electronics

MDPI

Article

Efficient Architectures for Full Hardware Scrypt-Based Block Hashing System

Duc Khai Lam [1,2,*], Vu Trung Duong Le [3] and Thi Hong Tran [4]

[1] Computer Engineering Department, University of Information Technology, Ho Chi Minh City 700000, Vietnam
[2] Vietnam National University, Ho Chi Minh City 700000, Vietnam
[3] Graduate School of Information Science, Nara Institute of Science and Technology, Nara 630-0192, Japan; le.vu_trung_duong.lp4@is.naist.jp
[4] Graduate School of Enginering, Osaka City University, Osaka 558-8585, Japan; hong@osaka-cu.ac.jp
* Correspondence: khaild@uit.edu.vn

Abstract: The password-based key derivation function Scrypt has been employed for many services and applications due to its protection ability. It has also been employed as a proof-of-work algorithm in blockchain implementations. Although this cryptographic hash function provides very high security, the processing speed and power consumption to generate a hashed block for the blockchain network are low-performance. In this paper, a high-speed and low-power hardware architecture of the Scrypt function is proposed to generate blocks for the Scrypt-based blockchain network. This architecture minimizes the number of main computational blocks to reduce the power consumption of the system. In addition, the proposed sharing resources and pipelined architectures make the calculation speed increase significantly while the hardware cost is reduced by half compared to the parallel non-pipelined architecture. The full hardware system is designed and implemented on Xilinx Virtex-7 and Aveo U280 FPGA platforms. The hash rate of the proposed system reaches 229.1 kHash/s. Its hash rate, hardware and energy efficiencies are much higher than those of the other works implemented on FPGA and GPU hardware platforms. The proposed hardware architecture is also successfully implemented in an ASIC design using ROHM 180 nm CMOS technology.

Keywords: Scrypt hash algorithm; SHA-256 algorithm; PBKDF2; Salsa20; blockchain

Citation: Lam, D.K.; Le, V.T.D.; Tran, T.H. Efficient Architectures for Full Hardware Scrypt-Based Block Hashing System. *Electronics* **2022**, *11*, 1068. https://doi.org/10.3390/electronics11071068

Academic Editors: Sang-Woo Jun, Yeseong Kim and Jisung Park

Received: 31 January 2022
Accepted: 15 March 2022
Published: 28 March 2022

1. Introduction

Cryptographic hash functions are security algorithms to parse input data of arbitrary size to output hash data of fixed length. Currently, these functions are widely applied to blockchain systems to provide barriers to attackers [1,2]. Depending on the security demand, different blockchain technologies employ different cryptographic hash functions to protect their networks [3]. There are a variety of hash functions applied for the blockchain networks, such as Secure Hash Algorithm-1 (SHA1), SHA3 [4], SHA256 and ETHASH [5]. In the Internet of Things (IoT) applications, blockchain technology has recently been considered to be used to protect the security of data exchanged between the low-end devices. An SHA-256 accelerator with power-efficient architecture for blockchain-based IoT applications is proposed in [6]. For evaluating different implementation styles and hardware architectural schemes of SHA2 hash processing accelerator, Raffaele et al. propose an easy-to-use workbench [7]. This workbench is a user-configurable parameter framework to explore the performance, resources and energy consumption of SHA2 designs on different target hardware technologies. More advanced cryptographic hash functions, called password-based key derivation functions (PBKDF), such as PBKDF2, Bcrypt and Scrypt, provide more security than others [8–10].

Among password-based key derivation functions, Scrypt is the maximal memory-hard function [11,12]. It has been created by Colin Percival [13]. It has been employed for many

services and applications due to its protection ability [14,15]. It has also been employed as a proof-of-work algorithm in the blockchain implementations [16]. Although this hash function provides very high security, it costs an enormous amount of delay and power to generate a hashed block for the blockchain network which raises concerns from many researchers [17–19].

There are some related works of implementing the Scrypt function. For the software platforms, R. Han et al. [20] formulate and implement the Scrypt function on the Graphic Processing Units (GPUs) to analyze the speed and power. Ntantogian et al. [21] propose a framework to quantify the delay time for password hashing. In addition, the popular Content Management Systems (CMS) and web applications are used as inputs of the proposed framework to evaluate the security of hashing schemes of these applications. Moreover, a set of solutions is proposed to enhance the security of the hashing schemes of CMS and Web application frameworks. This framework is also run on GPUs. Two Scrypt and Bcrypt functions are implemented and their hash rates are compared when they are implemented on GPUs [22]. Scrypt function execution time is sped up 1.5× by performing the simple data-intensive computation at near memory, called near-data-processing (NDP) [12]. In addition, several Litecoin mining systems using the Scrypt algorithm are implemented on different kinds of GPUs [23]. These GPU-based Scrypt hashing systems have to run many cores in a parallel manner to increase the hash rate; therefore, they cost huge amounts of power consumption.

For the hardware platforms, Luca et al. [24] propose a design architecture with multiple Salsa cores and multiple ROMix cores to exploit parallel instances of the Scrypt algorithm. This design provides high throughput, but it costs much in hardware resources and power consumption. An ASIC prototype for multiple Scrypt cores design architecture is implemented to obtain the high hash rate [25]. Duong et al. [26] propose a hardware design using two separate ROMix cores and apply the traditional pipelined technique; thus, the throughput of this design is low. Therefore, these approaches are not sufficient to apply to a blockchain network.

In this paper, a new Scrypt hardware architecture for hashing the blocks of the blockchain networks is introduced to increase the hashing speed, hardware and energy efficiencies. Accordingly, the sharing resources architectures for two PBKDF2 cores are proposed to reduce the hardware resources. In these PBKDF2 cores, the optimized SHA256 core is used as the main processing unit [27]. In addition, the Mix block and the F_Round module are proposed for the Salsa 20/8 core to lower the total hardware resource of the ROMix core. Finally, the sharing PBKDF2 cores and pipelined hardware architecture for Scrypt core are proposed to improve the hash speed, hardware and energy efficiencies. This architecture uses a sharing pair of PBKDF2 cores and sixty-four ROMix cores working together in a pipelined mechanism. When comparing to a parallel non-pipelined Scrypt architecture, the hash performance of the proposed design is increased up to 64 times. Meanwhile, it takes only half of the cost of hardware resources. This proposed hardware architecture is implemented on the Xilinx Virtex-7 field-programmable gate array (FPGA) and the ASIC ROHM 180nm CMOS technology. The hash rate, hardware efficiency and energy efficiency of the proposed implementation are evaluated and compared to those of related works.

The outline of this paper is shown as follows. The Scrypt algorithm is explained in Section 2. Then, the proposed hardware design of the Scrypt hashing system is mentioned in Section 3. Section 4 shows the results of our proposed hardware implementation. Finally, the conclusion of the paper is presented in Section 5.

2. Background

This section gives the backgrounds of the Scrypt block hashing in blockchain networks and the Scrypt hashing algorithm.

2.1. Scrypt-Based Block Hashing for the Blockchain Network

Figure 1 shows the Scrypt-based block hashing system in a computer node of the blockchain network. First, the pending transactions stored in the mempool are selected to form a candidate block. Then, this block is hashed and validated by the Scrypt function. Finally, the validated block is added to the blockchain network. Algorithm 1 shows the detail of the Scrypt block hashing system. Because our system is considered to apply to the blockchain network, the salt value specified by any blockchain hashing processes [28,29] is generated from the block header of the blockchain network and the padding.

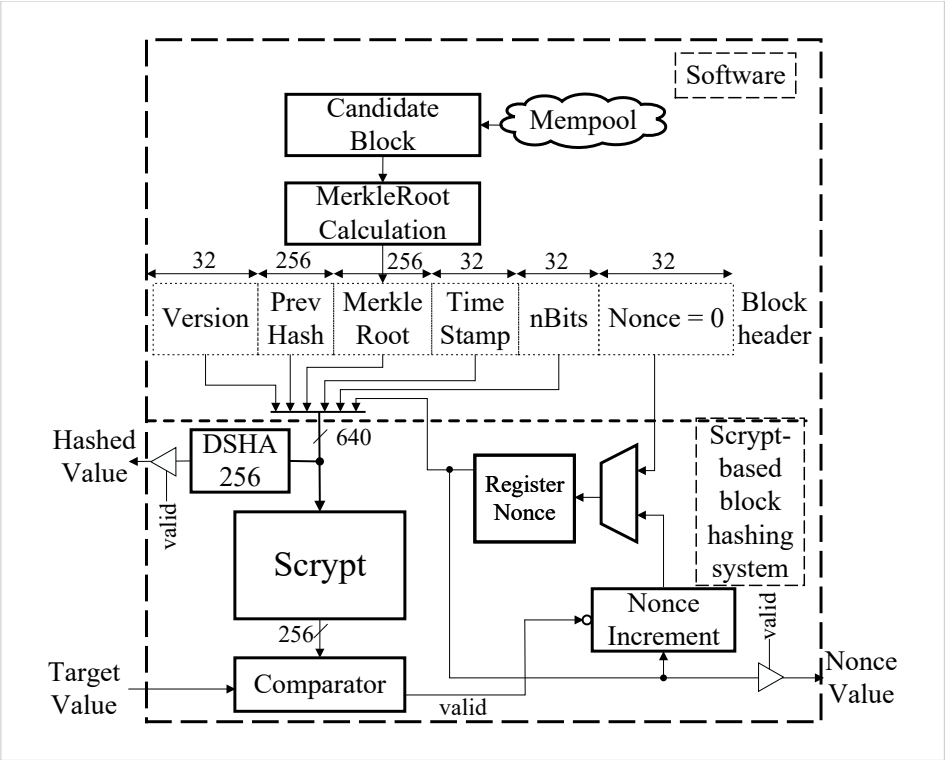

Figure 1. Scrypt-based block hashing for the blockchain networks.

Algorithm 1 (block_hash, nonce) = Scrypt-based_block_hashing(block_header)

```
nonce = 0
while nonce < 2^32 do
    block_hash = SHA256(SHA256(block_header))
    scrypt_hash = Scrypt(block_header)
    if scrypt_hash < target then return (block_hash, nonce)
    else
        nonce = nonce + 1
    end if
end while
```

In the blockchain networks, each block consists of two main fields: header and transactions. The transaction field contains the transactions which need to be executed. The header field, which is used for the block hashing process, consists of six sub-fields as described in Table 1.

The primary module of this block hashing system is the Scrypt. It takes the 640-bit block header to compute a 256-bit Scrypt hash. Then, the Comparator module checks if the hash output is lower than the target value; in which case, the block header with the current nonce is considered to be valid. Otherwise, the block header is changed by increasing its nonce value to re-calculate a new Scrypt hash until a valid block header is

found. Finally, the block, which includes the valid block header and transactions, is hashed by the double-SHA256 (DSHA256) module before being added to the blockchain network. Due to its high-security process, the Scrypt module costs an enormous amount of delay and power to generate a hashed block.

Table 1. Fields of block header. Reprinted with permission from [27], 2020, Lam Duc Khai.

Field	Length in Bit	Description
Version	32	Block version number
Previous Block Hash	256	256-bit hash of the previous block header
Merkle Root	256	256-bit hash based on all of the transactions in the block
Time Stamp	32	Current block timestamp as seconds since 1970-01-01T00:00 UTC
nBits	32	Current target in compact format
Nonce	32	32-bit number (starts at 0)

2.2. Scrypt Algorithm

Introduced by Colin Percival [13], the Scrypt algorithm is a password-based key derivation function. The Scrypt pseudo code is presented in Algorithm 2. The parameters of the Scrypt algorithm, (r, N, p), are specified by the users depending on their using purposes. These parameters affect how much memory and computational power are used. For the block hashing in the Litecoin blockchain application, these parameters used in our proposed Scrypt hashing system are specified as (1, 1024, 1) for (r, N, p), respectively [28]. This algorithm performs three steps using two functions, PBKDF2 and ROMix, as shown in Algorithm 2.

Algorithm 2 Out = Scrypt (blockheader)

Scrypt variables and parameters for hashed block mining:
message = blockheader (640 bits)
salt = blockheader, padding (1024 bits)
Block size factor (r) = 1
Number of iterations (c) = 1
Parallelization parameter (p) = 1
CPU/Memory cost parameter (N) = 1024
Length of DK in bits (dklen) = 256
Algorithm's steps:

1: p_out = PBKDF2(message, salt, c, 1024 × r × p)
2: r_out = ROMix(p_out, N, r)
3: scrypt_out = PBKDF2(message, r_out, c, dklen)
4: **return** scrypt_out

2.2.1. PBKDF2

Password-Based Key Derivation Function 2 (PBKDF2) is one of the key derivation functions used to reduce vulnerabilities to brute force attacks with intensive computations. In the Scrypt algorithm, PBDKF2 uses the Hash-based Message Authentication Code (HMAC) to input the message along with salt value to produce a derived key [30]. Algorithm 3 depicts more details regarding the PBKDF2 algorithm.

In cryptography, the HMAC is a message authentication code (MAC) using a cryptographic hash function and a secret cryptographic key. It is used for verifying the data integrity and authenticity of a message. The HMAC pseudo code is shown in Algorithm 4.

The HMAC for PBDKF2 uses the SHA256 as its cryptographic hash function. The SHA256 is a member of the Secure Hash Algorithm 2 (SHA2) family, created by the United States National Security Agency [31]. The SHA256 shown in Algorithm 5 includes three steps: Padding, Block Decomposition and Hash Computation. The input message of SHA256 is split into N of 512-bit blocks by the Padding, and then these N blocks are processed sequentially by the Block Decomposition and Hash Computation. The high-speed and low-power SHA256 hardware implementation approaches are presented in [27,32].

Algorithm 3 Out = PBKDF2(message, salt, c, dklen)

Out = ""
for i ← 1 to (dklen/256) **do**
    DK_i = HMAC({salt, i}, message)
    Out = {DK, DK_i}
end for
return Out

Algorithm 4 Out = HMAC(salti, message)

IPAD = 36363636...36_16 (256 bits)
OPAD = 5C5C5C5C...$5C_16$ (256 bits)
KHASH = SHA256(message)
IXOR = {(KHASH $\oplus$ IPAD), IPAD}
OXOR = {(KHASH $\oplus$ OPAD), OPAD}
IHASH = SHA256({IXOR, salti})
OHASH = SHA256({OXOR, IHASH})
Out = OHASH
return Out

Algorithm 5 digest = SHA256(message_in)

$M^{[0:N-1]}$, N = Padding_Splitting (message_in)
$H^{(0)}$ = H_Constants
for t ← 0 to (N-1) **do**
    W = BlockDecomposition($M^{(t)}$)
    $H^{(t+1)}$ = HashComputation($H^{(t)}$, K_Constants, W)
end for
return digest = {$H_1^{(N)}$, $H_2^{(N)}$, $H_3^{(N)}$, $H_4^{(N)}$, $H_5^{(N)}$, $H_6^{(N)}$, $H_7^{(N)}$, $H_8^{(N)}$}

2.2.2. ROMix

A sequential memory-hard function used in Scrypt is the ROMix. In the ROMix, the message block is mixed by two main phases: writing memory phase and reading memory phase. The detail of the ROMix algorithm is depicted in Algorithm 6. In our proposed Scrypt block hashing system, the memory required for the ROMix core is 128 KB.

The BlockMix algorithm, shown in Algorithm 7, uses the Salsa 20/8 function to mix data. Salsa20 is an original cipher function developed by Daniel J. Bernstein [33]. Salsa 20/8 is a hash function whose input is a set of sixteen 32- bit strings in little-endian format [34]. The description of Salsa 20/8 is mentioned in Algorithm 8. Salsa 20/8 has a total of eight rounds: four rounds of *ColumnRound* and four rounds of *RowRound*, which are calculated by Equations (1) and (6), respectively.

$$(y_0, y_1, \ldots, y_{15}) = ColumnRound(x_0, x_1, \ldots, x_{15}) \quad (1)$$

where the *ColumnRound* function is calculated by four *QuarterRound* blocks in Equations (2)–(5).

$$(y_0, y_4, y_8, y_{12}) = QuarterRound(x_0, x_4, x_8, x_{12}) \tag{2}$$

$$(y_5, y_9, y_{13}, y_1) = QuarterRound(x_5, x_9, x_{13}, x_1) \tag{3}$$

$$(y_{10}, y_{14}, y_2, y_6) = QuarterRound(x_{10}, x_{14}, x_2, x_6) \tag{4}$$

$$(y_{15}, y_3, y_7, y_{11}) = QuarterRound(x_{15}, x_3, x_7, x_{11}) \tag{5}$$

$$(x_0, x_1, \ldots, x_{15}) = RowRound(y_0, y_1, \ldots, y_{15}) \tag{6}$$

where the *RowRound* function is calculated by four *QuarterRound* blocks in Equations (7)–(10).

$$(x_0, x_1, x_2, x_3) = QuarterRound(y_0, y_1, y_2, y_3) \tag{7}$$

$$(x_5, x_6, x_7, x_4) = QuarterRound(y_5, y_6, y_7, y_4) \tag{8}$$

$$(x_{10}, x_{11}, x_8, x_9) = QuarterRound(y_{10}, y_{11}, y_8, y_9) \tag{9}$$

$$(x_{15}, x_{12}, x_{13}, x_{14}) = QuarterRound(y_{15}, y_{12}, y_{13}, y_{14}) \tag{10}$$

where the *QuarterRound* function is calculated by Rotation Left (*RotL*) operators in Equations (11)–(15).

$$(n_0, n_1, n_2, n_3) = QuarterRound(m_0, m_1, m_2, m_3) \tag{11}$$

where

$$n_0 = m_0 \oplus RotL[(m_3 + m_2, 18)] \tag{12}$$

$$n_1 = m_1 \oplus RotL[(m_0 + m_3, 7)] \tag{13}$$

$$n_2 = m_2 \oplus RotL[(m_1 + m_0, 9)] \tag{14}$$

$$n_3 = m_3 \oplus RotL[(m_2 + m_1, 13)] \tag{15}$$

Algorithm 6 block = ROMix (block, N, r)

```
for i ← 0 to (N-1) do
    Mem_i = block
    block = BlockMix(block, r)
end for
for j ← 0 to (N-1) do
    j = block[489:480] (block's 10-bit from 480 to 489)
    block = BlockMix(block ⊕ Mem_j, r)
end for
return block
```

Algorithm 7 block = BlockMix(block, r)

```
X = block[1023:512] (block's 512 high bits)
for i ← 0 to (2 × r − 1) do
    X = X ⊕ block[511:0] (block's 512 low bits)
    X = X + Salsa20/8(X)
    if (i == 0) then
        OutH = X
    else
        OutL = X
    end if
end for
block = {OutH, OutL}
return block
```

Algorithm 8 Out = Salsa20/8(message)

```
{x0, x1, ... , x15} = message
for i ← 0 to 3 do
    {y0, ... , y15} = ColumnRound({x0, ... , x15})
    {x0, ... , x15} = RowRound({y0, ... , y15})
end for
Out = {x0, x1, ... , x15}
return Out
```

3. Proposed Scrypt Hardware Architecture

In this section, the Scrypt hardware architectures for the block hashing system are proposed to increase the hashing speed, hardware resources and energy efficiencies.

3.1. Low-Resource and High-Throughput Non-Pipelined Scrypt Architecture

As mentioned in Algorithm 2, the Scrypt function includes two PBKDF2 cores and one ROMix core. The first arguments, message, of two PBKDF2 cores are similar. Then, we can see from the Algorithms 3 and 4 that outputs IXOR and OXOR at step 4 and step 5 of Algorithm 4 are not changed for both PBKDF2 cores. Thus, these outputs should be generated once to reduce the latency and hardware resources. Our proposed non-pipelined Scrypt architecture is shown in Figure 2. In this architecture, two PBKDF2 cores are implemented in a different manner. PBKDF2_1 core and PBKDF2_2 are implemented for step 1 and step 3 in Algorithm 2, respectively. The values IXOR and OXOR are generated once from PBKDF2_1 core, and then they are stored to use for PBKDF2_2 core. Therefore, the PBKDF2_2 core is not only faster in processing time but smaller in hardware resource use than those of the PBKDF2_1 core.

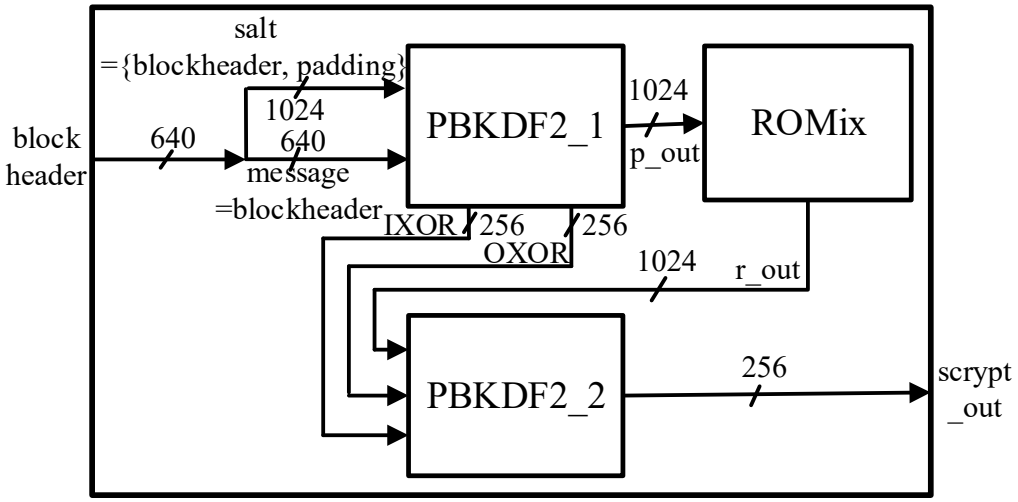

Figure 2. Proposed non-pipelined Scrypt architecture.

3.2. Low-Resource and High-Throughput PBKDF2 Core Architectures

In this sub-section, new hardware architectures for two PBKDF2 cores are proposed to reduce the number of processing modules and unnecessary calculations by sharing the data and hardware resource between two PBKDF2 cores. In the PBKDF2 function, the SHA256 core is used as the key processing unit. Duong et al. [26] use three SHA256 cores for the PBKDF2_1 core and two SHA256 cores for the PBKDF2_2 core. Because these SHA256 modules must be performed sequentially, the hardware resource is used inefficiently if one module performs whereas the others are idle. In addition, this design has a lack of sharing resource and data between two PBKDF2 cores. Therefore, our proposed architecture uses only one SHA256 core for the PBKDF2_1 core and one SHA256 core for the PBKDF2_2 core. The hardware resources are reduced considerably. Although a SHA256 core can be shared to both PBKDF2_1 and PBKDF2_2 cores, two SHA256 cores are still necessary to perform the pipelined technique, as discussed in Section 3.4. In our design, the architecture of the high-speed and low-power SHA256 is used [27,32].

Figure 3 shows the high throughput architecture for the PBKDF2_1 core described in Algorithm 3. Since the input argument dklen of PBKDF2_1 core is 1024, the HMAC core is operated iteratively 4 times when the factor i changes from 1 to 4. From Algorithms 3 and 4, the values IXOR and OXOR do not change while the HMAC core is being operated iteratively, and then the IOXOR instance, which performs steps 1 to 5 of Algorithm 4, only needs to operate once at the first iteration. Then, these outputs of IOXOR are stored in registers to use for three next iterations; thus, the total throughput of PBKDF2_1 core is increased.

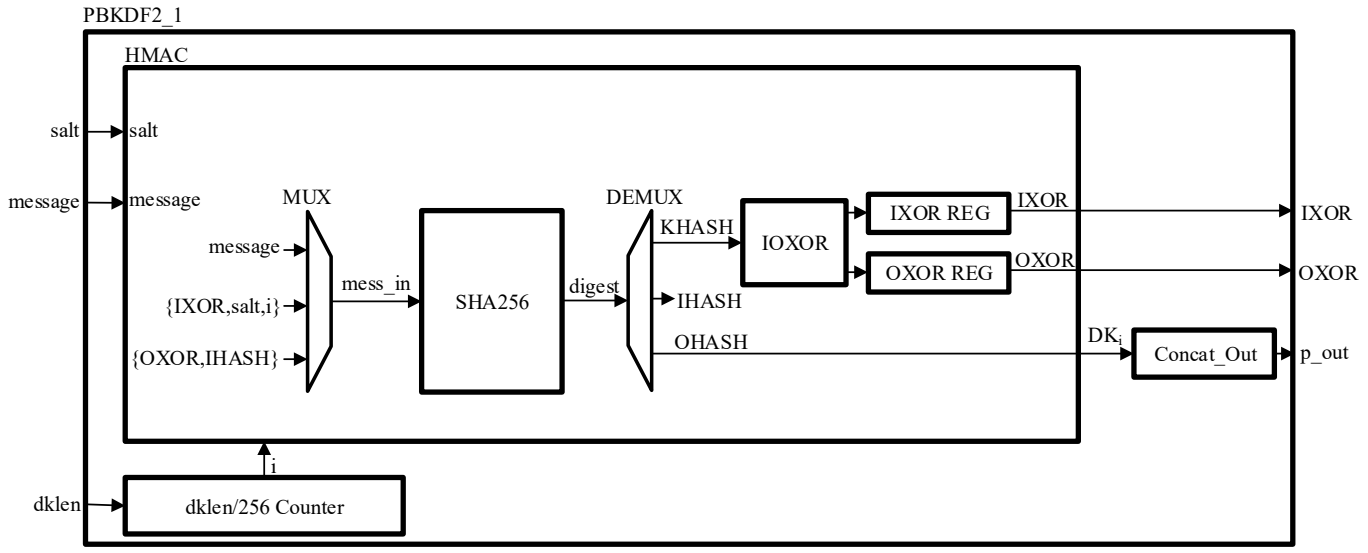

Figure 3. Proposed PBKDF2_1 core.

In addition, as mentioned in Section 3.1, the values IXOR and OXOR are also reused for PBKDF2_2 core; thus, the PBKDF2_2 core and its HMAC core have less complexity and higher throughput as shown in Figure 4. Since the input argument dklen of PBKDF2_2 core is 256, the HMAC core in the PBKDF2_2 core is operated only once.

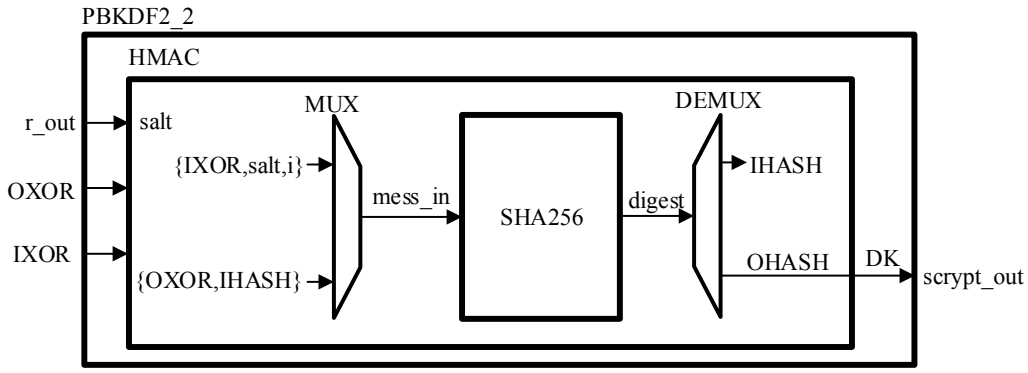

Figure 4. Proposed PBKDF2_2 core.

3.3. Proposed Architecture for ROMix Core

As mentioned in Algorithm 6, the ROMix is divided into two phases: writing memory phase and reading memory phase. Each phase consists of 1024 loops. Its hardware architecture is presented in Figure 5. There are two main parts in the ROMix core: one 128 KB BRAM and one BlockMix core. In the writing phase, 1024 loops are performed to write the outputs from the BlockMix core into BRAM with the write addresses generated from a UP-COUNTER. Then, in the reading phase, 1024 loops are performed to read data from the BRAM with the read address generated from the BlockMix core. In BlockMix core, the Salsa20/8 instance is the key processing unit.

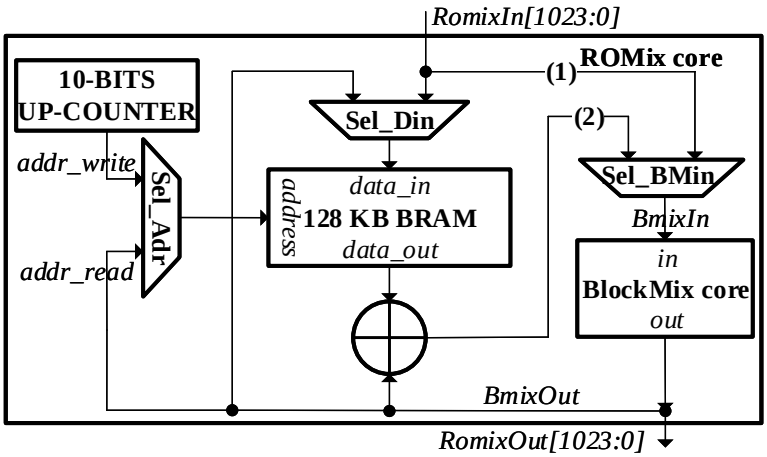

Figure 5. ROMix core.

Figure 6 shows the hardware architecture of the BlockMix core. BlockMix performs two loops of Salsa20/8 operation, and each Salsa20/8 performs four loops of *ColumnRound* and *RowRound* alternatively. If a Salsa20/8 core uses both *ColumnRound* and *RowRound* modules, the total area is large. It can be realized that the outputs of *ColumnRound* and *RowRound* modules depend on each other; therefore, these modules cannot be performed in parallel. In addition, the functions of *ColumnRound* and *RowRound* are similar. Therefore, in our design, only one module, named F_Round, is needed for both *ColumnRound* and *RowRound* to reduce half of the hardware resources without any latency loss compared to the conventional design. F_Round module includes four QR modules that perform the *QuarterRound* function as mentioned in Section 2.

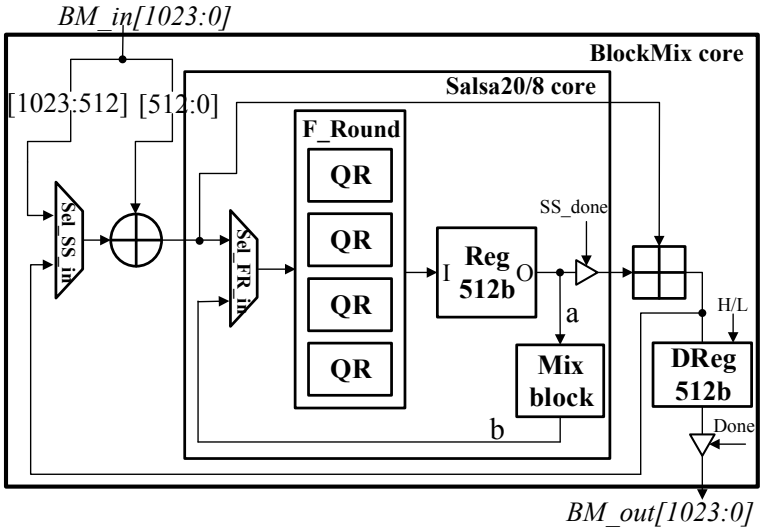

Figure 6. BlockMix core.

The function of the *Mix* block is to mix 512-bit signal a to 512-bit signal b. This function is shown in Equation (16).

$$(b_0, b_1, \ldots, b_{15}) = Mix(a_0, a_1, \ldots, a_{15}) \tag{16}$$

where

$$\begin{array}{llll} b_0 = a_6 & ; b_1 = a_9 & ; b_2 = a_{12} & ; b_3 = a_3; \\ b_4 = a_{10} & ; b_5 = a_{13} & ; b_6 = a_0 & ; b_7 = a_7; \\ b_8 = a_4 & ; b_9 = a_1 & ; b_{10} = a_{14} & ; b_{11} = a_{11}; \end{array}$$

$$b_{12} = a_2 \quad ; b_{13} = a_5 \quad ; b_{14} = a_8 \quad ; b_{15} = a_{15}.$$

where a_i and b_i are 32-bit signals.

3.4. Shared Resources—Pipelined Scrypt Architecture

In this section, a shared resource and pipelined architecture for the Scrypt core is proposed to increase the hashing speed while reducing the hardware resources and power consumption. This pipelined architecture is proposed to make sure that all processing modules such as PBKDF2 cores and ROMix cores have no free time and are utilized with the highest efficiency.

The non-pipelined Scypt architecture includes one PBKDF2_1 core, one ROMix core and one PBKDF2_2 core. The latencies of these cores are shown in Table 2. Thus, the total processing latency of the non-pipelined Scrypt core is 57,012 cycles.

From the results, the latency of the ROMix core is 64 times longer than that of the PBKDF2_1 core. In addition, the ROMix core cannot be split into smaller pieces to divide its latency into 64 times for applying pipeline as the traditional way. Thus, the latency to perform one hash input of the Scrypt function equals the total latencies of the PBKDF2_1, ROMix and PBKDF2_2 cores. In other words, this Scrypt core needs 64 times this latency to perform 64 hash inputs of Scrypt function. This latency number can be reduced if several Scrypt cores perform parallel; however, the hardware cost is correspondingly increased.

The hardware resources (HR) of the instances in our implementation in Xilinx Virtex-7 and Aveo U280 FPGA platforms are also shown in Table 2. For both platforms, the total resources of two PBKDF2 cores approximately are equal to those of the ROMix instance. Therefore, a new pipelined Scrypt core architecture is implemented as Figure 7 to speed up the throughput while saving the hardware resources when compared to the 64 parallel non-pipelined Scrypt cores. In this architecture, one PBKDF2_1 core, 64 ROMix cores and one PBKDF2_2 core are used. Using this core, after the PBKDF2_1 core finishes processing the first hash input to make the output for the first ROMix core, it continues processing the second hash input to output for the corresponding ROMix core without any waiting. When all 64 inputs are already given to PBKDF2_1 core, the first ROMix core also finishes its work for the first hash input to produce the output to PBKDF2_2 core. Applying this technique, the hashing speed of our proposed pipelined Scrypt core is increased approximately 64 times compared to that of the single non-pipelined Scrypt core. In addition, the hardware resources are saved when compared to the 64 parallel non-pipelined cores because only one PBKDF2_1 core and one PBKDF2_2 core are used in the proposed pipelined core, instead of 64 PBKDF2_1 cores and 64 PBKDF2_2 cores when 64 non-pipelined cores are used for processing in parallel.

Table 2. Instances hardware resources.

Instances	Latency (No. Clocks)	HR Vertex 7 (No. Slices)	HR Aveo U280 (No. Slices)
PBKDF2_1 (P1)	873	509	519
PBKDF2_2 (P2)	267	451	452
ROMix	55,872	965	949

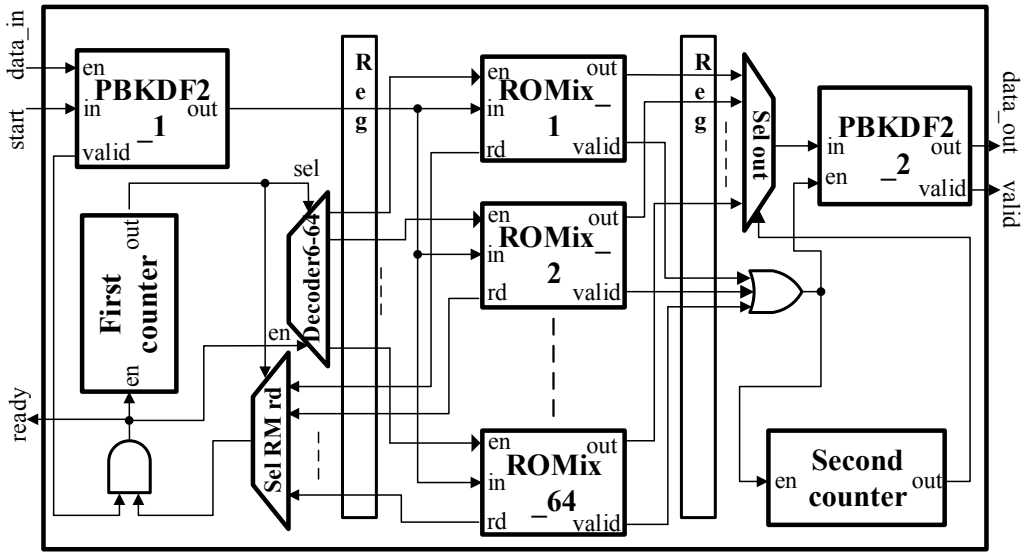

Figure 7. Proposed pipelined Scrypt architecture.

Figure 8 shows the timing schedule of the pipelined Scrypt core. Each hash input is processed by three modules: PBKDF2_1 (P1), ROMix and PBKDF2_2 (P2). In our design, the P1 takes 873 cycles, the ROMix takes 55,872 cycles which is 64 times of P1, and the P2 takes 267 cycles. The proposed Scrypt core works pipeline for a sequence of hash inputs as follows: the 1st hash input is processed by P1 core, then the output is continued to process by ROMix_1 core, and finally the output hash of the 1st hash input is conducted by P2 core. Since there are 64 ROMix cores in the Scrypt core and the latency of each ROMix core is 64 times that of P1 core, the 2nd to the 64th hash inputs can be processed pipeline as indicated in Figure 8. After the 64th hash input is processed by P1, the pipeline processing is continued for the 65th hash input. The 1st hash input is conducted by ROMix_1 core, and at the same time the 65th hash input is also conducted by P1 core; thus, the 65th hash input is continued to be processed by ROMix_1 core without any waiting time. This pipelined procedure is the same for the next hash inputs. The *n* th input is processed by the core ROMix_*m*, which *m* is (*n mod 64*).

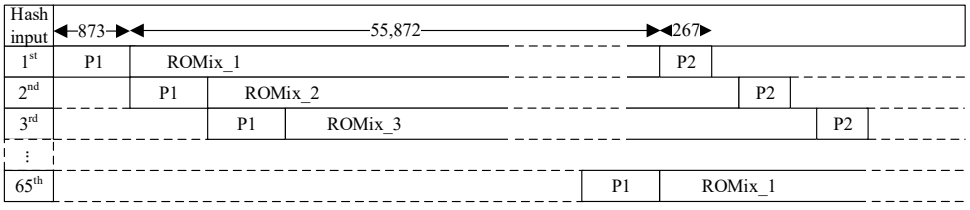

Figure 8. Pipelined Scrypt timing chart.

In the blockchain network, the Scrypt-based block hashing system has to compute a huge number of hash inputs until the satisfying output is found, and then if those inputs are performed by the proposed pipelined Scrypt core, the processing latency reaches 873 cycles per hash input. Therefore, its hash performance is increased approximately 64 times compared to the single non-pipelined Scypt core.

4. Results

4.1. Function Verification Results

The Scrypt-based block hashing system is designed by the Verilog Hardware Description Language (HDL), and then it is implemented and verified on FPGA Xilinx Virtex 7.

The verification flow is presented in Figure 9. To verify the implemented system, the real data of the hashed blocks are collected from the Litecoin blockchain network [35]. These data include the block header, target, nonce and block hash values. The block header and the target value are input to our Scrypt-based block hashing system to be computed. The outputs of our system are the nonce and block hash values. Then, these outputs are compared to the expected values collected from the blockchain network. The system is verified by 10,000 different input values.

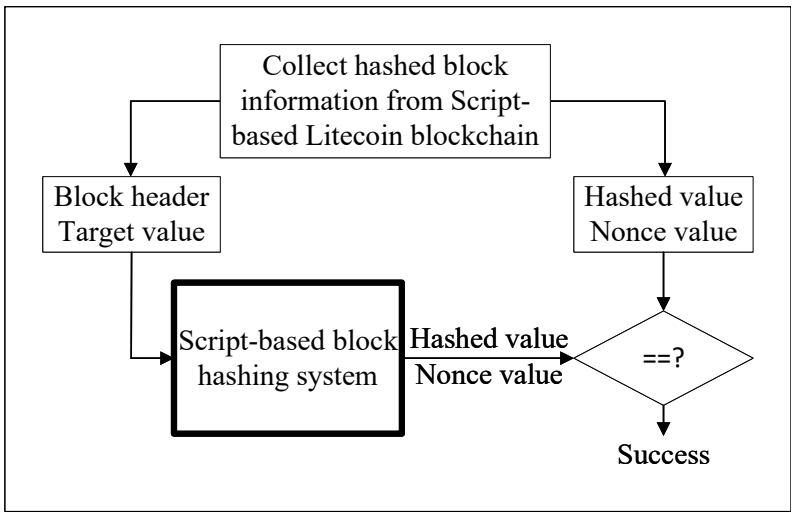

Figure 9. Verification flow.

4.2. Implementation Performance Results

To prove our proposed designs obtain higher performance than [25,26] on the same platform, the proposed designs are implemented in the Xilinx Virtex 7 FPGA platform. However, if this FPGA does not have enough Block RAM (BRAM) resources to implement our proposed pipelined design with 64 ROMix instances, then the proposed pipelined design with 32 ROMix instances is implemented on this FPGA. To achieve the best performance as explained in Section 3.4, our proposed pipelined design with 64 ROMix instances is implemented in the Xilinx Aveo U280 FPGA platform, which has enough BRAM resources for our design. The BRAM resources used on the targeted FPGA devices are shown in Table 3. In this section, all results, including the hardware resource cost (HW), processing frequency (Freq.) and power consumption (Power), are collected after our design pass the place and route process on the Xilinx Vivado Design Suite.

Table 3. BRAM utilization comparison (Parameter (N, r, p) = (1024, 1, 1)).

Design	Devices	No. Cores	No. BRAM	BRAM Utilization (%)
Proposed Non-pipelined	Virtex-7 VX485T	1	28.5	28.5/1030 (2.8%)
Proposed Non-pipelined	Virtex-7 VX485T	32	912	912/1030 (88.5%)
Proposed Pipelined 32 ROMix	Virtex-7 VX485T	1	912	912/1030 (88.5%)
Proposed Pipelined 64 ROMix	Alveo U280	1	1824	1824/2016 (90.5%)

The implementation results of the pipelined Scrypt core with 32 ROMix instances and 64 ROMix instances, single non-pipelined Scrypt core and 32 parallel non-pipelined Scrypt cores are shown in Table 4. In addition, these results are also compared to the previous designs. All designs utilize the on-chip memory for the BRAM of ROMix cores. The configuration parameters (N, r, p) of these designs are (1024, 1, 1). The implementation results are shown in Tables 4 and 5. As explained in Section 3.4, the processing latency of the non-pipelined core is 57,012 cycles, and the processing latency of the pipelined Scrypt core is 1746 cycles for 32 ROMix instances or 873 cycles for 64 ROMix instances, respectively. The frequencies of the single non-pipelined Scrypt core, 32 parallel non-pipelined Scrypt cores and the proposed pipelined Scrypt core with 32 ROMix instances are the same (156.05 Mhz) because their critical paths implemented on the same Virtex 7 FPGA platform are the same. The *Hashrate*, hardware efficiency ($HW.Eff$) and energy efficiency ($Ener.Eff$) are calculated by Equations (17)–(19), respectively.

$$Hashrate = \frac{Frequency}{\text{Number of latency cycles}}(Hash/s) \tag{17}$$

$$HW.Eff = \frac{Hashrate}{\text{Number of HW.slices}}(Hash/s/slice) \tag{18}$$

$$Ener.Eff = \frac{Powerconsumption}{\text{Hash rate}}(Joule/Hash) \tag{19}$$

The proposed single non-pipelined Scrypt core includes one PBKDF2_1 instance, one ROMix instance and one PBKDF2_2 instance. The hardware architectures of these instances are proposed in Sections 3.2 and 3.3 to reduce hardware resources.

From the results in Table 4, hardware efficiency of our design (1.17 Hs/Slice) is much higher than that of [25,26]. Our hardware efficiency is slightly slower than that of [24], but our power consumption (0.55 W) is four times lower than that of [24].

Table 4. Hardware implementation comparison results (Parameter (N, r, p) = (1024, 1, 1)).

Design	Devices	No. Cores	HW. (Slice)	Freq. (MHz)	Hash Rate (kHash/s)	HW. Eff. (Hash/s/Slice)	Power (W)
[24]	Spartan-6 LX45	1	5859	50.00	6.90	1.18	2.25
[25]	Virtex-6 LX240T	1	4670	200.00	1.33	0.28	NA
[26]	Virtex-7 VX485T	1	52,298	355.05	5.33	0.10	NA
Proposed Non-pipelined	Virtex-7 VX485T	1	2347	156.05	2.74	1.17	0.55
Proposed Non-pipelined	Virtex-7 VX485T	32	75,104	156.05	87.68	1.17	17.6
Proposed Pipelined 32 ROMix	Virtex-7 VX485T	1	35,041	156.05	89.38	2.34	7.03
Proposed Pipelined 64 ROMix	Alveo U280	1	68,431	259.94	229.1	3.35	14.45

Table 5. Other platform implementation comparison results.

Design	Devices	No. Cores	Parameter (N, r, p)	Freq. (MHz)	Hash Rate (kHash/s)	Power (W)	Ener. Eff. (J/kHash)
[20]	NVDIA-CUDA 9.1	1	(1024, 1, 1)	2600.00	41.33	NA	NA
[21]	NVIDIA-GTX 1070	1	(8192, 1, 1)	1683.00	0.12	NA	NA
[22]	NVIDIA-GTX 480	1	(4096, 1, 1)	701.00	27.75	NA	NA
[12]	ARMv7-Cortex A15	1	(16,384, 8, 1)	2500.00	0.01	NA	NA
[23]	NVIDIA-GX TITAN	1	(1024, 1, 1)	797.00	6.06	2.04	0.336
[23]	AMD-AX 7990	1	(1024, 1, 1)	950.00	17.04	6.13	0.360
Proposed Pipelined 64 ROMix	Alveo U280	1	(1024, 1, 1)	259.94	229.1	14.45	0.063
Proposed Pipelined 64 ROMix	ROHM 180 nm ASIC	1	(1024, 1, 1)	136.30	156.13	0.82	0.005

The implementation results of the 32 parallel non-pipelined Scrypt cores are also shown in Table 4. From the results of the single core, the results of 32 parallel cores are scaled up 32× in terms of all parameters. The hash rate is increased by using multiple parallel cores (175.36 kHash/s), and the hardware resource and power consumption are also increased by the same factor. Moreover, the hash rate of our proposed pipelined Scrypt core with 32 ROMix instances (89.38 kHash/s) is not only higher than that of the 32 parallel cores, but the hardware resource of the proposed pipeline implementation (35,041 Slices) is only half of that of the implementation of 32 parallel cores (75,104 Slices). The reason is that half the resources go into ROMix core (965 slices on Virtex 7 or 949 slices on Aveo U280) and half into PBKDF core (960 slices on Virtex 7 or 971 slices on Aveo U280) as shown in Table 2. In the 32 parallel non-pipelined Scrypt cores, both ROMix and PBKDF cores have to multiplied by 32, while in the proposed pipelined Scrypt core with 32 ROMix instances, only the ROMix core has to be multiplied by 32. Our hash rate is 13 times higher than that of [24]. The great increase in the hardware efficiency by our proposed pipeline Scrypt core architecture (2.34 Hs/Slice) compared to that of previous works is highlighted.

Our proposed pipelined Scrypt core with 64 ROMix instances achieves highest performance when it is implemented on the Aveo U280 FPGA platform. The design not only speeds up the hash rate to 229.1 kHash/s but also achieves a hardware efficiency (3.35) much higher that of previous designs.

The hash rate and power consumption comparisons between our proposed implementation and other GPU platforms are also shown in Table 5. For a fair comparison between FPGA and GPA platforms, in our FPGA platform, the top SoC module including the Scrypt proposed design IP, embedded MicroBlaze soft CPU and transmission module (UART-Lite) is implemented. The hash rate of our proposed FPGA implementation is much higher than that of the GPU platforms. Moreover, the energy efficiency, defined as Joule per hash (J/H), of our Aveo U280 FPGA implementation is approximately five times higher than that of the NVIDIA and AMD GPU implementations [23].

Besides FPGA implementation, the proposed pipelined Scrypt core is also successfully implemented in ASIC using the Synopsis electronic design automation (EDA) tool suite. The design methodology is defined by Synopsis design flow [36]. This proposed core is implemented in ROHM 180 nm CMOS technology. This ASIC is an energy-saving technology producing low frequencies with super low power consumption. From Table 5, the ASIC implementation provides a maximum frequency of 136.30 MHz; thus, its hash rate reaches 156.13 kHash/s. These results are worse than those of FPGA implementation because there are two reasons: (1) the FPGA Xilinx Virtex 7 is using 28- nm high-speed

technology; (2) implementing on different technology platforms generates different critical paths. However, the ASIC implementation result reaches the power consumption of 0.82 W that helps the energy efficiency (0.005 J/kHash) be 12 times higher than that of Aveo U280 FPGA implementation.

5. Conclusions

In this paper, the new pipelined hardware architecture for hashing the Scrypt-based blocks of the blockchain network is presented. This system is implemented on both Xilinx Virtex-7 FPGA and ROHM 180 nm ASIC. The proposed individual PBKDF2 and ROMix cores help to reduce the hardware resources and improve the latency. In addition, the approached pipelined Scrypt core architecture helps to improve not only the hash rate but also the hardware and energy efficiencies. Compared to the related works implemented on the FPGA platform, the hash rate of our pipelined design implementation is more than 40 times greater and the hardware efficiency is more than 30 times greater. The hash speed and energy efficiency of our implementation are also much higher than those of the related works implemented on the GPU platform.

Author Contributions: Investigation, V.T.D.L.; Methodology, D.K.L. and T.H.T.; Project administration, D.K.L.; Supervision, D.K.L. and T.H.T.; Writing—original draft, V.T.D.L.; Writing—review and editing, D.K.L. and T.H.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by Vietnam National University, HoChiMinh City (VNU-HCM) under grant number DSC2021-26-01.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Leible, S.; Schlager, S.; Schubotz, M.; Gipp, B. A Review on Blockchain Technology and Blockchain Projects Fostering Open Science. *Front. Blockchain* **2019**, *2*, 16. [CrossRef]
2. Kshetri, N. Blockchain's roles in strengthening cybersecurity and protecting privacy. *Telecommun. Policy* **2017**, *41*, 1027–1038. [CrossRef]
3. Zhai, S.; Yang, Y.; Li, J.; Qiu, C.; Zhao, J. Research on the Application of Cryptography on the Blockchain. *J. Phys. Conf. Ser.* **2019**, *1168*, 32–77. [CrossRef]
4. Fu, J.; Qiao, S.; Huang, Y.; Si, X.; Li, B.; Yuan, C. A Study on the Optimization of Blockchain Hashing Algorithm Based on PRCA. *Secur. Commun. Netw.* **2020**, *2020*, 8876317. [CrossRef]
5. Wang, L.; Shen, X.; Li, J.; Shao, J.; Yang, Y. Cryptographic primitives in blockchains. *J. Netw. Comput. Appl.* **2019**, *127*, 43–58. [CrossRef]
6. Martino, R.; Cilardo, A. Designing a SHA-256 processor for blockchain-based IoT applications. *Internet Things* **2020**, *11*, 100254. [CrossRef]
7. Martino, R.; Cilardo, A. A Flexible Framework for Exploring, Evaluating, and Comparing SHA-2 Designs. *IEEE Access* **2019**, *7*, 72443–72456. [CrossRef]
8. Wiemer, F.; Zimmermann, R. High-speed implementation of bcrypt password search using special-purpose hardware. In Proceedings of the 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), Cancun, Mexico, 8–10 December 2014; pp. 1–6. [CrossRef]
9. Ertaul, L.; Kaur, M.; Gudise, V.A.K.R. Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms. In Proceedings of the 2016 International Conference on Wireless Network (ICWN'2016), Las Vegas, NV, USA, 25–28 July 2016; pp. 66–72.
10. Hatzivasilis, G. Password-Hashing Status. *Cryptography* **2017**, *1*, 10. [CrossRef]
11. Alwen, J.; Chen, B.; Pietrzak, K.; Reyzin, L.; Tessaro, S. Scrypt Is Maximally Memory-Hard. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, 30 April–4 May 2017; Springer: Cham, Switzerland, 2017; pp. 33–62. [CrossRef]
12. Jiwon, C.; Tali, M.; Iris, B.R.; Maurice, H. Attacking Memory-Hard Scrypt with near-Data-Processing. In Proceedings of the International Symposium on Memory Systems, MEMSYS '19, Washington, DC, USA, 30 September–3 October 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 33–37. [CrossRef]
13. Percival, C.; Josefsson, S. *The Scrypt Password-Based Key Derivation Function*; Technical Report RFC 7914. Available online: https://doi.org/10.17487/RFC7914 (accessed on 8 March 2022). [CrossRef]

14. Hatzivasilis, G.; Papaefstathiou, I.; Manifavas, C. Password Hashing Competition—Survey and Benchmark. *IACR Cryptol. ePrint Arch.* **2015**, *2015*, 265.
15. Hatzivasilis, G. Password Management: How Secure Is Your Login Process? In *Model-Driven Simulation and Training Environments for Cybersecurity*; Hatzivasilis, G., Ioannidis, S., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 157–177.
16. Padmavathi, M.; Suresh, R.M. Secure P2P Intelligent Network Transaction using Litecoin. *Mob. Netw. Appl.* **2019**, *24*, 318–326. [CrossRef]
17. Ling, R.; Srinivas, D. Bandwidth Hard Functions for ASIC Resistance. In *Theory of Cryptography*; Yael, K., Leonid, R., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 466–492.
18. Almeida, L.C.; Andrade, E.R.; Barreto, P.S.L.M.; Simplicio, M.A., Jr. Lyra: Password-based key derivation with tunable memory and processing costs. *J. Cryptogr. Eng.* **2014**, *4*, 75–89. [CrossRef]
19. Alwen, J.; Chen, B.; Kamath, C.; Kolmogorov, V.; Pietrzak, K.; Tessaro, S. On the Complexity of Scrypt and Proofs of Space in the Parallel Random Oracle Model. In *Advances in Cryptology—EUROCRYPT 2016*; Fischlin, M., Coron, J.S., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 358–387.
20. Han, R.; Foutris, N.; Kotselidis, C. Demystifying Crypto-Mining: Analysis and Optimizations of Memory-Hard PoW Algorithms. In Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 24–26 March 2019; pp. 22–33. [CrossRef]
21. Ntantogian, C.; Malliaros, S.; Xenakis, C. Evaluation of password hashing schemes in open source web platforms. *Comput. Secur.* **2019**, *84*, 206–224. [CrossRef]
22. Dürmuth, M.; Kranz, T. On Password Guessing with GPUs and FPGAs. In *Technology and Practice of Passwords*; Mjølsnes, S.F., Ed.; Springer International Publishing: Cham, Switzerland, 2015; pp. 19–38.
23. Litecoin Foundation. Litecoin Wiki, "Mining Hardware Comparision". 2018. Available online: https://litecoin.info/index.php/Mining_hardware_comparison (accessed on 8 March 2022).
24. Luca, P. *Master Thesis: Hardware Implementation for Litecoin Mining*; Technical Report; University in Leuven: Leuven, Belgium, 2015.
25. Alpha Technology (INT) LTD. *Scrypt ASIC Prototyping Preliminary Design Document*; Technical Report; Alpha Technology (INT) LTD: Manchester, UK; London, UK, 2013.
26. Duong, L.V.T.; Hieu, D.V.; Luan, P.H.; Hong, T.T.; Khai, L.D. Hardware Implementation For Fast Block Generator Of Litecoin Blockchain System. In Proceedings of the 2021 IEEE International Symposium on Electrical and Electronics Engineering (ISSE), Ho Chi Minh, Vietnam, 15–16 April 2021; pp. 1–6. [CrossRef]
27. Duong, L.V.T.; Thuy, N.T.T.; Khai, L.D. A fast approach for bitcoin blockchain cryptocurrency mining system. *Integration* **2020**, *74*, 107–114. [CrossRef]
28. Litecoin. Block Hashing Algorithm. 2018. Available online: https://litecoin.info/index.php/Block_hashing_algorithm (accessed on 8 March 2022).
29. Bitcoin. Block Hashing Algorithm. 2018. Available online: https://en.bitcoin.it/wiki/Block_hashing_algorithm (accessed on 8 March 2022).
30. Kaliski, B. *PKCS #5: Password-Based Cryptography Specification Version 2.0*; Technical Report RFC 2898; The Internet Research Task Force Organization: USA, 2000. Available online: https://www.rfc-editor.org/info/rfc2898 (accessed on 8 March 2022). [CrossRef]
31. Dang, Q.H. *Secure Hash Standard*; National Institute of Standards and Technology. Available online: https://doi.org/10.6028/nist.fips.180-4 (accessed on 8 March 2022). [CrossRef]
32. Pham, H.L.; Tran, T.H.; Phan, T.D.; Le, V.T.D.; Lam, D.K.; Nakashima, Y. Double SHA-256 Hardware Architecture With Compact Message Expander for Bitcoin Mining. *IEEE Access* **2020**, *8*, 139634–139646. [CrossRef]
33. Bernstein, D.J. The Salsa20 Family of Stream Ciphers. In *New Stream Cipher Designs*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 84–97. [CrossRef]
34. Cohen, D. On Holy Wars and a Plea for Peace. *Computer* **1981**, *14*, 48–54. [CrossRef]
35. Litecoin Explorer. Available online: https://live.blockcypher.com/ltc/ (accessed on 8 March 2022).
36. Bhatnagar, H. *Advanced ASIC Chip Synthesis: Using Synopsys Design Compiler Physical Compiler and Prime Time*; Kluwer Academic Publishers: New York, NY, USA, 2002. [CrossRef]