Received June 18, 2021, accepted August 19, 2021, date of publication August 30, 2021, date of current version September 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3108912

Energy-Efficient Task Scheduling in Design of Multithread Time Predictable Real-Time 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 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 for Statutory Activities under Project BKM-669/RAU-11/2020.

ABSTRACT The paper presents balanced heuristic techniques of static tasks scheduling in multi-core real-time system architecture. The main objective was to minimize the energy consumed by the system without causing deadlines to be missed. The authors proposed a few scheduling scenarios based on modifications of different parameters of the system and defined appropriate algorithms. The methodology was verified and tested on the original hardware platform of the system developed by the authors. The system consists of the reconfigurable set of cores based on an interleaved pipeline processing scheme. The entire system was modeled as original IP at the RTL level in VERILOG and implemented on a Virtex7 FPGA platform. The tasks were executed as a set of programs based on Mälardalen WCET benchmarks; commonly used by the PRET community for validation time predictable systems and worst-case analyses. Many series of experiments were carried out and the results validated the approach and showed that it is possible to radically reduce the energy consumed by the system while retaining timing conditions (i.e., deadlines). The authors gathered and discussed results and formulated a set of recommendations for the safety real-time system's design flowchart aimed to minimize the energy consumed by the system.

INDEX TERMS Real-time systems, dynamic scheduling, multitasking, pipeline interleaving, power optimization, energy-efficient systems.

NOMENCLATURE

t_i *i*-th	task symbol.
T_i *i*-th	task model.
C_i	Program length, (number of instructions except for M_i).
M_i	Number of memory access instructions that refer to data of other threads.
D_i	Deadline of the task (time).
M_dur	Duration of the memory operations (WCET).
ThN	Number of threads processed by the system.
Req_{proc}	Number of clock cycles necessary for the bus request processing for the longest pipeline.
$Pipeline_{start}$	Number of clock cycles necessary for reinitialization of the requesting pipeline.
TF_i *i*-th	task frequency.
$\mathbb{TF}$	Matrix of all task frequencies (TF_i).
$\mathbb{TF}_{sorted}$	Sorted matrix of all task frequencies.
TF_{MEAN}	Mean value of elements of the matrix $\mathbb{TF}$.
CN	Number of processing cores.
F_{sys}	Frequency of the system.
F_{margin}	Empirical parameter.
PE	Set of processing elements (cores).
pe_i *i*-th	processing element.
$P(pe_i)$	Power consumed by pe_i.
$M(pe_i)$	Mapping of tasks to pe_i.
$pt(t_i)$	Processing time of the task t_i.
GF	Optimization goal function.
TS	Total sum of all tasks' frequencies (TF_i) allocated in a given core.
DTS	Total sum of all tasks' frequencies (TF_i) allocated in a dedicated core.
STS	Total sum of all tasks' frequencies (TF_i) allocated in a standard core.
P_{total}	Total power dissipated in the system.

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

$P_{constraint}$	Power constraint (maximum allowed power dissipated in the system).
BC	Bus controller.
BUS SRQ	Bus send request phase.
BUS RRQ	Bus receive request phase.
CP	Configurable pipeline.
DF	Data fetch.
EXE	Execute.
GPR	General purpose register.
ID	Instruction decode.
SA	Select address.
SD	Select data.
SFT	Shift.
SR	Select register.
THj	Identifier of the j-th thread.
WB	Write back.
E_{taski}	Energy consumed by a single task $task_i$.
α	Switching activity of the circuit.
C_L	Switching capacity.
V_{DD}	Supply voltage.
$P_{forecast}$	The value of the forecasted dynamic power.
P_k	Dynamic power consumed by k-th task.
F_{border}	Theoretical border frequency of the system.
RISC	Reduced instruction set computer.

I. INTRODUCTION

One of the most important properties (and the most crucial requirement) of contemporary real-time embedded systems is their predictability. Modern electronic devices have become more and more complex. Strong competition between vendors implies that they offer the equipment supporting new, sophisticated functionalities. However, this intricacy of electronic embedded system's architectures may discard their predictability and the engineers involved in the design process are required to spend hundreds of hours to keep the systems predictable. Over the last decade, one can observe that researchers from all over the world have made a significant effort to find architectures assuring timing repeatability. Real-time systems are commonly used in many applications (not only safety systems) and must be fully predictable [1], [2]. All procedures used in the system design flow must check not only the correct functionality of the devices but most of all, they should pay attention to timing dependencies between events occurring in the system. The issue of timing predictability of electronic systems is known in the literature [4], [5] as a paradigm of precision time machines (PRET) formulated by S. Edwards and E Lee [3] during the Design Automation Conference (DAC) in 2007.

The paper presents the next contribution in the research on time-predictable systems. The authors deal with the lowest level of the system architecture, i.e., hardware level development. This research on PRET systems follows on from the previous works that were carried out by the Department of Electronics, Electrical Engineering and Microelectronics (former Institute of Electronics) for many years [6]–[8], [24].

The paper consists of seven parts: the second section analyzes the related work and other solutions concerning hardware as well as software elements of time-predictable real-time systems. In the third section, the authors present the main building blocks of their architecture that are based on pipeline processing and the threads' interleaving. The fourth section formulates the motivation of the work and main contributions of the paper. The fifth section presents the practical properties of the platform obtained by a set of initial experiments. In the sixth section, we define the tasks' model, main parameters, the optimization objectives and the goal function. In the next, we propose modifications of our scheduling algorithms and presents different design techniques on an example. The section number eighth describes the testing environment and shows the results of the experiments and the final, ninth section summarizes the paper and presents some conclusions.

II. RELATED WORK

Designers involved with the process of constructing time-predictable systems have to investigate many aspects of the entire architecture. They have to develop hardware, software and propose appropriate communication protocols. It is recommended to address these problems at the early stages of the design flow. Thiele and Wilhelm [5] formulated some recommendations and guidelines that should be considered by designers of safety-critical embedded systems. The methodology shows how to meet strict, real-time constraints during the technology mapping, system architecture selection and software implementation. The authors also mention the PRET concept [3], which is suggested to extend the microarchitecture (ISA) of RISC processors [9]. They postulated the addition of a new timing instruction controlling the processing time - *deadline*. A very interesting solution was developed in the CHESS laboratory at UC Berkeley [11]. This solution was based on the interleaved pipeline [10] that allows avoiding problems with data and control hazard occurrences in the pipeline. The CHESS group also recommended the memory wheel controller for controlling the access to the system memory by many tasks (threads). Pułka and Milik [6] expanded the CHESS solution and proposed the flexible, time-predictable architecture consisting of a dynamic interleave controller of the threads, a memory access control unit and thread porch memory.

In general, we can observe that research on PRET architectures requires the exploration of various issues of hardware [4], [5], [15], [20]–[22] or software [16]–[19] development. Frequently, researchers seek general guidelines [1], [5], [12]–[15], while some works give dedicated solutions [17], [19], [20], [22].

Recently, in [24] we have proposed our system based on the flexible pipeline configuration and optimal scheduling algorithms. In the presented paper we investigated the problem of energy optimization in time-predictable systems and propose our solution in the field. Many examples in the literature show this problem in modern mobile and remote electronic embedded systems becoming very important. In [23] and [25] the problem of load-balanced scheduling was addressed and the authors of the proposed techniques reported a reduction of the energy consumed by the systems. The authors of [26] presented a very interesting technique of effective utilization of processing elements (PE) in the clusters of processors with the shared L1 cache memory. The solution optimized synchronization and communication between processing elements. The result was not exactly dedicated to real-time systems, however, it allowed radical reduction of the energy consumed by the system. The authors of [27] proposed a set of scheduling algorithms and investigated various approaches based on deadline slack analysis, global dynamic voltage, frequency scaling energy-efficient scheduling (G-Dvfs-ES) and non-Dfvs energy-efficient scheduling; the presented techniques can be applied to real-time systems. Another set of contributions highlighted scheduling techniques and energy optimization based on AI algorithms and heuristics, such as evolutionary computation [29], machine learning [37] and linear programming [38].

In the subsequent sections of the paper, we present our architecture with a set of dedicated scheduling mechanisms and design methodologies that make use of these algorithms. The approach allows us to control the power dissipated in the system, ensuring its predictability during the system design process.

III. SYSTEM ARCHITECTURE

Our multithread system consists of a set of reconfigurable RISC processors [24] based on interleaved pipeline processing. The number of cores of the system and the number of threads assigned to cores can be adjusted during the system configuration before the synthesis. Fig. 1 presents the scheme of the entire, regular system architecture, while the internal structure of a single core is depicted in Fig. 2. Each core processes several threads that are interleaved within its pipeline, i.e., in a given moment of time (clock cycle), different threads are processed on each stage of the pipeline. This mechanism ensures the avoidance of complicated analysis of instruction dependencies in tasks' programs, which protects the system from hazards [10]. However, general-purpose register (GPR) files should be replicated and the structure contains the thread interleaving controller (TIC) responsible for switching the tasks. We assumed that memory spaces are separated, i.e., each thread has its own program memory, but threads can communicate with one another via data memories.

A. PIPELINE AND BUS CONTROLLER

The number of stages of a given core's pipeline can be configured from the range 8–12. The propagation times of the stages responsible for the selection of the resources, namely Select Address, Select Data and Select GPR Bank, strongly depend on the number of processed threads by a given core. When a core handles a small number of threads, or when the system clock frequency is significantly below the maximal possible value, output registers of these stages may be turned off.

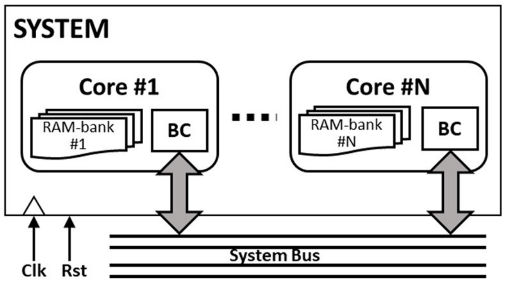


FIGURE 1. The multi-core system architecture.

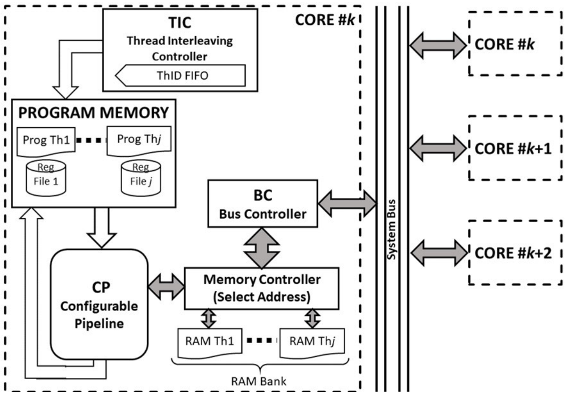


FIGURE 2. Structure of a single core.

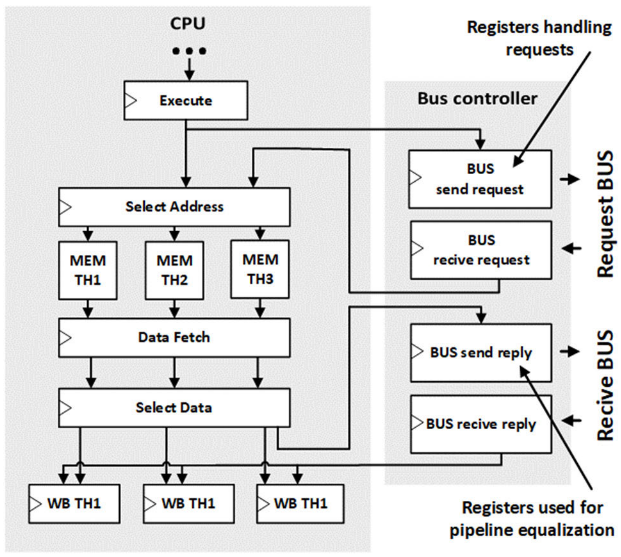


FIGURE 3. The structure of the basic, five-stage pipeline together with the bus controller.

Thus, it is possible to merge them with the next stage, resulting in a decrease of the total length of the pipeline that detailed description being found in [24]. Note that in the presented version, the communication is simplified due to the bus controllers (BC) (Fig. 3). The sequence of processed threads is stored in the queue ThID FIFO of the

TIC controller. Based on this sequence, the bus controller decides when (i.e., during what clock cycles) the banks of threads' data memories are in the idle states (are not used). These moments of time are used for data exchange on the system bus. This method of the memory system organization allows avoiding complicated memory access arbitrations mechanisms. Fig. 3 displays a fragment of the pipeline together with the cooperating bus controller.

In order to reduce the relationship between the number of threads processed by a single core, the maximum frequency of the clock and to simplify inspection of timing predictability of the system, the pipeline can be extended with additional stages, but we need to remember that a core with the extended pipeline must process the appropriate number of threads [10].

B. SIMPLIFIED COMMUNICATION BETWEEN THREADS

The new bus, in comparison to the previous solution [24], was divided into two separate buses: The Request BUS and the Receive BUS. This solution allows the elimination of the 'IDLE' state on the single bi-directional bus. The delay between a request sending and the reply delivery is fixed to a constant number of clock cycles depending on the system configuration. Therefore, this solution provides the predictability of the bus request handling. To guarantee the predictability for the request sending mechanism, every thread has its own cyclic time window for requests and the request can only be sent during this moment of time.

IV. MAIN MOTIVATION AND CONTRIBUTION OF OUR WORK

Once we developed our platform, we began its experimental testing. In [24] we presented the first version of our real-time system and some algorithms that allowed mapping many tasks into the proposed architecture, minimizing the resource utilization factor. Here we present the modified structure with simplified, inter-task data exchange. Our main goal and motivation were to obtain a methodology that allows energy-efficient mapping of tasks into the available structure while maintaining all the timing requirements of the processed tasks. Due to the fact that we focused on the safety, real-time system design process our approach concerns static scheduling problems.

The main contributions of the work are:

1) The development of an original multi-core and multitasking platform based on the threads' interleaving pipeline processing;
2) The modification of the three scheduling algorithms for different types of tasks. The new algorithms fit to the goal function of optimizing the energy consumed in the real-time system;
3) The development of two design methodologies based on the proposed heuristic algorithms;
4) A series of analyzes and experiments validating the approach.

We divided our approach into three phases:

1) **Initial phase:** covers simulations and implemen-tations of different tasks and various scenarios; during this phase, a variety of parameters are measured;
2) **Project phase:** application of the selected design methodology during which the actual design is implemented into the system with the use of the proposed algorithms;
3) **Final (post-layout) phase:** we estimate results, carry out the post-layout simulation and perform a series of analyses.

V. PRACTICAL VERIFICATION OF PLATFORM PROPERTIES

In the beginning, we carried out a series of experiments with tasks that were planned to be a part of applications working within the system platform. Detailed analyses of the results of these experiments delivered properties of the task and their parameters that can be used during the process of the actual system application design.

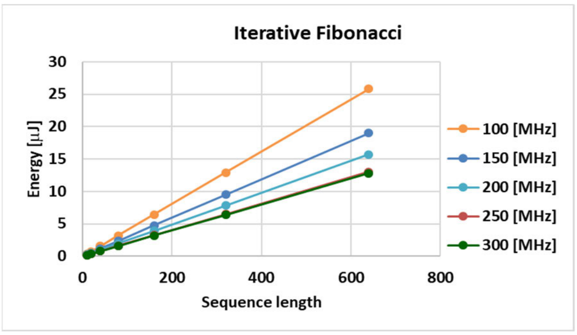
A. TASKS IDENTIFYING

These experiments were conducted to find if there exists a relationship between the energy consumed by the system and its configuration. Moreover, we wished to describe this relationship in a formal manner. As an example, we chose 60 testing programs taken from Mälardalen benchmarks [35], such as: bubble sort, cyclic redundancy check, iterative Fibonacci, etc. All of these benchmarks (tasks) were implemented in the assembler language for our microarchitecture and were compiled to machine codes (see Fig. 15 in the conclusions). Then, these codes were incorporated into the Verilog RTL model of the system and simulated in the Xilinx Vivado ISE environment.

The results of the experiments during the initial phase allowed us to formulate relationships between the amount of processed data and the number of instructions (refer to the task model in the next section) and the relationship between the average power necessary for the execution of a given task (Fig. 4-5). In all analyses and evaluations, we considered the worst-case scenarios. Moreover, the results of the experiments allowed us the express reasonable, possible to achieve deadlines for all simulated cases.

B. MULTITASKS SIMULATION

In the next part of these experiments, we constructed a set of 60 various benchmarks and divided the executed tasks into 6 different scenarios. These scenarios define various configurations of the system, i.e., they have a different number of cores and processed threads per core. To minimize the description of different scenarios we use the following code: a x b, where a represents the number of processing cores and b stands for the number of working threads in a single core. In cases 2–6 below, when the number of cores is greater than 1, tasks are executed concurrently, so naturally, we should slow down the frequency to obtain the same



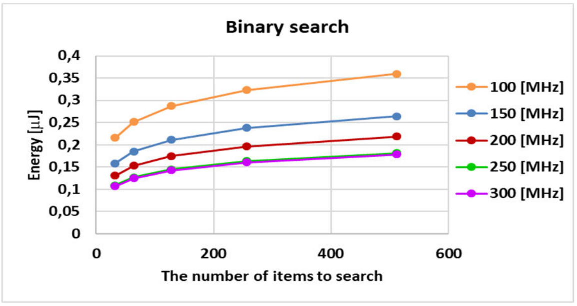
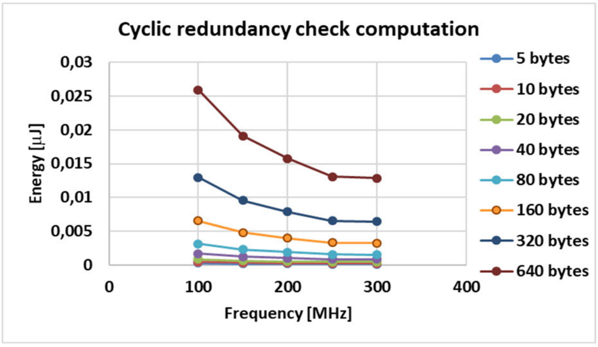


FIGURE 4. Selected tasks' properties obtained during the initial phase (dependence between energy of the task and its data size).



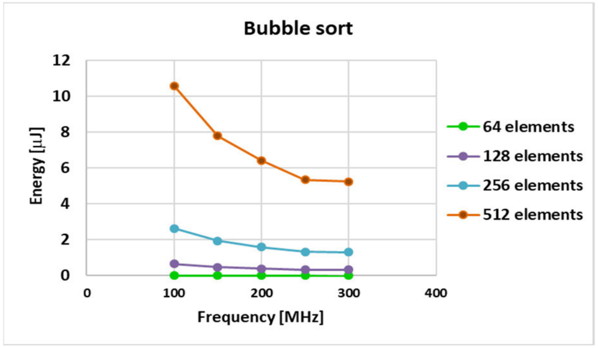


FIGURE 5. Relationship between the energy consumed by a task and the system frequency for the selected tasks.

processing time for all scenarios. In other words, we have the following:

1. 1 × 60 threads with 150 MHz clock
2. 2 × 30 threads with 75 MHz clock
3. 3 × 20 threads with 50 MHz clock
4. 4 × 15 threads with 37.5 MHz clock
5. 5 × 12 threads with 30 MHz clock
6. 6 × 10 threads with 25 MHz clock

The total execution time of the tasks for each configuration was experimentally verified by a set of simulations. The differences were very small: the values of the makespan varied within a range from 11 279 ns for scenario 1, to 11 645 ns for scenario 6 (Fig. 17). This difference comes from the effect of the pipeline initialization time and equals 10 clock cycles. However, the disproportion is respectively small (here only 3.5%) and it should tend towards zero when the execution time grows. In other words, we can assume that in each scenario, the system performed the same tasks at the same time.

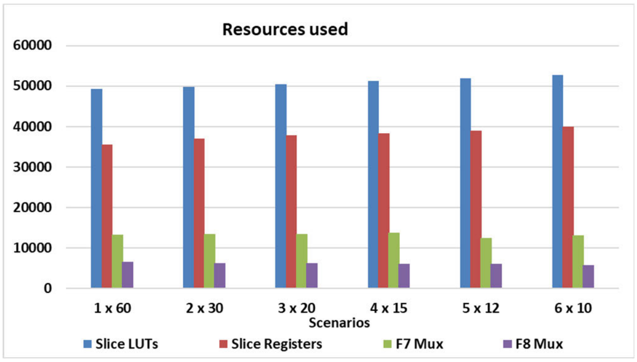


FIGURE 6. Resources utilization.

Every case, i.e., the system was configured in a way defined in the scenarios and was synthesized and implemented in the Virtex-7 XC7VX485T-2FFG1761C FPGA device. In Fig. 6, we summarized the resource utilization for all scenarios. The diagram shows that the number of logical resources utilization (Slice LUTs and Slice Registers) rises as the number of cores increases. It is connected with the necessity of the replication of the processing element, such as arithmetic-logic units or shifters. A similar tendency can be observed for switching resources (F7 Mux and F8 Mux), but in this case, the growth is significantly lower as cores with a lower number of threads do not require large multiplexors.

We made a series of probabilistic analyses of the power dissipated in the system. The obtained power prognoses are depicted in Fig. 7. We can conclude that partitioning the entire system into a greater number of processing cores allows for a decrease in the system clock frequency and a huge reduction of the energy consumed by the system for the execution of all tasks (in the current example, over 17 times larger). This shows the advantages of the configurable hardware architecture of the real-time systems. We present, in the subsequent sections, that the maximization of processing elements is the next logical step after the structure adjustment to tasks and their deadlines, in order to obtain energy-efficient, real-time applications.

Additionally, based on the initial experiments (Fig. 4-5), we can estimate the energy necessary for the execution of

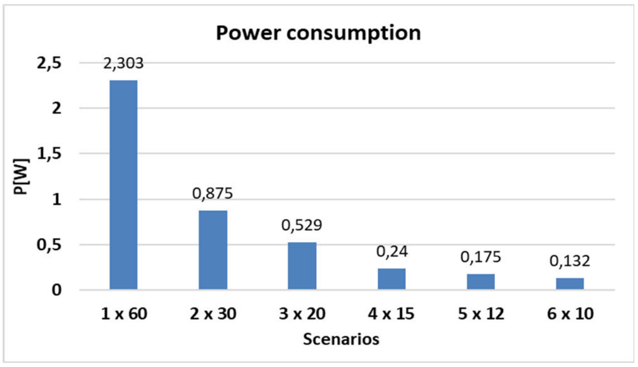


FIGURE 7. Power consumption prognoses for different scenarios.

a given task before synthesis (pre-layout) by constructing a special empirical function that estimates the relationship between the energy, the amount of data and the system frequency for a given number of processing cores (*CN*) for a specific task *i*:

$$E_{task_i} = f(data\_size, F_{sys}, CN) \tag{1}$$

VI. OPTIMIZATION GOALS–PROBLEM FORMULATION

The initial phase presented above, gave us a significant amount of information concerning the tasks processed within the system. We assume that these tasks are independent (there is no scheme fork-join) and they can use common data (i.e., see data interchange in the next section). Furthermore, we assume that all tasks can be processed concurrently and all have equal priorities. The most important parameter of all the tasks is the deadline, i.e., we require a system that guarantees 100% of timing predictability (the requirement for safety real-time systems). Generally, we deal with aperiodic tasks, however, the methodology can be simply extended to periodic tasks.

We assumed that every task, besides the empirical energy characteristic function (1), can be described by the following model [24]:

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

where:

C_i	– the program length, i.e., number of instructions when implementing a given task (except M_i);
M_i	– number of memory access instructions that refer to the data of other threads;
D_i	– deadline of the task (i.e., maximal acceptable execution time).

Another notable parameter of the system is *M_dur*. It defines the duration of the memory operations and reflects the worst-case assumptions for unpredictable access to the system memory. Thanks to the modifications described in the previous section of the paper, we can simplify this factor in comparison to [24]:

$$M_{dur} = 2 \cdot ThN + Req_{proc} + Pipeline_{start} \tag{3}$$

where:

ThN	– number of threads processed by the system;
Req_{proc}	– number of clock cycles necessary for the bus request processing for the longest pipeline;
$Pipeline_{start}$	– number of clock cycles necessary for reinitialization of the pipeline sending the request.

In case of a reconfigurable system, we applied a task model based on the number of executed instructions instead of using average or critical task execution times. Such a solution turned out to be much more convenient and the mapping of tasks was independent of the system frequency.

We also defined a task frequency [24] parameter that expresses the processing rate of a given task. As we showed in the paper, *TF* parameters (4) simplify the control over the tasks' deadlines:

$$TF_i = \frac{C_i + M_i \cdot M\_dur}{D_i} \tag{4}$$

Recently, in [24], we proposed three scheduling algorithms: BLIS, COTAS and STODER that were applied for scheduling independent tasks, co-operating tasks and scheduling in the system consisting of different, configured pipelines, respectively. All of these algorithms were designed for resource minimization under the general condition of meeting time predictability. One of the first operations is setting the minimal number of processing cores. However, in the current version, the goal is different (see next subsection) as we would like to find a solution with reduced energy consumption. Our research showed that there is a strong relationship between power dissipated in the system and the clock frequency. As we will show in the experimental part of our work, we should assume the frequency (i.e., the clock cycle duration) and then set an appropriate number of processing cores.

Thus, if we express the number of the processing cores as the ratio of the total sum of all tasks' frequencies (*TS*) and the clock cycle duration, rounded to the nearest integer, the equation becomes:

$$CN = \frac{TS}{Sys\_clk} \tag{5}$$

From this, we obtain the expression for the system frequency:

$$F_{sys} = (1 + F_{margin}) \cdot \frac{\sum_{i=1}^{N} TF_i}{CN} \tag{6}$$

This formula shows that in order to reduce the system frequency we need to increase the number of the processing cores, in order to keep all deadlines met. Moreover, we need to introduce an additional empirical parameter,

F_{margin} (approx. 4–12%) that is used for determining the system frequency F_{sys}. We address this problem in the conclusions. The experimental part of our work verifies this hypothesis.

A. GOAL FUNCTION

To formulate the goal function of our scheduler assume that: ***Th*** denotes the set of all threads executed by the system; ***PE*** is the set of all processing elements (cores); $\boldsymbol{E}(pe_i)$ represents the energy consumed by the i-th processing element; $\boldsymbol{M}(pe_i)$ denotes the mapping of tasks to a given pe_i and $pt(t_i)$ is the processing time of the task t_i. With these, we can define the goal function ***GF*** (7) as:

$$\begin{cases} \boldsymbol{GF}\left(\boldsymbol{Th}\right) \overset{\text{def}}{=} \min\limits_{M} \sum_{i}^{N} \boldsymbol{E}\left(pe_i\right) \\ \qquad\qquad \text{and} \\ \boldsymbol{PE} = \left\{pe_1, .., pe_i, .. pe_N\right\} \\ \qquad\qquad \text{and} \\ \underset{t_i \in \boldsymbol{Th}}{\forall} \; \underset{\boldsymbol{M}\left(pe_i\right)}{\exists} t_i \in \boldsymbol{M}\left(pe_i\right) \\ \qquad\qquad \text{and} \\ \underset{i \neq j}{\forall} \boldsymbol{M}\left(pe_i\right) \neq \boldsymbol{M}\left(pe_j\right) \\ \qquad\qquad \text{and} \\ \underset{t_i \in \boldsymbol{Th}}{\forall} D_i \geq pt\left(t_i\right) \end{cases} \tag{7}$$

VII. DESIGN METHODOLOGY

The process of task scheduling is one of the crucial procedures carried out during the real-time system design. The efficient scheduling process may guarantee optimal utilization of available resources and satisfy design constraints [23]–[32]. There exist many techniques and algorithms in the field of scheduling. These take into account various factors and properties of the destination platform and different processing efficiencies of the system components.

Based on the analyses of the literature we can formulate the condition of the schedulability of a given set of tasks as: "determine the set of properties of the system that allow appropriate task scheduling and fulfilling all the constraints concerning system timing, resources, power, etc."

The main taxonomy [14], distinguishes static and dynamic scheduling algorithms. The first approach dynamically maps tasks into the processing elements during the system run, while the static approach is performed during the system design; we focused on the static methods. The previous version described in [24] was based on the idea of resource minimization and accommodation of the core architecture to properties of a specific task. In the current version of the approach, we extend the methodology to the reduction of the energy dissipated by the designed system.

A. SCHEDULING OF INTERLEAVED THREADS

The specificity of interleaving pipeline processing makes the scheduling process somehow different. In such systems we cannot talk about sequential execution of tasks, margins and deadline slacks [27], [29], [32] (Fig. 8). Moreover, the order and the frequency of the appearance of the processed tasks may differ, i.e., the tasks that should be executed faster (the ratio of its instructions to the deadline is bigger) are present in the sequence more frequently than others. The thread interleaving controller block (Fig. 2) is responsible for the preparation of the appropriate sequences stored in the ThID FIFO. This solution is based on the previous experiences of our team [6]–[8]. The entire sequence consists of a fixed part that includes the identifiers of all threads processed by a given core and the configured part, where threads with shorter deadlines appear more frequently. Moreover, because of the hazard control, the minimal distance between two identical identifiers should be at least N-2, where N is the length of the pipeline. The latter requirement necessitates introducing bubbles. or idle states, into the pipeline, especially when particular tasks are terminated. This mechanism reduces the throughput but ensures predictability. Fig. 9 presents the subsequent stages of the pipeline processing and timeline of 10 different tasks scheduled to a given core. The diagram presents a 12-stage pipeline, so to eliminate the problem of dynamic hazards, e.g., control or data, we need to map at minimum, 9 different tasks into the core (see Fig. 3).

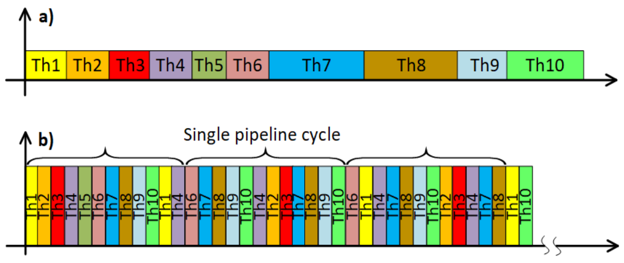

a)
Th1
Th2
Th3
Th4
Th5
Th6
Th7
Th8
Th9
Th10
b)
Single pipeline cycle
Th1 Th2 Th3 Th4 Th5 Th6 Th7 Th8 Th9 Th10 Th1 Th4 Th6 Th7 Th8 Th9 Th10 Th4 Th2 Th3 Th7 Th8 Th9 Th10 Th6 Th1 Th4 Th7 Th8 Th9 Th10 Th2 Th3 Th4 Th7 Th8 Th1 Th10

FIGURE 8. Sequential scheduling of tasks (a) and interleaved tasks scheduling (b).

B. ENERGY EFFICIENT SCHEDULING

As the above section defined our goal function (7), we can modify our scheduling algorithms.

In the case of BLIS algorithms that concern the simplest case (i.e., threats not interacting with one another), we should modify the beginning of the procedure as shown in Fig. 10. We also balanced the initial tasks assignment mechanism (Step 7). Moreover, thanks to the new inter-thread communication mechanism, we can avoid time-consuming iteration as the *M_dur* parameter is independent of *CN*. Because of the latter issue, the COTAS II and BLIS II algorithms differs only in one step (Step 1), necessary for *M_dur* calculation [24].

Fig. 11 contains the modified version of the STODER II algorithm. This set of modified scheduling algorithms I-III is the toolset for the energy reduction design methodologies.

TABLE 1. Description of tasks in the tested example.

nr	algorithm[a)]	C_i	D_i [µs]	TF_i	nr	algorithm[a)]	C_i	D_i [µs]	TF_i	nr	algorithm[a)]	C_i	D_i [µs]	TF_i
1	fibcall(19)	152	17.712	8.58	18	b_search(8)	34	4.62	7.36	35	fibcall(161)	634	200.716	3.16
2	CRC(94)	861	148.193	5.81	19	duff(159)	600	69.628	8.62	36	CRC(82)	754	293.385	2.57
3	duff(194)	720	131.627	5.47	20	CNT(6x6)	323	191.124	1.69	37	CRC(80)	734	167.198	4.39
4	duff(16)	153	20.788	7.36	21	b_sort(12)	1221	138.377	8.82	38	duff(87)	375	47.051	7.97
5	duff(16)	153	20.4	7.5	22	fibcall(63)	565	94.958	5.95	39	CRC(49)	453	80.6	5.62
6	duff(16)	153	18.836	8.12	23	CRC(3)	40	5.442	7.35	40	duff(87)	375	100.806	3.72
7	fibcall(45)	401	46.231	8.67	24	CRC(3)	40	4.975	8.04	41	b_sort(13)	1232	150.272	8.2
8	fibcall(45)	401	78.016	5.14	25	CRC(2)	32	4.348	7.36	42	fibcall(30)	354	40.638	8.71
9	fibcall(45)	401	55.259	7.26	26	b_sort(10)	710	101.553	6.99	43	fibcall(86)	768	108.322	7.09
10	CNT(9)	600	72.631	8.26	27	b_sort(10)	710	110.059	6.45	44	CRC(61)	564	69.544	8.11
11	CRC(36)	334	49.408	6.76	28	fibcall(100)	900	150.03	6	45	duff(87)	375	168.161	2.23
12	CRC(24)	232	28.97	8.01	29	duff(111)	452	60.775	7.44	46	fibcall(41)	364	244.295	1.49
13	CRC(49)	453	53.628	8.45	30	b_sort(14)	1521	180.053	8.45	47	fibcall(96)	864	100.816	8.57
14	fibcall(2)	10	1.633	6.12	31	CRC(70)	643	105.757	6.08	48	duff(80)	347	42.014	8.26
15	CRC(8)	88	19.172	4.59	32	fibcall(61)	545	90.485	6.02	49	duff(239)	865	220.663	3.92
16	CRC(8)	88	25.593	3.44	33	fibcall(61)	545	118.21	4.61	50	fibcall(51)	456	101.261	4.5
17	b_search(8)	34	4.266	7.97	34	duff(22)	123	20.466	6.01					

[a)]The symbols are taken from the Mälardalen benchmarks [35]. In the brackets, we show the size of data in Bytes

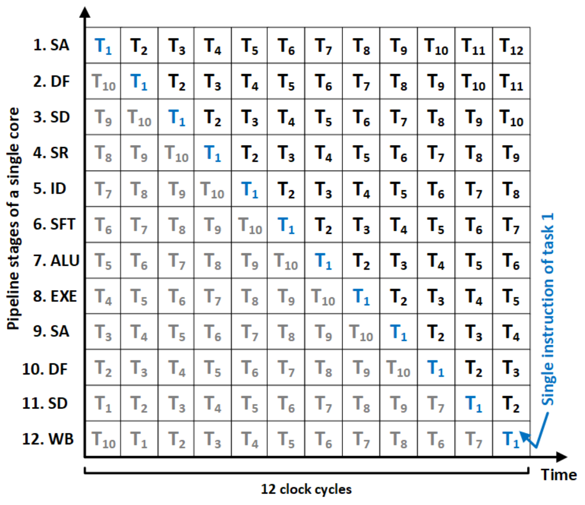

Pipeline stages of a single core
1. SA: T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12
2. DF: T10 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11
3. SD: T9 T10 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10
4. SR: T8 T9 T10 T1 T2 T3 T4 T5 T6 T7 T8 T9
5. ID: T7 T8 T9 T10 T1 T2 T3 T4 T5 T6 T7 T8
6. SFT: T6 T7 T8 T9 T10 T1 T2 T3 T4 T5 T6 T7
7. ALU: T5 T6 T7 T8 T9 T10 T1 T2 T3 T4 T5 T6
8. EXE: T4 T5 T6 T7 T8 T9 T10 T1 T2 T3 T4 T5
9. SA: T3 T4 T5 T6 T7 T8 T9 T10 T1 T2 T3 T4
10. DF: T2 T3 T4 T5 T6 T7 T8 T9 T10 T1 T2 T3
11. SD: T1 T2 T3 T4 T5 T6 T7 T8 T9 T7 T1 T2
12. WB: T10 T1 T2 T3 T4 T5 T6 T7 T8 T6 T7 T1
Single instruction of task 1
12 clock cycles
Time

FIGURE 9. Interleaved 12-stage pipeline processing 10 different tasks.

We explain these methodologies through an example. To facilitate the clearance of the differences between the approaches, we use the same example of tasks. The parameters of tasks were obtained utilizing the technique described above (see section V).

C. DESCRIPTION OF THE TESTED EXAMPLE

To simplify the analysis and make it more readable, we assumed that the tasks were independent ($M_i = 0$). This does not impact the generality of the considerations.

Our example consists of 50 different tasks based on the Mälardalen benchmarks [35] described in Table 1. The table also contains numbers of instructions (C_i), tasks' deadlines (D_i) and tasks' frequencies (TF_i).

D. SIMPLE APPROACH

When the set of energy constraints is empty (i.e., a simple design) we can directly use the algorithm BLIS II. We should start by determining the matrix, $\mathbb{TF}$, of task frequencies (see in Table 1).

And after sorting we obtain:

$$\begin{aligned}\mathbb{TF}_{\text{sorted}} = \{&\mathrm{TF}_{46}, \mathrm{TF}_{20}, \mathrm{TF}_{45}, \mathrm{TF}_{36}, \mathrm{TF}_{35}, \mathrm{TF}_{16}, \mathrm{TF}_{40},\\ &\mathrm{TF}_{49}, \mathrm{TF}_{37}, \mathrm{TF}_{50}, \mathrm{TF}_{15}, \mathrm{TF}_{33}, TF_{8}, \mathrm{TF}_{3}, \mathrm{TF}_{39},\\ &\mathrm{TF}_{2}, \mathrm{TF}_{22}, \mathrm{TF}_{28}, \mathrm{TF}_{34}, \mathrm{TF}_{32}, \mathrm{TF}_{31}, \mathrm{TF}_{14}, \mathrm{TF}_{27},\\ &\mathrm{TF}_{11}, TF_{26}, \mathrm{TF}_{43}, \mathrm{TF}_{9}, \mathrm{TF}_{23}, \mathrm{TF}_{18}, \mathrm{TF}_{25}, \mathrm{TF}_{4},\\ &\mathrm{TF}_{29}, TF_{5}, \mathrm{TF}_{17}, \mathrm{TF}_{38}, \mathrm{TF}_{12}, \mathrm{TF}_{24}, \mathrm{TF}_{44}, \mathrm{TF}_{6},\\ &\mathrm{TF}_{41}, \mathrm{TF}_{48}, \mathrm{TF}_{10}, \mathrm{TF}_{13}, \mathrm{TF}_{30}, \mathrm{TF}_{47}, \mathrm{TF}_{1}, \mathrm{TF}_{19},\\ &\mathrm{TF}_{7}, \mathrm{TF}_{42}, \mathrm{TF}_{21}\}\end{aligned}$$

The parameter TS for all tasks equals 319.26 and was determined in Step 1. With this, we can determine the number of cores and set up the clock frequency (Steps 2–3). Based on the system analysis [24] we can assume that, for 50 threads working in the system, consisting of 12-stage pipelines, the maximal frequency can be equal to approximately 200 MHz. This information allows us to take 2 cores (the minimal number of cores that can assure time predictability). For the maximal safety margin ($F_{marg} = 12\%$) we have $F_{sys} \approx 179$ MHz ($F_{sys} \leq F_{max}$). For further evaluation, we assume 180 MHz.

```
Algorithm I (II*): BLIS II (COTAS II*) – tasks scheduling

Given a set of N hardware tasks $\mathbb{T} = \{T_1, T_2, .. T_N\}$ described by
pairs of parameters, i.e., $T_i = \{C_i, D_i\}$ and the hardware platform
with the set of processing cores $\mathbb{PE} = \{pe_1, pe_2, .. pe_k\}$.
The goal: determine necessary resources (number of
          processing cores) and allocate all tasks to them.

PHASE 1 (Initial allocation)
{(*)Step 1  In case of cooperating tasks use the COTAS II version
            Calculate the $M\_dur$ from equation (3)}
Step 1'  Based on the matrix $\mathbb{T}$, determine the matrix $\mathbb{TF}$, of
         task frequencies; find the total sum of all elements:
             $TS = \sum_{i=1}^{N} TF_i$
Step 2  Calculate the processing frequency of the system:
             $F_{sys} = (1 + F_{margin}) \cdot \frac{\sum_{i=1}^{N} TF_i}{CN}$
Step 3  Check if the processing frequency $F_{sys} \le F_{max}$:
        If not increase CN and go return to Step 2
Step 4  Find the mean value, $TF_{MEAN}$, of elements in the matrix
        $\mathbb{TF}$
Step 5  Re-arrange the matrix $\mathbb{TF}$ in ascending order of the
        task frequency parameters ($TF_i$)
Step 6  for $k = 0$ to $CN-1$ do loop
        6.1. Take the core # k, and allocate to it tasks with the
             highest and lowest value of TF from $\mathbb{TF}_{sorted}$, i.e.,
             take the last task with highest $TF_i$ and the first one
             with the lowest $TF_i$, alternately. During these
             jumps, control the SUM of $TF_i$ of the selected
             tasks
        6.2. If $SUM > F_{sys}$ move back one jump (neglect the
             last jump and try to take the next task with the
             lower value of $TF_i$)
        6.3. Remove all of the already allocated tasks from the
             matrix $\mathbb{TF}_{sorted}$
   If $\mathbb{TF} = \varnothing$ end loop else repeat Step 6.

PHASE 2 (Load balancing)
Step 7  Find the relative difference between the most loaded
        (ML) and the least loaded (LL) cores $\delta L$:
             $\delta L = \frac{[SUM_{ML} - SUM_{LL}]}{F_{SYS}} \cdot 100\%$
Step 8   Until $\delta L > \varepsilon$
Step 9   Flip tasks between the most and the least loaded core
Step 10 End.
```

FIGURE 10. BLIS II and COTAS II scheduling algorithms for energy reduction.

Eventually, after PHASE 2 (the load balancing procedure) we obtain the following assignments of tasks to cores:

$$\text{Core\#0} \leftarrow \{T_{21}, T_{19}, T_{35}, T_1, T_{40}, T_{13}, T_{49}, T_{48}, T_{15}, T_{41},$$

```
Algorithm III: STODER II - tasks scheduling

Given a set of N hardware tasks $\mathbb{T} = \{T_1, T_2, .. T_N\}$ described by
the triple parameters, i.e. $T_i = \{C_i, M_i, D_i\}$ and the hardware
platform with the system clock Sys_clk. The platform has
already defined set of dedicated cores with final assignments:
    DCore #1 ← $\{T_{11},\dots, T_{1,k1}\}$;
    …
    DCore #i ← $\{T_{i1},\dots, T_{i,ki}\}$.
The goal: determine necessary resources (number of processing
          cores) and allocate all the remaining tasks to them.

Step 1   Evaluate the $M\_dur$ from the equation (3),
Step 2   Check if all so far defined dedicated DCores can
         handle all assigned to them tasks. (If not replicate
         necessary DCore),
Step 3   Verify if all remaining (so far not assigned) tasks can
         be implemented in the DCores (if not define another
         standard Core #(i +1) ),
Step 4   Determine the matrix, $\mathbb{TF}$, of task frequencies; Find
         the total sums of all elements for task that are (can be)
         allocated in DCores and for tasks that can be mapped
         to standard Cores (a set $\mathbb{ST}$):
             $DTS = \sum_{i=1}^{i} TF_i;\quad STS = \sum_{i=1}^{CN} TF_i$
Step 5  Calculate the processing frequency of system, i.e.:
             $F_{sys} = (1 + F_{margin}) \cdot \frac{DTS + STS}{CN}$
Step 6   First supplement DCores using balanced philosophy
         (BLIS based approach),
Step 7   Find the mean value $TF_{MEAN}$ of elements for the set $\mathbb{ST}$
         in the sub-matrix $\mathbb{STF}$
Step 8   Re-arrange the matrix $\mathbb{STF}$ in ascending order of the
         task frequency parameters ($TF_i$)
Step 9   for all elements stored in $\mathbb{STF}_{sorted}$ execute Step 6
         of the BLIS II algorithm
Step 10  Execute Phase 2 of the BLIS II algorithm for Cores,
         but limited only to the allowed moves.
Step 11  End.
```

FIGURE 11. STODER II scheduling algorithm for energy reduction.

$$T_{33}, T_6, T_{39}, T_{12}, T_2, T_{38}, T_{34}, T_{29}, T_{32}, T_4, T_{27}, T_{23}, T_{11}, T_{43}\}\ \textit{with SUM} = 160.94 < F_{sys};$$

and

$$\text{Core\#1} \leftarrow \{T_{20}, T_{42}, T_{45}, T_7, T_{16}, T_{30}, T_{37}, T_{10}, T_{50}, T_{48}, T_8, T_{44}, T_3, T_{24}, T_{22}, T_{17}, T_{28}, T_5, T_{31}, T_{25}, T_{14}, T_{18}, T_{26}, T_9, T_{46}, T_{36}\}\ \textit{with SUM} = 158.32 < F_{sys}.$$

From the result of applying this simple methodology, we obtained a 2-core structure, where the first processes 24 different threads, while 26 threads are executed in the second core. Based on the previous experiments, for the given set of tasks, we can forecast the average power consumed by the system with a frequency of180 MHz at the level 1.95 W.

E. SFERA METHODOLOGY

Our first proposal of the design methodology of energy-efficient, real-time systems was called SFERA (Scheduling for Energy Reduced Applications). The algorithm consists of five steps, shown in Fig. 12, that realize our goal function defined by (7). The method combines scheduling for time predictability with searching for threads mapping; this process assures energy reduction. However, we would like the structure of the system to remain as compact as possible by using a minimal number of cores. The SFERA approach makes use of all three modified scheduling algorithms: BLIS II, COTAS II and STODER II.

Methodology I: SFERA

```
Step 1  **Set** the frequency of the system clock (use the
        reasonable, i.e. possible parameter for your platform)
Step 2  According to the specificity of the application and
        tasks executed by the system **run one** of the
        scheduling algorithms (BLIS II, COSTAS II or
        STODER II)
Step 3  Estimate total power consumed by the system:
        $P_{total} = \sum_{i}^{N} \boldsymbol{P}(pe_i)$
Step 4  **If** $P_{total} \leq P_{constraint}$ **terminate**
Step 5  **If not** increase $CN$ and **go back** to Step 1
```

FIGURE 12. First system design methodology: SFERA for scheduling tasks in the regime of efficient energy dissipation with a small amount of resource utilization.

The SFERA methodology is useful, when we have the energy constraint (this parameter is required). Here, we have assumed that the maximum average power consumed by the tasks cannot exceed 0.8 W. In the first step, we must find the maximum allowed frequency, 180 MHz, the same as in the subsection D (section V) for the simple approach. Then the appropriate scheduling algorithm should be selected. Since we wished to compare both methods, we assume the same tasks and choose BLIS II. Then we should execute subsequent steps (1–6), namely: 1) calculate $TS = 319.26$; 2) assume the number of cores – we start from $CN = 3$; 3) estimate the processing frequency for the experimentally selected margin $F_{marg} = 10\%$ we have $F_{sys} \approx 117$ MHz; 4) check if $F_{sys} \leq F_{max}$ (the frequency is allowed); 5–6) sort $\mathbb{TF}$ matrix – the result is the same as previously in BLIS II (compare to $\mathbb{TF}_{\text{sorted}}$). This methodology requires two iterations. After the first mapping we obtain:

$$\begin{aligned}
\text{Core \#0} \leftarrow \{&\text{T}_{46}, \text{T}_{16}, \text{T}_{47}, \text{T}_{40}, \text{T}_{30}, \text{T}_{33}, \text{T}_{6}, \text{T}_{8}, T_{44}, T_{28},\\
&\text{T}_{5}, \text{T}_{34}, \text{T}_{29}, \text{T}_{11}, \text{T}_{9}, \text{T}_{11}, \text{T}_{26}\}\\
&\textit{with SUM} = 106.37;\\
\text{Core \#1} \leftarrow \{&\text{T}_{20}, \text{T}_{42}, \text{T}_{35}, \text{T}_{1}, \text{T}_{49}, \text{T}_{13}, \text{T}_{15}, \text{T}_{41}, \text{T}_{3}, \text{T}_{24},\\
&\text{T}_{22}, \text{T}_{17}, \text{T}_{32}, \text{T}_{25}, \text{T}_{27}, \text{T}_{23}, \text{T}_{45}\}\\
&\textit{with SUM} = 104.14;\\
\text{Core \#2} \leftarrow \{&\text{T}_{45}, \text{T}_{7}, \text{T}_{36}, \text{T}_{19}, \text{T}_{37}, \text{T}_{48}, \text{T}_{50}, \text{T}_{10}, \text{T}_{39}, \text{T}_{12},\\
&\text{T}_{2}, \text{T}_{38}, \text{T}_{18}, \text{T}_{31}, \text{T}_{4}, \text{T}_{14}, \text{T}_{21}\}\\
&\textit{with SUM} = 108.42.
\end{aligned}$$

This structure consists of three cores: Core #0 with 17 threads, Core #1 with 17 threads and Core #2 with 16 threads. The total power consumed by this implementation (Step 3 of the SFERA algorithm) was estimated with the average power predicted for this case being 1.15 W. However, this value does not satisfy the power constraint; thus, the value of cores must be incremented and the SFERA algorithm restarted. The new value of the frequency, F_{sys}, is 88 MHz. We need to assign tasks to 4 cores:

$$\begin{aligned}
\text{Core \#0} \leftarrow \{&\text{T}_{46}, \text{T}_{21}, \text{T}_{49}, \text{T}_{13}, \text{T}_{37}, \text{T}_{48}, \text{T}_{2}, \text{T}_{38}, \text{T}_{22}, \text{T}_{17},\\
&\text{T}_{31}, \text{T}_{9}, \text{T}_{35}\}\ \textit{with SUM} = 79.53;\\
\text{Core \#1} \leftarrow \{&\text{T}_{20}, \text{T}_{42}, \text{T}_{40}, \text{T}_{30}, \text{T}_{50}, \text{T}_{10}, \text{T}_{39}, \text{T}_{12}, \text{T}_{28}, \text{T}_{5},\\
&\text{T}_{14}, \text{T}_{23}, \text{T}_{16}\}\ \textit{with SUM} = 79.37;\\
\text{Core \#2} \leftarrow \{&\text{T}_{45}, \text{T}_{7}, \text{T}_{47}, \text{T}_{15}, \text{T}_{41}, \text{T}_{3}, \text{T}_{24}, \text{T}_{34}, \text{T}_{29}, \text{T}_{27},\\
&\text{T}_{4}, \text{T}_{43}\}\ \textit{with SUM} = 80.12;\\
\text{Core \#3} \leftarrow \{&\text{T}_{36}, \text{T}_{19}, \text{T}_{1}, \text{T}_{33}, \text{T}_{6}, \text{T}_{8}, \text{T}_{44}, \text{T}_{32}, \text{T}_{25}, \text{T}_{11},\\
&\text{T}_{18}, \text{T}_{26}\}\ \textit{with SUM} = 80.24.
\end{aligned}$$

The final structure consists of four cores: Core #0 with 14 threads, Core #1 with 13 threads, Core #2 with 12 threads and Core #3 with 11 threads. The presumed value of the average power dissipated by this implementation is approximately 0.78 W; thus, it meets the demanded constraint (0.8 W) and we may terminate the scheduling.

F. MAXPRO METHODOLOGY

A brief analysis of the first experiments concluded that the dynamic energy of the system can be reduced when we decrease the frequency. However, in the time-predictable application cases, the necessity of increasing the number of processing cores occurs. Therefore, by using the highest possible number of cores we can obtain the most energy-efficient implementation of the system.

OWith these results, we propose another strategy of starting from the selection of a maximal number of processing cores. The only limitation is the minimal number of threads that should be mapped into the core. As it was mentioned above, due to the threads interleaving mechanism, we need to determine the appropriate number of processed threads within a given pipeline [10], followed by finding the balanced load of all cores (PHASE 2 of the BLIS II algorithm). Next, for the most loaded core (ML), calculate the working frequency of the system and finally, check the energy consumed by the system. This philosophy is represented by the third algorithm, MAXPRO (MAXimum PROcessing units minimum energy).

The first step of the third algorithm – MAXPRO is determining the maximal number of cores that can be applied. At the beginning (p. B) we assumed a 12-stage pipeline;

TABLE 2. Resources utilization comparison for used design methods.

	BLIS	SFERA	MAXPRO
Slice LUTS	41708	43374	43904
SLICE Registers	30941	32343	33213
F7 Muxes	11185	11155	10971
F8 Muxes	5395	4824	4818

this means that a given pipeline should process at least 10 different and interleaved threads in order to avoid hazards [10]. This simple consideration leads to the conclusion that, for 50 hardware threads, we should take a maximum of 5 cores. After the step 2 we obtain the following mapping:

$$\text{Core \#0} \leftarrow \{T_{46}, T_{21}, T_{16}, T_{47}, T_{41}, T_{2}, T_{38}, T_{31}, T_{25,}\boldsymbol{T_{22}},\}$$
$$\text{with } SUM = 63.69;$$
$$\text{Core \#1} \leftarrow \{T_{20}, T_{42}, T_{40}, T_{30}, T_{33}, T_{6}, T_{17}, T_{14}, T_{18,}\ \boldsymbol{T_{43}}\}.$$
$$\text{with } SUM = 63, 84;$$
$$\text{Core \#2} \leftarrow \{T_{45}, T_{7}, T_{49}, T_{13}, T_{8}, T_{44}, T_{28}, T_{5}, T_{27}, T_{23}\}.$$
$$\text{with } SUM = 63.82;$$
$$\text{Core \#3} \leftarrow \{T_{36}, T_{19}, T_{37}, T_{10}, T_{3}, T_{24}, T_{34}, T_{29}, T_{11}, T_{9}\}.$$
$$\text{with } SUM = 64.81;$$
$$\text{Core\#4} \leftarrow \{T_{35}, T_{1}, T_{50}, T_{48}, T_{39}, T_{12}, T_{32}, T_{4}, T_{26}, \boldsymbol{T_{15}}\}$$
$$\text{with } SUM = 63.09.$$

The maximally loaded core is Core #3 – its $SUM = 64.81$, thus we can set the system frequency (F_{sys}) to 65 MHz. The predicted average power consumed by this implementation of the system equals approximately 0.49 W.

G. POST-LAYOUT ESTIMATION–EXAMPLE SUMMARY

This section explains the last stage of our methodology, namely post-layout. All the implementations of the system, described above, were synthetized to an FPGA Virtex7 device. The numbers of used resources are depicted in Table 2.

As expected, based on the previous research (compare to [24] and section IV.B), the structure containing more processing cores requires more logical resources (Slice LUTs and Slice Registers); while the amount of switching resources (F7 Mux and F8 Mux) grows when the ratio of threads per core is larger. However, the differences do not exceed 10% for each of the considered cases. The first approach, BLIS, generates results very quickly and this approach can be used in conjunction with a smaller number of resources and the energy consumed in the system isn't important. The second method, SFERA, combines two factors: reduced energy and the minimization of resources. The main drawback of this approach is the necessity of tedious, iterative calculations, but finally, it reaches the energy constraint. The third methodology, MAXPRO, obtains the energy-efficient solution immediately, but at the expense of a relatively big structure. All methods give time-predictable devices, i.e., the execution times of all tasks are the same; thus, there exists a direct relationship between the energy consumed in a given solution and the system configuration. For all examined and presented cases we estimated the average power dissipated by the system utilizing professional Xilinx Vivado ISE tools (Fig. 14). We gathered the estimated values and documented them in Table 3. The results show that our prognoses are pessimistic. However, the estimated values based on the probabilistic estimations are closer to the actual values.

TABLE 3. Average power consumed in the three implementations.

	Simple method	SFERA	MAXPRO
Prognosed power [W]	1.95	0.78	0.49
Estimated actual power [W]	1.56	0.55	0.29

Methodology II: MAXPRO

Given a set of N hardware tasks $\mathbb{T} = \{T_1, T_2, .. T_N\}$ described by pairs of parameters, i.e. $T_i = \{C_i, D_i\}$ and the hardware platform with the set of processing cores $\mathbb{PE} = \{pe_1, pe_2, .. pe_k\}$.

The goal: determine the most economical implementation of the system in terms of energy consumption.

Step 1 **Analyze** a set of N hardware tasks $\mathbb{T} = \{T_1, T_2, .. T_N\}$ and find the maximal number of cores (find initial mapping),

Step 2 Depending on the tasks' and cores' structure run appropriate scheduling algorithm **BLIS II (COTAS II)** or **STODER II**

Step 3 For the most loaded core (ML) find the sum of TF_i tasks assigned to it, i.e.:

$$SUM_{ML} = \sum_{i=1}^{N\_ML} TF_i$$

Step 4 **Calculate** the processing frequency of system, i.e.:

$$F_{sys} = SUM_{ML}$$

Step 5 Estimate total power dissipated the system:

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

Step 6 **End.**

FIGURE 13. Second system design methodology: MAXPRO for scheduling tasks in the regime of minimal energy dissipation with the maximal number of processing cores.

VIII. EXPERIMENTS AND RESULTS

To verify our approach, we carried out a series of different experiments. The scheduling algorithms used by the design methodologies, described in the previous section, were implemented as heuristic programs in the LPA Prolog language [39], based on the backtracking mechanism. Following this, all prepared system configurations (scenarios) were implemented in the Verilog model of our platform. For the device, we used a Xilinx Virtex 7 FPGA working

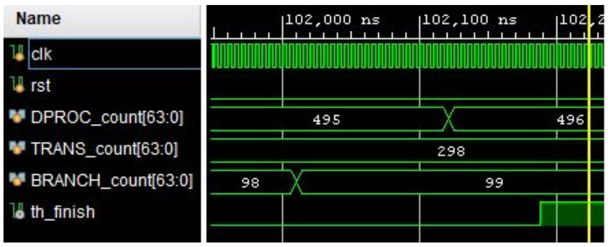
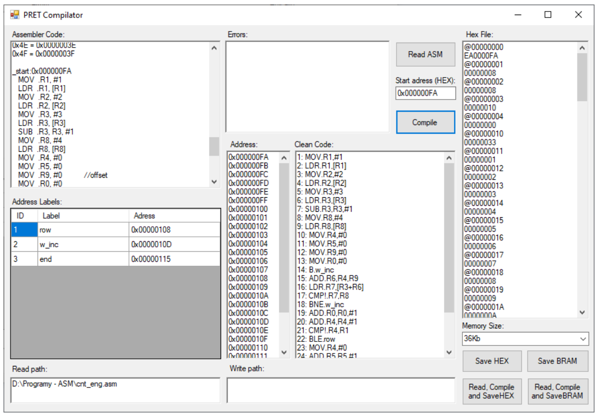


FIGURE 14. Example of instructions counting (Vivado simulator).

on the platform VC707. The entire system was modeled in Verilog and synthesized in the Vivado ISE environment. Tasks were implemented as assembler programs and compiled into machine code by the special compiler implemented in C++ (Fig. 15). During the task identification described in section V, special simulation models are executed in the Vivado ISE simulator. Those models were supplied with special counters, which tallied the tasks' length (number of instructions). Moreover, we can identify the type of program instructions (Fig. 14). After a system was configured, i.e., we selected the number and types of processing cores, and mapped tasks (machine codes of programs) into the local memories, we synthesized it into the FPGA platform and analyzed the obtained parameters.



Assembler Code:

```
0x4E = 0x0000003E
0x4F = 0x0000003F

_start:0x000000FA
  MOV  .R1, #1
  LDR  .R1, [R1]
  MOV  .R2, #2
  LDR  .R2, [R2]
  MOV  .R3, #3
  LDR  .R3, [R3]
  SUB  .R3, R3, #1
  MOV  .R8, #4
  LDR  .R8, [R8]
  MOV  .R4, #0
  MOV  .R5, #0
  MOV  .R9, #0        //offset
  MOV  .R0, #0
```

Address Labels:

ID	Label	Adress
1	row	0x00000108
2	w_inc	0x0000010D
3	end	0x00000115

Read path: D:\Programy - ASM\cnt_eng.asm

Start adress (HEX): 0x000000FA

Address:	Clean Code:
0x000000FA	1: MOV.R1,#1
0x000000FB	2: LDR.R1,[R1]
0x000000FC	3: MOV.R2,#2
0x000000FD	4: LDR.R2,[R2]
0x000000FE	5: MOV.R3,#3
0x000000FF	6: LDR.R3,[R3]
0x00000100	7: SUB.R3,R3,#1
0x00000101	8: MOV.R8,#4
0x00000102	9: LDR.R8,[R8]
0x00000103	10: MOV.R4,#0
0x00000104	11: MOV.R5,#0
0x00000105	12: MOV.R9,#0
0x00000106	13: MOV.R0,#0
0x00000107	14: B.w_inc
0x00000108	15: ADD.R6,R4,R9
0x00000109	16: LDR.R7,[R3+R6]
0x0000010A	17: CMP!.R7,R8
0x0000010B	18: BNE.w_inc
0x0000010C	19: ADD.R0,R0,#1
0x0000010D	20: ADD.R4,R4,#1
0x0000010E	21: CMP!.R4,R1
0x0000010F	22: BLE.row
0x00000110	23: MOV.R4,#0
0x00000111	24: ADD.R5,R5,#1

Write path:

Hex File:

```
@00000000
EA0000FA
@00000001
00000008
@00000002
00000008
@00000003
00000010
@00000004
00000000
@00000010
00000033
@00000011
00000001
@00000012
00000002
@00000013
00000003
@00000014
00000004
@00000015
00000005
@00000016
00000006
@00000017
00000007
@00000018
00000008
@00000019
00000009
@0000001A
0000000A
```

Memory Size: 36Kb

FIGURE 15. Tasks the compiler used in the system configuration.

The set of characteristics depicted in Fig. 16 show the linear relationship between the system frequency and the average power dissipated in the device. This relation was already reported in the literature [29]. In our case, we can express the dynamic power by:

$$P_{dynamic} = \alpha \cdot C_L \cdot V_{DD}^2 \cdot F_{sys} \tag{8}$$

where:

α	denotes the switching activity of the circuit,
C_L	represents the switching capacity (a constant)
and V_{DD}	is supply voltage.

It can be noted that the rate of rise (dP/dF) grows as the number of processed threads increases. This can be explained by the growing switching activity for the larger number of processed tasks. Every point of a given curve in Fig. 16 corresponds to a power iteration of the algorithm SFERA.

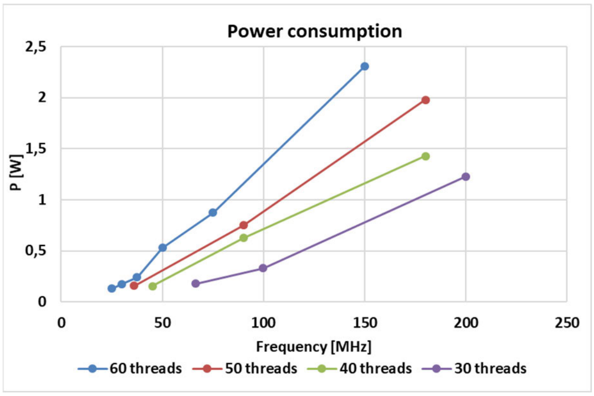


FIGURE 16. Waveforms showing the relationship between power consumed in the system and the frequency for a different number of threads (all obtained by the SFERA algorithm).

Fig. 17 shows the correspondence between the energy and the system configuration (scenario) in time-predictable systems. These examples of waveforms shows that all tasks are ready before the deadline (the red line); although, when the frequency is bigger, they are completed earlier (see the yellow stamps denoting the tasks' termination time).

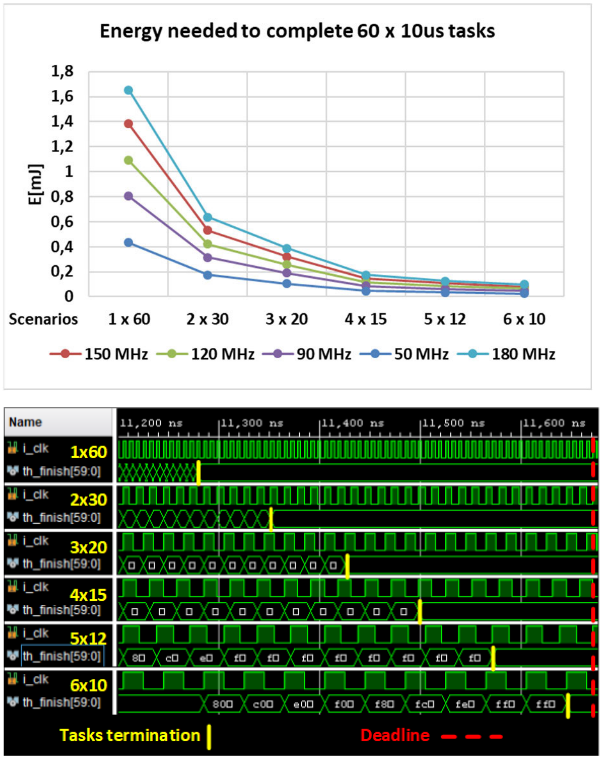


FIGURE 17. The total energy consumed by the system processing 60 different tasks during 10 microseconds in various scenarios (the waveform was taken from Vivado ISE simulator).

Fig. 18 and Fig. 19 present relationships between power and the system configuration for a single-core architecture and the multi-core structure, respectively. These diagrams show how the energy dissipated by the real-time systems can be controlled and how we may consider possible savings during the system design.

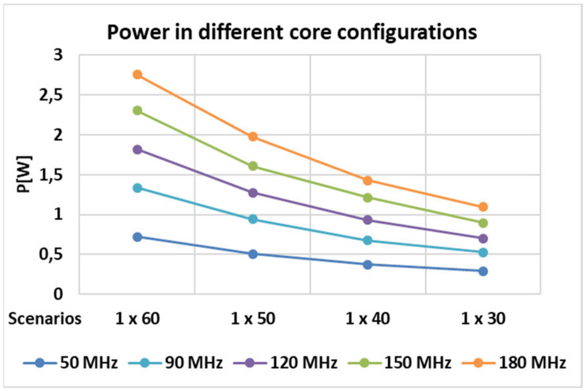


FIGURE 18. **Average power consumed in a single core for a different number of processed threads.**

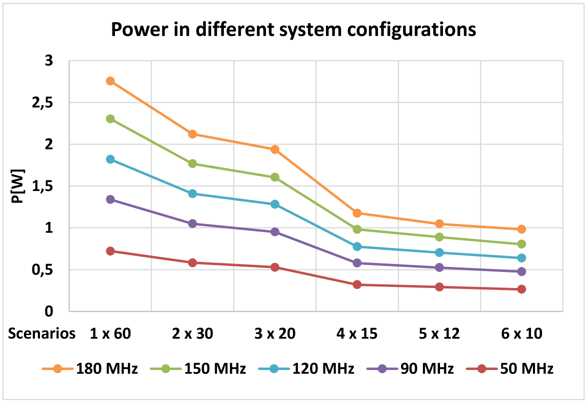


FIGURE 19. **Average power dissipated by the system depending upon the configuration.**

Fig. 20 shows that the forecasted values of the power dissipated by a given implementation of the system are pessimistic. Based on the tasks' characteristics obtained in the initial phase and assuming that there are N tasks and the k-th task consumes power p_k, we have:

$$P_{forecast} = \sum_{k=1}^{N} p_k \tag{9}$$

The actual values estimated for the implemented system in hardware are smaller, so the implementation always meets the constraints.

Due to dealing with real-time systems, the deadlines of the tasks are very important. On the other hand, the processing time of a given task is strongly connected with the number of instructions that need to be executed. The relationship between the number of instructions and data size handled by a task depends on the type of a given task and may vary (Fig. 21 shows this property for different benchmarks). Data sets were generated randomly. The appropriate configuration of the simulation models of the tasks allowed us to calculate these parameters (Fig. 22).

Fig. 23 presents how the relative change of the system frequency affects the overall system predictability. We introduced a borderline at point 0 % for the theoretical frequency F_{border} defined as:

$$F_{border} = \frac{\sum_{i=1}^{N} TF_i}{CN} \tag{10}$$

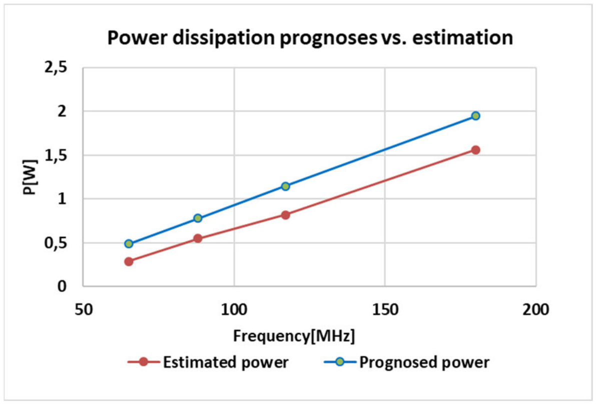


FIGURE 20. **Diagrams presenting the prognosed and estimated average power for different system frequencies.**

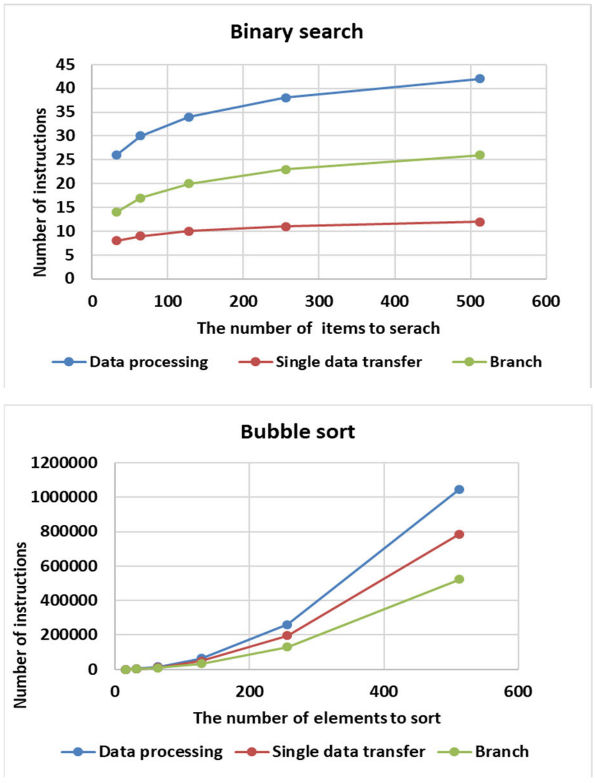


FIGURE 21. **Diagrams showing the relationship between the task complexity and the data size for two selected benchmarks.**

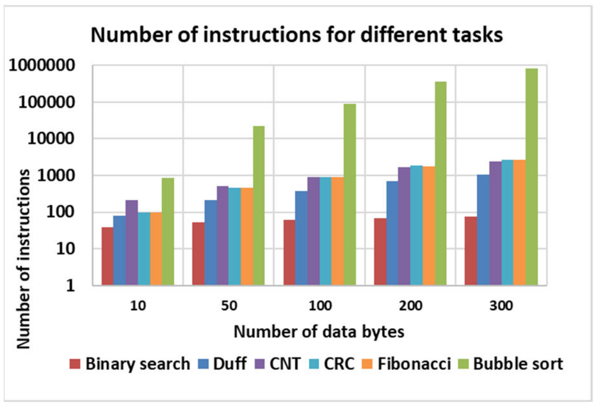


FIGURE 22. Total number of instructions vs. data size for selected Mälardalen benchmarks (original symbols [35]).

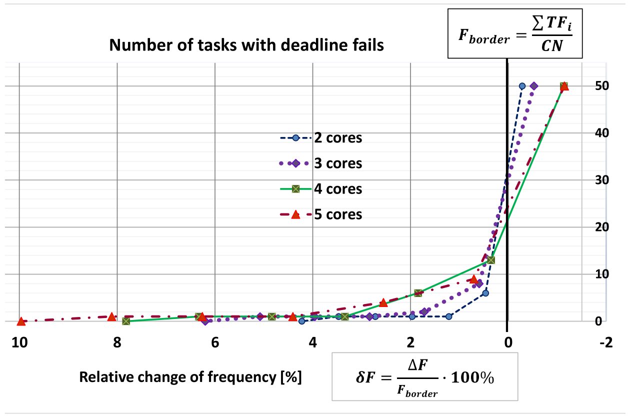


FIGURE 23. Relationship between the number of failed deadlines and the relative change of the system frequency.

where:

N	– a number of tasks,
TF_i	– task frequency of i-th task
and CN	– number of processing cores.

While the relative change of the system frequency is defined as the difference between the actual frequency of the system, F_{sys}, and the theoretical border frequency, F_{border}, and can be expressed in a percentage, namely:

$$\delta F = \frac{F_{sys} - F_{border}}{F_{border}} \cdot 100\% \tag{11}$$

Fig. 23 shows that in order to obtain the reliable real-system that guarantees 100% predictability (all deadlines are met), we need to introduce a safety margin, i.e., the system frequency should exceed the theoretical level stemming from the sum of tasks' frequencies. Our experiments showed that this margin may vary between 4 % and 12 % depending on the system configuration and the number of threads per processing core.

IX. SUMMARY

A. GENERAL REMARKS

The paper addresses some issues concerning energy-efficient tasks scheduling in time-predictable embedded systems. We presented a proposal of an energy-efficient design methodology that can be an important part of real-time system's design flow. We focused on low-level, hardware design and safety systems with a defined set of tasks and procedures, i.e., static scheduling. The main goal of the presented approach is to guarantee 100% predictability; however, we showed that it could potentially give further energy savings. The entire methodology was incorporated into the process of tasks mapping to the original real-time platform and based on the threads interleaving [24].

As it was presented, the methodology is based on three phases (steps). The first phase is the most tedious and time-consuming, but it gives information about the tasks' behavior in the presented system architecture and delivers their important parameters. This information is used during the second stage of the methodology, i.e., the selection of the structure of the system (number of processing cores) and mapping (scheduling) tasks to the system resources. For this purpose, we proposed three versions of scheduling algorithms that can be applied to different types of tasks (independent and cooperating) and pipelines (general-purpose – standard or dedicated). Based on these algorithms, we proposed three different methodologies: the simple, SFERA and MAXPRO. After the system configuration is the last, third phase – the system design. The presented results obtained for a set of Mälardalen WCET benchmarks [35] validate our approach.

B. DESIGN AND TESTING ENVIRONMENT

In our experiments, we used the professional version of the Xilinx Vivado ISE design tool. The entire system is described as an original IP in Verilog hardware description language (HDL); tasks were compiled into machine code (Fig. 15) and stored in local memories (defined as Xilinx macros). We used Xilinx Vivado and Mentor Graphics simulators for timing and functional simulation. Real-time systems were synthesized and implemented in a Xilinx Virtex7 FPGA device on the platform VC707 (Fig. 24). The probabilistic power analyses were performed with a Vivado ISE analyzer.

We used the artificial intelligence language, LPA PROLOG [39], for the scheduling algorithm's implementation. PROLOG, unlike many other such tools, is a declarative language of programming in logic based on the backtracking search and linear resolution. For the tested examples we obtained results after less than 10 second. However, this parameter is not so important taking into account that the scheduling process is static (off-line). Finally, the tasks' compiler was implemented in the C++ language.

C. COMPARISON TO OTHER APPROACHES

Table 4 gathers some selected parameters comparing our methodology to other approaches. Though it is very difficult to compare different platforms and processing tasks,

TABLE 4. Comparison with other approaches in the field of multi-task scheduling.

Approach	Type of the algorithm	System architecture	Platform	Complexity	Reconfig.	Power constraints	Predict.	Other properties
Ours	heuristic	multi-core	FPGA	$O(n \cdot \ln(n)) + O(n)$	yes	yes	yes	static scheduling threads interleaving
[25]	mathematical	cloud computing/ resource management	Cloud	$O(n \cdot \ln(n)) + O(n)$	yes	-	-	dynamic energy efficient resource allocation
[27]	heuristic	multi-processor	Java simulated CPUs	$O(n^2 \times CN)$	yes	yes	yes	global DVFS-enabled energy-efficient scheduling
[29]	heuristic-evolutionary	fixed	Not reported	O(1)	no	no	yes	suboptimal solution genetic algorithms
[32]	mathematical	multi-core/ muti-processor	PC multi-core	not reported	yes	yes	no	task stealing
[37]	mathematical	fixed big.LITTLE	Smartphone CPU	not reported	no	-	yes	machine learning techniques
[38]	mathematical and heuristic	multi-core	PC multi-core	$O(n \cdot \ln(n)) + O(n)$	yes	yes	yes	dynamic scheduling non-preemptive

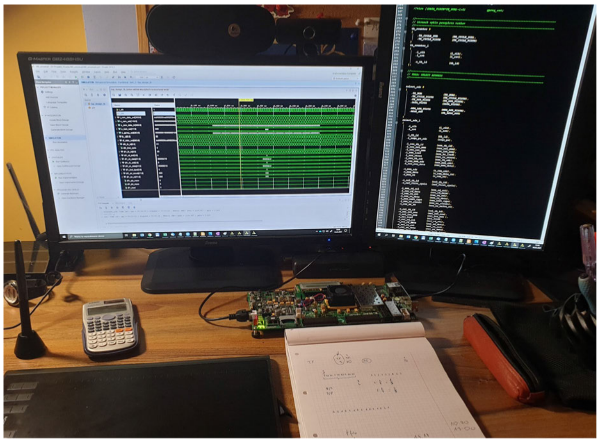


FIGURE 24. Our research stand.

we determined common properties of selected methodologies that were reported in the literature. Most of the methodologies were based on heuristics, but there are also purely algorithmic (mathematical) approaches. Many techniques are dedicated to reconfigurable platforms (systems) consisting of many processors and/or many cores. In regards to computational complexity, our approach, as well as those reported in [25] and [38], has a complexity of $O(n \cdot \ln(n)) + O(n)$. The first, dominating component is caused by the necessity of sorting tasks. The approach reported in [29] and based on evolutionary computing and genetic algorithms, has the lowest value but it gives sub-optimal solutions. All techniques gave reasonable and noticeable power savings generally related to a single-core system implementation. Our methodology was also compared to a single core solution. Although our system cannot, in its current state, process many tasks, it is fully predictable and based the static scheduling, so the amount of time spent on off-line calculations is not as important.

Fig.25 shows a comparison of the maximum and minimum energy savings for the references presented in Table 4. Such a comparison seems to be most justified because of the differences in architectures, operation frequencies and abstraction levels of other algorithms. Our solution is comparable to the work of [27], [29], [38]. We used FPGA platform and our hardware-matched approach could adapt the structure to the tasks performed on it. Other solutions affected at most the clock frequency and/or supply voltage.

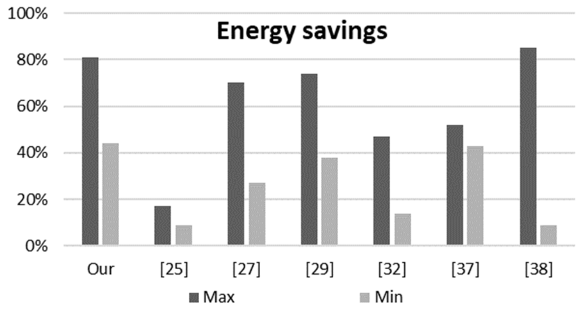


FIGURE 25. Comparison of percentage of the energy saving for different approaches.

In Fig. 26 and 27, we showed the percentage energy savings obtained with our solution and the approach presented in [38]. Fig. 26 shows the energy savings in a given core relative to a reference core that processes 40 tasks while in Fig. 27, the obtained energy savings are related to a 2-core reference architecture. Although both implementations are different, the comparisons concern similar applications. In the proposed approach, we focused on the analysis only the dynamic energy, which has a decisive impact on the possible energy savings. As shown on the diagram depicted in Fig. 20, there is a fairly large discrepancy between the energy predicted and the energy actually consumed by the system. However, such solutions indicate a large margin of safety in terms of time predictability of the system. The work presented in [38] gives more accurate prognoses, but the prognoses are optimistic and meeting deadlines for tasks is more risky.

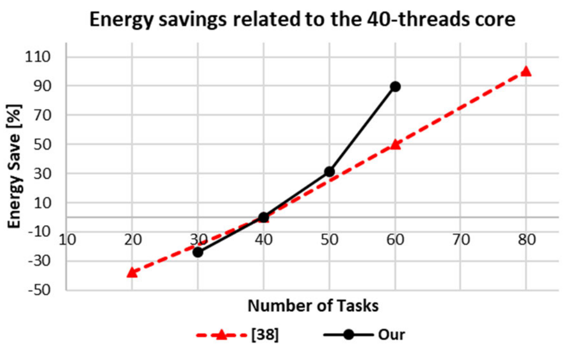


FIGURE 26. Energy savings related to the reference core.

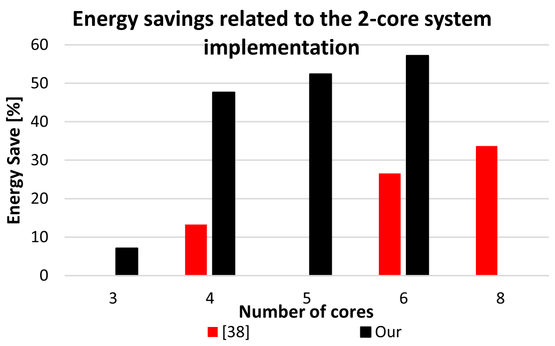


FIGURE 27. Comparison of percentage of the energy savings for different system architectures related to the 2-core structure.

D. CONCLUSION

The presented methods and algorithms are based on the system architecture described in [24]; however, we have extended the structure and modified the tasks interchange mechanism. Thus, the memory operations time, (M_dur) in a new version of our architecture, is independent of the number of employed cores. In this work, we focused on the problem of the reduction of the energy dissipated in the real-time system, but the main goal remained the same: we required a system that assures time predictability. Therefore, the speed of the system was reduced to 200 MHz. In the previous version of our system [24], we reported the maximal frequency up to 500 MHz. It was possible to achieve that under the assumption of PLL-based generator usage, available in Virtex 7, and the implementation of all tasks in small scratch-pad memories. That architecture allows us manual modification of final the mappings and connecting resources. The new, extended architecture can process large amounts of data, occupies more area and the net delay parameter combined with the memory access delay reduces the maximal working frequency. Although the device's throughput is important, the predictability of the entire system is the most crucial factor. Additionally, we have control over the energy consumed by the system and, if necessary, we can reduce it by 80% (in comparison to the one core implementation).

We developed our own design methodology of real-time systems and paid attention to the design automation process. Instead of slacks and deadlines' margin analyses, we defined the processing frequency of a task (*TF*) parameter that combines a task's complexity (i.e., a total number of various types of instructions) and its deadline. This parameter allows the simple and efficient mapping of tasks and guarantees the time predictability of the system. To make the results possible, we have to assume a margin for the system frequency, F_{sys}, the F_{margin} parameter in the (5) was added. Based on the results of our experiences, this parameter should be taken from the range of 4–12%.

E. AREAS FOR FURTHER RESEARCH

The design methodology based on the utilization of the proposed algorithms is being constantly developed and tested. We plan to verify it in the design of safety systems. Other possibilities of further research include investigations on increasing the overall system performance by reducing memory access delay impact on the frequency and reducing the synchronization, i.e., the introduction of GALS (globally asynchronous locally synchronous) into data interchange protocols. The GALS bus will allow the selection of a different clock frequency for each core in the system. We would also like to test various mechanisms of threads identifier generation in a TIC controller (Fig. 2). Finally, in the near future, we also plan to move the implementation from the prototype (FPGA) into an ASIC developed in the SYNOPSYS environment.

REFERENCES

[1] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *J. Syst. Archit.*, vol. 61, no. 9, pp. 449–471, Oct. 2015.

[2] Y. Liu, G. Xie, Y. Tang, and R. Li, "Improving real-time performance under reliability requirement assurance in automotive electronic systems," *IEEE Access*, vol. 7, pp. 140875–140888, 2019.

[3] S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, Jun. 2007, pp. 264–265.

[4] 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.

[5] L. Thiele and R. Wilhelm, "Design for timing predictability," *Real-Time Syst.*, vol. 28, nos. 2–3, pp. 157–177, Nov. 2004.

[6] A. Pułka and A. Milik, "Dynamic rescheduling of tasks in time predictable embedded systems," *IFAC Proc. Volumes*, vol. 45, no. 7, pp. 305–310, 2012.

[7] Ł. Golly, A. Milik, and A. Pulka, "High level model of time predictable multitask control unit," *IFAC-PapersOnLine*, vol. 48, no. 4, pp. 348–353, 2015.

[8] L. Golly, "Concurrent modeling of time predictable multitask electronic systems in SystemC language," Ph.D. dissertation, Dept. Autom. Control, Electron. Comput. Sci., Silesian Univ. Technol., Gliwice, Poland, 2016.

[9] W. Stallings, "Reduced instruction set computer architecture," *Proc. IEEE*, vol. 76, no. 1, pp. 38–55, Jan. 1988.

[10] E. Lee and D. Messerschmitt, "Pipeline interleaved programmable DSP's: Architecture," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 35, no. 9, pp. 1320–1333, Sep. 1987.

[11] 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., Architectures Synth. Embedded Syst. (CASES)*, 2008, p. 137.

[12] 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.

[13] D. Broman, M. Zimmer, Y. Kim, H. Kim, J. Cai, A. Shrivastava, S. A. Edwards, and E. A. Lee, "Precision timed infrastructure: Design challenges," in *Proc. Electron. Syst. Level Synth. Conf. (ESLsyn)*, May/Jun. 2013, pp. 1–6.

[14] G. C. Buttazzo, *Hard Real-Time Computing Systems*. New York, NY, USA: Springer, 2011.

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

[16] L. M. AlBarakat, P. V. Gratz, and D. A. Jimenez, "MTB-fetch: Multithreading aware hardware prefetching for chip multiprocessors," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 175–178, Jul. 2018.

[17] A. Alhammad and R. Pellizzoni, "Time-predictable execution of multithreaded applications on multicore systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[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)*, 2012, pp. 175–184.

[19] M. Schoeberl, "A Java processor architecture for embedded real-time systems," *J. Syst. Archit.*, vol. 54, nos. 1–2, pp. 265–286, Jan. 2008.

[20] B. Akesson and K. Goossens, *Memory Controllers for Real-Time Embedded Systems*. New York, NY, USA: Springer, 2012.

[21] 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. 9th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Oct. 2011, p. 99.

[22] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, and W. C. Probst, "Towards a time-predictable dual-issue microprocessor: The Patmos approach," in *Proc. 1st Workshop Bringing Theory Pract.: Predictability Perform. Embedded Syst. (PPES)*, 2011, pp. 11–21.

[23] 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.

[24] 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.

[25] A. U. Rehman, Z. Ahmad, A. I. Jehangiri, M. A. Ala'Anzy, M. Othman, A. I. Umar, and J. Ahmad, "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.

[26] 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.

[27] G. Xie, G. Zeng, X. Xiao, R. Li, and K. Li, "Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3426–3442, Dec. 2017, 10.1109/TPDS.2017.2730876

[28] 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.

[29] 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.

[30] 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)*, Oct. 2018, pp. 1951–1957.

[31] R. Pathan, P. Voudouris, and P. Stenstrom, "Scheduling parallel real-time recurrent tasks on multicore platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 915–928, Apr. 2018.

[32] S. I. Kim and J.-K. Kim, "A method to construct task scheduling algorithms for heterogeneous multi-core systems," *IEEE Access*, vol. 7, pp. 142640–142651, 2019.

[33] T. H. Cormen and T. H. Cormen Eds., *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.

[34] *VC707 Evaluation Board for the Virtex-7 FPGA*. Accessed: Nov. 2019. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf

[35] *Mälardalen: WCET Benchmark Programs*. Accessed: Mar. 2021. [Online]. Available: http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[36] 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.

[37] 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.

[38] *Logic Programming Associates*. Accessed: Mar. 2021. [Online]. Available: https://www.lpa.co.uk/

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 1989, 1997, and 2013, respectively.

He is currently a University Professor of the Silesian University of Technology 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, 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 a member of the Electronics Commission, Polish Academy of Sciences.