Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A highly efficient reconfigurable rotation unit based on an inverse butterfly network*

Chao MA[†1], Zi-bin DAI[†‡1], Wei LI[2], Hai-juan ZANG[3]

([1]*Department of Electrical Engineering, Zhengzhou Institute of Information Science and Technology, Zhengzhou 450004, China*)

([2]*State Key Lab of ASIC and System, Fudan University, Shanghai 200240, China*)

([3]*School of Computer Engineering, Jiangsu University of Technology, Changzhou 213001, China*)

[†]E-mail: wenlu_ma@163.com; Daizb2004@126.com

Received May 17, 2016; Revision accepted Sept. 14, 2016; Crosschecked Nov. 20, 2017

Abstract: We propose a reconfigurable control-bit generation algorithm for rotation and sub-word rotation operations. The algorithm uses a self-routing characteristic to configure an inverse butterfly network. In addition to being highly parallelized and inexpensive, the algorithm integrates the rotation-shift, bi-directional rotation-shift, and sub-word rotation-shift operations. To our best knowledge, this is the first scheme to accommodate a variety of rotation operations into the same architecture. We have developed the highly efficient reconfigurable rotation unit (HERRU) and synthesized it into the Semiconductor Manufacturing International Corporation (SMIC)'s 65-nm process. The results show that the overall efficiency (relative area×relative latency) of our HERRU is higher by at least 23% than that of other designs with similar functions. When executing the bi-directional rotation operations alone, HERRU occupies a significantly smaller area with a lower latency than previously proposed designs.

Key words: Rotation operations; Self-routing; Control-bit generation algorithm; Inverse butterfly network
https://doi.org/10.1631/FITEE.1601265 **CLC number:** TP331.2

1 Introduction

The operations of rotation and bit-level permutation are of prime importance for many emerging applications, such as cryptography, image processing, and bioinformatics. For instance, in cryptography, good diffusion properties (Lee *et al.*, 2004) are widely applied in cryptography algorithms, such as the permutation-only encryption scheme in analog pay TV (Jolfaei *et al.*, 2015) and the permutation-substitution network for image encryption schemes (Belazi *et al.*, 2016). However, bit permutation is typically not well supported by current word-oriented microprocessors according to Hilewitz and Lee (2006). Therefore, it is important to implement bit permutation efficiently because it can increase the throughput. Rotation and permutation operations are both bit-rearrangement operations; however, they are classified into different types and implemented in most application-specific instruction set processors (ASIPs) (Sayilar and Chiou, 2015, Shi *et al.*, 2008). Rotation operation is a simple form of permutation operations, where all bits of an operand move by the same magnitude. If these two operations are implemented separately, an overhead resulting from additional hardware resources is incurred.

Dynamic multistage interconnection networks can realize connections among nodes by changing the switches states to enable the self-routing abilities of the networks. This flexibility can help complete data rearrangement operations (Nassimi and Sahni, 1981; Dai and Shen, 2007). Thus, Hilewitz and Lee (2009) proposed a shift-permute unit based on a kind of dynamic multistage interconnection network called an inverse butterfly. This unit can perform both basic rotation operations and sophisticated bit manipulations, including rotation (ROT) (Hilewitz and Lee, 2007), parallel extraction (PEX) and parallel deposit

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61404175)

(PDEP) (Hilewitz and Lee, 2008), group (GRP) (Hilewitz *et al.*, 2004), block extraction (EXTR) and insertion (DEP) (Intel Corporation, 2006), butterfly (BFLY) and inverse butterfly (IBFLY) (Hilewitz and Lee, 2006). More importantly, Hilewitz and Lee (2009) first proposed a special routing algorithm for rotation operations. The resulting delay in their proposed overall unit was only slightly longer than that in the arithmetic-logic unit (ALU). In short, they provided a new approach for designing a reconfigurable rotation unit.

However, the algorithm for the rotations noted above is serial and costly. Each step of the algorithm is complex and must be executed sequentially. This limits the processing performance and leads to considerable resource consumption. For instance, Chang *et al.* (2013) proposed a routing algorithm for rotation operations with a high degree of parallelism based on an inverse butterfly network. However, this algorithm involves significant consumption of hardware resources.

In this paper, we propose a routing algorithm for rotation operations that can increase computational speed and reduce resource consumption. We also enhance the algorithm to directly support multiple sets of sub-word parallel rotation operations, and design a powerful unit called the highly efficient reconfigurable rotation unit (HERRU). The contributions of this paper are as follows:

1. We propose a highly efficient, low-cost algorithm based on the inverse butterfly network, which can generate control bits for the rotation, bi-directional rotation, and parallel sub-word rotation operations.

2. We have tested the implementation of our powerful HERRU and compared its performance with that of the shift-permute unit proposed by Hilewitz and Lee (2009). The results show that our unit is faster (by 92.6% compared to the latency of their unit) and smaller (covering only 71% of the area occupied by their unit). Moreover, our unit can support eight more operations.

2 Structure and applications of butterfly and inverse butterfly networks

Fig. 1 shows the structure of the inverse butterfly network, where n=8. The network consists of lg n stages. Each stage is composed of n/2 two-input switches, and each switch can execute a two-bit input data 'pass-through' or 'swap' function under the control of 'Sel'. Two bits of input data form a small group in each stage by 2^{i-1} positions (i varies from 1 to lg n and indicates stages). All grouped data are then transmitted through the two-input switches at each stage, which can realize specific operations with appropriate control bits for all switches. The network can yield $2^{(n/2)\times \lg n}$ permutation results. From Fig. 1a, we can see that the network has the characteristics of recursion and split. If the final stage of the network is discarded, the rest of the stages can be seen as forming two independent n/2-bit-wide sub-inverse butterfly networks. The butterfly network has the inverse structure of the inverse butterfly network, and its interval of grouped input data is $2^{\lg n-i}$ bits apart at each stage of the network.

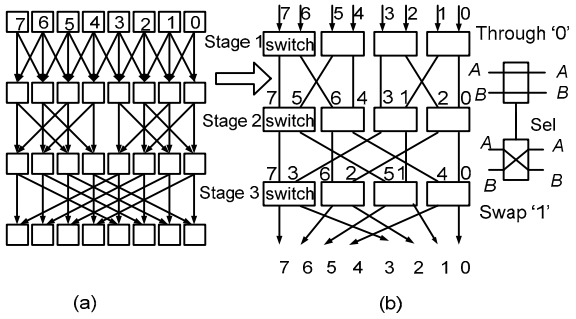

Fig. 1 Eight-bit inverse butterfly network (a) and eight-bit inverse butterfly network hardware circuit (b)

Hilewitz and Lee (2009) studied a special routing algorithm for a specific permutation based on inverse butterfly and butterfly networks, and proposed a series of complex bit permutation instructions, such as GRP, PEX/PDEP, and ROT. Thus, they significantly expanded the flexibility of networks and made an outstanding contribution to the field of bit permutation.

The GRP instruction gathers to the right data bits in register R2 selected by 1's in the mask register R3, and to the left those selected by 0's in the mask register R3 (Fig. 2). These can be regarded as two parallel operations: grp_left and grp_right. The GRP instruction, which has a resistance to linear and differential attacks, has been widely applied to design a new generation of cryptographic algorithms (Saraswathi and Venkatesulu, 2014; Bansod *et al*., 2014).

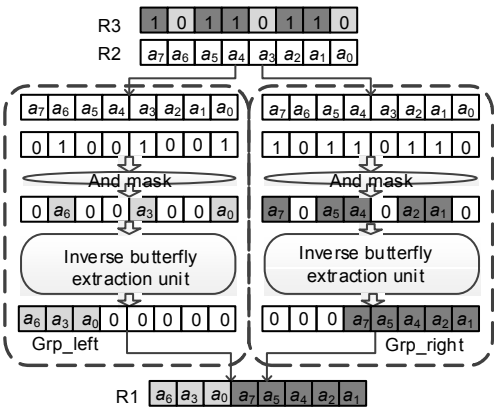

Fig. 2 The implementation of Group (GRP) instruction

The PEX and PDEP instructions can separately perform bit gather and bit scatter operations. The PEX instruction extracts and compacts bits from register R2 from positions selected by 1's in register R3 to register R1. The rest of the bits in register R1 are cleared to 0's (Fig. 3a). Thus, the PEX can be viewed as the right half of the GRP operation, which is implemented by only one inverse butterfly network. The PDEP instruction, implemented by the butterfly network, is the reverse operation of the PEX instruction (Fig. 3b). Notably, the two instructions are added to Intel's Haswell processor instruction set in 2013 (Intel Corporation, 2015).

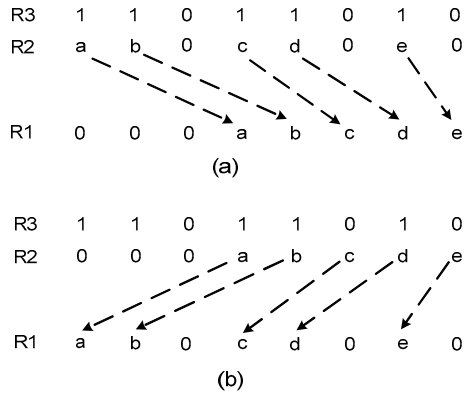

Fig. 3 PEX and PDEP operations: (a) an 8-bit PEX operation; (b) an 8-bit PDEP operation
PEX: parallel extraction; PDEP: parallel deposit

The ROT instruction is implemented mainly by a traditional barrel shifter (Pillmeier, 2002) or a log shifter (Pillmeier *et al.*, 2002). There has been no significant breakthrough in the hardware architectures of rotational operations in this decade. An n-bit log-rotation shifter has lg n stages, and the inputs are rotated by decreasing powers of 2 from the first stage to the last stage (Fig. 4). This is a simple structure because the magnitude of binary rotation $S(s_{\lg n-1}, s_{\lg n-2}, \ldots, s_0)$ can directly control all multiplexers; however, it is inflexible and cannot cope with advanced bit operations. Hilewitz and Lee (2009) first proposed a special routing algorithm for rotation operations based on the inverse butterfly/butterfly networks. They unified basic rotation operations with advanced bit-level manipulations in the same architecture. However, the algorithm for rotation operations is recursive, and all control bits are generated stage by stage. Therefore, it has a low degree of parallelism and causes a large critical path delay. When the width of the network is expanded, the algorithm's complexity increases drastically and hardware implementation becomes costly. Chang *et al.* (2013) also proposed a routing algorithm for rotation operations based on the same network with parallel features that effectively reduces critical path delay. According to rotation magnitude S and the initial control bits for all switches in the inverse butterfly network, the final control bits at all stages can be generated in parallel. However, the core step of the algorithm itself is a kind of rotation operation, and hence the hardware resource consumption is still higher than that of Hilewitz and Lee (2009)'s algorithm.

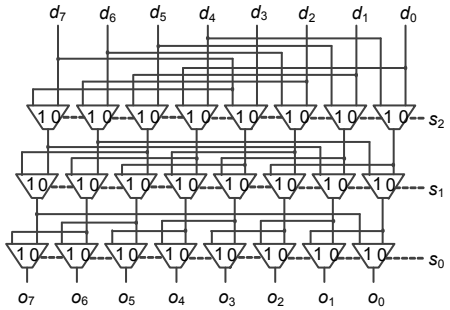

Fig. 4 Eight-bit log-rotation shifter

Therefore, this paper focuses on a highly efficient and low-cost algorithm to concurrently generate control bits for rotation operations based on the inverse butterfly network. We also enhance the algorithm to support bi-directional rotation operations and

sub-word rotation operations. Compared with the past work in this area, our algorithm has less complexity, a shorter critical path, and higher re-configurability.

3 Highly efficient reconfigurable rotation unit

3.1 Control bits for rotations in an inverse butterfly network

3.1.1 Generating crude control bits for rotations

From the structural properties of the inverse butterfly network, we can infer the following fact:

Fact 1 The input data do not move or move only by 2^{i-1} bits in each stage i, where i starts from 1.

This fact is similar to the indirect binary n-cube function (Rajkumar and Goyal, 2014), whereby the $(i-1)$th bit of the input data binary address can be changed only at the ith stage of the network. Therefore, when each input data item passes through the entire network (a total of lg n stages), it can reach any position.

From Fact 1, we can see that the least significant bit of the binary address of each input data can be changed only at the first stage. As shown in Fig. 5, to shift the seventh bit of input data from position 7 to position 5, the seventh binary address should be changed from '111' to '101'. According to the characteristics of binary address change, we propose a generation algorithm for crude control bits for rotations based on the 'Exclusive OR (XOR)' operation shown in Fig. 6.

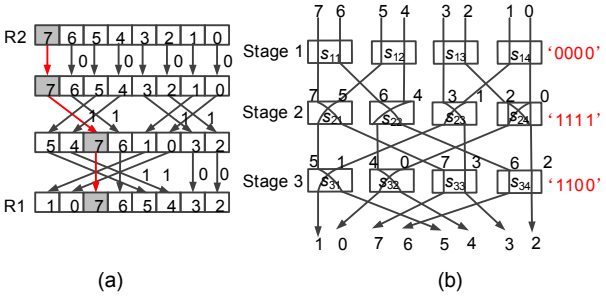

Fig. 5 Eight-bit left rotation by 6: (a) data stream structure; (b) hardware circuit structure

Step 1: Each binary address of input data from source register R2, denoted by the original bit position in Fig. 6, is subjected to an addition operation for rotation modulo-n, with the extent of the rotation denoted by shamt(S)—that is, the original bit position plus extent of rotation S modulo n—to yield the final rotated binary position, denoted by the destination bit position.

Step 2: Each destination bit position executes an 'XOR' operation on its original bit position to obtain the crude control bits of each input data item.

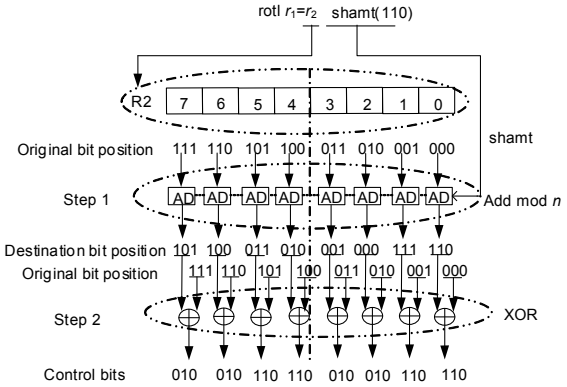

Fig. 6 Generation algorithm for crude control bits
Rotl: rotate left; shamt: rotate amount; XOR: exclusive OR

The crude control bits are rewritten in the longitudinal form as shown in Fig. 7. For example, the crude control bits of the least significant input data are '110'. Thus, the states of the corresponding switches $\{S_{14}, S_{24}, S_{32}\}$ (Fig. 5b) from the first stage to the final stage are 0, 1, 1.

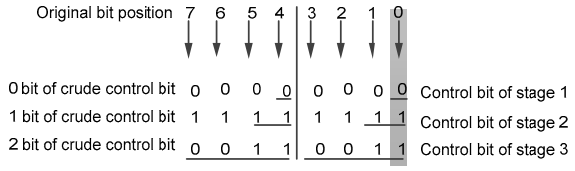

Fig. 7 Crude control-bit conversion table

A total of $n\times\lg n$ crude control bits are hence generated, whereas an n-bit inverse butterfly network has only $n/2\times\lg n$ switches, and requires $n/2\times\lg n$ control bits. Thus, at least 50% of the crude control bits are redundant. Furthermore, the efficient mapping of these crude control bits to the appropriate switches becomes a pressing problem. For example, in Fig. 5b although the crude control bit of the least significant input data item in stage 3 is '1', there is no way to directly map the '1' to the appropriate switch from (S_{31} ... S_{34}) in stage 3. We must first configure the switch state of S_{14} at stage 1, and successively

configure the switch state of S_{24} at stage 2 according to the state of S_{14}. Finally, the appropriate switch to be mapped in stage 3 is determined by the states of the previous two switches (S_{14} and S_{24}). That is, mapping the crude control bits to switches at the final stage must wait until all mappings to switches in previous stages are accomplished, which significantly increases the length of the critical path.

3.1.2 Refining crude control bits for rotations

The generated crude control bits are redundant and cannot configure switches in the inverse butterfly network directly. Therefore, we propose a parallel algorithm to refine the crude control bits for rotations. This algorithm not only terminates the relationship with regard to the control bits between any two stages to effectively shorten the critical path, but also significantly reduces the consumption of hardware resources.

An n-bit inverse butterfly network can be considered as two $n/2$-bit sub-networks with (lg n−1)-stage circuits followed by a stage that passes through or swaps grouped bits that are $n/2$ positions apart. To left-rotate the input data $\{a_{n-1}\ a_{n-2}\ \ldots\ a_0\}$ by S positions, the two sub-networks must have left-rotated their half input data $\{a_{n-1}\ a_{n-2}\ \ldots\ a_{n/2}\}$ and $\{a_{n/2-1}\ a_{(n-1)/2-1}\ \ldots\ a_0\}$ by $S'=S$ mod $(n/2)$. Recursively, each of the two $n/2$-bit sub-networks can be considered as two $(n/4)$-bit sub-networks with (lg n−1)−1 stages followed by a stage that passes through or swaps grouped bits that are $n/4$ positions apart. Each sub-network must have left-rotated their $n/4$ input data by $S'=S$ mod $(n/4)$. Therefore, to achieve left-rotation by S positions using the n-bit-wide inverse butterfly network, the input data must be left-rotated by $S'=S$ mod 2^i within each 2^i-bit wide sub-network at each stage i (Fig. 8).

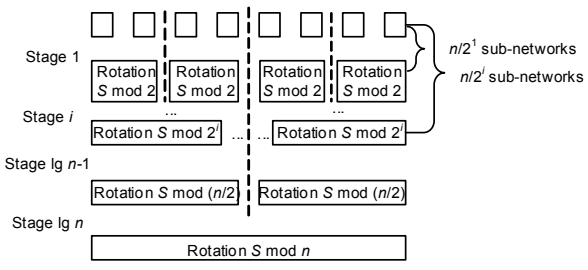

Fig. 8 Sub-inverse butterfly network

There are $n/2$ sub-networks in the first stage and $n/2^i$ sub-networks in stage i (i>1). The extent of rotation S' is the same in the sub-networks at the same stage. Hence, the control bits for a rotation must be identical in each sub-network at the same stage. That is to say, in stage 1, there are $n/2$ sub-networks, and each sub-network contains only one switch requiring one bit to be controlled. In stage i, there are $n/2^i$ sub-networks, and each sub-network contains 2^{i-1} switches requiring 2^{i-1} bits to be controlled. In the final stage, there is only one sub-network containing $n/2$ switches requiring $n/2$ bits. Thus, the inverse butterfly network requires $n-1$ bits to control all switches for a rotation operation (see the gray part in Fig. 9). Therefore, we downsize $n\times$lg n crude control bits to $n-1$ bits.

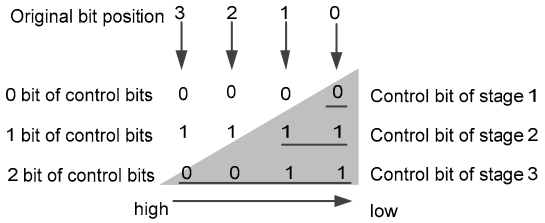

Fig. 9 Optimizing the number of crude control bits

Moreover, to achieve left-rotation by S positions, the two half-input data items must be rotated by $S'=S$ mod $n/2$ at stage lg $n-1$ (Fig. 10), and the input data at stage lg n is of the following form:

$$a_{n-s'-1}\ldots a_{n/2}a_{n-1}\ldots a_{n-s'}\ \|\ a_{n/2-1-s'}\ldots a_0 a_{(n/2)-1}\ldots a_{n/2-s'}. \quad (1)$$

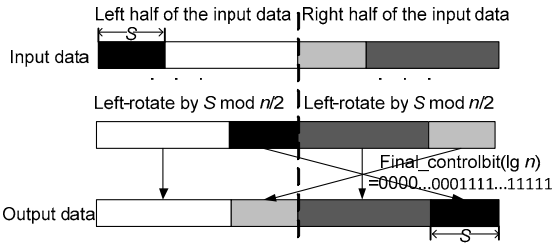

Fig. 10 Left-rotation by S positions

This is true for S less or greater than $n/2$. From the conclusion of Hilewitz and Lee (2009), we see that the control bits, denoted as Final_controlbit(lg n), for a left-rotation of S bits at the final stage, are

$$\text{Final\_controlbit}(\lg n) = \begin{cases} 0^{n/2-(S \bmod n/2)} \,||\, 1^{S \bmod n/2}, & S \bmod n < n/2, \\ 1^{n/2-S \bmod n/2} \,||\, 0^{S \bmod n/2}, & S \bmod n \geq n/2, \end{cases} \quad (2)$$

where a^k is a string of k a's, '1' means 'swap', and '0' means 'pass-through'. Furthermore, due to the recursive structure of the inverse butterfly network, we can generalize Eq. (3) by substituting i for lg n, 2^i for n, and 2^{i-1} for $n/2$ to Eq. (2) as

$$\text{Final\_controlbit}(i) = \begin{cases} 0^{2^{i-1}-(S \bmod 2^{i-1})} \,||\, 1^{S \bmod 2^{i-1}}, & S \bmod 2^i < 2^{i-1}, \\ 1^{S \bmod 2^{i-1}} \,||\, 0^{2^{i-1}-S \bmod 2^{i-1}}, & S \bmod 2^i \geq 2^{i-1}. \end{cases} \quad (3)$$

From Eq. (3), we can conclude that the form of the final control bits for a rotation operation at each stage is $\{0^x||1^y\}$ or $\{1^y||0^x\}$, where $x+y=2^{i-1}$.

Fig. 8 shows that the crude control bits of the rightmost 2^i-bit-wide sub-network at each stage are sufficient to control all switches in the inverse butterfly network. Each binary address of the right half of the input data $\{2^{i-1}-1, 2^{i-1}-2, \ldots, 1, 0\}$ with extent of rotation S can generate the crude control bits for stage i. At first, each binary address of the right half of the input data undergoes 'addition' with extent of rotation S, and then is subjected to a modulo-n operation to obtain the destination bit position:

$$\begin{cases} \text{sum}_{2^{i-1}-1} = ((2^{i-1}-1) + s \bmod n)_b, \\ \text{sum}_{2^{i-1}-2} = ((2^{i-1}-2) + s \bmod n)_b, \\ \quad \ldots \\ \text{sum}_0 = (0 + s \bmod n)_b. \end{cases} \quad (4)$$

Sum_j is the destination bit position of the jth input data item (j=0, 1, …, $2^{i-1}-1$). There are lg n bits in sum_j with the most significant bit denoted as lg $n-1$. '$(\,)_b$' means the binary form of the number. Then, each sum_j executes an 'XOR' operation with the binary address of the jth input data item to obtain an lg n-bit crude control bit string denoted by crude_sum_j [lg $n-1$:0]:

$$\begin{cases} \text{crude\_sum}_{2^{i-1}-1}[\lg n-1:0] = \text{sum}_{2^{i-1}-1} \oplus (2^{i-1}-1)_b, \\ \text{crude\_sum}_{2^{i-2}-2}[\lg n-1:0] = \text{sum}_{2^{i-1}-2} \oplus (2^{i-1}-2)_b, \\ \quad \ldots \\ \text{crude\_sum}_0[\lg n-1:0] = \text{sum}_0 \oplus 0_b. \end{cases} \quad (5)$$

The ($i-1$)th bit is extracted from the crude control bits of each input data item to form a special crude control bit string for stage i:

$$\text{crude\_controlbit}(i) = \{\text{crude\_sum}_{2^{i-1}-1}[i-1],\ \text{crude\_sum}_{2^{i-2}-1}[i-1],\ \ldots,\ \text{crude\_sum}_0[i-1]\}. \quad (6)$$

From Eq. (5), we know that the ($i-1$)th bit of each binary address of the right half of the input data is '0' because the binary address of the input data is less than 2^{i-1}. When it executes the 'XOR' operation with the corresponding ($i-1$)th bit of sum_j, the number of crude control bits in stage i is equal to the length of the string in the ($i-1$)th bit of each sum_j. Hence, Eq. (6) can be changed to

$$\text{crude\_controlbit}(i) = \{\text{sum}_{2^{i-1}-1}[i-1],\ \text{sum}_{2^{i-1}-2}[i-1], \ldots, \text{sum}_0[i-1]\}. \quad (7)$$

Eq. (7) shows that the crude control bits in each stage i are from the ($i-1$)th bit of all sum_j. More importantly, it is easy to see that all $\text{sum}_j[i-1]$ (j varies from 0 to $2^{i-1}-1$) form a special string $\{0^x||1^y\}$ or $\{1^y||0^x\}$, where $x+y=2^{i-1}$.

Using the special string combined with the conclusion drawn from Eq. (3), we can obtain the following fact:

Fact 2 The final mapping control bits for stage i are from the ($i-1$)th crude control bit string or the reverse form of the string:

$$\text{Final\_controlbit}(i) = \text{crude\_controlbit}(i) \text{ or } \sim(\text{crude\_controlbit}(i)), \quad (8)$$

where '$\sim$' indicates reverse of the string.

To generate the final control bits for each stage i, we present a '0' position-tracing method to determine whether the strings mentioned in Eq. (8) should be

selected. The method works as follows: The lowest position of input data item a_0 is a constant value 0. Thus, the crude control bits of the lowest input data item are the same as the extent of rotation $(S)_b$. We can know the move track of the lowest input data at a certain stage through the binary value of $S=\{s_{\lg n-1}, s_{\lg n-2}, \ldots, s_0\}$. If any of $\{s_{i-2}, s_{i-3}, \ldots, s_0\}$ is equal to 1, it means that the lowest input data item has been moved prior to stage i (i>1), such that the final mapping control bits are the reverse string of the crude control bits in stage i. Only when all values of $\{s_{i-2}, s_{i-3}, \ldots, s_0\}$ are equal to 0, are the final mapping control bits identical to the crude control bits in stage i, because in this condition, the position of the lowest input data item has not been moved prior to stage i. Thus, the string of crude control bits is identical to that of the final mapping control bits. Note that the last magnitude of binary rotation S_0 is the control bit for stage 1.

Hence, we present a novel parallel algorithm to generate the control bits for rotation operations based on an inverse butterfly network.

Algorithm 1 consists of two parts: generation of the crude control bits and refinement of these crude control bits in conjunction with the generation of the final mapping control bits for all switches at each stage. We take left-rotate by 6 ($s_2s_1s_0$=110) as an example. At stage 1, s_0=0, and the final control bit is 0. At stage 2, the crude control bits are 11. Because the previous stage s_0 is 0, the final control bits are the same as the crude control bits. At stage 3, the crude control bits are 0011. Because $\{s_1 \text{OR} s_0\}$=1, the final control bits are 1100, which is the reverse form of the crude control bits.

Algorithm 1 Generating control bits from S for the inverse butterfly network

Input: S $\{s_{\lg n-1}, s_{\lg n-2}, \ldots, s_0\}$ // extent of rotation
$a_{n-1}a_{n-2}\ldots a_0$ // n-bit input data to be rotated
Output: Final_controlbit(i) // i=1, 2, …, lg n
Part 1:
1: **for** j←0 to n/2−1 **do**
2: Suma_j←(P_{aj}+$\{s_{\lg n-1}, s_{\lg n-2}, \ldots, s_0\}$) modulo n
// P_{aj} stands for the binary position of
// each input data item
3: Crude_suma_j← Suma_j XOR P_{aj}
// we can safely jump over this step
4: **end for**
Part 2:
5: **if** i==1
6: Final_controlbit(1)←s_0
7: **else**
8: **for** i←2 to lg n **do** // i stands for each stage
9: **if** ($s_{i-2}|s_{i-3}\ldots|s_0$==1)
10: Final_controlbit(i)←Crude_suma_0[i−1], …,
Crude_sum$a_{2^{i-1}-2}$[i−1], Crude_sum$a_{2^{i-1}-1}$[i−1]
11: **else**
12: Final_controlbit(i)←Crude_sum$a_{2^{i-1}-1}$[i−1],
Crude_sum$a_{2^{i-1}-2}$[i−1], …, Crude_suma_0[i−1]
// [i−1] stands for the ith bit of Crude_suma_i
13: **end if**
14: **end for**
15: **end if**

3.2 Determining the control bits for multi-mode operations

1. Determining the appropriate rotations control bits

Rotation operations include left- and right-rotation operations. We describe only how to generate the left-rotations of control bits for simplicity, as we can obtain the right-rotation of control bits using the symmetric characteristic of the network. Consequently, we obtain the following fact:

Fact 3 The final control bits mapping to the inverse butterfly network for right-rotation by S are the reverse of the control bits mapping for left-rotation by the same S at each stage.

2. Determining the parallel sub-word rotation control bits

Parallel sub-word rotations are a kind of split mode of n-bit wide rotations. The n-bit input data are divided into k=n/m groups (n, m are power indices of 2, and n>m), and each group can be rotated in parallel with m-bit width. The inverse butterfly network is composed of several identical small, wide sub-networks at each stage. There is no correlation among the input data in each sub-network. Hence, this network has the properties of split and iteration, and is very suitable for sub-word rotation operations. Completing an m-bit data rotation operation requires only the preceding lg m stages, and the generation algorithm to map the final control bits is identical to Algorithm 1, whose maximum value for i is lg m. The remaining lg n−lg m stages need to be maintained as 'pass-through'. That is, all switch states need to be directly configured to 0, as shown in Fig. 11.

3.3 Architecture of the highly efficient reconfigurable rotation hardware unit

1. Hardware implementation of Algorithm 1

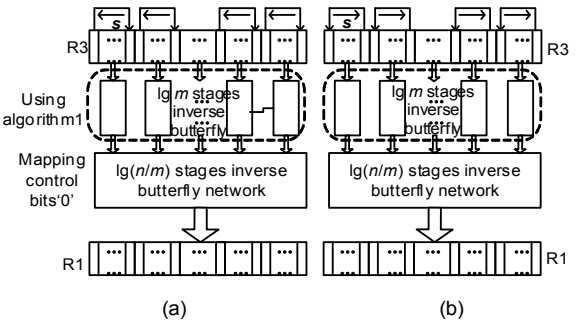

Fig. 11 Parallel sub-word rotation operations: (a) left parallel sub-word rotation; (b) right parallel sub-word rotation

The hardware implementation of the proposed algorithm is shown in Fig. 12, including the bi-directional rotation and sub-word rotation operations. Each source binary address of the inputs abbreviated as P_{aj} (j=0, 1, …, n/2−1) makes modulo-n additions to the binary rotation S to yield the destination position denoted by suma_j. The crude control bits b_j[lg n−1:0] are the result of suma_j 'XOR' P_{aj}, where [lg n−1:0] indicates the length of the crude control bits. All b_x[i−1] form a crude control bit string at stage i, where x=0, 1, …, 2^{i-1}−1. The crude control bit string and its reverse are selected by sel_i to obtain the final mapping control bits for stage i. The sel_i is a one-bit signal from the result of {s_{i-2} or s_{i-3} or … s_0}.

Fact 3 shows that the final mapping control bits of right rotations are the reverse of the final mapping control bits of left rotations. L_R is a one-bit signal of mode selection, and if L_R=1, the hardware unit generates the final mapping control bits for left rotation. Otherwise, the hardware unit generates the final mapping control bits for right rotation by choosing ~sel_i to control the multiplexers of each stage (see the top-left corner of Fig. 12). When it performs parallel sub-word rotation operations, the control bits for the previous lg m stages can be generated by sharing the same unit proposed above and the remaining lg n−lg m stages can be configured to 0, which can effectively reduce the circuit area and enhance re-configurability.

2. Hardware implementation of the new rotation units

In Table 1, we propose three functional units based on the inverse butterfly network. HERRU

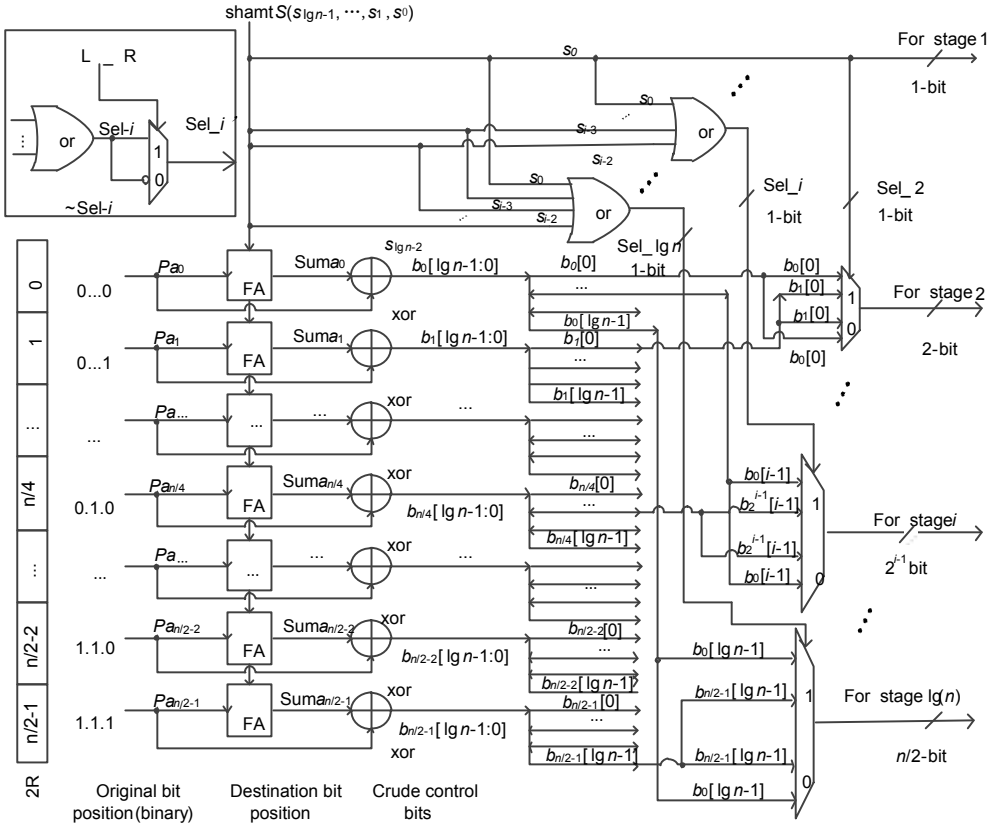

Fig. 12 Hardware circuit for rotation control bit generation

already contains the functions of the previous two units. Therefore, in this part, we introduce only the structure of HERRU. HERRU consists of two parts (Fig. 13): the generation unit for the control bits (control path) and the network unit for the inverse butterfly (data path).

The generation unit for the control bits is the most important module of the entire design, as it is used to generate the final mapping control bits in real time according to the extent of rotation *S* and the execution mode. The functions of the input ports are shown in Table 2.

Table 1 Three function units

Unit name	Left rotation	Right rotation	Sub-word rotation
Basic rotation shifter	√		
L_R rotation shifter	√	√	
HERRU shifter	√	√	√

HERRU: highly efficient reconfigurable rotation unit, which supports bi-directional and parallel sub-word rotation operations for (2^2, 2^3, 2^4, 2^5) bit width

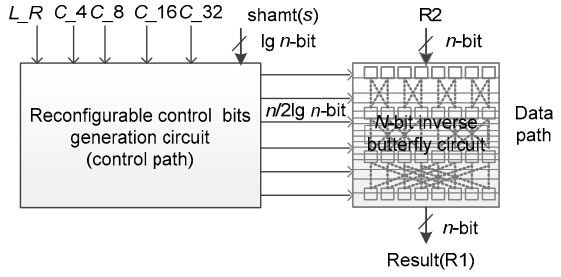

Fig. 13 Highly efficient reconfigurable rotation unit (HERRU) architecture

Table 2 Functions of mode ports

Mode func{*C_4*, *C_8*, *C_16*, *C_32*}	*L_R*	
	0 (right)	1 (left)
0000	64-bit	64-bit
1xxx	4-bit	4-bit
01xx	8-bit	8-bit
001x	16-bit	16-bit
0001	32-bit	32-bit

4 Results and performance analysis

For an objective comparison with related research, we set up the width of all designs to 64-bit and added the shifter operations mentioned in Hilewitz and Lee (2009). We started from two aspects of functionality and performance to evaluate the proposed multi-functional rotation units.

Table 3 lists the operations supported by different rotation shifters. The results show that the log-rotation shifter and our basic shifter can achieve only logic shift and rotation shift operations in a single direction. Chang *et al.* (2013)'s shifter, Hilewitz and Lee (2009)'s basic shifter, and our three shifters, based on the inverse butterfly network, are more flexible for complex-bit permutation operations. In addition, Chang *et al.* (2013) and Hilewitz and Lee (2009)'s basic shifters have the same functions as our *L_R* shifter. However, HERRU supports a large number of bi-directional sub-word rotation operations, such as 4-, 8-, 16-, and 32-bit rotation operations. Therefore, from the perspective of functionality, HERRU is more powerful.

We also synthesized all designs to the gate level, including the log-rotation shifter in the same environment (process 1.0, temperature −40 °C, voltage 1.08 V), optimizing for the shortest latency and the minimum area by using Design Compiler mapping to an SMIC 65-nm standard cell library (Semiconductor Manufacturing International Corporation, 2012). The results are summarized in Table 4.

Note that the relative area of Hilewitz and Lee (2009)'s basic shifter is 1.38 in Table 4, whereas Hilewitz and Lee (2009) calculated the value of the relative area as 0.69. The reason for this inconsistency is that Hilewitz and Lee (2009) compared the area of the basic shifter with the two areas occupied by the log-rotation shifter. Moreover, if the unit is compared with a single log-rotation shifter, the relative area should be upsized to 1.38. Table 4 shows the results for latency and the area of the three newly proposed units. The basic rotation shifter has 0.87 times the latency and 0.71 times Hilewitz's area (Hilewitz and Lee, 2009). Moreover, it has almost the same latency while a smaller area, in comparison with the log-rotation shifter. The *L_R* rotation shifter has the same functions as Hilewitz's (Hilewitz and Lee, 2009), but a greater speed and a smaller area.

Although HERRU is a complex unit that can support more operations, it has 0.97 times the latency and 0.79 times Hilewitz's area (Hilewitz and Lee, 2009). In comparison with Chang *et al.* (2013)'s

Table 3 Operations supported by rotation shifters

Operation	Log rotation shifter	Chang's basic shifter	Hilewitz's basic shifter	Our basic shifter	Our *L_R* shifter	Our HERRU shifter
64-bit PEX		√*	√*	√*	√*	√*
64-bit IBFLY		√*	√*	√*	√*	√*
64-bit logic shift	√~	√	√	√~	√	√
64-bit rotation *L_R*	√~	√	√	√~	√	√
32-bit rotation *L_R*						√
16-bit rotation *L_R*						√
8-bit rotation *L_R*						√
4-bit rotation *L_R*						√

~ represents that the relevant operation supports only single-direction rotation operation; * represents that the relevant operation can be supported by adding a new control bit generation circuit. Chang *et al.* (2013)'s shifter is based on the inverse butterfly network using the SMIC 65-nm standard cell library; Hilewitz and Lee (2009)'s shifter is based on the inverse butterfly network using the TSMC 90-nm standard cell library. PEX: parallel extraction; IBFLY: inverse butterfly

Table 4 Comprehensive performance comparison

Functional unit	Total area (μm^2)	Relative area	Buffer area (μm^2)	Latency (ns)	Relative latency	Efficiency*
Log-rotation shifter	2846.16	1.00	551.88	0.53	1.00	1.00
Hilewitz and Lee (2009)'s basic shifter	–	1.38	–	–	1.18	1.62
Chang *et al.* (2013)'s basic shifter	4293.32	1.51	758.32	0.60	1.13	1.71
Our basic shifter	2776.32	0.98	432.36	0.55	1.03	1.01
Our *L_R* shifter	3037.32	1.06	573.12	0.58	1.09	1.16
Our HERRU	3124.44	1.09	670.68	0.61	1.15	1.25

* Efficiency=relative area×relative latency

shifter, the HERRU area is significantly smaller (only 72% of Chang *et al.* (2013)'s design) and the operations supported by ours are greater in number.

To make a quantitative comparison with other designs, an efficiency parameter is proposed here. This parameter is the product of relative area and relative delay, and can be used to objectively evaluate the performance of all designs. From Table 4, we can see that our basic shifter, which has the same functions as the log-rotation shifter, has the highest efficiency (1.01). Although the relative latency of Chang *et al.* (2013)'s shifter is lower than Hilewitz and Lee (2009)'s, the efficiency is not higher. This is because the area occupied by Chang *et al.* (2013)'s shifter is too large. The *L_R* shifter, which has the same functions as Chang *et al.* (2013) and Hilewitz and Lee (2009)'s shifters, records higher efficiency (1.16). Moreover, the efficiency of HERRU, which supports more operations, is slightly lower than that of the *L_R* shifter, but better than the efficiency of the others.

When we remove the buffers in the basic shifter, *L_R* shifter, and log-rotation shifter from Table 4, the rest of the areas of our units are slightly greater than that of the log-rotation shifter (by about 50 and 170 μm^2, respectively). Specifically, the hardware resource consumption of our data-path parts (inverse butterfly network) is identical to that of the log-rotation shifter. This means that the remaining area is due to the control-bit generation circuits. Therefore, the proposed algorithm for rotations based on the inverse butterfly network yields higher speed and has a simpler hardware implementation than Hilewitz and Lee (2009)'s design.

5 Conclusions

In this study, we have designed a novel rotation unit called HERRU with better functionality and higher efficiency than previously proposed methods. The unit can not only support 64-bit Single Instruction Single Data (SISD) stream instructions (bi-directional rotation operations), but also support MultiMedia eXtensions/Streaming SIMD Extensions

(MMX/SSE) instructions (parallel sub-word rotation operations) (Intel Corporation, 2015). We can also add some special control-bit generation circuits to support a series of complex bit operation instructions, such as GPR, PEX, and PDEP. Thus, it is a much more powerful functional unit with a wide range of prospective applications. We recommend this powerful unit to be embedded in the PEX hardware circuit, which has been integrated into the Intel Haswell processor. We also hope to promote the research of a new rotation shifter with other more complex permutation operations.

References

Bansod, G., Raval, N., Pisharoty, N., 2014. Implementation of a new lightweight encryption design for embedded security. *IEEE Trans. Inform. Forens. Secur.* **10(1)**:142-151. https://doi.org/10.1109/TIFS.2014.2365734

Belazi, A., El-Latif, A., Belghith, S., 2016. A novel image encryption scheme based on substitution-permutation network and chaos. *Signal Process.*, **128**:155-170. https://doi.org/10.1016/j.sigpro.2016.03.021

Chang, Z., Hu, J., Zheng, C., *et al.*, 2013. Research on shifter based on iButterfly network. *Commun. Comput. Inform. Sci.*, **396**:92-100. https://doi.org/10.1007/978-3-642-41635-4_10

Dai, H., Shen, X., 2007. Rearrageability of the 7-stage 16×16 shuffle exchange network. *Acta Electron. Sin.*, **35**(10): 1875-1891 (in Chinese).

Hilewitz, Y., Lee, R.B., 2006. Fast bit compression and expansion with parallel extract and parallel deposit instructions. IEEE Int. Conf. on Application-Specific Systems, p.65-72. https://doi.org/10.1109/ASAP.2006.33

Hilewitz, Y., Lee, R.B., 2007. Performing advanced bit manipulations efficiently in general-purpose processors. IEEE Symp. on Computer Arithmetic, p.251-260. https://doi.org/10.1109/ARITH.2007.27

Hilewitz, Y., Lee, R.B., 2008. Fast bit gather, bit scatter and bit permutation instructions for commodity microprocessors. *J. Signal Process. Syst.*, **53**(2):145-169. https://doi.org/10.1007/s11265-008-0212-8

Hilewitz, Y., Lee, R.B., 2009. A new basis for shifters in general-purpose processors for existing and advanced bit manipulations. *IEEE Trans. Comput.*, **58**(8):1035-1048. https://doi.org/10.1109/TC.2008.219

Hilewitz, Y., Shi, Z.J., Lee, R.B., 2004. Comparing fast implementations of bit permutation instructions. Proc. 38th Annual Asilomar Conf. on Signals, Systems, and Computers, p.1856-1863. https://doi.org/10.1109/ACSSC.2004.1399486

Intel Corporation, 2006. Intel Itanium Architecture Software Developer's Manual, Vol. 3, Rev. 2.2. https://www.intel.com/content/www/us/en/processors/itanium/itanium-architecture-vol-3-manual.html [Accessed on April 5, 2016].

Intel Corporation, 2015. Intel® 64 and IA-32 Architectures Software Developer's Manual. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf [Accessed on April 10, 2016].

Jolfaei, A., Wu, X., Muthukkumarasamy, V., 2015. On the security of permutation-only image encryption schemes. *IEEE Trans. Inform. Forens. Secur.*, **11**(2):235-246. https://doi.org/10.1109/TIFS.2015.2489178

Lee, R.B., Shi, Z.J., Yin, Y.L., *et al.*, 2004. On permutation operations in cipher design. Int. Conf. on Information Technology: Coding and Computing, p.569-577. https://doi.org/10.1109/ITCC.2004.1286714

Nassimi, D., Sahni, S., 1981. A self-routing benes network and parallel permutation algorithm. *IEEE Trans. Comput.*, **30**(8):332-340. https://doi.org/10.1109/TC.1981.1675791

Pillmeier, M.R., 2002. Barrel Shifter Design, Optimization, and Analysis. MS Thesis, Lehigh University, Bethlehem, Pennsylvania.

Pillmeier, M.R., Schulte, M.J., Walters, E.G.III, 2002. Design alternatives for barrel shifters. Proc. Advanced Signal Processing Algorithms, Architectures, and Implementations, p.436-447. https://doi.org/10.1117/12.452034

Rajkumar, S., Goyal, N.K., 2014. Design of 4-disjoint gamma interconnection network layouts and reliability analysis of gamma interconnection networks. *J. Supercomput.*, **69**(1):468-491. https://doi.org/10.1007/s11227-014-1175-0

Saraswathi, P.V., Venkatesulu, M., 2014. A block cipher based on Boolean matrices using bit level operations. IEEE/ACIS 13th Int. Conf. on Computer and Information Science, p.59-63. https://doi.org/10.1109/ICIS.2014.6912108

Sayilar, G., Chiou, D., 2015. Cryptoraptor: high-throughput reconfigurable cryptographic processor. IEEE/ACM Int. Conf. on Computer-Aided Design, p.155-161. https://doi.org/10.1109/ICCAD.2014.7001346

Semiconductor Manufacturing International Corporation, 2012. SMIC 55nm Low Leakage Logic Process Standard Cell Library Databook v2.0. http://www.smics.com/eng/design/libraries_smic.php

Shi, Z.J., Yang, X., Lee, R.B., 2008. Alternative application-specific processor architectures for fast arbitrary bit permutations. *Int. J. Embed. Syst.*, **3**(4):219-228. https://doi.org/10.1504/IJES.2008.022393