Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

Design of Optimized Pipelined RIPEMD-160 with High Frequency and Throughput

S. Suhaili[*,a], *T. Watanabe*[b]

[a,*]Electronic Department, Faculty of Engineering, Universiti Malaysia Sarawak (UNIMAS), 94300 Kota Samarahan, Sarawak

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

[a,*]*sushamsiah@unimas.my*, [b]*watt@waseda.jp*

Abstract –*The main objective of this paper is to design optimized pipelined RIPEMD-160 with efficient design strategies. There are two proposed designs in this paper such as iterative design and pipelined design. The results show the pipelined design provides high frequency as well as throughput of the design. The improvement of these designs based on HDL design style where the placement of register need to be considered, by adding the several register to the design and by using Quartus II Advisor tools. Furthermore, by using TimeQuest Timing Analyzer, timing requirement of the design such as setup and hold time of the design can be achieved and the maximum frequency of the design can be obtained. This paper focuses on maximum frequency of the designs. Therefore, the methodology on how to improve the speed of the design needs to be taken into consideration. In addition, nowadays high throughput of the hash function design is important since all the design need to be fast enough through some application. By using pipelined design, the frequency and throughput of the design can be improved which is about 250 MHz and 7.805 Gbps respectively with small area implementation.*

Keywords: Hash Function, iterative, pipelined, RIPEMD-160

1.0 INTRODUCTION

Hash Function is widely used for some cryptographic application. It receives arbitrary input and provides fixed length of the output based on the structure of hash function. The output of hash function known as hash code or message digests. There is no key for hash function as shown in Figure 1 which is the block diagram of hash function. There are several different types of hash function such as MD5, SHA-1, RIPEMD-160 and etc. Design and implementation of these hash functions on reconfigurable hardware have been done by many researchers [2,3,4,5,6]. Nowadays, efficient design of hash function is important for some application. Therefore, some techniques need to be taken into consideration in order to improve the performance of hash function in terms of frequency and throughput of the design.

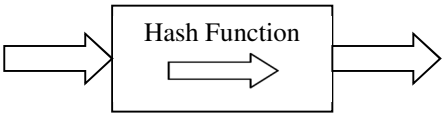

Figure 1: Hash Function

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

Motivation of this paper by choosing RIPEMD160 as hash function for some application is because of the structure of hash function where it considers both side left and right line for shift and message values. Figure 2 shows the requirement of hash function. M represent message, H is hash function and h is hash code of hash function. From Figure 2(a), it shows that hash code will be obtained after receiving the message input. However, the process to obtain the message from hash code cannot be done by using hash function. This refers to one way property of hash function. Furthermore, if there are two different messages and there will be two different messages digest of hash function. This is because hash function is strong enough against collisions. Collision is not desired characteristics of hash function. This process shows in Figure 2(c) and sometimes called strong collision resistance.

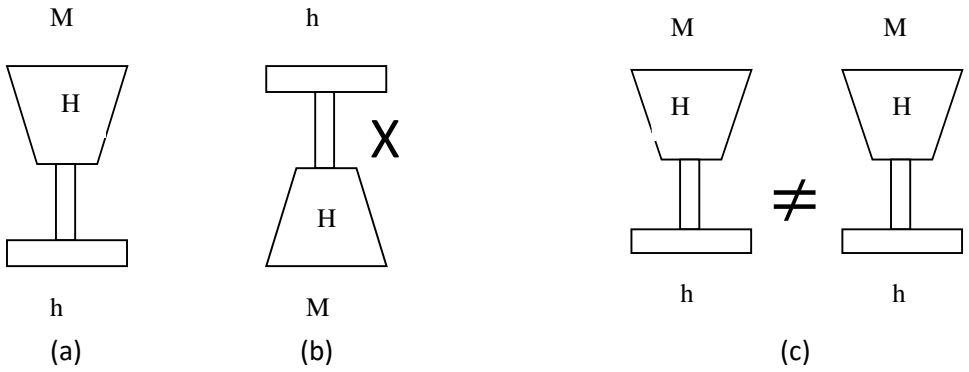

Figure 2: Requirement of Hash Function [1]

2.0 METHODOLOGY

2.1 RIPEMD-160 Algorithm

The structure of RIPEMD-160 (Race Integrity Primitives Evaluation Message Digest) algorithm produces 160-bit length of the hash code which consists of five 32-bit words. The output of RIPEMD-160 is in little-endian format. The algorithm 1 shows the RIPEMD-160 algorithm. There are 80 steps processing for five 32-bit different initial inputs.

Algorithm 1 : RIPEMD160 algorithm

for t=0 { $A = H_0, B = H_1, C = H_2, D = H_3\ E = H_4;$
$A' = H_0, B' = H_1, C' = H_2, D' = H_3, E' = H_4;$
for t=0 to 79 {
$T = rol_{s(t)}\left(A + f_t(B,C,D) + X[m(t)] + K(t)\right) + E;$
$A = E;\ E = D;\ D = rol_{10}(C);\ C = B;\ B = T;$

$T' = rol_{s'(t)}\left(A' + f_{79-t}(B',C',D') + X[m'(t)] + K(t)\right) + E';$
$A' = E';\ E' = D';\ D' = rol_{10}(C');\ C' = B';\ B' = T';$
}
$H_0 = H_1 + C + D';\ H_1 = H_2 + D + E';\ H_2 = H_3 + E + A';$
$H_3 = H_4 + A + B';\ H_4 = H_0 + B + C';$ }

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

From algorithm 1, five initial inputs H_0, H_1, H_2, H_3, and H_4 will be given to five inputs from left namely A, B, C, D and E and five inputs from right namely A', B', C', D' and E'. There are two parallel process of RIPEMD-160 occur during execution. Figure 3 illustrates the structure of RIPEMD-160 algorithm. From this figure, there are non-linear function *f*(B,C,D) for input B, C and D. The output D and D' shift (rotation) to the left over 10 positions.

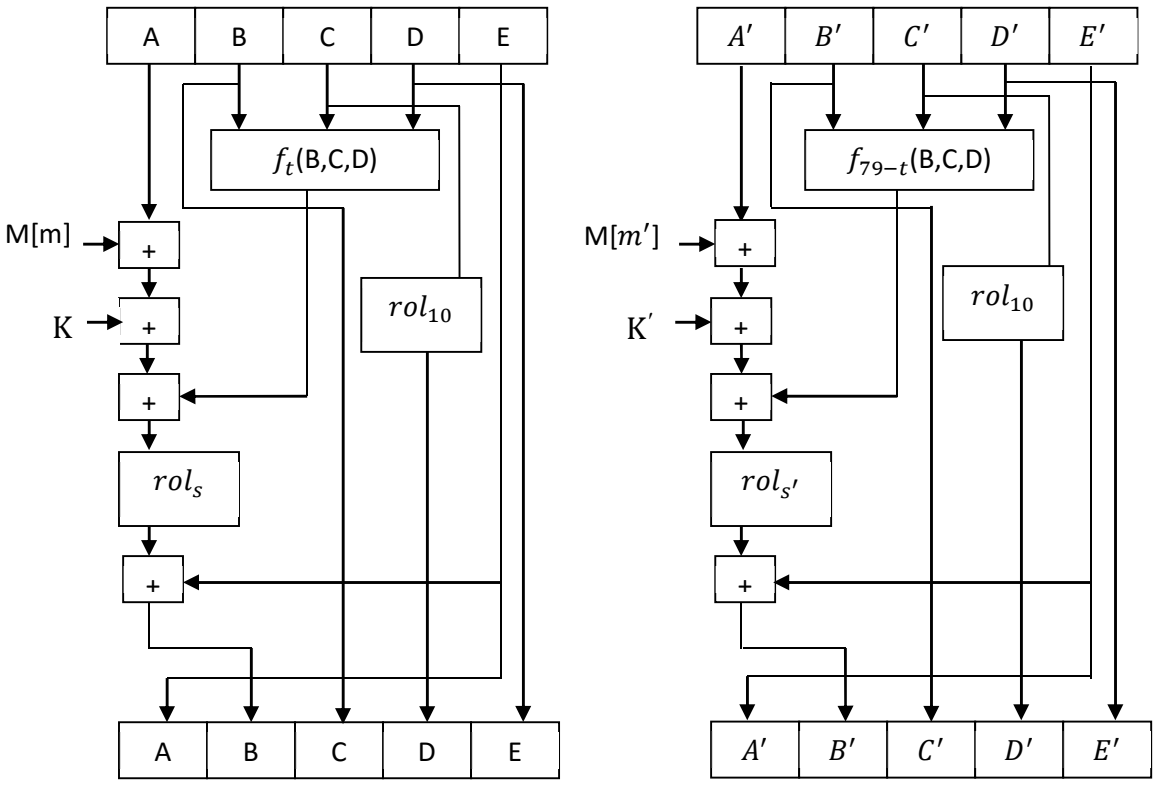

Figure 3: Compression Function of RIPEMD-160 Block Diagram

Figure 4 shows the top level for RIPEMD-160 algorithm where the initial input, input value for message selection, shift rotation and constant are described in the next following table. For iterative design, only one block of compression function will be used in the process while for pipelined design, there are five block of compression function which consists of different non-linear function *f*(B,C,D). From this Figure, M and M' represent 32-bit message input with message selection for each step, ten constant inputs $K$and K' for five parallel operation and shift value for 80 steps operation. Finally, the output of hash function will be obtained by adding with five initial inputs, five variables A, B, C, D and E from the left and five variables A', B', C', D' and E' from right. Since the output of RIPEMD-160 hash function in little-endian format, the final output need to be changed back into normal value in order to obtain the correct output of RIPEMD-160 hash function.

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

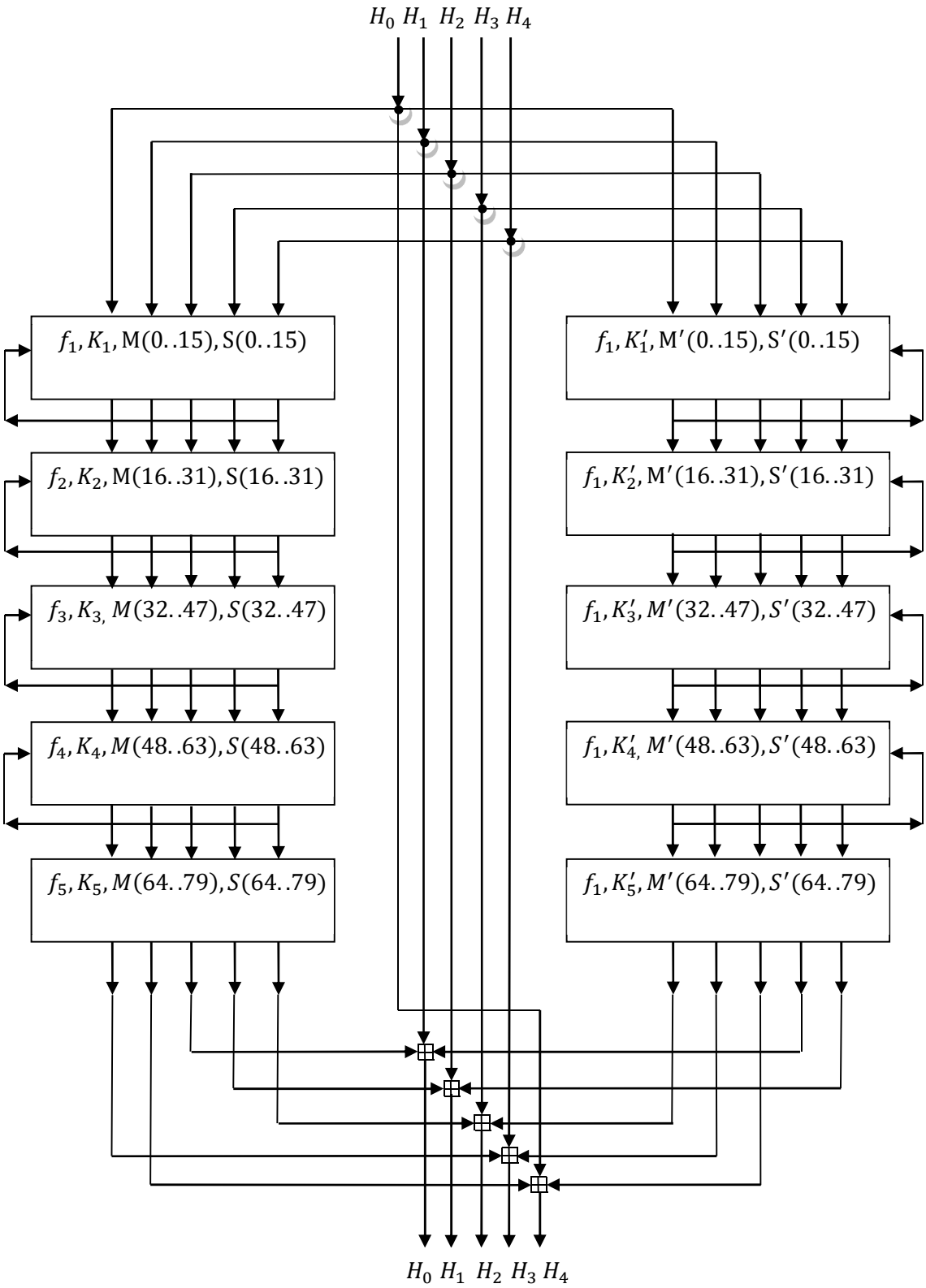

Figure 4: RIPEMD-160 Top-Level

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

The following tables show some parameter operation to describe RIPEMD-160. Table 1 shows initial input for RIPEMD-160. There are five initial input H_0, H_1, H_2, H_3, and H_4.

Table 1: Initial Value

Register	Buffer Initialization
H_0	32'h67452301
H_1	32'hefcdab89
H_2	32'h98badcfe
H_3	32'h10325476
H_4	32'hc3d2e1f0

There are five different non-linear function for *f*(B,C,D) of RIPEMD-160. Table 2 shows the non-linear function for five different steps f_1, f_2, f_3, f_4 and f_5. The operation $\oplus$, $\wedge$, $\vee$ and $\neg$ represent bitwise XOR, AND, OR and complement respectively. Table 3 illustrates the rounding of boolean function for both left and right. The sequence of the function will be based on Table 3 where for the right lines will use reverse order of non-linear function.

Table 2: Non-linear Function of RIPEMD-160

Step number	Non-linear Function, *f* (B,C,D)	
$\mathbf{0 \leq t \leq 15}$	f_1 (B,C,D)	B $\oplus$ C $\oplus$ D
$\mathbf{16 \leq t \leq 31}$	f_2 (B,C,D)	(B $\wedge$ C) $\vee$ ($\neg$B $\wedge$ D)
$\mathbf{32 \leq t \leq 47}$	f_3 (B,C,D)	(B $\vee \neg$ C) $\oplus$ D
$\mathbf{48 \leq t \leq 63}$	f_4 (B,C,D)	(B $\wedge$ D) $\vee$ (C $\wedge \neg$ D)
$\mathbf{64 \leq t \leq 79}$	f_5 (B,C,D)	B $\oplus$ (C $\vee \neg$ D)

Table 3: Round Function

Lines	Round 1	Round 2	Round 3	Round 4	Round 5
Left	f_1	f_2	f_3	f_4	f_5
Right	f_5	f_4	f_3	f_2	f_1

In order to produce the output of RIPEMD-160 hash function, ten different constants which are five from the left and five from the right will be used in this design. Only constant, K for $0 \leq t \leq 15$ and constant, K' for $64 \leq t \leq 79$ have the same value which is 32'h00000000. Table 4 shows five Constant, K and K' for five round of RIPEMD-160 in hexadecimal.

Table 4: Constant

Step number	Constant, K	Constant, $\mathbf{K'}$
$\mathbf{0 \leq t \leq 15}$	32'h00000000	32'h50A28BE6
$\mathbf{16 \leq t \leq 31}$	32'h5A827999	32'h5C4DD124
$\mathbf{32 \leq t \leq 47}$	32'h6ED9EBA1	32'h6D703EF3
$\mathbf{48 \leq t \leq 63}$	32'h8F1BBCDC	32'h7A6D76E9
$\mathbf{64 \leq t \leq 79}$	32'hA953FD4E	32'h00000000

Since RIPEMD-160 use two parallel lines which are from left, and right, there will be two different values for message selection, m and m' as shown in Table 5. In this case, the data

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

needs to be kept in memory first in order to make sure that the data is already in appropriate place. Once the data already is in place, the counter will be used to call the message based on value from Table 5. The messages are divided into 16 parts where each part consists of 32-bit message word. In this design, there are five rounds for both lines where each round consists of 16 steps. There are 80 steps processing to complete the output of this design.

Table 5: Message Selection

Step number	Message word, m															
0 ≤ t ≤ 15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16 ≤ t ≤ 31	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
32 ≤ t ≤ 47	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12
48 ≤ t ≤ 63	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2
64 ≤ t ≤ 79	4	0	5	9	7	12	2	10	14	1	3	8	11	6	15	13
Step number	Message word, m'															
0 ≤ t ≤ 15	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
16 ≤ t ≤ 31	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
32 ≤ t ≤ 47	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
48 ≤ t ≤ 63	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14
64 ≤ t ≤ 79	12	15	10	4	1	5	8	7	6	2	13	14	0	3	9	11

Similar with Shift value, there are two Shift such as Shift from left,s and Shift from right, s′. Table 6 shows the 4-bit left rotation operation from left and right of RIPEMD-160. By using the combination method for both message value and Shift value into one variable in Verilog code, the output of the design can be monitored easily.

Table 6: Shift value

Step number	Shift, s															
0 ≤ t ≤ 15	11	4	15	12	5	8	7	9	11	13	14	15	6	7	9	8
16 ≤ t ≤ 31	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12
32 ≤ t ≤ 47	11	13	6	7	14	9	13	15	14	8	13	6	4	12	7	5
48 ≤ t ≤ 63	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12
64 ≤ t ≤ 79	9	15	5	11	6	8	13	12	5	12	13	14	11	8	5	6
Step number	Shift, s′															
0 ≤ t ≤ 15	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
16 ≤ t ≤ 31	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
32 ≤ t ≤ 47	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
48 ≤ t ≤ 63	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8
64 ≤ t ≤ 79	8	5	12	9	12	5	14	6	8	13	6	5	15	13	11	11

Penerbit Akademia Baru
Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

2.2 Proposed Design

In this paper, there are two designs have been proposed such as iterative design and pipelined design. Iterative design uses only one compression function while pipelined design uses five different compression functions. The proposed pipelined design based on modification on paper [5]. Slight modification has been done in order to produce optimized RIPEMD-160.

2.2.1 Iterative Design

Iterative design is basic design of hash function. Figure 5 shows iterative design for step function of RIPEMD-160. It consists of non-linear function, shift rotation over 10 positions, left circular shift with shift value. M and K represent message and constant of RIPEMD-160 respectively. In this design only one block of step function will be used during the execution process. All the parameters stated in the previous section will be used in this design in order to obtain the correct output. Maximum frequency of the design can be achieved by using advisors in terms of resource, timing and power. In addition, by putting register at the end of the design output port can also increase the maximum frequency as we know register-to-register delay in one of the largest delays in modern circuit design.

There is only one non-linear function block for iterative design that consists of five different non-linear functions. First, message will be kept in memory with input load which means if there is input load with logic'1', the data will transfer to specific length of register. Then, message will be kept in memory starting from 0 until 15 locations. By using this method, it is easier to call the message based on message selection ordering as shown in previous table. Counter can be used to call the message,m,m' and Shift, s, s' value. The iteration of this architecture will be processed until 80 steps. Finally, the output $A, B, C, D,$ and E from left and A', B', C', D' and E' from right will be added with initial value H_0, H_1, H_2, H_3 and H_4 at the end of this design.

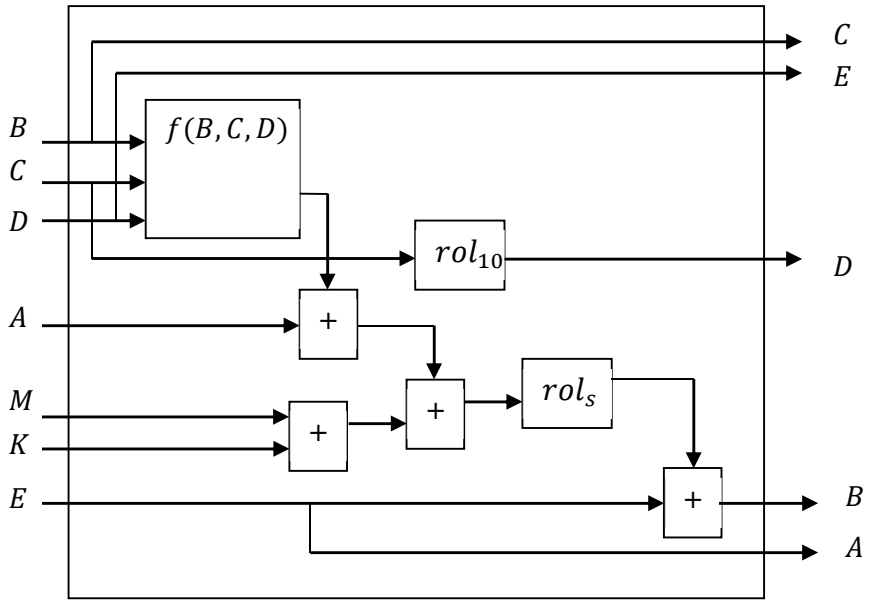

Figure 5: RIPEMD-160 Iterative Architecture Design

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

2.2.2 Pipelined Design

Figure 6 shows the concept of pipelined design. There are five stages of RIPEMD-160 design starting from round 1 until round 5. Each round consists of 16 steps and overall steps for this design are 80 steps. By using pipelined design the performance of the design can be improved significantly even though there will be slight increment in terms of area implementation. However Quartus II advisor can help to reduce the logic elements that have been used for this design. As we know timing is important in order to obtain the maximum frequency of the design. Therefore, timing requirement for setup and hold time need to be met. In addition, TimeQuest Timing Analyzer will be used by giving the appropriate constraint to the design.

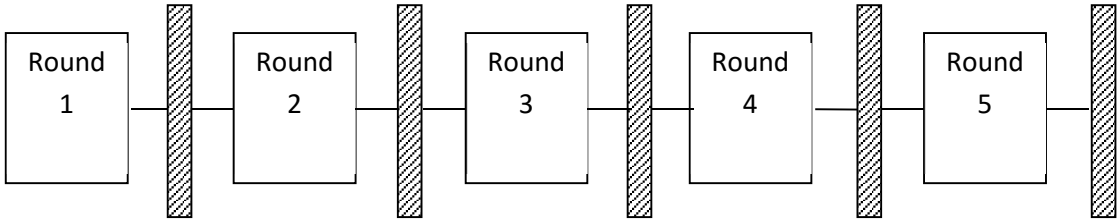

Figure 6: Pipelined Design

By using the concept of pipelined design as shown in Figure 6, RIPEMD-160 pipelined design can improve the performance of the design. Besides, modification on idea from paper [5] has been taken into consideration. Figure 7 illustrates the pipelined architecture design for single block of RIPEMD-160. The difference between paper [5] and this paper is the Message,M_3 and Constant, K_3 will be added with input A.

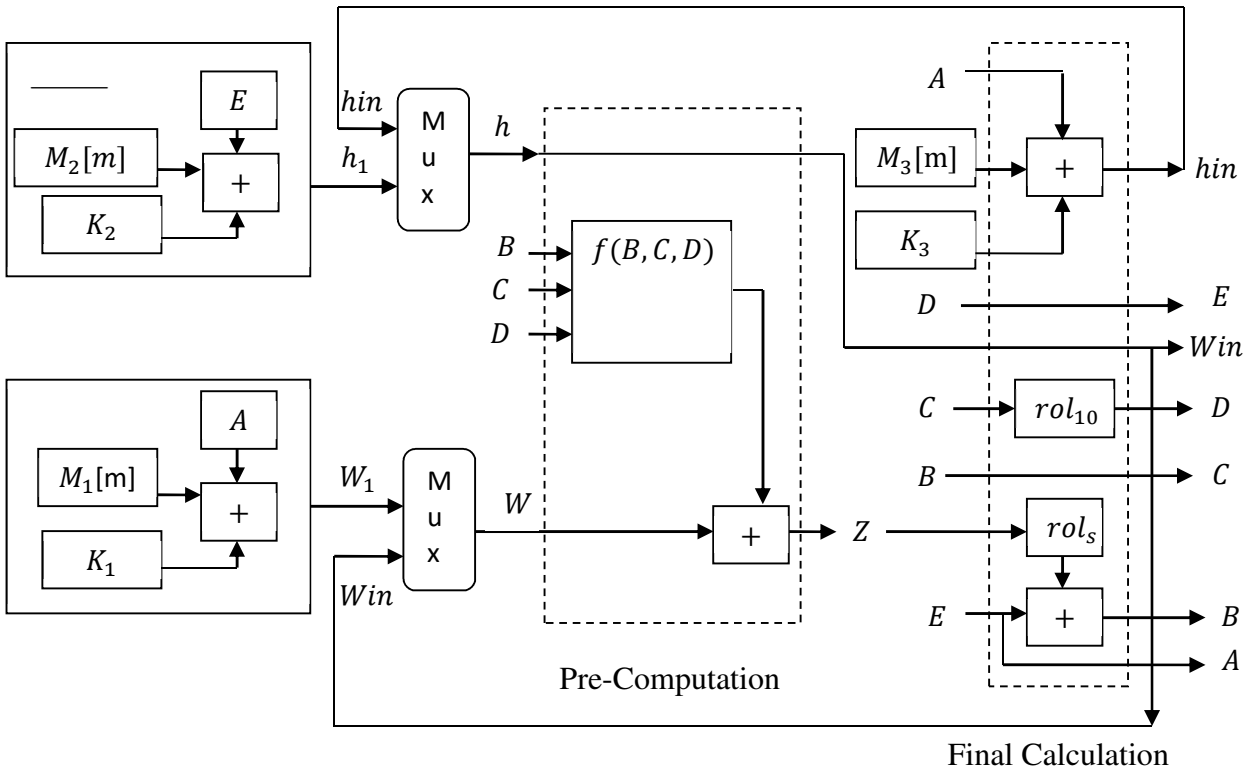

Figure 7: RIPEMD-160 Pipelined Architecture Design for Single Block

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

By using the same sequence as normal RIPEMD160 algorithm for input A, Message, M_3 and Constant, K_3 the design can be improved significantly if compared with previous design. From Figure 7, there are two initialization values for variables h and W. First, Let us consider the input M_1 and K_1. These two inputs will be added with input A and gives output h_1. Then, the output will go to Win and will be added to non-linear *f*(B,C,D). Finally, the output Z will shift to the left with 4-bit amount and will be added with input E in order to obtain the output B, B'. This concept is the same as original idea which is based on algorithm 1 of RIPEMD-160 algorithm. Now, for second cycle, $M_2$and K_2will be added with E value. As we know, the input for A_{t+1} is equal to E_t as shown in the following equation (1). In this case, the output will be directly added with non-linear *f*(B,C,D). The sequence to obtain output B,B' is similar with previous method but with different input. By using algorithm 1 of RIPEMD-160 algorithm, the output A, C, D and E from left and A', C', D' and E' from right can be obtained.

$$A_{t+1} = E_t \tag{1}$$

Table 7 shows the proposed pipelined design of RIPEMD-160. Some modification has been done in order to increase the performance of the design. In this design, there are five pre-computation block because of different five different types of non-linear function while for Final-Calculation, only one block unit will be used. Calculation for M_3 and K_3 start at this stage. The output of hin will be used for the rest of calculation of RIPEMD-160 hash function. This output h will go to Win. This Win will be selected as W input for each of Pre-Computation. The process execute in pipelined design and selection needs to be done when comes to different rounding of RIPEMD-160.

Table 7: RIPEMD-160 input for Pipelined Design

Pre-Computation	Final-Calculation
$A_t = A_t$	$A_{t+1} = E_t$
$B_t = B_t$	$B_{t+1} = E_t + Rol_s(Z_t)$
$C_t = C_t$	$C_{t+1} = B_t$
$D_t = D_t$	$D_{t+1} = Rol_{10}(C_t)$
$E_t = E_t$	$E_{t+1} = D_t$
$Z_t = f_t$(B,C,D) + W_t	$W_{t+1} = h_t$
$h_t = h_t$	$h_{t+1} = M_t + K_t + A_t$

Pipelined design can contribute to fast design but it suffers with large area implementation. However, by using Quartus II advisors, it can help to reduce the area implementation, increase the timing and reduce the power consumption. HDL coding style needs to consider as well in designing the design because this design will be implemented into gate level implementation. The design can be improved by using proper and efficient coding style. Moreover, by using appropriate constraint to the design, the setup and hold time requirement will be met. In this design, sdc input clk 4 is given to the design in order to obtain maximum frequency of the design. Besides, by using similar method with proposed iterative design, the speed of the design can be enhanced by putting register at the end of the output port of the design. This delay contributes to register-to-register delay. In this design, performance of RIPEMD-160 pipelined design increase significantly if compared with iterative design.

Penerbit Akademia Baru

Journal of Advanced Research in Computing and Applications
ISSN (online): 2462-1927 | Vol. 3, No. 1. Pages 17-27, 2016

3.0 RESULT AND DISCUSSION

The proposed RIPEMD-160 for both iterative design and pipelined design were successfully synthesized and implemented by using Altera Quartus II on Arria II GX. The designs were written in Verilog code and simulated in functional and timing simulation using ModelSim-Altera 10.3d. Table 8 shows the synthesis and implementation results of RIPEMD-160 hash function. From this table, it shows that the maximum frequency for pipelined design increase significantly from 135.61 MHz to 250.0 MHz if compared with iterative design. In addition, Quartus II timing optimization advisor can also increase the speed of the design. Besides, the usages of register reduce from 581 to 517. This method can be done by using Quartus II resource optimization advisor.

Table 8: Synthesis and Implementation of RIPEMD-160 Hash Function

Proposed Design	Device	ALUTs	Reg	FMax (MHz)	Throughput (Mbps)
RIPEMD160	Arria II GX	1192	581	134.35	838.87
RIPEMD160_pipeline	Arria II GX	2200	517	250.0	7804.88

The throughput of the design can be calculated using the following equation (2). From this design, RIPEMD-160 pipelined design achieves high throughput which is about 7.805 Gbps with 82 number of clock cycle. Table 9 shows the comparison results of other publications of RIPEMD-160 hash function design. From this table, it shows that both proposed designs on Arria II GX provide high frequency and throughput if compared with other designs. Besides, these designs give small area implementation where it suitable for any cryptographic application that needs high speed design with small area implementation.

$$\text{Throughput} = \frac{512 \text{ x Maximum Frequency}}{\text{No.of clock Cylces}} \quad (2)$$

Table 9: RIPEMD-160 design comparison with previous works

Design	Device	ALUTs/CLB	Reg	FMax (MHz)	Throughput (Mbps)
[3]	Xilinx Virtex 300E	1004 CLB	-	42.9	65
[4]	EPF10K50SBC356-1			26.6	84
*[5]	Xilinx Virtex-E	-	-	87.6	2803
[6]	XC2VP30	4410 ALUTs	-	100.05	624
Proposed RIPEMD-160	Arria II GX	1192 ALUTs	581	134.35	838.87
*Proposed RIPEMD-160 Pipelined	Arria II GX	2200 ALUTs	517	250.0	7804.88

*Pipelined

4.0 CONCLUSION

In conclusion, both proposed designs were successfully synthesized and implemented on Quartus II Arria II GX. Pipelined design shows the improvement of the design performance in terms of maximum frequency and area implementation. Besides, high throughput can be achieved by using pipelined technique. The maximum frequency increase about 46% and the percentage increment of throughput is approximately 89% if compared with iterative design. By using pipelined design, the frequency and throughput of the RIPEMD-160 hash function design can be improved which is about 250 MHz and 7.805 Gbps respectively with small area implementation.

ACKNOWLEDGMENT

The author would like to thank for the support given to this research by Universiti Malaysia Sarawak (UNIMAS) for providing opportunity and necessary facilities to support this research work.

REFERENCES

[1] F. Rodriguez-Henriquez, N.A. Saqib, A. Diaz-Perez, C. Kaya Koc, “Cryptographic Algorithms on Reconfigurable Hardware”, Springer Series on Signals and Communication Technology, 2006. pp.211-242.

[2] H. Dobbertin, A. Bosselaers, B. Preneel, “RIPEMD-160, a strengthened version of RIPEMD”, Fast Software Encryption, LNCS 1039,Springer-Verlag, 1996, pp. 71-82.

[3] S. Dominikus,“A hardware implementation of MD4-family hash algorithms”,*Proc. 9th Int. Conf. on Electronics, Circuitsand Systems*, vol. 3, 2002, pp. 1143-1146.

[4] C. Ng, T. Ng and K. Yip,**“**A Unified Architecture of MD5 and Ripemd-160 Hash Algorithms”, Proceedings of the 2004 International Symposium on Circuits and Systems, 2004. ISCAS '04,Vol.2, ,23-26 May 2004, pp. 889-892.

[5] H.E.Michail, V.N,Thnanasoulis, D.M. Schinianakis, G.A.Panagiotakopoulos, and C.E.Goutis, Application of Novel Technique in Ripemd-160Aiming at High-Throughput, IEEE International Symposium onIndustrial Electronics, 2008. ISIE 2008, June 30 2008-July 2 2008, Cambridge, pp.1937 – 1940.

[6] M. Kneˇzevi´c, K. Sakiyama, Y. K. Lee and I. Verbauwhede,On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm,International Conference onApplication-Specific Systems, Architectures and Processors,2008. ASAP 2008, 2-4 July 2008,Leuven, pp.85 – 90.