Int. J. Appl. Math. Comput. Sci., 2026, Vol. 36, No. 1, 155–172
DOI: 10.61822/amcs-2026-0012

TWO HEURISTIC METHODS OF HARDWARE THREADS INTERLEAVING IN A TIME PREDICTABLE MULTITASKING SYSTEM

ANDRZEJ PUŁKA [a,*], ERNEST ANTOLAK [a], BARTŁOMIEJ TRUŚ [a]

[a] Department of Electronics, Electrical Engineering and Microelectronics
Silesian University of Technology
ul. Akademicka 16, 44-100 Gliwice, Poland
e-mail: {apulka,ernest.antolak}@polsl.pl

This paper presents and tests several task scheduling methods in a real-time multitasking system based on the thread interleaving mechanism. The original configurable multicore time-predictable system architecture for multitasking is briefly discussed. The essential requirement of the system is the predictability of tasks, regardless of their number and when they are initialized. The paper focuses on the appropriate configuration of core interleaving registers, which determine the order and frequency of execution of individual tasks. This study develops various heuristic algorithms based on genetic programming and task execution rate analysis, and implements these in the Python language and PROLOG. The experiments are conducted with different task scenarios and system work requirements: with minimization of resources, energy, and operating frequency. A special task analyzer is used to evaluate the quality of the resulting system configuration. The results obtained are tested on a real hardware structure implemented in an FPGA chip. The proposed approach can be a useful tool for configuring real-time multitasking systems.

Keywords: real-time systems, timing simulation, task scheduling, multitasking, pipeline interleaving.

1. Introduction

Modern electronic systems, which are based on multiprocessor task processing, have the capability to implement highly complex algorithms. The hardware complexity of these systems is continually expanding, a phenomenon that can be attributed to the ever-increasing expectations of the market and the requirements of users. The development of semiconductor technology has resulted in digital systems clocked with progressively faster clock signals. Paradoxically, this technological progress has caused problems in predictability of timing in such systems (Schoeberl *et al*., 2015). Initially, processor timing predictability problems were addressed at the software level by developing real-time operating systems (RTOSs) (Paolieri *et al*., 2011; Ungerer *et al*., 2010). However, this approach is prone to inaccuracy due to the high level of abstraction at which task handling occurs (Gajski *et al*., 2009; Lee, 2005). Most programming languages, such as C (Andalam *et al*., 2014) or C++, do not provide direct control over the elapsed task time (Broman *et al*., 2013). This, in conjunction with a substantial number of abstraction levels, allows only approximate time control (Henzinger and Kirsch, 2002; Lee, 2006; Cazorla *et al*., 2006).

The authors of this paper have dedicated their research to real-time systems issues for an extended period, with a focus on enhancing hardware-level solutions. In recent years, they have successfully developed and implemented a configurable multicore hardware framework (Antolak and Pułka, 2020; 2021; 2023). Numerous algorithms and methodologies have been proposed to appropriately size resources, allocate tasks to individual cores of the system (Antolak and Pułka, 2022), and calculate its appropriate operating frequency. Ensuring full-time predictability is paramount for all of these actions. The proposed solution is based on pipelined processing and interleaving of hardware threads, and thus a crucial aspect is effective invocation of individual tasks. This article emphasizes the generation of a sequence of task identifiers, which is a pivotal concern in pipelined thread processing. It is demonstrated that the sequence and frequency of tasks in the core pipeline dictate the

*Corresponding author

system's overall performance and predictability.

The primary contribution and innovation of this paper reside in the following aspects:

- the development and implementation of algorithms for configuring the content of interleaved cycle registers (ICRs), which serve as a hardware alternative to RTOSs, represent a significant advancement in the field;
- secondly, the demonstration of the effectiveness of the aforementioned algorithms is substantiated by the pre-layout worst case timing analyzer (WCTA);
- the paper also details hardware implementation of these solutions in an FPGA structure.

The work is organized into six sections. The first three are devoted to a concise discussion of related work, followed by an examination of the primary components of the proposed multitasking system architecture. Section 4 details the implementation of the WCTA. The fifth section presents the two proposed heuristic approaches: genetic and proportional. The final section encompasses the experimental setup, the results of the conducted experiments, and a discussion of the results. The article concludes with a summary of the key findings.

2. Related work

The development of real-time systems has been an active area of research for many years. First, Edwards and Lee (2007) formulated the paradigm of time-predictable systems, called precision timed machines (PRETs). This paradigm postulated the development of a system that would consistently, regardless of workload, execute scheduled tasks within the allotted time frame. This concept has been extensively researched and developed by numerous scholars. In their seminal work, Thiele and Wilhelm (2004) provided a comprehensive set of recommendations and guidelines with the objective of improving the design process of time-critical embedded systems. Ip and Edwards (2006) proposed expanding the command list for RISC processors to include the DEADLINE instruction to enable the control of time-critical tasks. Researchers at the Center for Hybrid and Embedded Software Systems (CHESS), based at the University of California, Berkeley (Lickly *et al*., 2008), proposed a PRET processor architecture with memory access based on the so-called "memory wheel." In subsequent years, the CHESS group engaged in collaborative research with Saarland University and Linköping University, culminating in the publication of a novel PRET architecture in 2010 (Liu *et al*., 2010) that facilitates concurrent program execution. This architecture enabled the integration of distinct components while preserving the temporal characteristics of tasks. The solution incorporated a thread interleaving mechanism, scratchpad memory, and a composable and predictable DRAM (dynamic random access memory) controller.

Another article of the same team (Liu *et al.*, 2012) presented a significant implementation of a precision-clocked machine, known as the PTARM architecture. It is based on ARM family processors and implements a subset of the ARMv4 microarchitecture. The PTARM solution has been shown to improve system performance through the implementation of a sophisticated thread-interleaved pipeline, an exposed memory hierarchy, and a recursive DRAM controller. In 2014, the CHESS group developed a novel platform for processing mixed-criticality tasks, dubbed FlexPRET (Zimmer *et al*., 2014), that delineates a distinction between hard-time threads (HRTT) and soft-time threads (SRTT). The FlexPRET architecture, generated by the CHISEL tool, provided hardware isolation for HRTT tasks, while SHTT tasks efficiently utilized available processing resources. The authors introduced dedicated timing instructions into an ISA based on the RISC-V processor family and proposed a thread scheduler that maintained control over the threads processed by the pipeline. The entire structure was also implemented in an FPGA device.

The paper by Andalam *et al*. (2014) dealt with the analysis of time-predictable systems at a higher level of abstraction, introducing a new lightweight and concurrent language, PRET-C (precision time version of C). With its syntax, synchronous semantics, and very simple time handling mechanisms, it is well suited for predictable PRET architectures. The authors further proposed a hardware gas pedal for executing PRET-C on soft-core processors, thereby enabling time-predictable execution with high performance. This time-predictable architecture was named ARPRET.

Another interesting solution is presented by Oliveira *et al*. (2011), who propose their own time-predictable architecture, dubbed ARPA-MT. It comprises three primary components: the main computing unit, two Cop0-MEC coprocessors responsible for managing memory operations and handling exceptions and interrupts, and Cop2-MEC implementing and accelerating hardware RTOS functions. The ARPA-MT structure (Oliveira *et al*., 2011) contains a 5-stage pipeline that is notable for its innovative design. The first two stages (IF and ID) are replicated for each thread, while the remaining three stages (EX, MA, and WB) process different threads, which are interleaved. This innovative solution demonstrates the potential for hardware-software synergy in the design of real-time systems.

A group of researchers from various international backgrounds recently unveiled a novel multicore processor architecture, named Patmos (Schoeberl

et al., 2012). Based on this innovation, in 2015, a consortium of 24 European researchers presented the T-CREST project (Schoeberl *et al*., 2015), which showcased a multicore approach to the original concept of the PRET system. Addressing the challenges posed by the concurrent execution of tasks in multicore systems, research efforts have been focused on identifying solutions, as evidenced by Fernández *et al*. (2012), Alhammad and Pellizzoni (2014), Moulik *et al*. (2018), Pathan *et al*. (2018), Kim *et al.* (2020a), or Chen *et al*. (2019).

Research on time-predictable systems, which has been conducted for many years, has encompassed the analysis of a wide range of issues related to hardware development (Cazorla *et al*., 2006; Thiele and Wilhelm, 2004; Schoeberl *et al*., 2012; Forsberg *et al*., 2018; Akesson and Goossens, 2012; Reineke *et al*., 2011) and software development (Fernández *et al*., 2012; Alhammad and Pellizzoni, 2014; AlBarakat *et al*., 2018; Schoeberl, 2008). Researchers frequently seek general solutions (Andalam *et al*., 2014; Broman *et al*., 2013), yet numerous works address dedicated ones (Schoeberl *et al*., 2012; Akesson and Goossens, 2012). Axer *et al*. (2014), provide an overview of numerous issues related to the design of embedded time predictable systems.

Kim *et al.* (2020b) and Rehman *et al*. (2020) examined the scheduling problem with load balancing and proposed techniques to reduce the energy consumption of the systems. Glaser *et al*. (2021) present a technique for efficient use of processing elements in processor clusters with shared L1 cache. This technique is based on optimized synchronization and communication between the components of the system. Some works demonstrate the use of heuristic methods and artificial intelligence algorithms in the task scheduling process (Bahn and Cho, 2020; Chniter *et al*., 2020).

The authors of the present publication in recent years have developed an original architecture for a time-predictable multitasking system (Antolak and Pułka, 2020; 2021; 2023). This solution allows flexible pipeline configuration and utilizes a set of dedicated scheduling algorithms. The authors have also demonstrated the feasibility of optimizing the energy consumption of such a system and accurately estimating its energy demand. An essential component of the proposed system is the appropriate configuration of the sequence of thread identifiers corresponding to the tasks processed in the pipeline. The dedicated worst case timing analyzer (WCTA) was used for the analyses and verification of timing conditions. This tool allows accurate system timing analysis at the system design stage and ensures that no task exceeds its designated deadline. The current publication focuses on refining the thread identifier generation method and improving system performance, presenting two novel heuristic methods that

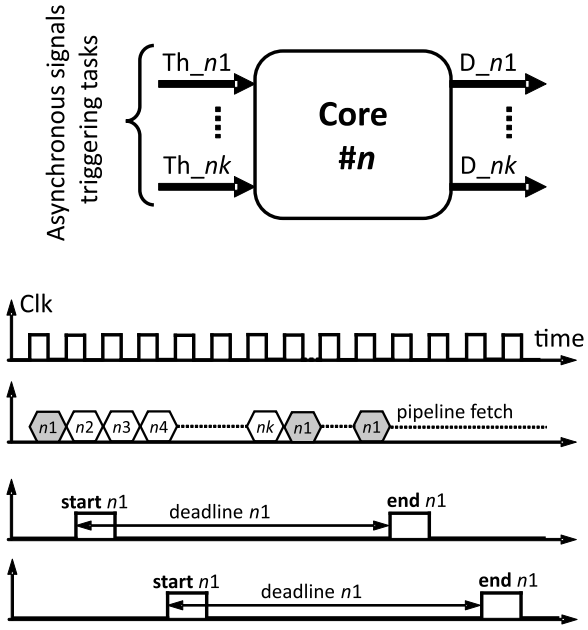

Fig. 1. Tasks processed in a single core and two possible scenarios.

achieve superior system throughput with the same task configuration. It is imperative to acknowledge that, in the context of multitasking digital systems, numerous parameters contribute to the quality of these systems. In view of the ever-increasing requirements, one of the factors raised in the publications discussed is the energy requirements of such systems and the search for opportunities to minimize dissipated power. Among the crucial parameters affecting the dissipated dynamic power of such systems is the operating frequency. The authors of this paper demonstrate that it can be reduced by optimizing the configuration of the ICR while ensuring full timing predictability of all tasks executed. A series of simulation experiments were conducted to verify the validity of the method and hardware implementations in programmable FPGA structures which justified the approach.

3. Architecture of the time predictable system

A thorough delineation of the system's architecture is provided by Antolak and Pułka (2023). In this paper, we shall confine our discussion to a concise summary of the fundamental principles underpinning the proposed PRET architecture. The fundamental requirement for the system under consideration is that, irrespective of the system's load, each task executed within the system must be completed prior to a predetermined execution time, herein referred to as a "deadline." In addition, it is assumed that these tasks are inherently asynchronous,

given their potential to be triggered by external interrupts, the occurrence of which is difficult, if not impossible, to predict (Fig. 1). Before system initiation, the number and type of tasks mapped to a core are known while the timing of tasks is not. In contrast, the temporal parameters of each task and, consequently, the aggregate load on the system at any given moment of operation are not known. So, to meet the time-predictable assumption, the system should operate as if it were required to execute the complete sequence of all tasks in succession (the worst-case scenario). In the case where a task was not activated and, according to the established work schedule, the pipeline was expected to fetch the instruction for that task in the subsequent time slot, the cycle would be considered idle, meaning that no actions would be executed. The subsequent sections will address thread interleaving in greater detail.

The proposed multitasking architecture is based on pipelined processing and can consist of one to eight reconfigurable cores. It is possible to reconfigure pipelines, which can contain between five and 12 stages, depending on the application requirements. These limitations are due to the capacity of the Xilinx Virtex 7 and Kintex UltraScale FPGA chips used for implementation and the number of tasks processed (up to 256). The basic five stages of a pipeline are as follows: IF (instruction fetch), ID (instruction decode), EXE (execution), MA (memory access), and WB (write back). These stages can be extended (divided) into substages, depending on the specific application and system needs. A detailed description of these pipeline stages will also be omitted here.

In Fig. 2, a simplified schematic of a single system core is presented. The diagram shown in the figure is a conceptual model of the system. Assuming that k tasks are mapped to a given core, they are processed alternately in that core according to the principle of thread interleaving. The order in which the threads enter the core is determined by the contents of the ICR. Whether a task is executed is determined by an external start signal that activates it. If the task has not been called, then the time slot allocated to it is in the NOP (no operation) state. A task is deactivated after its program execution is complete. Thus, as can be seen from this diagram, properly configuring the contents of the ICR is important so that all tasks are completed on time (meet their deadline requirements) at the lowest possible operating frequency, regardless of the number of active tasks (operation scenario). As mentioned above, the system as a whole can contain up to eight cores, interconnected by a shared bus, which facilitates potential information exchange between threads. The system cores communicate with external peripherals via input/output ports, which are mapped as part of each thread's data memory.

Each thread can have its own individual input/output

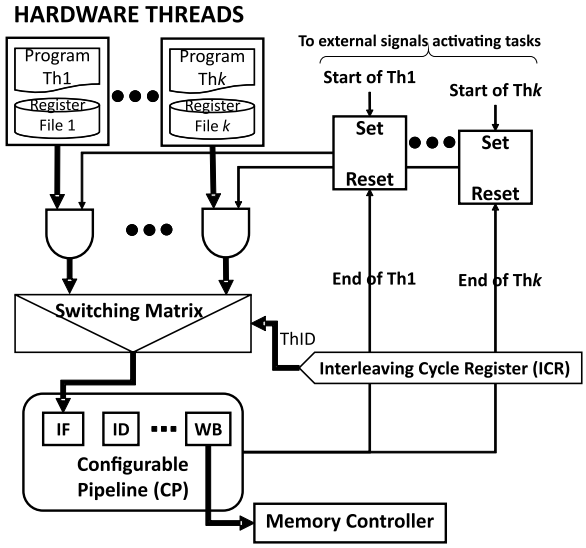

Fig. 2. Structure of a single core of the system.

ports and is started by its corresponding external signal. The procedure for launching a given thread can be implemented either by hardware-based (using an external interrupt) or software-based means by executing the corresponding instruction by another processor thread. It should be noted that the current implementation of the system enables concurrent execution of up to 255 distinct tasks, with the potential for any distribution of these tasks across cores. Theoretically, the maximum number of cores is also 255. Each core is equipped with a configurable number of supported threads, which are autonomous entities that do not depend on each other. Consequently, these threads must have their own resources, such as memory and register files, to ensure their independence and efficient operation.

The solution utilizes the interleaving hardware thread mechanism (Lee and Messerschmitt, 1987), which, under specific conditions, can effectively prevent data and control hazards. In this regard, instructions belonging to different tasks are executed in the pipeline at any given time. The absence of hazards conditions depends on the minimum distance $Min_{indistance}$ (measured in clock cycles) between consecutive instructions of the same hardware thread (task):

$$Min_{indistance} = Pipelength - 1. \tag{1}$$

In order to meet both the time criterion (deadline) and the above condition (1), it is first necessary to choose the appropriate system configuration (number and type of cores) and allocate tasks to resources in the correct manner. Antolak and Pułka (2023) delineate the precise process of task scheduling and subdivide it into five major steps:

1. task mapping,

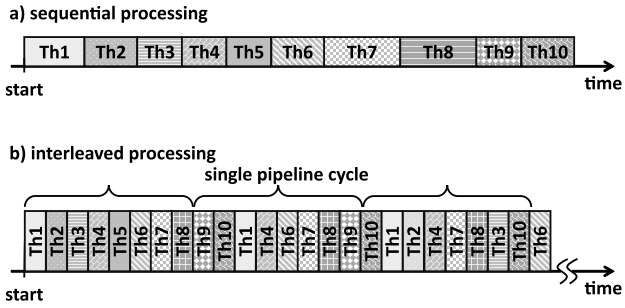

Fig. 3. Multitask processing schemes: sequential (a), concurrent with thread interleaving (b).

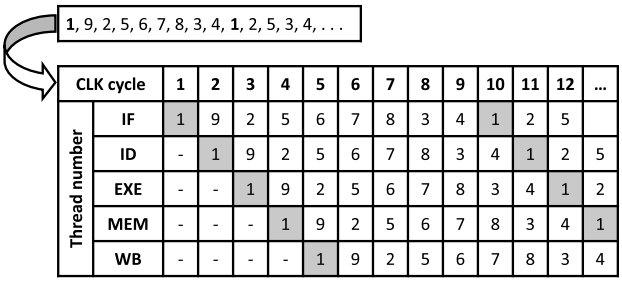

Fig. 4. Example of an ICR sequence processed in the pipeline.

2. task arrangement,
3. initial simulation,
4. determination of the operating frequency,
5. final simulation.

As demonstrated in previous studies, the task mapping process is dependent on the design strategy and any additional requirements imposed on power consumption, structure cost, and related factors. During this phase, the number of cores is selected and the initial operational frequency is established. In the second stage of scheduling, the dimensions and state of the specialized interleaving cycle register, which contains a sequence of thread identifiers, must be determined.

3.1. Interleaving cycle register. In multitasking systems, successive tasks can be processed sequentially or in parallel with resource access time sharing (Fig. 3). The implementation of a solution based on dynamic thread interleaving requires special attention when the time requirements of the executed tasks are very strict.

The tasks executed by threads are alternated (interleaved) in accordance with the sequence stored in a designated interleaving cycle register (ICR). This register contains the identifiers of the threads. Its length can also be selected accordingly. The ICR plays a pivotal role in managing the process of task execution, as the sequence and frequency with which instructions of individual tasks are executed are contingent on its contents. A well-designed ICR enables thread execution to be prioritized based on their frequency requirements, thereby ensuring efficient utilization of processing resources. This mechanism enables the core to adjust tasks in such a way that threads with very strict timing constraints and those with less strict ones can run within a single core.

In Fig. 4, an example of an ICR content is depicted in a fundamental core comprising 5-stage pipeline. The upper portion of the figure illustrates the initial segment of the ICR content. In subsequent processing cycles, threads with numbers contained in the ICR are loaded into the pipeline. This sequence is determined during the execution of the "task arrangement" phase. The figure demonstrates a task from ID = 1, which is repeated every eight cycles. This sequence is initiated only under the condition that the task has been activated by an external asynchronous signal. A detailed examination of this section reveals that the task with ID = 3 occurs much more frequently (every five cycles), indicating that it has more rigorous timing requirements.

In order to reduce the dependence between the number of threads processed by a single core, minimize the maximum clock frequency, and simplify the control of the predictability of the system clocking, the pipeline can be extended with additional stages. However, it should be remembered that a core with an extended pipeline must process a sufficient number of threads (Lee and Messerschmitt, 1987).

The present study focuses on the second stage of scheduling, namely, task arrangement. An essential issue in this scheduling stage is the appropriate configuration of the ICR. In our previous work (Antolak and Pułka, 2023), a somewhat complex solution was presented based on a set of timing registers that count time and determine the position of a task in the list based on the so-called time windows. Before alternative solutions are presented, some concepts related to processed tasks should be recalled.

3.2. Taxonomy of performed tasks. We assume that all tasks are of the hard type (Buttazzo, 2011; Lamie, 2009), which means that no thread can fail to meet the deadline. However, a certain priority mechanism is introduced in the scheduling process, in which tasks are divided into several groups. Tasks that are crucial to the security of system operation have been identified as strong hard timed (SHT). Such tasks have the highest priority and are set first in the serialization process. The remaining tasks are divided into three groups:

1. Non-cooperative thread (NCT): These tasks are

defined as independent, i.e., not cooperating or communicating with other tasks.

2. Cooperative thread (CT): Cooperative tasks in which data exchange time fluctuates between certain minimum and maximum values.

3. Cooperative thread memory hard (CTHM): Cooperative tasks with fixed data exchange time.

It is important to emphasize that the integration of the CTHM category enables the mitigation of the disparity between the maximum and minimum execution times of a task, albeit at the cost of increased system load (i.e., a reduced number of such tasks can be allocated to a single core of the system). Furthermore, it should be noted that only tasks in the NCT or CTHM category can be assigned the highest priority (STH).

3.3. Tasks' modeling parameters. The task model parameters consist of numerical issues that describe the complexity of the i-th task and the time requirements as well as some flags that indicate the type of a given task:

$$T_i = C_i, M_i, D_i, CTHM_i, SHT_i, \tag{2}$$

where C_i denotes the total number of standard instructions of the i-th task, M_i is the total number of memory access instructions of the i-th task not included in C_i, D_i denotes the deadline of the i-th task (maximum execution time of the task), $CTHM_i$ is the flag denoting whether the i-th task is of CT type (0) or CTHM type (1), and SHT_i expresses the fact that, when the flag is set (1), this means that the i-th task is strong hard timed (SHT).

Antolak and Pułka (2020) proposed a significant parameter, the task frequency (TF), which facilitates the estimation of the computing power requirements for each task. This parameter can be determined for each task assigned to the core of the system using the following equation:

$$TF_i[MHz] = \frac{C_i + M_i \cdot \left[\frac{M_{dur}}{Min_{indistance}}\right] + 2}{D_i[\mu s]}, \tag{3}$$

where M_{dur} denotes the number of clock cycles required to perform a data exchange operation with another thread.

The task frequency presented in this manner, which correlates the complexity of the task with the time requirements, streamlines the scheduling process and can be used in the procedure to estimate the energy requirements of the system (Antolak and Pułka, 2024).

4. ICR task arrangement algorithms

4.1. Initial assumptions. Prior to embarking upon this stage of analysis, it is assumed that the TF_i of each task

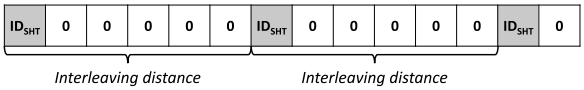

Fig. 5. Example of the reset ICR sequence containing SHT task identifiers.

are already known, and thus the total sum (TS) of all task frequencies can be evaluated:

$$TS = \sum_{i=1}^{k} TF_i. \tag{4}$$

The initialization of each ICR is achieved by assigning the initial $SICL$ (single interleave cycle length) of the register in question. For example, the initialization of an ICR with an $SICL$ value of 3000 results in the register being reset to a state where all positions are marked with "0", thereby indicating idle cycles.

Concurrently, the duty factor (DF), which quantifies the extent to which the register is populated by the identifiers of a specific task, should be calculated for each task. The factor can be expressed as

$$DF = \frac{TF_i}{TS}. \tag{5}$$

This factor will subsequently enable the determination of the number of anticipated occurrences of a given task tag within the entire ICR sequence:

$$N_{expcnt} = \Big[DF \cdot SICL\Big]. \tag{6}$$

However, this value is typically calculated at the time of insertion of identifiers of a given task, when $SICL$, the length of the ICR, is finally determined. In the presented system, the assumption is made that one SHT task has been assigned to each core of the system, which means that the process of register configuration always begins with the deployment of SHT task tags. The procedure commences with the calculation of the frequency of occurrence of identifiers of the task, that is, the determination of the interval between consecutive SHT task IDs in the ICR sequence. However, this interval must not be less than $Min_{indistance}$ (1):

$$SHT_{Distance} = \left[\frac{SICL}{N_{expcnt}}\right]. \tag{7}$$

The proposed algorithms will start with the utilization of a meticulously prepared ICR, which contains the SHT task identifiers (Fig. 5).

Due to the looping of the ICR, that is, the round-robin scheme of ID processing, where the end of the register is affixed to its beginning, it is essential

to pay close attention to the proper placement of edge elements (Fig. 6). In view of this, after the end of the sequence is joined with its beginning, a tail-head subsequence is obtained. This subsequence must satisfy all the requirements related to the interposition of task identifiers. This requires adhering to the $Min_{indistance}$ spacing between equal indices. This issue must be fully considered in all the methods discussed; however, it will only be addressed in Section 4.4.

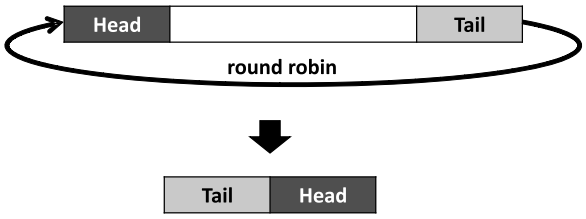

Fig. 6. Round-robin processing scheme of ICR identifiers.

4.2. WCTA. It is imperative to analyze and simulate the results obtained during the scheduling process repeatedly. Consequently, a specially developed time analyzer, WCTA, has become an integral component of the design environment. As illustrated in Fig. 1, the execution of a task is triggered by the external signal and, consequently, is asynchronous to the processing cycle established in the pipeline. This can result in fluctuations in the execution times of individual tasks. This variation in execution times depends on the temporal synchronization between the activation signal and the allotted time slot for each task. A comprehensive analysis of this phenomenon is presented by Antolak and Pułka (2023).

In this paper, we will limit ourselves to stating that the WCTA facilitates the determination of two key metrics. First, the maximum execution time of a task is expressed by the corresponding number of standard instructions, herein denoted $MaxI_i$. Second, the minimum operating frequency of a given core of the system is determined, allowing the achievement of all timing requirements and the meeting of all deadlines. It is evident that the simulation process enables the acquisition of a substantially more extensive array of information when the WCTA is employed. However, the present study does not intend to explore this aspect in depth.

4.3. Genetic approach. Issues related to optimization have been identified in a wide range of fields of knowledge (Kochenderfer and Wheeler, 2019). A large number of optimization techniques have been developed, and the subject is comprehensively addressed in the existing literature. The selection of an appropriate method is contingent on the analysis of the specific problem. To ensure method selection that is both effective and appropriate, it is essential to define the relevant function with precision.

In our case, the configuration of the ICR is a matter of complexity. To determine an optimal resolution to this quandary, one must optimize the result for all tasks simultaneously, since the ultimate minimum frequency of the system is the maximum value impacted by all tasks. Consequently, it can be expected that the solution manifold will comprise a substantial number of local minima. As stated above, it is imperative to examine the distribution of identical identifiers with regard to each other, in addition to the aggregate sum of their instances. Another pertinent issue is the imposition of limitations on the spacing between successive identifiers of a given task, a parameter that is denoted by $Min_{indistance}$.

In view of these requirements, it was determined that a solution based on a genetic algorithm should be employed. The use of genetic methods is based on an extensive search for the solution space, a strategy that facilitates the avoidance of local minima. An additional reason for employing genetic programming is the relative ease with which it can be parallelized. This is a particularly salient property in this context because of the computationally intensive nature of calculating the fitting function.

4.3.1. Initial calculations. Before the actual algorithm is run, the input data should be properly prepared by performing some preliminary calculations.

Initially, the number of expected occurrences of the identifier for each task, denoted by $N_{i_{expcnt}}$ (6), must be determined (6). Subsequently, the size of the ICR can be adjusted, such as by rounding up the $SICL$ number, which consequently becomes the total sum of all identifiers:

$$SICL = \sum_{i=1}^{k} N_{i_{expcnt}}. \tag{8}$$

Furthermore, it is assumed that the probability of generation (occurrence) of a given identifier in the ICR sequence is equivalent to the DF coefficient (5).

4.3.2. Algorithm description. The genetic algorithm used is based on the classical evolutionary approach (Goldberg and Goldberg, 2012) with some modifications. Its description is given in Algorithm 1 (Truś, 2023). A detailed discussion of the implementation and modification of the algorithm is presented in the next section of the paper.

Let us review a few of these steps, pointing out the constraints concerning the construction of the ICR that is responsible for controlling the threads' interleaving process.

Algorithm 1. Genetic generation of the ICR.

Step 1: Initialization. Determine $N_{i_{expct}}$ for all tasks.

Step 2: Setting-up the ICR. Correct the length of the ICR,

$$SICL = \sum_{i=1}^{k} N_{i_{expcnt}},$$

and insert identifiers of the SHT task (Fig. 5).

Step 3: Starting generation. Generate the initial generation and set the process parameters.

Step 4: Crossover. Generate the next generation by crossing the genetic material.

Step 5: Mutation. Randomly change the contents of the elements.

Step 6: Evaluation. Calculate the fitness function for all elements of the new generation.

Step 7: Selection. Select the elite and parents for the new generation.

Step 8: Decision. Check if all generations are produced? **IF YES GOTO Step 9 ELSE GOTO Step 4**.

Step 9: Result. Select the best candidate as a final result.

Step 3: Starting generation. In order to initiate the evolutionary procedure aimed at identifying the optimal solution, that is, the ICR configuration, it is necessary to provide an initial generation. The members of this generation are created randomly; however, certain restrictions apply with respect to the frequency of occurrence of each identifier (5) and the minimum interval between successive occurrences of the same identifier. It is assumed that only the position of SHT task identifiers is fixed and inviolable, i.e., not subject to change. It is imperative to ensure that the edge elements of the ICR sequence are properly configured with each other. Specifically, the tail-head subsequence must meet all of the aforementioned requirements. The verification and correction method for this subsequence is described below (see Section 4.4).

Step 4: Crossover. Crossing between individuals, designated as "parents", constitutes the fundamental operation used in the generation of a novel generation of descendants. Nevertheless, due to the particular characteristics of ICR sequence construction and the stipulations associated with the sequence's edges, a random exchange of genetic material is not permitted. This prohibition arises from the necessity to insert idle cycles into the sequence, which would result in a substantial augmentation in task processing. Consequently, the mixing of genetic material can be carried out according to the paradigm shown in Fig. 7, where the marginal elements invariably belong to one of the parents. The magnitude of the exchange is adjustable,

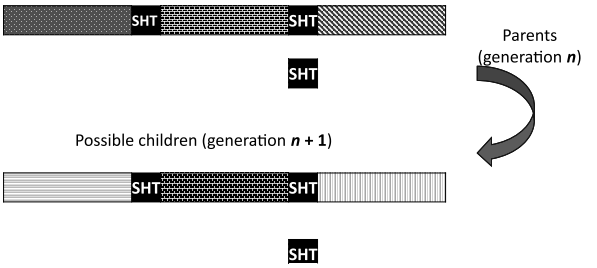

Fig. 7. Crossover and exchange of the genetic material.

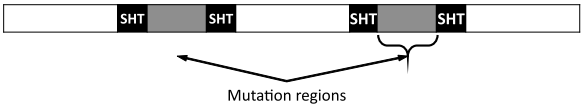

Fig. 8. Example of mutation regions of an ICR sequence.

and the boundary points invariably correspond to the positions of the SHT identifiers.

Step 5: Mutation. A similar situation occurs with mutations. In general, mutations are used to move the genetic algorithm away from local extremes, thus allowing it to find optimal solutions. Mutations involve randomly changing parts of the genetic code (Fig. 8). In the case of ICR sequences, such operations are quite risky and, as will be shown, do not yield the expected results. The introduction of random immigrants was associated with improved outcomes (Michalak, 2021).

Step 6: Evaluation. The quality of a given ICR sequence is evaluated using the WCTA, which serves as a metric for the adaptation of an individual. This assessment is crucial because the ability to meet all the timing requirements at the lowest possible operating frequency of the system is essential, particularly in the cases of significant energy demands. To this end, the fitness function, denoted by F_f, is formulated as follows:

$$F_f = \frac{TS}{F_{sim_N}}, \tag{9}$$

where F_{sim_N} denotes the minimal operating frequency obtained from the simulation of the N-th ICR sequence in the WCTA.

Furthermore, the value of TS signifies a theoretical limit, representing the sum of the frequencies of the individual tasks necessary to achieve their respective deadlines. Consequently, the objective should be to achieve results that approximate unity as closely as possible.

Step 7: Selection. It was decided that the solution would be based on the principle of elitism (Ahn and Ramakrishna, 2003), whereby a select number of the most adept individuals of the previous generation are passed to the next.

Algorithm 2. Proportional generation of the ICR.

Step 1: Initialization. Determine $N_{i_{expct}}$ for all tasks.

Step 2: Setting-up the ICR. Correct the length of the ICR,

$$SICL = \sum_{i=1}^{k} N_{i_{expcnt}},$$

and insert identifiers of the SHT task (Fig. 5).

Step 3: Single step progress calculation. For all non SHT tasks, calculate the SSP parameter,

$$SSP = \frac{1}{N_{expcnt}}.$$

Step 4: SSP sorting. Sort SSP coefficients in ascending order.

Step 5: Initial insertion of tasks' IDs. Insert identifiers of all tasks according to the order of SSPs.
Create progress lists of all tasks storing the progress of their execution.

Step 6: Final insertion of tasks' IDs. The ICR sequence is to be completed with subsequent identifiers according to the progress of the tasks. That means that, each time an identifier of a given task is placed, its progress is increased by the quantum SSP and shifted in the progress list.
This phase of the procedure lasts until all positions of the ICR contain tasks' IDs.

Step 7: Tail-head subsequence checking and correction. Check if the tail-head subsequence of the ICR has no violations. Correct all bugs.

The subsequent section is dedicated to an examination of the Python implementation of the genetic algorithm and the parameters that control the sequence generation process.

4.4. Proportional approach. The second proposed approach was designated as the proportional one. This technique, in contrast to the previous one, is based on the natural observation of the effects of inserting successive task identifiers into the ICR string and making dynamic decisions.

Algorithm 2 delineates the essential steps that must be followed to achieve the desired effect.

Steps 3–5: Single step progress. The single step progress coefficient (SSP) determines the incremental progress of a task after executing one of its instructions. Subsequent to determining this parameter for all tasks and arranging the tasks according to the incremental value of SSP, successive identifiers are entered into the next available positions of the ICR.

Step 6: Final insertion of tasks' IDs. A special progress list is created to track the progress of each task. The progress value of a task is increased by the factor SSP after the execution of an instruction, that is, after implementing one instruction of a given task. The updated progress value is then allocated to the corresponding progress list position. This mechanism ensures that the task with the least progress is prioritized for the ICR sequence. It is important to note that the procedure outlined here is designed to maintain a minimum distance between identifiers of the same task, referred to as $Min_{indistance}$. The overall operation is conducted until all identifiers are exhausted and the entire ICR sequence is completed.

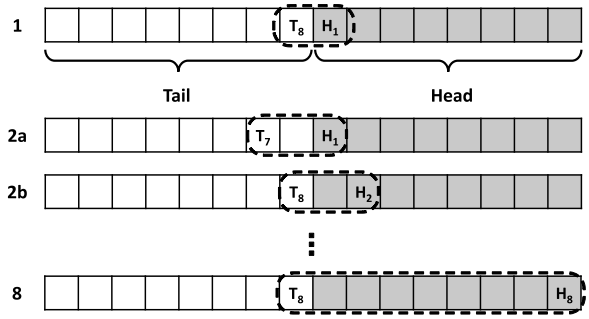

Fig. 9. Tasks processed in a single core and two possible scenarios.

Step 7: The verification/correction procedure for tail-head subsequence. The necessity of verifying the tail-head subsequence refers to the mechanism of round-robin operation of the ICR. As mentioned above, the elements of the ICR sequence reflect the order in which the processing instructions are processed for individual tasks in the pipeline. Consequently, during its creation, all rules related to the specificity of task interleaving are checked.

As illustrated in Fig. 9, the verification-correction procedure for $Min_{indistance} = 9$ involves eight series of comparisons in the worst case.

The first issue is the appropriate selection of the lengths of the tail and head sub-sequences, which should correspond exactly to the value of $Min_{indistance}$. Then a procedure is carried out to check the violation of the distance $Min_{indistance}$ between identical identifiers, i.e., are there identical identifiers in the tail and head parts? The operation is performed until the first violation is found, starting with a window of dimension 2. If further steps are deemed necessary, the window is incremented until the full length of $Min_{indistance}$.

If no violation is detected, the ICR sequence remains unchanged. However, if a violation is found, a list of possible task IDs is created. These IDs are sorted according to the order of their progress (progress list). Then, at the end of the sequence, a string of such identifiers of appropriate length is inserted between the

tail and head parts. In this way, the ICR sequence becomes slightly longer (by a few symbols), but the timing is not fundamentally affected.

5. Experiments

This section presents the results of experimental studies. It is divided into three parts. The first one discusses the experiments carried out with the genetic algorithm while analyzing the implementation in Python (Lutz, 2019). The second subsection is devoted to the proportional method implemented in the Prolog language (Bratko, 2012). The third part of the section presents and discusses the results obtained with both methods and compares them with the original solution by Antolak and Pułka (2023) as a reference one.

5.1. Python implementation of the genetic algorithm. The evolutionary method was implemented in the Python language (Truś, 2023), which is well suited for the analysis of text files. The process of generating individuals representing ICR sequences was parameterized, facilitating their control and observation of the influence of individual phenomena on the obtained results.

The following parameters were introduced:

- POP_SIZE: defines the population size,
- $ITER\_COUNT$: determines the number of generations generated,
- $CROSSOVER\_POINTS$: the number of points at which crossover occurs, i.e., the exchange of genetic material of the parents during the creation of a descendant,
- $MUTATION\_SIZE$: the number of consecutive identifiers generated by mutation,
- $MUTATION\_COUNT$: a random number of positions where mutation occurs,
- $ELITE\_PERCENTAGE$: the percentage of the best individuals retained for future generations,
- $CULL\_PERCENTAGE$: the percentage of individuals with the lowest degree of adaptation that are removed from each generation,
- $NEW\_POPS\_PERCENTAGE$: percentage of new individuals generated in the next generation.

Subsequent experiments demonstrate that this limitation does not affect the quality of the method, given the system under consideration and the number of tasks per single core, which varies between 10 and 60.

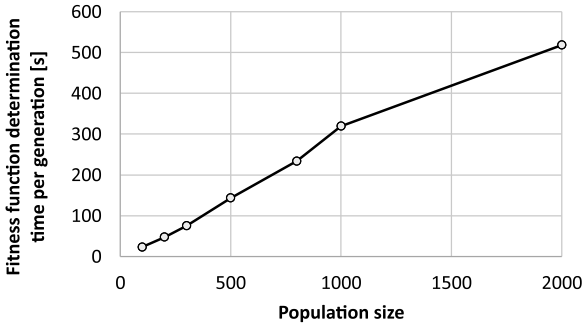

Fig. 10. Relationship between the time of determining the fitness function for all objects of a given generation and the size of the population.

Table 1. Calculation times of a single generation.

Population size	Calculation time [s]
100	23.6
200	47.58
300	75.34
500	143.63
800	233.83
1000	319.54
2000	518.24

Table 2. Minimum operating frequency of the system obtained for ICR sequences achieved for different values of the percentage of new items ($TS = 90$ MHz).

New items [%]	Min. op. frequency [MHz]
0	98
5	97
10	97
15	98
20	99
25	100

The problem under consideration involves static scheduling. The time required for the final system configuration is not a key parameter. However, in order to reduce the computation time and resources required, which unnecessarily increase the system design time and costs, it was decided to limit the size of a single generation.

The graph depicted in Fig. 10 illustrates the linear dependence of the computation time of the fitness function (9) on the population size. This operation requires the most time due to the necessity of executing the WCTA. Table 1 further illustrates the time required to calculate the next generation, after the execution of all crossover, mutation, and selection operations.

For the first experiments, the population size was limited to 100 items in each generation, and the number

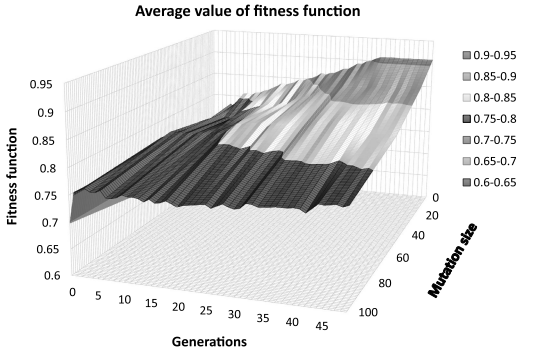

Fig. 11. Relationship between the average values of the fitness function of the successive generations and parameter $MUTATION\_SIZE$ for one mutation point ($MUTATION\_COUNT = 1$).

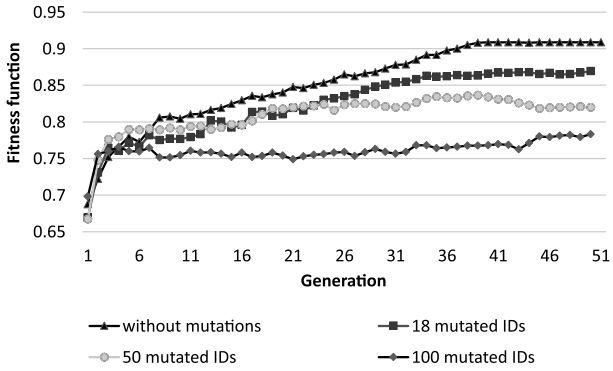

Fig. 12. Dependence of fitness function in successive generations from Fig. 11 for selected mutation sizes ($MUTATION\_COUNT = 1$).

of iterations corresponding to the number of generations was limited to 50.

During the course of the experiments, particular attention was devoted to the examination of the impact that the various algorithmic parameters exerted on the quality of the solution and the convergence of the calculations. Initially, the impact of the mutation on the results obtained was investigated.

The number of mutation points ($MUTATION\_COUNT$) and the size of the subsequence that was subject to mutation ($MUTATION\_SIZE$) were both modified. The first parameter was altered from 1 to 4, and the second from 0 (absence of mutation) to 100. As illustrated in Fig. 11, the 3-D dependence of the fitness function on the parameter $MUTATION\_SIZE$ is demonstrated for a single mutation point $MUTATION\_COUNT = 1$. Figure 12 shows the same relationship for some selected values of the parameter $MUTATION_SIZE$ in a 2-D fashion. In contrast, the subsequent graph (Fig. 13) compares the changes of F_f at $MUTATION\_SIZE = 18$ for three cases: no mutation, one region of occurrence of the mutation, and four areas of mutation.

In general, on the basis of several hundred experiments, it can be concluded that, for most cases, mutations do not work well for the generation of ICR sequences by the evolutionary method. Although mutations indeed result in faster convergence, all results yield F_f values below unity, indicating that there is no improvement in the F_m parameter. However, as discussed in the conclusion of this section, there are situations in which the evolutionary method can be a reasonable alternative.

The dependence of the function F_f on the number of new individuals transferred to a new generation was analyzed. Experiments were carried out for a percentage ranging from 0 to 25% of the population. The averaged graph of the fitness function for successive generations is shown in Fig. 14, while Fig. 15 presents the set of fitness functions for the best matched individuals for different percentages of new individuals. The analysis of these graphs indicates that the optimal range for the percentage of new individuals is between 5% and 10%. The fitness function achieved the best fit the fastest for a parameter value of 10%, as illustrated in Fig. 15. Table 2 presents examples of the minimum frequency of operation for $TS = 90$ MHz.

A subsequent series of experiments involved the evaluation of the impact of the percentage of eliminated elements in the population ($CULL\_PERCENTAGE$). As illustrated in Fig. 16, the dependence of the average values of the matching function of successive generations on the number of individuals eliminated demonstrates that a minimum of 25% of individuals should be eliminated to ensure the convergence of the algorithm.

Figure 17 demonstrates the relationship between the fitness function for the best matched individuals and the value averaged for the $CULL\_PERCENTAGE$ parameter of 40%. For this particular example, with $TS = 90$ MHz, the final ICR sequence obtained enables the system to reach a minimum operating frequency of 94 MHz.

Based on the results of previous experiments, the following algorithm parameters were used to determine the optimal number of crossover points:

- $POP\_SIZE = 100$,
- $ITER\_COUNT = 50$,
- $MUTATION\_SIZE = 0$,
- $MUTATION\_COUNT = 0$,
- $ELITE\_PERCENTAGE = 0.05$,
- $CULL\_PERCENTAGE = 0.25$,

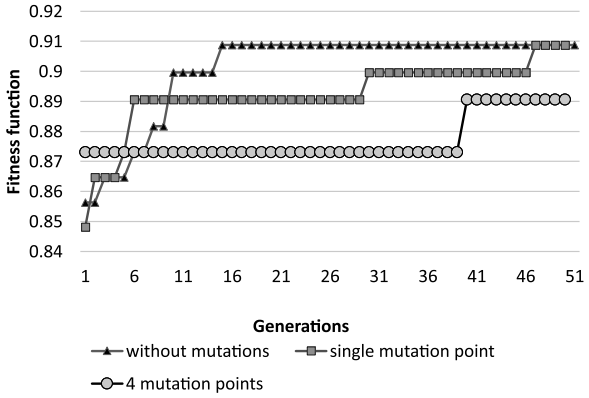

Fig. 13. Comparison of fitness function changes at $MUTATION\_SIZE = 18$ for 0, 1 and 4 mutation regions.

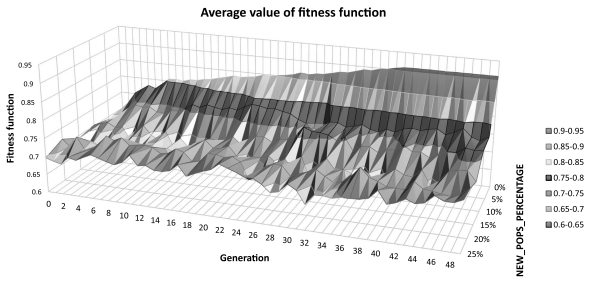

Fig. 14. Relationship between the average values of the fitness function of the successive generations and parameter $NEW\_POPS\_PERCENTAGE$.

- $NEW\_POPS\_PERCENTAGE = 0.10$.

As illustrated in Fig. 18, the results obtained for different numbers of crossover points are collected. As mentioned in the description of the evolutionary method, due to the special importance of SHT tasks, these points cannot be arbitrarily chosen (see Fig. 7). An analysis of the results indicates that, for the purpose of generating ICR sequences by means of the evolutionary method, optimal results and the most substantial convergence of the algorithm can be attained with a number of crossover points ranging from 1 to 3. The graph in Fig. 19 illustrates the alterations in the values of the fitness function for the optimal sequences and the values averaged over successive generations for $CROSSOVER\_POINTS = 3$.

The author of one of the seminal works on evolutionary methods (Alander, 1992) proposes that the optimal population size for optimal evolutionary outcomes should fall within certain limits. The formula for this calculation is as follows:

$$log_2(N) \leq S_{opt} \leq 2log_2(N), \tag{10}$$

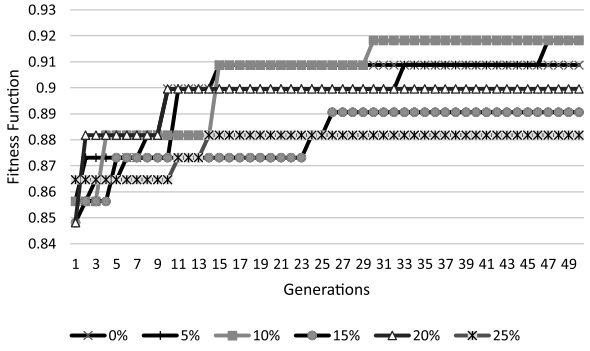

Fig. 15. Fitness functions for the best matched individuals for different percentages of new individuals.

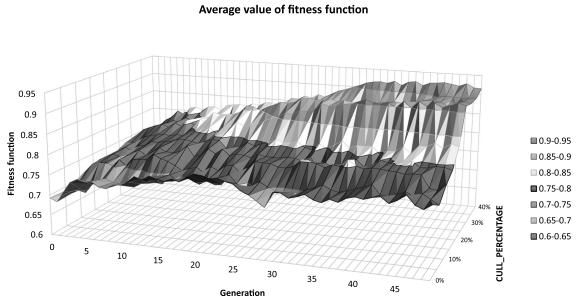

Fig. 16. Average fitness functions for different percentages of eliminated items.

where N corresponds to the number of all combinations of genes.

Assuming that the length of the ICR sequence is equivalent to $SICL = 2880$ and the number of all tasks assigned to a given core $n = 60$, the resulting calculation is as follows:

$$N = \frac{(SICL + n - 1)!}{SICL!(n-1)!} \approx 1.69 \cdot 10^{124}. \tag{11}$$

In view of this, based on the formula (10), the optimal population size is in the range

$$413 \leq S_{opt} \leq 826. \tag{12}$$

To confirm this assumption, a series of experiments were carried out, the results of which are shown in the graph in Fig. 20. However, due to the long calculation time (see Fig. 10), the maximum population size was assumed to be 100.

At the conclusion of this experiment, it should be noted that the number of iterations set at 50 proved to be sufficient. This is evidenced by the observation that, in the majority of the examined cases, the population attained a minimum value after approximately forty iterations.

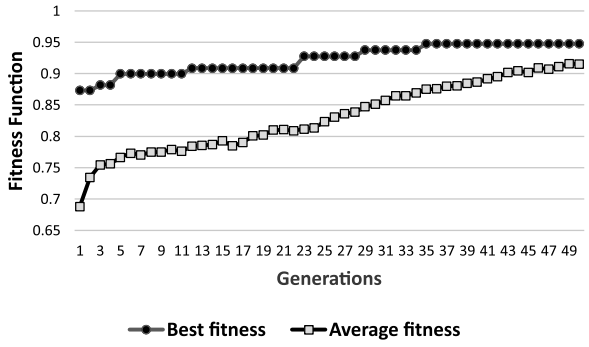

Fig. 17. Best and average values of the fitness function for $CULL\_PERCENTAGE = 40\%$.

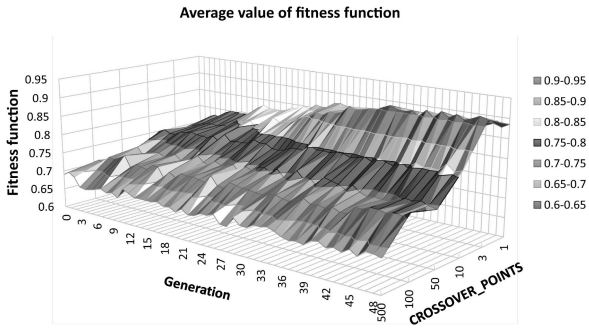

Fig. 18. Average fitness functions for different numbers of crossover points.

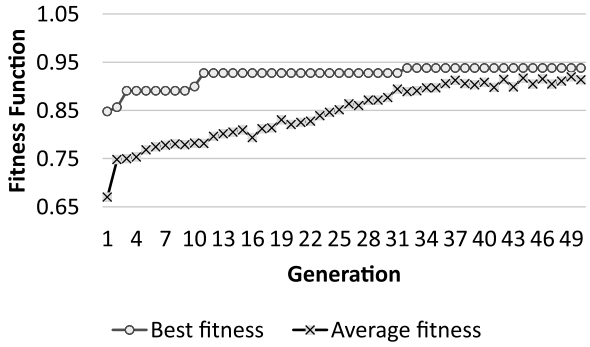

Fig. 19. Best and average values of fitness function for $CROSSOVER\_POINTS = 3$.

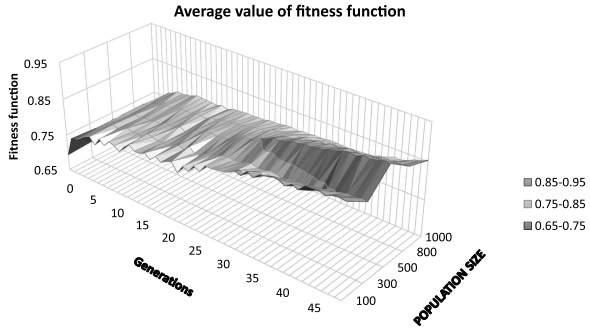

Fig. 20. Average fitness functions for different numbers of crossover points.

5.2. Prolog implementation of the proportional algorithm. The proportional method was implemented in the LPA Prolog language, which is consistent with the programming standard in logic (Clocksin and Mellish, 2003). This tool offers very useful mechanisms for processing lists in a very easy and transparent way. Lists have proven to be a very convenient way to represent a sequence of task identifiers, that is, an ICR sequence.

As demonstrated by the conducted experiments, in most cases the results obtained were superior to those of the reference method. The proportional method is the optimal tool for configuring the ICR, taking into account the short processing time, which did not exceed a few seconds, depending on the number of tasks mapped on a given core.

Table 3 compiles the results obtained for various optimization scenarios, employing different configurations of the number of cores and the number of tasks assigned to them. Our previous paper (Antolak and Pułka, 2023) used terminology related to optimization goals of minimizing power or resources, while the current publication focuses on the number of tasks and their timing parameters, which more accurately capture the essence of the problem under consideration, the optimal configuration of the ICR sequence.

Table 3. Results obtained with the proportional algorithm for different numbers of tasks.

Tasks	TS [MHz]	$SICL$	$T_{\min}$ [MHz]
12	15.67	1023	18
18	22.51	1602	35
22	29.51	1988	36
29	44.22	2876	45
31	44.23	2857	46
40	60.62	2872	62
50	74.01	2853	76
60	89.06	2881	91

It should be noted that, for a small number of tasks, if the required value of $Min_{indistance}$ needs to be reduced to avoid idle cycles in pipeline processing, it is necessary to reduce the length of the pipeline. This approach also allows an efficient implementation of the tail-head subsequence correction (see Step 7 of Algorithm 2), since it is possible that the potential list of insertable identifiers is insufficient. In such a case, it would be necessary to add idle cycles to the end of the sequence.

As mentioned above, since the value of TS can

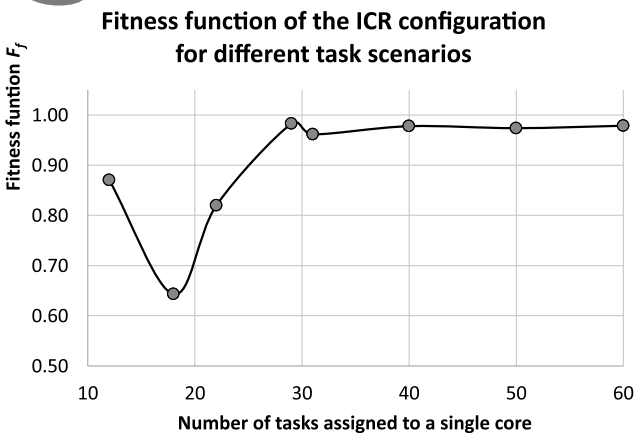

Fig. 21. Fitness function of the ICR configurations obtained by the proportional method for different task scenarios.

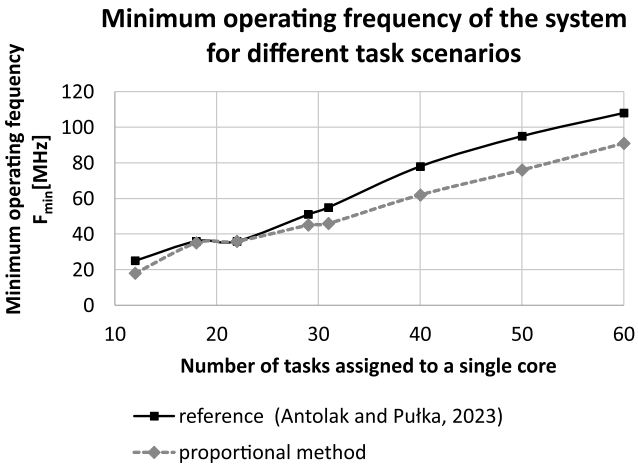

Fig. 22. Minimal operating frequencies obtained by the reference and proportional method for different task scenarios.

be used as a benchmark, an indicator identical to that determined by the fitness function F_f for the evolutionary method can be used to evaluate the quality of the obtained results. As the graph in Fig. 21 illustrates, the values obtained are close to unity. However, in the case of lightly loaded cores, when the number of tasks is relatively small, it appears that the system must be clocked at a noticeably higher frequency to achieve 100% time predictability. This issue is also discussed in the summary of the paper.

In Fig. 22, a direct comparison is made between the results obtained with the proportional method presented here and the original one by Antolak and Pułka (2023). The results suggest that the proportional method has considerable potential, as it often allows achieving significantly lower operating frequencies. However, it is important to note that the graph also indicates some challenges related to the configuration of the ICR in both methods for cases of weak core load.

5.3. Comparison of both approaches. The graphs (Figs. 23 and 24) summarize the comparative results of the two methods. The initial illustration (Fig. 23) presents the absolute results of the evaluated benchmarks in relation to the reference solution. The graph depicted in Fig. 24 compares the fitness function of evolutionary and proportional methods. The obtained results demonstrate a clear superiority of the dedicated solution, i.e., the proportional method, which effectively tracks the progress of tasks and depends on the order of their processing in the pipeline. The use of a method characterized by substantial temporal investment may be a viable solution in scenarios where the number of tasks is relatively small compared to the number of stages in the core pipeline.

5.4. Verification and validation of the approach. The verification of the system's operation and the evaluation of the quality of the solutions were carried out in the implemented WCTA. This analysis allowed the determination of the minimum operating frequency of the system. It was concluded that this frequency was necessary for all tasks to meet their deadline requirements. To further substantiate the discussed methodologies, hardware implementations of individual solutions in FPGAs were employed. Virtex 7 and Kintex Ultrascale are two distinct examples of this phenomenon. In the former case, the maximum number of concurrently processed tasks reached 256 at a maximum operating frequency of 200 MHz. The Kintex Ultrascale chip, despite its technological superiority, exhibits a comparatively reduced capacity, allowing approximately 60 tasks in practice. The latter possesses the advantage of being able to accurately measure dynamic power; however, this is not the subject of this paper.

6. Summary

This paper presented two novel alternative approaches to generating an ICR sequence, which defines the order and frequency of execution of tasks in a pipelined processor running interleaved hardware threads in the regime of time predictability. Both algorithms were thoroughly tested and compared with the original approach. In the case of the genetic algorithm, the worst results were obtained. It was observed that, particularly in cases involving a substantial number of iterations and a population size, this algorithm exhibited a general linear computational complexity of $O(n)$, necessitating the most resources and time. It is acknowledged that computation time is not a critical factor in static (offline) scheduling, particularly given the necessity of executing multiple WCTA analyses to determine the value of the fitness function. However,

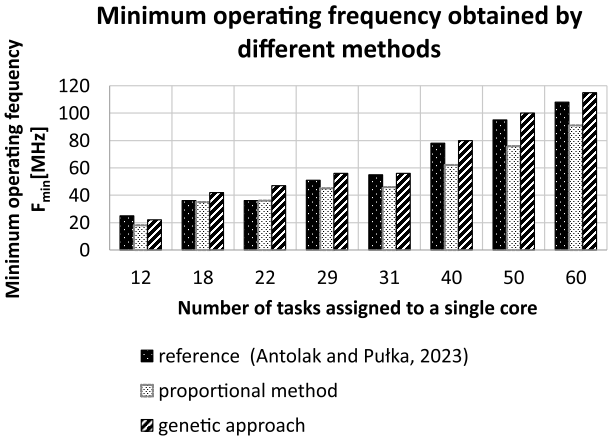

Fig. 23. Comparison of minimal operating frequencies obtained by all discussed methods for different task scenarios.

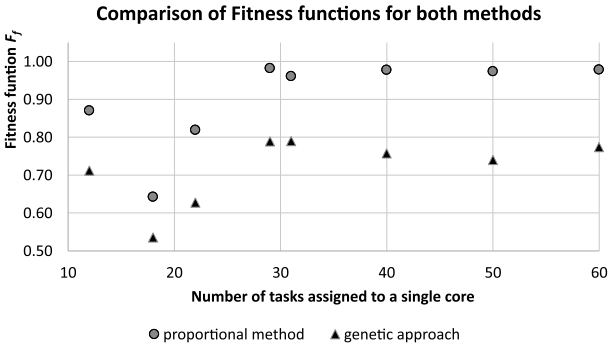

Fig. 24. Comparison of the fitness function values of the ICR configurations obtained by both methods for different task scenarios.

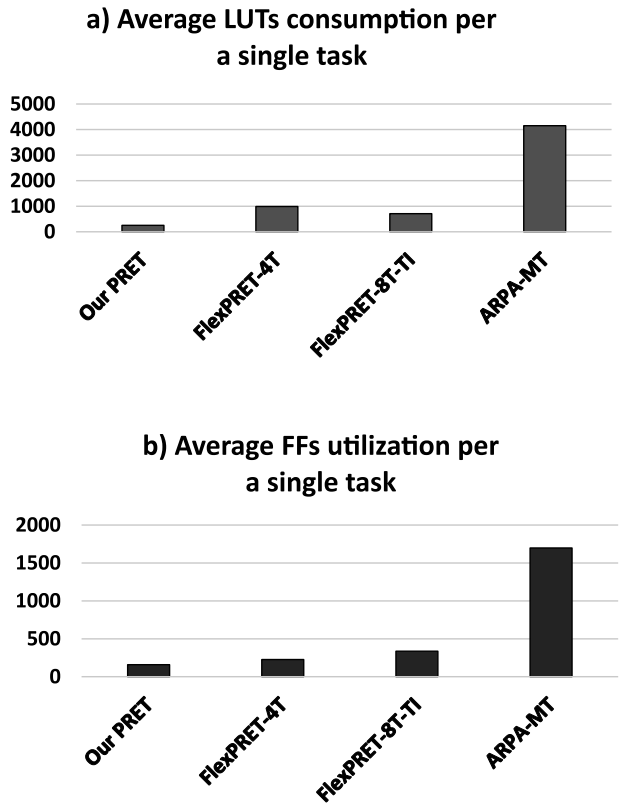

Fig. 25. Comparison of FPGA resource consumption for various time-predictable platforms related to a single task.

the amount of memory resources can impose a substantial constraint. Furthermore, mutations were observed to be not advantageous as a result of their inability to improve the solutions obtained. Experiments conducted on short ICR sequences and cores, which process a relatively modest number of tasks in relation to the size of the pipeline, demonstrated that the genetic method can be regarded as a viable alternative. Specifically, for short sequences, the computational time is notably reduced.

The second method, the proportional one, was found to be a highly effective solution, despite its increased computational complexity $k\cdot$ O$(n \cdot ln(n))$, which is mainly attributed to the need to perform sorting operations multiple times. The calculation times for this method are significantly shorter, with a maximum duration of a few seconds, depending on the PC used and the $SICL$ values. It was observed that, in the case of SHT tasks, the optimal outcome was achieved when the TF values of these tasks were maximized, facilitating a more efficient distribution of the most critical tasks throughout the ICR sequence. A comparative analysis of all the results obtained against the values of TS was conducted, which revealed the existence of some reserves and possible opportunities to improve the ICR configuration algorithms. This topic will be continued in the future research.

It is important to note the implementation of the WCTA, which was incorporated into the designed architecture of the real-time system. This analyzer facilitates a precise analysis of the time dependencies of the executed tasks.

A direct comparison with other approaches is quite difficult due to the variety of existing solutions. Table 4 compiles a series of properties pertaining to analogous interleaved pipeline solutions: FlexPRET (Zimmer *et al.*, 2014), ARPA-MT (Oliveira *et al.*, 2011) and PTARM (Liu *et al.*, 2010), and undertakes a comparative analysis with the presented architecture. While the diagram depicted in Fig. 25 shows the average hardware resource consumption per single task for different solutions, this paper is not concerned with this aspect, but the presented comparison shows the advantages of the discussed architecture. Our PRET architecture has been demonstrated to consume a comparatively modest amount of resources. In contemporary multitasking systems, the

Table 4. Comparison of different time-predictable platforms implemented in FPGAs.

Approach	Platform	RTOS	Type of tasks	Number of cores
Our PRET	Virtex 7	Hardware (ICR) mimics RTOS	NCT, CT, SHT	Configurable(1 to 8)
FlexPRET	Virtex 5	Hardware based isolation and context switch	HRTT, SRTT	Multicore
ARPA-MT	Virtex 5	Dedicated coprocessor (hardware)	Periodic and jitter tasks	Single core
PTARM	Spartan 3	Not stated	Independent, parallel tasks	Single core

objective is to minimize power dissipation within the system. A viable method for achieving this objective is to impose constraints on the operating frequency of the system. The methodologies delineated in this study constitute an integral component of the prevailing research paradigm.

Acknowledgment

This research was funded by the Polish Ministry of Science and Higher Education under statutory activities (BK-250/RAu-11/2025).

References

Ahn, C.W. and Ramakrishna, R. (2003). Elitism-based compact genetic algorithms, *IEEE Transactions on Evolutionary Computation* **7**(4): 367–385.

Akesson, B. and Goossens, K. (2012). *Memory Controllers for Real-Time Embedded Systems*, Springer, New York, DOI: 10.1007/978-1-4419-8207-0.

Alander, J. (1992). On optimal population size of genetic algorithms, *CompEuro 1992: Proceedings Computer Systems and Software Engineering, The Hague, Netherlands*, pp. 65–70.

AlBarakat, L.M., Gratz, P.V. and Jimenez, D.A. (2018). MTB-Fetch: Multithreading aware hardware prefetching for chip multiprocessors, *IEEE Computer Architecture Letters* **17**(2): 175–178.

Alhammad, A. and Pellizzoni, R. (2014). Time-predictable execution of multithreaded applications on multicore systems, *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, Dresden, Germany*, pp. 1–6.

Andalam, S., Roop, P.S., Girault, A. and Traulsen, C. (2014). A predictable framework for safety-critical embedded systems, *IEEE Transactions on Computers* **63**(7): 1600–1612.

Antolak, E. and Pułka, A. (2021). Energy-efficient task scheduling in design of multithread time predictable real-time systems, *IEEE Access* **9**: 121111–121127.

Antolak, E. and Pułka, A. (2020). Flexible hardware approach to multicore time predictable systems design based on the interleaved pipeline processing, *IET Circuits, Devices & Systems* **14**(5): 648–659, DOI: 10.1049/iet-cds.2019.0521.

Antolak, E. and Pułka, A. (2022). An analysis of the impact of gating techniques on the optimization of the energy dissipated in real-time systems, *Applied Sciences* **12**(3): 1630.

Antolak, E. and Pułka, A. (2023). Validation of task scheduling techniques in multithread time predictable systems, *IEEE Access* **11**: 46979–46997.

Antolak, E. and Pułka, A. (2024). Power consumption prediction in real-time multitasking systems, *Electronics* **13**(7): 1347–1366.

Axer, P., Ernst, R., Falk, H., Girault, A., Grund, D., Guan, N., Jonsson, B., Marwedel, P., Reineke, J., Rochange, C., Sebastian, M., Hanxleden, R.V., Wilhelm, R. and Yi, W. (2014). Building timing predictable embedded systems, *ACM Transactions on Embedded Computing Systems* **13**(4): 1–37, DOI: 10.1145/2560033.

Bahn, H. and Cho, K. (2020). Evolution-based real-time job scheduling for co-optimizing processor and memory power savings, *IEEE Access* **8**: 152805–152819.

Bratko, I. (2012). *Prolog Programming for Artificial Intelligence*, 4th Edn, Addison-Wesley, Harlow.

Broman, D., Zimmer, M., Kim, Y., Kim, H., Cai, J., Shrivastava, A., Edwards, S.A. and Lee, E.A. (2013). Precision timed infrastructure: Design challenges, *Proceedings of the 2013 Electronic System Level Synthesis Conference (ESLsyn), Austin, USA*, pp. 1–6.

Buttazzo, G.C. (2011). *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Springer US, Boston, DOI: 10.1007/978-1-4614-0676-1.

Cazorla, F., Knijnenburg, P., Sakellariou, R., Fernandez, E., Ramirez, A. and Valero, M. (2006). Predictable performance in SMT processors: Synergy between the OS and SMTs, *IEEE Transactions on Computers* **55**(7): 785–799.

Chen, J., Du, C., Han, P. and Zhang, Y. (2019). Sensitivity Analysis of strictly periodic tasks in multi-core real-time systems, *IEEE Access* **7**: 135005–135022.

Chniter, H., Mosbahi, O., Khalgui, M., Zhou, M. and Li, Z. (2020). Improved multi-core real-time task scheduling of reconfigurable systems with energy constraints, *IEEE Access* **8**: 95698–95713.

Clocksin, W. and Mellish, C. (2003). *Programming in Prolog*, Springer, Berlin/Heidelberg.

Edwards, S.A. and Lee, E.A. (2007). The case for the precision timed (PRET) machine, *2007 44th ACM/IEEE Design Automation Conference, San Diego, USA*, pp. 264–265.

Fernández, M., Gioiosa, R., Quiñones, E., Fossati, L., Zulianello, M. and Cazorla, F.J. (2012). Assessing the suitability of the NGMP multi-core processor in the space domain, *Proceedings of the 10th ACM International Conference on Embedded Software, Tampere, Finland*, pp. 175–184, DOI: 10.1145/2380356.2380389.

Forsberg, B., Benini, L. and Marongiu, A. (2018). HePREM: Enabling predictable GPU execution on heterogeneous SoC, *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany*, pp. 539–544.

Gajski, D.D., Abdi, S., Gerstlauer, A. and Schirner, G. (2009). *Embedded System Design: Modeling, Synthesis and Verification*, Springer US, Boston, DOI: 10.1007/978-1-4419-0504-8.

Glaser, F., Tagliavini, G., Rossi, D., Haugou, G., Huang, Q. and Benini, L. (2021). Energy-efficient hardware-accelerated synchronization for shared-L1-memory multiprocessor clusters, *IEEE Transactions on Parallel and Distributed Systems* **32**(3): 633–648.

Goldberg, D.E. and Goldberg, D.E. (2012). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Boston.

Henzinger, T.A. and Kirsch, C.M. (2002). The embedded machine: Predictable, portable real-time code, *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation, Berlin, Germany*, pp. 315–326, DOI: 10.1145/512529.512567.

Ip, N.J.H. and Edwards, S.A. (2006). A processor extension for cycle-accurate real-time software, *in* D. Hutchison *et al.* (Eds), *Embedded and Ubiquitous Computing*, Springer, Berlin/Heidelberg, pp. 449–458, DOI: 10.1007/11802167_46.

Kim, D., Ko, Y.-B. and Lim, S.-H. (2020a). Energy-efficient real-time multi-core assignment scheme for asymmetric multi-core mobile devices, *IEEE Access* **8**: 117324–117334.

Kim, Y., Kong, J. and Munir, A. (2020b). CPU-accelerator co-scheduling for CNN acceleration at the edge, *IEEE Access* **8**: 211422–211433.

Kochenderfer, M.J. and Wheeler, T.A. (2019). *Algorithms for Optimization*, MIT Press, Cambridge.

Lamie, E.L. (2009). *Real-Time Embedded Multithreading Using ThreadX*, 2nd Edn, Newnes, Amsterdam.

Lee, E. (2005). Absolutely positively on time: What would it take?, *Computer* **38**(7): 85–87.

Lee, E. (2006). The problem with threads, *Computer* **39**(5): 33–42.

Lee, E. and Messerschmitt, D. (1987). Pipeline interleaved programmable DSP's: Architecture, *IEEE Transactions on Acoustics, Speech, and Signal Processing* **35**(9): 1320–1333.

Lickly, B., Liu, I., Kim, S., Patel, H.D., Edwards, S.A. and Lee, E.A. (2008). Predictable programming on a precision timed architecture, *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, Atlanta, USA*, pp. 137–146, DOI: 10.1145/1450095.1450117.

Liu, I., Reineke, J. and Lee, E.A. (2010). A PRET architecture supporting concurrent programs with composable timing properties, *2010 Conference Record of the 44th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, USA*, pp. 2111–2115.

Lutz, M. (2019). *Programming Python*, 4th Edn, O'Reilly, Beijing.

Michalak, K. (2021). Evolutionary algorithm using random immigrants for the multiobjective travelling salesman problem, *Procedia Computer Science* **192**: 1461–1470.

Moulik, S., Devaraj, R. and Sarkar, A. (2018). COST: A cluster-oriented scheduling technique for heterogeneous multi-cores, *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan*, pp. 1951–1957.

Oliveira, A.S.R., Almeida, L. and Ferrari, A.D.B. (2011). The ARPA-MT embedded SMT processor and its RTOS hardware accelerator, *IEEE Transactions on Industrial Electronics* **58**(3): 890–904.

Paolieri, M., Quinones, E., Cazorla, F.J., Wolf, J., Ungerer, T., Uhrig, S. and Petrov, Z. (2011). A software-pipelined approach to multicore execution of timing predictable multi-threaded hard real-time tasks, *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, Newport Beach, USA*, pp. 233–240.

Pathan, R., Voudouris, P. and Stenstrom, P. (2018). Scheduling parallel real-time recurrent tasks on multicore platforms, *IEEE Transactions on Parallel and Distributed Systems* **29**(4): 915–928.

Rehman, A.U., Ahmad, Z., Jehangiri, A.I., Ala'Anzy, M.A., Othman, M., Umar, A.I. and Ahmad, J. (2020). Dynamic energy efficient resource allocation strategy for load balancing in fog environment, *IEEE Access* **8**: 199829–199839.

Reineke, J., Liu, I., Patel, H.D., Kim, S. and Lee, E.A. (2011). PRET DRAM controller: Bank privatization for predictability and temporal isolation, *Proceedings of the 7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Taipei, Taiwan*, pp. 99–108, DOI: 10.1145/2039370.2039388.

Schoeberl, M. (2008). A Java processor architecture for embedded real-time systems, *Journal of Systems Architecture* **54**(1-2): 265–286.

Schoeberl, M., Abbaspour, S., Akesson, B., Audsley, N., Capasso, R., Garside, J., Goossens, K., Goossens, S., Hansen, S., Heckmann, R., Hepp, S., Huber, B., Jordan, A., Kasapaki, E., Knoop, J., Li, Y., Prokesch, D., Puffitsch, W., Puschner, P., Rocha, A., Silva, C., Sparsø, J. and Tocchi, A. (2015). T-CREST: Time-predictable multi-core architecture for embedded systems, *Journal of Systems Architecture* **61**(9): 449–471.

Schoeberl, M., Schleuniger, P., Puffitsch, W., Brandner, F. and Probst, C.W. (2012). Towards a time-predictable dual-issue microprocessor: The Patmos Approach, *Open Access Series in Informatics* **18**: 11–21.

Thiele, L. and Wilhelm, R. (2004). Design for timing predictability, *Real-Time Systems* **28**(2/3): 157–177, DOI: 10.1023/B:TIME.0000045316.66276.6e.

Truś, B. (2023). *Implementation of the Scheduling Mechanisms of Tasks in a Multicore Real Time System*, Silesian University of Technology, Gliwice, (in Polish).

Ungerer, T., Cazorla, F., Sainrat, P., Bernat, G., Petrov, Z., Rochange, C., Quinones, E., Gerdes, M., Paolieri, M., Wolf, J., Casse, H., Uhrig, S., Guliashvili, I., Houston, M., Kluge, F., Metzlaff, S. and Mische, J. (2010). MERASA: Multicore execution of hard real-time applications supporting analyzability, *IEEE Micro* **30**(5): 66–75.

Zimmer, M., Broman, D., Shaver, C. and Lee, E.A. (2014). FlexPRET: A processor platform for mixed-criticality systems, *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), Berlin, Germany*, pp. 101–110.

Andrzej Pułka received his MSc, PhD and DSc degrees in electronics from Silesian Technical University, Gliwice, Poland, in 1988, 1997 and 2013, respectively. Currently, he is a university professor there and the head of the Department of Electronics, Electrical Engineering and Microelectronics. He is an author and coauthor of approximately 100 scientific papers, including journal articles, book chapters, and conference papers. His research interests cover automated design of digital and mixed signal circuits in FPGAs, modeling and simulation of electronic embedded systems, VHDL, Verilog, SystemVerilog, SystemC, real-time systems: precision time machines (PRET), design of energy efficient systems, power optimization in SoC, AI and commonsense reasoning modeling, and applications of FPGA platforms for hardware acceleration of complex computations in bioinformatics. He is an IEEE senior member and a member of the Electronics Commission of the Polish Academy of Sciences.

Ernest Antolak received his MSc in electronics and telecommunications engineering and his PhD in automation, electronics, and space technologies from the Silesian University of Technology, Gliwice, Poland, in 2018 and 2022, respectively. He currently holds a position at that university, where he is involved in research and teaching activities. His professional interests include real-time scheduling, hardware description languages, design safety, cyber-physical systems, systems on chips, and energy-efficient systems.

Bartłomiej Truś graduated with an MSc in electronics and telecomunication from the Silesian University of Technology, Gliwice, Poland, in 2023. He is professionally involved with software development for ADAS domain ECUs in automotive industry.

Appendix

Some examples and codes are available in the GitHub repository at `https://github.com/apulka64/AMCS25`.

Received: 25 March 2025
Revised: 21 September 2025
Accepted: 30 October 2025