Received 21 April 2023, accepted 6 May 2023, date of publication 11 May 2023, date of current version 17 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3275437

RESEARCH ARTICLE

Validation of Task Scheduling Techniques in Multithread Time Predictable Systems

ERNEST ANTOLAK AND ANDRZEJ PUŁKA, (Senior Member, IEEE)
Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, 44-100 Gliwice, Poland

Corresponding author: Ernest Antolak (ernest.antolak@polsl.pl)

This work was supported in part by the European Social Funds through ''CyPhiS—the program of modern Ph.D. studies in the field of cyber-physical systems'' under Project POWR.03.02.00-00-I007/17-00, and in part by the Ministry of Science and Higher Education under Grant BKM-573/RAu-11/2022 and Grant BK-246/RAu-11/2022.

ABSTRACT This paper presents a simulation-based environment for verification of static task scheduling methodology in a time predictable system. Different types of processed tasks are distinguished and presented a unified system design methodology consisting of the selection of real time system configuration, task mapping, scheduling, and generation of a sequence of task identifiers to control the interleaved pipeline. An original Worst Case Timing Analyzer (WCTA) has been developed to automate the design process. The methodology was introduced into the original PRET (PREcision Timed) architecture recently presented (Antolak and Pulka, 2020), (Antolak and Pulka, 2021). The PRET system was implemented on a Virtex7 FPGA (Field Programmable Gate Array) platform. A dedicated verification environment is proposed that allows on-line real time system monitoring, analysis of timing parameters, and comparing the results with initial requirements and design constraints. The practical experiments presented in the paper proved the correct operation of the author's hardware architecture. The obtained results confirmed the validity of the proposed scheduling method and the concept of calculating the execution times of tasks before they are started, which allows for optimal hardware matching to the tasks to be performed.

INDEX TERMS Real-time systems, timing simulation, dynamic scheduling, multitasking, pipeline interleaving, multithreading.

I. INTRODUCTION

Contemporary processors and multiprocessor systems are capable of processing very complex software algorithms, but they exhibit a very high degree of hardware complexity. In addition, the rapid development of semiconductor technology means that these systems are clocked with ever faster clock signals. Paradoxically, these technological advances are causing timing predictability issues to arise for such systems [3]. Initially, problems with processor timing predictability were solved at the software layer, creating real-time operating systems, or so-called RTOSs (Real-Time Operating System) [4], [5]. Unfortunately, this approach to the problem is fraught with a great deal of inaccuracy, since task handling is done at a very high level of abstraction [6], [7]. Most programming languages such as C [8], or C++ do not give direct control over the elapsed time of tasks [9], which, combined with a large number of abstraction levels, allows only very rough control of timing [10], [11], [12].

The paper discusses scheduling methodologies in the system presented in [2]. It focused on the generation of task identifiers sequence that is the key issue of interleaved pipeline processing of threads. We introduced a new category of tasks with the highest priority, called strong hard timed tasks and we adjusted the scheduling process according to the type of task. It was shown that the appropriate order and frequency of tasks in a core's pipeline decides overall system efficiency and predictability. The proposed real-time system was implemented in the Verilog language, and its synthesis and implementation were carried out in the Vivado 2018.3 environment. Full simulation of such a system consumes a huge amount of time, and it is practically impossible to get a complete picture of the system's behavior. Mapping the hourly operation of

The associate editor coordinating the review of this manuscript and approving it for publication was Laxmisha Rai.

a system implemented in an FPGA requires more than a week of simulation. Hence, the idea of developing a dedicated program to enable efficient time analysis of the developed system emerged. Moreover, an offline Worst Case Timing Analyzer (WCTA) dedicated to the designed system was created. WCTA delivers many timing parameters that validate the developed methodology. It was assumed that regardless of when a task is started, it should be completed within the deadline. The proposed WCTA analyzer is based on a time model of the pipelined processing path with interleaved threads. The authors presented the subsequent stages of this processing in the paper. A series of experiments proved the correctness of the approach.

The main contribution of this paper is:

- Proposing an Interleaving Cycle Register (ICR) of task identifier sequences;
- Developing an extended task model;
- Dividing the task scheduling process into stages;
- Unifying the task mapping method - developing a universal BLTS (Balanced Load Task Scheduling) algorithm;
- Proposing a method of task scheduling by composing the contents of the ICR (hardware alternative for RTOS);
- Developing a pre-layout Worst Case Timing Analyzer (WCTA) to perform a timing analysis of the designed system, and in particular to estimate the execution times of individual tasks;
- Final hardware validation of the system.

The paper consists of six sections. First, the related work is briefly discussed, then the main elements of the proposed multitask system architecture and assumed tasks model are recalled. In the fourth section the main stages of the scheduling process implemented in the methodology are described. Section five covers the experiments and discussion of the obtained results. The paper is summarized with the final conclusions.

II. RELATED WORK

The idea of time-predictable systems, the PREcision Timed machines (PRETs), was formulated and presented as early as 2007 by Edwards and Lee [13]. It assumed that such a system should always, regardless of load, perform scheduled tasks on time. The concept was developed by many researchers: Thiele and Wilhelm [14] formulated a set of recommendations and guidelines to facilitate the design of time-critical embedded systems. Ip and Edwards [15] proposed extending the command list of RISC processors to include a DEADLINE instruction to enable time-critical task control. The CHESS (Center for Hybrid and Embedded Software Systems) [16] group at UC Berkeley, led by Prof. Edward Lee, developed the idea of the PRET processor, and proposed, among other things, a way to predictably access memory by proposing a method called "memory wheel". Then, the CHESS group, in cooperation with centers from Germany (Saarland University) and Sweden (Linköping University), developed their ideas. This work resulted in the 2010 publication [43] of the new PRET architecture that supported the concurrent execution of programs. This architecture allowed integrating independent components maintaining temporal properties of tasks. This solution used threads' interleaving mechanism, scratchpad memories, and a composable and predictable DRAM (dynamic random-access memory) controller. Another paper from this research team [44] presented a substantial implementation of a precision-timed machine – PTARM architecture. This architecture is based on the ARM family processors and implements a subset of ARMv4 microarchitecture. The PTARM solution improved system performance by using a refined thread-interleaved pipeline, an exposed memory hierarchy, and a repeatable DRAM memory controller. In 2014, the CHESS group developed a new platform for processing mixed-criticality tasks called FlexPRET [42]. This paper distinguished hard-time threads (HRTT) and soft real-time threads (SRTT). The FlexPRET architecture generated by the CHISEL tool [46] provided hardware-based isolation to HRTT tasks, while SHTT tasks efficiently utilized available processing resources. The authors of [42] introduced dedicated timing instructions to ISA based on the RISC-V processors family and proposed thread scheduler that kept control over the threads processed by the pipeline. The entire structure was also implemented in a FPGA device.

The paper [8] addressed the analysis of time-predictable systems at a higher level of abstraction. The authors of this work introduce a new lightweight and concurrent language, PRET-C (Precision Time version of C). PRET-C, thanks to its syntax, synchronous semantics, and very simple mechanisms handling time, is well suited for predictable PRET architectures. The authors also proposed a hardware accelerator for PRET-C execution over soft-core processors allowing time-predictable execution of tasks with high efficiency. This time-predictable architecture was called ARPRET.

Yet another interesting solution is presented in another paper [45], in which the authors presented their own time-predictable architecture called ARPA-MT. ARPA-MT architecture consists of 3 main elements: the main processing unit, two coprocessors Cop0-MEC responsible for memory operations management and exceptions and interrupts handling, and Cop2-MEC implementing and accelerating RTOS functions in hardware. The ARPA-MT [45] structure contains a very interesting 5-staged pipeline with the first two stages (IF and ID) replicated for each thread. While the other 3 stages of the pipeline (EX, MA and WB) process different threads that are interleaved. This solution presented interesting results of hardware-software synergy while designing real-time systems.

A group of researchers from Denmark, Austria, France, and the US presented the new original architecture of the multi-core processor called Patmos [17]. Then in 2015, a group of 24 European researchers presented the T-CREST project [3], which demonstrated a multi-core approach to the original PRET system concept. Problems arising from

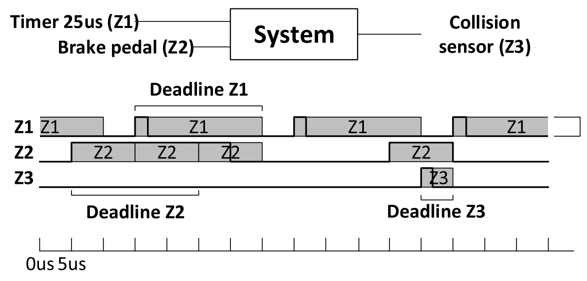


FIGURE 1. An example of the system run.

scheduling tasks running in parallel in multicore systems were published in [18], [19], [20], [21], [22], and [23], along with attempts to solve them.

It is noticeable that the research carried out on time predictable systems for many years involved the analysis of very different issues related to both hardware [12], [14], [17], [24], [25], [26] and software [18], [19], [27], [28] development. Researchers often look for general solutions [8], [9], but many works concern dedicated solutions [17], [25]. In [29] authors provide an overview of many issues related to the design of embedded time-predictable systems.

Authors of some papers [30], [31] analyzed the problem of load-balanced scheduling and proposed techniques enabling reduction of the energy consumed by the systems. In [32] one can find a technique of the effective utilization of processing elements in the clusters of processors with the shared L1 cache memory based on the optimized synchronization and communication between the system components. Some works demonstrated the application of heuristic methods and AI algorithms in the task scheduling process [33], [34], [35].

In the paper [1], an original real-time system solution was proposed. It was based on thread interleaved pipeline processing. The solution allows flexible configuration of the pipeline and uses a set of dedicated scheduling algorithms. In a subsequent paper [2], the problem of energy optimization in time-predictable systems was analyzed and the authors proposed a new solution of this problem in the form of enhanced and dedicated scheduling algorithms used for various design goals and constraints. In the presented paper, the problem of generating sequences of thread identifiers in the process of scheduling tasks of different types was more thoroughly addressed. The methodology was based on analyzing the timing conditions on a specially constructed Worst Case Timing Analyzer (WCTA), which makes it possible at the system design stage to accurately analyze the timing of the system. Unlike other approaches where the authors analyze Deadline Miss Ratio (DMR) [36], this solution does not allow any task to exceed the deadline time.

III. OVERWIEW OF THE SYSTEM

Every task executed in the system must be completed before its strictly defined execution time (deadline) regardless of the system load. Such tasks are inherently asynchronous since they can be triggered by external interrupts, the occurrence of which is very difficult or even impossible to predict. Before starting the system, only the number and type of tasks being executed are known, while the number of different tasks and the timing of their startup are unknown. Therefore, to ensure the above assumptions, it is necessary to find the most heavily loaded task sequence on the system and prove before system startup that the task will be executed on time.

As an example of the application of such a system, one can recall an automotive safety system (see the diagram in Fig. 1). Such a system can run a cyclic task, called every 25 μs by a hardware timer (Task Z1). This task calculates the approximate braking distance at a given speed and weather conditions, and all operations must be completed in less than 20 μs. Additionally, each time the driver presses the brake, the system must calculate whether, and if so, which wheels have skidded. To make sure that safety systems such as ABS (Anti-lock Braking System) or traction control react correctly, this event must be detected no later than, for example, 10 μs after the brake pedal is pressed (see Task Z2). The wheels can also get into a skid after braking has started, so checking whether the wheels have gotten into a skid should take place all the time the brake is depressed (high state of the signal Z2).

A. SYSTEM ARCHITECTURE

The system architecture uses the thread interleaving method to avoid data and control hazards. In that case, there is no need for complex forwarding and jump prediction circuits. The thread interleaving method has been extended by the authors [1], [2] and has also been used to exchange data between tasks. The proposed architecture is based on pipeline processing. It can consist of 1 to 8 reconfigurable cores. It is possible to reconfigure the pipelines, which can contain from 5 to 12 stages [2], depending on application requirements. As described in [2], the basic five stages of the pipeline can be expanded to include sub-stages. Thus, the IF (Instruction Fetch) stage can contain three sub-stages (Select Bank and Instruction Address, Instruction Fetch, Select Instruction); the ID (Instruction Decode) stage can be divided into two sub-stages (Select General Purpose Register Bank, Instruction Decode); the EXE (Execution) stage can be expanded to three processing cycles (Shift, ALU, EXE); similarly, the MA (Memory Access) stage can also be expanded to three cycles (Select Bank and MEM Data Address, MEM Data Fetch, Select MEM Data).

Interleaving threads requires the introduction of multiplexers to switch task data, therefore the authors proposed to place the multiplexers in separate stages of the pipeline. This minimizes the impact of the number of tasks on the maximum operating frequency of the system. The microarchitecture of our system is modeled after the ISA (Instruction Set Architecture) of the ARM processor family. ISA was enhanced with special timing instructions [1]:

- **AD counter_number** – the instruction activating the appropriate deadline counter;
- **DD counter_number**– the instruction deactivating the appropriate deadline counter;

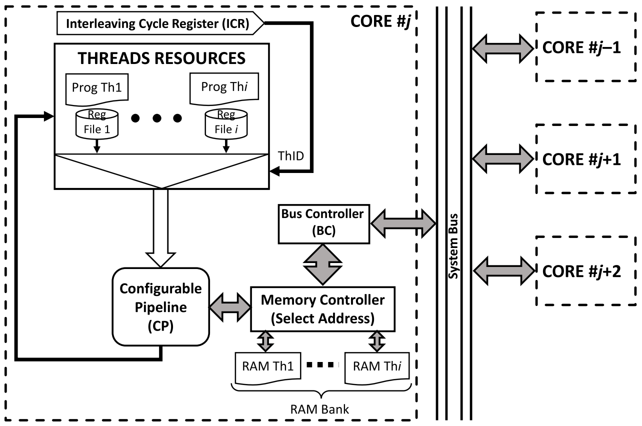


FIGURE 2. The structure of the core.

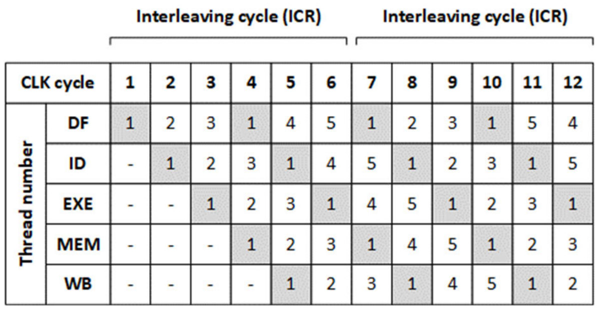
- **SD counter_number value** – the instruction loading the value to the deadline counter (setting the deadline);
- **WFD counter_number value** – the instruction suspending the execution of the program until the deadline counter reaches 1.

The system structure consists of one or more cores connected to a common bus. The bus is used to exchange information between threads. The system's cores communicate with external peripherals via input/output ports, which are mapped as part of the data memory of each thread. The system's cores communicate with external peripherals via input/output ports, which are mapped as part of the data memory of each thread. Every thread can have its own individual input/output ports. In addition, each thread is started by a corresponding individual external signal. The procedure of launching a given thread can be implemented either hardware-wise (using an external interrupt) or software-wise by executing the corresponding instruction by another processor thread. The system architecture is based on the authors' previous solutions [1], [2], [29]. The current implementation of the system allows simultaneous processing of up to 255 different tasks, with theoretically any distribution of these tasks between cores (the theoretical maximum number of cores is also 255).

A configurable number of supported threads are provided for every core. Because these threads are not dependent on each other, they must have their own independent resources such as memory and register files. In order to make the cores as compact as possible, only these independent resources shall be additional elements dedicated to a thread. The remaining elements of the core, such as the processing pipeline, are shared by threads mapped and executed in a single core. Fig. 2 presents the structure of a configurable core. The size of a core depends on the number of threads processed in a given core. For this purpose, a special parameter is defined for every core. The value of this parameter determinates the number of register files, memory banks and size of the multiplexors responsible for switching threads' resources. The rest of the core is designed to be independent of this parameter. So, the increase in resource requirements as the number of supported threads rises is minimal. If the system is built of several cores, some tasks can be processed concurrently. Moreover, the configurable cores may differ in their internal structures, so the hardware architecture of a designed system can be adjusted to the number of processed tasks and their timing requirements [1], [2].

B. PIPELINE CYCLE

As mentioned in the above section, every core has a dedicated number of assigned threads. The threads work in the interleaved scheme [38]. The interleaving eliminates the impact of the pipeline processing on the timing unpredictability. The order and frequency of the threads being processed is configurable to allow the best possible match between the core and the tasks being executed. The tasks performed by the threads are switched (interleaved) according to the order stored in a special Interleaving Cycle Register (ICR). The register contains threads' identifiers, and its length is also adjusted. With the appropriate ICR design, some threads can be processed more frequently than others. This mechanism allows the core to match tasks in such a way that threads with very strict timing constraints, and those with less rigorous constraints, can run within a single core. Fig. 3 shows an example of the contents of the ICR in a simplified five-thread core. ICR has a length of 6 threads and processes threads 1, 2, 3, 1, 4 and 5. Assuming that the timing requirements for thread 1 are much stricter, this thread must be executed twice as often as the other threads 2, 3, 4, and 5. Therefore, the identifier of this thread appears more frequently in the ICR. The entire sequence is repeated from the beginning of the system's operation until it is shut down.



		Interleaving cycle (ICR)						Interleaving cycle (ICR)					
	CLK cycle	1	2	3	4	5	6	7	8	9	10	11	12
Thread number	DF	1	2	3	1	4	5	1	2	3	1	5	4
	ID	-	1	2	3	1	4	5	1	2	3	1	5
	EXE	-	-	1	2	3	1	4	5	1	2	3	1
	MEM	-	-	-	1	2	3	1	4	5	1	2	3
	WB	-	-	-	-	1	2	3	1	4	5	1	2

FIGURE 3. An example of ICR content.

To reduce the relationship between the number of threads processed by a single core, the maximum frequency of the clock, and to simplify the inspection of timing predictability of the system, the pipeline can be extended with additional stages, but one needs to remember that a core with an extended pipeline must process the appropriate number of threads [38].

C. TYPES OF THE PROCESSED TASKS

In the designed system, all tasks are of the hard type [39], [40], which means that no thread is allowed to miss

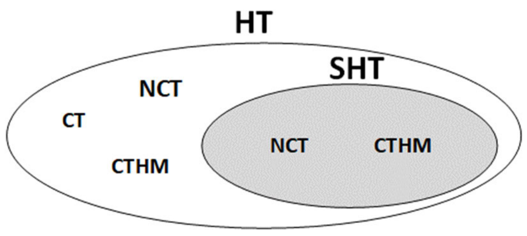


FIGURE 4. Various types of the processed tasks.

completion before its deadline time. The scheduling process calculates the maximum and minimum execution time for each task, which under certain conditions is always less than the deadline time. To facilitate the scheduling process, 3 types of tasks were introduced. They differ in the way they access the data exchange bus:

1) **NCT:** independent tasks that do not cooperate with one another;
2) **CT:** cooperating tasks in which the time of data exchange fluctuates between a certain minimum and maximum values;
3) **CTHM:** cooperating tasks with a fixed data exchange time.

The latter category of tasks (CTHM) allows minimizing the difference between the maximum and minimum execution time of a task at the expense of higher system load (fewer such tasks can be assigned to a single core of the system).

Moreover, another special group of tasks denoted by SHT (Strong Hard Timed) was introduced into the scheduling process. Only tasks in the NTC or CTHM category can belong to this group of tasks (Fig. 4). STH tasks have the highest priority during the scheduling process (especially since the proportion of such tasks is relatively small). Introducing such a priority minimizes the difference between the maximum and minimum execution time for such a task.

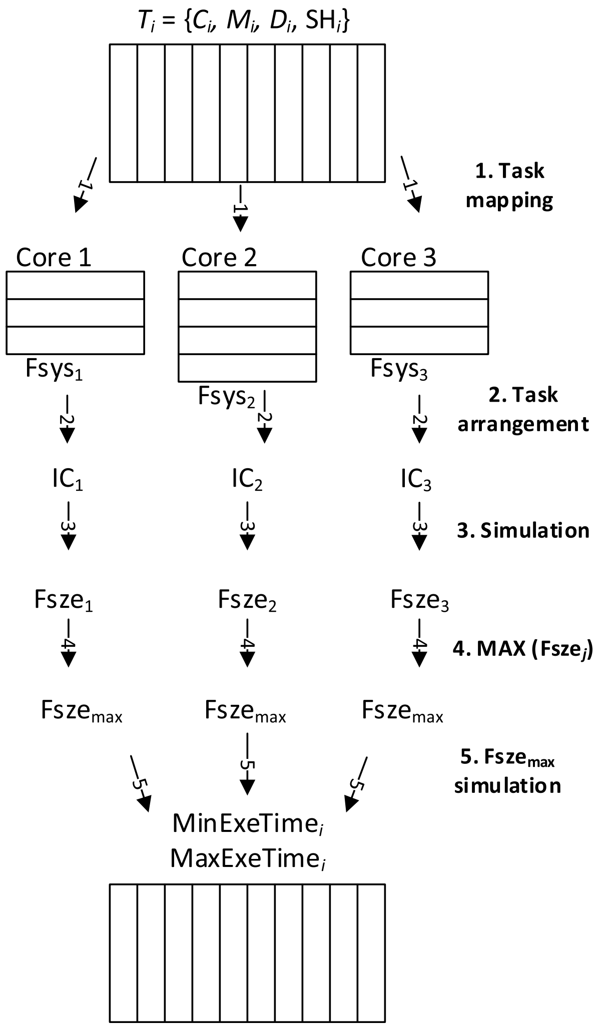


FIGURE 5. Main stages of the tasks' scheduling process.

D. MODEL OF TASKS

The task model was extended [2] with two new parameters (flags) related to the new types of tasks introduced. Thus, the model of a given task unambiguously indicates its type and processing. Finally, the assumed model of the task processed in the proposed system is the following:

$$T_i = \{C_i, M_i, D_i, \mathrm{CTHM}_i, \mathrm{SHT}_i\} \tag{1}$$

where:

C_i	number of standard (without M_i) instructions of i-th task;
M_i	number of memory access instructions that refer to the data of other threads;
D_i	maximum acceptable execution time (deadline) of i-th task;
CTHM_i	flag (valid for $M_i > 0$) indicating whether i-th task is of type CT ('0') or CTHM ('1');
SHT_i	flag indicating whether i-th task is strong hard timed ('1').

The previously defined TF_i (Task Frequency) parameter [1], [2] which allows determining the requirement for computational power for each task, was also remodeled. Changes introduced into the micro-architecture of the system made it necessary to modify the TF_i parameter. Its final version is presented by equation (2):

$$\mathrm{TF}_i[\mathrm{MHz}] = \frac{C_i + M_i \cdot \left[\frac{M_{\mathrm{dur}}}{\mathrm{Min}_{\mathrm{indistance}}}\right] + 2}{D_i[\mu \mathrm{s}]} \tag{2}$$

where:

M_{dur}	number of clock cycles required to execute an instruction of data exchange with another thread;
$\mathrm{Min}_{\mathrm{indistance}}$	interleave depth corresponding to the minimum allowed distance between pipeline stages processing the same task in a given moment of time.

IV. TASKS' SCHEDULLING PROCESS

The proposed tasks' scheduling procedure consists of several steps (Fig. 5). Before starting, it is necessary to complete a preliminary, preparation stage of the scheduling process, during which we determine all the necessary parameters describing the tasks (1) and calculate the task frequency TF_i parameter for every task (2).

The first stage of the scheduling procedure is called ***task mapping***. During this stage tasks are assigned to processing cores, Usually, a balanced load of cores is sought, i.e., that the sum of TFs for all cores is as close as possible. In this stage the minimum theoretical operating frequency of each core ($Fsys_j$) should also be estimated.

The second stage is called ***task arrangement***. During this step of the scheduling process, based on TF_i parameters, the sequences of tasks' identifiers are created. Each sequence is stored in the appropriate ICR of a given core. The length of the ICR and the sequence are selected so as to process each task at the appropriate frequency,

The ***initial simulation*** stage runs a special application (the WCTA developed by the authors) that estimates several factors of the system. First, the maximum number of clock cycles necessary to complete a task is calculated for every task mapped in a given core. Then, based on the frequencies $Fsys_j$ determined in the task splitting phase, the maximum execution time of each task is calculated. If the maximum execution time of any task is greater than its D_i time, a new frequency of the core ($Fsze_j$), in which the task is located, is calculated. This frequency is calculated based on the maximum number of clock cycles needed to execute the task and its deadline (D_i) time.

The fourth stage is ***frequency determination*** which selects the system frequency $Fsze_{max}$ common for all cores as a maximum value among $Fsze_j$. This solution will enable potential further communication between tasks allocated in different cores.

During the last step, the ***final simulation*** of the entire system is carried out. All cores operate at the same frequency $Fsze_{max}$. The simulation results provide a wide range of information, the most relevant of which seems to be data on the maximum and minimum execution time for each task.

Each stage of the task scheduling process is discussed in more detail below.

A. TASKS MAPPING

Depending on the design requirements, constraints, and many other factors, the task mapping process can be carried out in several ways.

In a case, when the number of resources required for the implementation of the system has to be minimized, one should aim to minimize the number of cores processing the tasks. This strategy is represented by '*Methodology*1' in Fig. 6. As our previous research [2] has shown, this approach does lead to some reduction in resources (in the extreme case to a single core with pipelined processing), but as a result, the

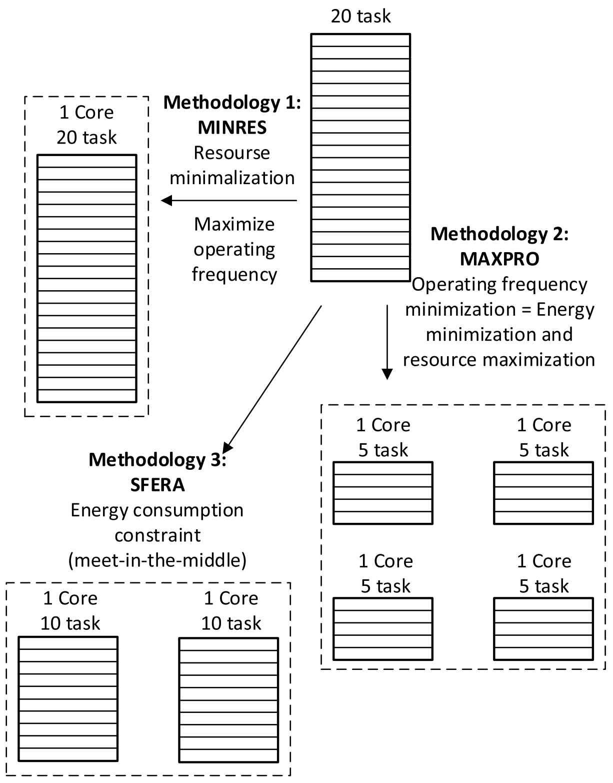


FIGURE 6. Various strategies of tasks mapping.

frequency of the clock signal must be increased significantly. This is because as the number of cores is reduced, the number of concurrent tasks decreases. In turn, raising the frequency of the system results in a significant increase in dynamic power consumption.

Another strategy, shown in Fig.6 as 'Methodology 2', is to reduce the energy consumed by the system. In this case, the designer should aim to increase the number of cores as much as possible. This would result in greater parallelization of calculations, i.e., multiple tasks could be performed simultaneously, and this, in turn, would enable the system clock frequency to be reduced. This will result in a reduction of power consumption. The disadvantage of this solution is the increased demand for hardware resources. In addition, note that there is an important limitation on the minimum number of tasks processed by the core pipeline [1], [2]. This limitation is a direct result of the thread interleaving [38] used. It should be noted that the task arrangement algorithm responsible for creating sequences of thread identifiers (implemented at the next scheduling stage) gives better results when the number of processed tasks is relatively large. Therefore, it is essential to find the optimal number of system cores. Other strategies tend to achieve a trade-off between energy consumption and resource utilization. These strategies are analyzed in the example presented in section V.

However, the starting point is the task mapping algorithm. The previous paper [2] presented two algorithms BLIS II and STODER II. The algorithms were responsible for selecting the appropriate system structure, the number of cores, and

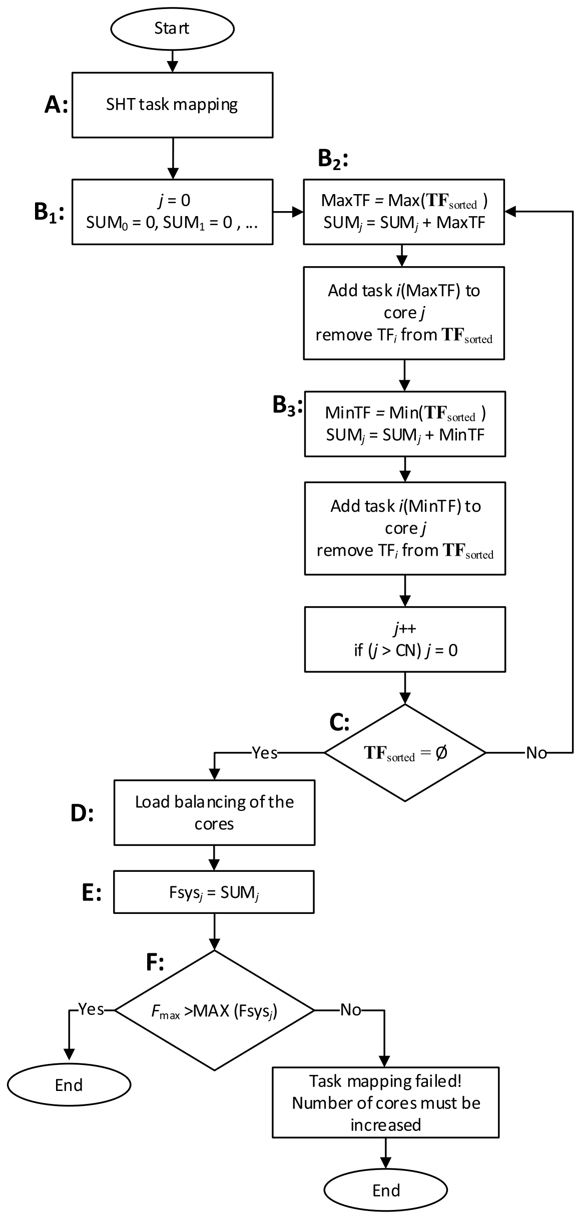


FIGURE 7. **Block diagram of BLTS algorithm.**

assigning tasks to these cores. The algorithms were presented on different optimization goals (methodologies). Here, based on further research and experiments, the modified approach using a single task mapping algorithm called BLTS (Balanced Load Task Scheduling) was presented. BLTS (Fig. 7) is designed for all types of tasks, including newly introduced SHT-type tasks. The algorithm has been adapted to the current version of the system architecture [2], in which the value of the M_{dur} parameter is independent of the number of working cores. The algorithm assigns tasks to the appropriate cores, based on such information as:

1) $F_{\max}$ – the maximum operation frequency of the system. It depends on the system implementation technology;
2) CN – the number of cores that depends on the selected optimization methodology (Fig. 6);
3) Types of tasks and their TF_i parameters.

As it was mentioned above, before starting the algorithm, a set $\mathbb{TF}$ containing the TF_i parameters of all tasks should be determined. Additionally, when performing these calculations, the elements of the $\mathbb{TF}$ matrix are ordered and further operations are performed on the sorted $\mathbb{TF}_{\text{sorted}}$ vector.

In general, it is assumed that the number of system cores is known before the BLTS algorithm starts. This is true for the first two methodologies: MINRES and MAXPRO. When it is necessary to reduce the cost of the system and minimize the required resources (MINRES), then the number of cores $\text{CN}_{\text{MINRES}}$ should be selected according to the formula (3):

$$\text{CN}_{\text{MINRES}} \geq \left\lceil \frac{\sum_{i=1}^{N} \text{TF}_i}{F_{\max}} \right\rceil \tag{3}$$

In the case of the MAXPRO methodology, when the energy consumed by the system is minimized, which is equivalent to minimizing the operating frequency, the number of cores is maximized. However, the number of working cores is limited by a condition related to the minimum distance expressed in cycles of pipelined processing between consecutive instructions of the same task $\text{Min}_{\text{indistance}}$. In practice, the number of cores $\text{CN}_{\text{MAXPRO}}$ handling N tasks will be determined from the relation (4):

$$\text{CN}_{\text{MAXPRO}} \geq \left\lceil \frac{N}{2 \cdot \text{Min}_{\text{indistance}}} \right\rceil \tag{4}$$

In contrast, the third SFERA methodology proposed in the paper [2] is an intermediate solution, in which the cost of the system with specific energy constraints is to be reduced. Unfortunately, in this case, it may be necessary to perform iterative calculations (Fig. 8). Usually, the procedure starts from taking a certain number of cores, resulting from the assumed initial system cost. After the mapping (BLTS algorithm), the power consumed by the system must be estimated. If the imposed constraints are not met, the number of cores must be increased and the entire procedure should be repeated (Fig. 8).

The mapping procedure (Fig. 7) starts from assigning SHT tasks to the cores (step A: of the algorithm). In the optimal

SFERA Methodology [2]

Step 1 **Set** the initial number of cores CN
Step 2 Start the BLTS algorithm
Step 3 Estimate total power consumed by the system:

$$P_{\text{total}} = \sum_{i}^{N} \boldsymbol{P}(pe_i)$$

Step 4 **If $P_{\text{total}} \leq P_{\text{constraint}}$ terminate**
Step 5 **If not** increase CN and **go back** to Step 1

FIGURE 8. **SFERA methodology for scheduling tasks in the regime of efficient energy dissipation with a small amount of resource utilization.**

case, each core should perform at most only one SHT task. If it turns out that the number of SHT tasks exceeds the number of cores, then in the next step (task arrangement), we should increase the values of their task's frequencies TF_i accordingly or freeze the TW_i time window value for each SHT task.

Then the procedure of pre-assignment of tasks to cores is performed (steps B_1-B_3), during which two consecutive tasks with maximum and minimum TF_i values are alternately assigned to successive cores. It should be noted that by operating on a vector of $\mathbb{TF}_{\text{sorted}}$, it is not necessary to search for the maximum and minimum values each time. The first and last elements of the $\mathbb{TF}_{\text{sorted}}$ vector are taken. When all tasks are assigned and the matrix $\mathbb{TF}_{\text{sorted}}$ is empty (step C), the procedure is terminated.

The next step (D) of the algorithm is called "Load balancing of the cores" and is similar to the BLIS II algorithm [2]. Namely, the relative difference between the most loaded core and the least loaded one for a certain used percentage ratio should be minimized.

When we obtain a system with sufficiently evenly loaded cores, we take the appropriate frequency values (step E) and check that all the obtained values of the operating frequencies of the cores are within the acceptable operating range (step F). If any operating frequency exceeds the permissible value of $F_{\max}$, the number of cores must be increased and the algorithm should be repeated from step A.

The entire procedure is presented in the example depicted in the section describing the experiments.

B. TASKS ARRANGEMENT

This stage of task scheduling is responsible for preparing a sequence of thread identifiers (ThID) which will ultimately be stored in the ICR. The order of ThIDs determines the order of tasks processed in the core pipeline. For simulation purposes, the algorithm generates the sequence of identifiers in ASCII code and stores it in the appropriate file. The length of the sequence is adjusted experimentally and this parameter is denoted by WL (Window Length). In order to ensure the continuity of tasks' processing so that the length of the ICR does not determine the processing time, the register works in a round-robin scheme.

The scheduling algorithm is implemented as a program in C# and mimics the behavior of a digital circuit implemented in hardware. Thus, this program can be transferred into the corresponding hardware structure quickly and easily. Each task has its own independent set of counting registers: TC_i, TW_i and TTW_i. The first TC_i reverse counting register (Thread Counter) stores the value corresponding to the number of the clock cycles after which i-th thread is to be loaded into the pipeline, in order to meet its deadline. When a given register TC_i reaches 0, it means that appropriate thread must be executed.

The second register TW_i (Time Window) contains the value representing i-th task's period (the number of clock cycles), i.e., the maximum period of the execution of consecutive instructions of a given thread to meet its timing requirements. The value of this parameter can be calculated from the following expression (5)

$$TW_i = \frac{Fcore_j - Fc_{\text{marg}}}{TF_i} \tag{5}$$

where:

$Fcore_j$[MHz]	Theoretical operating frequency of the j-th core of the system
Fc_{marg}[MHz]	The system's operating frequency margin

The Fc_{marg} parameter is an experimentally selected operating frequency margin of the system, and the results of previous experiments [2] have shown that the best results are achieved with a value of about 4-12% of the $Fcore_j$ parameter.

The decision determining which thread will be processed by the system in the next clock cycle is made based on the states of the TC_i registers. When the algorithm is started, the initial states of the TC_i registers are calculated by copying the values stored in the TW_i registers. However, there is one exception to this rule, no two registers TC_i and TC_j ($i \neq$ j) can contain the same value at any moment of time (a conflict would arise). To avoid such a situation, in case the specified value is already occupied, the nearest free lower state is searched and written to the given TC_i register.

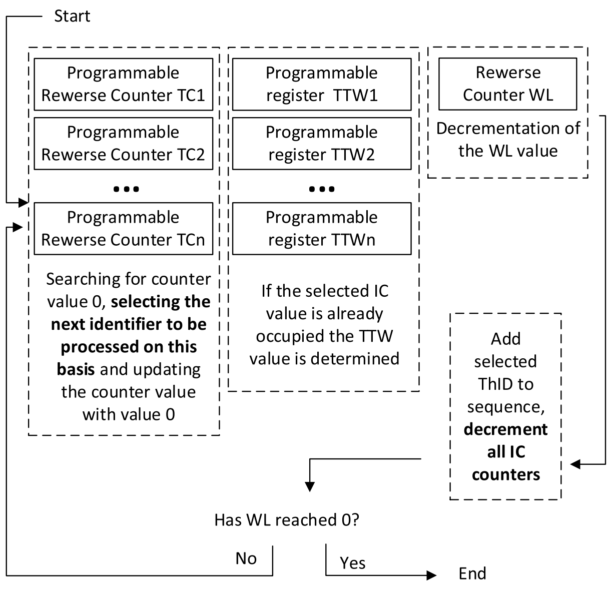

Start
Programmable Rewerse Counter TC1
Programmable Rewerse Counter TC2
•••
Programmable Rewerse Counter TCn
Searching for counter value 0, selecting the next identifier to be processed on this basis and updating the counter value with value 0
Programmable register TTW1
Programmable register TTW2
•••
Programmable register TTWn
If the selected IC value is already occupied the TTW value is determined
Rewerse Counter WL
Decrementation of the WL value
Add selected ThID to sequence, decrement all IC counters
Has WL reached 0?
No
Yes
End

FIGURE 9. ICR sequence generation algorithm.

The third register TTW_i (Temporary Time Window) is used just when conflict of TC values arise. This register holds the value corresponding to the difference between the actual cycle of i-th task occurrence and the cycle resulting from the TW_i value (5). The search procedure seeks to have a thread executed as early as possible, but in a critical situation, its service may occur later than the TW_i parameter (in which case the TTW_i value is negative). The task scheduling algorithm strives to make the average TW_i value of each task as close as possible to the value resulting from equation (5).

Fig. 9 shows the scheme of the ICR sequence generation procedure. In each loop of the algorithm, the value of all TC registers is decremented. When the value of any of them reaches 0, it means that such a thread will be processed in the next clock cycle. Then either the value of TW_i (if it is free at the time) is assigned to this TC_i counting register or a new value is searched for, and the difference is written to the TTW_i register.

It may also occur that in a given cycle none of the counters has reached the zero state, in which case a thread with ThID0 is added to the cycle. This identifier means that no thread will be processed in a given cycle - it will be a so-called idle cycle not processing any instruction. Then the WL value is decremented and the entire cycle is repeated until the value of the WL counter (ICR sequence length counter) reaches 0.

The discussed scheduling algorithm can also eliminate the resulting idle cycles. If the next thread to be added to the cycle is to be the ThID0 thread, the algorithm checks whether it is possible to swap this address for another one. If so, the algorithm swaps it and updates its TC and TTW register values accordingly. This could minimize the number of idle cycles at the expense of faster processing of some threads. This means that this procedure may cause a larger difference between the minimum and maximum execution time of standard threads. Therefore, this part of the algorithm is optional.

SHT threads are hardware threads that either have no instructions to exchange data with other threads or execute instructions to exchange data with other threads with a fixed execution time (Fig. 4). This means that when SHT threads are used, all task execution time discrepancies originating from the system microarchitecture are eliminated. To preserve these properties of SHT threads, these threads take priority in the scheduling process and always appear in the ICR sequence at the exact moments determined by their TW value. This ensures that the difference between their maximum and minimum execution times is minimal.

Unfortunately, the scheduling of threads of this type comes with some limitations. SHT-type threads have priority in scheduling of threads over regular threads, but there can be no conflict in scheduling of two threads of this type (SHT), because one of them could not be an SHT thread. To prevent such conflicts, the TW parameters of SHT-type threads running in a single core must be the same or must be a multiple of the smallest TW of the SHT-type thread. (If there is a conflict, the TW parameter of any of the SHT tasks should be increased, which will result in faster processing.) With the above assumption, the distances between the ThIDs of SHT threads in the generated sequence will always be constant, thus avoiding SHT thread conflicts.

C. INITIAL SIMULATION

The next stage of the proposed scheduling procedure is simulation. We have developed our own simulation environment and implemented it in C# in the form of a timing analyzer WCTA. Input data contains information about tasks (their parameters) and the threads' identifiers sequence (ICR) generated in the previous stage. The simulation of the configured system is done at a high level without stepping into details at the signal level.

During the initial simulation, the necessary information concerning on-line cores' timing parameters is collected. The initial simulation also allows reporting possible errors and suggesting the system's frequency correction. Obtained data shows also the spread of timing parameters. The following steps should be performed for each core operating in the system:

To calculate the maximum and minimum execution time of a given task, the length of the task must be expressed by the number of standard instructions. In the case of the not cooperating tasks (NCT), the problem is trivial, because this number corresponds to the C_i parameter of the task.

For CT type tasks (where $M_i \neq 0$) the length of the data exchange with another thread is expressed by M_{dur} parameter. The method of the estimation of M_{dur} was discussed in [1], [2]. M_{dur} corresponds to the number of clock cycles needed for the completion of the memory operation. To determine the maximum number of standard instructions (C_i) corresponding to the data exchange operation (M_i) (in the worst case), the entire ICR sequence should be analyzed for the number of occurrences of the identifier of a specific task within a time window of the length of M_{dur} cycles (Fig. 10). The entire process is illustrated in Fig. 10 for three tasks #1, #5 and #12. During the initial simulation, a time window of M_{dur} clock bars is created. The first part of the ICR sequence is analyzed in this time window. The number of occurrences of each ThID_i identifier is stored in the corresponding $\mathrm{M_{dur}Ins}_i$ parameter. The time window is then shifted by one symbol towards the end of the ICR sequence and these values are updated as necessary. To reduce the computational complexity of the algorithm, a search of the entire time window is performed only once at the beginning, while after each shift of the time window only the new identifier entering the time window from the right side and the identifier

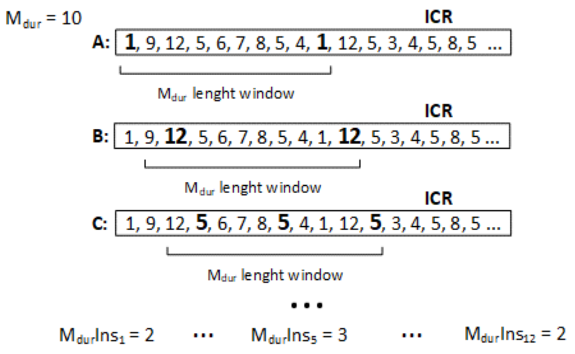

M_dur = 10
ICR
A: 1, 9, 12, 5, 6, 7, 8, 5, 4, 1, 12, 5, 3, 4, 5, 8, 5 ...
M_dur lenght window
ICR
B: 1, 9, 12, 5, 6, 7, 8, 5, 4, 1, 12, 5, 3, 4, 5, 8, 5 ...
M_dur lenght window
ICR
C: 1, 9, 12, 5, 6, 7, 8, 5, 4, 1, 12, 5, 3, 4, 5, 8, 5 ...
M_dur lenght window
M_durIns_1 = 2 ... M_durIns_5 = 3 ... M_durIns_12 = 2

FIGURE 10. Finding the maximum (WCT) number of standard instructions corresponding to the memory instruction for three tasks.

ejected from the window from the left side are checked. As a result, we obtain a set of $\mathrm{M_{dur}Ins}_i$ values corresponding to the number of standard instructions C_i, with a total execution time equivalent to the maximum time required to exchange data for the i-th thread. It should be noted that for tasks of type SHT, the value $\mathrm{M_{dur}Ins}_i$ is constant in all time windows.

Now the total maximum length of a given thread MaxI_i can be expressed by the number of standard instructions. The following relation (6) is, in practice, strongly overestimated (pessimistic) although it gives a 100 percent guarantee of time predictability, since, as is well known, operations with memory are affected by the greatest amount of uncertainty.

$$\mathrm{MaxI}_i = \mathrm{M_{dur}Ins}_i \cdot M_i + C_i \tag{6}$$

where:

MaxI_i	The equivalent of the maximum number of standard instructions needed to perform the i-th task
$\mathrm{M_{dur}Ins}_i$	Maximum number of standard instructions needed to execute a data exchange instruction with another thread

For NCT and CTHM tasks, the number of instructions needed will always be constant. For NCT and CTHM tasks, the number of instructions needed will always be constant. Thus, if MinI_i denotes the minimum number of standard instructions needed to execute a data exchange instruction with another thread, both parameters are equal, i.e. $\mathrm{MaxI}_i = \mathrm{MinI}_i$. Moreover, in particularly advantageous situations, when the data exchange operation takes the same length of time as the standard instruction, CT tasks can be completed faster and $\mathrm{MinI}_i = M_i + C_i$.

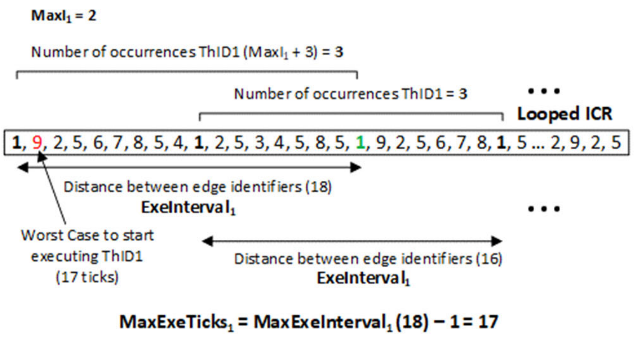

MaxI₁ = 2
Number of occurrences ThID1 (MaxI₁ + 3) = 3
Number of occurrences ThID1 = 3
Looped ICR
1, 9, 2, 5, 6, 7, 8, 5, 4, 1, 2, 5, 3, 4, 5, 8, 5, 1, 9, 2, 5, 6, 7, 8, 1, 5 ... 2, 9, 2, 5
Distance between edge identifiers (18)
ExeInterval₁
Worst Case to start executing ThID1 (17 ticks)
Distance between edge identifiers (16)
ExeInterval₁
MaxExeTicks₁ = MaxExeInterval₁ (18) − 1 = 17

FIGURE 11. Finding the maximum number of system ticks required to complete the task.

In the next step of the simulation, the maximum and minimum number of clock ticks needed to execute a given thread is calculated. The sequence of ThIDs checked in the simulation looks for all possible intervals in which a given task can be completely executed from the start to the end. In the ICR sequence, we look for substrings containing $\mathrm{MaxI}_i + 1$ identifiers of the i-th task for the maximum task execution time and MinI_i identifiers for the case of minimum task execution time, respectively. Each sub-sequence must begin and end with the identifier of the corresponding thread. In this way, the maximum and minimum number of clock cycles required to execute each task mapped to a given core can be determined. The method of determining the $\mathrm{MaxExeTicks}_1$ parameter is shown by the example depicted in Fig. 11. Assuming that the MaxI_1 parameter is 2, the longest sequence of job ThIDs that begins and ends with the identifier "1" and contains $\mathrm{MaxI}_1 + 1$ of these identifiers (3 in this example) is searched for in the sequence of identifiers stored in the ICR. The longest sequence ($\mathrm{MaxExeInterval}_1$) has a length of 18. In the worst-case scenario, i.e., the worst possible moment to start the task execution, the second ID is shown (marked in red). If the length of the task is 2 instructions ($\mathrm{MaxI}_1 = 2$), the task will complete as late as the point marked in green, that is, after 17 clock cycles. Thus, $\mathrm{MaxExeTicks}_1$ is going to be equal to 17.

In the best case, i.e., when the task starts at the best possible moment, the shortest sequence starting and ending with the identifier "1" and containing exactly MinI_1 such identifiers is searched. $\mathrm{MinExeTicks}_1$ will just be equal to the length of this sequence.

In practice, the $\mathrm{MaxExeTicks}_i$ and $\mathrm{MinExeTicks}_i$ values denoting the maximum and minimum number of the clock cycles, respectively, needed to execute the i-th task should be expanded by few cycles associated with the launch of the pipeline of a given core, depending on its configuration [1].

The parameter $\mathrm{MaxExeTicks}_i$ divided by the deadline D_i has a significant practical meaning, namely, it denotes the minimum operating frequency of the core at which the i-th task is predictable. Thus, the minimum operating frequency of the core can be determined from the relation (7):

$$\mathrm{Min_{Freq}} = \max_{i=1,\ldots,N} \left(\frac{\mathrm{MaxExeTicks}_i}{D_i} \right) \tag{7}$$

where:

N	Number of tasks mapped to the core
$\mathrm{MaxExeTicks}_i$	maximum number of the clock cycles needed to execute the i-th task

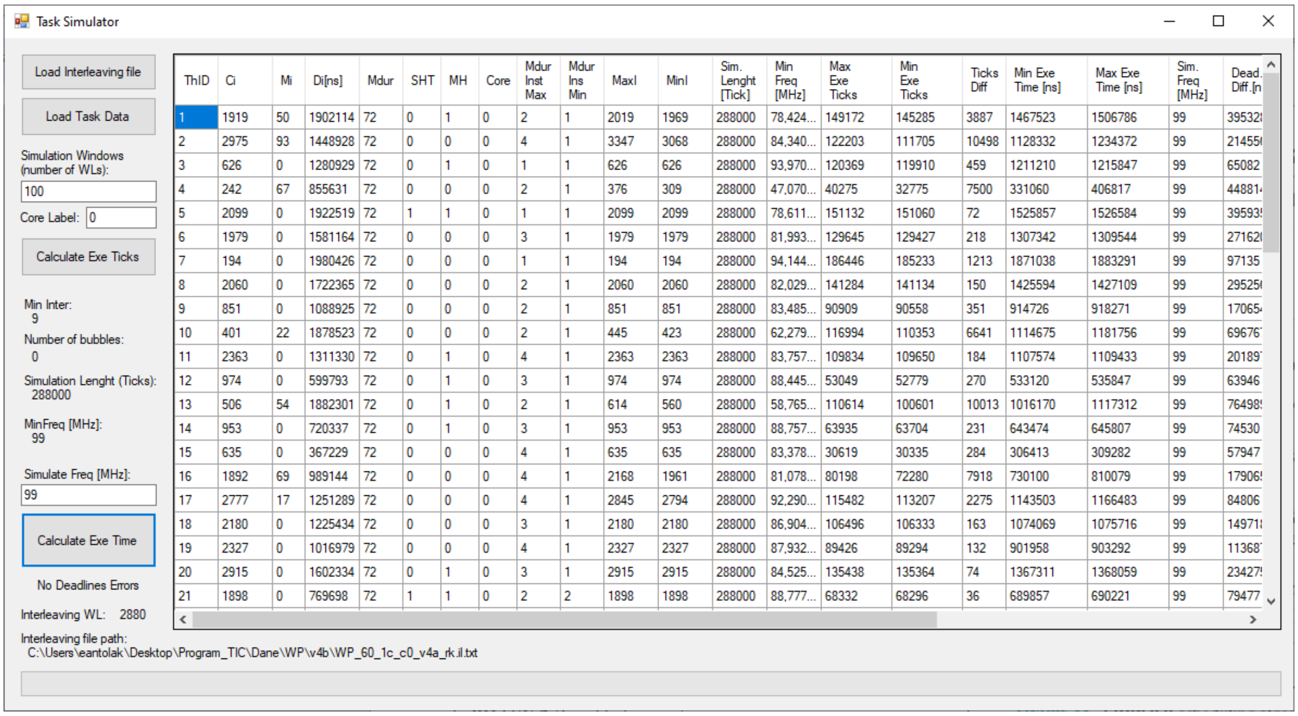
D. FREQUENCY DETERMINATION

If the frequency of the core calculated from the equation (5) is higher than the frequency Fsys_k determined at the stage of mapping tasks to the k-th core, the frequency should be updated and taken as the new value of the operating frequency Fsze_k of this core, otherwise $\mathrm{Fsze}_k = \mathrm{Fsys}_k$.

When the initial simulation of all cores is completed and their operating frequencies determined, the final operating frequency of the entire system should be calculated. To ensure time predictability of all tasks, the maximum value among Fsze_k should be taken. At this point, the final simulation of the entire system can proceed.

E. THE FINAL SIMULATION

Fig. 12 presents the main window (the interface) of the simulator. As with the initial simulation, the numerical parameters



Simulation Windows (number of WLs): 100
Core Label: 0
Min Inter: 9
Number of bubbles: 0
Simulation Lenght (Ticks): 288000
MinFreq [MHz]: 99
Simulate Freq [MHz]: 99
No Deadlines Errors
Interleaving WL: 2880
Interleaving file path: C:\Users\eantolak\Desktop\Program_TIC\Dane\WP\v4b\WP_60_1c_c0_v4a_rk.il.txt

ThID	Ci	Mi	Di[ns]	Mdur	SHT	MH	Core	Mdur Inst Max	Mdur Ins Min	MaxI	MinI	Sim. Lenght [Tick]	Min Freq [MHz]	Max Exe Ticks	Min Exe Ticks	Ticks Diff	Min Exe Time [ns]	Max Exe Time [ns]	Sim. Freq [MHz]	Dead. Diff.[n
1	1919	50	1902114	72	0	1	0	2	1	2019	1969	288000	78,424...	149172	145285	3887	1467523	1506786	99	39532
2	2975	93	1448928	72	0	0	0	4	1	3347	3068	288000	84,340...	122203	111705	10498	1128332	1234372	99	21455
3	626	0	1280929	72	0	1	0	1	1	626	626	288000	93,970...	120369	119910	459	1211210	1215847	99	65082
4	242	67	855631	72	0	0	0	2	1	376	309	288000	47,070...	40275	32775	7500	331060	406817	99	44881
5	2099	0	1922519	72	1	1	0	1	1	2099	2099	288000	78,611...	151132	151060	72	1525857	1526584	99	39593
6	1979	0	1581164	72	0	0	0	3	1	1979	1979	288000	81,993...	129645	129427	218	1307342	1309544	99	27162
7	194	0	1980426	72	0	0	0	1	1	194	194	288000	94,144...	186446	185233	1213	1871038	1883291	99	97135
8	2060	0	1722365	72	0	0	0	2	1	2060	2060	288000	82,029...	141284	141134	150	1425594	1427109	99	29525
9	851	0	1088925	72	0	0	0	2	1	851	851	288000	83,485...	90909	90558	351	914726	918271	99	17065
10	401	22	1878523	72	0	0	0	2	1	445	423	288000	62,279...	116994	110353	6641	1114675	1181756	99	69676
11	2363	0	1311330	72	0	1	0	4	1	2363	2363	288000	83,757...	109834	109650	184	1107574	1109433	99	20189
12	974	0	599793	72	0	1	0	3	1	974	974	288000	88,445...	53049	52779	270	533120	535847	99	63946
13	506	54	1882301	72	0	1	0	2	1	614	560	288000	58,765...	110614	100601	10013	1016170	1117312	99	76498
14	953	0	720337	72	0	1	0	3	1	953	953	288000	88,757...	63935	63704	231	643474	645807	99	74530
15	635	0	367229	72	0	0	0	4	1	635	635	288000	83,378...	30619	30335	284	306413	309282	99	57947
16	1892	69	989144	72	0	0	0	4	1	2168	1961	288000	81,078...	80198	72280	7918	730100	810079	99	17906
17	2777	17	1251289	72	0	0	0	4	1	2845	2794	288000	92,290...	115482	113207	2275	1143503	1166483	99	84806
18	2180	0	1225434	72	0	0	0	3	1	2180	2180	288000	86,904...	106496	106333	163	1074069	1075716	99	14971
19	2327	0	1016979	72	0	0	0	4	1	2327	2327	288000	87,932...	89426	89294	132	901958	903292	99	11368
20	2915	0	1602334	72	0	1	0	3	1	2915	2915	288000	84,525...	135438	135364	74	1367311	1368059	99	23427
21	1898	0	769698	72	1	1	0	2	2	1898	1898	288000	88,777...	68332	68296	36	689857	690221	99	79477

FIGURE 12. **The interface of the system timing analyzer WCTA.**

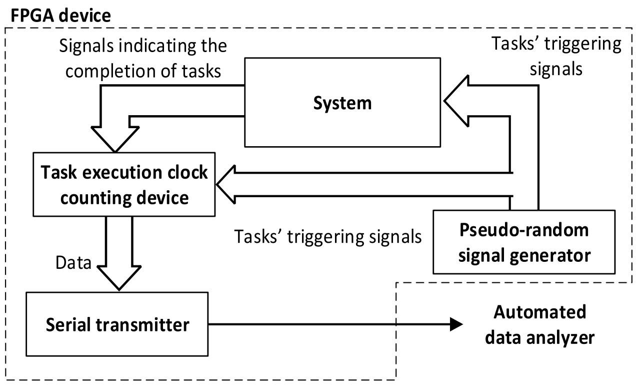

FPGA device
Signals indicating the completion of tasks
Tasks' triggering signals
System
Task execution clock counting device
Tasks' triggering signals
Pseudo-random signal generator
Data
Serial transmitter
Automated data analyzer

FIGURE 13. **The verification environment.**

and types of tasks (1) performed in the system are known at the input, as well as the generated sequence of the tasks identifiers ICR. While performing the final simulation, with the knowledge of $MaxExeTicks_i$ and $MinExeTicks_i$ parameters, the maximum and minimum execution time for each task can be determined, among other issues. This information is presented in columns "Max Exe Time" and "Min Exe Time", respectively. The "MinFreq [MHz]" field provides interesting information. Here, the simulator provides the value of the minimum system frequency at which all tasks will fit within their deadlines.

V. EXPERIMENTAL VERIFICATION OF THE SYSTEM

To verify and validate the proposed new time predictable design methodology, we have developed and arranged our own verification environment (Fig. 13). The previously published [2] task mapping methodology was modified and combined with the WCTA. The method was tested for a system processing 60 randomly generated tasks from a set of implemented programs [1]. In each case, different optimization goals and different system configurations were analyzed. All experiments were conducted using Xilinx's Virtex-7 XC7VX485T-2FFG1761 chip, where the entire system was implemented.

A. STRUCTURE OF THE VERIFICATION ENVIRONMENT

The structure of the proposed verification environment is depicted in Fig. 13. This environment enables independent monitoring of the execution time of each task. It consists of four main modules. The first module is an original multitasking PRET system [2]. According to the established concept of operation of the time predictable system [13], tasks are triggered asynchronously by independent external signals. This function is performed by the second module: the tasks' triggering signal generator module (Pseudo-random signal generator) implemented by means of the LFSR register.

The next module (Task execution clock counting device) counts the number of clock cycles between the signal that starts the task and the feedback signal coming from the system signaling task execution. This module stores the maximum and minimum number of clock cycles counted since the system startup.

Finally, the fourth module: the transmitter converts parallel data into serial and transmits it to the external computer. The transmission can be initialized automatically or manually.

The transmitter can be used to transmit actual data coming from the real-time system regarding similar parameters that were obtained during the simulation (Fig. 12). This data is read out using a dedicated software where it is then analyzed.

B. VERIFICATION OF THE ADOPTED SCHEDULING METHODOLOGY AGAINST THE TIME PREDICTABILITY REQUIREMENTS OF THE SYSTEM

A series of conducted experiments confirmed the advantages of the adopted methodology based on the use of the WCTA. In this section, generalized, aggregated results showing averaged data obtained from at least one million executions of each of the tested tasks are presented. It should be noted that each time the test set was randomized. A more detailed representative example showing our approach step by step is included in the next subsection.

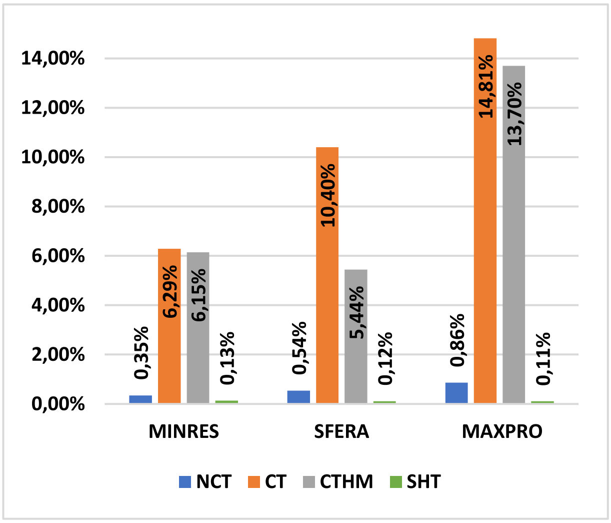


FIGURE 14. The relative difference between the maximum and minimum execution time obtained from WCTA related to the actual maximum execution time (mean values for various types of tasks).

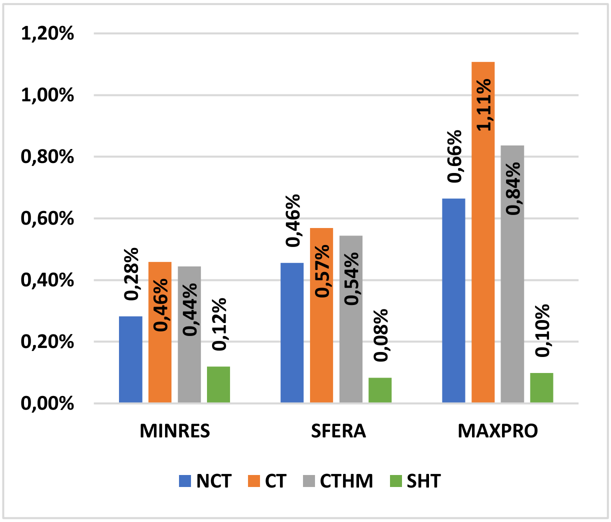


FIGURE 15. The relative difference between the maximum and minimum execution time obtained from the practical experiments related to the actual maximum execution time (mean values for various types of tasks).

The tests comprised successively running three different system configurations related to different design goals: minimizing cost (MINRES), minimizing power (MAXPRO), and aiming to meet given energy constraints while optimizing cost (SFERA). Next, each system was operated and performed the tasks assigned to the computing resources (cores) for about one hour. Then the data acquired from the measurement environment (the maximum and minimum execution time of each task) was compared to the data acquired from the WCTA analyzer and the deadline (D_i) parameter of the tasks. These experiments confirmed the correctness of the concept and execution of the scheduling methodology and no deadline was missed. Furthermore, the performance of the task execution analyzer was verified. Each of the maximum and minimum task execution times is within the allocation calculated by WCTA.

Experiments have shown that the best results are obtained for ICR sequences consisting of 2000-5000 identifiers (WL parameter). The next three diagrams show the averaged results obtained from a series of experiments. The complete analysis, on the other hand, can be traced through a representative example, which is included in the next point of this section.

Figures 14 and 15 show the relative difference between the maximum and minimum execution time of each type of task related to the maximum execution time of the task (the worst case) obtained from simulations and measurements of the actual system, respectively.

The results obtained from the simulation are more pessimistic than the measurements made in the real system realized in the FPGA. This is because in the analysis process, the simulator always assumes the worst case of the task run time. It can be seen from the graphs that the largest fluctuations in task execution time occur for CT and CTHM tasks. A fairly high repeatability of SHT task execution times of about 0.1% was observed.

Fig. 16 shows the relative difference between the deadline and the average task completion time. These charts show how much earlier a given result can be obtained. The largest differences reaching 44% were obtained for SHT tasks. This is due to the fact that in the scheduling process, such tasks are assigned the highest priority, which makes it necessary to increase the frequency of work to meet the time requirements for all tasks.

C. REPRESENTATIVE EXAMPLE

The following example shows the process of scheduling of randomly selected 60 test tasks. In addition, randomization of task parameters was carried out: C_i, M_i and D_i. The following constraints were assumed:

1) the maximum value of C_i is limited to 3000;
2) the maximum value of M_i is limited to 100;
3) the maximum value of TF_i is limited to 4;
4) the probability of random drawing SHT task equals 5%;

TABLE 1. Parameters of test tasks.

ThID	C_i	M_i	D_i [ns]	SHT	CTHM	TF_i
1	1919	50	1902114	-	Yes	1,1939
2	2975	93	1448928	-	-	2,5039
3	626	0	1280929	-	-	0,4903
4	242	67	855631	-	-	0,8333
5	2099	0	1922519	Yes	-	1,0928
6	1979	0	1581164	-	-	1,2529
7	194	0	1980426	-	-	0,0990
8	2060	0	1612365	-	-	1,2789
9	851	0	1088925	-	-	0,7833
10	401	22	1878523	-	-	0,2965
11	2363	0	1311330	-	-	1,8035
12	974	0	599793	-	-	1,6272
13	506	54	1882301	-	Yes	0,4707
14	953	0	720337	-	-	1,3258
15	635	0	367229	-	-	1,7346
16	1892	69	989144	-	-	2,4031
17	2777	17	1251289	-	-	2,3160
18	2180	0	1225434	-	-	1,7806
19	2327	0	1016979	-	-	2,2901
20	2915	0	1602334	-	-	1,8205
21	1898	0	769698	Yes	-	2,4685
22	1016	0	1099207	-	-	0,9261
23	1601	0	1256100	-	-	1,2762
24	275	21	902913	-	-	0,4696
25	602	0	422956	-	-	1,4280
26	1628	0	1900332	-	-	0,8577
27	1811	84	1891852	-	-	1,2691
28	2642	58	1924117	-	-	1,5851
29	317	0	1192492	-	-	0,2675
30	1352	0	989968	-	-	1,3677
31	2930	0	1442498	-	-	2,0326
32	2513	0	1034294	-	-	2,4316
33	2210	0	1415204	-	-	1,5630
34	404	73	1034817	Yes	Yes	0,8861
35	1892	0	974619	-	-	1,9433
36	1346	0	1162232	-	-	1,1598
37	1301	91	1250616	-	-	1,5512
38	959	85	1378492	-	-	1,1288
39	134	0	187292	-	-	0,7261
40	1424	55	1927832	-	-	0,9394
41	1253	39	1553470	-	-	0,9836
42	785	0	1094883	-	-	0,7188
43	374	74	1834138	-	Yes	0,4874
44	1286	0	487666	-	-	2,6412
45	1466	81	543915	-	Yes	3,7414
46	1847	10	492035	-	-	3,9001
47	2117	0	1267142	-	-	1,6723
48	1415	100	1551826	-	-	1,3642
49	2660	0	987253	-	-	2,6964
50	1685	0	1821961	Yes	-	0,9259
51	2300	0	817140	-	-	2,8171
52	1460	50	1624262	-	-	1,1156
53	1691	0	731547	-	-	2,3143
54	2009	0	1003412	-	-	2,0042
55	1229	97	1725713	-	Yes	1,1068
56	734	0	990527	-	-	0,7430
57	2804	37	1446128	-	-	2,1195
58	665	56	768467	-	Yes	1,3781
59	2159	35	1573557	-	Yes	1,5290
60	590	0	1813628	-	-	0,3264

5) the probability of random drawing CT task equals 25%;
6) the probability of random drawing CTHM task equals 5%.

Based on the above assumptions, 60 test tasks were generated, as shown in Table 1.

For the current implementation of the system in the Virtex-7 FPGA chip, $F_{\max} = 150$ MHz was assumed, and this value includes some safety margin [1], [2]. And because the total sum of all tasks' frequencies (TF_i) is lower than $F_{\max}$, all tasks can be allocated in a single core. We then proceeded to determine the number of cores needed in each of the three methodologies, with a power limit of 1 W for the SFERA configuration. The resulting parameters are gathered in Table 2, with 1 core sufficient for the MINRES methodology, and 3 cores for SFERA (assumed as the initial configuration) proving sufficient to meet the power requirements.

The obtained final tasks mapping based on these parameters and taking into account the previously determined task frequencies is shown in Table 3.

As a result of running the task arrangement procedure, ICR sequences with appropriately selected lengths are obtained. For example, in the case of an architecture implemented according to the MINRES scenario, the length of the window is 2880 and it is a multiple of the time window of the largest SHT-type thread ($\mathrm{TW}_5 = 72$). The WL parameters of the other scenarios were selected similarly. Table 4 collects these parameters and shows the minimum operating frequencies of individual cores and the final frequency of the entire system obtained from the simulation process.

Then, each version of the system was implemented in the FPGA and tested in the environment shown in Fig. 13. During the experiments, the power consumption of the different system configurations (Fig. 17) and the requirements for post-implementation hardware resources (Fig. 18) were compared. In the case of energy demand, the difference between the two extreme implementations of MINRES and MAXPRO is about 42% for total power and almost 61% for dynamic power, respectively.

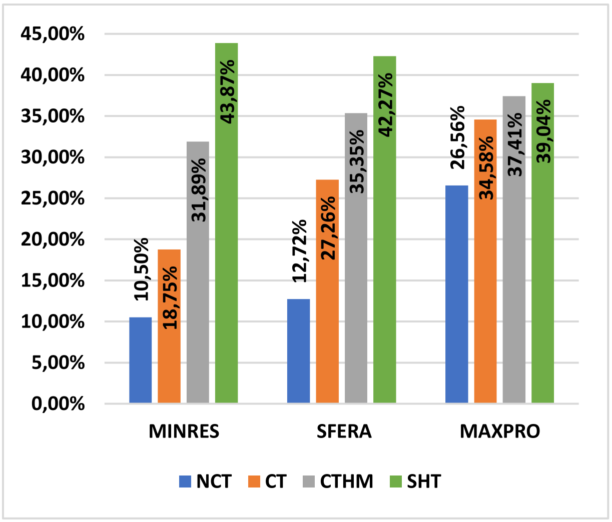


FIGURE 16. Relative difference between deadlines and average time to complete a task.

TABLE 2. Number of processing cores for different methodologies.

CN	MINRES	MAXPRO	SFERA
	1	2	3

TABLE 3. Results of tasks mapping algorithm.

Core #	Number of tasks	Task ThID	Total ΣTF_i	$Fcore_j$ [MHz]
		MINRES		
0	60	All 60 tasks	89,06	90
		MAXPRO		
0	22	5, 58, 51, 60, 2 ,43, 17, 56, 57, 26, 57, 26, 35, 41, 18, 38, 12, 6, 37, 8, 30, 14, 23	29,83	30
1	18	21, 45, 10, 44, 13, 16, 39, 19, 4, 54, 40, 11, 52, 47, 1, 33, 23, 25, 48	28,95	29
2	22	34, 50, 3, 7, 46, 29, 49, 24, 32, 42, 53, 9, 31, 22, 20, 55, 15, 36, 28, 27, 59, 14, 60	29,51	30
		SFERA		
0	29	5, 21, 46, 7, 51, 10, 44, 24, 32, 43, 17, 42, 19, 56, 31, 4, 35, 18, 41, 47, 52, 28, 36, 37, 6, 25, 23, 30, 14	44,33	45
1	31	34, 50, 20, 45, 29, 49, 60, 2, 13, 16, 3, 53, 39, 57, 9, 54, 26, 11, 40, 15, 55, 12, 38, 33, 1, 59, 27, 58, 8, 48, 22	44,23	45

The relationship between resource utilization and the complexity of the structure is much more complicated. While one can observe a certain increase in logical resources as the number of cores increases, in the case of resources responsible for switching, this relationship is not directly proportional. The number of multiplexers strongly depends on the number of tasks being performed, and in some cases high-level synthesis tools manage to achieve some optimum.

A detailed analysis of the time predictability of tasks of different types provides very interesting conclusions. At the same time, it should be taken into account that this analysis is based on averaged results, i.e., each task was performed more than a million times. Graphical representation of all data showing the obtained timing parameters of tasks is impossible due to the amount of data obtained. Therefore, for greater readability, the results for the most individual cases are presented. Example graphs are provided for NCT, CT, CTHM and SHT tasks in Figs. 19 – 22, respectively. The graphs show D_i time (Deadline), maximum (Max S) and minimum (Min S) task completion times calculated by WCTA, and maximum (Max M), minimum (Min M) and averaged (Mean M) task completion times measured in the hardware-implemented system.

Fig. 19 shows measurements for an NCT-type task (THID = 3). All the times are less than the D_i time. It can also be observed that this difference is varied in each scenario, which is due to the different ICR sequences determined in the scheduling process. In the case of this NCT task, the differences between WCTA indications and the measurements are very small and are at the level of single μs.

In the case of the CT task (Fig. 20), a significant difference can be observed between the maximum task execution time calculated by the timing analyzer and the actual task execution time of the system. This is an effect resulting from the WCET analysis performed during the simulation. Moreover,

TABLE 4. Selected results of tasks arrangement and initial simulation of the system.

Core #	$Fcore_j$ [MHz]	WL	Minimum operating frequency of the core obtained from the simulation [MHz]	Minimum operating frequency of the system obtained from the simulation [MHz]
			MINRES	
0	90	2880	100	100
			MAXPRO	
0	30	2700	37	38
1	29	2700	38	
2	30	3200	36	
			SFERA	
0	45	2400	51	51
1	45	3456	51	

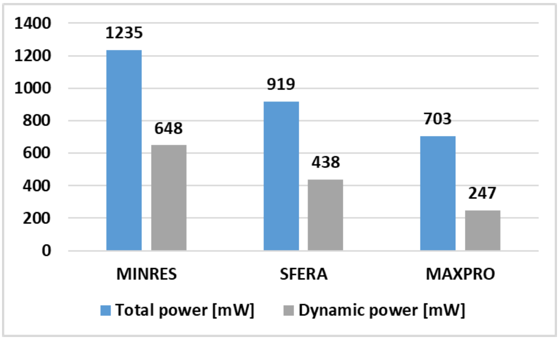


FIGURE 17. Comparison of the energy properties of the different implementations of the system.

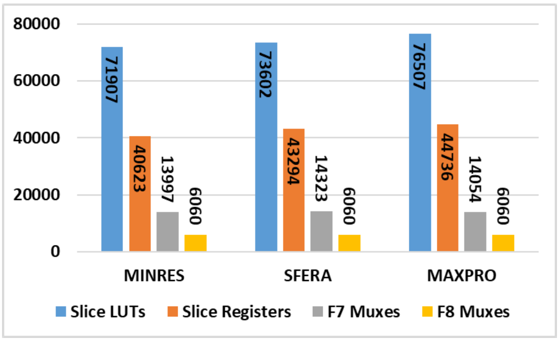


FIGURE 18. Comparison of the resources used for the different implementations of the system (post-implementation).

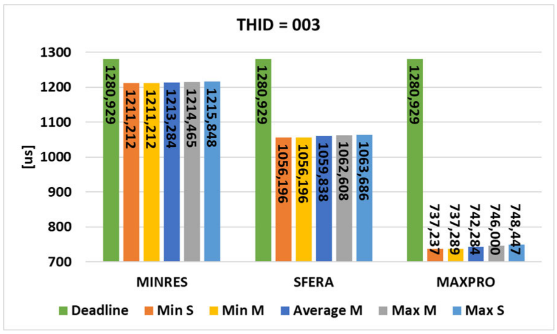


FIGURE 19. Results obtained for the not cooperating task (NCT) nr 3.

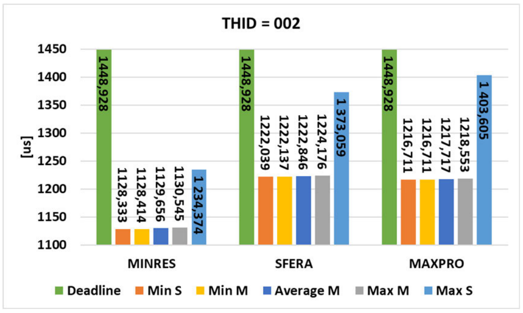


FIGURE 20. Results obtained for the cooperating task (CT) nr 2.

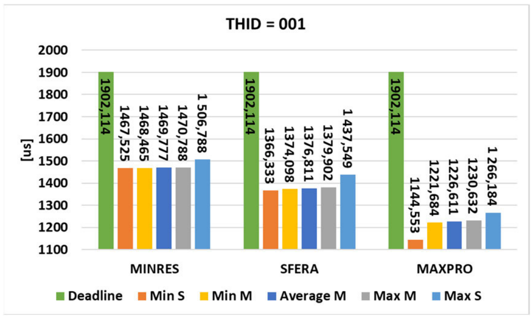


FIGURE 21. Results obtained for the cooperating task with fixed data exchange time (CTHM) nr 1.

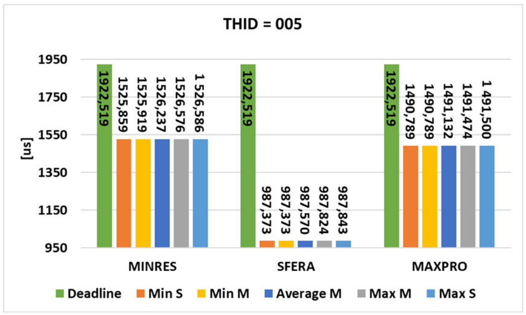


FIGURE 22. Results obtained for the strong hard timed task (SHT) nr 5.

with memory access operations the timing parameters are subject to the greatest degree of imprecision.

The results obtained for the CTHM task (Fig. 21) confirm the achievement of the optimization goal of minimizing the difference between maximum and minimum task execution time which, in this case, is significantly reduced compared to CT tasks.

In the case of the SHT task (Fig. 22), the highest consistency was achieved between the calculations obtained by simulation and the results of measurements of the practically implemented system.

In order to validate the concept of the scheduling process using a dedicated timing analyzer, the results obtained for all tasks are presented collectively. To make this possible, all the times are presented in percentage with respect to the D_i parameter of each task. The diagrams show the average execution times of each task for each system configuration along with the range of changes (min-max), with Fig. 23 showing the data obtained from the simulation and Fig. 24 illustrating the results of measuring the real system.

The presented system adopts a completely asynchronous mode of running tasks that depend on external factors. Moreover, tasks are not handled as a system interrupt. Therefore, the WCET (Worst case execution time) and the BCET (Best case execution time) are dependent on a task triggering moment, which is completely random in a real-time system (see Fig. 11). The authors of the survey concerning real-time embedded systems [29] suggested a special quality measure parameter defined as BCET to WCET ratio. It turns out that this indicator is better the closer its value is to unity. Fig. 25 presents the values of the quality measure index for all the tasks analyzed in the example. The results obtained in the process of simulating the system before launch are much worse than the results obtained when analyzing the operation of the system implemented in the FPGA chip. The best quality measure values were obtained for SHT tasks, which is consistent with the adopted scheduling strategy. Also, for NCT tasks, the values of these coefficients are close to unity, which is understandable because these tasks do not require communication between threads. However, for the actual parameters obtained from the hardware-implemented system, all quality measure values lie above the 0.93 level regardless of the adopted system configuration and design goals.

D. COMPARISON TO OTHER APPROACHES

A conclusive and fair quantitative comparison with other solutions proposed in the literature is quite difficult because different authors highlight different indicators of their solutions, there is no full access to all data, and the used FPGA platforms are made in different technologies. Therefore, the comparative analysis is divided into two parts: a comparison of the components of the solutions (descriptive analysis) and a comparison of the resources of the selected solutions. The results of the first analysis have been gathered in Table 5, while the results of the numerical analysis of hardware resources are included in Fig. 26 and 27.

The proposed solution is implemented in the new family of modern FPGA devices (implemented in 28 nm technology), but this was not the only factor that determined the quality

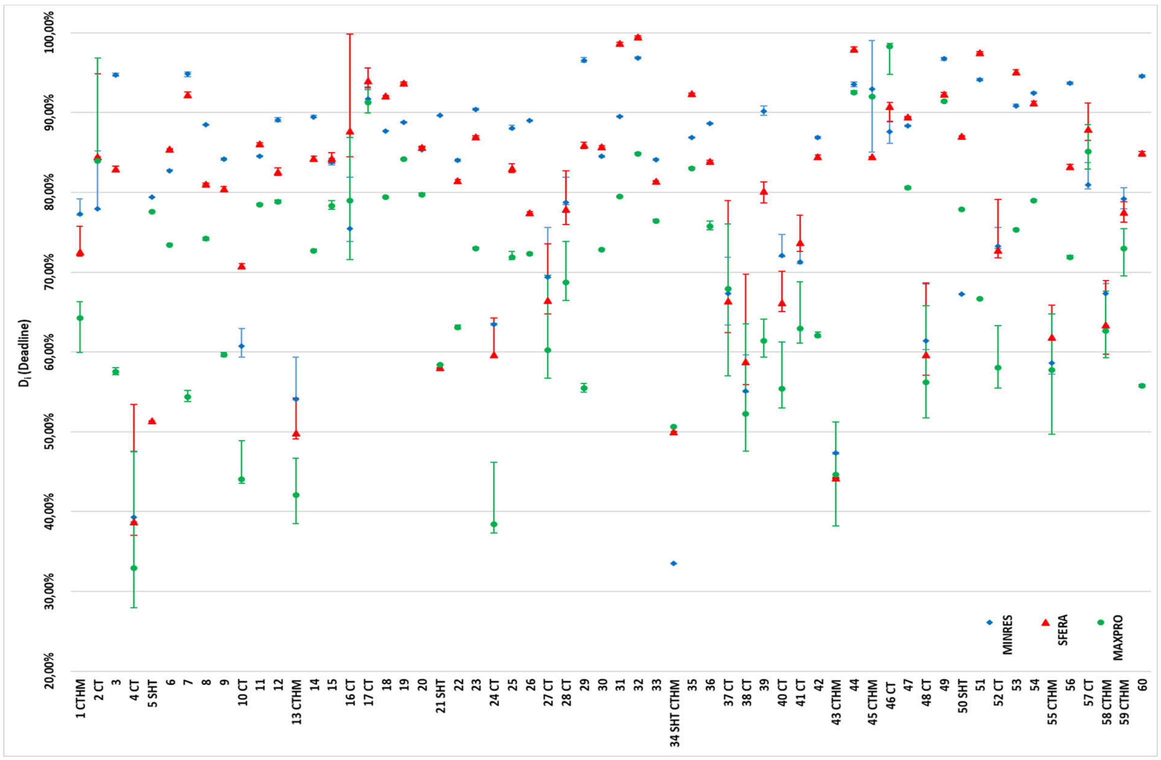


FIGURE 23. The average execution times of each task for each system configuration along with the range of changes (min-max) from WCTA.

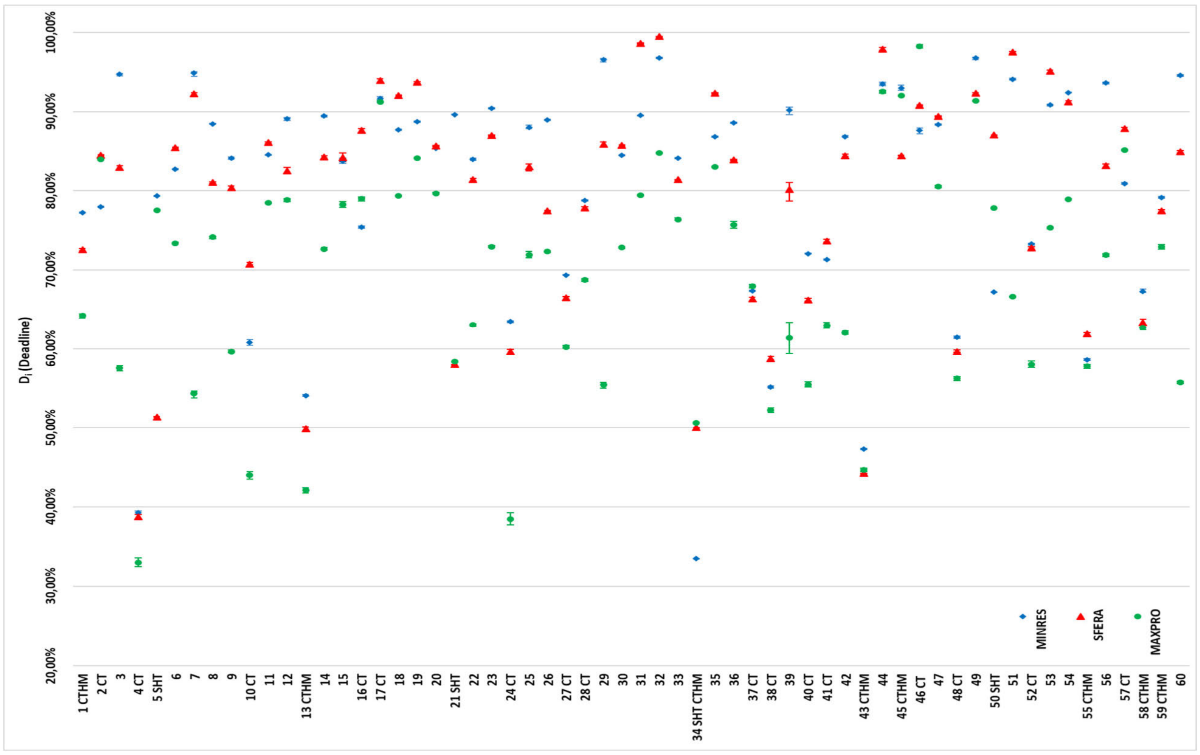


FIGURE 24. The average execution times of each task for each system configuration along with the range of changes (min-max) from the real system implemented in hardware.

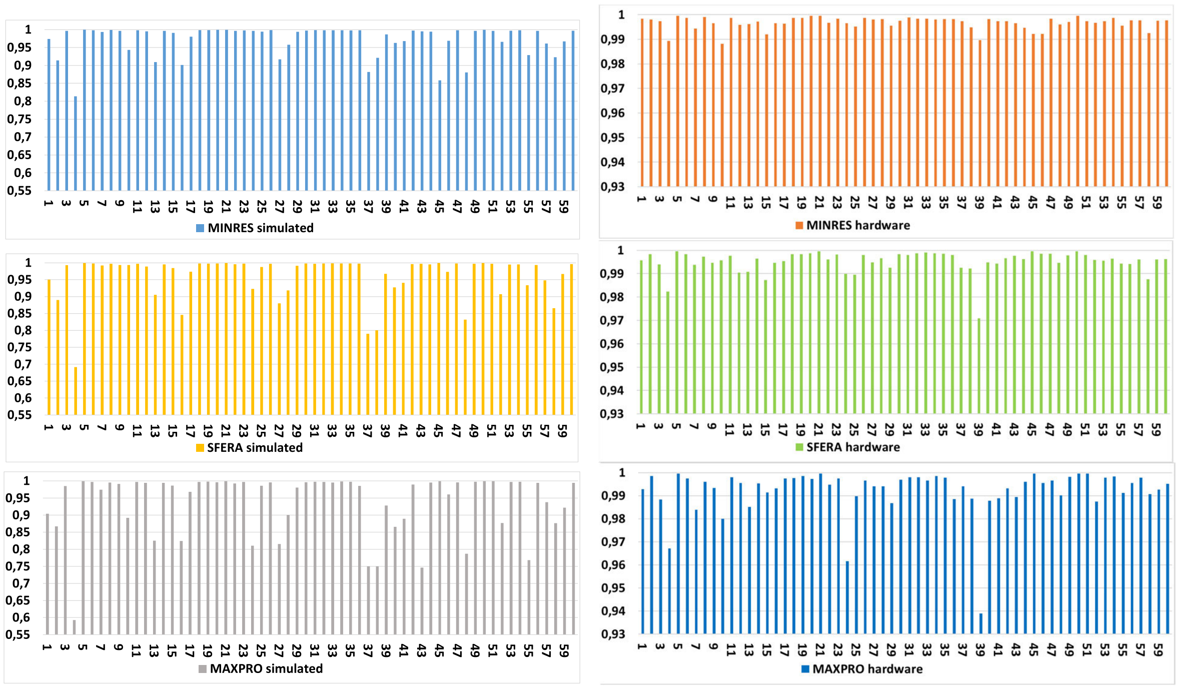


FIGURE 25. The quality measure factors [29] obtained for all tasks in three tested configurations in the simulation (left charts) and in the implemented system (right charts).

TABLE 5. Comparison to other approaches.

Approach	Pipeline processing	Hazards/Contex check	Platform	RTOS	Timing extension (ISA)	Frequency [MHz]	Type of tasks	Number of cores
Ours	Fully interleaved and reconfigurable 5-12 stages	Scheduler keeps control of $\text{Min}_{\text{indistance}}$	FPGA – Virtex 7	ICR mimics hardware RTOS	yes, timing instructions	up to 200 (config.)	NCT, CT, SHT	Multicore (up to 255 – configurable)
FlexPRET [42]	Fully interleaved, 5 stages	Context switch control, hazard detection for branching	FPGA – Virtex 5	Hardware based isolation and context switch	yes, timing instructions	80-100	HRTT, SRTT	Multicore
PTARM [44]	Fully interleaved, 5 stages	temporal isolation between tasks, repeatable latencies	FPGA – Virtex 5	Not stated	Not stated	75	Independent, parallel tasks	Single core
ARPA-MT [45]	Partially interleaved	Pipeline control unit responsible for it	FPGA – Spartan 3	Hardware (the dedicated coprocessor)	Specialized instructions implemented in the coprocessor	58.7	Periodic and jitter tasks	Single core

of the proposed solution. We have proposed a simple and effective task and context switching mechanism based on the interleaving cycle register controlling the pipeline processing. The processing cycles dedicated to a given task are continuously repeated and thus, without the need of interrupts, the task can be executed regardless of when it starts with full timing predictability. The proposed solution mimics RTOS at the hardware layer. In the proposed solution, it is possible to use a single thread as an operating system thread using timing instructions and deadline registers.

The two architectures closest to the proposed solution, in our opinion, are ARPA-MT [45] and FlexPRET [42], and we decided to compare the resources obtained after implementation with these two approaches. The cited papers provide resources for different system configurations and different numbers of tasks, so some normalization of the results has been introduced to make the comparison as relevant as possible.

The diagram depicted in Fig. 26 contains the comparison of resources consumed for the implementation of the presented architectures with the averaged resources required per single task in the ARPA-MT system [45]. That architecture [45] used hardware support for RTOS (Cop2-OSC), while tasks were processed in the main pipeline (Cop0). Such averaging

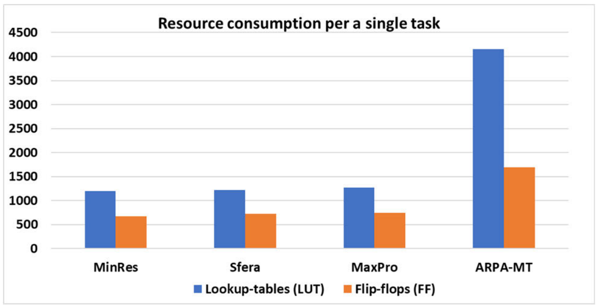


FIGURE 26. Comparison of resource consumption for the entire system implementation related to a single task.

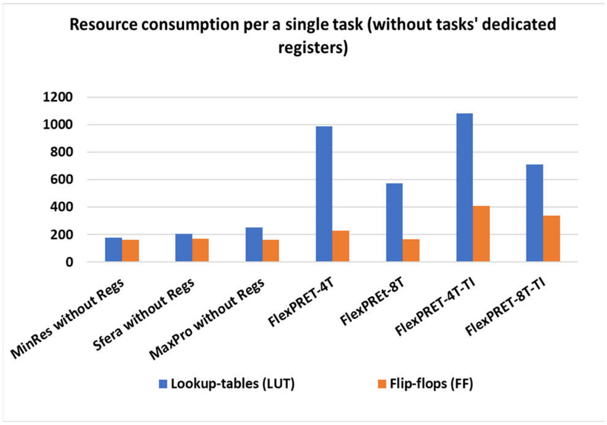


FIGURE 27. Comparison of 'processing' resources (without storing components) consumption per a single task.

can only provide a rough comparison; however, all results indicate that our solution is more resource efficient.

Since in the FlexPRET solution [42] the data related to the context of the tasks being processed (including register files) are stored in memory, the information reported on resources consumed ignores these items. Thus, in order to make the comparison adequate, the resources needed to implement the general-purpose registers of individual tasks in our solutions have been neglected. A direct comparison of resources, especially with a large number of processed tasks, would give a significantly falsified result.

VI. SUMMARY

The presented methodology allows adjusting the hardware structure so as to minimize the energy required for the system's operation or the number of resources required, whilst ensuring the critical time predictability of the system. The proposed method of simulating task execution allows to predict maximum and minimum task execution times even before the system starts up. The division into different types of tasks makes it possible to match the differences between maximum and minimum task execution times with the requirements of the task application.

We were able to achieve a 100% fulfillment rate of task completion time, i.e. DMR=0% [36] for all tasks. Although there are situations in which certain tasks may be completed clearly before their deadlines, this situation is particularly true for SHT tasks. For applications in secure systems, such a situation could be critical. However, taking into account that in our solution each task has a dedicated hardware deadline counter, it can be used to release the data/input signals of such a critical task.

The task processing strategy adopted in the approach, based on thread interleaving, does not require mechanisms for hazards control in the pipeline and other issues related to the prediction of branches, jumps, etc.

REFERENCES

[1] E. Antolak and A. Pułka, "Flexible hardware approach to multi-core time-predictable systems design based on the interleaved pipeline processing," *IET Circuits, Devices Syst.*, vol. 14, no. 5, pp. 648–659, Aug. 2020, doi: 10.1049/iet-cds.2019.0521.

[2] E. Antolak and A. Pulka, "Energy-efficient task scheduling in design of multithread time predictable real-time systems," *IEEE Access*, vol. 9, pp. 121111–121127, 2021, doi: 10.1109/ACCESS.2021.3108912.

[3] M. Schoeberl, "T-CREST: Time-predictable multi-core architecture for embedded systems," *J. Syst. Archit.*, vol. 61, no. 9, pp. 449–471, Oct. 2015, doi: 10.1016/j.sysarc.2015.04.002.

[4] M. Paolieri, E. Quinones, F. J. Cazorla, J. Wolf, T. Ungerer, S. Uhrig, and Z. Petrov, "A software-pipelined approach to multicore execution of timing predictable multi-threaded hard real-time tasks," in *Proc. 14th IEEE Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, Mar. 2011, pp. 233–240, doi: 10.1109/ISORC.2011.36.

[5] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, no. 5, pp. 66–75, Sep. 2010, doi: 10.1109/MM.2010.78.

[6] D. D. Gajski, *Embedded System Design: Modeling, Synthesis and Verification*. Dordrecht, The Netherlands: Springer, 2009.

[7] E. A. Lee, "Absolutely positively on time: What would it take? [embedded computing systems," *Computer*, vol. 38, no. 7, pp. 85–87, Jul. 2005, doi: 10.1109/MC.2005.211.

[8] S. Andalam, P. S. Roop, A. Girault, and C. Traulsen, "A predictable framework for safety-critical embedded systems," *IEEE Trans. Comput.*, vol. 63, no. 7, pp. 1600–1612, Jul. 2014, doi: 10.1109/TC.2013.28.

[9] D. Broman, "Precision timed infrastructure: Design challenges," in *Proc. Electron. Syst. Level Synth. Conf. (ESLsyn)*, Austin, TX, USA, May 2013, pp. 1–6.

[10] T. Henzinger and C. Kirsch, "The embedded machine: Predictable, portable real-time code," *ACM Trans. Program. Lang. Syst.*, vol. 29, p. 33, Dec. 2002, doi: 10.1145/512529.512567.

[11] E. A. Lee, "The problem with threads," *Computer*, vol. 39, no. 5, pp. 33–42, May 2006, doi: 10.1109/MC.2006.180.

[12] F. J. Cazorla, P. M. W. Knijnenburg, R. Sakellariou, E. Fernandez, A. Ramirez, and M. Valero, "Predictable performance in SMT processors: Synergy between the OS and SMTs," *IEEE Trans. Comput.*, vol. 55, no. 7, pp. 785–799, Jul. 2006, doi: 10.1109/TC.2006.108.

[13] S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, San Diego, CA, USA, Jun. 2007, pp. 264–265, doi: 10.1109/DAC.2007.375165.

[14] L. Thiele and R. Wilhelm, "Design for timing predictability," *Real-Time Syst.*, vol. 28, nos. 2–3, pp. 157–177, Nov. 2004, doi: 10.1023/B:TIME.0000045316.66276.6e.

[15] N. J. H. Ip and S. A. Edwards, "A processor extension for cycle-accurate real-time software," in *Embedded and Ubiquitous Computing*, vol. 4096, E. Sha, S.-K. Han, C.-Z. Xu, M.-H. Kim, L. T. Yang, and B. Xiao, Eds. Berlin, Germany: Springer, 2006, pp. 449–458, doi: 10.1007/11802167_46.

[16] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable programming on a precision timed architecture," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst. (CASES)*, Atlanta, GA, USA, 2008, p. 137, doi: 10.1145/1450095.1450117.

[17] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, and C. W. Probst, "Towards a time-predictable dual-issue microprocessor: The Patmos approach," in *Proc. 1st Workshop Bringing Theory Pract., Predictability Perform. Embedded Syst. (PPES)*, Grenoble, France, 2011, pp. 11–21. Accessed: Nov. 12, 2019. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2011/3077

[18] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla, "Assessing the suitability of the NGMP multi-core processor in the space domain," in *Proc. 10th ACM Int. Conf. Embedded Softw. (EMSOFT)*, Tampere, Finland, 2012, pp. 175–184, doi: 10.1145/2380356.2380389.

[19] A. Alhammad and R. Pellizzoni, "Time-predictable execution of multi-threaded applications on multicore systems," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, 2014, pp. 1–6, doi: 10.7873/DATE.2014.042.

[20] S. Moulik, R. Devaraj, and A. Sarkar, "COST: A cluster-oriented scheduling technique for heterogeneous multi-cores," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Miyazaki, Japan, Oct. 2018, pp. 1951–1957, doi: 10.1109/SMC.2018.00337.

[21] R. Pathan, P. Voudouris, and P. Stenström, "Scheduling parallel real-time recurrent tasks on multicore platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 915–928, Apr. 2018, doi: 10.1109/TPDS.2017.2777449.

[22] D. Kim, Y.-B. Ko, and S.-H. Lim, "Energy-efficient real-time multi-core assignment scheme for asymmetric multi-core mobile devices," *IEEE Access*, vol. 8, pp. 117324–117334, 2020, doi: 10.1109/ACCESS.2020.3005235.

[23] J. Chen, C. Du, P. Han, and Y. Zhang, "Sensitivity analysis of strictly periodic tasks in multi-core real-time systems," *IEEE Access*, vol. 7, pp. 135005–135022, 2019, doi: 10.1109/ACCESS.2019.2941958.

[24] B. Forsberg, L. Benini, and A. Marongiu, "HePREM: Enabling predictable GPU execution on heterogeneous SoC," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2018, pp. 539–544, doi: 10.23919/DATE.2018.8342066.

[25] B. Akesson and K. Goossens, *Memory Controllers for Real-Time Embedded Systems*, vol. 2. New York, NY, USA: Springer, 2012, doi: 10.1007/978-1-4419-8207-0.

[26] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee, "PRET DRAM controller: Bank privatization for predictability and temporal isolation," in *Proc. 7th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Taipei, Taiwan, 2011, p. 99, doi: 10.1145/2039370.2039388.

[27] L. M. AlBarakat, P. V. Gratz, and D. A. Jiménez, "MTB-fetch: Multi-threading aware hardware prefetching for chip multiprocessors," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 175–178, Jul. 2018, doi: 10.1109/LCA.2018.2847345.

[28] M. Schoeberl, "A Java processor architecture for embedded real-time systems," *J. Syst. Archit.*, vol. 54, nos. 1–2, pp. 265–286, Jan. 2008, doi: 10.1016/j.sysarc.2007.06.001.

[29] P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, N. Guan, B. Jonsson, P. Marwedel, J. Reineke, C. Rochange, M. Sebastian, R. V. Hanxleden, R. Wilhelm, and W. Yi, "Building timing predictable embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4, pp. 1–37, Dec. 2014, doi: 10.1145/2560033.

[30] Y. Kim, J. Kong, and A. Munir, "CPU-accelerator co-scheduling for CNN acceleration at the edge," *IEEE Access*, vol. 8, pp. 211422–211433, 2020, doi: 10.1109/ACCESS.2020.3039278.

[31] A. U. Rehman, "Dynamic energy efficient resource allocation strategy for load balancing in fog environment," *IEEE Access*, vol. 8, pp. 199829–199839, 2020, doi: 10.1109/ACCESS.2020.3035181.

[32] F. Glaser, G. Tagliavini, D. Rossi, G. Haugou, Q. Huang, and L. Benini, "Energy-efficient hardware-accelerated synchronization for shared-L1-memory multiprocessor clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 633–648, Mar. 2021, doi: 10.1109/TPDS.2020.3028691.

[33] H. Bahn and K. Cho, "Evolution-based real-time job scheduling for co-optimizing processor and memory power savings," *IEEE Access*, vol. 8, pp. 152805–152819, 2020, doi: 10.1109/ACCESS.2020.3017014.

[34] H. Chniter, O. Mosbahi, M. Khalgui, M. Zhou, and Z. Li, "Improved multi-core real-time task scheduling of reconfigurable systems with energy constraints," *IEEE Access*, vol. 8, pp. 95698–95713, 2020, doi: 10.1109/ACCESS.2020.2990973.

[35] *Welcome to LPA*. Accessed: Jan. 31, 2023. [Online]. Available: https://www.lpa.co.uk/

[36] Y. Gao, G. Pallez, Y. Robert, and F. Vivien, "Dynamic scheduling strategies for firm semi-periodic real-time tasks," *IEEE Trans. Comput.*, vol. 72, no. 1, pp. 55–68, Jan. 2023, doi: 10.1109/TC.2022.3208203.

[37] E. Antolak and A. Pułka, "An analysis of the impact of gating techniques on the optimization of the energy dissipated in real-time systems," *Appl. Sci.*, vol. 12, no. 3, p. 1630, Jan. 2022, doi: 10.3390/app12031630.

[38] E. Lee and D. Messerschmitt, "Pipeline interleaved programmable DSP's: Architecture," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-35, no. 9, pp. 1320–1333, Sep. 1987, doi: 10.1109/TASSP.1987.1165274.

[39] G. C. Buttazzo, *Hard Real-Time Computing Systems*, vol. 24. Boston, MA, USA: Springer, 2011, doi: 10.1007/978-1-4614-0676-1.

[40] E. L. Lamie, *Real-Time Embedded Multithreading: Using ThreadX and ARM*. San Francisco, CA, USA: CMP Books, 2005.

[41] A. Diavastos and T. E. Carlson, "Efficient instruction scheduling using real-time load delay tracking," *ACM Trans. Comput. Syst.*, vol. 40, nos. 1–4, pp. 1–21, Nov. 2022, doi: 10.1145/3548681.

[42] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Berlin, Germany, Apr. 2014, pp. 101–110, doi: 10.1109/RTAS.2014.6925994.

[43] I. Liu, J. Reineke, and E. A. Lee, "A PRET architecture supporting concurrent programs with composable timing properties," in *Proc. Conf. Rec. Forty 4th Asilomar Conf. Signals, Syst. Comput.*, Pacific Grove, CA, USA, Nov. 2010, pp. 2111–2115, doi: 10.1109/ACSSC.2010.5757922.

[44] I. Liu, J. Reineke, D. Broman, M. Zimmer, and E. A. Lee, "A PRET microarchitecture implementation with repeatable timing and competitive performance," in *Proc. IEEE 30th Int. Conf. Comput. Design (ICCD)*, Montreal, QC, Canada, Sep. 2012, pp. 87–93, doi: 10.1109/ICCD.2012.6378622.

[45] A. S. R. Oliveira, L. Almeida, and A. D. B. Ferrari, "The ARPA-MT embedded SMT processor and its RTOS hardware accelerator," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 890–904, Mar. 2011, doi: 10.1109/TIE.2009.2028359.

[46] *CHISEL Main Web Page*. Accessed: Mar. 16, 2023. [Online]. Available: https://www.chisel-lang.org/community.html

ERNEST ANTOLAK received the M.Sc. degree in electronics and telecommunication engineering from the Silesian University of Technology, Gliwice, Poland, in 2018. He is currently pursuing the Ph.D. degree in project methodology of designing real-time systems. His research interests include real-time scheduling, designing safety-critical embedded systems, cyber-physical systems, systems on chips, and energy-efficient digital architectures.

ANDRZEJ PUŁKA (Senior Member, IEEE) received the M.Sc., Ph.D., and D.Sc. degrees in electronics from Silesian Technical University, Gliwice, Poland, in 1988, 1997, and 2013, respectively.

Currently, he is the University Professor of the Silesian University of Technology, Gliwice, and the Deputy Head of the Department of Electronics, Electrical Engineering and Microelectronics. He is the author and coauthor of approximately 90 scientific papers, including journal articles, book chapters, and conference papers. His research interests include the automated design of digital and mixed signal circuits in FPGAs, the modeling and simulation of electronic embedded systems, VHDL, Verilog, SystemVerilog, SystemC, real-time systems—precision time machines (PRET), the design of energy efficient systems, power optimization in SoC, AI, and commonsense reasoning modeling, and the applications of FPGA platforms for hardware acceleration of complex computations in bioinformatics. He is a member of the Electronics Commission of the Polish Academy of Sciences.