BULLETIN of Cherkasy State Technological University

Journal homepage: https://bulletin-chstu.com.ua/en Vol. 30 No. 2. 2025

UDC 004.31: 004.056
DOI: 10.62660/bcstu/2.2025.10

Hardware implementation of the HDG hash function

Volodymyr Luzhetskyi*
Doctor of Technical Sciences, Professor
Vinnytsia National Technical University
21021, 95 Khmelnytske Shose Str., Vinnytsia, Ukraine
https://orcid.org/0000-0001-7466-7738

Vitalii Seleznov
Assistant
Vinnytsia National Technical University
21021, 95 Khmelnytske Shose Str., Vinnytsia, Ukraine
https://orcid.org/0009-0004-0225-9697

Abstract. Given the increasing role of the Internet of Things and related low-resource devices, research into hash functions that provide a high level of cryptographic strength with minimal hardware costs is relevant. The aim of the study was to propose a hardware implementation of a new HDG hash function designed for use in small devices in the form of a specialised processor in order to reduce the hardware costs of implementation. The research methods included structural design of each functional block, digital modelling in the Logisim-evolution environment, and synthesis on an ASIC platform using 0.18 μm technology with the standard UMCL18G212T3 library, as well as calculation of hardware complexity in Gate Equivalents. HDG meets the requirements of low-resource cryptography thanks to its byte-oriented architecture, which allows data processing at the level of individual bytes, ensuring high efficiency with limited memory resources and computing capabilities of devices. The structure of a specialised processor for hashing was presented. The HDG specialised processor is decomposed into four functional blocks, each of which implements a corresponding function: a register block for storing intermediate hash values; a shift register with linear feedback, which provides the generation of a pseudorandom sequence; a block for addition modulo 256 and a control block. The simulation results confirmed the correctness of the structure of the specialised processor and the interaction of its components. The calculated complexity of the HDG processor hardware implementation is 1,683 GE for calculating a 256-bit hash value, which meets the requirements of the international standard ISO/IEC FDIS 29192 for low-resource cryptography. A comparison with hardware implementations of well-known low-resource hash functions PHOTON, SPONGENT, S-Quark, GLUON, and HVH showed a reduction in hardware costs of 15% or more. In some cases, the HDG processor demonstrated lower implementation complexity for 256-bit hash values values of 256 bits compared to hash functions that provide calculations of hash values of 224 or 160 bits, which indicates the effectiveness of the developed structure and the feasibility of using such a specialised processor for devices with limited hardware resources

Keywords: hardware device; specialised processor; low-resource cryptography; hashing method; hardware complexity

Article's History: Received: 13.10.2024; Revised: 24.02.2025; Accepted: 16.06.2025.

Suggested Citation:
Luzhetskyi, V., & Seleznov, V. (2025). Hardware implementation of the HDG hash function. *Bulletin of Cherkasy State Technological University*, 30(2), 10-21. doi: 10.62660/bcstu/2.2025.10.

*Corresponding author

INTRODUCTION

The rapid development of the Internet of Things (IoT) and the use of various mobile devices imposes a set of requirements on cryptographic information protection measures, relating to the level of cryptographic resistance, hardware implementation costs and energy consumption. To meet these requirements, a separate branch of cryptography is being developed, known as low-resource cryptography. Among cryptographic transformations, hash functions play a key role, as they ensure the integrity of information. Therefore, research aimed at implementing hardware data hashing tools is particularly relevant. The use of modern cryptographic hash functions faces problems of limited memory and low computing power. One approach to solving these problems is to create specialised hardware that implements the data hashing function. The ISO/IEC 29192-1:2012 (2012) standard sets out requirements for low-resource cryptography devices. According to the provisions of this standard, typical requirements include ensuring a sufficient level of cryptographic strength under conditions of limited chip area, low device power consumption, and limited time for cryptographic transformation. To compare the hardware complexity of implementations, the standard provides for the use of a metric based on the number of equivalent logic elements (Gate Equivalents, GE). This indicator is a unified assessment of the microchip area, which allows for a comparative analysis of implementations, regardless of the manufacturing technology used. Hardware complexity, expressed in GE, is a key parameter in the design of cryptographic primitives for embedded systems and IoT devices.

In recent studies on the hardware implementation of low-resource hash functions, there has been active development of both existing algorithms and new designs focused on use in resource-constrained environments. The works of D.N. Gupta & R. Kumar (2021) and S. Windarta *et al*. (2022) presented a comparative analysis of existing low-resource hash functions such as HASH-ONE, QUARK, PHOTON, SPONGENT, and GLUON, which are based on the "sponge" design. The comparison was made based on six parameters: security level, hardware complexity, hash value size, throughput, power consumption, and the number of cycles required for execution. SPONGENT-256 provides a hardware complexity of 1,950 GE and the highest collision resistance. The authors concluded that there is no universal solution among the existing algorithms and emphasised the need to develop a new hash function that would provide an adequate level of security at low hardware costs. In particular, the article by D.N. Gupta & R. Kumar (2023) presented a flexible hardware implementation of the DeeR-Hash function for IoT, based on a sponge structure with bitwise transformations. DeeR-Hash supports output lengths of 88 and 160 bits and has low hardware complexity – 984 GE for the 160-bit version. The authors noted its suitability for resource-constrained devices due to its high performance and small chip area.

The article by S. Windarta *et al*. (2023) proposed two new lightweight hash functions, ALIT-Hash and TJUILIK-Hash, based on the Saturnin block cipher with a modified S-block and an operation mode similar to the Beetle sponge structure. The authors investigated the hash functions for resistance to differential and linear cryptanalysis and performed statistical testing of the developed methods using NIST STS. Modelling performed using Contiki-NG and the Cooja simulator shows that the two developed hash functions demonstrate better performance compared to PHOTON and SPONGENT, measured by five indicators. A hardware implementation of TJUILIK-Hash on the ATmega2560 microcontroller was proposed. The authors B. Widhiara *et al*. (2023) also proposed a lightweight hash function RM70 as an alternative to SPONGENT-88, which uses an ARX-SPN type permutation and computes an 88-bit hash. The researchers claimed that RM70 demonstrates better processing time and lower memory consumption compared to SPONGENT-88. The hash function was implemented on the Arduino Uno R3 platform, without providing an assessment of hardware complexity or comparison with FPGA/ASIC implementations.

The work by Y. Huang *et al*. (2021) presented a family of lightweight HVH hash functions based on a "sponge" construction using the VH block cipher. The hash functions support five hash length options ranging from 88 to 256 bits for different security levels and resource constraints. The main focus of the study was to improve the hashing speed, which for an 8-bit microcontroller in RFID reaches 1.47 Mbit/s, which is 10 times higher than SPONGENT-88 hashing. The study also presented a hardware implementation for the TSMC 180 nm process, where the total hardware complexity of HVH with a hash length of 256 bits is 2,009 GE.

The study by P.R. Lawhale *et al*. (2025) presented a lightweight hash function for IoT and big data, based on bit permutations, linear transformations, and S-boxes. The hardware implementation on FPGA showed high results in terms of power consumption and throughput with a hardware complexity of 2,378 logic resources (LR). The authors emphasised the compliance of the proposed solution with security standards and its suitability for a wide range of IoT applications. The article by S. Khan *et al*. (2023) presented optimised hardware implementations of the four finalists of the third round of the NIST LWC competition: PHOTON-Beetle, Ascon, Xoodyak and SPARKLE. The results showed that the PHOTON-Beetle implementation reduces the complexity of the hardware implementation to 50% of the original value. For Ascon and XOODYAK, an increase in throughput was achieved. The SPARKLE hash function was implemented for the first time using FPGA technology.

In the work of V.I. Seleznov & V.A. Luzhetskyi (2023), the HDG (Hash Data Generator) hashing method was

proposed, which takes into account the requirement for low hardware costs for implementation while maintaining the necessary level of security. The peculiarity of this hashing method is that intermediate and final hash values were calculated by adding data bytes at positions determined by the symbol 1 in the code generated by a pseudorandom number generator, which is why the authors called this approach to hashing "data generator". This byte-oriented approach to data hashing does not require padding of the last data block when its length is less than the specified block length. However, the paper provided only a theoretical estimate of the complexity of hardware implementation based on the structure of the device without detailing the specifics of the construction of individual blocks.

Among the hash functions considered, there are hash functions that have a hash value length of less than 128 bits and a hardware complexity of less than 1,000 GE, which are only allowed for use in devices with low security requirements. However, most hash functions calculate hash values with a length of 128-256 bits. The complexity of the hardware implementation of such hash functions ranges from 2,000 to 3,000 GE. Therefore, there is a need for hash functions with a hardware complexity of less than 2,000 GE for hash values with a length of 256 bits. The aim of the study was to reduce the hardware costs of implementing hash functions by developing a specialised processor structure that implements the HDG hash function.

MATERIALS AND METHODS

The HDG hashing function proposed in the work of V.I. Seleznov & V.A. Luzhetskyi (2023) implemented a byte-oriented model based on a modified Merkle-Demgard iterative structure. The peculiarity of the HDG function lies in the use of the "data-generator" hashing approach, according to which the hash value is calculated based on the input data, using the simplest arithmetic operations, a control pseudorandom sequence generated by a specialised generator. The message M to be hashed is presented as a sequence of L bytes:

$$M = \{m_1, m_2, \ldots, m_L\}. \quad (1)$$

The initial hash value h_0 is represented as a sequence of $k = l/8$ bytes, where l is the length of the hash value in bits:

$$h_0 = \{h_0, 0, h_{0,1}, \ldots, h_{0,k-1}\}. \quad (2)$$

The process of calculating intermediate hash values is provided by the compression function f. Each intermediate hash value $h_i = \{h_{i,0}, h_{i,1}, \ldots, h_{i,k-1}\}$ is calculated based on the previous h_{i-1} and the current value of the message byte m_i. For each value , the compression function implements an iterative process consisting of the following steps: generating pseudo-random bits $g_{i,j}$ for each j from 0 to; k-1 calculating the intermediate hash value h_i using the formula:

$$h_{i,j} = (h_{i-1,j} + m_i \cdot g_{i,(k-1-j)}) mod\ 256. \quad (3)$$

The final hash value of the message $h = \{h_{L,0}, h_{L,1}, \ldots, h_{L,k-1}\}$ is calculated as a result of executing the compression function f for the last byte of data M. To meet the requirements of low-resource cryptography, the described hash function is implemented in the form of a specialised processor (HDG processor). This processor is integrated into the computer system by connecting to the central processor. The system's central processor stores and forms the sequence of bytes of the input message for hashing. The HDG processor performs all arithmetic operations necessary for calculating hash values and stores intermediate hash values. The design of the specialised processor is based on the mathematical apparatus of digital automata theory. It consists of two key components: an operational automaton responsible for performing calculations and a control automaton that coordinates the sequence of operations performed (Harris & Harris, 2021). The complexity of implementing the hash function requires the decomposition of the operational automaton into blocks, each of which performs a specific set of functions, which avoids excessive complexity of the structure and increases the efficiency of data processing.

To verify the correct functioning of the specialised processor, the Logisim-evolution (n.d.) simulation environment was used, which allows the logical structure of the device to be created and verified at the level of the simplest elements and functional blocks, providing the ability to analyse in detail the operation of each block and their interaction. The use of modelling tools made it possible to evaluate the correctness of the device's operation and determine its hardware complexity even before the physical implementation stage. The hardware implementation of the developed HDG processor is planned on an ASIC platform using 0.18 μm technology and the UMCL18G212T3 library (Virtual Silicon Inc, 2004). The hardware complexity of the HDG processor was calculated in GE units. The research consisted of three main stages: development of structural components of the specialised HDG processor; modelling of individual components and the specialised processor as a whole in the Logisim-evolution environment; calculation of the hardware complexity of the HDG processor in conventional units (GE).

RESULTS AND DISCUSSION

Based on the characteristics of the HDG hash function, the structure of the specialised processor is represented as a set of four components: a block of registers (BRG) for storing intermediate hash values; a pseudorandom sequence generator implemented as a linear feedback shift register (LFSR); a modulo 256 addition block (ADDB); a control block (CB) that communicates with the central processor, implements the initialisation process and controls the data hashing device. The structural diagram of the HDG processor is shown in Figure 1.

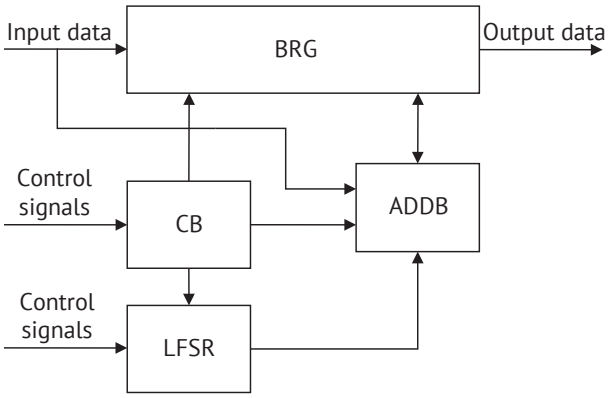

Figure 1. Block diagram of a specialised processor
Source: developed by the authors

The interaction between the HDG processor and the central processor takes place via a data exchange bus. The input bus provides byte-by-byte data input for hashing, while the output bus is used to transfer the hashing result of the processed data. The input signals of the HDG processor are clock signals and other control signals. The BRG block consists of 32 serially connected eight-bit registers that store intermediate hash values of 256 bits in length and are connected to data buses. By using a control input, the block provides the ability to set the initial state of the registers, which is the initial hash value. In the case of a keyless hash function, this initial state is known. If a keyed hash function is implemented, the initial state is a secret key.

The LFSR block is responsible for generating pseudo-random values. The LFSR is based on a 32-bit shift register with linear feedback of maximum period. The output signal of the LFSR is a generated pseudo-random bit *g*, which is then fed to the ADDB block. The LFSR also uses a separate control signal to initialise the initial state of the register. The ADDB block performs the addition operation modulo 256 of two 8-bit numbers. ADDB consists of eight 1-bit adders and eight AND logic elements, which together implement the operation described by formula (1). Unlike other blocks, ADDB does not require clock signals to operate. The CB block provides initialisation and synchronisation of other blocks. It consists of a five-bit counter and an RS flip-flop. Based on the control signal, the CB block sets the counter to zero and, during the next 32 cycles, controls the recording of the initial states of the BRG block registers and the LFSR register. After initialisation is complete, the device enters the hash value calculation mode. The HDG processor and its components are presented in the form of models:

$$HDG = \{D, R, S, Y, X\}, \quad (4)$$

where D is the set of input signals received by the device; R is the set of output signals resulting from hashing; S is the set of internal states used to store intermediate results during the execution of operations; Y is the set of operations performed by the device; X is the set of logical conditions. The set of input signals D includes: data bytes m_i to be hashed; bits S_0 that make up the initial state of the LFSR register. The set of output signals R includes the hashing result H. Among the internal states S of the device are: S_{BRG} – the state values of the BRG block registers; S_{LFSR} – the state of the LFSR; S_{CB} – the state of the CB block components. The set Y includes the following operations: synchronisation (clock signal clk0); initialisation of the initial state of the HDG processor and its components (signal rst_0); cumulative addition *ADD*. The model of the hardware device that implements the HDG hash function is shown in Figure 2.

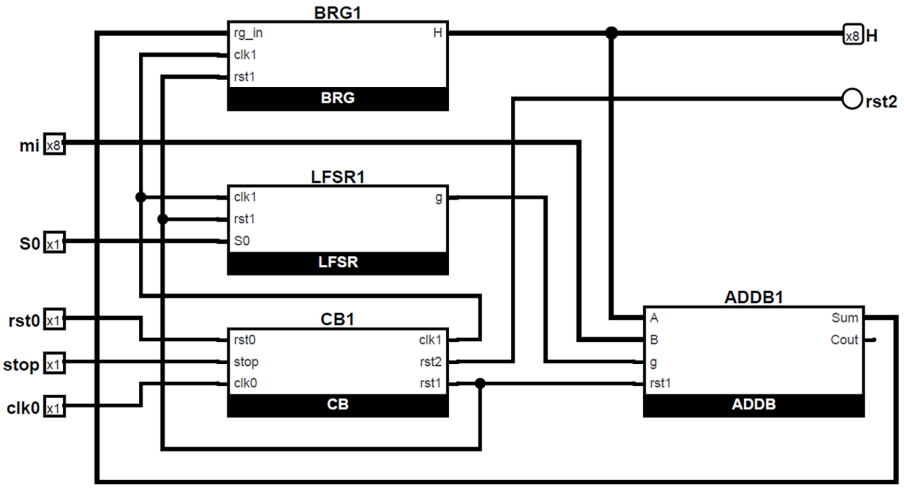

Figure 2. HDG processor model
Source: developed by the authors

Thus, the HDG processor model is described by the following set of sets.

1. Set of input signals $D = \{ m_i, S_0\}$.
2. Set of output signals $R = \{H\}$.

3. Set of internal states $S = \{ S_{BRG}, S_{LFSR}, S_{CB}\}$.
4. Set of operations $Y = \{clk_0, rst_0, ADD\}$.
5. Set of logical conditions $X = \{ x_0, x_1\}$, where: x_0 – condition that $rst_2 = 1$; x_1 – condition that $stop = 1$.

The LFSR block is a 32-bit shift register with linear feedback, the diagram of which is shown in Figure 3, and the algorithm of its operation is shown in Figure 4.

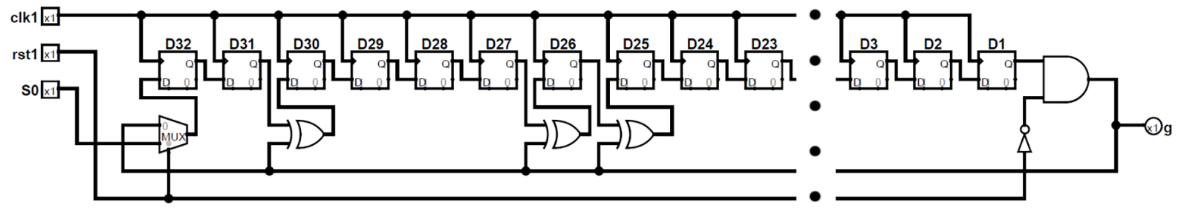

Figure 3. Fragment of the LFSR block model

Source: developed by the authors

Thus, the LFSR block model is described by the following set of sets.

1. Set of input signals $D = \{S_0\}$.
2. Set of output signals $R = \{g\}$.
3. Set of internal states $S = \{D_1 \dots D_{32}\}$.
4. Set of operations $Y = \{clk_1, rst_1, y_1, y_2\}$.

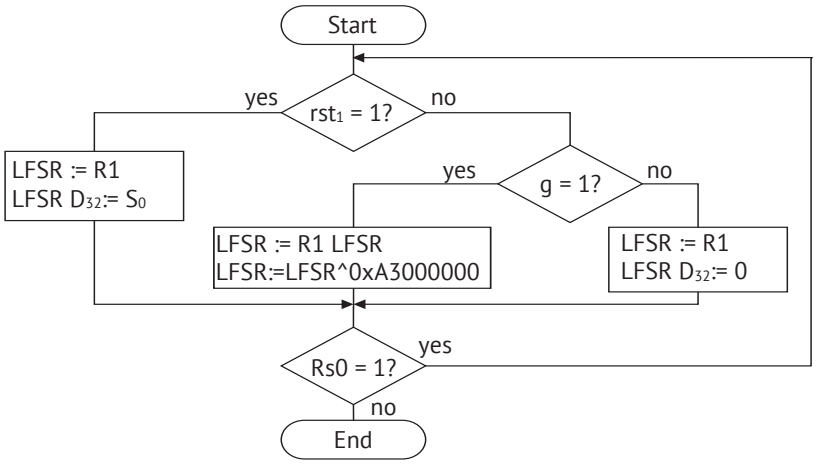

Figure 4. LFSR block operation algorithm

Source: developed by the authors

It consists of 32 D-triggers D_{32} – D_1, where each trigger is responsible for storing one bit of state. A 2-in-1 multiplexer (MUX) is connected to the input of trigger D_{32}, and an AND logic element is connected to the output of D_1, to which the inverted value of the rst_1 signal is supplied through another input. When the rst_1 signal is 1, the multiplexer transmits S_0 signals to the input of trigger D_{32} to initialise the initial state of the LFSR, and the AND element breaks the feedback from the output of trigger D_1 by forming $g = 0$. If the rst_1 signal is 0, the multiplexer restores the connection from the output of the AND element to trigger D_{32}, and the AND element restores the feedback to the XOR elements connected to the inputs of the D-triggers, the numbers of which are determined by a primitive polynomial:

$$P(x) = x^{32} + x^{30} + x^{26} + x^{25} + 1. \quad (5)$$

The presented configuration provides minimal hardware costs for implementing a shift register with linear feedback and a maximum period of the pseudorandom sequence $2^{32}–1$ (Ward & Molteno, 2012). The input signals D of the LFSR block include: bit S_0 (32 bits S_0 constitute the initial state of the LFSR register). The initial state of the LFSR is the secret key code in the case of a key hash function or a cryptographic salt, which provides an additional level of cryptographic resistance in the case of a keyless hash function. The output signal of the block is a pseudorandom bit g. Among the internal states S of the LFSR block are the values of the D-flip-flops ($D_1 \dots D_{32}$). The set Y includes the following operations: synchronisation (clock signal clk_1); initialisation of the initial state of the LFSR HDG processor and its components (signal rst_1); writing to a register with feedback (operation y_1) and writing without feedback (operation y_2).

The BRG block consists of 32 eight-bit registers Rg_1 – Rg_{32} connected in series, where each register is responsible for storing eight bits of state. An eight-bit AND logic element is connected to the output of register Rg_{32}, to the other input of which the inverted value of the rst_1 signal is supplied, as shown in Figure 5.

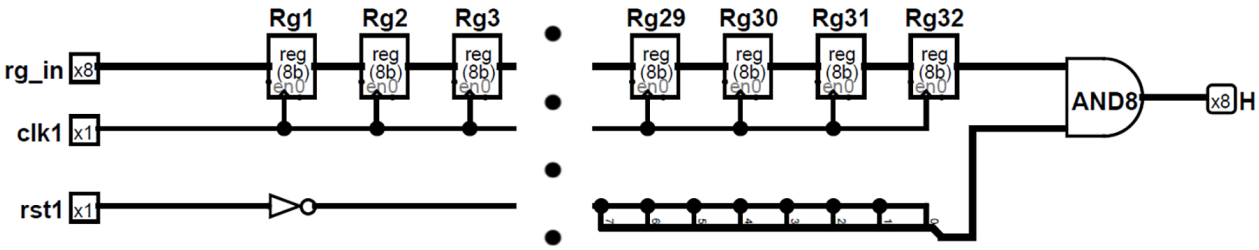

Figure 5. Fragment of the BRG block model

Source: developed by the authors

When $rst_1 = 1$, the AND element provides the BRG block output signal $H = 0$, which is necessary to set the initial state of the BRG block. Setting the initial state of BRG consists of sequentially writing the bytes of the initial hash value H_0 to the block registers. The H_0 bytes are sent to the HDG processor as an input signal m_i through the ADDB block, where they are added with zero bytes (output signal $H = 0$ of the BRG block) and written to the BRG block (signal *rg_in*). If $rst_1 = 0$, then the AND element ensures the transmission of the signal $H = Rg_{32}$. All these actions are reflected in the BRG block operation algorithm shown in Figure 6.

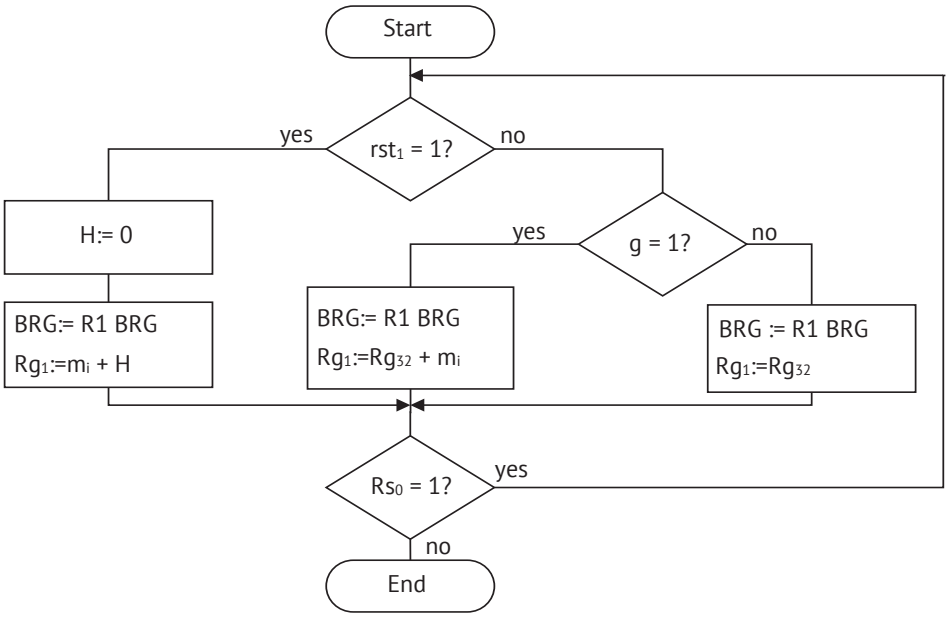

Figure 6. BRG block operation algorithm

Source: developed by the authors

The set of input signals *D* of the BRG block includes the *rg_in* byte. The output signal of the block is the *H* byte (the first 32 bytes H_i form an intermediate hash value, and the last 32 bytes *H* are the result of hashing). Among the internal states S of the BRG block are the values of the eight-bit registers ($Rg_1 \dots Rg_{32}$). The set *Y* includes the following operations: synchronisation (clock signal clk_1); initialisation of the BRG initial state (signal rst_1).

Thus, the model of the BRG block included in the HDG processor is described by the following set of sets.

1. Set of input signals $D = \{rg\_in\}$.
2. Set of output signals $R = \{H\}$.
3. Set of internal states $S = \{Rg_1 \dots Rg_{32}\}$.
4. Set of operations $Y = \{clk_1, rst_1\}$.

The ADDB block performs the addition of values *A* and *B*. ADDB is an 8-bit combinational adder with additional control of operand *B*, as shown in Figure 7. Operands A and B are 8-bit binary codes. Operand *B* is the current byte of the input message mi, and operand *A* is the output byte from the output of register Rg_{32} of the BRG block. The ADDB block has two control signals (signals rst_1 and *g*) that are fed to the inputs of the OR element. The value of the *g* signal is a pseudo-random bit coming from the LFSR block. The value $rst_1 = 1$ corresponds to the initialisation process of the BRG and LFSR blocks, during which $g = 0$ (provided by the control in the LFSR block), the zero code is supplied as operand *A* (provided by the control in the BRG block), therefore $Sum = B$. When $rst_1 = 0$, operand *B* is controlled by signal *g*. If $g = 1$, then operand *B* is sent to the ADDB input. Otherwise, a zero code will be sent instead of operand *B*.

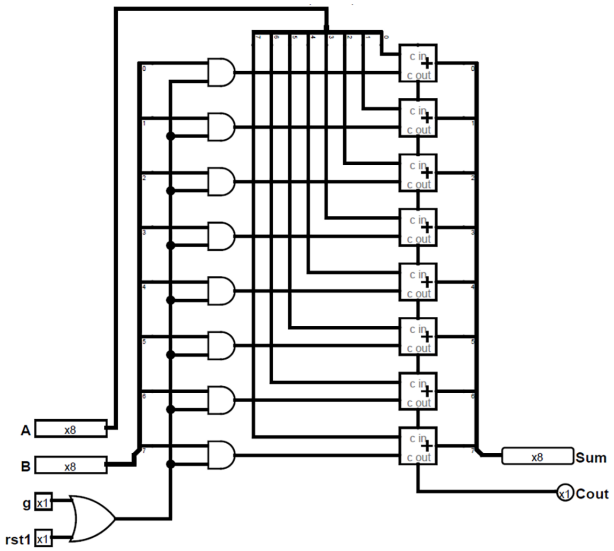

Figure 7. ADDB block model

Source: developed by the authors

The CB block controls the data hashing process and interacts with the central processor. Among the input signals *D* of the CB block, the clock signal clk_0, signals rst_0 and *stop* are described above for the HDG processor. The set of output signals of the block includes the clk_1 clock signal, the rst_2 signal, whose change indicates the completion of hashing of the *i*-th byte of data by the HDG processor, and rst_1, which controls other blocks. The internal state of the $S_{Counter1}$ and the states of the S_{Rs0}, S_{Rs1} flip-flops are included in the set *S* of block CB. The main operations are setting the triggers to the unit state (SET_{Rs0} and SET_{Rs1}), setting the triggers and counter to the zero state ($RESET_{Rs0}$, $RESET_{Rs1}$ та $RESET_{Counter1}$, respectively), and the *INC* counting operation.

The model of the CB block shown in Figure 8 is described by a set of sets.

1. Set of input signals D = {clk_0, rst_0, *stop*}.
2. Set of output signals R = {clk_1, rst_1, rst_2}.
3. Set of internal states S = { $S_{Counter1}$, S_{Rs0}, S_{Rs1}}.
4. Set of operations *Y* = {SET_{Rs0}, SET_{Rs1}, $RESET_{Rs0}$, $RESET_{Rs1}$, $RESET_{Counter1}$, *INC*}.

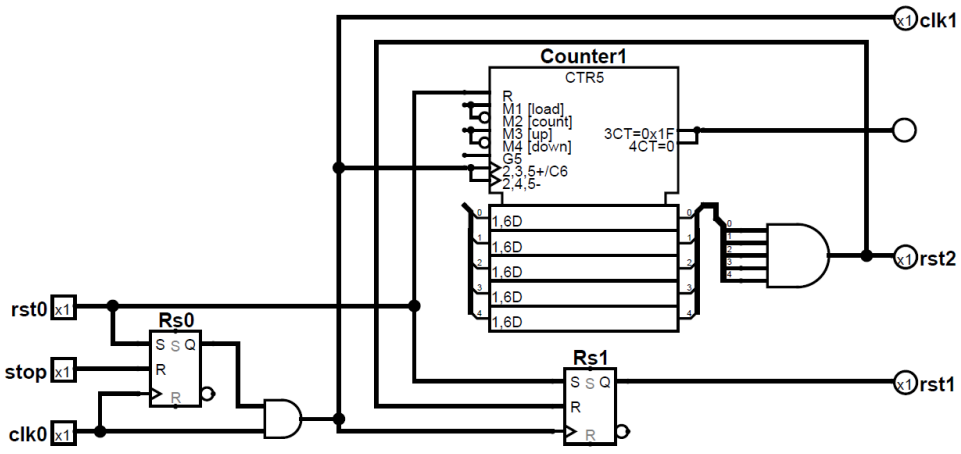

Figure 8. Model to CB block

Source: developed by the authors

The CB block operation algorithm is shown in Figure 9. The block initialisation begins with the rst_0 signal, which sets the Rs_0 trigger to the unit state, thanks to which the clk_1 clock signals begin to be received by the block. In addition, the Rs_1 flip-flop is set to the one state, and the *Counter1* counter is set to the initial zero state and starts counting. Thanks to the Rs_1 flip-flop, the rst_1 signal acquires a constant one value for 32 cycles, which allows the BRG and LFSR blocks to be initialised (Fig. 2). When the counter state reaches code 11111, a $rst_2 = 1$ signal is generated, which sets Rs_1 to zero, which in turn completes the initialisation mode. After initialisation is complete, the device enters the hash value

calculation mode. When the hashing of the entire message is complete, the central processor sends 32 bytes of zero values to the HDG processor to read the hashing result, and then sends a *stop* = 1 signal to complete the operation, which sets the Rs_0 flip-flop to zero, stopping the formation of clk_1 clock signals.

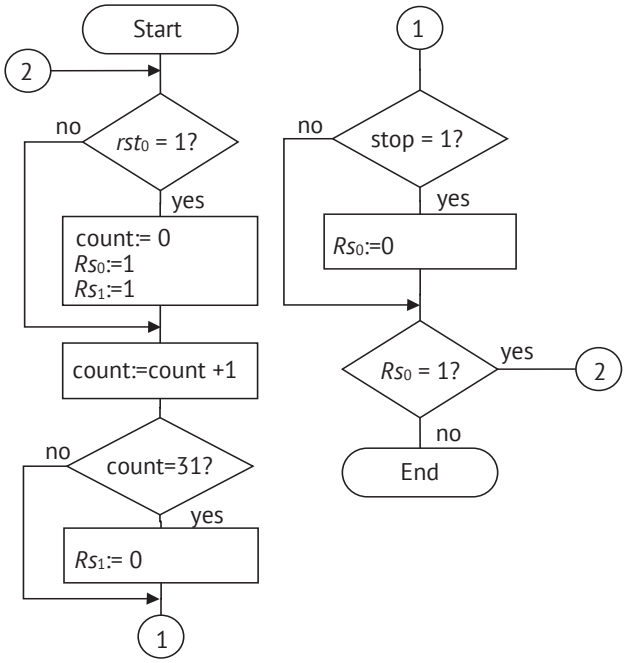

Figure 9. Algorithm of the CB block operation
Source: developed by the authors

The results of modelling using the Logisim-evolution environment confirmed the correctness of the logical structure of the specialised processor and the interaction of its components in accordance with the given algorithms. The developed models were used to calculate the hardware complexity of each component and the HDG processor as a whole. The total hardware complexity of the HDG processor was calculated using the formula:

$$S_{HDG} = S_{BRG} + S_{LFSR} + S_{ADDB} + S_{CB}, \quad (6)$$

where S_{BRG} – complexity of the BRG block; S_{LFSR} – complexity of the LFSR pseudorandom number generator; complexity of the ADDB addition block; S_{ADDB} – complexity of the CB control block. The BRG consists of 32 registers, each of which consists of eight *D Flip flops*, eight NAND elements and one NOT element, so the total complexity of the block is;

$$S_{BRG} = 32 \cdot 8 \cdot D + 8NAND + NOT. \quad (7)$$

The LFSR consists of 4 XOR elements, 32 *D*-triggers, a 2-in-1 MUX multiplexer, one NOT and one AND element, therefore:

$$S_{LFSR} = 4XOR + 32 \cdot D + MUX + NOT + AND. \quad (8)$$

The ADDB block consists of 8 AND elements, one OR element, and 8 single-bit adders, each of which, in turn, consists of two AND elements, one OR element, and two XOR elements:

$$S_{ADDB} = 8AND + OR + 8\,(2AND + OR + 2XOR). \quad (9)$$

The CB control module consists of two RS triggers, one AND element and a 5-bit counter. The RS trigger consists of 2 NAND elements. The controlled 5-bit counter consists of 4 AND elements and 5 *D* triggers. Complexity of the CB module:

$$S_{CB} = 4NAND + 5D + 5AND. \quad (10)$$

Based on formulas (7-10), the hardware complexity is as follows:

$$S_{HDG} = 293D + 2NOT + 29AND + 10NAND + 9OR + \\ + 20XOR + MUX. \quad (11)$$

The complexity of implementing an HDG processor that calculates 256-bit hash values on a microchip using 0.18 μm technology and the UMCL18G212T3 library in GE conventional units: $D = 5.33$; NOT = 0.67; NAND = 1; AND = 1.33; OR = 1.33; MUX = 2.33; XOR = 2.67. Based on this, the resulting complexity for the HDG-processor implementation is:

$$S_{HDG} = 1{,}683\text{GE}. \quad (12)$$

To compare the obtained estimate of the complexity of the hardware implementation of the HDG hash function, Table 1 shows estimates for known low-resource hash functions. Although the ISO/IEC 29192 standard allows the use of hash functions that generate hash values ranging from 80 to 256 bits for low-resource applications, there has been a recent trend towards higher bit-length hash values, namely up to 256 bits. Therefore, Table 1 lists the PHOTON, S-Quark, SPONGENT, and HVH hash functions, which generate output hash values with a length of 224 and 256 bits. At the same time, for the sake of completeness, functions with shorter hash values (160-224 bits), such as DeeR-Hash, Hash-One and GLUON, were also included.

Table 1. Hardware complexity of hashing devices

Hash function	Structure	Hash value length, bits	Hardware complexity, GE
DeeR-Hash	Sponge-based	160	984
Hash-One	Sponge-based	160	1,006
D-Quark	Sponge-based + NFSR	176	1,702
S-Quark		256	2,296

Continued Table 1.

Hash function	Structure	Hash value length, bits	Hardware complexity, GE
PHOTON	Sponge-based	224	1,736
		256	2,177
SPONGENT	Sponge-based	224	1,728
		256	1,950
GLUON	Sponge-based	160	2,800
		224	4,724
HVH	Sponge-based + block cipher VH	224	1,769
		256	2,009

Source: developed by the authors based on Y. Huang *et al.* (2021), S. Windarta *et al.* (2022), D.N. Gupta & R. Kumar (2023)

The PHOTON hash function proposed in the study by J. Guo *et al.* (2011) uses a sponge-type design with an AES-like internal permutation structure. PHOTON requires 2,177 GEs to compute 256-bit hash values. Each round of permutation uses operations based on S-blocks, bitwise addition of constants, cyclic row shifts, and linear column transformations. Compared to HDG, PHOTON has a more complex structure because it uses a classic cryptographic symmetric design with substitution tables. J.P. Aumasson *et al.* (2010) developed the D-Quark and S-Quark hash functions, which are also based on the "sponge" design, but unlike PHOTON, they use shift registers as their core. S-Quark uses three shift registers: two non-linear NFSRs and one linear LFSR, which provides bitwise processing without the use of substitution tables or matrices. Compared to HDG, which also uses register generation, S-Quark is limited to logical operations only. Unlike PHOTON, where the construction is based on S-blocks and linear transformations, transformations in S-Quark are implemented using a combination of gates. However, as the length of the hash value increases, the logic of registers and combined functions becomes more complex, leading to an increase in logical depth and, accordingly, to a more complex and slower hardware implementation.

As noted in the works of C. Manifavas *et al.* (2014) and I. El Gaabouri *et al.* (2022), for implementation on devices with very low resources (e.g., 4-bit microcontrollers), hardware complexity of up to 1,000 GE is considered acceptable, and for more general IoT platforms, implementations of up to 2,000-3,000 GE are acceptable. A separate section of the ISO/IEC 29192-5:2016 (2016) standard defines requirements for hash functions intended for use in environments with limited computing and energy resources. The key criterion here is to ensure cryptographic resistance to collisions and attacks on the first prototype with minimal hardware costs. The minimum acceptable hash length is considered to be 80 bits, while ensuring a basic level of collision resistance of 2^{40} operations, which is satisfactory for typical applications in the IoT environment. At the same time, given the growth in the computing capabilities of potential attackers and the increasing requirements for long-term security, modern scientific publications note a trend towards a gradual transition to hash functions with an output length of 256 bits, even in low-resource implementations (Bogdanov *et al.*, 2013; Buchanan *et al.*, 2017; Seleznov, 2023). This approach allows for collision resistance at the level of 2^{128} attempts, which is critical for secure systems with long lifecycles or high requirements for data confidentiality and integrity.

H. Martin *et al.* (2017) proposed a low-resource implementation of the Tav-128 hash function for use in RFID tags and sensor nodes. Tav-128 is based on the classic Merkle-Demgard design proposed in I. Damgård (1989) and R.C. Merkle (1990). The study presented three variants of the ASIC hardware implementation of this function, optimised to minimise area, maximise throughput, and achieve a balance between these parameters. The smallest area implementation of Tav-128 occupies 3,194 GE, which exceeds the authors' initial estimate (2,578 GE) due to the complexity of the control logic. A. Bogdanov *et al.* (2013) developed the SPONGENT hash function, which also implements a "sponge" structure with internal permutation based on the low-resource block cipher PRESENT. The main transformations are 4-bit S-boxes, bitwise permutations, and bitwise XOR operations, which makes SPONGENT similar to PHOTON. At the same time, unlike PHOTON, the authors use bitwise permutations as a linear diffusion transformation, which reduces the complexity of hardware implementation but worsens the diffusion properties. Research by D.N. Gupta & R. Kumar (2021) showed the possibility of implementing SPONGENT with a hardware complexity of 1,950 GE, which is one of the best indicators for 256-bit hashing.

The GLUON hash function calculates hash values based on a combination of Feedback with Carry Shift Registers (FCSR) and simple logic. As shown by research by D.N. Gupta & R. Kumar (2021), this structure allows for high entropy, but the authors only propose a hardware implementation, which provides faster hashing but comes with significant hardware costs: GLUON implementation requires 2,800 to 4,700 GE. Hash-One, developed by P.M. Mukundan *et al.* (2016), similar to the S-Quark hash function, implements a

"sponge" design with two non-linear feedback shift registers (NFSR). In addition, Boolean functions (P_j, Q_j, L_j) implemented exclusively using NAND gates are used as the main transformations. This allows for a hardware complexity of 1,006 GE to calculate 160-bit hash values. Unlike HDG, which uses a pseudorandom bit generator to control bytewise addition operations, Hash-One relies on fixed state update logic via NFSR. In turn, DeeR-Hash, presented by D.N. Gupta & R. Kumar (2023), also uses two NFSRs and additionally introduces one LFSR to increase diffusion. Both functions have a similar structure and implement state updates through Boolean functions, but DeeR-Hash uses a dynamic scheme for selecting variables from arrays, which allows for optimising the logic. A system of multiplexers and configuration arrays is used to select positions, forming updates with a minimal set of logic elements. Thanks to this, DeeR-Hash provides 160-bit hash value calculations with a hardware complexity of 984 GE. Proposed by Y. Huang *et al*. (2021), the HVH hash function implements a "sponge" design based on the lightweight VH block cipher with a block size of 64 bits. VH is constructed as a substitution-permutation network (SPN) that determines the nature of transformations during hashing. HVH supports five hash length options: 88, 128, 160, 224, and 256 bits, allowing it to be adapted to various resource-constrained scenarios. The authors estimate the complexity of the hardware implementation of HVH for 256-bit hash values at 2,009 GE, which is on par with the SPONGENT hash function.

Comparing the structural characteristics of all the hash functions considered, it can be noted that PHOTON, SPONGENT, and HVH use either AES-like transformations or block ciphers with S-blocks, which increases the complexity of the logic. On the other hand, GLUON, Quark, HASH-ONE, and DeeR-Hash, like HDG, are focused on using shift registers, bitwise operations, and simple logical functions. A common feature of all the hash functions studied is the use of a "sponge" construction or its modifications, except for HDG, which does not involve absorption and squeezing procedures, but instead implements the classic Merkle-Demgard iterative construction for forming hash values. Thus, the proposed HDG processor has a hardware complexity of 1,683 GE for a hash value length of 256 bits, which is the lowest complexity among all the low-resource hash functions considered. A comparative analysis of hardware complexities also showed that in some cases, the HDG processor is simpler than hardware devices that generate hash values of smaller bit lengths, in particular for the GLUON (160 bits), D-Quark (176 bits) and PHOTON (224 bits) hash functions.

CONCLUSIONS

The article presented the hardware implementation of the HDG hash function in the form of a specialised processor designed for use in low-resource devices thanks to its simple byte-oriented architecture, which ensures efficient data processing at the level of individual bytes. The development took into account the requirements for minimising hardware costs, simplicity of logic and ensuring an adequate level of cryptographic strength in accordance with the ISO/IEC 29192 standard. The proposed structure of the HDG processor for hashing was decomposed into four main modules: BRG, LFSR, ADDB, and a CB. Detailed design of each of the blocks allowed formalising the functional logic of the processor, adapted for digital implementation. The processor and its components were modelled in the Logisim-evolution environment, which allowed confirming the correct operation of the HDG processor and its compliance with the stated functional requirements.

During the modelling process, it was confirmed that the HDG processor correctly performs the algorithmically defined sequence of hashing operations and generates the expected hash value for the test input data, which indicates the functional consistency of the implemented hardware logic with the basic specification of the algorithm. The results of the hardware complexity analysis showed that the implementation of the HDG processor for calculating 256-bit hash values requires 1,683 GE, which meets the requirements for low-resource cryptography and indicates the possibility of using the HDG processor in environments with restrictions on hardware complexity.

A comparative assessment with hardware implementations of known hash functions, in particular PHOTON, SPONGENT, S-Quark, GLUON and HVH, showed that the proposed implementation provides a reduction in hardware costs of 15% or more, depending on the selected analogue function. In some cases, the HDG processor implementation even provides lower hardware complexity when generating a 256-bit hash than some known functions that work with shorter hash values, such as 224 or 160 bits, which demonstrates the effectiveness of the design underlying the HDG processor. This level of optimisation was achieved by using only elementary logical and arithmetic operations without involving complex nonlinear transformations. Thus, the developed HDG processor is suitable for use in devices with limited hardware resources and can be the basis for further research in the field of low-resource cryptography. It is promising to study the possibility of improving the architecture of the HDG processor for effective implementation using FPGA technology.

ACKNOWLEDGEMENTS

None.

FUNDING

None.

CONFLICT OF INTEREST

None.

REFERENCES

[1] Aumasson, J.P., Henzen, L., Meier, W., & Naya-Plasencia, M. (2010). QUARK: A lightweight hash. In S. Mangard & F.-X. Standaert (Eds.), *Cryptographic hardware and embedded systems – CHES 2010* (pp. 1-15). Cham: Springer. doi: 10.1007/978-3-642-15031-9_1.

[2] Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., & Verbauwhede, I. (2013). SPONGENT: The design space of lightweight cryptographic hashing. *IEEE Transactions on Computers*, 62(10), 2041-2053. doi: 10.1109/TC.2012.196.

[3] Buchanan, W.J., Li, S., & Asif, R. (2017). Lightweight cryptography methods. *Journal of Cyber Security Technology*, 1(3-4), 187-201. doi: 10.1080/23742917.2017.1384917.

[4] Damgård, I. (1989). A design principle for hash functions. *Lecture Notes in Computer Science*, 435, 416-427. doi: 10.1007/0-387-34805-0_39.

[5] El Gaabouri, I., Senhadji, M., & Belkasmi, M. (2022). A survey on lightweight cryptography approach for IoT devices security. In *2022 5th international conference on networking, information systems and security: Envisage intelligent systems in 5g//6G-based interconnected digital worlds (NISS)* (pp. 1-8). Bandung: IEEE. doi: 10.1109/NISS55057.2022.10085144.

[6] Guo, J., Peyrin, T., & Poschmann, A. (2011). *The PHOTON family of lightweight hash functions*. Retrieved from https://eprint.iacr.org/2011/609.

[7] Gupta, D.N., & Kumar, R. (2021). Sponge based lightweight cryptographic hash functions for IoT applications. *In 2021 international conference on intelligent technologies (CONIT)* (pp. 1-5). Hubli: IEEE. doi: 10.1109/conit51480.2021.9498572.

[8] Gupta, D.N., & Kumar, R. (2023). DeeR-Hash: A lightweight hash construction for Industry 4.0 / IoT. *Journal of Scientific & Industrial Research*, 82(1), 142-150. doi: 10.56042/jsir.v82i1.69938.

[9] Harris, S.L., & Harris, D. (2021). Digital design and RISC-V computer architecture textbook. In *2021 ACM/IEEE workshop on computer architecture education (WCAE)* (pp. 1-5). Raleigh: IEEE. doi: 10.1109/wcae53984.2021.9707615.

[10] Huang, Y., Li, S., Sun, W., Dai, X., & Zhu, W. (2021). HVH: A lightweight hash function based on dual pseudo-random transformation. In *Security, privacy, and anonymity in computation, communication, and storage* (pp. 492-505). Cham: Springer. doi: 10.1007/978-3-030-68884-4_41.

[11] ISO/IEC 29192-1:2012. (2012). *Information technology – security techniques – lightweight cryptography – Part 1: General*. Retrieved from https://www.iso.org/standard/56425.html.

[12] ISO/IEC 29192-5:2016. (2016). *Information technology – security techniques – lightweight cryptography – Part 5: Hash-functions*. Retrieved from https://www.iso.org/standard/67173.html.

[13] Khan, S., Lee, W.-K., Karmakar, A., Mera, J.M.B., Majeed, A., & Hwang, S.O. (2023). Area-time efficient implementation of NIST lightweight hash functions targeting IoT applications. *IEEE Internet of Things Journal*, 10(9), 8083-8095. doi: 10.1109/jiot.2022.3229516.

[14] Lawhale, P.R., Kale, S.N., Kasturiwale, H., & Thakare, Y.N. (2025). FPGA implementation of compact architecture for lightweight hash algorithm for resource constrained devices. *Communications on Applied Nonlinear Analysis*, 32(2), 679-691. doi: 10.52783/cana.v32.1861.

[15] Logisim-evolution. (n.d.). *Digital logic design tool and simulator*. Retrieved from https://surl.li/pjzuiy.

[16] Manifavas, C., Hatzivasilis, G., Fysarakis, K., & Rantos, K. (2014). Lightweight cryptography for embedded systems – a comparative analysis. In *Data privacy management and autonomous spontaneous security* (pp. 333-349). Berlin: Springer. doi: 10.1007/978-3-642-54568-9_21.

[17] Martin, H., Lopez, P.P., San Millan, E., & Tapiador, J.E. (2017). A lightweight implementation of the Tav-128 hash function. *IEICE Electronics Express*, 14(11), article number 20161255. doi: 10.1587/elex.14.20161255.

[18] Merkle, R.C. (1990). One way hash functions and DES. *Lecture Notes in Computer Science*, 435, 428-446. doi: 10.1007/0-387-34805-0_40.

[19] Mukundan, P.M., Manayankath, S., Srinivasan, C., & Sethumadhavan, M. (2016). Hash-One: A lightweight cryptographic hash function. *IET Information Security*, 10(5), 242-252. doi: 10.1049/iet-ifs.2015.0385.

[20] Seleznov, V.I. (2023). Analysis of low-resource hashing methods. In *LII scientific and technical conference of VNTU departments: Proceedings of the scientific conference* (pp. 21-23). Vinnytsia: Vinnytsia National Technical University.

[21] Seleznov, V.I., & Luzhetskyi, V.A. (2023). Low-resource hashing method of the "data – generator" type. *Cybersecurity: Education, Science, Technique*, 28, 84-95.

[22] Virtual Silicon Inc. (2004). *0.18 µm VIP standard cell library tape out ready (part number: UMCL18G212T3). Process: UMC logic 0.18 µm generic II technology*. Retrieved from https://www.eetimes.com/virtual-silicon-ships0-18-micron-libraries/.

[23] Ward, R., & Molteno, T.C.A. (2012). *Table of linear feedback shift registers*. Otago: University of Otago.

[24] Widhiara, B., Kurniawan, Y., & Susanti, B.H. (2023). RM70: A lightweight hash function. *IAENG International Journal of Applied Mathematics*, 53(1), 94-102.

[25] Windarta, S., Suryadi, Ramli, K., Pranggono, B., & Gunawan, T.S. (2022). Lightweight cryptographic hash functions: Design trends, comparative study, and future directions. *IEEE Access*, 10, 82272-82294. doi: 10.1109/access.2022.3195572.

[26] Windarta, S., Suryadi, S., Ramli, K., Lestari, A.A., Wildan, W., Pranggono, B., & Wardhani, R.W. (2023). Two new lightweight cryptographic hash functions based on saturnin and beetle for the Internet of Things. *IEEE Access*, 11, 84074-84090. doi: 10.1109/access.2023.3301128.

Апаратна реалізація геш-функції HDG

Володимир Лужецький

Доктор технічних наук, професор
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0000-0001-7466-7738

Віталій Селезньов

Асистент
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0009-0004-0225-9697

Анотація. З огляду на зростання ролі Інтернету речей та пов'язаних із ним малоресурсних пристроїв, дослідження геш-функцій, що забезпечують високий рівень криптографічної стійкості з мінімальними апаратними витратами, є актуальним. Метою дослідження було запропонувати апаратну реалізацію нової геш-функції HDG, призначеної для застосування в малих пристроях, у вигляді спеціалізованого процесора з метою зменшення апаратних витрат на реалізацію. Методи дослідження включали структурне проектування кожного функціонального блоку, цифрове моделювання в середовищі Logisim-evolution та синтез на ASIC-платформі за технологією 0,18 μm з використанням стандартної бібліотеки UMCL18G212T3, а також розрахунок апаратної складності в умовних одиницях Gate Equivalents. HDG відповідає вимогам малоресурсної криптографії завдяки своїй байторієнтованій архітектурі, що дозволяє обробку даних на рівні окремих байтів, забезпечуючи високу ефективність за обмежених ресурсів пам'яті та обчислювальних можливостей пристроїв. Представлено структуру спеціалізованого процесора для гешування. Здійснено декомпозицію спеціалізованого процесора HDG на чотири функціональні блоки, кожен із яких реалізує відповідну функцію: блок регістрів для зберігання проміжних геш-значень; регістр зсуву з лінійним зворотним зв'язком, що забезпечує генерування псевдовипадкової послідовності; блок додавання за модулем 256 та блок керування. Результати моделювання підтвердили коректність структури спеціалізованого процесора та взаємодії його компонентів. Обрахована складність апаратної реалізації HDG-процесора становить 1 683 GE для обчислення 256-бітного геш-значення, що відповідає вимогам міжнародного стандарту ISO/IEC FDIS 29192 для малоресурсної криптографії. Порівняння з апаратними реалізаціями відомих малоресурсних геш-функцій PHOTON, SPONGENT, S-Quark, GLUON та HVH показало зниження апаратних витрат на 15 % і більше. В окремих випадках HDG-процесор демонструє меншу складність реалізації для геш-значення довжиною 256 біт порівняно з геш-функціями, що забезпечують обчислення геш-значення довжиною 224 або 160 біт, що свідчить про ефективність розробленої структури та доцільність використання такого спеціалізованого процесора для пристроїв з обмеженими апаратними ресурсами

Ключові слова: апаратний засіб; спеціалізований процесор; малоресурсна криптографія; метод гешування; апаратна складність