

PAPER • OPEN ACCESS

## Power Efficiency in Unfolding RIPEMD-160: Dynamic Power Analysis Using Gray Encoding in FPGA Design

To cite this article: Shamsiah Suhaili *et al* 2025 *J. Phys.: Conf. Ser.* **3020** 012006

View the [article online](#) for updates and enhancements.

You may also like

- [The basins of attraction of the global minimizers of the non-convex sparse spike estimation problem](#)  
Yann Traonmilin and Jean-François Aujol
- [Enhanced total variation minimization for stable image reconstruction](#)  
Congpei An, Hao-Ning Wu and Xiaoming Yuan
- [High-resolution respiratory inductive plethysmography in rats: validation in anesthetized conditions](#)  
T Flénet, J Fontecave-Jallon, P-Y Guméry et al.



**UNITED THROUGH SCIENCE & TECHNOLOGY**

 **The Electrochemical Society**  
Advancing solid state & electrochemical science & technology

**248th  
ECS Meeting**  
Chicago, IL  
October 12-16, 2025  
*Hilton Chicago*

**Science +  
Technology +  
YOU!**

**Register by  
September 22  
to save \$\$**

**REGISTER NOW**

# Power Efficiency in Unfolding RIPEMD-160: Dynamic Power Analysis Using Gray Encoding in FPGA Design

Shamsiah Suhaili<sup>1\*</sup>, Norhuzaimin Julai<sup>1</sup> and Asrani Lit<sup>1</sup>, Maimun Huja Husin<sup>1</sup>

<sup>1</sup> Department of Electrical & Electronic Engineering, Universiti Malaysia Sarawak (UNIMAS), Kota Samarahan, Sarawak, Malaysia

\*E-mail: sushamsiah@unimas.my (corresponding author)

**Abstract.** The RIPEMD-160 hash functions are extensively used in many cryptographic applications, including digital signatures, Hash Message Authentication Codes (HMAC) and others. Unfolding RIPEMD-160 was designed to analyse the architecture of the design in terms of ALUTs area and design speed. This method was also applied to RIPEMD-160 designs to analyse the internal structure concerning area, maximum frequency, and throughput. The implementation of design using the unfolding transformation approach with a factor of four yields significant throughput performance. This project aims to enhance the power efficiency of the RIPEMD-160 hash function with an unfolding factor of 4 through the application of Gray encoding. The unfolding transformation factor of four approaches can increase RIPEMD-160's throughput to approximately 1753.50 Mbps. The performance-to-area ratio of RIPEMD-160, when unfolded with factor four designs, exhibits an increase of 1.51% relative to the iterative RIPEMD-160 design. The design was simulated to verify the accuracy of the RIPEMD-160 designs regarding functional and timing simulations. The dynamic power consumption of the RIPEMD-160 design using Gray encoding was reduced by 64.6% compared to binary encoding and this is attributed to the lower switching activity associated with Gray encoding.

## 1. Introduction

Various categories of cryptographic hash functions were designed in previous study[1]. When it comes to security applications, hash functions are an extremely important component. Furthermore, the RIPEMD-160 hash method is used to execute cryptocurrency transactions. Cryptocurrencies, as a form of digital currency, utilise blockchain technology, where hashes of prior blocks are included in each successive block to ensure data integrity. Thus, the RIPEMD-160 hash function is very relevant in modern electronic cash systems such as Bitcoin.

In the current landscape, securing financial transactions has become a crucial concern, whether for traditional currencies or cryptocurrencies. Various challenges arise during transactions, especially when the original data is vulnerable to unauthorised modifications by malicious users. This makes data integrity a robust and crucial security element to avoid such issues. Network security during data transmission is essential as cryptographic algorithms are required at the network layer. Efficient cryptographic hash functions are a critical element of security protocols. The objective of this research is to develop a RIPEMD-160 architecture characterised by low dynamic power consumption and high throughput through the implementation of Gray encoding. Consequently, the design framework of RIPEMD-160 is focused on minimising dynamic power usage, small area implementation and high frequency need to be analysed. The unfolding factor 4 of RIPEMD-160 was designed using Verilog HDL and implemented on FPGA. Techniques such as iterative and unfolding transformations are adopted



by the proposed design in order to attain high performance. Setup and hold times need to be met to avoid any slack violations in the timing report. This optimisation is able to maximise the frequency by reducing the longest path delays within the design. Appropriate clock constraints during the RIPEMD-160 implementation are important to achieve the desired performance. The design is also evaluated using ModelSim with various testbench files to verify its functionality and performance. Enhancing the efficiency of the RIPEMD-160 architecture is the primary motivation behind this endeavour. Additionally, the internal architecture of RIPEMD-160 is distinct from that of numerous hash functions that contain two sets of instructions for the right and left processes running in simultaneously, adding another layer of complexity due to its shift and constant values.

In contrast to other cryptographic structures, hash functions operate without a key, and the resultant output is termed a hash value, hash code, or message digest. Due to the increasing demand for high-performance designs, it is crucial to optimise the performance of hash functions concerning space, frequency, and throughput. This paper proposes a RIPEMD-160 design using unfolding transformations. Compared to conventional implementations, unfolding improves the throughput, and it depends on the unfolding factor used. Higher throughput can be attained through this technique, as it significantly increases the frequency maximum (FMax). The parallelism inherent in the unfolding transformation allows for simultaneous operations, thereby improving overall performance. Consequently, this optimised RIPEMD-160 design is well-suited for devices with abundant registers and Adaptive Logic Modules (ALUTs).

## 2. RIPEMD-160 Algorithm

The science of coding messages in secret codes, known as cryptography hash functions, ensures that only the intended user may encrypt and decrypt messages. No key is utilised for the hash function, as it represents one-way cryptography. A basic requirement of a hash function is that the output preserves a consistent length decided upon by the hash function used while the input may vary in length. It is a unidirectional function in which determining the input message  $x$  is impractical. As one of the numerous hash functions available, the RIPEMD-160 is the one selected for this research.

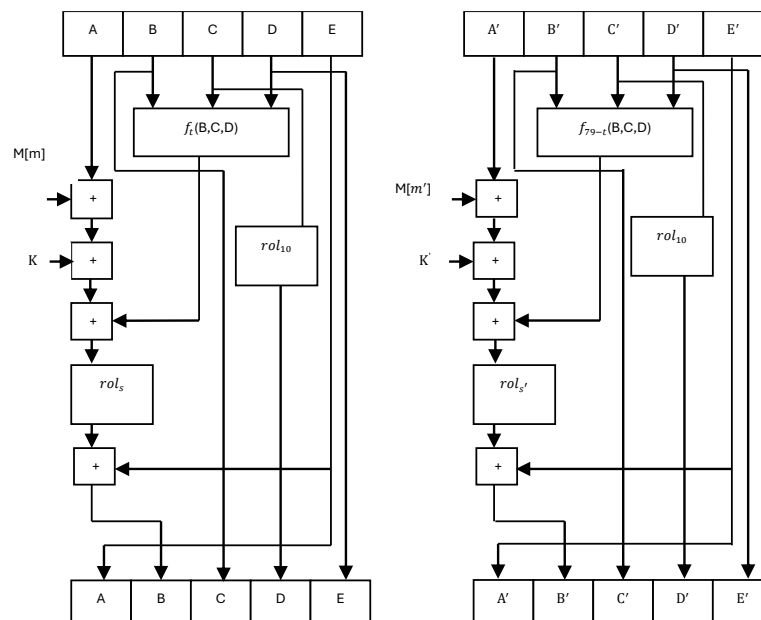
A 160-bit hash code made up of five 32-bit words is produced by the RIPEMD-160 algorithm. Listing 1 assigns five starting inputs to five left-and-five right-hand inputs. Execution of RIPEMD-160 involves two processes running in parallel. Figure 1 depicts the architecture of the RIPEMD-160 algorithm. From this figure, the function  $f(B, C, D)$  is non-linear and takes three variables  $B$ ,  $C$ , and  $D$  as inputs. A ten-place leftward shift (rotation) is applied to the outputs  $D$  and  $D'$ .

**Listing 1 : RIPEMD-160 Algorithm**

```

for t=0 {
    A = H0, B = H1, C = H2, D = H3, E = H4;
    A' = H0, B' = H1, C' = H2, D' = H3, E' = H4;
    for t=0 to 79 {
        T = rols(t)(A + ft(B, C, D) + X[m(t)] + K(t)) + E;
        A = E; E = D; D = rol10(C); C = B; B = T;

        T' = rols'(t)(A' + f79-t(B', C', D') + X[m'(t)] + K'(t)) + E';
        A' = E'; E' = D'; D' = rol10(C'); C' = B'; B' = T';
    }
    H0 = H1 + C + D'; H1 = H2 + D + E'; H2 = H3 + E + A';
    H3 = H4 + A + B'; H4 = H0 + B + C'; }
    
```



**Figure 1.** Structure of RIPEMD-160 Algorithm

Figure 2 shows the uppermost architecture of the hash function, including initial input, message selection input value, shift rotation, and constants of RIPEMD-160 algorithm. Iterative design uses one nonlinear compression function block (B, C, and D). For each of the 80 operational steps, there are 10 constant inputs: K and K' for five simultaneous operations, shift values, and M and M' as 32-bit message inputs, which signal message word. The hash function output is the sum

of five initial inputs, five left variables, and five right variables. The RIPEMD-160 hash function output is little-endian and must be converted to standard format to be accurate.

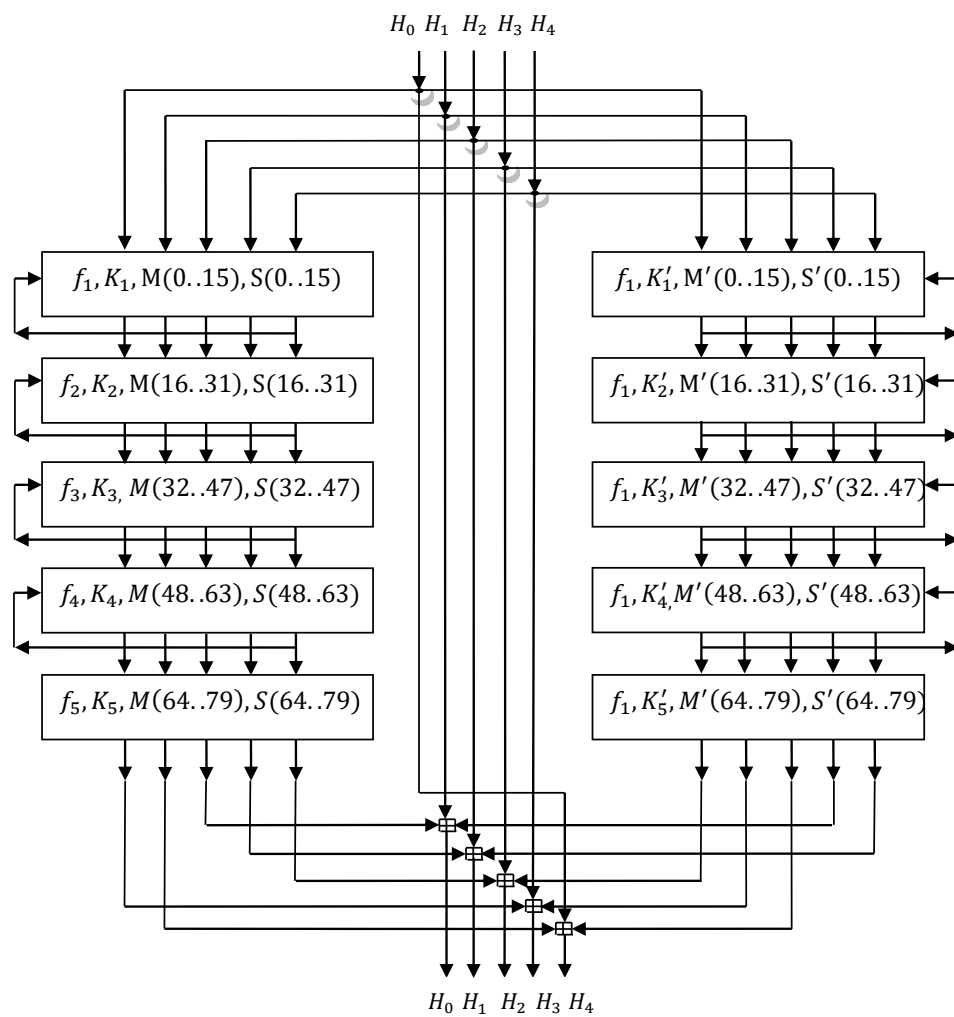


Figure 2. RIPEMD-160 Top Level Design

The subsequent tables present several operational parameters to characterise RIPEMD-160. Table 1 presents the starting input with initial inputs  $H_0$ ,  $H_1$ ,  $H_2$ ,  $H_3$ ,  $H_4$  and  $H_5$ . RIPEMD-160 employs five unique  $f(B,C,D)$  as a non-linear function.

**Table 1.** Buffer Initialisation

Buffer Initialisation	Register
32'h67452301	$H_0$
32'hefc dab89	$H_1$
32'h98badcfe	$H_2$
32'h10325476	$H_3$
32'hc3d2e1f0	$H_4$

Table 2 breaks down the non-linear functions for  $f_1$ ,  $f_2$ ,  $f_3$ ,  $f_4$ , and  $f_5$ . Table 3 illustrates the cycle of functions for right and left sides of row architecture. The cycle as stated in Table 2, will determine the order of functions where the right line will employ the reverse order of the non-linear function for the left line as shown in Table 3.

**Table 2.** Function of RIPEMD-160

Cycle	$f(B,C,D)$	
$0 \leq t \leq 15$	$B \oplus C \oplus D$	$f_1(B,C,D)$
$16 \leq t \leq 31$	$(B \wedge C) \vee (\neg B \wedge D)$	$f_2(B,C,D)$
$32 \leq t \leq 47$	$(B \vee \neg C) \oplus D$	$f_3(B,C,D)$
$48 \leq t \leq 63$	$(B \wedge D) \vee (C \wedge \neg D)$	$f_4(B,C,D)$
$64 \leq t \leq 79$	$B \oplus (C \vee \neg D)$	$f_5(B,C,D)$

**Table 3.** Cycle of RIPEMD-160 Function

Row	RoundCycle 1	RoundCycle 2	RoundCycle 3	RoundCycle 4	RoundCycle 5
Right	$f_5$	$f_4$	$f_3$	$f_2$	$f_1$
Left	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$

The design incorporates the generation of the RIPEMD-160 hash function output utilising 10 unique constants, comprising five from the left and five from the right. The constant  $K$  for the interval  $0 \leq t \leq 15$  and the constant  $K'$  for the interval  $64 \leq t \leq 79$  both possess the identical value. The five iterations of RIPEMD-160 are presented in hexadecimal notation in Table 4, along with five constants,  $K$  and  $K'$ .

**Table 4.** The RIPEMD-160 Constant

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

The data must be retained in memory first to ensure proper positioning in Table 5. RIPEMD-160 picks two message values,  $m$  and  $m'$ , using two parallel lines from the left and right. Upon the data being appropriately positioned, the counter will invoke the message according to the values from Table 5. The 16 segments contain 32-bit message words. This design has five 16-step rounds per line. The output of this RIPEMD-160 design requires 80 processing steps to finish.

**Table 5.** RIPEMD-160 Message Word

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

Table 6 displays RIPEMD-160's left and right 4-bit left rotation operation. The combination technique integrates the message value and shift value into a single Verilog variable, making output design observation easy.

**Table 6.** RIPEMD-160 Shift Value

Step number	Shift, $s$															
$0 \leq t \leq 15$	11	4	15	12	5	8	7	9	11	13	14	15	6	7	9	8
$16 \leq t \leq 31$	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12
$32 \leq t \leq 47$	11	13	6	7	14	9	13	15	14	8	13	6	4	12	7	5
$48 \leq t \leq 63$	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12
$64 \leq t \leq 79$	9	15	5	11	6	8	13	12	5	12	13	14	11	8	5	6
Step number	Shift, $s'$															
$0 \leq t \leq 15$	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
$16 \leq t \leq 31$	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
$32 \leq t \leq 47$	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
$48 \leq t \leq 63$	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8
$64 \leq t \leq 79$	8	5	12	9	12	5	14	6	8	13	6	5	15	13	11	11

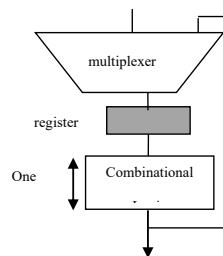
### 3. Proposed RIPEMD-160 Design Method

RIPEMD-160 consists of left and right components. This makes two parallel structures operate concurrently during the implementation of the RIPEMD-160 design. Several studies on RIPEMD-160 hash functions have been identified [3 – 18]. The recent advancement of RIPEMD-160 and the introduction of Mixed Integer Linear Programming (MILP)-based methods have enhanced the identification of differential traits, thereby increasing the efficacy of these attacks [19]. Most implementations of RIPEMD-160 utilise iterative architectures, which however present a drawback in terms of the hash function's low throughput performance, despite its efficient area implementation. The implementation of pipelining schemes generally necessitates significant

area utilisation. This fact hence explains the importance of the RIPEMD-160 hash function in reducing its cycle count using the unfolding technique.

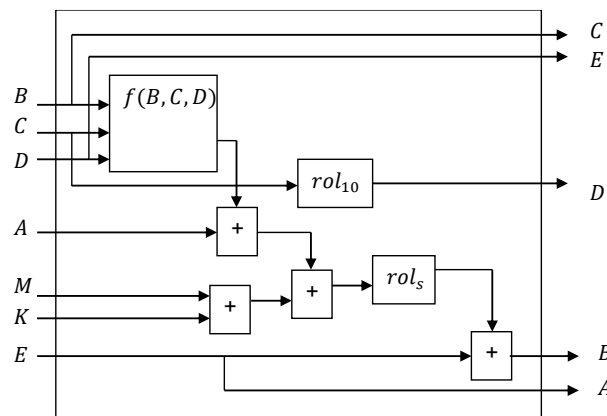
### 3.1 Iterative RIPEMD-160 design

Iterative processes were commonly employed in the conventional design of hash algorithms. Figure 3 depicts the block diagram of iterative step function iteration. Iterative design only uses one function from combinational design until the hash computation is finished after 80 rounds. Iterative design offers the advantage of compact area implementation, but it has a low throughput because the RIPEMD-160 hash function must operate for 80 rounds. Consequently, the latency of the design is decreasing employing unfolding transformation approaches, depending on the number of factors used. In this research, the RIPEMD-160 architecture is transformed using unfolding factor 4. Consequently, the reduced unfolding design latency allows for excellent throughput of the RIPEMD-160 architecture. The inner structure of the step function of the iterative design of RIPEMD-160 is depicted in Figure 3.



**Figure 3.** Iterative Looping [1]

Figure 4 shows that iterative design uses standard operations and is the basis of the hash function. From this figure, shift rotation across 10 positions, a non-linear function, and a left circular shift with a defined shift value are all included. The message is denoted by  $M$ , while the constant of RIPEMD-160 is  $K$ . Throughout the execution process, this design will only require a single block of step functions. This design will employ all the parameters specified in the preceding section to ensure that the desired output is achieved. The design's maximum frequency can be attained by utilising advice regarding resources, timing, and power.

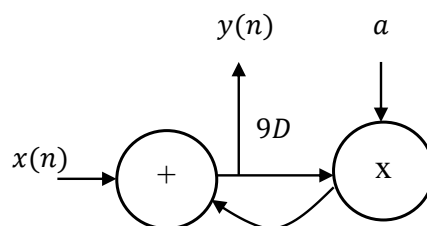


**Figure 4.** Non-Linear Function (Iterative)

Iterative design uses one non-linear function block with five functions. Messages are initially stored in memory with input load. This indicates that data will be sent to a register of a particular length in the event that the input load is logic '1'. The message will then be remembered for a period of 15 locations, beginning at 0. This method simplifies the process of calling the message in accordance with the order of message selection, as illustrated in the preceding table. Counters are utilised in the execution of the Shift,  $s$ ,  $s'$  and message  $m$ ,  $m'$  values. A total of 80 stages were executed during the iteration of this architecture. At the end of the design, the outputs of both will be mixed with the starting values  $H_0$ ,  $H_1$ ,  $H_2$ ,  $H_3$ , and  $H_4$ .

*3.2 Unfolding transformation of RIPEMD-160 design*

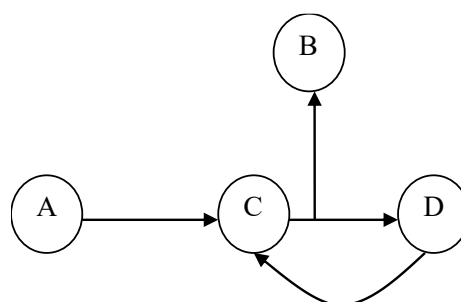
This design uses factor two and factor four unfolding transformations. Digital Signal Processing (DSP) applications use the unfolding algorithm to create a new programme that performs numerous repetitions of the original.  $J$ , the unfolding factor, indicates the number of programme iterations. The rules of the unfolding algorithm are stated in this example[2]. Equation 1 and Figure 5 illustrate an example of a DSP program that explains the structure of the unfolding algorithm.



**Figure 5.** The original DSP program example[2]

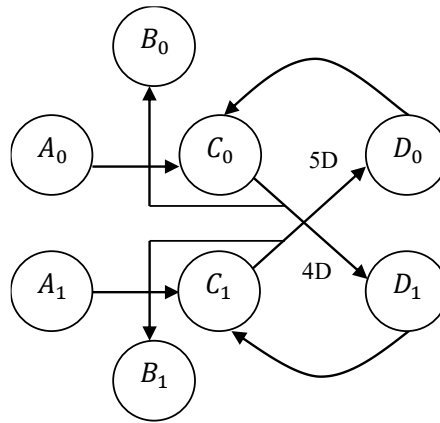
$$y(n)=ay(n-9)+x(n) \tag{1}$$

The Data Flow Graph (DFG) shows the original DSP program. Figure 6 depicts this configuration with input and output ports at nodes A and B and addition and multiplication at nodes C and D.



**Figure 6.** A DFG corresponding to the DSP programme [2]

According to the first rule of the unfolding algorithm, 8 nodes that represent  $i=0,1$ :  $A_0, B_0, C_0, D_0, A_1, B_1, C_1,$  and  $D_1$ . The unfolding algorithm's second phase involves connecting each edge ( $U \rightarrow V$ ) to the DSP programme. The edge  $U \rightarrow V$  without delay is broken into two parts:  $U_0 \rightarrow V_0$  and  $U_1 \rightarrow V_1$ . As a result, the edge  $C \rightarrow D$  with  $\omega=9$  delays become  $C_0 \rightarrow D((9+0)\%2)$  with  $[(9+0)/2]$  delays and  $C_1 \rightarrow D_1((9+1)\%2)$  with  $[(9+1)/2]$  delays. Figure 7 depicts the unfolded DFG corresponding to the 2-unfolded DSP programme. Finally, the 2-unfolded DFG is generated with four and five delays, respectively.



**Figure 7.** The Unfolded DFG According to the 2-Unfolded DSP Programme[2]

The use of the unfolding transformation method can reduce the number of cycles. By going from unfolding factor 2 to unfolding factor 4, RIPEMD-160 is able to achieve better throughput, which is an improvement in performance. The utilisation of the unfolding transformation technique has the potential to improve the performance of iterative design methodology. The unfolding design factor is significant in the development of unfolding transformation. The implementation of factor four requires a reduction of design cycles, leading it to necessitate a more complex algorithm compared to factor two. The following equations represent the architecture of Figure 1 for subsequent input for each register.

$$T = rol_{s(t)}(A + f_t(B, C, D) + M[m(t)] + K(t)) + E \tag{2}$$

$$A = E \tag{3}$$

$$E = D \tag{4}$$

$$D = rol_{10}(C) \tag{5}$$

$$C = B \tag{6}$$

$$B = T \tag{7}$$

$$T' = rol_{s'(t)}(A' + f_{79-t}(B', C', D') + M[m'(t)] + K(t)) + E' \tag{8}$$

$$A' = E' \tag{9}$$

$$E' = D' \tag{10}$$

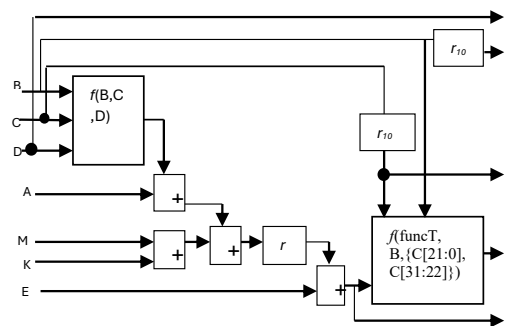
$$D' = rol_{10}(C') \tag{11}$$

$$C' = B' \tag{12}$$

$$B' = T' \tag{13}$$

### 3.2.1 Unfolding factor 2 of RIPEMD-160 design

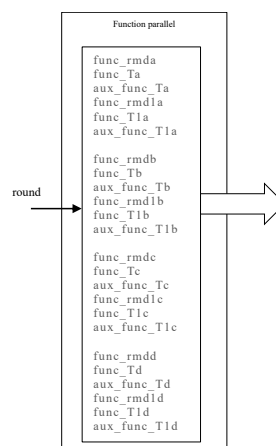
Unfolding design is a technique employed to derive a modified architecture that executes several iterations of the original program. RIPEMD-160's step function unfolding factor 2 design is shown in Figure 8. It has two non-linear functions, two ten-place shift rotations, and a left circular shift with a defined value.  $M$  denotes the message, while  $K$  represents the constant of RIPEMD-160. There are two sets of five functions in the unfolding design's non-linear function blocks. When you input data, the original message is stored in memory. The data will be transferred to a specific register length if the input load is logic high. Memory will hold the message from 0 to 15. This method helps retrieve messages in order of selection. The counter calls message and shift values  $m, m'$  and  $s, s'$ . Due to the unfolding factor 2 design, this architecture will be processed 40 times.



**Figure 8.** Unfolding factor 2 of RIPEMD-160 Design

### 3.2.2 Unfolding factor 4 of RIPEMD-160 design

For unfolding factor 4 design, alteration at the top-level design of RIPEMD-160 must incorporate the unfolding transformation with a factor of four. This approach reduces the cycle count in the RIPEMD-160 architecture from 40 to 20 rounds. This action will directly decrease the latency of the design, and resulting in a high throughput of the RIPEMD-160 architecture. Modifications must be made for some modules, including Coder, KConst, Message, and Function parallel, just like in the preceding section that deals with factor two.



**Figure 9.** Unfolding RIPEMD-160 factor four: Parallel Function

The function parallel module of RIPEMD-160 unfolding factor four is illustrated in Figure 9. This module contains non-linear functions based on the RIPEMD-160 algorithm. The non-linear function must be increased to four in order to produce an unfolding transformation with factor four. In other words, four concurrent non-linear functions are generated simultaneously to provide the result in a single cycle. Three distinct inputs of a non-linear function must be identified. The functions `func_rmda`, `func_rmd1a`, `func_rmdb`, and `func_rmd1b`, `func_rmdc`, `func_rmd1c`, `func_r added`, and `func_rmd1d` denote the non-linear functions operation for both the left and right sides, respectively. `Func(B,C,D)` illustrates the three distinct inputs for both the left and right paths of four non-linear functions inside this module.

This work presents a novel concept regarding the unfolding transformation that occur by a factor of four. All non-linear functions need to be carried out through four concurrent transformations, and the details of this are given below:

1. Compute the first set of non-linear functions:
  - `func_rmda(B, C, D)`
  - `func_rmd1a(B1, C1, D1)`
2. Compute the second set of functions, using shifted values of C and C1:
  - `func_rmdb(aux_func_Ta, B, {C[21:0], C[31:22]})`
  - `func_rmd1b(aux_func_T1a, B1, {C1[21:0], C1[31:22]})`
3. Compute the third set of functions, using the results of the previous step:
  - `func_rmdc(aux_func_Tb, aux_func_Ta, {B[21:0], B[31:22]})`
  - `func_rmd1c(aux_func_T1b, aux_func_T1a, {B1[21:0], B1[31:22]})`
4. Compute the fourth set of functions, again using the results from the previous step:
  - `func_r added(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]})`

### 3.3 Dynamic Power Estimation Analysis

This paper discusses the proposed unfolding factor 4 of the RIPEMD-160 design, emphasising low-power design implementation. A technique to reduce dynamic power is through the management of switching activity.

**Table 7.** State Encoding or Gray and Binary Encoding

Binary	Gray	State
000	000	S0
001	001	S1
010	011	S2
011	010	S3
100	110	S4
101	111	S5
110	101	S6
111	100	S7
3	1	Highest possible transition rate per clock cycle

This paper introduces a state encoding technique for managing switching activity in low power design to reduce power consumption in the unfolding of RIPEMD-160, utilising Verilog code. Table 7 illustrates an example of encoding transitions for eight states utilising Binary and Gray encoding methods. The table indicates that the maximum transitions per clock cycle for Gray encoding is one, while for Binary encoding, it is three. This occurs because, in the transition, Gray encoding alters only a single bit for each state. This happens due to the fact that, during the transition, Gray encoding modifies only one bit for each state. Dynamic power is generated when a signal toggles between bit 0 and bit 1 and vice versa. Gray encoding alters only a single bit when transitioning from one state to another within a cycle. Consequently, it has the potential to decrease the dynamic power of the design.

Dynamic power estimation in Quartus II encompasses a variety of methodologies and models, all of which responsible in predicting the power consumption of FPGA designs. These techniques can improve power efficiency and ensure the reliability of semiconductor devices. Strategies ranging from statistical models to machine learning approaches undeniably provide distinct benefits in terms of accuracy and speed. Incorporating these techniques into tools like Quartus II is seen to enhance the design process by offering early and accurate power estimates. Furthermore, this method can also be used to reduce power consumption in constructing a Finite State Machine(FSM).

Gray encoding is a method in digital systems that is responsible for mitigating switching activity involving the frequency of bit alterations between successive states. Optimal power efficiency can be achieved as Gray encoding ensures that only one-bit changes at a time. This directly reduces the power consumed during state transitions. Practically, this technique is applied in various contexts, such as state assignment in sequential circuits, microcontroller address buses, and high-performance processors.

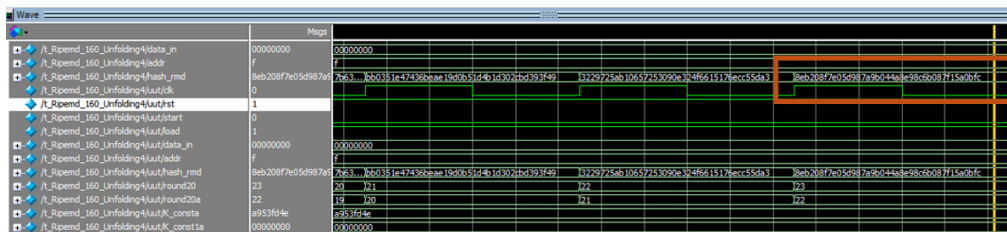
Gray encoding is generally known for its ability to minimise switching activity, hence directly lowering dynamic power consumption in digital circuits. But in FPGA-based cryptographic solutions, its advantages go beyond only power economy to include important stability and dependability. Gray encoding guarantees that just one-bit changes at a time, unlike binary encoding whereby several bits can flip simultaneously. This behaviour greatly lowers signal faults, thereby minimising unneeded power dissipation and increasing circuit dependability. Furthermore, Gray encoding guarantees consistent data transitions, therefore preventing common in high-frequency cryptographic circuits timing mistakes and metastability problems. Besides, Gray encoding reduces bit toggling in FSMs hence producing smoother state transitions and reduced power consumption in sequential logic circuits. Gray encoding reduces EMI problems, thereby strengthening cryptographic FPGA solutions against electromagnetic emissions and power threats.

With the growing demand for low-power cryptographic solutions, Gray encoding is critical for maximising energy efficiency while preserving security. Some critical areas where this technology is especially useful, such as IoT devices, rely on cryptographic hashing for secure authentication and encrypted communication. However, because these gadgets are generally battery-powered, every milliwatt saved increases device life. Besides, reduces power consumption in FPGA-based cryptographic accelerators, making secure IoT deployments more energy efficient. Mining cryptocurrencies such as Bitcoin requires high-speed hashing (SHA-256, RIPEMD-160), which consumes a significant amount of energy. Reduces switching activity in FPGA-based hashing cores, lowering heat dissipation and power costs while preserving computational integrity. Gray encoding is more than just a power-saving strategy; it is an essential

enabler of stability, security, and efficiency in cryptographic FPGA designs. Its ability to reduce glitches, EMI noise, and power spikes makes it suitable for IoT security, blockchain transactions, smart payment systems, and safe hardware cryptography.

#### 4. Result and Discussion

This project developed an iterative Verilog HDL-implemented, unfolding RIPEMD-160 algorithms implemented by a factor of four. The RIPEMD-160 designs were tuned on the Arria II GX to determine area, maximum frequency, and throughput. Quartus II advisors improve design performance. The TimeQuest Timing Analyser is used to determine the maximum frequency of the RIPEMD-160 design. The clock limits on the designs meet the timing requirements, and this method is repeated until the RIPEMD-160 design meets timing criteria. This effort gave the RIPEMD-160 unfolding a four-fold 8 ns clock limitation. All RIPEMD-160 designs meet timing criteri



**Figure 10.** Functional Simulation Waveform of RIPEMD-160 Unfolding Factor 4

##### 4.1 Simulation Waveform of RIPEMD-160 Unfolding transformation

Figures 10 and 11 show the simulation waveform results of proposed RIPEMD-160 designs. The designs undergo evaluation based on the parameters of area, maximum frequency, and throughput. The design throughput is determined using Equation (14). The latency for iterative design is 101.50. The latency for unfolding with a factor two design is 60.5. Ultimately, by implementing the factor four design, the latency or total number of clock cycles is reduced to 39.5. The clock cycle count varies for each design due to the structural characteristics of the RIPEMD-160 architecture.

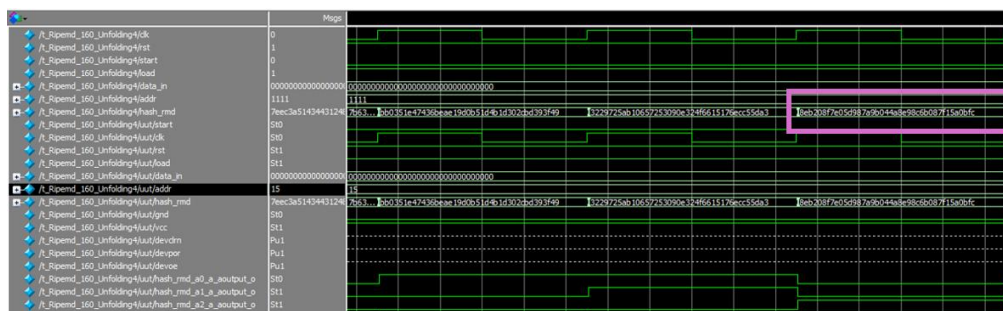
$$\text{Throughput} = (512 \times \text{Maximum Frequency}) / (\text{Number of clock cycles}) \tag{14}$$

According to the throughput formula, 512 represents the total bits of the message block. "abc" is the input message in this simulation result. The first step is to convert the input message into ASCII code, where the value 616263 in hexadecimal represents the letter "abc." Two process type such as pre-processing and hash computation must be considered in order to obtain 512-bit block message input. Upon receiving the message "abc," a single 1-bit is appended at the end, succeeded by 423 zero bits. The final 64-bit block message is then set reserved for message length. The final 64-bit of the entire message block contains the overall length of the message input "abc". Finally, the overall message block size is 512 bits

To determine the maximum frequency value (FMax), it is necessary to extract this information from Quartus II following the compilation of the design during the synthesis and place and route phases. This FMax need to be checked whether it meets the timing requirement

of the design or not. If a negative slack value for setup and hold time is identified, it is necessary to incorporate a clock constraint SDC file utilising the TimingQuest Timing Analyser. This measure is implemented to prevent any violation of the design specifications. The value for latency of design in equation 14 is obtained from the number of cycles of design from the simulation waveform. The latency of design has been previously discussed in the first paragraph of this section.

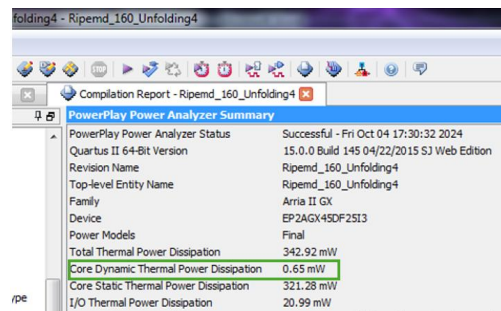
Figure 10 shows the functional simulation waveform of RIPEMD-160 with an unfolding factor of 4. The "abc" input message produces a hash value of "8eb208f7305d987a9b044a8e98c6b087f15a0bfc". The result clearly shows that the RIPEMD-160 hash output is valid. The design was simulated using ModelSim. This value can be verified using HashCalc software. The implementation of RIPEMD-160 design was based on the FPGA device.



**Figure 11.** Timing Simulation of RIPEMD-160 Unfolding Factor 4

This design was implemented on FPGA device; therefore, gate-level simulation must be considered. For timing simulation, some libraries and SDF files were integrated with the design and testbench files of the design. The timing simulation is displayed in Figure 11, where the result of hash function output is like the functional simulation of RIPEMD-160 unfolding factor 4. This timing simulation waveform clearly shows the signal wiring that considers the time of the design. Typically, if the design encounters issues with delay and timing due to the implementation on FPGA family device, the final output will yield incorrect values, as the timing and delay of the design will have altered after operating through several cycles of iteration. Timing simulation is therefore necessary to make sure that the design, which takes timing delay into account for each module of the design, embeds correctly into the FPGA device. The significance of clock cycles is essential as Equation 14 utilises latency to compute the throughput of the design. High throughput of the design can be achieved if the latency is low. As a result, the RIPMD-160 unfolding factor 4 offers higher throughput in comparison to traditional iterative designs. The reason for this is that the number of cycles decreases from 80 cycles in the iterative design to only 20 cycles for unfolding factor 4 in the RIPEMD-160 design.

Table 8 summarises the outcomes of the synthesis and implementation of the RIPEMD-160 design. The three RIPEMD-160 design types such as RIPEMD-160 with iterative, RIPEMD-160 with unfolding factor 2, and RIPEMD-160 with unfolding factor 4 method are illustrated in this table. This table presents results regarding area, namely ALUTs and registers, design speed which is maximum frequency (FMax), and throughput of RIPEMD-160 designs. The designs were



**Figure 12.** PowerPlay Power Analyser RIPEMD-160 Unfolding Factor 4

**Table 8.** Results of synthesis and implementation 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

executed on the Arria II GX device, with the number of registers, ALUTs, and configurable logic blocks for each design reflecting the area implementation of the design.

The unfolding factor 4 design of RIPEMD-160 clearly enhances the speed of the design. Thus, RIPEMD-160 unfolding factor 4 having a throughput of 1753.50 Mbps. This is based on throughput equation 14 where the number of design cycles significantly influences throughput performance. RIPEMD-160 unfolding factor 4 has fewer cycles than iterative design. As a result, the RIPEMD-160 unfolding factor 4 throughput greatly increased. As previously stated, 80 rounds of iteration are required to complete the iterative design of the RIPEMD-160. All RIPEMD-160 designs in this project use Arria II GX for synthesis and execution. The example of low-cost FPGA family devices include Cyclone, while Stratix, represents a high-performance of FPGA. Therefore, to achieve a balance between power and performance, the Arria II GX device has been selected, as it offers a cost-effective transceiver for FPGA devices as well as superior performance and energy efficiency. The throughput of the RIPEMD-160 architecture is significantly improved due to the design latency.

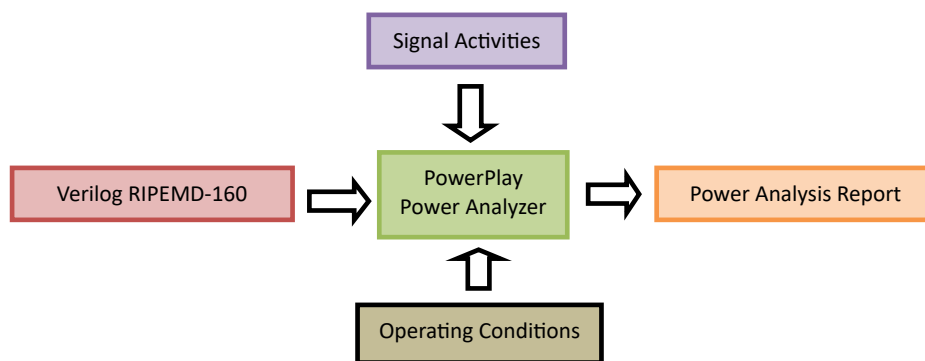
Figure 12 displays the Altera Quartus II PowerPlay Power Analyser simulation-based power estimation for the RIPEMD-160 Unfolding factor 4 design. This is a power analysis program that offers the most precise power estimation using simulation. The dynamic thermal power

dissipation in this figure results from signal toggling. Figure 12 shows the summary result of powerplay power analyser. From this figure, dynamic thermal power dissipation gives  $0.65\text{ mW}$ .

Figure 13 depicts the procedure of the PowerPlay Power Analyser. There are three inputs such as signal activity, Verilog RIPEMD-160 design, and operating conditions. The vcd file that was acquired during the RIPEMD-160 unfolding factor 4 design simulation procedure is called signal activities. The VCD file must be combined with the operating conditions and the Verilog RIPEMD-160 design to provide the power analysis report. Altera offers the most precise and thorough PLD power analysis and optimisation tools available in the industry.

#### 4.2 Simulation Waveform of RIPEMD-160 Unfolding transformation Gray Encoding

The clock cycle count varies for each design due to the structural characteristics of the RIPEMD-160 architecture. The analysis of Equation (14) indicates that the number of cycles for the RIPEMD-160 unfolding factor two is reduced from 80 cycles to 40 cycles, leading to an improvement in the throughput. The RIPEMD-160 unfolding factor will decrease by four, resulting in a reduction in cycles from 80 to 20. The optimisation of a restricted number of cycles can improve both the throughput and overall performance of the design.



**Figure 13.** Power Analysis of PowerPlay Power Analyser

The focus of this research is on the power efficiency of the RIPEMD-160 unfolding factor 4 design, specifically dynamic power analysis. Dynamic power emphasises the switching activity of the RIPEMD-160 unfolding factor 4. The Gray encoding approach was utilised in the counter of the RIPEMD-160 design to alter only one bit per state during transitions, whereas binary encoding can have a maximum of three transitions each clock cycle. This approach was utilised for every state transition in the RIPEMD-160 design to reduce dynamic power consumption. As stated in the previous section, the vcd file was required for the power analysis. In order to determine whether the hash value of RIPEMD-160 provides the proper result after applying Gray encoding state, functional and time simulations of the design are observed after the RIPEMD-160 design has been compiled. The vcd file was obtained following the simulation. This vcd file must be incorporated into the design to facilitate the power analysis procedure using the PowerPlay Power Analyser.



Figure 16 shows the result from PowerPlay Power Analyser of dynamic thermal power dissipation. From this figure, dynamic power dissipation is 0.23 mW, which is reduced by 64.6% compared to dynamic power of RIPEMD-160 unfolding factor 4 based on binary encoding. This significant reduction is associated with lower switching activity in the Gray encoding state. This analysis uses dynamic thermal power dissipation as the power consumption result because the design focuses on the RIPEMD-160 unfolding factor 4 switching activity. It is clear that the RIPEMD-160 unfolding factor 4 with Gray encoding achieves low dynamic power consumption, which is beneficial in obtaining high performance of the RIPEMD-160 hash function design. There are numerous methods to lower power usage. Digital design can use low power FPGA design approaches such as clock and signal gating, power-efficient data routes, pipelining and retiming, state machine encoding, and more advanced techniques such as delay balancing, HDL coding, and time multiplexing.

Gray encoding is a technique employed to minimise dynamic power consumption by relying on switching activity. As a result, Table 7 gives a more precise maximum bit change for both binary and grey encoding. The switching activity of state encoding in the RIPEMD-160 counter module influences the design and enhances power efficiency. Gray encoding reduces the amount of logic switching at each clock edge and lowers the average logic-switching frequency. As a result, the propagation delay of switching activity is reduced.

#### *4.3 Performance Evaluation*

Table 9 shows the synthesis and implementation results of the three proposed RIPEMD-160 designs. Other previous RIPEMD-160 designs and other types of hash function are also shown in this table for comparison. They are evaluated from the viewpoints of the area, maximum frequency, and throughput of the design. To verify the power efficiency of the proposed RIPEMD-160 design with Gray encoding, the result is compared with current FPGA-based cryptography implementations.

**Table 9.** Results of Synthesis and Implementation Results of Hash Function Design

Hash Function Design	Device	ALUTs/CLBs	Reg	FMax (MHz)	Throughput (Mbps)	Dynamic Power
RIPEMD-160 (iterative)	Arria II GX	1,269 ALUTs	677	133.03	671.05	-
RIPEMD-160 (Unfolding-2)	Arria II GX	2,015 ALUTs	530	125.19	1,059.46	-
RIPEMD-160 (Unfolding-4)	Arria II GX	3,386 ALUTs	851	135.28	1,753.50	(Gray Encoding) 0.23 mW
RIPEMD-160 (iterative) [4]	Xilinx Virtex 300E	1,004 CLBs	-	42.9	65	-
RIPEMD-160 (iterative) [5]	EPP10K50SBC356-1	-	-	26.6	84	-
RIPEMD-160 (iterative) [6]	XC2VP30	4,410 ALUTs	-	100.05	624	-
RIPEMD-160 (iterative) [7]	XC2V250	14,911 LUTs	-	43.47	137.4	-
SHA-3 (pipeline) [20]	Virtex 7	1150 Slice	-	478	11472.0	Total power 157 mW
SHA-3 (pipeline) [21]	Virtex 5	-	-	347.49	8340	-
SHA-256 [22]	Xilinx Zynq 7000 : XC7Z020	6367 Slice	-	181	917	Total power 0.1 W
SHA-256 [23]	Spartan-7 5CSXFC6	878 LUTs	-	139.04	1040	Total power 0.072 W 10 mW (10 MHz Spartan)
SHA-256 [24]	StratixIII-EP3SE50F780I4L	2378 Logic resource	-	143.85	700.546	Total power 31.64 mW
SipHash [25]	Xilinx's Zynq-7000	2355	1395	214.3	13180	-
SPONGENT-88 [26]	xc3s50-5cp132	143 LUTs		227	29.32	0.81 mW
LHash-96[26]		380 LUTs		97	61.84	1.64 mW
Rescue-PrimeA [27]	Virtex Ultrascale+ xcu250-figd2104-2L-e.	128,519 LUTs	-	100.21	17.84 Kops/s	Total Power 6.42 mW
Lyra2REv2 [28]	ZYNQ ULTRASCALE+ MPSOC 9EG	6138 LUTs	8321	225	31.25 MHash/s	25 W

The table 9 shows dynamic power usage and power savings for various hash function implementations. The throughput of the design can be calculated using the same equation. The RIPEMD-160 design's structure results in a variation in the number of clock cycles for each design. The number of clock cycles for iterative design is 101.50. The number of clock cycles is 60.5 for unfolding with a factor 2 design. Ultimately, the number of clock cycles decreased to 39.5 by employing a factor 4 design.

There are several other FPGA family devices, such as Stratix for high-performance FPGAs and Cyclone for low-cost FPGAs. Hence, to obtain balance of power and performance, Arria II GX device was chosen because Arria series provide a low-cost transceiver of FPGA device. Besides, it also delivers optimal performance and power efficiency. The proposed RIPEMD-160 design can increase the frequency maximum of the design. The RIPEMD-160 design necessitates the use of a greater number of registers, which results in an increase in the maximum frequency. By using unfolding and following the guidelines of writing better HDL coding, the frequency of the design can be improved significantly. Besides, the architecture of FPGA device also plays important roles in the RIPEMD-160 design. By selecting an appropriate FPGA device, the performance of MD5 design can be enhanced. Table 9 shows the previous implementation of RIPEMD-160 design. It is difficult to find the same device for the same design due to the limitation of budget and devices.

From this table, the area implementation of the design increases from iterative design to unfolding design. However, the throughput of the RIPEMD-160 unfolding with factor four design increases significantly. Furthermore, this design shows the highest throughput in comparison with other RIPEMD-160 designs. The number of cycles for RIPEMD-160 unfolding factor 2 decreased 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 decreased by four, in which the number of cycles decreased from 80 cycles to 20 cycles. The design's throughput and performance can be enhanced by a small number of cycles.

Notable for its usage in combination with SHA-256, the ISO/IEC standard RACE Integrity Evaluation Message Digest 160 (RIPEMD-160) hash function is notable in the production of Bitcoin addresses. Despite the emergence of different collision attacks, RIPEMD-160 has proven resilient, with the most effective attacks only completing a fraction of its 80-step procedure. Furthermore, RIPEMD-160 employs a double-branch structure, which strengthens its security against collision attacks [20]. It uses a combination of logical functions and permutations to generate a 160-bit hash output after 80 steps of processing data. In recent development of RIPEMD-160, the emergence of mixed integer linear programming (MILP)-based approaches has facilitated the detection of differential traits, making these attacks more successful [20]. Even though RIPEMD-160 is still a strong hash function, the continued study of collision attacks emphasises how important it is to continuously assess cryptographic standards, particularly as computing power increases. Table 10 lists a compilation of RIPEMD-160 hash function in different fields.

**Table 10.** Compilation of RIPEMD-160 Hash Function in Different Applications

No.	Hash Function	Overview	Finding	Reference
1	RIPEMD-160	The method utilises MILP for the search of signed differential features.	Explores RIPEMD-160 hash function collision attacks. Strongly improves collision and semi-free-start attacks.	[29]
3	RIPEMD-160	Recommended pre-image attacks on RIPEMD-160 and HAS-160. Attacks enhanced through intermediary measures and message padding.	Pre-image attacks have been proposed against RIPEMD-160 and HAS-160. Both of the hash functions have had their attack steps improved.	[30]
4	RIPEMD-160	Design on RIVYERA S6-LX150 performs well. Compared to PC and GPU, FPGA design saves energy.	Attained 245,000 passwords per second on multi-FPGA. FPGAs used 77 times less energy than PCs.	[31]

## 5. Conclusion

The RIPEMD-160 architecture synthesis was successfully implemented using Verilog code generated on Altera Quartus II. The functional and timing simulations were also conducted using ModelSim to validate the designs. The hash function produces the output value "8eb208f7e05d987a9b044a8e98c6b087f15a0bfc" in the waveform simulation, corresponding to the input message "abc". The power efficiency of the RIPEMD-160 unfolding factor 4 is considerably reduced by employing the Gray encoding approach. Gray encoding reduces the dynamic power consumption of RIPEMD-160 unfolding factor 4 design by 64.6% as compared to binary encoding. The high throughput of the architecture results in the high performance of RIPEMD-160 with an unfolding factor of 4. Thus, employing Gray encoding, RIPEMD-160 achieves high performance with unfolding factor 4.

The unfolding transformation approach can reduce the number of cycles while simultaneously enhancing the maximum frequency of the design. In the future, low dynamic power RIPEMD-160 architecture can be applied to any digital design to improve security design performance. Unfolding transformation and Gray encoding can be utilised in various algorithms or digital designs involving parallel operations, hence minimising timing processes to enhance design performance. The enhancement in digital design can also be achieved by focusing on HDL coding style, advisor in Altera for area implementation and clock constraint based on TimeQuest Timing Analyser. It is challenging to lower power consumption on FPGA designs due to certain design restrictions. The growing demand for low-power, high-performance cryptographic solutions indicates that the implementation of Gray encoding in FPGA-based cryptographic

accelerators will be increasingly vital in the advancing digital security environment. Future efforts may include further FPGA optimisation strategies, such as clock gating, pipelining, and power-aware resource scheduling, to enhance power efficiency and enhance cryptographic performance.

## References

1. F. Rodriguez-Henriquez, N.A. Saqib, A. Diaz-Perez, C. Kaya Koc (2006), Cryptographic Algorithms on Reconfigurable Hardware, Springer Series on Signals and Communication Technology, pp. 211-242.
2. K.K.Parhi, VLSI Digital Signal Processing Systems: Design and Implementation. John Wiley & Sons, 1999.
3. H. Dobbertin, A. Bosselaers, B. Preneel (1996), RIPEMD-160, a strengthened version of RIPEMD, Fast Software Encryption, LNCS 1039, Springer-Verlag, pp. 71-82.
4. S. Dominikus (2002), A hardware implementation of MD4-family hash algorithms, Proceeding 9th International Conference on Electronics, Circuits and Systems, vol. 3, pp. 1143-1146.
5. C. Ng, T. Ng and K. Yip (2004), A Unified Architecture of MD5 and Ripemd-160 Hash Algorithms, Proceedings of the 2004 International Symposium on Circuits and Systems, ISCAS'04, vol.2, pp. 889-892.
6. M. Knežević, K. Sakiyama, Y. K. Lee and I. Verbauwhede (2008), On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm, International Conference on Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008, Leuven, pp.85 – 90.
7. E. Khan, M Watheq El-Kharashi, F.Gebali, and M. Abd-El-Barr (2017), Design and Performance Analysis of a Unified, Reconfigurable HMAC-Hash Unit, IEEE Transaction on Circuits and Systems, pp. 2683-2695.
8. X. Wang, X.Lai, D.Feng, H.Chen, and X.Yu (2005), Cryptanalysis of the hash functions and MD4 and RIPEMD. Advances in Cryptology, EUROCRYPT 2005.
9. F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen (2006), On the Collision Resistance of RIPEMD-160, International Conference on Information Security, ISC 2006, LNCS 4176, pp. 101-116.
10. H. E Michail, A. Gregoriades, V. Kelefouras, G. Athanasiou, A. Kritikakou, C. Goutis (2020), Authentication with RIPEMD-160 and Other Alternatives: A Hardware Design Perspective, New Advanced Technologies, book chapter, pp. 103 – 124.
11. I. Giechaskiel, C. Cremers, K. B. Rasmussen (2018), When the Crypto in Cryptocurrencies Breaks: Bitcoin Security under Broken Primitives, IEEE Security & Privacy Journal, Vol. 16, Issue 4, pp. 46 -58
12. A. Abbas, R. Voß, L. Wienbrandt, M. Schimmler (2014), An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs, 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 454 – 461
13. A. Kuznetsov, K. Shekhanin, A. Kolhatin, D. Kovalchuk, V. Babenko, I. Perevozova (2019), Performance of Hash Algorithms on GPUs for Use in Blockchain, 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT), pp. 166 – 170.
14. H. Michail, A. Gregoriades, V. Kelefouras, A. Kotsiolis, D. Papagianopoulou, C. Goutis (2010), HW/SW co-Design Integrating High – Speed Authentication Module for IPsec/IPv6, 2010 Fifth International Conference on Digital Telecommunications, pp. 138 -142.
15. Y. K. Lee, H. Chan, I. Verbauwhede (2008), Design Methodology for Throughput Optimum Architectures of Hash Algorithms of the MD4-class, Journal of Signal Processing Systems, 53, pp. 89–102.
16. M. M. Al-Mhadawi, A. A. Albahrani (2019), Hybrid Method as Pseudo-Random Bits Generator, 2019 International Conference of Computer and Applied Sciences (CAS2019), pp. 250-255.
17. A. S. Dewi, H. Setiawan, (2019), Implementation of SHA-256 and AES-256 for Securing Digital Al Quran Verification System, 2019 Fourth International Conference on Informatics and Computing (ICIC).
18. 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 2019 - Proceedings, 166–170.
19. Li Y, Liu F, Wang G (2023), Automating Collision Attacks on RIPEMD-160, IACR Transactions on Symmetric Cryptology, Vol. 2023, No. 4, pp 112 – 142.
20. A. Sideris and M. Dasygenis, “Enhancing the Hardware Pipelining Optimization Technique of the SHA-3 via FPGA,” *Computation*, vol. 11, no. 8, Aug. 2023, doi: 10.3390/computation11080152.
21. A. Sideris, T. Sanida, and M. Dasygenis, “Hardware acceleration design of the SHA-3 for high throughput and low area on FPGA,” *J Cryptogr Eng*, vol. 14, no. 2, pp. 193–205, Jun. 2024, doi: 10.1007/s13389-023-00334-0.
22. M. Kammoun, M. Elleuchi, M. Abid, and M. S. BenSaleh, “FPGA-based Implementation of the SHA-256 Hash Algorithm,” *IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS)*, 2020.

23. B. Kieu-Do-Nguyen, T. T. Hoang, C. K. Pham, and C. Pham-Quoc, "A Power-efficient Implementation of SHA-256 Hash Function for Embedded Applications," in International Conference on Advanced Technologies for Communications, IEEE Computer Society, 2021, pp. 39–44. doi: 10.1109/ATC52653.2021.9598264.
24. P. R. Lawhale, S. N. Kale, H. Kasturiwale, and Y. N. Thakare, "FPGA Implementation of Compact Architecture for Lightweight Hash Algorithm for Resource Constrained Devices," 2025. [Online]. Available: <https://internationalpubs.com>
25. B. Welte and J. Zambreno, "An FPGA Implementation of SipHash," IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), May 2023.
26. C. A. Lara-Nino, M. Morales-Sandoval, and A. Diaz-Perez, "Small lightweight hash functions in FPGA," in 9th IEEE Latin American Symposium on Circuits and Systems, LASCAS 2018 - Proceedings, Institute of Electrical and Electronics Engineers Inc., Jun. 2018, pp. 1–4. doi: 10.1109/LASCAS.2018.8399948.
27. N. Sheybani, T. Gong, A. Ahmed, N. B. Njungle, M. Kinsy, and F. Koushanfar, "Gotta Hash 'Em All! Speeding Up Hash Functions for Zero-Knowledge Proof Applications," Jan. 2025, [Online]. Available: <http://arxiv.org/abs/2501.18780>
28. J.-F. Têtu, L.-C. Trudeau, M. Van Beirendonck, A. Balatsoukas-Stimming, and P. Giard, "A Standalone FPGA-based Miner for Lyra2REv2 Cryptocurrencies," May 2020, doi: 10.1109/TCSI.2020.2970923.
29. Li, Y., Liu, F., & Wang, G. (2023). Automating Collision Attacks on RIPEMD-160. IACR Transactions on Symmetric Cryptology, 2023(4), 112–142.
30. Shen, Y., & Wang, G. (2018). Improved preimage attacks on RIPEMD-160 and HAS-160. KSII Transactions on Internet and Information Systems, 12(2), 727–746.
31. Abbas, A., Voss, R., Wienbrandt, L., & Schimmler, M. (2014). An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs. In Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS (Vol. 2015-April, pp. 454–461). IEEE Computer Society.