computers

Article

Hardware-Based Implementation of Algorithms for Data Replacement in Cache Memory of Processor Cores

Larysa Titarenko [1,2], Vyacheslav Kharchenko [3], Vadym Puidenko [3,4], Artem Perepelitsyn [3,*] and Alexander Barkalov [1]

1 Institute of Metrology, Electronics and Computer Science, University of Zielona Gora, ul. Prof. Z. Szafrana, 2, 65-516 Zielona Gora, Poland; l.titarenko@imei.uz.zgora.pl (L.T.); a.barkalov@imei.uz.zgora.pl (A.B.)
2 Department of Infocommunication Engineering, Faculty of Infocommunications, Kharkiv National University of Radio Electronics, Nauky Avenue 14, 61166 Kharkiv, Ukraine
3 Department of Computer Systems, Networks and Cybersecurity, National Aerospace University "KhAI", 17, Chkalov Str., 61070 Kharkiv, Ukraine; v.kharchenko@csn.khai.edu (V.K.); v.puidenko@csn.khai.edu (V.P.)
4 Kharkiv Radio Engineering Professional College, Sums'ka Str., 18/20, 61057 Kharkiv, Ukraine
* Correspondence: a.perepelitsyn@csn.khai.edu

Abstract: Replacement policies have an important role in the functioning of the cache memory of processor cores. The implementation of a successful policy allows us to increase the performance of the processor core and the computer system as a whole. Replacement policies are most often evaluated by the percentage of cache hits during the cycles of the processor bus when accessing the cache memory. The policies that focus on replacing the Least Recently Used (LRU) or Least Frequently Used (LFU) elements, whether instructions or data, are relevant for use. It should be noted that in the paging cache buffer, the above replacement policies can also be used to replace address information. The pseudo LRU (PLRU) policy introduces replacing based on approximate information about the age of the elements in the cache memory. The hardware implementation of any replacement policy algorithm is the circuit. This hardware part of the processor core has certain characteristics: the latency of the search process for a candidate element for replacement, the gate complexity, and the reliability. The characteristics of the PLRU$_t$ and PLRU$_m$ replacement policies are synthesized and investigated. Both are the varieties of the PLRU replacement policy, which is close to the LRU policy in terms of the percentage of cache hits. In the current study, the hardware implementation of these policies is evaluated, and the possibility of adaptation to each of the policies in the processor core according to a selected priority characteristic is analyzed. The dependency of the rise in the delay and gate complexity in the case of an increase in the associativity of the cache memory is shown. The advantage of the hardware implementation of the PLRU$_t$ algorithm in comparison with the PLRU$_m$ algorithm for higher values of associativity is shown.

Keywords: digital automata; processor cores; cache memory; replacement policies; algorithms for data replacement; algorithms; LRU; PLRU; indicators of complexity; delays; reliability; adaptive algorithms

check for updates

Citation: Titarenko, L.; Kharchenko, V.; Puidenko, V.; Perepelitsyn, A.; Barkalov, A. Hardware-Based Implementation of Algorithms for Data Replacement in Cache Memory of Processor Cores. *Computers* **2024**, *13*, 166. https://doi.org/10.3390/computers13070166

Academic Editor: Paolo Bellavista

Received: 27 May 2024
Revised: 25 June 2024
Accepted: 2 July 2024
Published: 5 July 2024

1. Introduction

The cache memory of data or instructions is an essential component of the processor cores of computer systems. It is known that the loading of data from the cache that contains data or instructions into the processor core is faster than the loading from the corresponding segments or pages of the working memory of the computer system. It should be added that the event of a miss in the cache memory activates the corresponding policy of replacing the data or instructions. This is an effective algorithm for selecting an element for replacement. Among known replacement policies, the PLRU algorithm can be highlighted. It is implemented in the cache memory microarchitectures of i486 [1] and Pentium [2] processors. It is also implemented in Intel Core i7 processor cores of the following microarchitectures: Nehalem, Sandy Bridge, Ivy Bridge, Haswell, Broadwell, and

Skylake [3]. The synthesized circuits of the PLRU replacement policy, based on accepted research results, can be implemented with the use of FPGA.

1.1. Motivation

The main three parameters of the cache memory of the processor core are the informational size, the associativity (number of ways), and the size of the row element in bytes. The associativity (number of ways) of the cache memory of modern processor cores is 4, 8, and 16. But this is not the limit of this sequence, because there is a tendency to increase the size of the cache memory of modern processor cores that will lead to an increase in the number of ways of selecting data elements as replacement candidates. Also, this trend would affect the hardware characteristics of the PLRU policy circuits, which are divided into the policies $PLRU_t$ and $PLRU_m$. It should be noted that the characteristics of the circuits are the delay in the selection of a data element candidate for replacement, the hardware complexity measured in gates, and the reliability. However, most of the existing publications are not devoted to the study of these characteristics.

The description of the automata model and the further synthesis of the hardware implementation of the data replacement algorithm will allow us to determine the exact values of the mentioned characteristics and to investigate their dependence on the growth of associativity. It should also be added that the synthesized and researched $PLRU_t$ and $PLRU_m$ policy circuits provide an opportunity to conclude that there is adaptation of the cache memory of the processor core to each of the policies depending on the selected priority characteristic.

1.2. State of the Art

Replacement policies in the cache memory of the processor core are mainly characterized by the intensity of hits or misses. To understand the possible solutions, it is necessary to perform the analysis of known implementations and applications of the replacement policies.

The performed analysis of references shows that publications with this topic are focused on the following:

- The study of replacement policies during the caching of instructions and data in the cache memory of the CPU;
- The study of replacement policies during the caching in the device with FPGA;
- Research on the adaptive associativity of a reconfigurable cache scheme;
- The study of replacement policies in content-oriented computer networks;
- Research on data caching schemes for wireless networks;
- The study of the efficiency of different methods of caching of key-value storage for Blockchain in FPGA;
- The study of a cache-based matrix technique using the Hadoop Distributed File System (HDFS), where the EC volume consists of a Reed–Solomon code with parameters of (6, 3).

An analysis of existing solutions shows that there are FPGA-based implementations of configuration memory controllers with LRU and LFU replacement policies [4]. But the results of reconfiguration are known only for hit rates in sequences for cache sizes of 8, 16, and 32 blocks of a non-adaptive mechanism with the following policies: LRU, Random, and LFU.

Other VHDL implementations of LRU and PLRU policies with the use of two different methods for a reconfigurable cache memory show a 100% hit rate for an associative set of an eight-way cache memory [5].

The use of a multi-stage LRU cache replacement algorithm allows us to reduce the indirect costs during execution and to achieve greater accuracy both in the memory and in the CPU due to an increase in the ratio of cache hits compared to the LRU policy [6]. The investigation of the use of the PLRU policy with dependency on the parameters of the cache memory shows the reduction in the cache miss rate and instruction cache cycles [7].

Randomization greatly simplifies the integration of PLRU logic [8]. The performance of Rand-PLRU is comparable to that of the PLRU policy [9]. But there is no detailed investigation of the dependency of latency, complexity, and reliability for different parameters of the implementation of modifications of this policy.

The simulation of high-performance cache implementations in FPGA shows that the two-way multi-associative cache, $PLRU_{t,}$ is the best choice because it requires a smaller number of stored bits but offers the same performance. The considered options include the number of ways, lines, and words per line. The $PLRU_m$ policy shows the highest performance in all three cache levels, while the LRU policy gives the worst performance [10].

The PAC-PLRU replacement policy not only uses but also intelligently aggregates the prediction information. The use of PAC-PLRU with a prefetch block reduces the average L2 cache miss rate compared to an ordinary PLRU policy [11]. Most important is that PAC-PLRU requires only minor changes in the existing cache architecture to achieve these benefits. At the same, the simulation of the LRU policy shows the lowest cache miss rate for a private L1 data cache for the various benchmarks, regardless of the number of cores [12].

While the FSM model of the replacement logic of the PLRU policy of the four-way cache memory is well described [13], the synthesis of the hardware implementation with the estimation for a wide list of the parameter values of delay, complexity, and reliability is still an unsolved problem. But the values for conceptual models of the PLRU replacement policy for the cache memory of the processor core with four ways and eight ways are available for comparison [14]. Known variants of minimization of the PLRU algorithm for a q-way associative cache memory allow us to increase the characteristics of the speed and reliability of the logic circuit of the LRU block [15].

Implementations of the automata model of the pair of compatible adaptive algorithms with discrete functions and structural block diagrams are also known [16]. The results of the implementation of an adaptive associative reconfigurable cache scheme based on the decision tree in the GEM5 show in a simulation the dependency of the values of frequency and associativity for cache memories with a different number of ways [17].

The modifications of existing and new replacement policies find an application in network caching, like the policy Immature Used [18], and in wireless networks [19]. The simulation of the replacement policy taking into account the heterogeneity allows us to consider the improvement in the access speed and overall performance of asymmetric multicore systems with a separated Last Level cache memory of systems with different numbers of processor codes [20].

Research on data caching on the FPGA accelerator card from Xilinx programmed with Vitis 2020 is a possible way of experimentally investigating the performance of the implementation of replacement policies [21]. The use of FPGA also allows us to implement methods of caching for network purposes of key-value storage [22], as well as for distributed file systems [23]. But even for minimal hardware implementation of the PLRU algorithm, only a small number of ways are available for comparison [24].

Thus, a review of replacement policies shows the relevance of their study and use not only in the cache memory of computer systems with CPU and FPGA modules but also in network caching. Modern FPGAs are powerful enough for the implementation of the processor [25], with the possibility of evaluating the parameters [26]. The considered studies of replacement policies in the cache memory of computer systems are focused on the dependence of cache hit or miss ratios on the reconfiguration periods of adaptation, the associativity (number of ways), and cache memory volumes. Research on replacement policies in content-oriented computer networks focuses on performance evaluation (cache hit ratio, path length, delay, and link load). With regard to complexity evaluation studies, the complexity of $PLRU_t$ and $PLRU_m$ replacement policies is estimated as additional logic (LUT) in addressing lines in the cache and the total number of FF bits (PIM module) [7], and the cache circuit reconfiguration control logic is estimated [14]. Research on the delay during the search for a candidate element for replacement in a cache line, gate complexity, and reliability in a review of modern sources of studies on replacement policies also needs more clear and detailed explanation.

The summarized directions of the research on caching in each considered reference are presented in Table 1.

Table 1. A comparative analysis of the research directions in the considered studies.

The Direction of Research in the Reference	**Number of Considered References**						
	[6–9,11–16,20,24]	[4,5,10,21]	[17]	[18]	[19,27]	[22,25,26]	[23]
Study of replacement policies during the caching of instructions and data in the cache memory of the CPU	•						
Study of replacement policies during the caching in the device based on FPGA		•					
Research on the adaptive associativity of a reconfigurable cache scheme of the CPU			•				
Study of replacement policies in content-oriented computer networks				•			
Study scheme for wireless networks for growing data traffic and preventing overload					•		
Study of the efficiency of different methods of caching of key-value storage for Blockchain in FPGA						•	
Study of a cache-based matrix technique using the HDFS, where the EC volume consists of a Reed–Solomon code with the parameters of (6, 3)							•

1.3. Objectives and Structure

The goal of this article is to investigate and develop hardware solutions for implementing algorithms for data replacement in the cache memory of processor cores to improve the key characteristics of time delays, complexity, and reliability.

The main objectives are as follows:

- To analyze the structures of synchronous digital automata for further synthesis of the minimal hardware implementations of PLRU_t and PLRU_m replacement policies;
- To develop a minimal logic model for further research in a simulation environment;
- To evaluate the parameters of hardware implementations from the point of view of the delay of the transient process with the use of logical elements of NAND basis;
- To evaluate the complexity measured in logical elements and the reliability of hardware implementations;
- To analyze the possibility of the adaptation of the processor core cache memory to each of the PLRU_t and PLRU_m policies depending on the selected priority characteristic.

According to the research results, it will be possible to ascertain the possibility of adapting the cache memory of the processor core to each of the PLRU_t or PLRU_m policies depending on the selected priority characteristics, such as delay, gate complexity, and reliability. Further research on hardware solutions will contribute to the development of a method of adapting replacement policy mechanisms to change various algorithms depending on the parameters of the computing process.

The structure of this article is as follows: the methodology of the investigation is described in Section 2; Section 3 is dedicated to developing automata models and the technique of hardware implementation of PLRU_t and PLRU_m algorithms; the summarized results of the research on algorithm hardware implementation are presented in Section 4 for ways of q = 8, q = 16, and q = 32; Sections 5 and 6 discuss the results and conclude this study correspondingly.

2. Methodology and Stages of This Study

The research methodology consists of the following:

- A formal description of the structure of the synchronous digital automata followed by the synthesis of logical models of the circuits of $PLRU_t$ and $PLRU_m$ replacement algorithms, which are based on the theory of digital automata;
- The minimization of defined switching functions;
- An evaluation of gate complexity and the reliability of the circuits;
- The experimental implementation of the model in FPGA and an investigation of the parameters.

The methodology is implemented in the following stages:

Stage 1—a formal description of the structure of the synchronous digital automata implementation of the replacement algorithms $PLRU_t$ and $PLRU_m$.

Stage 2—the synthesis of logical models for these replacement algorithms.

Stage 3—the study of computer models for simulating automata and checking the correctness of the implementation of $PLRU_t$ and $PLRU_m$ replacement algorithms.

Stage 4—the study of the characteristics of the means of implementation of these replacement algorithms.

Stage 5—an analysis of the results of the comparison of the characteristics of means implementing the $PLRU_t$ and $PLRU_m$ algorithms and determining the possibility of the adaptation of the cache memory of the processor core by choosing one of the algorithms depending on the characteristics of the circuits.

3. Hardware Implementation of $PLRU_t$ and $PLRU_m$

The $PLRU_t$ and $PLRU_m$ replacement policies are variants of the PLRU replacement policy, which is close to the LRU policy in terms of cache hit percentage. The replacement policy $PLRU_t$ has the index t that indicates that the Least Recently Used element for replacement is determined based on the binary tree algorithm. The $PLRU_m$ replacement policy has the index m that indicates that the Least Recently Used element to replace is determined based on the age bit.

3.1. Automata Model of Algorithm *$PLRU_t$*

The automata model of the hardware implementation of the $PLRU_t$ replacement algorithm [15] (Figure 1) consists of a combination circuit of the CS_{dec}-way decoder and memory elements of the synchronous JK flip-flop type. The operation of the automata is described by the following transition and output functions ((1) and (2)):

$$Q_i^+ = f(\Phi_i(Q), \Psi_i(Q), H, R), \tag{1}$$

$$L_r = \Lambda_r(Q) \tag{2}$$

where $Q = \{Q_0, Q_1, \ldots, Q_{m'-1}\}$ is the set of the internal states of the automaton at the current moment of time t; $Q^+ = \left\{Q_0^+, Q_1^+, \ldots, Q_{m'-1}^+\right\}$ is the set of internal states of the automaton at the next moment t^+; $\Phi = \{\Phi_0(Q), \Phi_1(Q), \ldots, \Phi_{m'-1}(Q)\}$ is the set of simple switching functions for the excitation of the information inputs J of synchronous flip-flops; $\Psi = \{\Psi_0(Q), \Psi_1(Q), \ldots, \Psi_{m'-1}(Q)\}$ is the set of simple switching functions for the excitation of the information inputs K of synchronous flip-flops; $\Lambda = \{\Lambda_0(Q), \Lambda_1(Q), \ldots, \Lambda_{q-1}(Q)\}$ is the set of simple switching functions for the selection of q-ways in the set of lines with the data element L; $\Phi_i(Q)$, $\Psi_i(Q)$, $\Lambda_r(Q)$, Q_i, $Q_i^+ \in \{0, 1\}$; L_r represents the selected data item in the cache memory at the current direction r; $i \in \{0, 1, 2, \ldots, m' - 1\}$; $r \in \{0, 1, 2, \ldots, q - 1\}$; H is the signal of a cache hit/miss; R is the reset signal; m' is the number of memory elements of the synchronous JK flip-flop type; $q = 2^{m'}$ is the number of ways in the cache memory; $m', q \in \{N_0\}$.

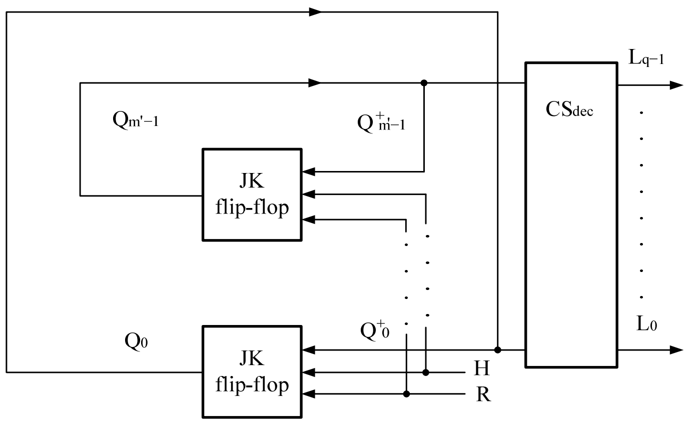


Figure 1. Automata model of hardware implementation of $PLRU_t$ algorithm.

3.2. *Synthesis of Hardware Implementation of $PLRU_t$*

The synthesized computer models of the $PLRU_t$ data replacement algorithm with q = 4 and q = 8 are described in [15] and [24]. These models provided the possibility to observe the corresponding sequences of changes in the q-indexes L_0 L_2 L_1 L_3 and L_0 L_4 L_2 L_6 L_1 L_5 L_3 L_7 of the data element L for replacement in the case of a cache hit event.

The observed sequence of changes in q-indexes allowed to synthesis of a general algorithm (Figure 2), changes in q-indexes, according to which it is possible to synthesize the computer models of the $PLRU_t$ data replacement algorithm with q = 16 and q = 32.

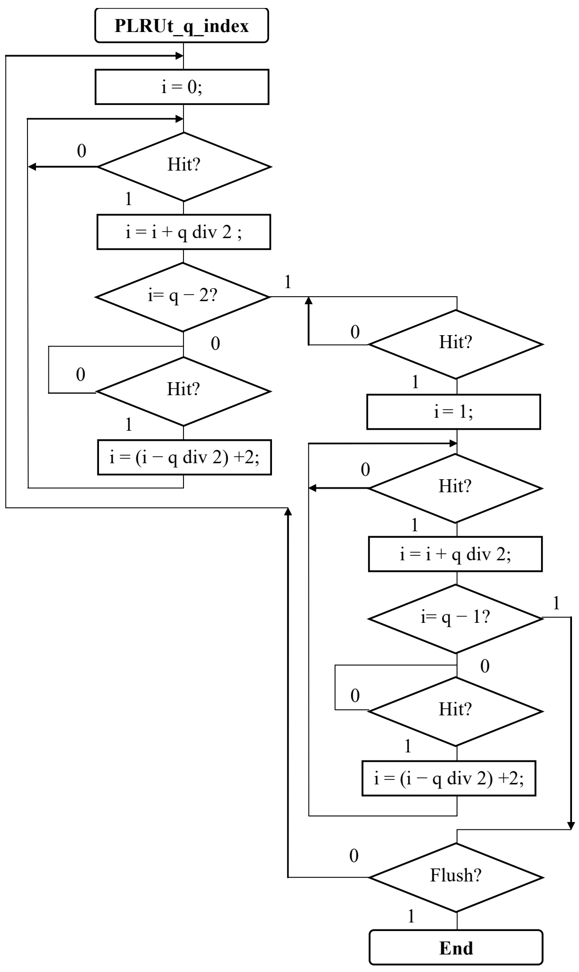
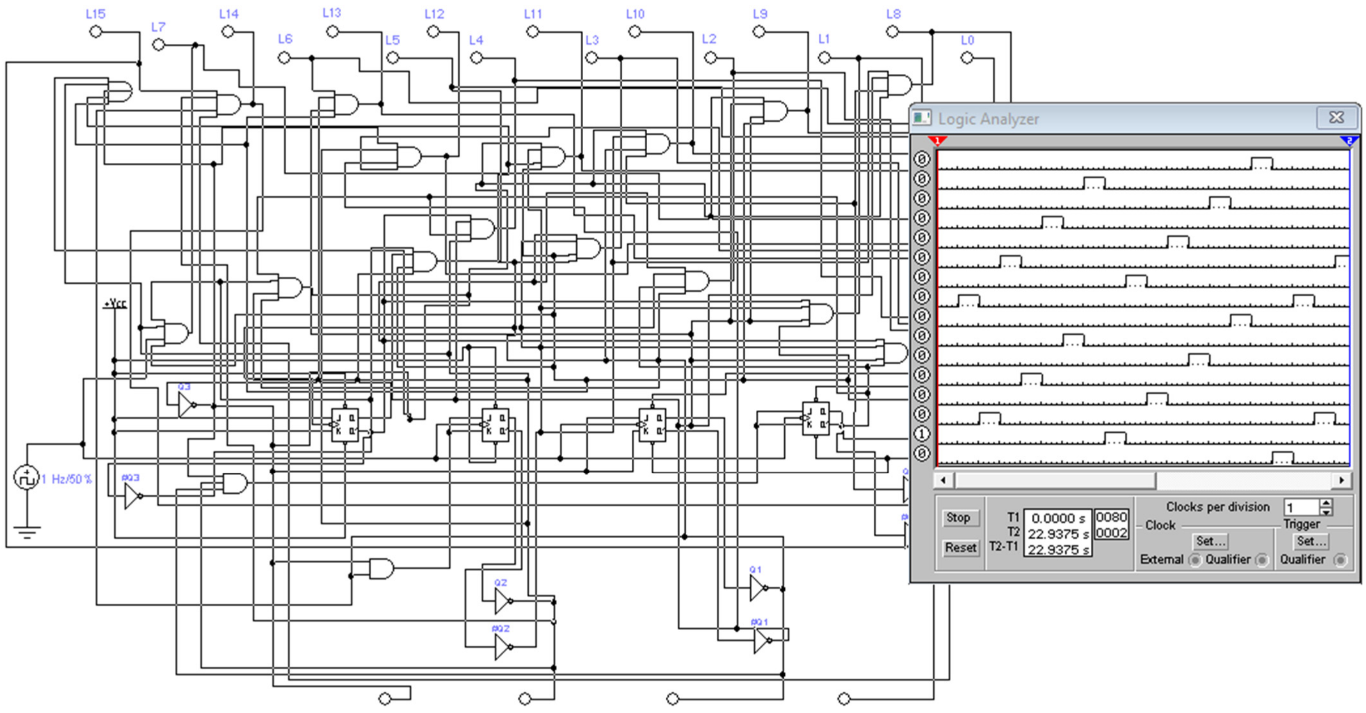


Figure 2. The general algorithm of changing the q-index according to the $PLRU_t$ algorithm.

Synthesized computer models of the PLRU$_t$ data replacement algorithm with q = 16 and q = 32 are described by logical equations, (3)–(6) and (7)–(11), where JK-type synchronous flip-flops are used as memory elements. The synthesized model is shown in Figure 3, taking into account the truth tables of the ways q of the decoders.



0.0000 s
22.9375 s
22.9375 s
0080
0002
Clocks per division 1

Figure 3. Computer model of hardware implementation of PLRU$_t$ algorithm for q = 16.

When studying computer models, the following sequences of q-index changes were observed:

q = 16: L_0 L_8 L_2 L_{10} L_4 L_{12} L_6 L_{14} L_1 L_9 L_3 L_{11} L_5 L_{13} L_7 L_{15}

q = 32: L_0 L_{16} L_2 L_{18} L_4 L_{20} L_6 L_{22} L_8 L_{24} L_{10} L_{26} L_{12} L_{28} L_{14} L_{30}
L_1 L_{17} L_3 L_{19} L_5 L_{21} L_7 L_{23} L_9 L_{25} L_{11} L_{27} L_{13} L_{29} L_{15} L_{31}

The observed corresponding sequences of changes in the q-index correspond to the general algorithm shown in Figure 2.

The logical equations of the computer model of the hardware implementation of the PLRU$_t$ algorithm for q = 16:

$$J_0 = K_0 = Q_3\&Q_2\&Q_1 \tag{3}$$

$$J_1 = K_1 = Q_3 \tag{4}$$

$$J_2 = K_2 = Q_3\&Q_1 \tag{5}$$

$$J_3 = K_3 = 1 \tag{6}$$

The logical equations of the computer model of the hardware implementation of the PLRU$_t$ algorithm for q = 32:

$$J_0 = K_0 = Q_4Q_3Q_2Q_1 = J_3\&Q_3 \tag{7}$$

$$J_1 = K_1 = Q_4 \tag{8}$$

$$J_2 = K_2 = Q_4\&Q_1 \tag{9}$$

$$J_3 = K_3 = Q_4\&Q_2\&Q_1 = J_2\&Q_2 \tag{10}$$

$$J_4 = K_4 = 1 \tag{11}$$

3.3. Research of Hardware Implementation for Algorithm PLRU$_t$

The synthesized computer models (including Figure 3) [15,24] allow us to create Table 2.

Table 2. The complexity of the hardware implementation of the PLRU$_t$ algorithm.

Num. of q	**Components**	**Number (Logical Elements per Line)**
q = 4	Logical elements AND-OR-NOT	4
q = 4	Logical elements NAND	4
q = 4	Logical elements AND	4
	Total (q = 4):	12
q = 8	Logical elements AND-OR-NOT	6
q = 8	Logical elements NAND	6
q = 8	Logical elements AND	9
	Total (q = 8):	21
q = 16	Logical elements AND-OR-NOT	8
q = 16	Logical elements NAND	8
q = 16	Logical elements AND	18
	Total (q = 16):	34
q = 32	Logical elements AND-OR-NOT	10
q = 32	Logical elements NAND	10
q = 32	Logical elements AND	35
	Total (q = 32):	55

The probability of fault-free operation is determined by Equation (12) and allows us to calculate the corresponding probabilities of fault-free operation:

$$P(t) = e^{-\lambda \times t}, \tag{12}$$

where e is a non-prime number; λ is the probability of failure-free operation of one gate or logical element; t is the working time before failure measured in hours.

The complexity values given in Table 2 and the calculated corresponding probabilities of failure-free operation allow us to create appropriate graphs of the dependence of the complexity of the hardware implementations of the PLRU$_t$ algorithm on the ways q = 4, q = 8, q = 16, and q = 32 (Figure 4) and the probabilities of failure-free operation in the time sections t = 100, t = 1000, t = 10,000, and t = 100,000 (Figure 5).

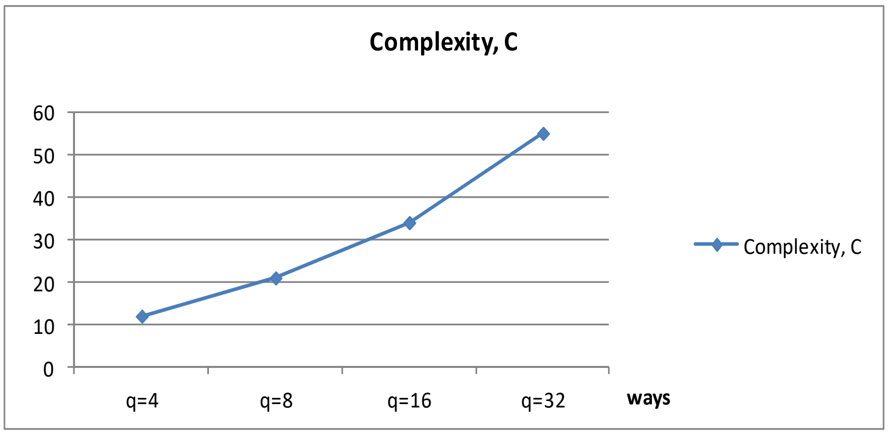
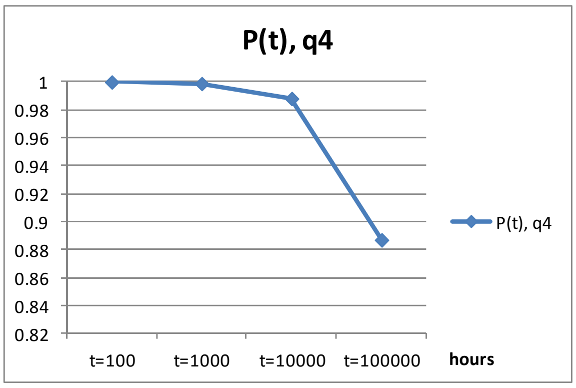
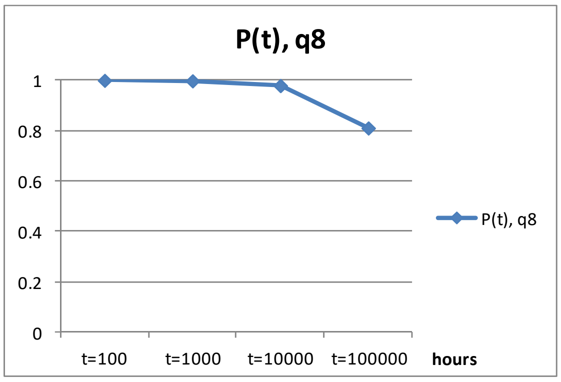


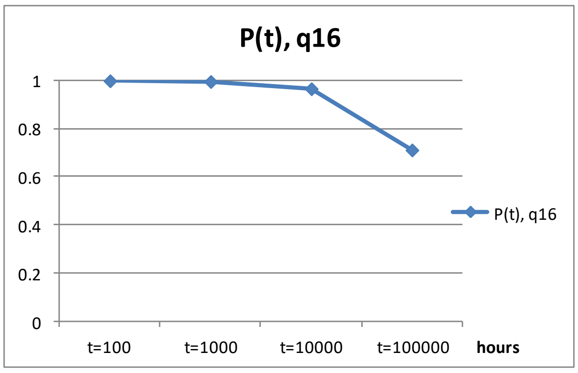
Figure 4. The dependence of the complexity of the PLRU$_t$ algorithm hardware implementations on the number of ways q in the associative cache memory of the processor.



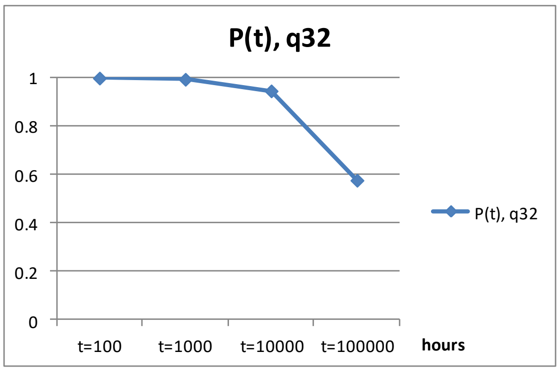
(**a**)



(**b**)



(**c**)



(**d**)

Figure 5. Probabilities of trouble-free operation of hardware implementations of $PLRU_t$ algorithm (**a**) for q = 4; (**b**) q = 8; (**c**) q = 16; (**d**) q = 32.

The total delay in the transient processes of hardware implementations of $PLRU_t$ and $PLRU_m$ algorithms with the way q = 4 is determined by Expression (13):

$$\tau_{total} = \tau_{mem} + \tau_{dec}, \tag{13}$$

where τ is the delay in the transient process of the logical element NAND; τ_{mem} is the delay in the transients of the synchronous flip-flop of JK type; τ_{dec} is the delays in the transient processes of the combinational logic of the CS_{dec}-way decoder; τ_{total} is the total delay in the transient processes of the hardware implementation of $PLRU_t$ and $PLRU_m$ algorithms with ways q.

The value of the complexity of these hardware implementations can be calculated using Formulas (14)–(16), including for the hardware implementations of the $PLRU_t$ and $PLRU_m$ algorithms:

$$C_{total} = C_{mem} + C_{dec} + C_{add\_log}, \tag{14}$$

where C_{mem} is the complexity measured in the logical elements or gates of the memory elements; C_{dec} is the complexity measured in the logical elements or gates of the way decoder; C_{add_log} is the synthesized or predicted additional gate complexity.

$$C_{mem} = N \times k, \tag{15}$$

where N is the number of memory elements; k is the complexity measured in the logical elements or the gates for one memory element (k = 4 for a memory element implemented based on the synchronous flip-flop of JK type).

$$C_{dec} = q \times v, \tag{16}$$

where q is the number of ways of the cache memory; v is the complexity measured in the logical elements or the gates for the q_i-way decoder.

The partial complexity v measured in the logical elements or the gates for the way decoder q_i is provided in Table 3. The columns v_{2and}, v_{3and}, and v_{4and} are the logic elements of AND with 2, 3, and 4 inputs, respectively.

Table 3. The complexity v measured in the logical elements or the gates for the way decoder q.

Num. of Ways, q	**Lines, N**	v_{2and}	v_{3and}	v_{4and}	v	τ_{dec}
4	2	1	0	0	1	τ
8	3	0	1	0	1	τ
16	4	0	0	1	1	τ
32	5	1	0	1	2	2τ
64	6	2	0	1	3	2τ
128	7	1	1	1	3	2τ
256	8	1	0	2	3	2τ
512	9	0	1	2	3	2τ
1024	10	1	1	2	4	2τ
2048	11	0	2	2	4	2τ

The value of additional gate complexity C_{add_log} is provided in Table 4. The synthesis of hardware implementations of the $PLRU_t$ algorithm for q = 4, q = 8, q = 16, and q = 32 shows that when the number of ways is doubled, the value of the additional gate complexity C_{add_log} increases by 1. Therefore, it allows us to predict the additional complexity measured in logical elements or the gates for $q \geq 64$ (Table 4).

Table 4. The complexity of the hardware implementations of the $PLRU_t$ algorithm for q = 4, q = 8, q = 16, q = 32, q = 64, q = 128, q = 256, q = 512, q = 1024, and q = 2048.

Num. of Ways, q	**Total Complexity in Gates per Line, C_{PLRUt}**	**Complexity of Memory Elements**		**Complexity of Decoder of Ways of Memory**		**Synthesized and Predicted Additional Gate Complexity**	
		C_{mem}	τ_{mem}	C_{dec}	τ_{dec}	C_{add_log}	C_{PLRUt} (%)
4	12	8	3τ	4	τ	0	0.00
8	21	12	3τ	8	τ	1	4.76
16	34	16	3τ	16	τ	2	5.88
32	55	20	3τ	64	2τ	3	5.46
64	220	24	3τ	192	2τ	4	1.82
128	417	28	3τ	384	2τ	5	1.20
256	806	32	3τ	768	2τ	6	0.74
512	1579	36	3τ	1536	2τ	7	0.05
1024	4144	40	3τ	4096	2τ	8	0.19
2048	8245	44	3τ	8192	2τ	9	0.11

Thus, a more complete picture of the dependence of the complexity of hardware implementations of the $PLRU_t$ algorithm on the number of ways q in the associative cache memory of the processor allows us to represent it in the form of a curve (Figure 6).

Thus, it is possible to reproduce the indicators of delay, complexity, and reliability of circuits of the $PLRU_t$ algorithm's implementation for the range from q = 4 to q = 2048 (Table 5).

However, it is necessary to take into account that the total delay in the hardware implementation of the $PLRU_t$ algorithm consists of the sum of two components, the delay of the flip-flop of JK type and the delay of the way decoder, that are provided in Table 4.

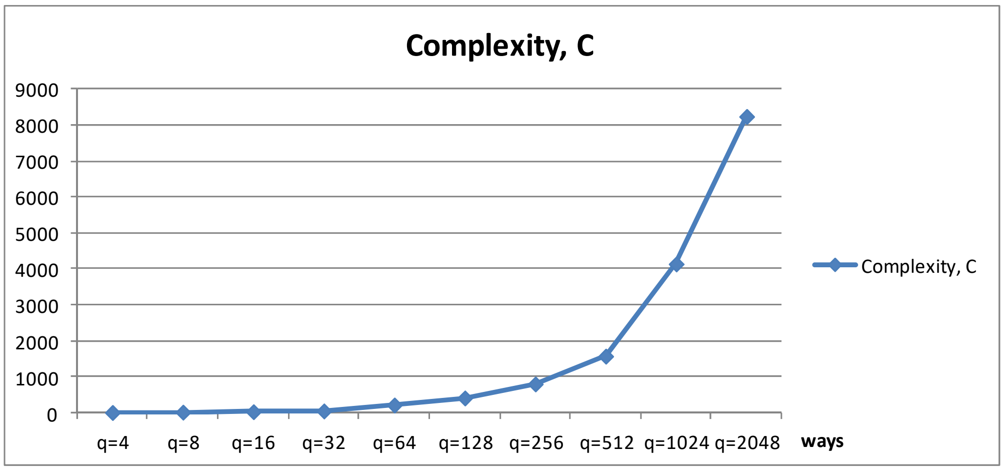


Figure 6. The extended dependence of the complexity of hardware implementations of the PLRU$_t$ algorithm on the number of ways q in the multi-associative cache memory of the processor core.

Table 5. The indicators of delay, complexity, and reliability of hardware implementations of the PLRU$_t$ algorithm for the range of $4 \leq q \leq 2048$.

Num. of Ways, q	**Complexity in Gates per Line, C_{PLRUt}**	**Reliability, P (t = 10,000)**	**Delay, τ**
4	12	0.9981	4τ
8	21	0.9792	4τ
16	34	0.9665	4τ
32	55	0.9465	5τ
64	220	0.8025	5τ
128	417	0.6590	5τ
256	806	0.4467	5τ
512	1579	0.2062	5τ
1024	4144	0.0159	5τ
2048	8245	0.0003	5τ

Table 5 allows us to represent the dependence of probabilities of failure-free operation of implementations of the PLRU$_t$ algorithm for the range from q = 4 to q = 2048 (Figure 7). The way selection delay in Figure 8 shows the step-like rise between q = 16 and q = 32.

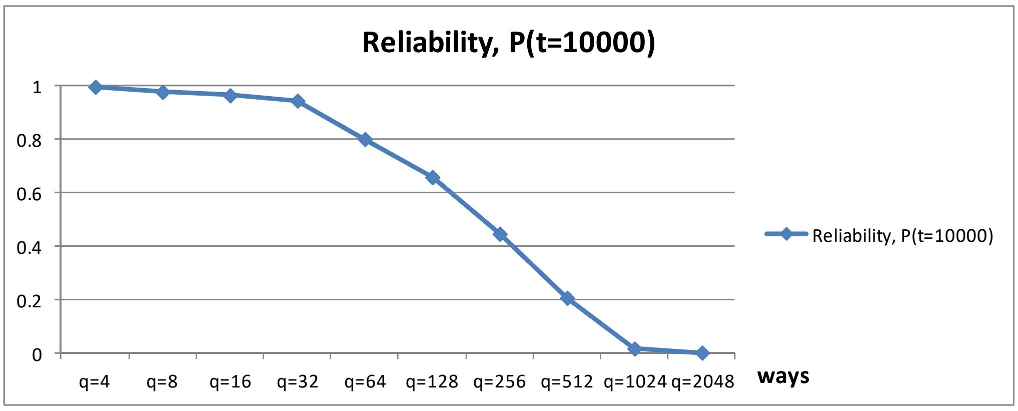


Figure 7. The probabilities of failure-free operation of the hardware implementation of the PLRU$_t$ algorithm for q = 4, q = 8, q = 16, q = 32, q = 64, q = 128, q = 256, q = 512, q = 1024, and q = 2048.

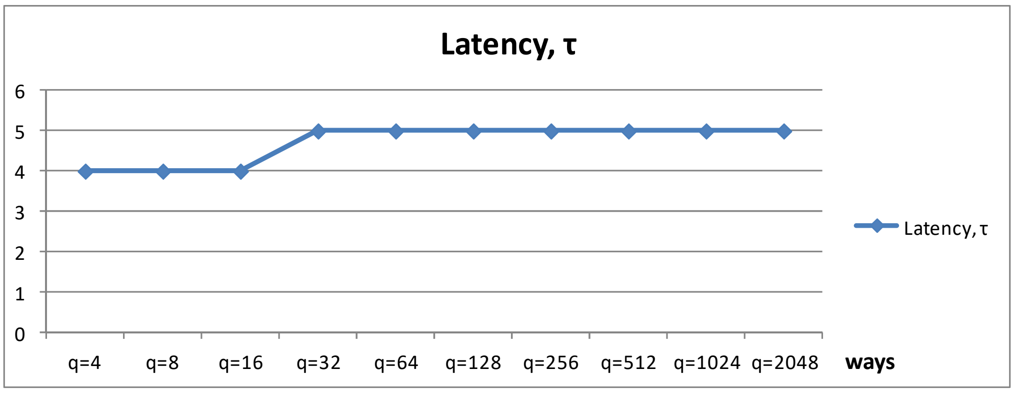


Figure 8. The delays in way selection in the hardware implementation of the PLRU$_t$ algorithm for q = 4, q = 8, q = 16, q = 32, q = 64, q = 128, q = 256, q = 512, q = 1024, and q = 2048.

3.4. Automata Model of Hardware Implementation of PLRU$_m$ Algorithm

The automata model of the hardware implementation of the PLRU$_m$ replacement algorithm (Figure 9) consists of a combinational circuit of the CS$_{dec}$-way decoder and the memory elements of the synchronous flip-flop of JK type. The operation of the automaton is described by the following transition and output functions ((17) and (18)):

$$A_i^+ = f(\Phi_i(A), \Psi_i(A), H, R), \tag{17}$$

$$L_r = \Lambda_r(A), \tag{18}$$

where $A = \{A_0, A_1, \ldots, A_{q-1}\}$ is the set of internal automata states at the current moment of time t; $A^+ = \{A_0^+, A_1^+, \ldots, A_{q-1}^+\}$ is the set of internal states at the next moment t$^+$; $\Phi = \{\Phi_0(A), \Phi_1(A), \ldots, \Phi_{q-1}(A)\}$ is a set of simple switching functions for the excitation of information inputs J of the synchronous flip-flops; $\Psi = \{\Psi_0(A), \Psi_1(A), \ldots, \Psi_{q-1}(A)\}$ is a set of simple switching functions for the excitation of information inputs K of flip-flops; $\Lambda = \{\Lambda_0(A), \Lambda_1(A), \ldots, \Lambda_{q-1}(A)\}$ is a set of q-way selection for simple switching functions with elements of the recently used data L; $\Phi_i(A), \Psi_i(A), \Lambda_r(A), A_i, A_i^+ \in \{0, 1\}$; L$_r$ is the selected data element in the cache memory by the current way r; $i \in \{0, 1, 2, \ldots, q-1\}$; $r \in \{0, 1, 2, \ldots, q-1\}$; q is the number of memory elements; H is a cache hit/miss signal; R is a reset signal; q is the number of ways in the cache memory; $q \in \{N_0\}$.

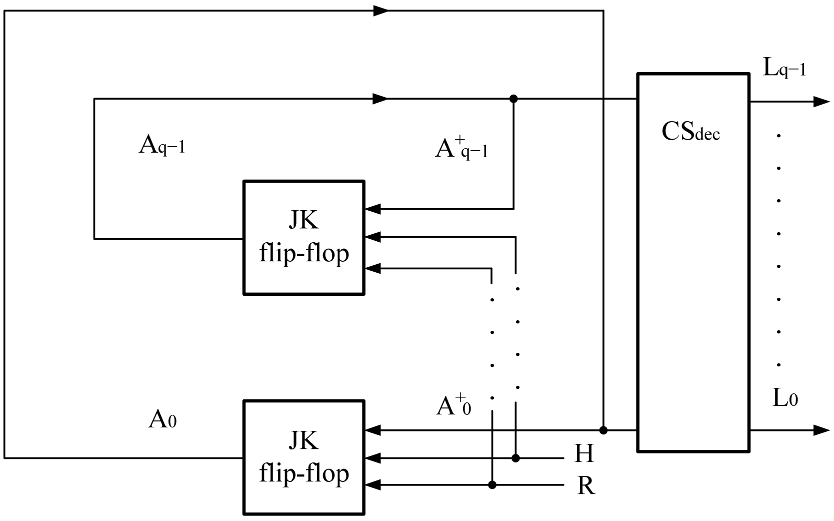


Figure 9. The automata model of the hardware implementation of the PLRU$_m$ algorithm.

3.5. Synthesis of Hardware Implementation of PLRU$_m$ Algorithm with Ways of q = 4

The synthesis of the hardware implementation is based on the idea of using the FIFO queue (Figure 10) to fill by using a logical value of 1 the additional bits of A_i in the case of an event of a cache hit (H = 1) (Figure 10). In the case of an event of a cache miss, the hardware implementation should automatically choose the way i + 1, according to which the recently used data element L with the corresponding bit A_{i+1} = 0 will be selected.

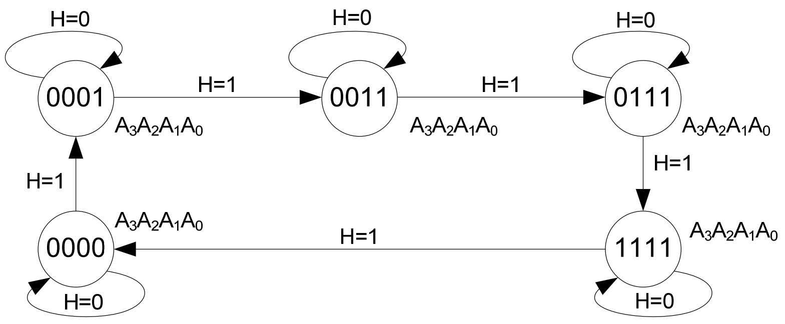


Figure 10. The directed graph of transitions of filling by a logical value of 1 the additional bits of A_i at the cache hit event (H = 1) for q = 4.

The computer model of the hardware implementation of the PLRU$_m$ algorithm with ways q = 4 is provided in Figure 11.

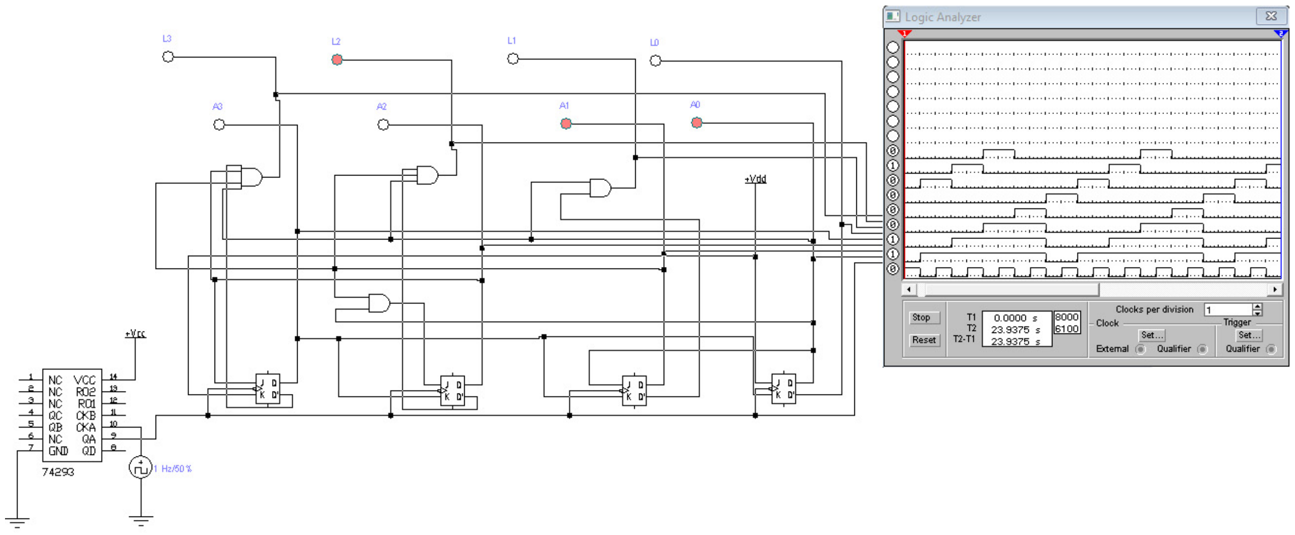


Figure 11. The computer model of the hardware implementation of the PLRU$_m$ algorithm for q = 4.

This model of the hardware implementation of the PLRU$_m$ algorithm with the parameter of ways q = 4 is represented by the following logical Equations (19)–(24):

$$J_0 = K_3 = 1, K_0 = K_1 = K_2 = A_3, \tag{19}$$

$$J_1 = A_0, \tag{20}$$

$$J_2 = A_1 \& A_0, \tag{21}$$

$$J_3 = A_2, \tag{22}$$

$$L_0 = \overline{A_0}, \tag{23}$$

$$L_{n+1} = \overline{A_{n+1}} \bigwedge_{i=0}^{n} A_i, \tag{24}$$

The study of the computer model of the PLRU$_m$ algorithm for q = 4 shows the compliance of their work with the theoretical expectations determined by the idea of the PLRU$_m$ replacement algorithm.

3.6. Research of Hardware Implementation of PLRU$_m$ Algorithm with Ways q = 4

3.6.1. Evaluation of Latency and Complexity

According to Expression (13), in the case of an event of a cache hit (H = 1), the total delay in the hardware implementation of the PLRU$_m$ algorithm with ways of q = 4 will be as follows:

$$\tau_{\text{total}} = 3\tau + \tau = 4\tau$$

Figure 12 shows the functional circuit of a JK flip-flop with static synchronization with a gate complexity of k = 4.

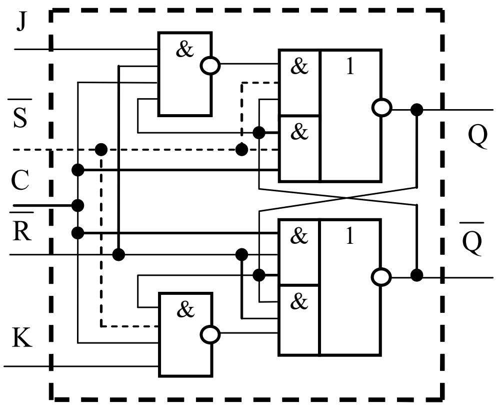


Figure 12. A functional diagram of a JK flip-flop with static synchronization with a complexity of k = 4.

Expressions (14)–(16) allow us to summarize the gate complexity of the hardware implementation of the PLRU$_m$ algorithm for a number of ways of q = 4 (Table 6).

Table 6. The gate complexity of the hardware implementation of the PLRU$_m$ algorithm with the parameter of ways of q = 4.

Num. of q	Symbol	Components	Number of Gates (V) per Line
q = 4	C_{mem}	Logical elements NAND with 3 inputs	8
q = 4		Logical elements AND with 4 inputs	8
q = 4	C_{add_log}	Logical elements AND with 2 inputs	1
q = 4	C_{dec}	Logical elements AND with 2 inputs	1
q = 4		Logical elements AND with 3 inputs	1
q = 4		Logical elements AND with 4 inputs	1
	C_{total} (q = 4)		V = 20

3.6.2. Evaluation of Reliability

Based on Expression (12) and taking into account Table 6, it is possible to represent the evaluation of the fault-free operation of the PLRU$_m$ algorithm hardware implementation with ways q = 4 in the form of a graph (Figure 13).

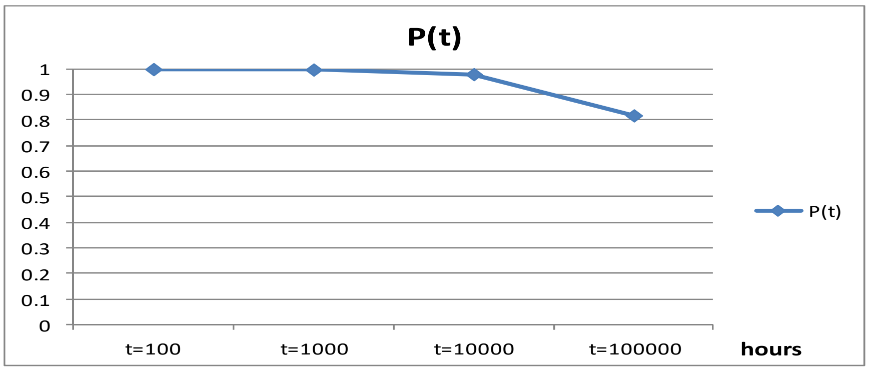


Figure 13. A graph of the failure-free operation of the hardware implementation of the PLRU$_m$ algorithm with ways of q = 4.

4. Summarized Results of Study of Hardware Implementation of PLRU$_m$ Algorithm with Ways of q = 8, q = 16, and q = 32

The synthesis of hardware implementations of the PLRU$_m$ algorithm with ways of q = 8, q = 16, and q = 32 is based on the synthesis of the hardware implementation of the PLRU$_m$ algorithm with ways of q = 4.

The computer model of the hardware implementation of the PLRU$_m$ algorithm with ways of q = 8 (Figure 14) is based on the following logical Equations (23)–(26):

$$J_0 = K_7 = 1, K_0 = K_1 = K_2 = K_3 = K_4 = K_5 = K_6 = A_7, \quad (25)$$

$$J_{m+1} = A_m, \quad (26)$$

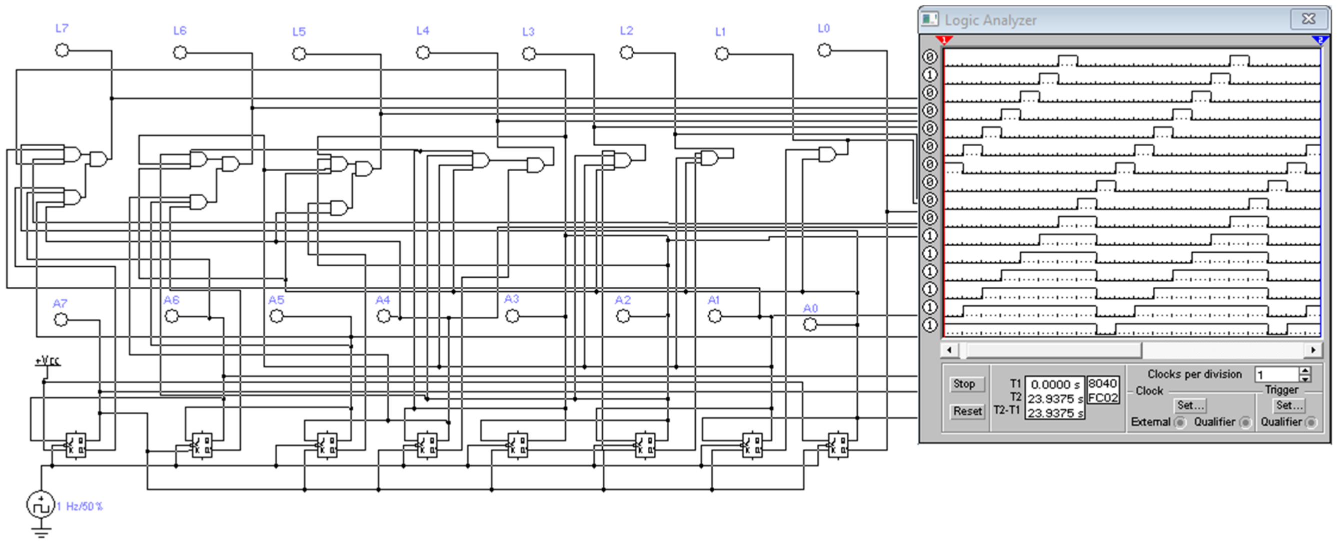


Figure 14. The computer model of the hardware implementation of the PLRU$_m$ algorithm for q = 8.

During the study of the computer models of the PLRU$_m$ algorithm for q = 8 and q = 16, the compliance of the results with the theoretical expectations determined by the idea of the PLRU$_m$ replacement algorithm is observed.

4.1. *Research of Hardware Implementations of PLRU$_m$ Algorithm with Ways of q = 8, q = 16, and q = 32*

4.1.1. Evaluation of Complexity and Reliability

Based on Expressions (14)–(16), it is possible to summarize the complexity measured in logical elements or the gates of hardware implementations of the PLRU$_m$ algorithm with ways of q = 8 and q = 16 (Table 7).

Table 7. The gate complexity of the hardware version of the $PLRU_m$ algorithm with ways of q = 8 and q = 16.

Num. of q	Symbol	Components	Number of Gates (V) per Line
q = 8	C_{mem}	Logical elements NAND with 3 inputs	16
q = 8		Logical elements AND with 4 inputs	16
q = 8	C_{dec}	Logical elements AND with 2 inputs	6
q = 8		Logical elements AND with 3 inputs	2
q = 8		Logical elements AND with 4 inputs	3
	C_{total} (q = 8)		V = 43
q = 16	C_{mem}	Logical elements NAND with 3 inputs	32
q = 16		Logical elements AND with 4 inputs	32
q = 16	C_{dec}	Logical elements AND with 2 inputs	8
q = 16		Logical elements AND with 3 inputs	8
q = 16		Logical elements AND with 4 inputs	11
	C_{total} (q = 16)		V = 91

The computer model of the hardware implementation of the $PLRU_m$ algorithm with ways of q = 16 allows us to perform an accurate estimation of the complexity of the CS_{dec} combinational circuit and hardware implementation of the $PLRU_m$ algorithm with ways of q = 32 without the process of synthesis. It should be noted that at the same time, the complexity of memory elements C_{mem} will increase by exactly two times compared to the hardware implementation of the $PLRU_m$ algorithm with ways of q = 16.

With the use of Expressions (14)–(16), it is possible to summarize the gate complexity of the hardware implementation of the $PLRU_m$ algorithm with ways of q = 32 (Table 8).

Table 8. The gate complexity of the hardware implementation of the $PLRU_m$ algorithm with the parameter of ways of q = 32.

Num. of q	Symbol	Components	Number of Gates (V) per Line
q = 32	C_{mem}	Logical elements NAND with 3 inputs	64
q = 32		Logical elements AND with 4 inputs	64
q = 32	C_{dec}	Logical elements AND with 2 inputs	13
q = 32		Logical elements AND with 3 inputs	18
q = 32		Logical elements AND with 4 inputs	44
	C_{total} (q = 32)		V = 203

4.1.2. Evaluation of Delay

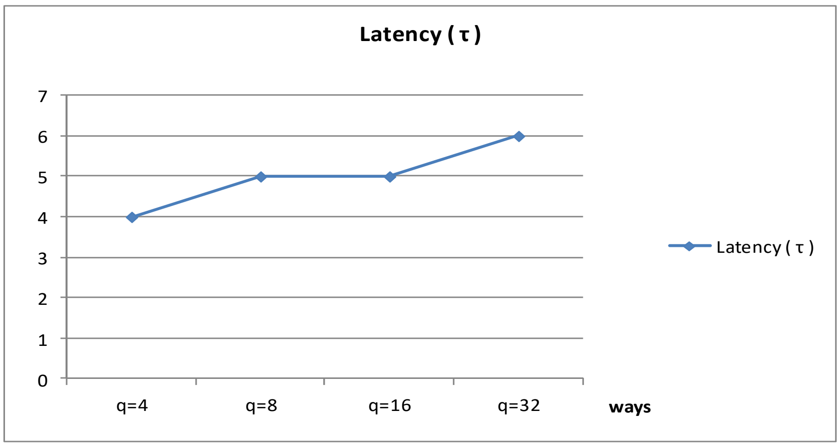
Based on Expression (13), in the case of an event of a cache hit (H = 1), the total delay in $PLRU_m$ hardware implementation with ways of q = 8, q = 16, and q = 32 will be as follows:

$$\tau_{total\ q=8} = 3\tau + 2\tau = 5\tau$$

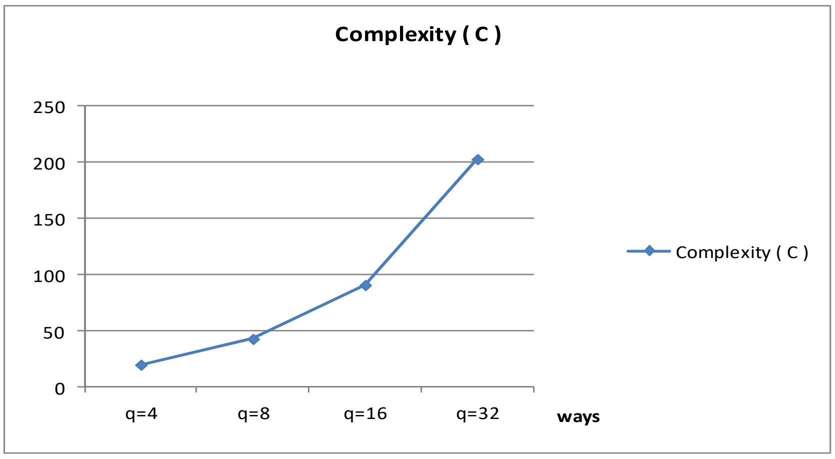
$$\tau_{total\ q=16} = 3\tau + 2\tau = 5\tau$$

$$\tau_{total\ q=32} = 3\tau + 3\tau = 6\tau$$

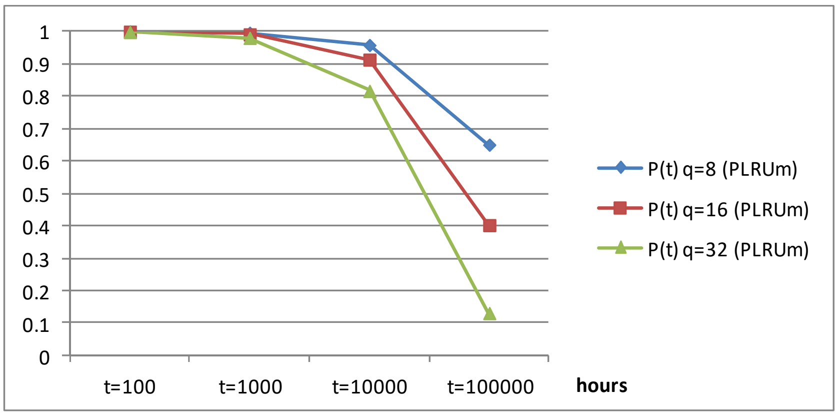
Based on Tables 6–8, it is possible to estimate the complexity and reliability of the hardware implementations of the $PLRU_m$ algorithm with ways of q = 4, q = 8, q = 16, and q = 32 and represent them in the form of the graphs (Figure 15).



(**a**)



(**b**)



(**c**)

Figure 15. Graphs of the delay, complexity, and reliability of hardware implementations of the $PLRU_m$ algorithm for ways of q = 4, q = 8, q = 16, and q = 32: (**a**) delays; (**b**) complexity; (**c**) reliability.

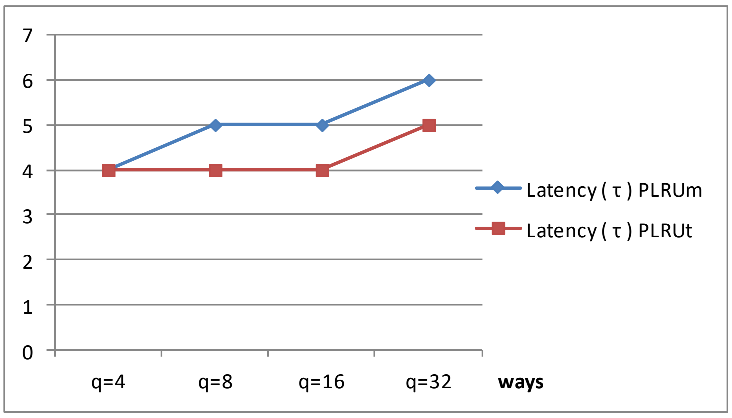
4.2. Possible Limitations and Challenges of Hardware Implementation of Both Algorithms

The current study is primarily targeted toward performing the investigation of parameters of the hardware implementation of both PLRUt and $PLRU_m$ algorithms. This understanding allows us to estimate the characteristics for different values of ways of parameter q without physical implementation and to determinate the dependency on this q parameter.

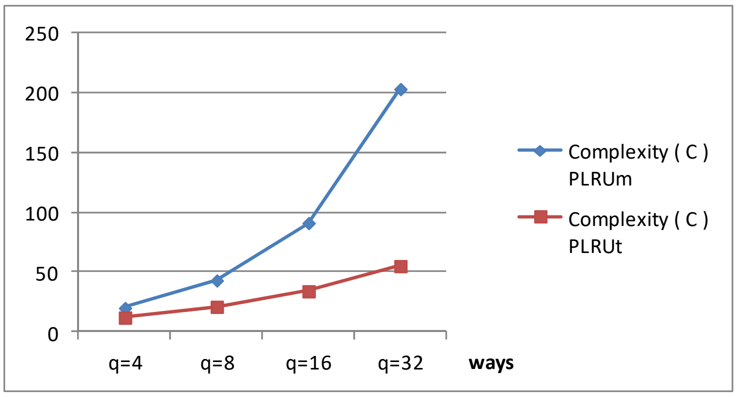
The performed investigation allows us to highlight the possible limitations or problems during the implementation of the hardware of the $PLRU_t$ algorithm. In this study, the investigation of probabilities of failure-free operation of the hardware implementation of the PLRU$_t$ algorithm is shown in Figure 7. It illustrates the probabilities of failure-free operation of the hardware implementation of the PLRUt algorithm for q = 4, q = 8, q = 16, q = 32, q = 64, q = 128, q = 256, q = 512, q = 1024, and q = 2048. Here, it can be seen that the reliability of the hardware implementation of the algorithm for q = 64, q = 128, q = 256, q = 512, q = 1024, and q = 2048 rapidly decreases from 0.8 to almost 0.

This is also valid for the hardware implementation of the PLRU$_m$ algorithm. The investigation results for this algorithm are provided in Figure 15.

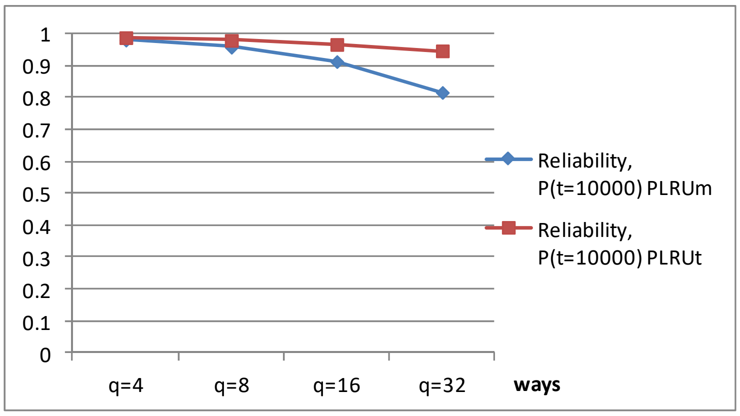
Therefore, the implementation of the hardware of these algorithms is reasonable for the parameters of ways of q = 4, q = 8, q = 16, and q = 32 (Figure 16).



(**a**)



(**b**)



(**c**)

Figure 16. Graphs of the comparison of the obtained delay, complexity, and reliability of the hardware implementations of the PLRU$_m$ and PLRU$_t$ algorithms for parameters of ways of q = 4, q = 8, q = 16, and q = 32: (**a**) delays; (**b**) complexity; (**c**) reliability.

5. Discussion

5.1. Comparative Analysis

The results of the performed investigation of the synthesized hardware implementations of the $PLRU_t$ and $PLRU_m$ algorithms of one cache memory line and the results of Table 3 of the replacement policy resource utilization from [10] are summarized in Table 9.

Table 9. The results of the performed investigation into the replacement policy resources.

Policy	**Ways**	**Single Cache Line**	**LUT**	**LUT (Table 3) [10]**	**FF**	**FF (Table 3) [10]**
$PLRU_m$	4	1	8	8	4	4
$PLRU_t$	4	1	6	6	2	3
$PLRU_m$	8	1	16	22	8	8
$PLRU_t$	8	1	11	13	3	7
$PLRU_m$	16	1	32	–	16	–
$PLRU_t$	16	1	20	–	4	–
$PLRU_m$	32	1	64	–	32	–
$PLRU_t$	32	1	37	–	5	–

The results of the performed investigation in Table 9 are provided for ways of $q = 4$ and $q = 8$, where a smaller number of LUTs and a smaller or equal number of FFs are observed for both algorithms compared to the results of Table 3. The replacement policy resources [10] or research data for ways of $q = 16$ and $q = 32$ are not available.

The proposed method of synthesizing the hardware implementation of the $PLRU_t$ algorithm allows us to obtain results for a higher number of ways (Table 10).

Table 10. The results of the performed investigation into the replacement policy resources for $64 \leq q \leq 2048$.

Policy	**Ways**	**Single Cache Line**	**LUT**	**LUT (Table 3) [10]**	**FF**	**FF (Table 3) [10]**
$PLRU_t$	64	1	70	–	6	–
$PLRU_t$	128	1	135	–	7	–
$PLRU_t$	256	1	264	–	8	–
$PLRU_t$	512	1	521	–	9	–
$PLRU_t$	1024	1	1034	–	10	–
$PLRU_t$	2048	1	2059	–	11	–

The universal form of the synthesized hardware implementations of the $PLRU_t$ algorithm for $q = 4$, $q = 8$, $q = 16$, and $q = 32$ allows us to determine the values of the delay, gate complexity, and reliability for the entire range of $4 \leq q \leq 2048$.

5.2. Adaptation

One of the goals of this study is to determine the possibility of the adaptation [16] of the cache memory of the processor core to each of the $PLRU_t$ or $PLRU_m$ policies depending on one of the following priority characteristics selected in the core: delay, complexity, or reliability. The characteristic values summarized in Table 11 provide an opportunity to build comparative graphs for each of the characteristics (Figure 16).

Based on the comparison of the hardware implementation of the algorithms provided in Figure 16, it is possible to conclude that the $PLRU_t$ algorithm is better than the hardware implementation of the $PLRU_m$ algorithm in terms of all characteristics. It makes the adaptation impossible, but based on the advantages, it increases the probability of the use of the hardware implementation of this algorithm for the cache memory of the CPU or implementation in FPGA.

Table 11. The values of the characteristics of gate complexity, reliability, and delay in hardware implementations of $PLRU_m$ and $PLRU_t$ algorithms with ways of q = 4, q = 8, q = 16, and q = 32.

Num. of Ways, q	Gate Complexity, C_{PLRUt}	Gate Complexity, C_{PLRUm}	Reliability, P (t = 10,000) $PLRU_t$	Reliability, P (t = 10,000) $PLRU_m$	Delay, τ $PLRU_t$	Delay, τ $PLRU_m$
4	12	20	0.9981	0.9802	4τ	4τ
8	21	43	0.9792	0.9579	4τ	5τ
16	34	91	0.9665	0.9130	4τ	5τ
32	55	203	0.9465	0.8163	5τ	6τ

5.3. Further Development

The obtained results allow us to make improvements to the hardware implementation of the algorithms of the replacement policy. The fast verification of the workability of such algorithms can be performed with the use of FPGA to create the hardware implementation of the required circuit.

The results are also promising for the improvement in the architectures of the existing processors and for prototyping the superscalar processor models. Modern FPGA chips allow for the implementation of the super-parallelized scalable and parameterizable dedicated data processing systems for intensive searches of data in dynamic HBM memory and the implementation of services with dedicated architectures [25].

The creation of a dedicated architecture before the production of the chip allows us to perform practical measurements to compare the results with the results of theoretical research [26].

The possibility of the implementation not only of the logic of a cache memory replacement policy but also of the queue of the commands for execution allows us to perform the next step of improving the performance of the processors while ensuring backward comparability with existing software for the processors. This step allows us to reduce the performance by means of out-of-order execution of commands without data dependency.

The practical experience of the creation of the dedicated architecture with powerful FPGA chips and the development flow with the software for prototyping allows us to apply the same approach for the creation of hardware models of such algorithms for proving the concepts and quickly checking the optimizations. It is possible to do this because at the moment, most FPGAs are produced by the same companies that are involved in the production of modern processors. The same technological process is used during the production. At the same time, the total performance of some FPGAs is higher than that of CPUs [27].

Thus, the finding of a new model of prediction and of candidates for replacement in the cache memory, and the creation of a queue of commons for execution are the most promising directions for the next steps of future research. Involving modern developed environments for FPGA project prototyping allows us to reduce the time taken for the implementation and testing of such theoretical models. Combining both theoretical research with practical experiments is the best way to enhance processor architectures with a prototyping process with low costs and efforts.

6. Conclusions

The main contribution of this study lies in developing a method of hardware implementation of algorithms for data replacement in the cache memory of processor cores, as well as the accessible assessments of complexity, latency, and reliability that allow us to obtain the proof of concept of the adaptive solutions.

This paper proposes the description of automata models and the results of the synthesis of hardware solutions of $PLRU_t$ and $PLRU_m$ data replacement algorithms. It allows us to determine the parameters and values of characteristics, such as the delay in the selection of a data element for a candidate for replacement, the complexity measured in logical elements, and the reliability of the equipment, as well as to investigate their dependence on the growth of the associativity. With the growth of the associativity of the cache memory, both the delay parameters and the complexity measured in logical elements increase.

Implementations of both algorithms are compared and considered in detail. The hardware implementation of the data replacement algorithm $PLRU_t$ has advantages in terms of values of the delay over the hardware implementation of the data replacement algorithm $PLRU_m$ as well as by one τ in the case of the associativity of 8, 16, and 32.

However, the implementation of the algorithms is equal at q = 4. According to the parameters of complexity and reliability, the hardware implementation of the $PLRU_t$ data replacement algorithm shows better results than the hardware implementation of the $PLRU_m$ data replacement algorithm at q = 4, 8, 16, and 32. This advantage increases the probability of using the implementation of this algorithm in the CPU cache memory module or for FPGA implementation.

Further research can be focused on adding and enhancing the set of algorithms for different replacement policies to determine their priority and selection based on preferred parameters. The possibility of rapid prototyping and testing with the use of FPGA helps to prove the workability of the algorithm. In addition, the development of a method of adapting the mechanisms of replacement policies for changing various algorithms depending on the parameters of the computing process is very promising.

Author Contributions: Conceptualization, A.B., V.K., and V.P.; methodology, V.K., V.P., and A.P.; software, V.P.; validation, L.T. and A.P.; formal analysis, A.B. and V.K.; investigation, V.P. and A.P.; resources, L.T.; data curation, V.P. and A.P.; writing—original draft preparation, V.P., A.P. and V.K.; writing—review and editing, L.T. and A.B.; visualization, V.P. and A.P.; supervision, V.K.; project administration, L.T., A.B. and V.K. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded by the Ministry of Education and Science of Ukraine, state grant registration number 0122U001065, project title "Methods, software and hardware tools, and technologies for providing dependability of intelligent industrial Internet of Things". This publication reflects the views only of the authors, and the Ministry of Education and Science of Ukraine cannot be held responsible for any use that may be made of the information contained therein.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Acknowledgments: The authors are grateful to the scientific staff of the Department of Computer Systems, Networks and Cybersecurity of the National Aerospace University "KhAI" and the Institute of Metrology, Electronics and Computer Science of the University of Zielona Gora for their invaluable inspiration, hard work, and creative analysis during the preparation of this paper.

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

References

1. *Embedded Intel486 Processor Hardware Reference Manual*; Intel Corporation: Santa Clara, CA, USA, 1997; p. 334.
2. Brey, B.B. *The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-Bit Extensions: Architecture, Programming, and Interfacing*; Pearson Prentice Hall: Old Bridge, NJ, USA, 2009; p. 925.
3. Adkins, A.; Ammeson, B.; Anouna, J.; Garside, T.; Hunker, L.; Mailand, S. Intel Core i7 Memory Hierarchy. Available online: http://web.cs.wpi.edu/~cs4515/d15/Protected/LecturesNotes_D15/Week3_TeamA_i7-Presentation.pdf (accessed on 23 May 2023).
4. Clemente, J.A.; Gran, R.; Chocano, A.; del Prado, C.; Resano, J. Hardware Architectural Support for Caching Partitioned Reconfigurations in Reconfigurable Systems. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2016**, *24*, 530–543. [CrossRef]
5. Omran, S.S.; Amory, I.A. Implementation of LRU Replacement Policy for Reconfigurable Cache Memory Using FPGA. In Proceedings of the 2018 International Conference on Advanced Science and Engineering (ICOASE), Kurdistan Region, Iraq, 9–11 October 2018; pp. 13–18.
6. Inoue, H. Multi-step LRU: Simd-based Cache Replacement for Lower Overhead and Higher Precision. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 174–180.
7. Wang, C.; Wang, J.; Wang, M. Cache Performance Research for Embedded Processors. *Phys. Procedia* **2012**, *25*, 1322–1328. [CrossRef]

8. Shimizu, A.; Townley, D.; Joshi, M.; Ponomarev, D. EA-PLRU: Enclave-aware Cache Replacement. In Proceedings of the HASP '19: Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy, Phoenix, AZ, USA, 23 June 2019.
9. Abel, A.; Reineke, J. Reverse engineering of cache replacement policies in Intel microprocessors and their evaluation. In Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2014), Monterey, CA, USA, 23–25 March 2014; IEEE Computer Society: Washington, DC, USA, 2014; pp. 141–142.
10. Roque, J.V.; Lopes, J.D.; Véstias, M.P.; de Sousa, J.T. IOb-Cache: A High-Performance Configurable Open-Source Cache. *Algorithms* **2021**, *14*, 218. [CrossRef]
11. Zhang, K.; Wang, Z.; Chen, Y.; Zhu, H.; Sun, X.-H. PAC-PLRU: A Cache Replacement Policy to Salvage Discarded Predictions from Hardware Prefetchers. In Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach, CA, USA, 23–26 May 2011; pp. 265–274.
12. Lentz, M.; Franklin, M. Performance of Private Cache Replacement Policies for Multicore Processors. In Proceedings of the 4th International Conference on Computer Science, Engineering and Applications, Dubai, United Arab Emirates, 7–8 March 2014; Volume 4, pp. 1–7.
13. Perez, W.J.H.; Sanchez, E.; Reorda, M.S.; Tonda, A.; Medina, J.V. Functional Test Generation for the Plru Replacement Mechanism of Embedded Cache Memories. In Proceedings of the 2011 12th Latin American Test Workshop (LATW), Beach of Porto de Galinhas, Brazil, 27–30 March 2011; pp. 1–6.
14. Grund, D.; Reineke, J. Toward Precise PLRU Cache Analysis. In Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis, WCET 2010, Brussels, Belgium, 6 July 2010; Volume 15, pp. 23–35.
15. Puidenko, V.; Kharchenko, V. The minimizating of logical scheme for implementation of pseudo LRU by inter-type transition in trigger structures. *Radioel. Comp. Syst.* **2020**, *2*, 33–47.
16. Puidenko, V. Automaton model, device synthesis and adaptive substitution algorithm for cache memory. *Radioel. Comp. Syst.* **2020**, *4*, 68–78.
17. Zhu, W.; Zeng, X. Decision Tree-Based Adaptive Reconfigurable Cache Scheme. *Algorithms* **2021**, *14*, 176. [CrossRef]
18. Rashid, S.; Razak, S.A.; Ghaleb, F.A. IMU: A Content Replacement Policy for CCN, Based on Immature Content Selection. *Appl. Sci.* **2022**, *12*, 344. [CrossRef]
19. Sheraz, M.; Shafique, S.; Imran, S.; Asif, M.; Ullah, R.; Ibrar, M.; Khan, J.; Wuttisittikulkij, L. A Reinforcement Learning Based Data Caching in Wireless Networks. *Appl. Sci.* **2022**, *12*, 5692. [CrossRef]
20. Fang, J.; Kong, H.; Yang, H.; Xu, Y.; Cai, M. A Heterogeneity-Aware Replacement Policy for the Partitioned Cache on Asymmetric Multi-Core Architectures. *Micromachines* **2022**, *13*, 2014. [CrossRef] [PubMed]
21. Knoben, P.; Alachiotis, N. Improving Performance of Hardware Accelerators by Optimizing Data Movement: A Bioinformatics Case Study. *Electronics* **2023**, *12*, 586. [CrossRef]
22. Siddiqui, M.F.; Ali, F.; Javed, M.A.; Khan, M.B.; Saudagar, A.K.J.; Alkhathami, M.; Abul Hasanat, M.H. An FPGA-Based Performance Analysis of Hardware Caching Techniques for Blockchain Key-Value Database. *Appl. Sci.* **2023**, *13*, 4092. [CrossRef]
23. Shin, D.-J.; Kim, J.-J. Cache-Based Matrix Technology for Efficient Write and Recovery in Erasure Coding Distributed File Systems. *Symmetry* **2023**, *15*, 872. [CrossRef]
24. Puidenko, V.; Kharchenko, V. The Minimizating of Hardware for Implementation of Pseudo LRU Algorithm for Cache Memory. In Proceedings of the 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Kyiv, Ukraine, 14–18 May 2020; pp. 65–71.
25. Perepelitsyn, A. Method of Creation of FPGA based Implementation of Artificial Intelligence as a Service. *Radioel. Comp. Syst.* **2023**, *3*, 27–36. [CrossRef]
26. Perepelitsyn, A.; Kulanov, V.; Zarizenko, I. Method of QoS Evaluation of FPGA as a Service. *Radioel. Comp. Syst.* **2022**, *4*, 153–160. [CrossRef]
27. Barkalov, A.; Titarenko, L.; Mazurkiewicz, M. *Foundations of Embedded Systems*; Studies in Systems, Decision and Control, V. 195; Springer: Cham, Switzerland, 2019; 167p.