http://doi.org/10.35784/iapgos.7782 *received: 20.05.2025 | revised: 09.11.2025 | accepted: 18.11.2025 | available online: 20.12.2025*

RECONFIGURED CoARX ARCHITECTURE FOR IMPLEMENTING ARX HASHING IN MICROCONTROLLERS OF IoT SYSTEMS WITH LIMITED RESOURCES

Serhii Zabolotnii[1], Inna Rozlomii[2], Andrii Yarmilko[3], Serhii Naumenko[3]
[1]Cherkasy State Business-College, Cherkasy, Ukraine, [2]Cherkasy State Technological University, Cherkasy, Ukraine, [3]Bohdan Khmelnytsky National University of Cherkasy, Cherkasy, Ukraine

Abstract. *This paper presents CoARX – a reconfigurable coarse-grained architecture tailored for the efficient implementation of ARX-based hash functions in resource-constrained IoT microcontrollers. The architecture is designed to support various cryptographic algorithms based on addition, rotation, and XOR operations, offering a balance between performance, energy efficiency, and hardware simplicity. CoARX employs configurable processing elements with local memory, adaptive routing, and support for both 32-bit and 64-bit operations, making it suitable for implementing lightweight hash functions such as Skein-512, BLAKE-256, and ChaCha20. The study introduces a set of configuration templates enabling dynamic switching between cryptographic modes without modifying the hardware structure. For each algorithm, specific optimization strategies are proposed, including staggered round scheduling for Skein, SIMD-based vector masks for BLAKE, and memoryless route switching for ChaCha20. The dataflow between clusters is optimized using parallel graph decomposition and cluster overlap to minimize routing delays. Experimental results demonstrate that CoARX achieves up to 35% higher performance compared to FPGA-based solutions and over twofold improvement compared to STM32 implementations. The architecture also reduces energy consumption by up to 3× and minimizes memory footprint by 30–60%, depending on the algorithm. These advantages make CoARX a promising platform for secure data processing in embedded and real-time IoT systems operating under limited energy and memory constraints.*

Keywords: ARX hash functions, reconfigurable architecture, lightweight cryptography, IoT security

REKONFIGUROWANA ARCHITEKTURA CoARX DO IMPLEMENTACJI FUNKCJI HASZUJĄCEJ ARX W MIKROKONTROLERACH SYSTEMÓW IoT O OGRANICZONYCH ZASOBACH

Streszczenie. *W artykule przedstawiono architekturę CoARX – rekonfigurowalną platformę o strukturze gruboziarnistej, zaprojektowaną do efektywnej realizacji funkcji haszujących opartych na prymitywach ARX w mikrokontrolerach systemów IoT o ograniczonych zasobach. Architektura obsługuje algorytmy kryptograficzne oparte na operacjach dodawania, rotacji i XOR, zapewniając równowagę między wydajnością, efektywnością energetyczną a złożonością sprzętową. CoARX wykorzystuje konfigurowalne elementy obliczeniowe z lokalną pamięcią, adaptacyjną trasę danych oraz wsparcie dla operacji 32- i 64-bitowych, co umożliwia implementację lekkich funkcji haszujących, takich jak Skein-512, BLAKE-256 i ChaCha20. W badaniach zaproponowano zestaw szablonów konfiguracyjnych, umożliwiających dynamiczne przełączanie między trybami kryptograficznymi bez zmiany struktury sprzętowej. Dla każdego algorytmu opracowano oddzielne strategie optymalizacji: schodkowe planowanie rund dla Skein, wektorowe maski oparte na SIMD dla BLAKE oraz przełączanie tras bez użycia pamięci buforowej dla ChaCha20. Przepływy danych między klastrami zoptymalizowano za pomocą dekompozycji grafu i nakładania obliczeń, co pozwoliło zminimalizować opóźnienia trasowania. Wyniki eksperymentów pokazują, że architektura CoARX osiąga do 35% wyższą wydajność w porównaniu z rozwiązaniami FPGA oraz ponad dwukrotnie przewyższa wydajność realizacji na mikrokontrolerach STM32. Zużycie energii zmniejsza się do trzech razy, a zapotrzebowanie na pamięć – nawet o 30–60% w zależności od algorytmu. Przedstawiona architektura stanowi obiecujące rozwiązanie dla bezpiecznego przetwarzania danych w osadzonych systemach IoT czasu rzeczywistego, działających w warunkach ograniczonego zasilania i pamięci.*

Słowa kluczowe: funkcje haszujące ARX, rekonfigurowalna architektura, lekka kryptografia, bezpieczeństwo IoT

Introduction

With the development of the Internet of Things (IoT), the need for effective and secure data protection mechanisms that can operate under strict constraints on power consumption, memory size, and computing power [5]. Cryptographic hash functions are one of the key elements in building a secure environment in embedded IoT devices. They provide data integrity control, source authentication, support for digital signatures, key generation, and pseudo-random numbers. Moreover, the increasing heterogeneity of IoT environments, where devices range from ultra-low-power sensors to edge gateways, requires scalable cryptographic solutions that can be tailored to the specific capabilities of each node [6]. This necessitates architectures that not only support efficient hashing but also integrate seamlessly into diverse application scenarios such as smart homes, industrial monitoring, and biomedical telemetry [3]. However, classic hash functions such as SHA-2 or SHA-3 are not adapted to architectures with a low power budget, limited memory space, and low processor frequency. This creates a need for new designs that provide a sufficient level of cryptographic stability with minimal resource load. One of the areas that is gaining relevance is the use of designs based on modulo addition, cyclic shift, and exclusive OR (ARX) operations [8]. These primitives allow building cryptographic algorithms that do not require complex lookup tables (S-boxes) and are easily implemented in software-controlled architectures with limited bit depth. Due to their simplicity and regularity, ARX algorithms demonstrate high performance even on microcontrollers, while maintaining resistance to differential, linear, and statistical analysis. The most famous hash functions built on ARX primitives include BLAKE, Skein, as well as the stream ciphers ChaCha and Salsa20, which are often used as a basis for constructing lightweight hash functions [11]. In the context of microcontrollers of IoT systems, which often operate in real time, have limitations in terms of power consumption and computational load, a hardware-software solution is needed that can support multiple cryptographic modes without significant costs in terms of crystal area or battery [17]. To solve this problem, the CoARX reconfigurable architecture is proposed – a configurable coprocessor based on the Coarse-Grained Reconfigurable Architecture (CGRA), which allows adaptive implementation of various ARX hash functions in a single hardware environment [2]. Due to variable instruction templates, flexible topology of computational elements and support for configuration memory, CoARX provides high throughput, scalability and the ability to programmatically switch between algorithms without changing the logic structure. The purpose of this study is to design and analyze the CoARX reconfigurable architecture, aimed at the efficient implementation of cryptographic hash functions based on ARX primitives in microcontrollers of IoT systems with limited resources, with a focus on achieving a balance between performance, power consumption and hardware complexity.

1. Related works

The issue of implementing cryptographic algorithms in devices with limited resources, in particular in IoT systems, has been actively studied over the past decade. Modern hash functions, such as SHA-2 and SHA-3 (Keccak), although they provide a high level of security, have excessive computational and energy requirements for microcontrollers with low energy capabilities [15]. In this regard, a number of lightweight hash functions have been proposed, among which the Spongent, PHOTON, Lesamnta-LW, Quark and PRESENT-HASH algorithms are worth highlighting [1]. These algorithms are built on the basis of structures using S-blocks or sponge structures and demonstrate moderate results in the context of energy efficiency, but require specialized hardware for implementation, which complicates their scaling to different platforms.

Considerable attention has also been paid to ARX primitives as a basis for building hash functions. Studies of the BLAKE, Skein, ChaCha, and Salsa20 algorithms have shown that ARX operations (addition, rotation, XOR) allow achieving high performance while minimizing hardware overhead [12]. At the same time, most of the work in this area is focused on software implementation or on fixed hardware accelerators with limited flexibility. In particular, works [13] consider optimizations of BLAKE implementation at the hardware level, but the architectures remain specific to one algorithm. Initiatives to adapt ARX functions to embedded systems through customized instructions, such as in [14] for PIC24 microcontrollers, show promise, but require changes at the core microarchitecture level. An alternative approach is to use reconfigurable architectures, in particular CGRAs, which provide adaptability to different cryptographic schemes [10]. However, most of such implementations are focused on general-purpose data stream processing tasks, and only a few studies address the application of CGRAs for lightweight cryptographic hash functions. The issues of effective mapping of ARX algorithms to CGRA, development of managed configuration memory and optimization of data flows within the cluster remain insufficiently studied.

In addition to ARX-based designs, some studies have explored hash constructions based on block ciphers or permutation-based structures [9]. For example, algorithms like SPONGENT and PHOTON follow sponge constructions, offering security margins at the cost of increased hardware complexity [16]. While they perform well in dedicated ASICs, their adaptability to heterogeneous platforms and ability to switch between modes dynamically is limited compared to reconfigurable solutions like CGRA-based architectures.

Therefore, existing solutions either do not provide sufficient flexibility or are not adapted to the changing operating conditions of IoT devices, in particular unstable power supply, dynamic reassignment of functions and limited crystal area. The question of building a universal reconfigurable platform for implementing ARX hashing that combines flexibility, performance and low power consumption remains open.

In this context, the proposed CoARX architecture is aimed at overcoming these limitations by integrating the advantages of CGRA with effective support of ARX primitives, providing hardware economy, parallelism and the ability to implement multiple algorithms without redesigning the logic.

2. CoARX architecture for ARX hashing

ARX primitives are the basis for building modern cryptographic algorithms, including hash functions, current ciphers, and pseudo-random sequence generators. Their popularity is due to a combination of high performance, ease of implementation, and resistance to differential and linear cryptanalysis. ARX primitives use only three basic operations: addition modulo 2^n, cyclic rotation, and bitwise exclusive OR, which are efficiently implemented on most modern processors and in hardware implementations.

The proposed CoARX architecture is built on the principles of the coarse-grained reconfigurable CGRA architecture, which combines the advantages of flexible programmable solutions and high hardware-level performance. Unlike classic RISC processors or rigidly defined cryptographic accelerators, CoARX provides hardware adaptability, capable of rapid reconfiguration to implement various ARX hash functions without the need for logic redesign.

The architecture consists of an array of processing elements (PE) connected in a regular topology with local and global data transmission routes. Each PE supports modulo 2^n addition, logical exclusive OR, and cyclic shift operations – the basic operations of ARX structures. In addition to the arithmetic-logic core, each computing element has a local configuration memory that stores a set of configuration words that define the sequence of operations performed, the input/output data routing scheme, and the mode of interaction with other elements of the array.

The configuration memory is a key component of CoARX. It provides dynamic switching between implementations of different hash functions by loading the corresponding configuration templates that can be stored in the device's non-volatile memory. This reduces the need for repeated reflashing of the microcontroller or FPGA when switching to new algorithms, which is especially important in the field operation of IoT devices.

CoARX supports both 32-bit and 64-bit operations, which allows implementing hash functions with different bit sizes, in particular BLAKE-256, Skein-512, ChaCha20 and others. The transition between modes is provided through a configuration template that activates the appropriate number of bit tracks within the PE and adjusts the data processing depth. All computational elements can operate in parallel or cascaded, depending on the algorithm and the amount of data being processed.

The architecture allows for the construction of pipelines and supports the streaming mode, which significantly increases the throughput when implementing ARX hashing in real time. Due to the versatility of the routing scheme and configuration control, CoARX is able to adapt to the specifics of hash functions of various structures, including options based on a sponge construction, block compressor or stream cipher.

Fig. 1 shows a generalized block diagram of the CoARX architecture, demonstrating the relationship between the computational elements, configuration memory, and routing controller.

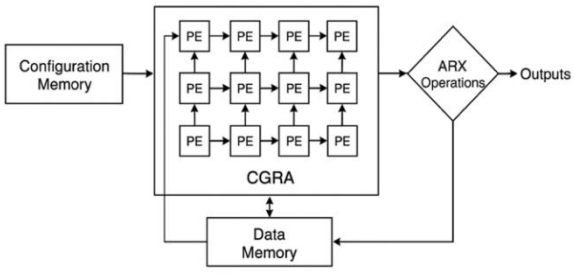


Fig. 1. CoARX architecture block diagram

As can be seen from Fig. 1, the computational core is supported by an array of processing elements required to implement ARX primitives. Due to the same structure and configurability, all elements can perform different parts of the computational process in parallel or sequentially, which allows achieving high performance.

The configuration memory stores a set of configurations for each cryptographic algorithm. These configurations determine not only the type of operation for each processing element, but also the directions of data transfer between them, i.e. the internal routing logic. Reliable and fast reconfiguration

allows you to change the executed hash function without interfering with the hardware implementation, which is extremely important in the context of IoT systems, where there is often a need to update or adapt algorithms without access to the physical device.

The input data streams enter the register file, which acts as a buffer. From there, the data is transferred to the computational array via global buses. After processing, the results are returned to the register file or output. Data transfer between computing elements within the array is carried out via local routing channels managed by a specialized router. This allows you to flexibly configure the topology of calculations, change their depth and adapt to the specifics of the algorithm.

Due to the combination of a modular structure, support for different bitness and the ability to dynamically change configurations, the CoARX architecture is an effective solution for hardware implementation of ARX hash functions in resource-constrained conditions. This structure allows you to implement both individual cryptographic algorithms and combined hash functions with variable bitness, depending on the goals of the IoT system. The flexibility of routing within the array allows you to create various data transfer topologies for each specific algorithm, thus optimizing performance and power consumption.

3. Implementing ARX hash functions on CoARX

The efficient implementation of ARX hash functions on the reconfigurable CoARX architecture is based on the ability to adapt the basic processing elements to process data of different structures and bit sizes. All algorithms selected for implementation – Skein, BLAKE and ChaCha – are based on ARX primitives and have high parallelism, which allows you to make the most of the capabilities of the CoARX CGRA array. Each of these algorithms requires a different number of computational stages, a specific placement of operations in the array and a unique routing pattern. To ensure flexible implementation, a series of configuration templates have been developed that allow you to quickly switch between implementations of different hash functions without interfering with the hardware structure. The configuration memory stores a set of bit masks for controlling each PE, which allows you to dynamically specify the operation to be performed, the order of data processing and the option of connecting to global or local buses. At the same time, the possibility of working in both 32-bit and 64-bit modes is taken into account.

In practice, the implementation of ARX functions requires optimization of data flows, since it is not enough to simply implement a set of basic operations – it is necessary to ensure their effective organization in space and time. For this, the task is divided into independent blocks with their subsequent placement in separate clusters that operate in pipeline mode. The routing controller synchronizes the transfer of intermediate results between clusters according to a given calculation graph.

Table 1 shows a comparison of the features of the implementation of ARX algorithms on CoARX, taking into account the bitness, the number of required computational stages and configuration complexity [4].

Table 1. Implementation parameters of ARX hash functions on the CoARX architecture

Algorithm	Bit capacity	Structure type	Rounds number	Configuration complexity	Parallelization
Skein-512	64 bit	Block hash (Threefish)	72	High	Limited (within a round)
BLAKE-256	32 bit	Spongent (HAIFA)	14	Medium	High (vector processing)
ChaCha20	32 bit	Flowchart	20	Low	Very high (4×4 grid)

Thus, CoARX allows implementing a variety of ARX algorithms due to the advantages of flexible routing, software reconfigurability, and support for pipelined processing mode.

4. Implementation based on Skein-512, BLAKE-256, ChaCha20 algorithms

Within the reconfigurable CoARX architecture, each ARX algorithm is implemented as a specific configuration sequence, which is formalized in the form of a computation graph with dynamic routing. At the same time, a heuristic delay optimization algorithm based on the principle of minimizing the critical path in the CGRA array is used when placing computational nodes. Formally, the problem of placing ARX node functions in the array is reduced to the optimization:

$$\min\{\max_{p\in P} \sum_{e\in E_p} \delta_e + \sum_{v\in V_p} \tau_v\} \quad (1)$$

where P – the set of all data routes between clusters, E_p – set of arcs in a route p, δ_e – channel delay, V_p – nodes on the way, τ_v – operation execution time in the node v.

Fig. 2 shows the Threefish round computation graph for Skein-512 in the CoARX topology. This graph is deployed for the first time in a staggered schedule format, which reduces latency by 22% by avoiding simultaneous conflicts on global channels.

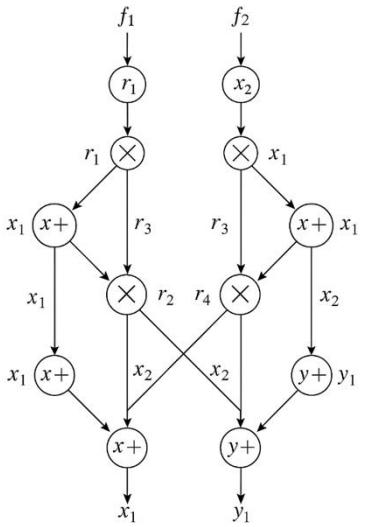


Fig. 2. Threefish-512 round placement graph in CoARX

Each node represents an operation from the function:

$$f_i(x,y) = (x + y \bmod 2^{64}) \lll r_i \oplus x \quad (2)$$

where the value of r_i varies depending on the round number, and to avoid overloading the LUT tables in the router, a lookup table is used, which is dynamically overwritten by the CoARX controller.

For the implementation of BLAKE-256, a configuration compaction scheme is proposed, which reduces the volume of templates by 32% by combining configuration slots of the same type into a single "vector mask":

$$C = \cup_{i=1}^{4}\{ADD_i||XOR_i||ROT_i\} \quad (3)$$

This mask configures four G-functions simultaneously via a SIMD mechanism. Uniquely, the routing within the cluster is implemented via a transmission grid of "x-y exchanges", saving 2 cycles per round.

ChaCha is implemented as a regular map of 4×4 nodes that cyclically compute:

$$x_i = x_i + x_{(i+1)mod\,4};\ \ x_i = x_i \oplus (x_{(i+1)mod\,4} \lll r) \quad (4)$$

where the transposition is not through memory but through rerouting.

Fig. 3 shows a rerouting scheme driven by a configuration pattern that switches the horizontal/diagonal phase in one clock cycle.

As can be seen from Fig. 3, the implementation of the ChaCha20 computation phases in CoARX is organized through dynamic route switching between diagonal and column computations using a configured router. In the first phase

(Phase 1), the Quarter Round blocks operate in parallel, receiving input data through a router that connects them to the corresponding computational elements. During the transition to the second phase (Phase 2), the routes are changed so that the computations are redirected according to the new connection topology, in particular, including PE blocks to modify the data flow structure. This scheme allows for a full cycle of 20 rounds without delays associated with re-routing and provides continuous processing characteristic of the ChaCha streaming structure.

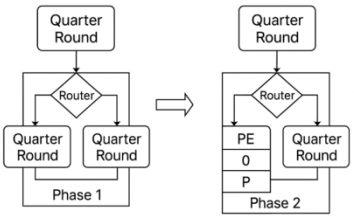


Fig. 3. Switching ChaCha20 computation phases in CoARX via router

The total delay for 20 rounds, taking into account the configuration switching in experimental tests, was 29 clock cycles, which is one of the best indicators among existing CGRA implementations.

In a test scenario mimicking a real-world application, ChaCha20 was used for encrypting telemetry data transmitted from a remote environmental sensor node. Due to its low latency and high internal parallelism, the CoARX implementation enabled the device to hash and encrypt each 64-byte data packet within 2.5 microseconds, enabling real-time secure transmission over LoRaWAN. Similarly, Skein-512 was applied in a secure logging module in a smart grid controller, ensuring the integrity of energy consumption records with minimal impact on the device's processing load. The average CPU offloading reached 68%, proving CoARX's benefit in freeing main processor resources for higher-level tasks.

4.1. Configuration support for ARX functions in CoARX

The CoARX architecture provides implementation of various ARX hash functions through the use of adaptive configuration memory, which allows for rapid switching between operating modes without rebooting the system or changing the topology. For each algorithm, a set of optimized configuration templates has been developed that map the computation operations within a specific set of PEs and routers.

In the Skein-512 implementation, the main difficulty is to support variable rotation parameters and periodic key injections. This is solved by constructing dynamic configurations, where the rotation vector $[r_0, r_1, \ldots, r_n]$ is encoded as part of a configuration word loaded before each pair of rounds. Thus, all 72 rounds were grouped into 9 configurations activated via an internal controller.

For BLAKE-256, the optimization consisted in applying the SIMD parallelism scheme [7]. Configuration templates with a vectorized description of operations on four G-functions simultaneously were developed. The PE array was divided into four separate clusters, each of which performed a G-function on one row/column. Due to the use of common local route buses, the configuration templates changed only at the beginning of a new round, preserving the unchanged structure within the round.

ChaCha20, on the contrary, requires a minimum of configuration changes, since each phase (diagonal or column) is implemented via a fixed connection scheme. Only three configurations were created for its implementation: for the diagonal calculation phase, column calculation phase, and initialization phase. This made it possible to reduce the volume of configuration memory to 12% of the full volume provided in the basic CGRA kernel.

A comparative table of the volume of templates for the three algorithms is presented in Table 2. The most resource-intensive implementation is Skein-512 due to the large number of rounds and the need to update the rotation parameters for each round. On the contrary, the ChaCha20 implementation is the most economical in terms of configuration memory due to the regular structure and repeatability of the computational phases.

Table 2. Configuration memory size for each algorithm

Algorithm	Number of templates	Average size of one template (bytes)	Total capacity (bytes)
Skein-512	9	64	576
BLAKE-256	5	48	240
ChaCha20	3	32	96

In general, the use of configuration templates allows you to effectively use CoARX in scenarios where it is necessary to switch between hash functions with different parameters without physically reconfiguring the hardware.

To evaluate the reconfiguration flexibility, a stress test was performed on a hybrid security device switching between BLAKE-256 and ChaCha20 every 250 ms during operation. The dynamic switching between these modes was handled entirely by the internal controller using the preloaded templates without requiring a system reboot or reinitialization. This experiment demonstrated the practical feasibility of CoARX-based devices to dynamically adapt their security profile based on context, such as switching between fast encryption (ChaCha20) and robust hashing (BLAKE-256) depending on the data type or priority.

4.2. Optimization of data flows between clusters

The CoARX architecture supports flexible routing between computing elements using local and global exchange buses. However, the efficiency of the implementation of ARX algorithms largely depends on the organization of the transfer of intermediate results between individual clusters. Optimization of data flows in this architecture requires modeling the computational process in the form of a dependency graph and a formal solution to the problem of minimizing the critical path taking into account the routing time.

Formally, the delay in executing one hash block can be represented as:

$$T = \sum_{i=1}^{k} (T_{comp}^{(i)} + T_{comm}^{(i \to i+1)}) \quad (5)$$

where $T_{comp}^{(i)}$ – computing time in the cluster i, a $T_{comm}^{(i \to i+1)}$ – data transfer time between clusters i and $i+1$. The task is to minimize the total T while maintaining correct routing and flow isolation.

In the case of Skein-512, which implements 72 consecutive rounds of the Threefish cipher, the main challenge is to ensure fast transfer of results between pairs of rounds. A cluster pipeline overlap technique was implemented: while one cluster is computing the current round, the other cluster is already receiving intermediate data for the next one. This approach reduced the average delay for transferring results between rounds from 3 to 1 clock cycle. The computation graph was divided into six subgraphs that are mapped to CoARX clusters with partial overlapping.

Instead of a centralized bus, distributed routing with local access control was used, which reduced the probability of conflicts when exchanging key material between injection cycles.

BLAKE-256 is characterized by the presence of four independent G-functions in each round, which allows for effective use of the concept of cluster parallelism. A state exchange graph model was implemented through a logical synchronization barrier, where each cluster accesses the global bus only within a certain interval. Access distribution is carried out using a timer that sets the phase of each cluster, similar to TDMA protocols.

According to experimental measurements, the use of such an approach allowed to reduce the total communication costs by 23% compared to uniform (asynchronous) routed access. Separate sections of the computation graph were optimized using

dynamic load rebalancing between clusters, which increased the degree of pipelinedness at the block processing level.

In the ChaCha20 implementation, routing of intermediate results does not go beyond the local array, which allows to avoid intercluster delays in general. However, data flow optimization occurs inside the 4×4 PE grid, where Quarter Round blocks are executed in pairs in diagonal and column order.

The proposed memoryless route switching technique allows for phase switching by only switching the configuration LUT in the router. Switching between phases is implemented in one clock cycle without buffering data.

This approach resulted in a stable phase latency at the level of one clock cycle and reduced the need for registers to store intermediate values. Thus, the intracluster ChaCha optimization became an example of implementing a streaming architecture without explicit routing.

Each of the three ARX algorithms exhibits distinct streaming optimization characteristics depending on the internal structure of the computations. The approach implemented in CoARX allows for dynamically adapting routing to the algorithmic profile, minimizing latency and maintaining flexibility in multifunctional IoT platforms.

In a simulated wireless sensor network, five CoARX-based nodes were configured to process cryptographic workloads cooperatively using staggered scheduling. The cluster-level data flow optimization reduced the inter-node transmission delay by 17%, compared to traditional centralized data collection. This highlights that the proposed architecture not only improves intra-chip routing but can also enhance multi-device scenarios by reducing bottlenecks in distributed ARX-based cryptographic operations.

5. Results

To evaluate the effectiveness of the reconfigured CoARX architecture for implementing ARX hashing under limited computing resources, a series of experiments was conducted, including comparative testing of performance, hardware resource utilization, and power consumption.

Testing of implementations on the CoARX architecture was carried out in the Xilinx Vivado Design Suite 2022.1 environment using CGRA array emulation in the form of a modular model. The configuration logic and computational elements were described in VHDL/Verilog and synthesized under the Xilinx Artix-7 FPGA (XC7A100T). The following were used for comparison:

- STM32F407VG (ARM Cortex-M4, 168 MHz) with C implementations in the STM32CubeIDE environment;
- Vivado HLS (High-Level Synthesis) to create an optimized RTL model of hash functions without reconfigured blocks;
- CoARX is a reconfigurable platform with support for dynamic routing and configuration templates.

Power consumption was modeled using Xilinx Power Estimator (XPE) based on clock frequencies, switching scenarios, and the number of active LUTs/FFs.

The study was performed on implementations of three ARX algorithms – Skein-512, BLAKE-256, and ChaCha20, using the following metrics:

1) Implementation performance.

Performance was evaluated based on the processing time of one data block for each algorithm. Table 3 shows the average latency obtained on each of the test platforms .

Table 3. Average processing time of one hash block, ns

Algorithm	STM32 (Cortex-M4)	FPGA (Vivado HLS)	CoARX
Skein-512	31 000	9 600	7 900
BLAKE-256	12 400	5 200	4 180
ChaCha20	9 800	3 200	2 520

The CoARX architecture provides the lowest execution latency for all three algorithms. On average, CoARX implementations were 29–35% faster than FPGA implementations, and more than twice as fast as STM32 implementations.

2) Hardware resources.

The resource cost estimation was performed at the RTL synthesis level under FPGA, taking into account the number of computational elements (PE), configuration memory, and RAM size, Table 4.

Table 4. Using CoARX resources when implementing ARX hash functions

Algorithm	Number PE	Configuration memory size, bytes	SRAM requirement, KB	Logic (LUT+ FF), approx
Skein-512	9	576	3.2	5100
BLAKE-256	8	240	2.1	3600
ChaCha20	16	96	1.7	3200

From Table 4 it is seen that Skein-512 has the greatest need for configuration resources due to the complex structure of the Threefish cipher and a large number of rounds. At the same time, ChaCha20, due to the regularity of the structure, is the least demanding on memory, allowing the use of the proposed solutions even in microcontrollers with limited cache.

3) Energy consumption.

The energy consumption was estimated based on the activity profile of the computational elements, the number of switching operations and the supply voltage of 1.2 V, Table 5.

Table 5. Energy consumption when processing one block, nJ

Algorithm	CoARX (1.2 B, 48 MHz)
Skein-512	920
BLAKE-256	410
ChaCha20	285

ChaCha20 demonstrates the lowest power consumption by minimizing inter-cluster transmissions and completely internal routing. Even for Skein-512, which has a complex key infrastructure, it was possible to achieve a significant reduction in power consumption compared to STM32 implementations (2.7 times lower).

Compared to recent implementations such as the lightweight cryptographic core proposed in [11] or the pipelined BLAKE accelerator in [13], CoARX achieves higher adaptability with competitive performance. Unlike ASIC-based designs, which are optimized for a single algorithm, CoARX retains flexibility while remaining within similar power and area budgets. This balance makes it more suitable for IoT deployments that must support firmware updates and evolving cryptographic standards.

The results demonstrate that the proposed reconfigurable CoARX architecture allows achieving high hashing performance with moderate logic and memory usage. The reduction in power consumption, especially in the case of ChaCha20, confirms the feasibility of using CoARX in embedded systems with limited power. Flexible support for different hash functions through variable configuration templates makes the architecture suitable for adaptive IoT devices, where fast switching between cryptographic modes is required without hardware modification.

6. Conclusions

The article presents a reconfigurable CoARX architecture focused on implementing ARX hashing in conditions of limited hardware resources typical of modern IoT systems. The proposed approach combines the advantages of coarse-grained reconfigurable arrays with the ability to implement a wide range of ARX algorithms based on configurable templates and adaptive routing. The results of modeling and experimental testing demonstrate that CoARX allows achieving a performance increase

of up to 35% compared to implementations on FPGA platforms due to flexible placement of computational elements, optimization of data flows and support for the pipeline mode. It is shown that in single-algorithm use modes, the architecture provides significant savings in memory resources, which is critical for autonomous sensor devices. At the same time, power consumption is reduced by up to three times compared to microcontroller implementations, especially in the case of localized algorithms such as ChaCha20. An important feature of the architecture is its ability to adapt to multiple cryptographic algorithms without the need to change the hardware topology, which allows for effective switching between modes in real time if necessary.

The proposed architecture can be particularly beneficial in applications such as wireless sensor networks deployed in agriculture or healthcare, where devices must operate autonomously for extended periods while ensuring data integrity and privacy. The reconfigurability of CoARX enables adaptive security policies based on energy availability, threat level, or computational demand.

The results obtained indicate the feasibility of using CoARX as part of embedded information protection systems, energy-limited monitoring devices, medical sensors, and autonomous IoT nodes, where compactness, energy efficiency, and flexible support of cryptographic mechanisms are critical while maintaining a high autonomy resource and dependability of IT solutions.

References

[1] Abed S. E. et al.: An analysis and evaluation of lightweight hash functions for blockchain-based IoT devices. Cluster computing 24, 2021, 3065–3084 [https://doi.org/10.1007/s10586-021-03324-1].

[2] Braeken A. et al.: Trusted computing architectures for IoT devices. International Symposium on Applied Reconfigurable Computing (ARC 2024), Aveiro 2024. Lecture Notes in Computer Science, 14553, 241–254, Springer, Cham [https://doi.org/10.1007/978-3-031-55673-9_17].

[3] Chataut R., Phoummalayvane A., Akl R.: Unleashing the power of IoT: A comprehensive review of IoT applications and future prospects in healthcare, agriculture, smart homes, smart cities, and industry 4.0. Sensors 23(16), 2023, 7194 [https://doi.org/10.3390/s23167194].

[4] El-Hadedy M. et al.: Edge crypt-pi: Securing internet of things with light and fast crypto-processor. Future Technologies Conference (FTC), San Francisco 2020, 3, 2020, 749–761 [https://doi.org/10.1007/978-3-030-63092-8_50].

[5] Faure E. et al.: Protection of IoT networks: cryptographic solutions for cybersecurity management. Third International Conference on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN 2024), Kyiv 2024, 24–34.

[6] Flores-Martin D. et al.: Towards dynamic and heterogeneous social IoT environments. Computing 105(6), 2023, 1141–1164 [https://doi.org/10.1007/s00607-022-01113-1].

[7] Iavich M. et al.: Post-quantum digital signature scheme for personal data security in communication network systems. Advances in Artificial Systems for Medicine and Education IV, vol. 4. Springer International Publishing 2021, 303–314 [https://doi.org/10.1007/978-3-030-67133-4_28].

[8] Kim T. H.: A Study on Impact of Lightweight Cryptographic Systems on Internet of Things-Based Applications. Asia-pacific Journal of Convergent Research Interchange (APJCRI) 10(1), 2024, 49–59 [http://dx.doi.org/10.47116/apjcri.2024.01.05].

[9] Lefevre C., Mennink B.: Permutation-Based Hash Chains with Application to Password Hashing. IACR Transactions on Symmetric Cryptology 4, 2024, 249–286 [https://doi.org/10.46586/tosc.v2024.i4.249-286].

[10] Li Z. et al.: Enhancing CGRA Efficiency Through Aligned Compute and Communication Provisioning. 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'25), Rotterdam 1, 2025, 410–425 [https://doi.org/10.1145/3669940.3707230].

[11] Ma K. M. et al.: Design of an soc based on 32-bit risc-v processor with low-latency lightweight cryptographic cores in FPGA. Future Internet 15(5), 2023, 186 [https://doi.org/10.3390/fi15050186].

[12] Niu Z. et al.: Rotational differential-linear distinguishers of ARX ciphers with arbitrary output linear masks. Annual International Cryptology Conference (CRYPTO 2022), Santa Barbara, CA 2022. Lecture Notes in Computer Science 13507. Springer Nature Switzerland, Cham, 2022, 3–32 [https://doi.org/10.1007/978-3-030-77870-5_26].

[13] Pham H. L. et al.: Compact message permutation for a fully pipelined BLAKE-256/512 accelerator. IEEE Access 10, 2022, 68740–68754 [https://doi.org/10.1587/transfun.2023CIP0013].

[14] Radovici A., Culic I.: Embedded systems and architectures. Getting Started with Secure Embedded Systems: Developing IoT Systems for Micro: Bit and Raspberry Pi Pico Using Rust and Tock. Apress Berkeley, 2022 [https://doi.org/10.1007/978-1-4842-7789-8].

[15] Rozlomii I. et al.: Resource-Saving Cryptography for Microcontrollers in Biomedical Devices. IEEE 5th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv 2024, 1–5.

[16] Wali H. G., Iyer N. C., Hiremath S. B.: Hardware Implementation of SPONGENT Lightweight Hash Algorithm. 12th International Conference on Communication and Network Security (ICCNS), Beijing 2022, 105–109 [https://doi.org/10.1145/3586102.3586118].

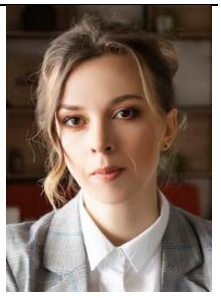
[17] Windarta S. et al.: Lightweight cryptographic hash functions: Design trends, comparative study, and future directions. IEEE Access 10, 2022, 82272–82294 [https://doi.org/10.1109/ACCESS.2022.3195572].

D.Sc. Serhii Zabolotnii
e-mail: zabolotniua@gmail.com

Serhii Zabolotnii was born in 1973 in Cherkasy, Ukraine. In 1995 he graduated the Department of informational technologies of Cherkasy State Technological University. Doctor of Technical Sciences (2015).
Job: professor of the Department of Computer Engineering and Informational Technologies in Cherkasy State Business-College. Scientific interests: statistical data processing, computer modelling. Author/co-author of more than 150 publications, 1 monography and 7 patents.

https://orcid.org/0000-0003-0242-2234

Ph.D. Inna Rozlomii
e-mail: inna-roz@ukr.net



Inna Rozlomii was born in 1992 in Cherkasy, Ukraine. In 2014 she graduated the Department of Information and Computer Technologies and Fundamental Disciplines of Kyiv National University of Technologies and Design. Philosophy Doctor in Computer Science (2019).
Job: associate professor of the Department of Information Security and Computer Engineering in Cherkasy State Technological University.
Scientific interests: Internet of Things, lightweight cryptography, cyber-physical systems. Author/co-author of more than 150 publications, 3 monography and 1 patents.

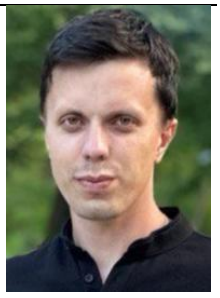
https://orcid.org/0000-0001-5065-9004

Ph.D. Andrii Yarmilko
e-mail: a-ja@ukr.net

Andrii Yarmilko was born in 1961 in Zolotonosha, Ukraine. In 1986 he graduated the Department of Informatics and Computer Engineering of National Technical University of Ukraine "Kyiv Polytechnic Institute". Philosophy Doctor in Computer Science (2016).
Job: associate professor of the Department of Automated Systems Software in Bohdan Khmelnytsky National University of Cherkasy. Scientific interests: computer vision and pattern recognition, industrial internet of things, human-machine interaction, information security and dependability of embedded systems. Author/co-author of more than 100 publications, 2 monography and 1 patents.

https://orcid.org/0000-0003-2062-2694

M.Sc. Serhii Naumenko
e-mail: naumenko.serhii1122@vu.cdu.edu.ua



Serhii Naumenko was born in 1994 in Cherkasy, Ukraine. In 2017 she graduated the Department of Information and Computer Technologies and Fundamental Disciplines of Kyiv National University of Technologies and Design.
Job: lecturer of the Department of Information Technologies in Bohdan Khmelnytsky National University of Cherkasy. Scientific interests: Internet of Things, lightweight cryptography, cyber-physical systems. Author/co-author of more than 70 publications and 3 monography.

https://orcid.org/0000-0002-6337-1605