SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

April 2016, Vol. 59 042403:1–042403:16
doi: 10.1007/s11432-015-5381-z

Dynamically reconfigurable architecture for symmetric ciphers

Bo WANG & Leibo LIU*

National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, *China*

Received April 2, 2015; accepted May 8, 2015; published online March 1, 2016

Abstract In this paper, a very large scale integration (VLSI) architecture for a reconfigurable cryptographic processor is presented. Several optimization methods have been introduced into the design process. The interconnection tree between rows (ICTR) method reduces the interconnection complexity and results in a small area overhead. The hierarchical context organization (HCO) scheme reduces the total context size and increases the dynamic configuration speed. Most symmetric ciphers, including AES, DES, SHACAL-1, SMS4, and ZUC, can be implemented using the proposed architecture. Experimental results show that the proposed architecture has obvious advantages over current state-of-the-art architectures reported in the literature in terms of performance, area efficiency (throughput/area) and energy efficiency (throughput/power).

Keywords reconfigurable cryptographic architecture, symmetric cryptography, algorithm flexibility, performance, area efficiency, energy efficiency

Citation Wang B, Liu L B. Dynamically reconfigurable architecture for symmetric ciphers. Sci China Inf Sci, 2016, 59(4): 042403, doi: 10.1007/s11432-015-5381-z

1 Introduction

Symmetric ciphers [1] are generally used to maintain the confidentiality of information when it is traveling through networks. However, the real demands of network security offer tough challenges for symmetric cryptographic architecture implementation. First, as the quest for higher system speeds is never-ending, security networks at the Gbps level, 10 Gbps level or even 100 Gbps level [2,3] present new performance challenges. Additionally, the corresponding power consumption should also be taken into account under such high throughput conditions. Second, algorithm flexibility is often a requirement. Many security protocols, such as IPsec (Internet Protocol Security), allow a variety of optional algorithms and real-time switching of these algorithms [4]. In addition, there are also applications that require existing algorithms to be modified or upgraded by changing certain key parameters or components, such as the substitution box (S-box). Traditional architectures such as general purpose processors (GPPs) and application-specific integrated circuits (ASICs) can be used to implement symmetric ciphers. Software implementations using GPPs offer ease of design, use and portability. However, the execution speed is much slower than that of hardware implementations and, while ASICs can achieve the maximum theoretical performance, ASIC functions cannot be changed after fabrication, which demonstrates a complete lack of flexibility.

* Corresponding author (email: liulb@tsinghua.edu.cn)

Reconfigurable architectures can not only produce significantly higher performance levels than software implementations, but can also perform algorithm switching and upgrading during runtimes. Also, reconfigurable architectures typically consist of many structurally-similar processing units. It is therefore difficult to obtain the algorithm information directly by reverse analysis of the circuit, which can be implemented in ASICs in cases where the algorithm itself must be kept confidential. All these benefits mean that reconfigurable architectures for symmetric ciphers are attracting increasing attention.

Researchers have made considerable efforts to explore reconfigurable architectures for symmetric ciphers. As a special case of fine-grained reconfigurable architectures, field programmable gate arrays (FPGAs) are widely used to implement symmetric ciphers [5–11]. The FPGA is a commercially successful reconfigurable architecture that uses look-up tables (LUTs) and fine-grained interconnection networks to meet the diverse function requirements of symmetric ciphers. In addition to FPGAs, numerous other reconfigurable architectures are available with middle grain, coarse grain or even mixed grain. These architectures have all been proposed for implementation of symmetric ciphers, such as ADRES [12, 13], XPP-III [14,15], MorphoSys [16], DREAM [17], and COBRA [18]. These works improve the performance levels of symmetric ciphers to varying degrees under the premise of ensuring algorithm flexibility. However, the real demands for symmetric cryptographic implementation are still not being met. FPGAs are general architectures that are not specifically designed for symmetric cipher implementation. Their excessive hardware redundancy results in relatively low area efficiency (i.e., throughput/area) and low energy efficiency (i.e., throughput/power). For example, the interconnections in FPGAs can account for up to 90% of their total area and up to 85% of the total power consumption [19]. This would increase both the chip area and the power when symmetric ciphers are implemented. Additionally, the other reconfigurable architectures that are currently in use are intended to fit various applications rather than cryptographic algorithms alone. Their design spaces have not yet been fully explored for symmetric cipher applications and their performance and area efficiency must be improved.

This paper presents a very large scale integration (VLSI) architecture for a reconfigurable processor for symmetric ciphers. Several optimization methods have been introduced in the design of this architecture. First, the interconnection tree between rows (ICTR) method effectively reduces the complexity of the interconnections based on the characteristics of the symmetric ciphers. This technique ensures that the ratio of the interconnection area to the total area remains approximately constant, i.e., it does not increase with an increase in the array scale. Second, a context group scheme called hierarchical context organization (HCO) is introduced, in which the contexts are divided into three levels. Using an index, higher level contexts can call lower level contexts. In this way, the higher level contexts can reuse the lower level contexts. This avoids duplication of contexts, such that it reduces the total size of the contexts and results in a fast and dynamic configuration. The proposed architecture is suitable for processing of most symmetric ciphers and hash functions. The Advanced Encryption Standard (AES) [20], the Data Encryption Standard (DES) [21], SHACAL-1 [22], SMS4 [23] and ZUC [24] have already been implemented using this architecture. The results demonstrate that the performance, area efficiency and energy efficiency of this architecture offer obvious advantages over the state-of-the-art architectures described in the literature.

2 Algorithm analysis

Symmetric ciphers can be further divided into block ciphers and stream ciphers. The block cipher takes a single block, such as 64 or 128 bits of data, as a processing unit. It then generally uses multiple iterations to improve its security, i.e., it uses identical or similar operations named round functions to iterate several times to obtain the final cipher text. Many block ciphers can be characterized as Feistel networks [1]. A basic Feistel network, as shown in Figure 1(a), divides the 2w-bit data into two halves. At the point when the f-function is dealing with one half of the 2w-bit data using a sub-key, the block cipher makes the output of the f-function perform an exclusive-OR with the other half of the data. Many types of operation can be found within the f-function to implement confusion and diffusion in

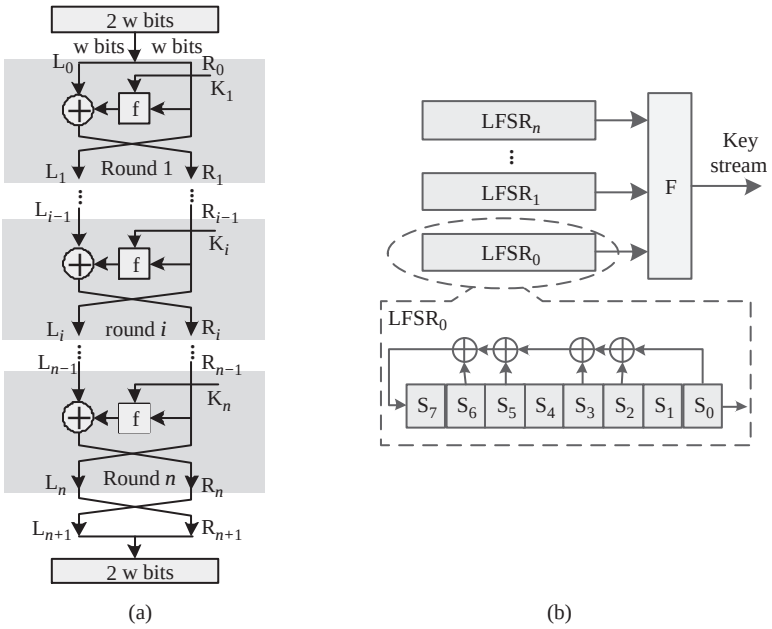

Figure 1 Block diagrams of symmetric ciphers. (a) Block cipher based on basic Feistel network; (b) LFSR-based stream cipher.

cryptography. Many of these operations have distinct characteristics and some of them are hardly seen in other type of algorithm. For example, a 32-bit permutation can make every bit in 32-bit input data exchange with each other in a given order to obtain 32-bit output data. Additionally, the granularities of the data processing for these operations have several levels, i.e., the levels for the block lengths (e.g., 64-bit or 128-bit) used in permutation operations, the levels for large byte lengths (e.g., 8-bit, 16-bit, 32-bit), and the levels for small byte lengths (e.g., 4-bit, 6-bit). The stream cipher implements encryption by performing plaintext stream exclusive-OR operations with the key stream. The kernel of the entire process is the generation of the key stream. A general model for stream ciphers can be built using a linear feedback shift register (LFSR). Figure 1(b) shows such a model, which consists of one or several LFSRs and a nonlinear function F. Here, an eight-stage LFSR is used as an example, and it can be divided further into exclusive-OR and shift operations. The function F is very important in ensuring the security of the algorithm. An increasingly complex function allows the corresponding algorithm to achieve higher levels of security. The types of operations in the function F are almost the same as those in the f-function of the Feistel network. As an example, ZUC is a LFSR-based stream cipher. Its function F includes basic arithmetic and logic operations, LUTs and rotation operations.

3 Basic architecture

The framework of the proposed reconfigurable architecture is shown in Figure 2(a) and further details of the reconfigurable cell array (RCA) are shown in Figure 2(b). The reconfigurable architecture consists of two components: the data path and the configuration path. (1) The data path contains the RCA and the data memory. The RCA, as the core component of the entire reconfigurable architecture, is mainly responsible for data flow calculation. It consists of reconfigurable cells (RCs) and the interconnections. The data memory, which includes the input/output FIFO (first-in first-out) circuit and the inner buffer, is responsible for storage of the inputs, outputs and intermediate data of the RCA. (2) The configuration path consists of the context memory and the corresponding control logic. The context memory is responsible for storage of the configuration information. The control logic is responsible for analysis and distribution of the contexts to the data path to control its function.

The data path design directly affects the performance of cryptographic algorithms, and can be summarized in two aspects, i.e., the RC design and the interconnection design. In the following, the RC

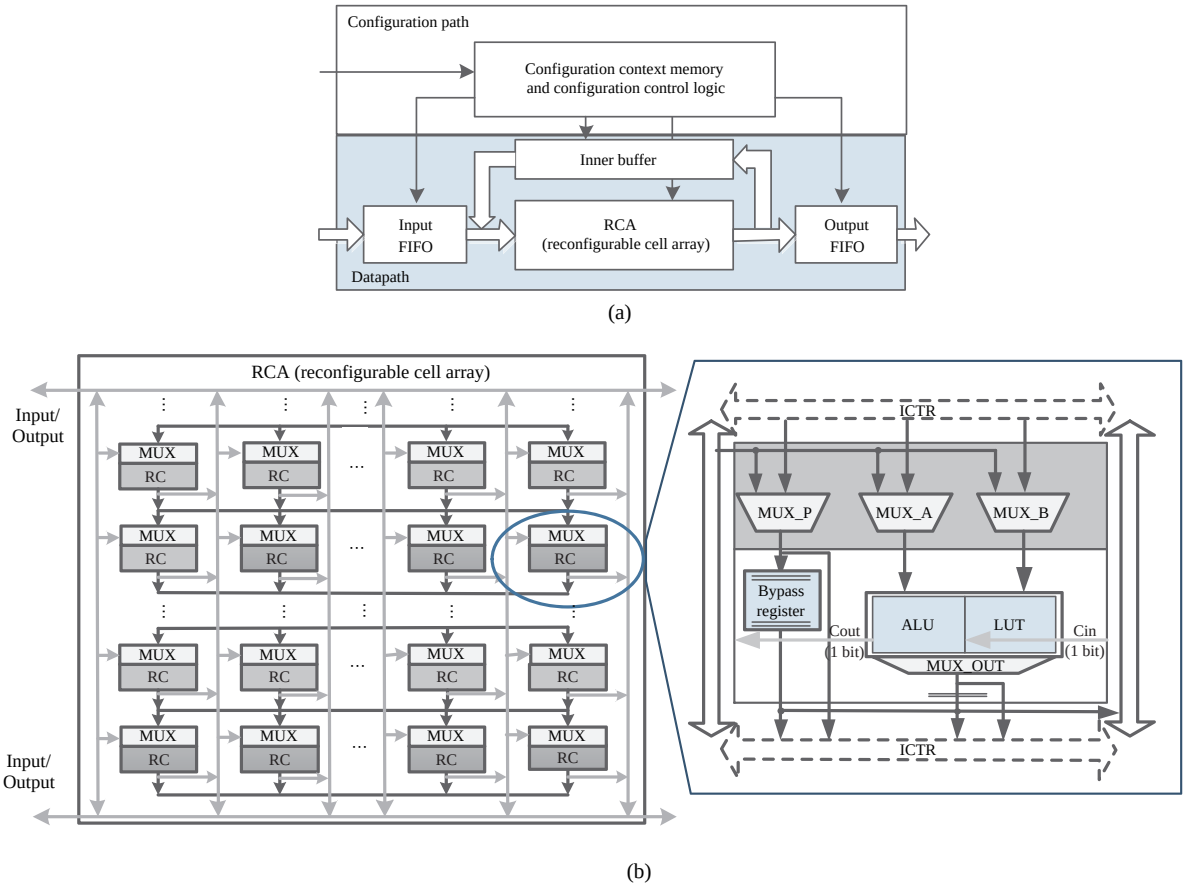

Figure 2 Proposed reconfigurable architecture. (a) The framework of the reconfigurable architecture; (b) the reconfigurable cell array architecture.

design is illustrated first. Because the RC architecture is not the main focus of this paper, only a brief discussion is given. The general RC architecture consists of an arithmetic logic unit (ALU) and corresponding registers, and the design is similar among various reconfigurable architectures [25]. The main design variables for RCs are processing granularity and the operation set of the ALU. During selection of the granularity, trade-offs should be made between flexibility and area efficiency. Small granularity means a more flexible architecture that allows more complex connections between finer processing units, while at the same time resulting in larger area requirements. In this paper, the goal is to ensure sufficient flexibility for symmetric implementations while maintaining area efficiency as far as possible. Because the smallest granularity that is commonly used for symmetric ciphers is 4 bits (as mentioned in Section 2), the RC granularity is thus determined to be 4-bit. It should be noted here that the use of heterogeneous RCs with different granularities in the same array is not recommended. This is because the splicing and splitting of data would have to be done between the inputs and outputs of RCs with different granularities, and this would make the design of the interconnections very complicated. Also, the mapping process for the algorithms would be more complex than that of the homogeneous case. The operator set of the ALU is determined by the operations in the symmetric ciphers. The sets can be classified into the following four types. (1) Basic logic operators, including XOR, AND, OR, and NOT. (2) Modular arithmetic, including modular addition, and modular subtraction. (3) Galois field operators, including multiplication, inversion, and affine transformations. (4) Other operators, including bypass, shift, rotation, and operations (such as shift or addition) with conditions. The proportions of the four types in the operation set are also relatively balanced, and are all at a proportion of 20%–30%. Each operator in the ALU corresponds to an operation code (opcode). The ALU can be used to execute one of these functions by configuring specific bits in the context with the corresponding opcode. It should also be noted that a LUT is also added to an RC to accomplish the required substitution operations in

cryptographic algorithms, and can logically be seen as an operator of the ALU.

Adjacent RCs in the same row can be used to accomplish operations with granularities that are larger than 4 bits. This is illustrated using the following two examples of modular addition and LUT-based substitution. The modular addition operations in symmetric ciphers can be represented by $(p + q)$ mod 2^n, where p and q are two n-bit binary numbers. Compared with the commonly-used addition operation, the Cout of the most significant bit and the Cin of the least significant bit in modular addition are always zero, which means that the sum of two n-bit numbers is also n bits. The carry-select adder is used to accelerate modular addition in the proposed architecture. The 4-bit addition operator in each RC (with an additional carry path and a multiplexer) is one stage of the carry-select adder, and can be further connected using the carry bits. In the proposed architecture, the width limit for the modular addition operation is 32 bits, i.e., addition operations with widths ranging from 4 bits (modular 2^4) to 32 bits (modular 2^{32}) can be accomplished using 1 to 8 adjacent RCs directly. In addition to the modular addition operation, the 256-bit LUTs in each of the RCs can also be combined to accomplish larger substitutions. For example, eight nearby RCs can produce an 8-8 (8-bit input and 8-bit output) LUT in AES. In LUTs, the input can be regarded as the address of the desired output data. Taking a $u - v$ LUT as an example, one v-bit output is selected from the 2^u groups of v-bit data. By using multiplexers as hardware-assisted logic (for details, please refer to the methods used in FPGAs [26]), a LUT with a storage capacity of M-bits can be organized into a $u - v$ LUT, where $2^u \times v \leqslant M$. In this case, the 256-bit LUT in each RC can be seen as an 8-1 LUT, where $2^8 \times 1 = 256$. An 8-8 LUT can be achieved by splicing the 1-bit results of eight adjacent RCs together. This implementation detail of the LUTs is transparent to the other RCs, which means that the 8-bit output is split into two 4-bit data outputs and can be used as the ordinary outputs of these RCs.

In the data path, the interconnection design is crucial in the process of ensuring high area efficiency. To increase the proportion of actual computing resources (RCs) in the RCA, the main design objective for the interconnections is to minimize the area overhead under the premise of guaranteeing processing efficiency. In Section 4, a method of using multiplexers (MUXs) to represent the interconnection overhead is introduced, and an optimized interconnection method (ICTR) is determined after a series of discussions.

The main objective of the configuration path design is to increase the dynamic configuration speed and reduce the area overhead. This objective is mainly affected by the size of the configuration contexts. A large context size would increase the context transformation time for each configuration and the storage space of the on-chip context memory. In Section 5, a novel context organization scheme (HCO) is introduced, and a discussion of how to use the characteristics of symmetric ciphers to reduce context duplication is given.

4 Interconnection design

If the RCs in the reconfigurable array are abstracted into vertices and the interconnections between them are abstracted into directed edges, then the interconnections in the reconfigurable array can be abstracted into a directed graph. All the possible interconnection routes between the RCs form a directed graph set, and with increasing array scale, the number of elements in the set becomes so large that it becomes difficult to conduct a comprehensive analysis of all the elements. Here, a subset is selected for examination, i.e., an interconnection structure with one direction. In this structure, each row of RCs is taken as a single stage and numbered in ascending order, from the top to the bottom of the array. The RCs in each stage can only take the outputs of RCs with smaller stage numbers as their inputs. The selection of this subset is reasonable because it coincides with the singular direction of the mapping graphs of the symmetric ciphers.

To quantify the area overhead and thus ease comparisons among the different kinds of interconnections, a method of using the number of 2-to-1 MUXs to represent the interconnection overhead is introduced. In a reconfigurable array, the logic circuit overhead of the interconnection structure mainly comes from MUXs. For example, assume that there are two rows of RCs and that the number of RCs in the first and

second rows are 2^a and 2^b, respectively. If the inputs of the RCs in the second row can come from the output of any RC in the first row, then the number of 2^a-to-1 MUXs that is needed is 2^b. Noting that, $\sum_{i=0}^{a-1} 2^i$ named (2^a-1), 2-to-1 MUXs are required to achieve the same function as that of the 2^a-to-1 MUX, and then a total number of $(2^a-1)\times 2^b$ 2-to-1 MUXs are required to connect the two rows. Although the actual logic circuit for the interconnection may not be built with 2-to-1 MUXs in the technology library, this calculation method can still allow designers to have a clear understanding of the interconnection overhead in the early stages of the design.

The most flexible interconnection of the subset is that where the inputs of the RCs of each stage can come from the outputs of the RCs of any stage with a smaller stage number. However, a high area cost must be paid for this flexibility. The 2-to-1 MUX overhead (represented by α) is obtained as shown in (1) for a reconfigurable array that consists of 2^x columns and 2^y rows of RCs. For a square array ($x=y$), Eq. (2) shows the ratio factor β relating the interconnection area and the area of the RCs (assuming that the area of each RC is constant and equal for every RC). β is approximately proportional to the square scale of the rows (or columns). A slight increase in the array scale can cause a substantial increase in the ratio of the interconnection resource to the total array resource. Ultimately, this would lead the reconfigurable architecture towards a situation similar to that in FPGAs, i.e., excessive interconnection resource redundancy that sharply reduces the area efficiency. Another possible and widely used interconnection structure is full interconnection between rows, i.e., where each RC in one stage can interconnect with any RC in the adjacent stage. This form of interconnection greatly reduces the area overhead by removing direct connections between nonadjacent rows. Eq. (3) shows the 2-to-1 MUX overhead of this type of interconnection. For a square array, when assuming that $2^x \gg 1$, β follows the relationship shown in (4) and is approximately proportional to the scale of the rows. While this growth is much slower when compared with that of the previous example, β will still be high when the array scale increases to a certain extent.

$$\begin{aligned}\alpha &= (1+2+3+\cdots+(2^y-1))\times((2^x-1)\times 2^x)\\ &= \frac{2^y\times(2^y-1)}{2}\times(2^x-1)\times 2^x\\ &= 2^{y-1}\times(2^y-1)\times(2^x-1)\times 2^x, \end{aligned} \tag{1}$$

$$\beta \propto \frac{2^{y-1}\times(2^y-1)\times(2^x-1)\times 2^x}{2^x\times 2^y} \propto \frac{(2^x-1)^2}{2}, \tag{2}$$

$$\alpha = (2^y-1)\times\left((2^x-1)\times 2^x\right), \tag{3}$$

$$\beta \propto \frac{(2^y-1)\times(2^x-1)\times 2^x}{2^x\times 2^y} \propto 2^x, \tag{4}$$

$$\alpha = (2^y-1)\times\left((2^z-1)\times 2^x\right), \tag{5}$$

$$\beta \propto \frac{(2^z-1)\times 2^x\times(2^y-1)}{2^x\times 2^y} \propto \text{const.} \tag{6}$$

Strong data locality can be found in the mapping graphs through the mapping of varieties of ciphers, which means that intensive data exchanges only exist between relatively close RCs. On this basis, an optimized interconnection format called ICTR is introduced. The input of each RC in ICTR is only taken from the outputs of nearby 2^z RCs in the adjacent row. The characteristics of the symmetric ciphers determine the value of z, which is a constant that is independent of the array scale. The corresponding α and β are shown in (5) and (6) respectively. β is a constant and does not increase with increasing array scale. After analysis of a large number of the mapping graphs of the symmetric ciphers, it was found that most of the intensive data exchanges are within the nearest 32 bits. The option in this case is that each RC interconnects with the nine adjacent RCs (4-bit granularity for each RC) in the previous row. This allows flexible interaction among the nearest 32-bit data and ensures the symmetry of the interconnection topology at the same time. While the value of 9 here is not a power of 2, it can apparently be used in the above formula by simply replacing 2^z with 9. Figure 3 shows a schematic diagram of ICTR. Note that in Figure 3, the corresponding interconnections of two RCs are given for clarity of illustration only.

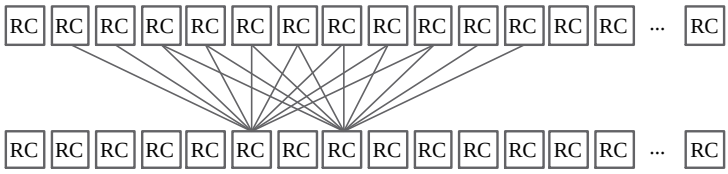

Figure 3 ICTR structure.

It should be noted here that the existing interconnections cannot support bit-wise permutations and irregular rotations very well. While these operations can theoretically be accomplished through the cooperation of RCs and ICTR, the operations take many clock cycles and reduce the overall performance. It is therefore necessary to add functional units to specifically accelerate these operations. The proposed functional unit is called a bit reformer, and it can achieve bit-wise permutation and rotation efficiently based on the Benes network architecture (a type of multistage switching network) [27]. Because ICTR is a row-based interconnection with a single direction, bit reformers can be added directly between some of the rows of the array. The data processing width (i.e., the overall width of the input/output data) of bit reformers can be set to 128 bits to coincide with the data width of a single RC row. It is then very straightforward to map the permutations or rotations onto the bit reformer. A 64-bit permutation is taken as an example to illustrate this process. If the 64-bit data is input into the lower bits (in fact, any 64 consecutive bits are fine) of the bit reformer, then the corresponding bits of the output are the desired data after permutation. To generate the configuration context of the bit reformer automatically, a dedicated program can be written in a high-level language (such as C++ or MATLAB). By taking the bit-order in the permutation as the input, the binary configuration bit-stream can then be output by the program. In fact, these types of programs can already be found as open source codes that aim to support the usage of non-blocking networks such as Benes networks.

The advantage of the proposed interconnection can be further illustrated through comparison with the interconnections of other reconfigurable architectures. The interconnection details of most reconfigurable cryptographic architectures given in the literature are insufficient to complete the comparison. Here, ADRES and XPP-III are used as examples to illustrate this. ADRES and XPP-III are both claimed to be suitable for applications like cryptography [13, 15]. The interconnection of ADRES consists of a basic mesh network and some optional extra interconnections [28]. The interconnection area overhead in ADRES is relatively small, and is at the same level as ICTR. In ICTR, each RC is connected with the nine adjacent RCs in the next row; in ADRES with nearest neighbor and next hop connections, the functional unit (FU) is connected to eight FUs, including four neighbor FUs and four hop FUs. The interconnection in ADRES does not follow a particular direction, which makes the data transmission more flexible. However, operations such as irregular rotations and bit permutations cannot be accomplished using this interconnection. For each FU, the output data can be transmitted in any one of the four directions of the array through the interconnection. Functional modules such as bit reformers, which have a data processing width that is equal to that of multiple RCs, do not match up with this kind of interconnection and therefore cannot be added directly to the array. These operations can only be performed in the very long instruction word (VLIW) processor part of ADRES and ultimately reduces the overall performance. The interconnection of XPP-III is mainly composed of two-dimensional buses. Packet-oriented routing buses are used for point-to-point connections, and they can be segmented using configurable switch objects. This bus connection can connect any two of the objects in a bus, which leads to good algorithm flexibility. However, it occupies a relatively large area because of the complexity of the packet-oriented architecture. Also, the switch object in XPP-III adds a register delay by itself. This would introduce many extra cycles for data communication between nonadjacent processing elements.

As an extension of this discussion, the influence of granularity (where the granularity of the RC is the same as that of the interconnection) on the interconnection overhead is illustrated. For a specified data processing capacity (i.e., where both the data processing width and the data processing depth of the reconfigurable array are constants), the interconnection overhead varies under different granularities. The interconnection between rows in a single-direction interconnection subset, such as full interconnection

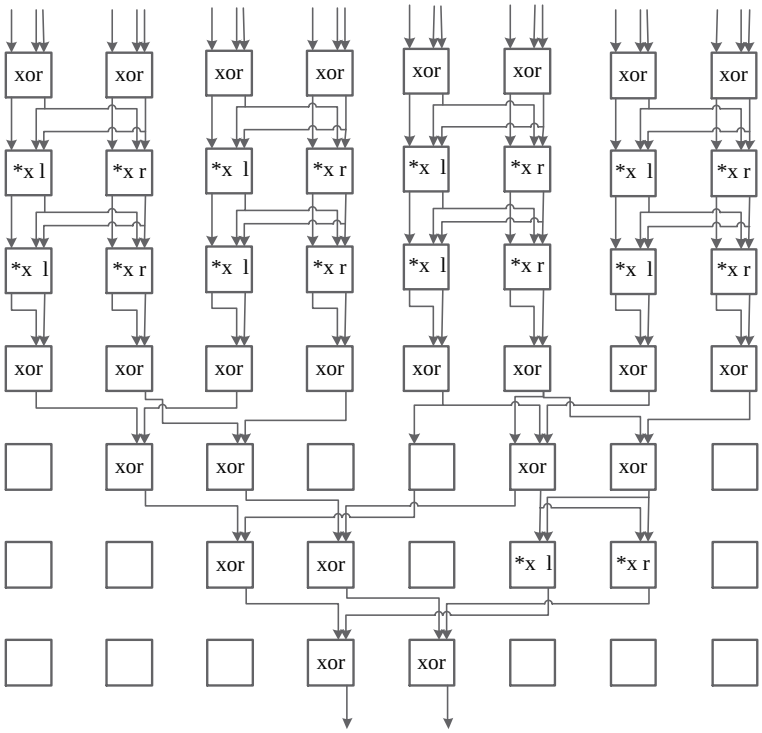

Figure 4 Mapping block of Inverse MixColumns in AES (for one byte of the state).

between rows or ICTR, is used as an illustrative example. The total interconnection overhead is the product of the array scale and the overhead for each RC. First, the relationship between granularity and the scale of the array must be determined. Because the data processing width is the product of the granularity (represented by G) and the number of columns in the array, the number of columns must then be proportional to $1/G$. Unlike the number of columns, the number of rows is equal to the data processing depth and remains constant. Then, the scale of the array is proportional to $1/G$. For each RC, the interconnection overhead is proportional to the width of the data that are connected directly to it. This width is equal to the product of G and the number of other RCs connected to the RC of interest (i.e., the number of inputs for the MUX). In the case of interconnection between rows, the width of the data connected to each RC is constant. For example, this value is equal to the data processing width of the array for full interconnection between rows, and is roughly equal to 32 bits in the case of ICTR for symmetric ciphers. Therefore, the interconnection overhead for each RC is a constant. In this case, the total interconnection overhead is then proportional to $1/G$. This means that the overhead decreases with increasing granularity. As mentioned in Section 3, the disadvantage of higher granularity is a lack of flexibility. In this case, mapping efficiency decreases for algorithms with fine-grained operations. For example, when a 4-bit operation is executed on a 16-bit RC, 75% of the computing resources may be wasted. In this paper, sufficient flexibility for common symmetric ciphers and even confidential symmetric ciphers for national defense or other special applications must be ensured first. Therefore, 4-bit is chosen as the granularity in this case. However, for other designs that have target sets of algorithms that have larger granularities, the researcher may then select a higher granularity for the array to perform a trade-off between flexibility and area overhead.

Finally, an example of partial mapping graphs is given to better illustrate the way in which the algorithms are accomplished using ICTR. Figure 4 shows the mapping block for Inverse MixColumns for AES. Inverse MixColumns is the primary source of diffusion in the AES round function. For different bytes of the state (where the state is the intermediate result of AES), operations in Figure 4 should be executed in parallel. The opcodes that are marked *xl or *xr denote the left half part or the right half part of the multiplication of x in GF(2^8), respectively. The connections in the mapping graph show strong data locality and can be supported very well by ICTR.

5 Organization form of configuration contexts

In most cases, the entire mapping graph of a single algorithm is several times larger than the dimension of the RCA, so dynamic reconfiguration is needed. The most important factor in the configuration path design is determination of the organization form of the configuration contexts, which will affect the total size of the contexts and the speed of dynamic configuration.

The iterative nature of cryptographic algorithms should be used to its full advantage in this design process. Without loss of generality, it can be assumed that a specific cryptographic algorithm has i different kinds of round functions, and each round function should be iterated mi times. For example, if a cipher has ten rounds, and the first round and the last round are different from the other rounds, then we have $i = 3$ and $m1 = 1$, $m2 = 8$, and $m3 = 1$. When mapped onto the RCA, one round function can be further divided into several stages by the registers of the RCs, which means that several data groups can be encrypted in a pipeline during a single iteration. Among the mi iterations of a single round function, the operations inside the round function are the same, but there are differences among the addresses of the processed data. This can be illustrated by the following three examples. (1) RCA input data come from the input FIFO for the first round, but then come from the inner buffer for the middle rounds. (2) When stored in the inner buffer, the immediate data from the different pipeline stages usually adopt different addresses to avoid overwriting of the valid data. (3) Some of the fixed values (e.g., the round keys and the round constants) also differ among the mi iterations. All these aspects make the configuration contexts for the mi iterations different to each other. In this case, if the configuration of the data input, the data output, the intermediate data buffer and the functions of RCA are integrated into a single context, mi iterations require mi different contexts. There will be a great deal of duplicate information (e.g., the configuration information about the functions of the RCs and the interconnections) stored between the different contexts and this will consume a great deal of time and power when the configurations are switched. By taking these factors into account, a new scheme called HCO is introduced.

Figure 5(a) shows a schematic view of the HCO scheme. The contexts are divided into three levels, and are called level by level by using the index. The data exchange contexts are separated from the contexts of the RCA itself, and in this way, the contexts of one level can be reused by the contexts of a higher level. This avoids the duplication of contexts, thus reducing the total size of the contexts, and leads to a fast and dynamic configuration. Among the three levels of contexts, level 2 is the core context, and mainly contains several row contexts. Row context is the basic context, and contains the configuration information of one row of RCs and the corresponding interconnections. It means that the RCA configuration can be switched by row. In this case, the array can be reconfigured row by row like waves. After one row finishes its calculation, the reconfiguration of this row can take place at the next clock cycle and does not need to wait until the calculation of the whole array is complete. Therefore, the impact of dynamic configuration on pipeline efficiency in the array is minimized. The configuration contexts of the LUTs and the bit reformers are separated with the row contexts. These contexts generally only exist in one core context of the entire algorithm. Level 1 is the group context, and mainly consists of multiple group context kernels that contain the internal data exchange information in the data path, the sequence control for the core context and the index of the core context. The sequence control part contains the sequence information required to operate the core context properly, including the number of iterations, and the required interval between iterations. The internal data exchange part contains the data exchange information between the RCA and the internal memory (i.e., the input/output FIFO and the inner buffer), which mainly consists of the address information. Level 0 is the top context, and corresponds to a unique algorithm. The top context contains the data exchange information between the entire reconfigurable architecture and the external memory, which mainly controls the transmission of plaintexts and ciphertexts.

Figure 5(b) shows the simplified hardware structure of the configuration path used to support the HCO scheme, in which many details are omitted to highlight the main structural framework. Group context memory (GCM) and core context memory (CCM) are used to store group contexts and core contexts, respectively, and the configuration interface, the group context parser and the core context parser are

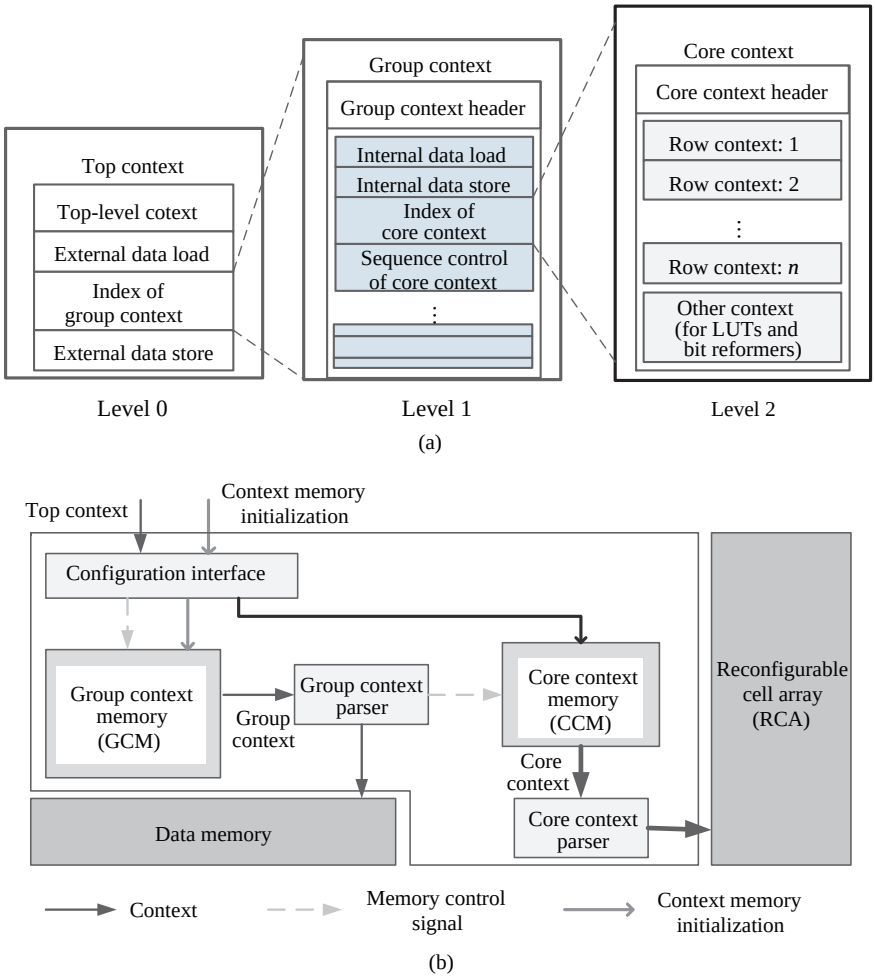

Figure 5 HCO. (a) Schematic view of HCO scheme; (b) architecture of configuration path supporting the HCO scheme.

Table 1 Size of configuration contexts in HCO

Hierarchical context organization			Size (word)
Top context	Total		5
	Units	Top context kernel	4
		Index	1
Group context	Total		$4+5u$[a)]
	Units	Group context head	4
		Group context kernel	$4+1$[b)]
Core context	Total		$30n+256m+56i+40j$[c)]
	Units	Row context	30
		Others	256 for LUTs[d)] 56/40 for bit reformers

a) u denotes the number of group context kernels.

b) Four words for the address/sequence control and one word for the index.

c) $0 \leqslant n \leqslant 16; 0 \leqslant m \leqslant 8; i, j \in \{0, 1\}$.

d) Although 256 is much larger than 30, the contexts of the LUTs usually only exist in one core context of the entire algorithm, and thus the row contexts still dominate in size.

used for analysis and distribution of the contexts at each level.

By using HCO, the size of the overall contexts can be reduced significantly. Tables 1 and 2 show the sizes of the configuration contexts for HCO and further compare them with the sizes of the contexts for the non-hierarchical case. Because of the removal of the duplicated contexts, the overall context has a

Table 2 Size of configuration contexts in different cases

Task	Non-hierarchical	Proposed HCO				Reduction (%)
	Overall context (word)	Top context (word)	Group context (word)	Total core context (word)	Overall context (word)	
AES	11444	5	99	1864	1968	82.8
DES	7536	5	54	892	988	86.9
SHACAL-1	21928	5	204	1188	1397	93.6
SMS4	6440	5	44	832	881	86.3

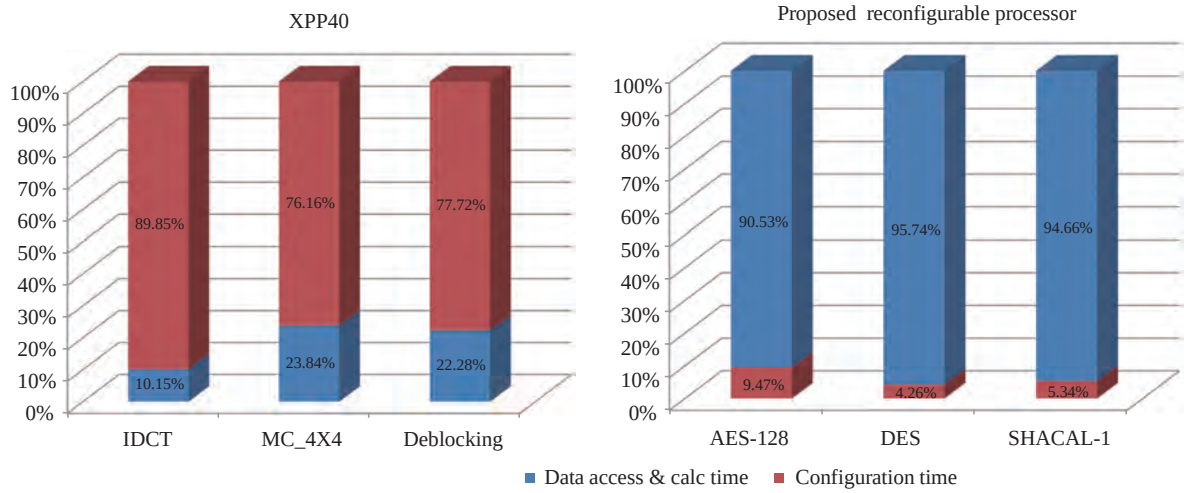

Figure 6 Proportions of configuration times and data access & calculation times for XPP40 and for the proposed reconfigurable architecture.

different degree of reduction that ranges from 82.8% to 93.6% for the various algorithms. When a smaller context size is used, the configuration speed of the proposed architecture can be greatly increased. When coupled with the ability to reconfigure by row, the influence on the pipeline efficiency is minimized and fast and efficient dynamic reconfiguration can finally be achieved. The reason why we mainly discuss block ciphers here is that they have higher dynamic reconfiguration requirements because of their computational complexities, which means that the different bottom contexts usually need to be switched during execution of the algorithm.

Figure 6 shows the proportions of the configuration time and the data access and calculation time required for the proposed reconfigurable architecture and for the XPP40 [29]. Because the proportion of configuration time required for most reconfigurable cryptographic architectures is not mentioned in the literature, the XPP40 (when executing sub-tasks for H.264 decoding) is used as an example for comparison. While the symmetric cipher and the decoding algorithm are two different types of application, they are both computationally-intensive algorithms, and it also makes sense to compare the average proportions of their configuration times. As shown in Figure 6, the configuration time of the proposed reconfigurable architecture only accounts for 4.26%–9.47% of the total execution time. This time is much lower than that of XPP40. As a commercial architecture, the detailed organizational form of the contexts in XPP40 cannot be found in public literature. However, it is obvious that the serial configuration method used in XPP40 has a major influence on the dynamic configuration speed. The configuration of each processing array element (PAE) in XPP40 is performed sequentially with a delay of $R + C$ clock cycles, where R and C are the numbers of rows and columns of the target PAE, respectively. The contexts are loaded from the sole configuration port at one corner of the array. To configure a specific PAE, the contexts are first transferred to the desired row, and are then transferred to the corresponding column. In this case, the configuration affects the overall computing efficiency dramatically, which means that the calculation may be interrupted and forced to wait until the configuration is complete. Unlike the XPP40 configuration scheme, the proposed architecture can be configured row by row, which means that

Table 3 Implementation results of a REPROC core

Tech. (nm)	Frequency (MHz)	Array Scale (row × column)	Area (mm^2)	Throughput (Gbps)	
65	400	16×32	4.28	AES-128	2.16
				DES	2.72
				SHACAL-1	1.55
				SMS4	6.30
				ZUC	0.49

Table 4 REPROC architecture gate counts

Element	Gates
RCs	949120
Interconnections	259491
Data memories	108253
Context memories and controllers	819200
Total	2136064

Table 5 REPROC on FPGA (Stratix IV EP4SE820)

Element	RCs	Interconnections	Data memories	Context memories and controllers	Total
ALMs[a)]	55424	45525	2332	2268	105549
Block memory (bits)	74752	0	7680	397,312	479744

a) Adaptive logic modules (ALMs) are the basic logic units in Altera FPGAs (similar to slices in Xilinx FPGAs).

the parallel configuration of a single row can be performed in a single clock cycle. This would ultimately reduce the configuration cost and increase overall performance.

Because the context size is relatively large, a program is developed to automate context generation. In this program, the binary bit strings in the contexts are represented by a series of parameters to reduce the researcher's workload. For example, if the function of a specific RC in the array is exclusive-OR, then the researcher only needs to assign "*xor*" to the variable "*OPCODE*". This saves a great deal of manpower and time when compared with inputting of real bits into the contexts (such as "001101" for the opcode). In this case, to generate the desired contexts, the researcher should determine both the mapping graph and the algorithm execution sequence first, according to the hardware constraints. Using this information, the corresponding parameters that are required in these three context levels should then be input to the program. Finally, the program organizes these bit strings automatically to form a binary context file. It should be noted that the context generation tool for the bit reformer that was mentioned in Section 4 is also integrated into this program. Based on our experience, a researcher who is not very familiar with the proposed architecture can still complete the mapping of an algorithm and generate the contexts within a week.

6 Experiments and comparison

A reconfigurable architecture called REPROC (reconfigurable encryption processor core) was determined after the previous analysis. The synthesis and place-and-route processing of the core are performed using Synopsys tools. AES-128, DES, SHACAL-1, SMS4 and ZUC are all implemented in REPROC, and the results are shown in Table 3 and Table 4. The REPROC core is also implemented on an Altera FPGA, and the results are shown in Table 5.

As noted in Section 1, high throughput (e.g., dozens of Gbps) is usually required for network applications. Theoretically speaking, a single core with a very large array scale or multiple cores with a specified array scale can be used to meet such high performance requirements. However, in practice, the former is not a good choice for the following reasons.

Two parameters can be used to describe the array scale: the number of columns and the aspect ratio.

Table 6 Data path area efficiency for different array scales (row × column)

Area efficiency (Gbps/mm^2)	8 × 16	16 × 16	16 × 32	32 × 32
AES-128	0.64	0.46	0.83	0.94
DES	0.60	0.99	1.05	0.98
SHACAL-1	0.34	0.18	0.59	0.40
SMS4	1.16	1.22	2.42	1.30
ZUC	0.20	0.18	0.19	0.22

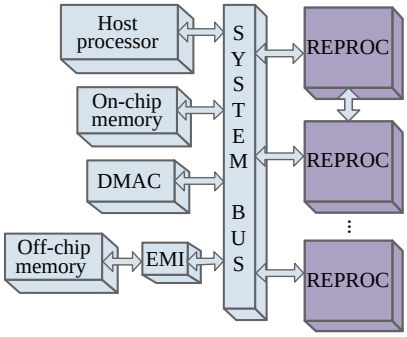

Figure 7 Multi-core mode.

The number of columns characterizes the data processing width of the array, and this parameter should coincide with the data width of the mapping graphs. If the number of columns is too small, then the mapping graphs must be divided into multiple small graphs to ensure that they map onto the array. However, this division reduces the pipeline efficiency and the parallelism. In contrast, when the number of columns is too large, the computational resources may be wasted and the difficulty of the place-and-route process may also increase dramatically. In addition to the number of columns, the other parameter that can be used is the aspect ratio, which is the ratio of the number of columns to the number of rows. Because the data stream in the array flows from top to bottom, this parameter represents the ratio of the data processing width to the data processing depth. For a certain number of columns, a small aspect ratio can mean a relatively large number of pipeline stages. However, because the area required for the input/output selection logic increases very rapidly with increasing numbers of rows (following a square relationship, for example), the number of parallel input/output interfaces typically does not grow with the same speed as the data processing depth. In this case, when the aspect ratio is small, the input/output data must be divided into multiple groups. This affects the pipeline efficiency greatly and ultimately reduces the throughput.

Because the data processing width for most commonly used symmetric ciphers is approximately 128 bits (e.g., AES-128, SMS4 and SHACAL-1) or less (e.g., 64 bits for DES and 32 bits for ZUC), the number of columns is selected to be 32 (4-bit $\times 32 = 128$-bit). Experiments are also carried out to determine the aspect ratio. Table 6 shows the area efficiency of the data path for different array scales (using Taiwan Semiconductor Manufacturing Company 65-nm technology). For the same number of columns, when the aspect ratio is 1, the area efficiency of some algorithms is much lower than that when the aspect ratio is 2 because of the reduced pipeline efficiency. The aspect ratio is therefore selected to be 2, which means that the scale of the array is 16 × 32. The results in Table 6 also show that the area efficiency in the 32 column case is higher than that in the 16 column case. This further confirms that 32 columns is a wise choice.

As shown in Table 3, a single REPROC core with a 16 × 32 array scale may not satisfy the high throughput requirements mentioned above. In this case, multiple cores can be used together. Figure 7 shows the framework for the multi-core mode. The host processor controls the entire system. The on-chip memory stores the data and contexts required for fast access, while the slower off-chip memory stores larger amounts of data and can be accessed through the external memory interface (EMI). A direct memory access controller (DMAC) is also added to enhance the parallelism of the system. The REPROC

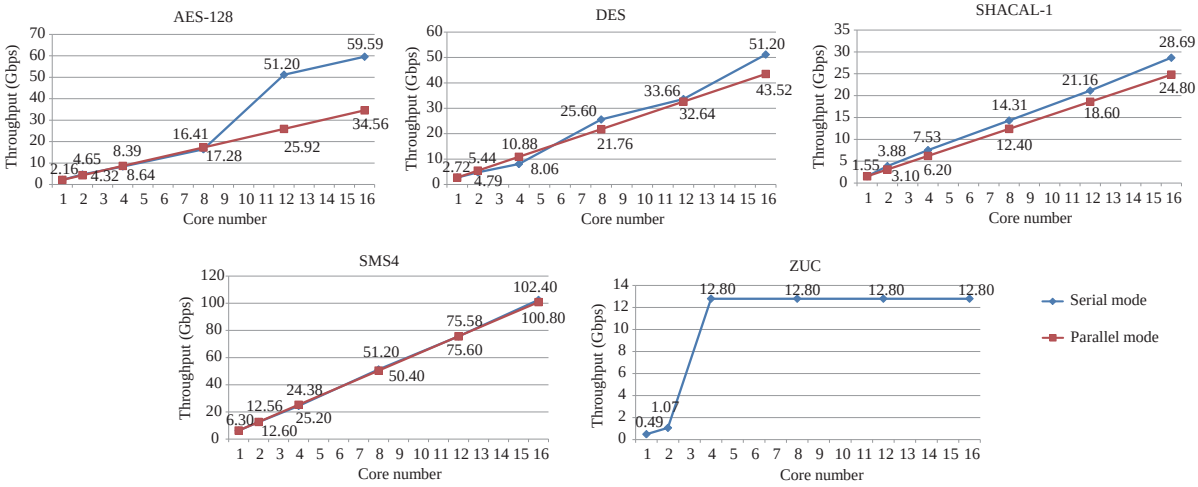

Figure 8 Throughput in multi-core mode.

cores are the major computing components of this system. There are two alternative computing modes, depending on the connections of these cores, i.e., whether they are in parallel mode or in serial mode. In parallel mode, there is no data exchange among the cores. Each core receives plaintexts and sends ciphertexts independently through the system bus. In contrast, in serial mode, the inputs and outputs of each core are connected together to form a data dependency structure. The plaintexts are input to the first core and ciphertexts are output from the final core. In this case, a larger number of pipeline stages can be unrolled to these cores. The throughputs of the cryptographic algorithms vary for the two modes. The throughputs are affected by a combination of factors, including the algorithm characteristics and the topologies of the mapping graphs. It is thus necessary to compare the performances of an algorithm when using the two different modes to determine which mode is superior for the given number of cores.

Experiments are performed to analyze the throughput trends for the two modes for multiple algorithms. Figure 8 shows the results, assuming that the bandwidth is unlimited. In parallel mode, the throughput increases in proportion to the number of cores, which is consistent with the definition of this mode. However, this simple rule cannot be applied equally to the serial mode because of a number of factors. For example, on the one hand, the resources that were originally idle in the adjacent cores may be used together to unroll more rounds. In this case, the number of pipeline stages per REPROC core is larger than that in the parallel mode. This makes the throughput of the serial mode higher in general than that of the parallel mode. On the other hand, the calculation of a certain algorithm is sometimes accomplished by executing multiple subgraphs. The number of pipeline stages required in these subgraphs may be different. When more rounds are unrolled in the serial mode, this difference increases and finally leads to reduction of the pipeline efficiency. This sometimes makes the serial mode throughput lower than that of the parallel mode. Also, significant performance boosts can even be found in the serial mode for certain algorithms, such as AES and ZUC. This is because the algorithms are fully unrolled onto the hardware at those particular points, at the point at which the pipeline efficiency reaches a maximum value and the performance is no longer related to the critical path of the algorithm. In this case, the algorithm can accomplish encryption of a complete data group (such as a 128-bit block) in a single clock cycle. It should be noted, however, that in the stream cipher case, the serial mode is the only available mode and an upper performance limit exists in this mode because of the feedback feature (e.g., 32 bits per clock cycle for ZUC).

To meet the specific performance requirements of 20 Gbps and 10 Gbps for block ciphers and stream ciphers, respectively (the performance of stream ciphers is usually lower because of their feedback feature), a 12-core architecture is finally selected. Table 7 shows the implementation results of AES-128 (which is one of the most commonly used symmetric ciphers) on the different platforms. The three implementations are all performed using the same technology (65 nm) to ensure fair comparison. When compared with the other results, the energy efficiency and the area efficiency are fairer evaluation standards than the

Table 7 Results for AES-128 on different 65-nm platforms

		Virtex-5 FPGA (XC5VLX50T) [10]	Many-core GPP Array [30]	Purposed reconfigurable architecture
Frequency (MHz)		350	1210	400
Area (mm^2)		132[a)]	6.63	51.36
Throughput	Original	4.1 Gbps with 400 slices[b)]	1.019 Gbps	51.2 Gbps
	Converted	58.75 Gbps[c)]		
Power (Watt)		1.398[d)]	1.58	0.584
Converted area efficiency (Gbps/mm^2)		0.45	0.1537	0.99
Converted energy efficiency (Gbps/W)		42.02	0.64	87.6

a) The FPGA area is obtained by X-ray imaging of the die.

b) In Xilinx FPGAs, logic resources are grouped in slices. One slice in the FPGA contains a set of LUTs, flip-flops and multiplexers. To compare the area efficiency of the FPGA with other types of hardware, Gbps/slice is converted to the more general standard of Gbps/mm^2.

c) To XC5VLX50T FPGA with 7200 slices, using a slice utilization rate of 80% (higher than the average value), and the original throughput level can achieve the required converted throughput, i.e. converted throughput = 10.2 Mbps/slice [10] $\times$ 7200 slices $\times$ 80% resource utilization.

d) The FPGA power (without the I/O part) is obtained by Xpower Estimator [31] based on the information about the clock frequency and the logic resource usage.

throughput, because they require comprehensive consideration of the performance, area and power. When compared with an industrially successful reconfigurable processor, the Xilinx Virtex-5 FPGA [10], the reconfigurable architecture used in this work achieves a 2.1$\times$ energy efficiency improvement and a 2.2$\times$ increase in area efficiency. This advantage is inseparable from the multiple optimization methods that have been introduced based on the characteristics of the symmetric algorithms. When compared with state-of-the-art many-core GPP arrays [30], the reconfigurable architecture proposed here achieves a 6.4$\times$ area efficiency improvement because of the significantly improved performance. Additionally, because operations such as instruction fetch and decode must be carried out in each calculation, the GPP has much lower energy efficiency than the reconfigurable architecture. In fact, the energy efficiency of the architecture proposed in this work is as much as 137 times that reported in [30]. It should be noted that there may be more recent papers on FPGA implementations than [10], but to obtain the converted FPGA throughput to form the required comparison, we finally choose an implementation based on a 65-nm FPGA (to avoid technology conversion) without using any FPGA RAMs. The comparison results for the other algorithms are not given here because of a lack of detailed comparison data. Most of the corresponding implementations are based on FPGAs, and it is difficult to determine the exact areas and powers for the different FPGA implementations.

7 Conclusion

A reconfigurable cryptographic architecture is presented in this paper. A series of optimization methods, including ICTR and HCO, have been introduced into the design. The results show that the proposed architecture can be used to realize most mainstream symmetric ciphers. The performance, energy efficiency and area efficiency of the architecture have obvious advantages over the state-of-the-art architectures reported in the literature. There is still much to be explored with regard to the corresponding reconfigurable architecture design because of the diversity of the calculation forms and operational granularities of the symmetric ciphers. Research in this area will be highly valuable in the cryptography field, and would also play a significant role in the research into general reconfigurable architectures to support a variety of calculation forms and granularities.

Acknowledgements This work was supported by the project from State Grid Cooperation of China (Grant No. SGRIDGKJ[2013]548).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Stallings W. Network and Internetwork Security: Principles and Practice. Upper Saddle River: Prentice Hall, 2010
2 Hiertz G R, Denteneer D, Stibor L, et al. The IEEE 802.11 Universe. IEEE Commun Mag, 2010, 48: 62–70
3 LAN/MAN Standards Committee. IEEE Std 802.3-2008. 2008
4 O'Melia S, Elbirt A J. Enhancing the performance of symmetric-key cryptography via instruction set extensions. IEEE Trans Very Large Scale Integr Syst, 2010, 18: 1505–1518
5 Bossuet L, Grand M, Gaspar L, et al. Architectures of flexible symmetric key crypto engines: a survey from hardware coprocessor to multi-crypto-processor system on chip. ACM Comput Surv, 2013, 45: 41
6 Granado-Criado J M, Vega-Rodrguez M A, Snchez-Prez J M, et al. A new methodology to implement the AES algorithm using partial and dynamic reconfiguration. Integration, 2010, 43: 72–80
7 Taherkhani S, Ever E, Gemikonakli O. Implementation of non-pipelined and pipelined data encryption standard (DES) using Xilinx Virtex-6 FPGA technology. In: Proceedings of IEEE 10th International Conference on Computer and Information Technology (CIT), Bradford, 2010. 1257–1262
8 Wang L, Jing J W, Liu Z B, et al. Evaluating optimized implementations of stream cipher ZUC algorithm on FPGA. In: Proceedings of 13th International Conference on Information and Communications Security, Beijing, 2011. 202–215
9 Venugopal V, Shila D M. High throughput implementations of cryptography algorithms on GPU and FPGA. In: Proceedings of IEEE International Instrumentation and Measurement Technology Conference, Minneapolis, 2013. 723–727
10 Bulens P, Standaert F, Quisquater J, et al. Implementation of the AES-128 on Virtex-5 FPGAs. In: Proceedings of 1st International Conference on Cryptology in Africa, Casablanca, 2008. 16–26
11 Standaert F X, Piret G, Rouvroy G, et al. FPGA implementations of the ICEBERG block cipher. Integration, 2007, 40: 20–27
12 Yang H, Basutkar N, Xue P, et al. Software-defined DVT-T2 demodulator using scalable DSP processors. IEEE Trans Consum Electron, 2013, 59: 428–434
13 Garcia A, Berekovic M, Aa T V. Mapping of the AES cryptographic algorithm on a coarse-grain reconfigurable array processor. In: Proceedings of International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Leuven, 2008. 245–250
14 Rossi D, Mucci C, Campi F, et al. Application space exploration of a heterogeneous run-time configurable digital signal processor. IEEE Trans Very Large Scale Integr Syst, 2013, 21: 193–205
15 PACT, X. XPP-III processor overview. White Paper Version. 2006
16 Majzoub S, Diab H. MorphoSys reconfigurable hardware for cryptography: the twofish case. J Supercomput, 2012, 59: 22–41
17 Mucci C, Vanzolini L, Campi F, et al. Interactive presentation: implementation of AES/Rijndael on a dynamically reconfigurable architecture. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), EDA Consortium, 2007. 355–360
18 Elbirt A J, Paar C. An instruction-level distributed processor for symmetric-key cryptography. IEEE Trans Parall Distr Syst, 2005, 16: 468–480
19 Cong J, Xiao B J. MrFPGA: a novel FPGA architecture with memristor-based reconfiguration. In: Proceedings of IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), San Diego, 2011. 1–8
20 NIST. Advanced encryption standard (AES). 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
21 NIST-FIPS. Data Encryption Standard. Federal Information Processing Standards (FIPS) Publication. 1999. http://csrc.nist.gov/encryption/tkencryption.html
22 Handsehuh H, Naccache D S. SHACAL. In: Proceedings of 1st Open NESSIE Workshop, 2000. 13–14. http://www.oscca.gov.cn/UpFile/200621016423197990.pdf
23 OSCCA (Office of State Commercial Cryptography Administration, China). The SMS4 Block Cipher. 2006. http://www.oscca.gov.cn/UpFile/200621016423197990.pdf
24 ETSI/SAGE Specification. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification. Version 1.5. 2011
25 Todman T J, Constantinides G A, Wilton S J, et al. Reconfigurable Computing: architectures and design methods. IEE Proc-Comput Dig Tech, 2005, 152: 193–207
26 Xilinx. Virtex-5 FPGA User Guide. 2009
27 Gentry C, Halevi S, Smart N P. Fully homomorphic encryption with polylog overhead. In: Proceedings of 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, 2012. 465–482
28 Lambrechts A, Raghavan P, Jayapala M, et al. Interconnect exploration for energy versus performance tradeoffs for coarse grained reconfigurable architectures. IEEE Trans Very Large Scale Integr Syst, 2009, 17: 151–155
29 PACT. White Paper of Video Decoding on XPP-III. 2006
30 Liu B, Baas B M. Parallel AES encryption engines for many-core processor arrays. IEEE Trans Comput, 2013, 3: 536–547
31 Xilinx. XPower Estimator User Guide. 2012