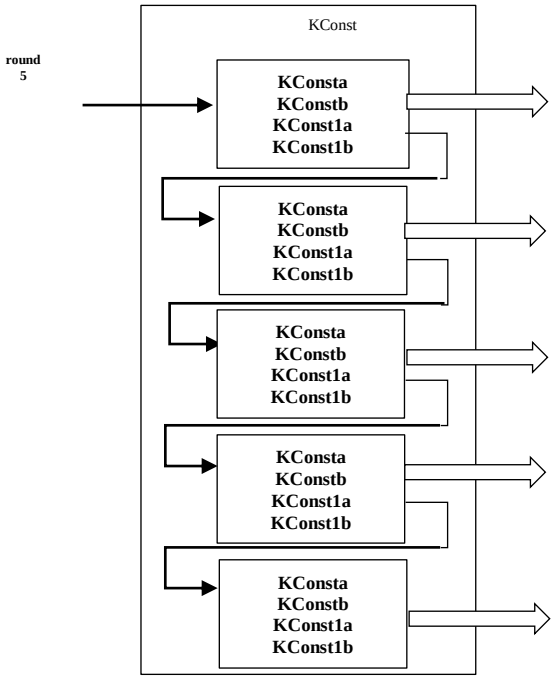

Fig. 9 KConst module of RIPEMD-160 unfolding factor two

Similarly, message module also generates the output for both left and right path of this design. It consists of message and shift values as shown in the previous table. First, all the data input of RIPEMD-160 will be divided into 16 parts. Each part comprises 32-bit of data. This data is stored in the 16-memory location. The counter will generate the input in order to execute the correct sequence of the data. Figure 10 shows the block diagram of message module of RIPEMD-160 unfolding factor two. In this message module design, message and shift values are combined into one 8-bit block where the first 4 bits is for message input and shift value will be for the second 4 bits. Overall, there are eight data will be generated from this module, 4 data are for message value left and right. Then, 4 data are for shift value left and right.

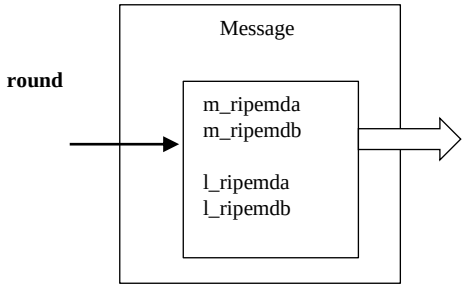

Fig. 10. Message module of RIPEMD-160 unfolding factor two

The last module that need to be modified in this design is Function parallel module. Figure 11 shows the Function parallel module of RIPEMD-160 unfolding factor two. There are two non-linear function will be executed in the parallel mode. The first non-linear function consists of input B, C, and D as shown in the Equation (2) and (8) for both left and right. Let say the output for first non-linear function is func_rmda and func_rmd1a. Both outputs represent data from both left and right data. Then, at the same time, the second non-linear function need to generate the output from three different inputs.

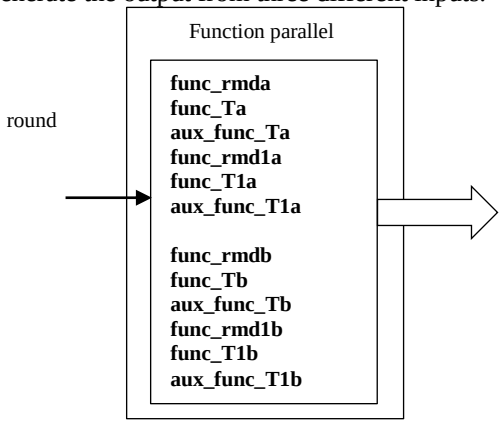

Fig. 11. Function parallel of RIPEMD-160 unfolding factor two

The following Algorithm 1 shows the derivation input for second non-linear function. Messagea, KConsta and shifta are message, constant and shift value for left path. Message1a, KConst1a and shift1a represent data for right path. The output for both second non-linear function are func_rmdb and func_rmd1b. Three different inputs for this function are obtained from Algorithm 1. The algorithm illustrates the compression function of RIPEMD-160 algorithm for both parts.

Algorithm 1: Compression Function of RIPEMD-160

```
func_rmda(B,C,D)
func_Ta = A + func_rmda + Messagea + KConsta
rot1_la = func_Ta << shifta
rot2_la = func_Ta  >> (32 – shifta)

func__rmd1a(B1,C1,D1)
func_T1a = A + func_rmd1a + Message1a + KConst1a
rot1_ra = func_T1a << shifta
rot2_ra = func_T1a  >> (32 – shifta)

aux_func_Ta = (rot1_la | rot2_la) + E
aux_func_T1a = (rot1_ra | rot2_ra) + E1

func_rmdb (aux_func_Ta, B, {C[21:0],C[31:22]})
func_rmd1b (aux_func_T1a,B1,{C1[21:0],C1[31:22]})
```

There are two non-linear functions will operate in one cycle. From Algorithm 1, func_rmda and func_rmd1a refer to the first non-linear function for both left and right path. Then func_rmdb and func_rmd1b represent the second non-linear function for both left and right path. In RIPEMD-160 unfolding factor two, two non-linear function will operate at the same time which is func_rmda and func_rmdb with three different input. In conventional compression function, there is only one non-linear function. The structure of compression function of conventional design is not really complicated compared with unfolding factor two. The input for second non-linear function, func_rmdb must be correct in order to obtain the result. The final output of this function generate output in the func module. The output will be different from compression function as shown in Figure 4 because there are two parallel data execute at the same time in one cycle in order to fulfill the unfolding transformation technique. Finally, the output of hash code is obtained after 40 rounds of looping.

3.3. RIPEMD-160 unfolding design with factor four

Modification on top level RIPEMD-160 design need to be designed in order to apply the unfolding transformation with factor four. In this method, the number of cycles in RIPEMD-160 architecture are reduced from 40 cycles to 20 cycles. By reducing number of cycles of the design the number of latencies will be reduced. Thus, this leads to high throughput of the RIPEMD-160 design. Similar with previous section which is unfolding with factor two, the modification needs to be designed for some modules such as Coder, KConst, Message and Function parallel. Figure 12 shows Coder module of RIPEMD-160 unfolding factor four. In this design, coder module consists of five rounds where each round comprises four steps.

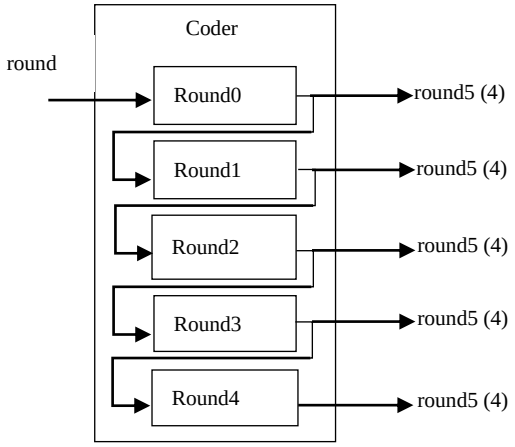

Fig. 12. Coder module of RIPEMD-160 unfolding factor four

Figure 13 shows KConst module of RIPEMD-160 unfolding factor four. KConst module provides constant value for both left and right path. In each round, there will be eight constants execute in parallel form. Message module needs a slight modification where the round is reduced into 20 rounds and the data for both message and shift value are compress into eight data of 8-bit input. From this figure, four data from the left and fours data from the right, KConst1a, KConst1b, KConst1c, KConst1d.

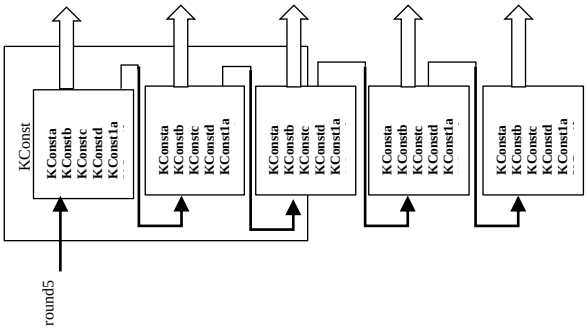

Fig. 13. KConst module of RIPEMD-160 unfolding factor four

Figure 14 shows message module of RIPEMD-160 unfolding factor four. In message module, it consists of message and sh value. There are eight output of messages, four from the left path which is m_ripemda, m_ripemdb, m_ripemdc, m_ripemdd and four from the right path which is l_ripemda, l_ripemdb, l_ripemdc, l_ripemdd. This module will operate the operation in 20 rounds.

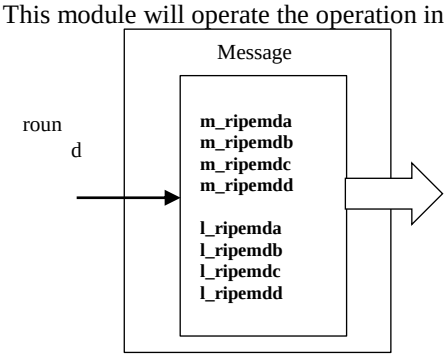

Fig. 14. Message module of RIPEMD-160 unfolding factor four

Figure 15 shows Function parallel module of RIPEMD-160 unfolding factor four. As we know, there are non-linear function in this module which is based on RIPEMD-160 algorithm. In order to generate unfolding transformation with factor four, the non-linear function need to be increased into four. In other words, there will be four parallel non-linear functions generate at the same time in order to produce the output in one cycle. Similar like previous section, unfolding transformation with factor two. Three different input of non-linear function need to be identified. The output func_rmda, func_rmd1a, func_rmdb and func_rmd1b represent first and second non-linear function for both left and right respectively. While for the third and fourth non-linear function in this module, it is represented by func_rmdc, func_rmd1c, func_rmdd and func_rmd1d respectively.

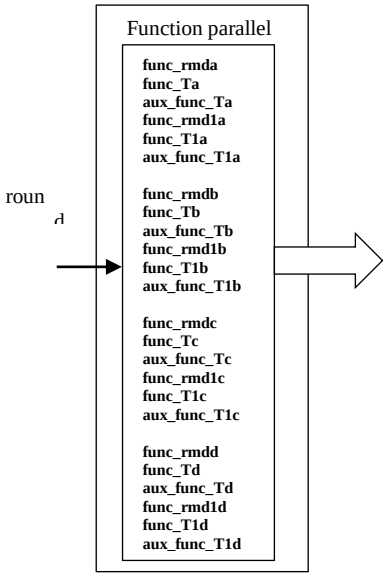

Fig. 15. Function parallel module of RIPEMD-160 unfolding factor four

The Algorithm 2 shows the three different input for both left and right path of four non-linear function in this module. The new idea is comprised in this paper especially the unfolding transformation with factor four where all the non-linear function needs to be executed in four parallel transformation.

Algorithm 2: Input for four non-linear function

```
func_rmda(B,C,D)
func__rmd1a(B1,C1,D1)

func_rmdb (aux_func_Ta, B, {C[21:0],C[31:22]})
func_rmd1b (aux_func_T1a,B1,{C1[21:0],C1[31:22]})

func_rmdc (aux_func_Tb, aux_func_Ta, {B[21:0],B[31:22]})
func_rmd1c (aux_func_T1b,aux_funcT1a,B1[21:0],B1[31:22])

func_rmdd(aux_func_Tc,aux_func_Tb,{aux_func_Ta[21:0],
aux_func_Ta[31:22])
func_rmd1d(aux_func_T1c,aux_func_T1b,{aux_func_T1a[21:
0], aux_func_T1a[31:22])
```

The compression function of RIPEMD-160 unfolding factor four almost similar to RIPEMD-160 unfolding factor two as shown in Figure 12. By adding other two

non-linear function from both left and right which are func_rmdc, func_rmd1c, func_rmdd, and func_rmd1d. The three different input for each non-linear function for unfolding factor four are shown in algorithm 2. The three different input must be evaluated correctly. This is the most important part in designing unfolding design. The five output of RIPEMD-160 hash function depends on the correct result because it will be looping until the last rounds in order to obtain the hash value from the Equation (14) until (18).

After getting the result from four non-linear function. The output for A, B, C, D, and E for both left and right paths of this design are computed. The following equation shows the final output from the left which are l_A, l_B, l_C, l_D, and l_E. While the output from the right path are r_A, r_B, r_C, r_D and r_E. These outputs are looping until 20 rounds because of the implementation of unfolding transformation with factor four. Finally, all these outputs are combined by using the formula from Equation (14) to (18) to obtain the hash codes. All the outputs need to be converted to little endian mode because the result for RIPEMD-160 is in little endian format. Unfolding technique will reduce the number of latency of the RIPEMD-160 design. High throughput will be obtained if the unfolding factor increase which is based on Equation (19). From this equation, the number of cycles of RIPEMD-160 unfolding will reduce based on the number of unfolding factor. If the number of unfolding factor increase, the number of cycles reduce. This leads to high throughput of RIPEMD-160 unfolding transformation technique.

4. Results and Discussions

4.1. Testing and verification

In this project, there are three proposed RIPEMD-160 algorithms are designed such as iterative, unfolding RIPEMD-160 with factor two and unfolding RIPEMD-160 with factor four using Verilog HDL code. The RIPEMD-160 designs are compiled with Arria II GX in order to identify the area, maximum frequency and throughput of the designs. By using Quartus II advisors, the performance of the design can be improved. TimeQuest Timing Analyzer is used to obtain the maximum frequency of the RIPEMD-160 design. By giving the appropriate clock constraint to the designs, the timing requirement of the design will be met. This process is repeated until the RIPEMD-160 design achieve the target of the timing requirement of the design. In this project, 8 ns clock constraint was given to the all designs namely RIPEMD-160 iterative, RIPEMD-160 unfolding with factor two and RIPEMD-160 unfolding with factor four. All the RIPEMD-160 designs meet the timing requirement.

4.2. Result and simulation waveform

Table 4 shows the synthesis and implementation results of three proposed RIPEMD-160 designs. Other previous RIPEMD-160 design is also shown in this table for comparison. They are evaluated from the viewpoints of the area, maximum frequency and throughput of the design. The throughput of the design can be calculated using the following Equation (29). The number of clock cycles for each design is different because of the structure of RIPEMD-160 design. For iterative design, the number of clock cycles is 101.50. While for unfolding with factor two design, the number of clock cycles is 60.5. Finally, unfolding with factor four design, the number of clock cycles is reduced to 39.5.

$$\text{Throughput} = \frac{512 \times \text{Maximum Frequency}}{\text{No.of clock cycles}} \qquad (19)$$

Table 4
Synthesis and implementation results of RIPEMD-160

Design	**Device**	**ALUTs / CLBs**	**Reg**	**FMax (MHz)**	**Throughput (Mbps)**
RIPEMD-160 (iterative)	Arria II GX	1269 ALUTs	677	133.03	671.05
RIPEMD-160 (Unfolding-2)	Arria II GX	2015 ALUTs	530	125.19	1059.46
RIPEMD-160 (Unfolding-4)	Arria II GX	3224 ALUTs	1011	135.28	1753.50
RIPEMD-160 (iterative) [6]	Xilinx Virtex 300E	1004 CLBs	-	42.9	65
RIPEMD-160 (iterative) [7]	EPF10K50SBC356-1	-	-	26.6	84
RIPEMD-160 (iterative) [8]	XC2VP30	4410 ALUTs	-	100.05	624
RIPEMD-160 (iterative) [9]	XC2V250	14911 LUTs	-	43.47	137.4

There are several other FPGA family devices such as Stratix for high-performance and Cyclone for low-cost FPGAs. Hence, in order to obtain balance of power and performance, Arria II GX device was chosen because Arria series provide low-cost transceiver of FPGAs device. Besides, it also delivers optimal performance and power efficiency. The proposed RIPEMD-160 design can increase the frequency maximum of the design. The improvement of frequency maximum because of the number of registers required in designing the RIPEMD-160 design. By using unfolding and following the guidelines of writing of better HDL coding, the frequency of design can be improved significantly. Besides, the architecture of FPGA device also plays important roles in designing the RIPEMD-160 design. By selecting an appropriate FPGA device, the performance of MD5 design can be enhanced. Table 4 shows the previous implementation of RIPEMD-160 design. It is difficult to find the same device for the same design because of the limitation of budget and devices.

From this table, the area implementation of the design increase from iterative design to unfolding design.

However, the throughput of the RIPEMD-160 unfolding with factor four design increase significantly. The percentage of increment is about 62%. Furthermore, this design shows the highest throughput if compared with other RIPEMD-160 designs. Based on Equation (29), the number of cycles for RIPEMD-160 unfolding factor two is reduced by two which is from 80 cycles to 40 cycles. Then, the improvement of throughput will be obtained. Similarly, the RIPEMD-160 unfolding factor four will reduce by four which is the number of cycles reduce from 80 cycles to 20 cycles. The small number of cycles can increase the throughput of the design as well as performance of the design. Figure 16 shows the simulation results for RIPEMD-160 unfolding design with factor four. The message input for this simulation is 'abc' which is '616263' in hexadecimal form. The message input needs to be padded before start the processing. The output hash code is '8eb208f7305d987a9b044a8e98c6b087f15a0bfc'. From the simulation waveform, the output hash code provides the correct results in terms of both functional and timing simulation.

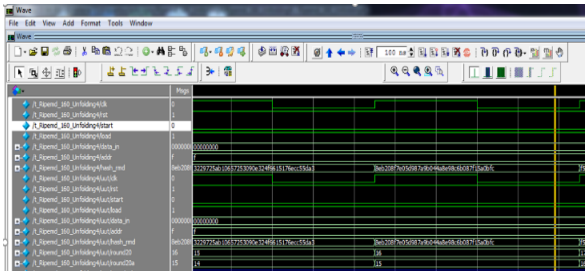

Fig. 16. Simulation of RIPEMD-160 unfolding with factor four design

4.3. Performance to area ratio

The proposed RIPEMD-160 achieves high throughput with unfolding transformation design. RIPEMD-160 designs are compared in terms of performance to area ratio. The comparison of performance to area ratio is shown in Table 5. From the results, the performance to area ratio of RIPEMD-160 with factor four increase about 1.51% if compared with iterative design of RIPEMD-160. By using the unfolding technique, the throughput of the RIPEMD-160 can be improved significantly with the increment of performance to area ratio. For these design implementations, performance(bps), area (the number of ALUTs/CLBs) and performance to area ratio (Mbps/ALUTs or CLBs) are evaluated.

Table 5
Comparison of performance to area ratio

Design	RIPEMD-160 (iterative)	RIPEMD-160 (Unfolding-2)	RIPEMD-160 (Unfolding-4)	RIPEMD-160 (iterative) [6]	RIPEMD-160 (iterative) [8]	RIPEMD-160 (iterative) [9]
Mbps/ALUTs	0.5288	0.5257	0.5439	0.0647	0.1415	0.009

5. Conclusion

In conclusion, implementation of unfolding transformation can improve the performance of RIPEMD-160 hash function by reducing the number of cycles where the data generate in parallel transformation. This leads to high throughput of RIPEMD-160 design. From the results, it shows that the maximum frequency of RIPEMD-160 unfolding with factor four gives the highest maximum frequency among others. The throughput of RIPEMD-160 design also increase significantly by using this methodology which is about 1753.50 Mbps.

Acknowledgment

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

References

Rodriguez-Henriquez, Saqib, F., Diaz-Perez, N.A. Kaya, & Koc A. C. (2006), Cryptographic Algorithms on Reconfigurable Hardware, Springer Series on Signals and Communication Technology, pp. 211-242.

Abu Bakar, Rosbi,N. Uzaki S., K. (2017), Cryptocurrency Framework Diagnostics from Islamic Finance Perspective: A New Insight of Bitcoin System Transaction, International Journal of Management Science and Business Administration, 4(1),19-28.

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

Dobbertin, H. Bosselaers, A. & Preneel B. (1996), RIPEMD-160, a strengthened version of RIPEMD, Fast Software Encryption, LNCS 1039, Springer-Verlag, 71-82.

Dominikus S. (2002), A hardware implementation of MD4-family hash algorithms, Proceeding 9th International Conference on Electronics, Circuits and Systems, 3, 1143-1146.

Ng, C., Ng T. & Yip, K. (2004), A Unified Architecture of MD5 and Ripemd-160 Hash Algorithms, Proceedings of the 2004 International Symposium on Circuits and Systems, ISCAS'04, .2,889-892.

Kneˇzevi´c, M. Sakiyama, K., Lee Y. K., & Verbauwhede I. (2008), On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm, International Conference on Application-Specific Systems, Architectures and Processors,2008. ASAP 2008, Leuven, 85 – 90.

Khan, E., Watheq El-Kharashi, M., Gebali, F. & Abd-El-Barr, M. (2017), Design and Performance Analysis of a Unified, Reconfigurable HMAC-Hash Unit, IEEE Transaction on Circuits and Systems, pp. 2683-2695.

Wang, X. Lai, X. Feng, Chen, D. H. & Yu, X. (2005), Cryptanalysis of the hash functions and MD4 and

RIPEMD. Advances in Cryptology, EUROCRYPT 2005.

Mendel, F. Pramstaller, Rechberger, N. C. & Rijmen, V. (2006), On the Collision Resistance of RIPEMD-160, International Conference on Information Security, ISC 2006, LNCS 4176, 101–116.

Michail, H. E., Gregoriades, Kelefouras, A. V., Athanasiou,G., Kritikakou, A. & Goutis, C. (2020), Authentication with RIPEMD-160 and Other Alternatives: A Hardware Design Perspective, New Advanced Technologies, book chapter, 103 – 124.

Giechaskiel, I. & Cremers ,C. & Rasmussen, K. B. (2018), When the Crypto in Cryptocurrencies Breaks: Bitcoin Security under Broken Primitives, IEEE Security & Privacy Journal, 16(4), 46 -58

Abbas, A. Voß, R., Wienbrandt , L. Schimmler, M. (2014), An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs, 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 454 – 461

Kuznetsov, A. Shekhanin, K. Kolhatin,A Kovalchuk,. D. Babenko,V. & Perevozova, I. (2019), Performance of Hash Algorithms on GPUs for Use in Blockchain, 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT), 166 – 170.

Michail, H. Gregoriades, A. V Kelefouras,. Kotsiolis, A. Papagianopoulou,D. & Goutis, C. (2010), HW/SW co-Design Integrating High – Speed Authentication Module for IPSec/IPv6, 2010 Fifth International Conference on Digital Telecommunications, 138 - 142.

Lee, Y. K., Chan, H. & Verbauwhede, I. (2008), Design Methodology for Throughput Optimum Architectures of Hash Algorithms of the MD4-class, Journal of Signal Processing Systems, 53, 89–102.

Al-Mhadawi, M. M., & AlbahranA. A. i (2019), Hybrid Method as Pseudo-Random Bits Generator, 2019 International Conference of Computer and Applied Sciences (CAS2019), 250-255.

Dewi, A. & Setiawan,S. H. (2019), Implementation of SHA-256 and AES-256 for Securing Digital Al Quran Verification System, 2019 Fourth International Conference on Informatics and Computing (ICIC)

Michail, H., Thanasoulis,V., Schinianakis, Panagiotakopoulos, D. G. & Goutis, C. (2008), Application Of Novel Techniques In RIPEMD-160 Hash Function Aiming At High-Throughput, 2008 IEEE International Symposium on Industrial Electronics, Cambridge, UK.

Michail, H. E., Thanasoulis,V. N., Panagiotakopoulos,G. A. A., Kakarountas,P. & Goutis, C. E. (2008), Efficient Pipelined Hardware Implementation of RIPEMD-160 Hash Function, World Academy of Science, Engineering and Technology, International Journal of Electrical and Computer Engineering, 2(2).

This article can be cited:
Suhaili, S., Watanabe, T., Julai, N. (2022). Throughput Improvement of RIPEMD-160 Design using Unfolding Transformation Technique. *Journal of Optimization in Industrial Engineering*, 15(1), 207-216.

http://www.qjie.ir/article_686437.html
DOI: 10.22094/joie.2021.1941138.1896