A New Approach for Optimal Implementation of Multi-core Reconfigurable Real-time Systems

Wafa Lakhdhar[1,5], Rania Mzid[2,3], Mohamed Khalgui[1,4] and Georg Frey[5]
[1]*LISI Lab INSAT, University of Carthage, INSAT Centre Urbain Nord BP 676, Tunis, Tunisia*
[2]*ISI, University Tunis-El Manar, 2 Rue Abourraihan Al Bayrouni, Ariana, Tunisia*
[3]*CES Lab ENIS, University of Sfax, B.P:w.3, Sfax, Tunisia*
[4]*SystemsControl Lab, Xidian University, August Bebel Str 70, Halle, China*
[5]*Automation and Energy Systems, Saarland University, Saarbrucken 66123, Germany*

Keywords: Real-time and Reconfiguration, Multi-core, POSIX, Task and Function, MILP and Optimization.

Abstract: This paper deals with a multi-core reconfigurable real-time system specified with a set of implementations, each of which is raised under a predefined condition and executes multiple functions which are in turns executed by threads. The implementation as threads generates a complex system code. This is due to the huge number of threads and the redundancy between the different implementations which may lead to an increase in the energy consumption. Thus we aim in this paper to optimize the system code by avoiding the redundancy between implementations and reducing the number of threads while meeting all related real-time constraints. The proposed approach adopts mixed integer linear programming (MILP) techniques in the exploration phase in order to provide a feasible task model. An optimal reconfigurable POSIX-based code of the system is manually generated as an output of this technique. An application to a case study and performance evaluation confirm and validate the expected results.

1 INTRODUCTION

A real-time system has to respond to externally generated input stimuli within a finite and specified delay (Burns and Wellings, 2009). Such system may have many implementation scenarios, the transition from an implementation to another is called reconfiguration. Reconfiguration refers to the architectural or behavioral modifications of a software system during its execution (Polakovic et al., 2007) to meet user requirements. Currently, some real-time systems such as automotive electronics, avionics, telecommunications, and consumer electronics become more complex and need more computational power. Thus, the necessity for multi-core architecture is a common answer. The multi-core technology allows increasing the processor clock frequency, which is limited by available instruction-level parallelism and leads to challenging power requirements (Geer, 2005). This paper deals with multi-core reconfigurable real-time systems.

One challenge during the development of multi-core reconfigurable real-time systems is to ensure an appropriate partitioning and scheduling of the applicative functions across the target platform such that the timing constraints are met. In that context, different scheduling policies have been proposed in the literature (Khan and Hafiz, 2014) (Lakshmanan, 2011). Existing multi-core scheduling policies can be classified into three different classes: the partitioned, the global and semi-partitioned approach. The partitioned scheduling allows to choose a core for all tasks and then runs a local scheduler on each core (i.e., off-line scheduling). However, the global scheduling allows to choose a task and to assign it to one of the cores (i.e., on-line scheduling). As opposed to the partitioned approach, different instances of the same task can execute on different cores. The semi-partitioned approach presents an improvement of the partitioning scheduling allowing the controlled tasks migration. It is a hybrid between partitioned and global scheduling (Lakshmanan, 2011). In this paper, we adopt a partitioned approach because it is easier to implement and to analyze. Also, it allows no task migration, thus has low runtime overheads (Funk and Baruah, 2005). For the three multi-core scheduling approaches, several scheduling algorithms have been proposed such as Rate Monotonic RM (Liu and Layland, 1973) which

Lakhdhar, W., Mzid, R., Khalgui, M. and Frey, G.
A New Approach for Optimal Implementation of Multi-core Reconfigurable Real-time Systems.
DOI: 10.5220/0006698100890098
In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, pages 89-98
ISBN: 978-989-758-300-1

will be adopted for tasks scheduling in this paper.

The multi-core introduces additional challenges that are still difficult to deal with in real world industrial domains. Indeed, the huge number of tasks exhibits a high complexity by increasing the energy consumption, invoking many redundancies between the different implementations, and producing a complex system code. Thus, we propose a multi-objective optimization approach that minimizes both the energy consumption and the number of tasks in order to reduce the redundancy between the implementation sets. Such optimization may reduce the time overhead and the complexity of the generated code. At the specification level, the developer defines the function sets, the condition sets, and the core sets. At the design level, this approach (i) generates the implementation sets, (ii) affects the functions to the tasks sets which are in turns assigned to the core sets, and (iii) generates a feasible and optimized task model by using the mixed integer linear programming (MILP) formulation. Finally, at the implementation level, each task is transformed to a thread to execute the applicative functions.

The originality of the proposed approach follows from the fact that (i) it deals with the multi-core (i.e., partitioning), reconfiguration, and real-time problems simultaneously, and (ii) it proposes a multi-objective optimization to minimize both the energy consumption and the number of tasks.

The rest of the paper is organized as follows. Section 2 presents the state of the art on multi-core reconfigurable real-time systems. In Section 3, we give the system formalization. We present the proposed approach in Section 4. In Section 5 we present the Global Positioning System (GPS) case study which is considered to evaluate the proposed approach. Finally, we summarize our work and discuss future directions in Section 6.

2 RELATED WORKS

In the area of real-time multi-core systems, some existing works focus on the synthesis problem (Wang et al., 2016), (Yehia et al., 2011), (Geismann et al., 2017). In (Wang et al., 2016), the authors propose a formalization of periodic tasks adapted to engine control applications in multi-core automotive systems. The work reported in (Yehia et al., 2011) presents a system level synthesis approach for multi-core system architectures from Task Precedence Graphs (TPG) models. In (Geismann et al., 2017), the authors present an approach for a semi-automatic synthesis of models into a deterministic scheduling that respects real-time requirements for multi-core systems. Despite the importance of above related synthesis works, none of these solutions considers the reconfiguration property. In contrast, this paper focuses on how the feasible code may be generated for multi-core reconfigurable real-time system.

Another related area is the safe deployment to multi-core real-time systems, where approaches focus on the mapping of tasks on multi-core platforms (Monot et al., 2012), (Saidi et al., 2015), (Yi et al., 2009), (Vulgarakis et al., 2014) (Faragardi et al., 2013). In (Monot et al., 2012), the authors develop a heuristic algorithm for the function mapping on a multi-core architecture. In this work, the functions are grouped and distributed across cores, then they are mapped to tasks. Similarly to (Saidi et al., 2015), which proposes a heuristic algorithm to create a task set according to the mapping of runnable entities on the cores. In (Yi et al., 2009) a linear program is developed for task partitioning, mapping, and scheduling on embedded multi-core systems. Some other related works propose an end-to-end approach to generate a full real-time system. In (Vulgarakis et al., 2014), the authors present a process for the automatic deployment of control applications on multi-core platforms and generate Java code. The work reported in (Faragardi et al., 2013) addresses the mapping problem of hard real-time systems composed of periodic AUTOSAR runnables in the context of the multi-core.

The proposed approach differs from the cited works in several points. First of all, it considers reconfigurable systems with real-time properties and multi-core architecture. Secondly, we address the synthesis, the partitioning, the scheduling problem, and the optimization simultaneously. Finally, the majority of them do not propose general solutions. They target a well defined system.

3 SYSTEM FORMALIZATION

In this section, we present a formal description of a multi-core reconfigurable processor system. We present in addition real-time prerequisites required to introduce the paper's contribution.

3.1 System and Architecture Modeling

It is assumed in this work that a reconfigurable real-time system Sys is defined as a set of m implementations: $Sys = \{imp_1, imp_2 \dots imp_m\}$. We denote by $Sys(t)$ the implementation defining the system at particular time t (i.e., $Sys(t) = imp_i$). An

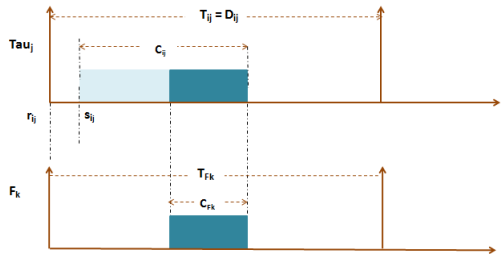

Figure 1: Task and function models.

implementation imp_i is composed of N_i tasks (i.e., $imp_i = \{\tau_1, \tau_2, \tau_3 \ldots \tau_{N_i}\}$). Each task τ_j in the implementation imp_i is characterized by (top of Fig 1): $(r_{ij}, s_{ij}, T_{ij}, C_{ij}, D_{ij}, P_{ij}, E_{ij})$ where r_{ij} is its release time, we assume that $r_{ij} = 0$, its start time s_{ij} which denotes the effective starting time of a task τ_j, its activation period T_{ij}, C_{ij} denotes the capacity or worst case execution time, its deadline D_{ij} which is assumed to be equal to its period in this work $D_{ij} = T_{ij}$, the priority P_{ij} that is inversely proportional to the period T_{ij} as we use the RM policy, E_{ij} presents the energy consumption of task τ_j which is computed as the sum of the energy consumed by the functions implemented by τ_j.

The task τ_j may implement a single or several functions which must have the same period $\tau_j = \{F_1, F_2, F_3, \ldots F_{p_j}\}$. Each function F_k is characterized by static real-time parameters (bottom of Figure 1) $(T_{F_k}, C_{F_k}, E_{F_k})$ where T_{F_k} is the activation period of the function F_k, C_{F_k} is an estimation of its worst case execution time (WCET) and E_{F_k} is the energy consumed by the function F_k during its execution. Note that these parameters are considered as inputs for the proposed approach and must be specified by the user.

The system consists of one processor, containing a set of M identical cores $\{\zeta_1, \zeta_2 \ldots \zeta_M\}$ that share common memory. Each core runs a set of tasks. We assume that the tasks are independent and periodic.

3.2 Energy Model

Each function F_k is described by two parameters: (i) the function's frequency f_{F_k}, and (ii) the function's voltage V_{F_k}. The energy consumption for the execution of function F_k that we denote by E_{F_k} is computed as $E_{F_k} = f_{F_k} V_{F_k}^2 C_{F_k}$. The energy consumption E_{ij} of task τ_j is then equal to the sum of the energy consumed by the implemented functions $E_{ij} = \sum_{k \in \{1 \ldots p_j\}} E_{F_k} = \sum_{k \in \{1 \ldots p_j\}} f_{F_k} V_{F_k}^2 C_{F_k} = f_{ij} \quad V_{ij}^2 C_{ij}$ where (f_{ij}, V_{ij}) are two parameters characterizing task τ_j in implementation imp_i. We assume that $f_{ij} = \sum_{k \in \{1 \ldots p_j\}} f_{F_k}$ and $V_{ij} = \sum_{k \in \{1 \ldots p_j\}} V_{F_k}$. Thus the total energy consumption (Lei et al., 2016) of implementation imp_i is given by expression 1

$$E_i = \sum_{j \in \{0, N_i\}} E_{ij} = \sum_{j \in \{0, N_i\}} f_{ij} V_{ij}^2 C_{ij} \qquad (1)$$

In expression 2, we denote by f_n and V_n the normalized frequency and voltage of the system. We denote by η_j the reduction factor of voltage when τ_j is executed, $V_{ij} = \frac{V_n}{\eta_j}$ and $f_{ij} = \frac{f_n}{\eta_j}$. In addition, we denote by C_n the computation time at the normalized processor frequency i.e., $C_{ij} = C_n \eta_j$. Thus, the total energy consumption of the implementation imp_i according to (Chniter et al., 2014) is given by

$$E_i = \sum_{j \in \{0, N_i\}} \frac{f_n * V_n^2 * C_n}{\eta_j^2} = K \sum_{j \in \{0, N_i\}} \frac{C_n}{\eta_j^2} \qquad (2)$$

where $K = V_n^2 \, f_n$.

3.3 Processor Utilization Factor

Let U_i be the processor utilization factor of the implementation imp_i is defined by: $U_i = \sum_{j=1}^{N_i} \frac{C_{ij}}{T_{ij}}$ (Klein et al., 1993). As we perform Rate-Monotonic (RM) assignment and preemptive scheduling, the real-time system is feasible when the test given by expression 3 is verified.

$$\forall i \in \{1..m\}, U_i \leq N_i (2^{\frac{1}{N_i}} - 1). \qquad (3)$$

3.4 Reconfiguration Time

We define in addition the reconfiguration time T_{reconf} (Lakhdhar et al., 2016) as the sum of the time required to add/remove tasks (i.e., time spent by the system to jump from one implementation to another) and the time required for task's migration. The time for task's migration refers to the period of time required for a task to move from one core to another when the system load a new implementation. Thus, we define the reconfiguration time as follow:

$$T_{reconf} = (A + B) * T_{cost} + C * T_{migration} \qquad (4)$$

Where A is the number of the deleted tasks, B is the number of created tasks, T_{cost} is the spent time to delete/add a task, C is the number of migrated tasks and $T_{migration}$ is the time spent to migrate from a core to another. One objective of the present work is to reduce the reconfiguration time T_{reconf} of the multi-core reconfigurable real-time system with the aim to improve its reactivity.

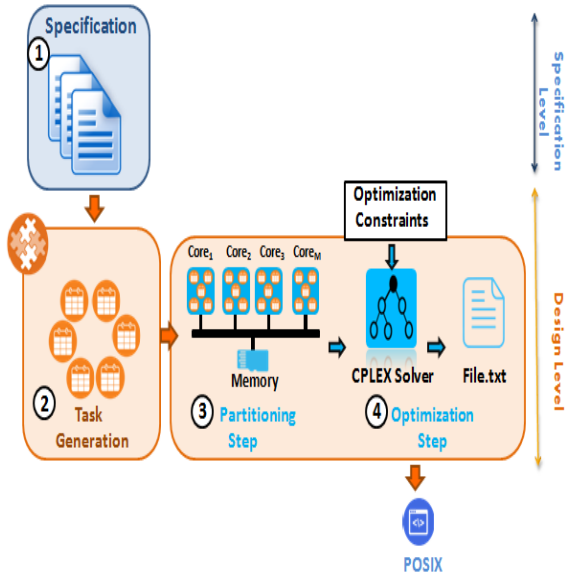

Figure 2: Process overview.

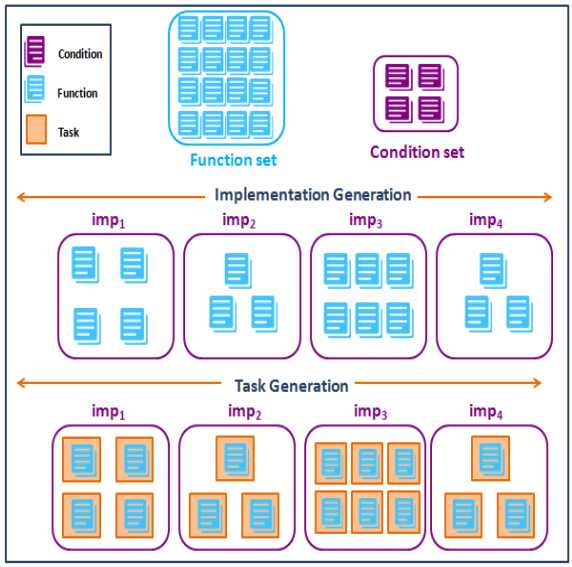

Figure 3: Example Of Initial Task Model.

4 PROPOSED APPROACH

In this section, we present the full working process used in this paper. As shown in Figure 2, the proposed process is composed of four main phases: (i) Specification, (ii) Task generation, (iii) Partitioning, and (iv) Optimization.

4.1 System Specification

In the specification level, the designer provides the specification model which defines (i) the reconfiguration conditions, (ii) the functions that must be executed under a considered condition. Each function is defined by a set of temporal parameters, and (iii) the set of cores. This model presents the input of the generation task step.

4.2 Task Generation

In this step, we aim to generate the initial task model from the specification model. As shown in Figure 3, this step is carried out in two sub-steps: (i) the generation of implementation sets from the condition sets and the assignment of functions to the appropriate implementation, and (ii) the generation of the task model from the function sets, such that each function is assigned to a task which will take the same parameters of the corresponding function (i.e., in this step, the task number is equal to the function number). Let us note that for the generation of this model, the real-time feasibility is not considered.

4.3 Task Partitioning

The partitioning of tasks into multiple cores must consider real-time feasibility. Under the hypotheses considered in this paper, the partitioning corresponds to an RM (Wang et al., 2014) scheduling problem related to periodic tasks. Each generated task must be affected to a specific core, then in each core, we run a local scheduler. This partitioning is characterized by (i) no migration at run time (i.e., in a given implementation a task must always run on a given core), (ii) the possibility of applying end-to-end worst case response time analysis, and (iii) off-line core assignment (Tindell and Clark, 1994). The basic principles of an RM-based partitioning heuristic are:

1. Order tasks according to RM policy,
2. Do task assignment according to their order,
3. For each task, look for an available core, by applying one of the following policies (Singhoff, 2014),
   - Best Fit policy: for each task i, we start with core $j = 0$ and assign task i on the core on which the feasibility test is true and on which the processor utilization factor has the highest value
   - First Fit policy: for each task i, we start with core $j = 0$ and assign task i on the first core on which the feasibility test (expression 3) is true. In this paper, we use the First Fit policy.
4. Stop when all tasks are assigned.

Algorithm 1 that illustrates this step considers as inputs: the implementations, tasks and the cores set. It generates as output the partitioning task model.

Algorithm 1: Partitioning Task Model.

```
Input:
- Imp: Implementation set
- Task: Task set
- Core: Core set
Output:
- PartTask: Partitioning Task Model
Notations:
- m: implementation number
- N_i: Number of tasks in the implementation i.
- M: Core number
for i ← 0 to m do
    /*** Task Index ***/
    j ← 0
    /*** Core Index ***/
    k ← 0
    for j ← 0 to N_i do
        for k ← 0 to M do
            if Faisabilty is true then
                AssignTask[j]toCore[k]
                PartTask[j][k] = Core[k]
            else
                k++
return PartTask
```

4.4 Optimization Step

In order to ensure a reliable implementation of the multi-core reconfigurable real-time system from the initial task model and the partitioning task model taking into consideration the different constraints (Figure 2) (Real-time, no migration, energy), we propose in this section a MILP model which consists of a linear objective function to be optimized and a set of linear inequalities (constraints).

4.4.1 Variable Definition

Let (i) $Merge_{jq}$ be a boolean variable used to mention whether two tasks τ_j and τ_q are merged such that $Merge_{jq}$ is equal to 1 if task τ_j and task τ_q are merged, the merge corresponds to the situation in which τ_j absorbs τ_q, to be deleted from the model, (ii) x_{js} be a boolean variable used to mention if τ_j is executed in core s. Thus if the value of x_{js} is equal to 1, then the corresponding task τ_j is running in core s, (iii) y_{ij} be a boolean variable used to mention if τ_j is in the implementation i, (iv) T be the set of period of N task, (v) $C_{new_{ij}}$ be the new WCET of the task τ_j in imp_i, (vi) $T_{new_{ij}}$ be the new period of the task τ_j in imp_i, (vii) μ_{jq} be a binary variable where μ_{jq} = 1 when τ_q is executed before τ_j.

4.4.2 Objective Function

$$maximize \sum_{j,q\in\{0..N\}} Merge_{jq} - \sum_{i\in\{1..m\}} \sum_{j\in\{0,N\}} E_{ij} \quad (5)$$

The expression 5 defines the objective function. It aims to maximize the number of merges while minimizing the total energy consumption.

4.4.3 Merging Situation Constraints

The constraints 6 and 7 introduce the merging condition such as the two tasks $\tau_j \in \zeta_s$ and $\tau_q \in \zeta_s$ will be merged if they have the same period.
$\forall j,q \in \{1..N\}\ s \in \{1..M\}$

$$if(T_j * x_{js} - T_q * x_{qs} = 0)\ then\ Merge_{jq} = 1; \quad (6)$$

$$if(T_j * x_{js} - T_q * x_{qs} <> 0) \quad then\ Merge_{jq} = 0; \quad (7)$$

The constraint in 8 is used to avoid a non-meaningful situations which corresponds to the merge of a task already merged i.e., $\forall j,q,r \in \{1..N\}$,

$$Merge_{jq} \leq 1,\ ,q,r \neq j, Merge_{jq} + Merge_{rj} \leq 1 \quad (8)$$

4.4.4 Real-time Constraints

In each implementation, every pair of tasks $\forall j,q \in \{1\ldots N\}$ τ_j and τ_q, we should respect the constraints 9 and 10 to ensure that only one task will be executed at a single time. $\forall i \in \{1\ldots m\}$

$$s_{ij} - s_{iq} >= C_{new_{iq}} - M * \mu_{jq} \quad (9)$$

$$s_{iq} - s_{ij} >= C_{new_{ij}} - M * (1 - \mu_{jq}) \quad (10)$$

where $C_{new_{ij}}$ and $C_{new_{iq}}$ are the WCET of the tasks τ_j and τ_q. Constraints 4.4.4, 12 , and 13 give the computation formula of $C_{new_{ij}}$.
If a task $\tau_j \in imp_i$ does not be merged with any task in all implementations, the WCET of τ_i does not change. $\forall j \in \{1..N\}\ \forall i \in \{1..m\}$

$$if(\sum_{q\in 1..N} Merge_{qj} + \sum_{r\in 1..N} Merge_{jr} = 0)\ then\ C_{new_{ij}} = C_{ij}; \quad (11)$$

Else if a task τ_j is merged with another task in the same implementation or not, the WCET of τ_j is calculated in two cases:(i) the task τ_j and τ_q are in the same implementation, so the resulting WCET is the sum of C_{ij} and C_{iq}, and (ii) the task τ_j and τ_q are not in the same implementation, so the resulting task has two different WCETs in the two implementations. Constraints 12 , and 13 allow to compute the resulting WCET in the two cases. $\forall j,q \in \{1..N\}\ i,l \in \{1..m\}$

$$if(Merge_{jq} + y_{ij} + y_{lp} = 3)\ then\ C_{new_{ij}} = C_{ij} + C_{lq} \quad and\ C_{new_{lq}} = 0; \quad (12)$$

$$if(Merge_{jq}+y_{ij}+y_{lp}=2)\ then\ C_{new_{ij}} = C_{ij}$$
$$and\ C_{new_{lq}} = C_{lq}; \quad (13)$$

Constraint 14 ensures the feasibility of the system, $\forall i \in \{1..m\}$:

$$U_i \leq N(2^{\frac{1}{N}}-1) \quad (14)$$

where U_i is given by

$$U_i = \sum_{j=1}^{N} \frac{C_{new_{ij}}}{T_{new_{ij}}} \quad (15)$$

Where $T_{new_{ij}}$ is computed as follow: $\forall j,q \in \{1..N\}\ i,l \in \{1..m\}$
$if(Merge_{jq}+y_{ij}+y_{lp}=3)\ then T_{new_{ij}} = T_{ij}$

$$and\ T_{new_{lq}} = 0; \quad (16)$$
$$if(Merge_{jq}+y_{ij}+y_{lp}=2)\ then\ T_{new_{ij}} = T_{ij}$$
$$and\ T_{new_{lq}} = T_{lq}; \quad (17)$$

The start time should respect Constraint 18 i.e.,

$$\forall j \in \{1..N\}, \forall i \in \{1..m\}, s_{ij} >= r_{ij} \quad (18)$$

4.4.5 Energy Constraints

$$E_{ij} = K \sum_{j \in \{1,N\}} \frac{C_n}{\eta_j^2} \quad (19)$$

The energy consumption's equation is fractional, thus we simplify this program in order to be interpretable by using the CPLEX solver that maximizes the reduction factor η_j which is inversely proportional to the energy consumption as defined in Section 3. The objective function becomes

$$Maximise \sum_{j,q \in \{1,N\}} Merge_{jq} + \sum_{i \in \{1,m\}} \sum_{j,q \in \{0,N\}} x \quad (20)$$

To ensure the no simultaneous execution of tasks the constraints 9 and 10 become respectively 21 and 22: $\forall j,q \in \{1 \ldots N\}\ \tau_j$ and $\tau_q\ \forall\ i\ \in\ \{1 \ldots m\}$

$$s_{ij} - s_{iq} >= C_n * \eta_q - M * \mu_{jq} \quad (21)$$

$$s_{iq} - s_{ij} >= C_n * \eta_j - M * (1-\mu_{jq}) \quad (22)$$

$\forall i \in \{1..m\},\ j\ \in\ \{1..N\}$

To ensure that the start time is always greater than the release time, Constraint 23 is considered:

$$s_{ij} >= r_{ij} \quad (23)$$

We limit the value of x by Constraint 24

$$x \leq \eta_j \quad (24)$$

The proposed approach allows to manually generate the code from the optimized task model. The optimized task model has properties providing information on (i) the core set, (ii) the implementation set, (iii) the task set, (iv) the function set, (v) the task assignment, and (vi) the scheduling order. For each task in the optimized task model, we implement a POSIX thread by using the POSIX *pthread*. In the POSIX code, the assignment of the tasks to the core is giving by *stick_this_thread_system*, which allow moving from implementation to another, following well-defined conditions (i.e., user requirements).

5 CASE STUDY

In this section, we illustrate the proposed approach through a classical case study: a Global Positioning System (GPS) (Lakhdhar et al., 2016).

5.1 Specification Level

The GPS is used to define the position of an object on a plan or a map using the information provided via radio signals by the associated satellites. In the GPS, the satellite sends to the terminal an encrypted signal containing various information relevant to the location and timing. The terminal collects and converts radio signals received into information about the position, speed and time (Lakhdhar et al., 2016). In order to illustrate the proposed approach, we have enriched and extended this case study by introducing two modes: (i) default mode which consists of a default use of GPS, and (ii) secure mode which represents a restricted use of GPS with safety requirements.

The software architecture of the studied application is composed of seven functions in default mode and of eight functions in the secure mode such that every function is characterized by a period, a WCET and an energy consumption (Table 1). It is mapped to a preemptive execution platform composed of one processor which contains two cores C_1 and C_2. A tabular description of the specification model is given in Table 1.

Table 1 depicts two execution modes *Default Mode* and *Secure Mode*. Each mode is characterized by a set of functions defined by a set of timing parameters.

5.2 Initial Task Model

The second step consists in generating the implementations and their tasks. Proceeding from the specification model, for each condition we generate an im-

Table 1: Specification Model.

Execution mode	Condition	Function Name	Period ms	WCET ms	Energy mW
Default	Sec=Desabled	F_1: ControlBase	100	20	1687.5
		F_2: GpsSatellite	200	20	30
		F_3: Position	300	20	270
		F_4: Receiver	300	50	2031.25
		F_5: Decoder	400	40	5760
		F_6: TreatmentUnit	400	50	15000
		F_7: Encoder	500	30	15360
Secure	Sec=Enabled	F_1: ControlBase	100	20	1687.5
		F_2: GpsSatellite	200	20	30
		F_3': Position Secure	300	20	270
		F_4: Receiver	300	50	2031.25
		F_5: Decoder	400	40	5760
		F_6: TreatmentUnit	400	50	15000
		F_7: Encoder	400	50	15360
		F_8:AccessController	400	50	10800

plementation so we have two implementations. Then, we assign each function to a task. The resulting task model is given in Table 2 that shows two implementations which are composed of fifteen tasks. Each task is characterized by the same real-time parameters of the executed function.

Table 2: Initial Task Model.

Implementation	Task	Period ms	WCET ms	Energy mW
Default	τ_1	100	20	1687.5
	τ_2	200	20	30
	τ_3	300	20	270
	τ_4	300	50	2031.25
	τ_5	400	40	5760
	τ_6	400	50	15000
	τ_7	500	30	15360
Secure	τ_8	100	20	1687.5
	τ_9	200	20	30
	τ_{10}	300	20	270
	τ_{11}	300	50	2031.25
	τ_{12}	400	40	5760
	τ_{13}	400	50	15000
	τ_{14}	400	50	15360
	τ_{15}	400	50	10800

5.3 Partitioning Task Model

The next step consists in distributing the task model into a specific multi-core architecture. The targeted multi-core architecture contains 2 cores and a shared memory. In order to assign the tasks to the cores, we apply Algorithm 1. Table 3 presents the partitioning task model.

Table 3: Partitioning Task Model.

Implementation	Core	Task	Period/ Deadline ms	WCET ms
Default	ζ_1	τ_1	100	20
		τ_2	200	20
		τ_3	300	20
		τ_4	300	50
	ζ_2	τ_5	400	40
		τ_6	400	50
		τ_7	500	30
Secure	ζ_1	τ_8	100	20
		τ_9	200	20
		τ_{10}	300	20
		τ_{11}	300	50
	ζ_2	τ_{12}	400	40
		τ_{13}	400	50
		τ_{14}	500	30
		τ_{15}	600	50

For each implementation, we assign the tasks to the appropriate core based on the feasibility tests. The resulting partitioning task model represents the input of the optimization step.

5.4 Optimized Task Model

Once the initial solution is defined, we process the optimization step. This step involves the execution using CPLEX solver of the proposed linear program. We model the input of this step as two matrices: “tasks to implementation mapping matrix” (i.e., y) and “tasks to core assignment matrix” (i.e., x). These two matrices are defined as follows:

$$y = \begin{pmatrix} & imp_1 & imp_2 \\ \tau_1 & 1 & 0 \\ \tau_2 & 1 & 0 \\ \tau_3 & 1 & 0 \\ \tau_4 & 1 & 0 \\ \tau_5 & 1 & 0 \\ \tau_6 & 1 & 0 \\ \tau_7 & 1 & 0 \\ \tau_8 & 0 & 1 \\ \tau_9 & 0 & 1 \\ \tau_{10} & 0 & 1 \\ \tau_{11} & 0 & 1 \\ \tau_{12} & 0 & 1 \\ \tau_{13} & 0 & 1 \\ \tau_{14} & 0 & 1 \\ \tau_{15} & 0 & 1 \end{pmatrix} \begin{pmatrix} & \zeta_1 & \zeta_2 \\ \tau_1 & 1 & 0 \\ \tau_2 & 1 & 0 \\ \tau_3 & 1 & 0 \\ \tau_4 & 1 & 0 \\ \tau_5 & 0 & 1 \\ \tau_6 & 0 & 1 \\ \tau_7 & 0 & 1 \\ \tau_8 & 1 & 0 \\ \tau_9 & 1 & 0 \\ \tau_{10} & 1 & 0 \\ \tau_{11} & 1 & 0 \\ \tau_{12} & 0 & 1 \\ \tau_{13} & 0 & 1 \\ \tau_{14} & 0 & 1 \\ \tau_{15} & 0 & 1 \end{pmatrix} = x$$

The linear program generates the following Merge matrix:

$$Merge = \begin{pmatrix} 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix}$$

We note that the MILP program allows to merge the tasks: (i) τ_1 with τ_8, (ii) τ_2 with τ_9, (iii) τ_3 with τ_4, τ_{10} and τ_{11}, (vi) the task τ_5 with τ_6, τ_{12} and τ_{13}, and (v) τ_7 with τ_{14}. The input matrices become:

$$y = \begin{pmatrix} & imp_1 & imp_2 \\ \tau'_1 & 1 & 1 \\ \tau'_2 & 1 & 1 \\ \tau'_3 & 1 & 1 \\ \tau'_5 & 1 & 1 \\ \tau'_7 & 1 & 1 \\ \tau_{15} & 0 & 1 \end{pmatrix} \begin{pmatrix} & \zeta_1 & \zeta_2 \\ \tau'_1 & 1 & 0 \\ \tau'_2 & 1 & 0 \\ \tau'_3 & 1 & 0 \\ \tau'_5 & 0 & 1 \\ \tau'_7 & 0 & 1 \\ \tau_{15} & 0 & 1 \end{pmatrix} = x$$

The optimized task model is presented in Table 4.

Table 4: Optimized Task Model.

Implementation	Core	Task	Period ms	WCET ms	E_{old} mW	E_{new} mW
Default	ζ_1	τ'_1	100	20	1687.5	1181.25
		τ'_2	200	20	30	21
		τ'_3	300	20	2301.25	1610.87
	ζ_2	τ'_5	300	50	20760	14532
		τ'_7	400	40	15360	10752
Secure	ζ_1	τ'_1	100	20	1687.5	1181.25
		τ'_2	200	20	30	21
		τ'_3	300	20	2301	1610.87
	ζ_2	τ'_5	300	50	20760	14532
		τ'_7	400	40	15360	10752
		τ_{15}	400	50	10800	7560

We note that the task number is reduced as well as the energy consumption. Finally, we generate a POSIX code from the optimized task model describing the GPS. Listing 1 gives an excerpt of the GPS code. The role of the controller which corresponds to the main function in Listing 1, is to switch from one implementation to another under a considered condition.

```
#include <pthread.h>
void* F1 (void* arg);
void* F2 (void* arg);
...
/********* Controller POSIX code *********/
int main (void){
pthread_t tau_prime_1;
pthread_t tau_prime_2;
pthread_t tau_prime_3;
//  Default mode
pthread_create (&tau_prime_1, NULL, F1, (void *) 100);
      // Craation of tau_prime_1 thread
pthread_create (&tau_prime_2, NULL, F2, (void *) 200);
      // Craation of tau_prime_2 thread
pthread_create (&tau_prime_3, NULL, F3, (void *) 300);
      // Craation of tau_prime_3 thread
...
stick_this_thread_to_core(1); //assign thread 1 to core
      1 //
pthread_join (tau_prime_1, NULL);
stick_this_thread_to_core(1);
pthread_join (tau_prime_2, NULL);
......
stick_this_thread_to_core(2);
pthread_join (tau_prime_5, NULL);
...
// Secure Mode
if (cnd="Secure"){
pthread_create (&tau_prime_1, NULL, F8, (void *) 100);
      // Craation of tau_prime_1 thread
pthread_create (&tau_prime_2, NULL, F9, (void *) 200);
      // Craation of tau_prime_2 thread
  ...}
return 0;}
...
void* F1 (void* arg)
{///}
// core_id = 0, 1, ... n-1, where n is the system's
      number of cores
int stick_this_thread_to_core(int core_id) {
   int num_cores = sysconf(_SC_NPROCESSORS_ONLN);
   if (core_id < 0 || core_id >= num_cores)
      return EINVAL;
   cpu_set_t cpuset;
   CPU_ZERO(&cpuset);
   CPU_SET(core_id, &cpuset);
   pthread_t current_thread = pthread_self();
   return pthread_setaffinity_np(current_thread, sizeof(
      cpu_set_t), &cpuset);}
```

Listing 1: GPS POSIX code.

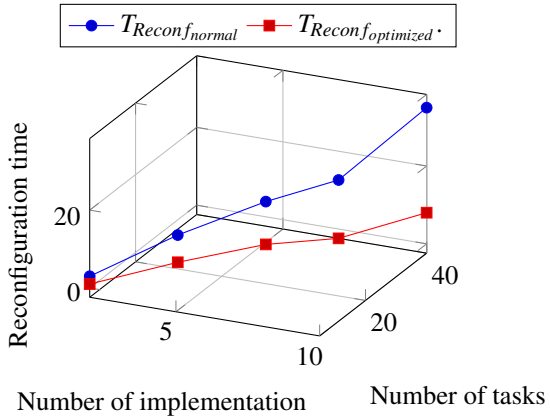

Figure 4: Evaluation of the Reconfiguration Time.

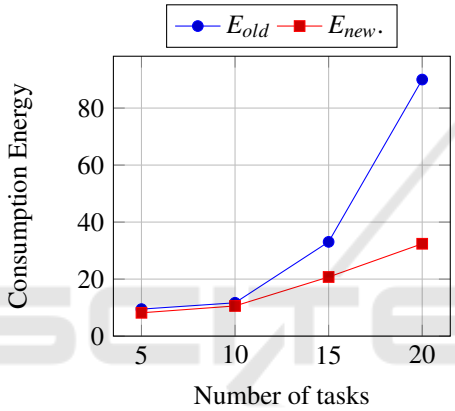

Figure 5: Evaluation of the Energy Consumption.

5.5 Evaluation

In order to generalize the performance evaluation of our strategy, we generate a random system with random task set. The experiments are carried-out on Intel Core i5-4200U processor running at 2.8 GHz with 4GB of cache memory. The curve in Figure 4 shows the variation of the reconfiguration time of the system described in section 3 depending on the number of tasks and the number of implementations.

In Figure 4 we compare the reconfiguration time using the proposed approach with the normal reconfiguration time. It can be seen that the proposed approach allows obtaining a lower reconguration time. This is due to the task merging technique.

Figure 5 depicts the impact of our approach on the total energy consumed by the system.

In this figure, we compute the energy consumption and we compare it with the energy consumed by the system before applying the proposed approach. It is clear from this figure that we have obtained better results.

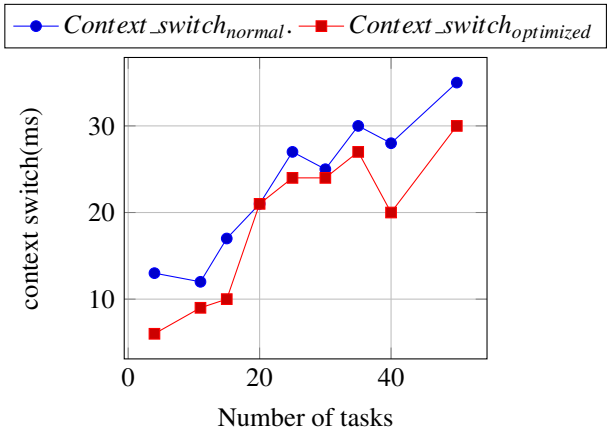

Figure 6: Evaluation of the Context switching.

We also compare in Figure 6 the context switching of the proposed approach to the context switching before applying this approach in a randomly generated system with a number of tasks that varies between 6 and 50. We note that this comparison shows the efficiency of the proposed approach.

The originality of this paper is manifested in the fact that the proposed approach allows the implementation of the multi-core reconfigurable real-time systems by reducing: (i) the task number, (ii) the energy consumption, (iii) the reconfiguration time , (iv) the time overhead in terms of context switching, and (v) the redundancies between implementations.

6 CONCLUSIONS

In this paper, we have described a process for the semi-automatic synthesis of an energy-aware POSIX code for multi-core reconfigurable real-time systems. Firstly, we showed how from the input specification model we generate an initial task model, assign tasks to the cores using partitioning scheduling while meeting timing properties. Secondly, in order to reduce the time overhead, energy consumption and the redundancies between the implementations caused by the huge number of tasks, we proposed a MILP formulation for the problem, which can find the optimal solution for the proposed system model. Thirdly, we generate the optimized POSIX code from the optimized task model. We evaluated the performance of the proposed approach by comparing the obtained results against the performance of the system before applying the approach. As a future work, we want (i) to introduce other criteria to be optimized, and (ii) to fully automate the proposed approach by introducing transformation techniques.

REFERENCES

Burns, A. and Wellings, A. (2009). *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley Educational Publishers Inc, USA, 4nd edition.

Chniter, H., Jarray, F., and Khalgui, M. (2014). Combinatorial approaches for low-power and real-time adaptive reconfigurable embedded systems. In *Proc. Pervasive and Embedded Computing and Communication Systems 4th*, pages 151–157.

Faragardi, H. R., Lisper, B., and Nolte, T. (2013). Towards a communication-efficient mapping of autosar runnables on multi-cores. In *Emerging Technologies & Factory Automation (ETFA), IEEE 18th Conference on*, pages 1–5. IEEE.

Funk, S. and Baruah, S. (2005). Task assignment on uniform heterogeneous multiprocessors. In *Real-Time Systems, Proceedings. 17th Euromicro Conference on*, pages 219–226. IEEE.

Geer, D. (2005). Chip makers turn to multicore processors. *Computer*, 38(5):11–13.

Geismann, J., Pohlmann, U., and Schmelter, D. (2017). Towards an automated synthesis of a real-time scheduling for cyber-physical multi-core systems. In *MODELSWARD*, pages 285–292.

Khan, M. and Hafiz, G. (2014). Simulation of multi-core scheduling in real-time embedded systems. Master's thesis.

Klein, M. H., Ralya, T., Pollak, B., Obenza, R., and Harbour, M. G. (1993). Analyzing complex systems. In *A Practitioners Handbook for Real-Time Analysis*, pages 535–578. Springer.

Lakhdhar, W., Mzid, R., Khalgui, M., and Trèves, N. (2016). Milp-based approach for optimal implementation of reconfigurable real-time systems. In *Proc. International Joint Conference on Software Technologies (ICSOFT) - Volume 1: ICSOFT-EA, Lisbon, Portugal, July 24 - 26, 11th*, pages 330–335.

Lakshmanan, K. S. (2011). *Scheduling and Synchronization for Multi-core Real-time Systems*. PhD thesis, Carnegie Mellon University Pittsburgh, PA.

Lei, H., Wang, R., Zhang, T., Liu, Y., and Zha, Y. (2016). A multi-objective co-evolutionary algorithm for energy-efficient scheduling on a green data center. *Computers & Operations Research*, 75:103–117.

Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61.

Monot, A., Navet, N., Bavoux, B., and Simonot-Lion, F. (2012). Multisource software on multicore automotive ecuscombining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942.

Polakovic, J., Mazare, S., Stefani, J., and David, P. (2007). Experience with safe dynamic reconfigurations in component-based embedded systems. In *Proceedings of the 10th International Symposium on Component-Based Software Engineering (CBSE), USA*, pages 242–257. Springer.

Saidi, S. E., Cotard, S., Chaaban, K., and Marteil, K. (2015). An ilp approach for mapping autosar runnables on multi-core architectures. In *Proceedings of the Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, page 6. ACM.

Singhoff, F. (2014). Real-time scheduling analysis.

Tindell, K. and Clark, J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3):117–134.

Vulgarakis, A., Shooja, R., Monot, A., Carlson, J., and Behnam, M. (2014). Task synthesis for control applications on multicore platforms. In *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, pages 229–234. IEEE.

Wang, H., Shu, L., Yin, W., Xiao, Y., and Cao, J. (2014). Hyperbolic utilization bounds for rate monotonic scheduling on homogeneous multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1510–1521.

Wang, W., Camut, F., and Miramond, B. (2016). Generation of schedule tables on multi-core systems for autosar applications. In *Design and Architectures for Signal and Image Processing (DASIP), Conference on*, pages 191–198. IEEE.

Yehia, K., Safar, M., Youness, H., AbdElSalam, M., and Salem, A. (2011). A design methodology for system level synthesis of multi-core system architectures. In *Electronics, Communications and Photonics Conference (SIECPC), Saudi International*, pages 1–6. IEEE.

Yi, Y., Han, W., Zhao, X., Erdogan, A. T., and Arslan, T. (2009). An ilp formulation for task mapping and scheduling on multi-core architectures. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 33–38. IEEE.