ELSEVIER

Microelectronics Journal ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Microelectronics Journal

www.elsevier.com/locate/mejo

RTDT: A static QoS manager, RT scheduling, HW/SW partitioning CAD tool

H. Tmar[a,b,*], J.-Ph. Diguet[a], A. Azzedine[a], M. Abid[b], J.-L. Philippe[a]

[a]*LESTER Lab, University of South Britanny, France, Lorient*
[b]*GMS Lab, National Engineers school of Sfax, Tunisia, Sfax*

Abstract

The hardware/software partitioning/scheduling relies on two subtasks: the cost function and the real time (RT) analysis. Besides these two subtasks, the proposed generic framework, also called RT design trotter (RTDT), processes the problem of the Quality of Service (QoS) management. The aim is to add a new dimensions to solution selection, namely the guarantee of QoS from both application quality and RT issue points of view. The proposed framework defines an iteration loop of three steps that solve the sub-problems. The cost function takes into account the system on chip (SoC) area and the static and dynamic power dissipation. We show how our tool can be used to rapidly evaluate the impact of the application quality and the RT constraints choices (QoS parameters) over the final cost.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: RT system; Power model; RT scheduling; Static QoS manager; SoC

1. Introduction

CAD tools are now crucial for System On Chip (SoC) industry in order to get back a vital benefit from the joint evolution of applications and VLSI circuits. Basically the issue is no more the amount of transistors available on chips but the way to follow up the potential they offer with reduced design delays and low cost methodologies. The questions related to the complexity of SoC are many-fold and include different issues like reliability, design delay, power and real time constraints (RTCs). As previously performed in other industrial domains like avionics and automotive, the microprocessor industry is evolving towards unavoidable knowledge management methods in order to reduce design cost and delay while focusing on few real value-added innovations [1]. In the domain of SoC the designers rely on reuse of a reconfigurable software (SW) or hardware (HW) intellectual property blocks (IP). IP based framework for simulations are available in both academic [2] and industry [3] areas. However, exhaustive IP libraries with qualified components in terms of power and execution time for various targets, offer a real interest only if CAD tools can speed up the associated design space exploration and validate the set of selected solutions. The objective of this work is to provide a framework for low power real time embedded systems codesign in order to rapidly select promising architectures.

This kind of systems is typically reactive, real time, increasingly control dominated, and data dependent for optimization purposes. The design of such complex systems requires high-level design tool in order to rapidly select and synthesize promising architectures. Due to hard RTCs, HW implementation (e.g. IP based) of critical functions must be performed. It is then necessary to use a SW/HW codesign approach, which must allow a minimal design cost and a minimum time to market. In order to ultimately avoid costly redesigns, the system architecture has to more or less meet timing requirements, on the first try, at an early phase in the design process. Where there have been some research efforts which addressed this problem, the approaches used remain more pessimistic than necessary. They are based on worst case analysis technique [4–7], that consists of a priori time slot reservation for each task, namely, worst case execution time (WCET) consideration

*Corresponding author. GMS Lab, National Engineers School of Sfax, Tunisia, Sfax. Tel.: +216 74 276 400; fax: +216 74 274 437.
E-mail address: tmar@iuplo.univ-ubs.fr (H. Tmar).

0026-2692/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2006.07.028

to guarantee RTCs. While this technique is necessary for hard RT systems, it is less justified for soft RT systems, where methods based on probabilistic schedulability analysis are more and more studied in the embedded SW systems design domain [8–13]. Thus, the solution which appears is no longer a real time management but rather a QoS management [14]. For these reasons, we propose to use a notion of QoS instead of RT during the partitioning/scheduling step. Thus, we add a new category of task for periodic and aperiodic "soft RT" tasks. This kind of tasks respect the RT constraints with a given probability. The rest of the paper is organized as follows.

In the next section, we place our work within the state of the art. In Section 3 we specify the problem. The architecture model is presented in Section 4. In Section 5 the generic design space exploration framework is detailed including area and power models, and the basic RT scheduling assumptions. Section 6 presents the QoS model. A football player robot application is experimented in Section 7. Finally we draw conclusions and perspectives.

2. Related work

Our research results can be viewed in the context of two areas of related works: high-level HW/SW partitioning-scheduling for RT embedded systems, and quality of service management. The codesign literature is an active domain that embraces various topics like system specification, area/power/delay estimations for HW/SW candidates, HW/SW partitioning, HW/SW communication synthesis and cosimulation. A recent overview of the different domains can be found in [15]. The specific topic addressed in this paper is related to the automatic HW/SW partitioning issue under QoS constraints based on an optimization cost function involving power and area. Some other important features relative to our work are the multi-rate, the task preemption (or switching) overhead and the aperiodic task scheduling from a RTOS point of view, the multi-granularity regarding the design space exploration, and finally the genericity of the architecture/application specification concerning the CAD tool.

A complete framework for automatic HW/SW partitioning is detailed in [4]. It includes multi-granularity selection in the context of performance optimization. In the context of real time scheduling, static non-preemptive scheduling [5,6] is usually adopted in embedded real-time systems since dynamic scheduling cannot guarantee the RTCs and incurs a computational overhead. The objectives formulated in [16,17] are quite close to ours; the multi-rate issue is handled and preemptive scheduling is considered. It also includes the RTOS overhead but does not take into account preemptions due to the access to critical or shared resources. The cost function includes area and power but with a very simple model based on average power dissipation. The method is extended to aperiodic tasks where the time slots are reserved within the hyperperiod. This technique uses inter-instance minimum delay which means a pseudo-periodization, it can lead to very costly design if the tasks are rarely launched or if their execution is not critical. An interesting clustering method is used to reduce the partitioning complexity.

QoS has been often addressed in multimedia, video, and networking research communities, but rarely in the design community. However, where there have been some research efforts for co-synthesis of multi-task embedded systems only a few research results exist for QoS management. Previous works, which can be found in the domain of PC-based servers for video tracking [18] or web applications [19], propose an interesting close-loop approach for QoS and CPU Bandwidth adaptation. In the domain of mobile system, Agile [20] proposes some extensions to the eOS NetBSD for media delivering. The authors have implemented the concept of fidelity to drive the QoS management in term of video cadence and picture quality. The main conceptual result in system design literature was presented in [21]. The authors study how multiple voltages can be used to simultaneously satisfy HW requirements and minimize power consumption while preserving the requested level of QoS; in that case satisfying latency and synchronization requirements. Given task sets and a processor with multiple voltages, they search all the feasible competitive schedules with the minimal energy consumption and memory requirement assuming that two schedules are competitive if neither outperforms the other in both energy consumption and memory requirement. However, they do not consider the resource sharing possibility between tasks and assume that all tasks are run on the processor. Compared with this last previous approach, our work differs in three aspects: first we address the domain of RT HW/SW co-synthesis. Second, we process the problem of QoS in terms of application quality and RT constraints choices. Third, we consider the possibility of HW resource and coprocessor sharing between tasks.

3. Problem specification

Globally we address the partitioning of a dependent task graph over a multi-processor architecture in a real time context. Solving this question directly is extremely complex and heuristics can lead to oversized solutions if dynamic systems are considered. This aspect is becoming more critical since we observe a trend toward systems with very unprecise WCET. This unpredictability is increasing in modern embedded systems for various reasons that can be related to the adaptive or data dependent task specification or to the complexity of processor architectures [22]. Given that fact, we have split the problem and defined realistic methodology and tool, which are interactive and based on a four steps strategy.

- *PACM clustering:* the first step consists in assigning a set of tightly dependent and communicating tasks to an enhanced processor architecture (PACM cf. Fig. 1);

- *QoS selection and Trade-off tuning:* the second step interacts with the designer in order to select a category for each task: Hard, soft or non-real time. It uses the Radha Rathan Tool [23] to get the time constraint interval for each task;

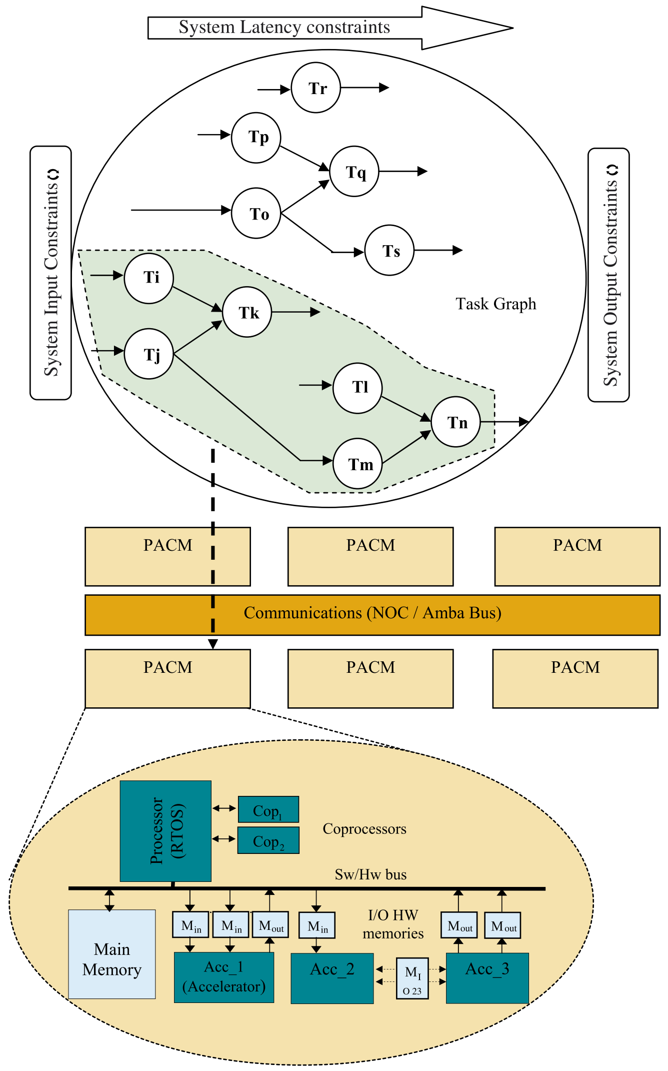


Fig. 1. Task graph assignation to the target architecture.

- *HW/SW partitioning and real time scheduling*: the third step proceeds the HW/SW partitioning within a real time context. At this level, the designer controls the cost function in terms of Area/Power tradeoff and the IP candidates for each task.
- *Post-partitioning memory optimization*: finally, memory-merging opportunities are analyzed, this step cannot be included in the partitioning/scheduling loop but can be efficiently performed over a small set of solutions.

This paper specifically addresses the second and third steps.

4. Monoprocessor architecture model

In this section, we describe the PACM architecture. Note that the PACM architecture is composed by one Processor Accelerators, Coprocessors and Memories. Then, we detail our approach for SW/HW communication modeling.

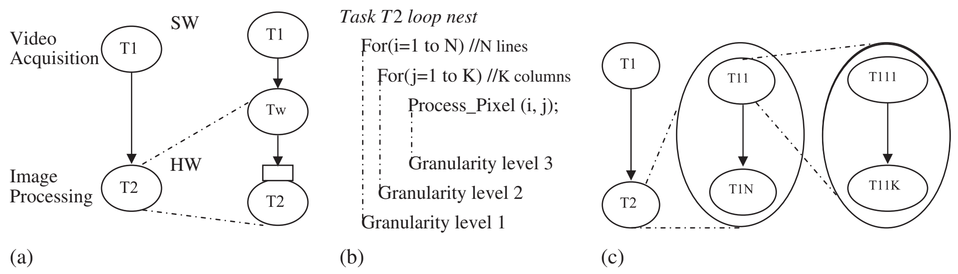
4.1. PACM architecture

Fig. 1 presents the PACM model. Basically our architecture is built around a processor core (e.g. IP NIOS), which offers configurations opportunities for adding coprocessors acceded through the processor registers. The processor is communicating with dedicated HW accelerators through a standard bus (e.g. Avalon). Hence, three families of implementations can be considered: (i) SW, (ii) SW with coprocessors that can be shared with other tasks and (iii) dedicated HW.

4.2. HW/SW communications

4.2.1. General case

The communications between HW and SW are implemented as a particular new task during the partitioning/scheduling process. The period of the communication task depends on the granularity level of the HW implementation as indicated in Fig. 2. If we consider two tasks, producer task T_P produces a data for the consumer task T_C, four cases can be distinguished (see Fig. 3):



Level	M_{in} size (pixel)	# transfers	Pxel/transfer	Communication delay with $T=T_{init}+N_{data}*T_{data}$
1	N*K	1	N*K	$T_{init}+N*K*T_{data}$
2	K	N	K	$N*T_{init}+N*K*T_{data}$
3	1	N*K	1	$N*K*T_{init}+N*K*T_{data}$

Fig. 2. Multi-granularity hardware solutions for a loop nest.

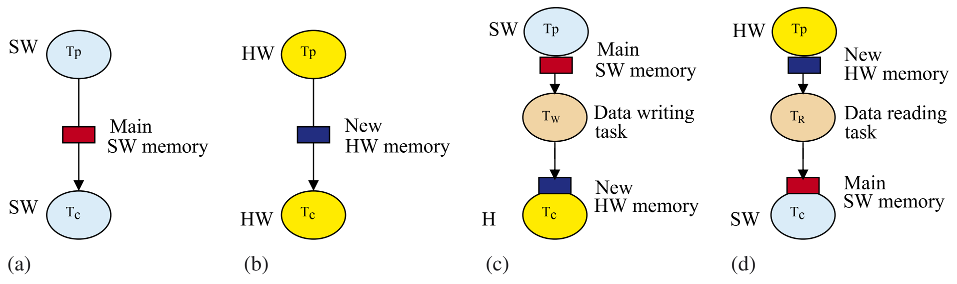


Fig. 3. Memory and tasks implementations for HW/SW communications.

- If T_P and T_C are SW, there is no need for communication task neither additional memory.
- If T_P and T_C are HW there is no need for communication task, however a communication memory (output for T_P and input for T_C) is added.
- If T_P is SW and T_C is HW, a new communication task is created, this task writes the data produced by T_P in a new input HW memory.
- If T_P is HW and T_C SW, a new communication task is created, this task reads the data produced by T_P in a new output HW memory.

4.2.2. *Communication task features*

The first point is the period value and the amount of data to transmit. The period of a communication task T_{com} between two tasks T_P and T_C equals to the smallest period of the two dependent tasks. The amount of data transmitted during the minimal period equals to the amount of data produced or consumed during this period. The second point relies to the communication task execution time C_{com}, which is computed as follows:

If DataWidth < BusWidth:

$$C_{com} = \left(N_{cycles\_init} + \frac{N_{Data}}{\left\lfloor \frac{BusWidth}{DataWidth} \right\rfloor} N_{cycles\_com} \right) Clock_{Bus} \tag{1}$$

Else:

$$C_{com} = \left(N_{cycles\_init} + N_{Data} \left\lceil \frac{DataWidth}{BusWidth} \right\rceil N_{cycles\_com} \right) Clock_{Bus}, \tag{2}$$

where N_{cycles_init} is the number of cycles required to initiate the data transfer N_{Data} is the amount of data to transmit, N_{cycles_com} is the number of cycle per data transfer. All these parameters are generic and can be easily specified.

4.2.3. *Memory implementation*

The task schedulability is computed while considering independent tasks; actually the question of task dependencies is solved by shifting release dates of consumer tasks [24]. However, this assumption is valid only if the communication memories have been correctly selected. Two main issues must be considered.

- Data availability. The memory size must be large enough to store, without overwritten data, the produced data to be read by the consumer task.
- Data access conflicts. This problem is solved by the RTOS when tasks are SW but if one of the tasks is HW then conflicts can occur. Our framework proposes two solutions to address this problem. In the first one the communication memory is implemented as a critical resource with priority ceiling. In such a case, the task that accesses the memory cannot be interrupted by another one when the data writing (or reading) is not accomplished. In the second one, the communication memory is implemented in a pipeline way with two memories alternatively used for writing and reading exchanged data.

The default implementation (without any designer directive) is non-blocking communications, in that case the memory size MS is computed as follows (P: producer C: consumer):

If $P_P \geqslant P_C (P_P = n\, P_C$ and $n \geqslant 1)$

$$MS = 2\,N\ Dataout\_P = 2\,n\,N\ DataIn\_C, \tag{3}$$

Else if $P_P < P_C (n\, P_P = P_C$ and $n > 1)$

$$MS = 2\,n\,N\ Dataout\_P = 2\,N\ DataIn\_C, \tag{4}$$

where P_P and P_C are the periods of the producer task and the consumer task, respectively.

4.2.4. *Multiple inputs/outputs*

Including memory reuse optimization during the partitioning/scheduling process incurs a complexity gap, which is not acceptable. Moreover memory optimization can efficiently be performed afterwards over a small set of promising solutions. So, in case of multiple dependencies, (1 producers and several consumers 1 consumer with several producers), HW/SW, HW/HW and SW/HW communications are implemented as single links with one dedicated memory. A shared memory must be explicitly defined by the designer within the specification task graph.

5. Design space exploration and evaluation

In this section, we outline some basic RT scheduling assumptions. Then, we detail the generic design space exploration framework including area and power model.

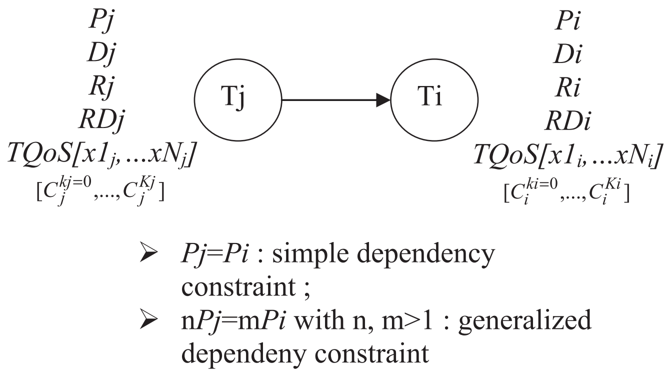
5.1. Real time scheduling strategy

5.1.1. Task classification

Usually, the real-time embedded systems require a simple and safe scheduler, which can guarantee that critical aperiodic or periodic tasks meet their deadlines. For these reasons, a static high priority first (HPF) scheduling policy has been adopted where the fixed priorities are computed as the inverse of the task period. The worst-case response time is computed with an exact analysis [25].

In a first approach we consider two kinds of tasks (we will show in Section 6 how the QoS management can bring a third kind of task). The first category is composed by the periodic tasks that are scheduled by means of hard RTCs and by sporadic tasks with hard RTC. Like in [16] we consider the sporadic task as periodic task with period equals to the minimum delay between two subsequent executions, this value is provided by the Radha Ratan tool [23]. The second category includes the non-critical sporadic tasks, which are handled, by a server task with the lowest priority that can be fixed by the designer. The priority is computed as the inverse of the period task. The dependencies, which can be related to precedence constraints, are eliminated by modifying absolute deadlines and release dates, as detailed in our previous work [24]. The computations of new release dates are performed in the precedence order within the task graph. More formally, the schedulability analysis is performed while considering independent tasks, then when an implementation solution is found, the new release dates RD_i are computed in order to respect the assumption, namely the precedence constraints. The computations are performed in the precedence order within the task graph, for example, regarding the tasks T_i and T_j from Fig. 4.

If T_j is mapped on HW and T_i is mapped on SW; or T_j and T_i are mapped on HW following the HW/SW



Pj
Dj
Rj
RDj
$TQoS[x1_j,...xN_j]$
$[C_j^{kj=0},...,C_j^{Kj}]$
Tj
Ti
Pi
Di
Ri
RDi
$TQoS[x1_i,...xN_i]$
$[C_i^{ki=0},...,C_i^{Ki}]$
Pj=Pi : simple dependency constraint ;
nPj=mPi with n, m>1 : generalized dependeny constraint

Fig. 4. Precedence generalized constraints.

partitioning decision, then:

$$\mathrm{RD}_i = \max\{\mathrm{RD}_i;\ \mathrm{RD}_j\ + (n-1)P_j + C_j\}$$

For the three remaining cases (T_j is a SW task and T_i is a HW task, T_j and T_i are a SW tasks with theT_j priority is higher than the T_i priority, and T_j and T_i are a SW tasks with the T_i priority is higher than the T_j priority) refers to [24].

In Fig. 4, P_j is the period, D_j is the deadline, R_j the response time, RD_j is the release date, $\mathrm{TQoS}_j[x1_j,\ldots,xN_j]$ is the QoS vector where xk_j is a ratio representing different aspects of QoS measurement (detailed in Section 6) and $[C_j^{kj=0},\ldots,C_j^{kj}]$ is the execution vector where C_j^{kj} is the delay associated to the kjth implementation of task j.

Remark. . Tasks being periodic, each period is a fixed quantity. Absolute task deadlines are assumed to lie within the periods and, for simplification here, to coincide with the next task request. Uncertainty in computation time is being taken care of by letting each C to be a random variable characterized by a probability distribution. Regarding aperiodic tasks, the period means the minimum delay between two successive executions of the task. So in the case of hard RTC, this delay is the lowest bound but with soft RTC this delay is an average value.

5.1.2. Response time computation

The exact response time is computed iteratively with the following equation:

$$\forall T_j \in \mathrm{HP}(i), \exists R_i \leqslant D_i \backslash R_i = (C_i + B_i) + \sum_{j\in \mathrm{HP}(i)} \left\lceil \frac{R_i}{P_j} \right\rceil (C_j + C_{\mathrm{sw}}), \tag{5}$$

where HP(i) is the set of tasks with higher priority comparing to task i; R_i the worst case response time of task i; C_i the execution time of task i; B_i the longest time that task i can be delayed by lower priority tasks (e.g. resource sharing), P_j the period of task j, $C_{\mathrm{sw}} = \delta_0 + \Sigma_k \delta(k)$ with δ_0 is the context switching overhead without any coprocessor and δ_k is the overhead due to the coprocessor k.

The context-switching overhead is the delay between the suspension of a given task and the activation of another task. The difficulty is that δ_{sw} depends not only on the target processor and on the RTOS and its configuration but also on the number of tasks in the system and on the number of co-processors which both are evolving during the HS/SW partitioning. The influence of the number of tasks is not insignificant but can be neglected compared to the coprocessor context saving influence. Moreover, without coprocessor, the available overhead metric is usually an average value estimated with different task sets. The influence of a co-processor is obviously related to the number of data and registers status.

5.2. Design space exploration for HW/SW partioning

5.2.1. Cost function

The cost function takes into account the global area of the SoC and its energy consumption. At a high level of abstraction only relative estimations can be used for SW and HW IPs, the cost function is used to guide the selection of reduced set of solutions where the designer should be a "good" solution after the refining steps. In order to eliminate units, relative costs are used to evaluate the cost value for a given schedulable solution S:

$$\mathrm{Cost}(S) = \alpha\,\frac{\mathrm{Area}(S) - \mathrm{MinArea}}{\mathrm{MinArea}} + \beta\,\frac{\mathrm{Pw}(S) - \mathrm{MinPw}}{\mathrm{MinPw}}, \tag{6}$$

with $\alpha + \beta = 1$ and where MinArea is the schedulable solution with minimal area without any power consideration and MinPw the schedulable solution with minimal power without any area consideration. Note that the area cost influences the power consumption through the static power evaluation so the parameter α also act on the power optimization.

5.2.2. Area cost

The area cost includes the data and code memory size for SW implementations, the area of coprocessors that can be shared by various tasks, the area of HW accelerators and finally the area of memories added for communications.

5.2.3. Power cost

The model for power evaluation is much more complex. Firstly the dynamic power consumption depends on the SoC activity, which is strongly related to the task scheduling and switching. Secondly, the evolution of VLSI technology shows that static power consumption [26], especially in FPGAs, can no more be neglected. Finally, in mobile embedded systems the important metric is the system life span. It means that the energy use must be optimized. However, in our context of periodic tasks the energy optimization is equivalent to the average power minimization over the hyper period. Our power model for an implementation S is given by

$$\mathrm{Pw}(S) = \mathrm{Pw_d} + \mathrm{Pw_s}, \tag{7}$$

where $\mathrm{Pw_d}$ is the average dynamic power dissipated during a hyperperiod T_G and $\mathrm{Pw_s}$ the average static power.

5.2.4. Dynamic power/energy metric

Let $\mathrm{Pw_d}$ the average dynamic power dissipated during a hyperperiod T_G.

$$\mathrm{Pw_d} = \frac{E_d}{T_G}, \tag{8}$$

$$E_d = E_d(\mathrm{sw}) + E_d(\mathrm{hw}) // E_d\text{:}$$
$$\text{consumed during a period}: T_G, \tag{9}$$

$$E_d(\mathrm{sw}) = E_d(\mathrm{idle}) + E_d(\mathrm{switch}) + E_d(\mathrm{exe}), \tag{10}$$

$$E_d(\mathrm{exe}) = T_G \sum_{i\in \mathrm{SW}} P_{wd}(i)\frac{C_i}{P_i} // P_{wd}(i)\text{:}$$
$$\text{average power for task } i, \tag{11}$$

$$E_d(\mathrm{switch}) = T_G P_{wd}(\mathrm{switch}) \sum_{i\in \mathrm{SW}} \frac{C_{sw}}{P_i} // P_{wd}(\mathrm{switch})\text{:}$$
$$\text{avg task switching power}, \tag{12}$$

$$E_d(\mathrm{idle}) = P_{wd}(\mathrm{idle}) T_G \left(1 - \sum_{i\in \mathrm{sw}} \frac{C_i + C_{sw}}{P_i}\right) // P_{wd}(\mathrm{idle})\text{:}$$
$$\text{avg proc. idle power}, \tag{13}$$

$$E_d(\mathrm{hw}) = T_G \sum_{i\in \mathrm{HW}} P_{wd}(i)\frac{C_i}{P_i}. \tag{14}$$

Important note: For flexibility and genericity concerns, the task average dynamic power values $P_{wd}(i)$ are normalized versus the supply voltage and clock frequency and the average task static power is expressed by area unit (W/gate or W/μm^2 as indicated in [27]).

5.2.5. Static power/energy metric

The available static power, usually given by means of mW/area, depends mainly of the leakage power, the supply voltage, the transistor count and a technology-dependent parameter:

$$\mathrm{Pw_s} = f(N_{tr} K_{design} I_{leakage} V_{dd}).$$

Our model uses $\mathrm{Pw_s}$(sw) and $\mathrm{Pw_s}$(hw) for SW and HW parts, respectively. A dynamic strategy can be adopted for static power management if HW accelerator power supply can be switched off when unused. In such a case the average static power dissipation is given by

$$\mathrm{Pw_s} = \mathrm{Pw_{offSW}}\,\mathrm{Area(SW)} + \mathrm{Pw_{offHW}} \sum_{i\in \mathrm{HW}} \mathrm{Area}(i)\frac{C_i}{P_i}.$$

Without HW dynamic power supply management, we obtain:

$$\mathrm{Pw_s} = \mathrm{Pw_{offSW}}\,\mathrm{Area(SW)} + \mathrm{Pw_{offHW}} \sum_{i\in \mathrm{HW}} \mathrm{Area}(i).$$

5.2.6. Partitioning algorithm

5.2.6.1. Solution evaluation. The main difficulties during the partitioning/RT scheduling algorithm are firstly the size of the design space, especially, since multiple granularity solutions can be considered for each HW task implementation, and secondly the iterative scheduling of task worst case response time.

A solution is valid if firstly all tasks meet their deadlines and secondly if the current cost belongs to the N first best costs. Contrary to the response time computation, the cost is not iterative and must be evaluated first. Thus the schedulability is computed in a three steps (see Fig. 5) in

```
Boolean **Schedulable** (S){
    U = ProcUseRate(S) // Processor use rate
    Step 1 : if (U+rs > 1) //rs: server task cpu ratio
        return false;
    Step 2 : else if $U + rs \le n*(2^{\frac{1}{n}} - 1)$ return true;
    Step 3 : else {
        for all Ti by Increasing Priority Order
            Ri = ExactResponseTimeAnalysis(Ti);
            if Ri > Pi return false;
            else return true;
        }
}
```

Fig. 5. Schedulability test.

order to restrict the use of iterative response time computations. The algorithm first test if the processor rate is lower than 1. As a second test, the fast rate monotonic analysis (RMA) is performed; it gives a sufficient but not necessary condition for schedulability. Finally if the first tests are valid an exact analysis is performed. Note that the designer can specify the CPU ratio *r*s to be guarantied for the server task.

5.2.7. *Design space exploration*

Two methods are currently available, the first one is exact and based on the Branch & Bound algorithm, the second one is heuristic and uses a simulated annealing approach (SA). The B&B starts with a left edge branch representing a complete SW solution and progresses towards a complete HW solution with the finest granularity degree, the tasks are ranked in a branch according to the priority order. On a given branch, for each task added, the cost is first evaluated; if the cost is lower than the best current solution then the task schedulability is computed according to the method described in Fig 5. When the cost is larger than the best value or when the solution is not schedulable then a new task implementation is evaluated. If no more implementation is available, another implementation is considered for the previous task in the current branch and so on. The main difficulty occurs when a HW solution with a fine granularity implies the insertion of a communication task with a shorter period than its predecessor in the branch. In such a case the schedulability of previous tasks with a lowest priority must be computed again. The B&B is efficient even for large graphs (100 tasks) when there are a few schedulable solutions, but its computation is prohibitive when numerous solutions are proposed for each task. When the response time computation dramatically slows down the design space exploration, the SA heuristic can efficiently relay the B&B.

5.3. *Generic codesign framework*

One of our objectives was to carry out an interactive tool for designers to easily test various configurations in terms of tasks versions and architectural implementations. Our flow is described in Fig. 6.

We have opted for the task graph defined in [23] in order to use the Radha/Ratan tool to obtain internal task constraints from Input/Output system constraints. The uncertainty on I/O events and periods are expressed as a period intervals [$P_{\min}$, $P_{\max}$] for each system task.

Each task is described in a C code file, the Design Trotter framework [28] first generates a hierarchical data flow graph (HDFG) from which different kind of estimation can be produced like delay/area of FPGA HW components [29] or power SW estimation by hierarchically combining of CData Flow Graph from [30]. Another solution consists in using qualified SW or HW IP specification.

By combining estimations data, the initial task graph and designer choices, a new file is generated, this “file.cde” (see Fig. 6) includes the task constraints selected and the description of all task implementations. The “file.Arch” gives the architectural parameters, like the V_{dd}/clock modes V_{dd}, for HW and SW parts, the bus protocol and so on.

The final solutions selected after the partitioning/scheduling step are finally stored in the “file.imp”.

6. Static QoS manager

In this section, we present the justifications and components of the QoS model. Then, we address the coherence checking for the static QoS manager and the feedback scheduling analysis.

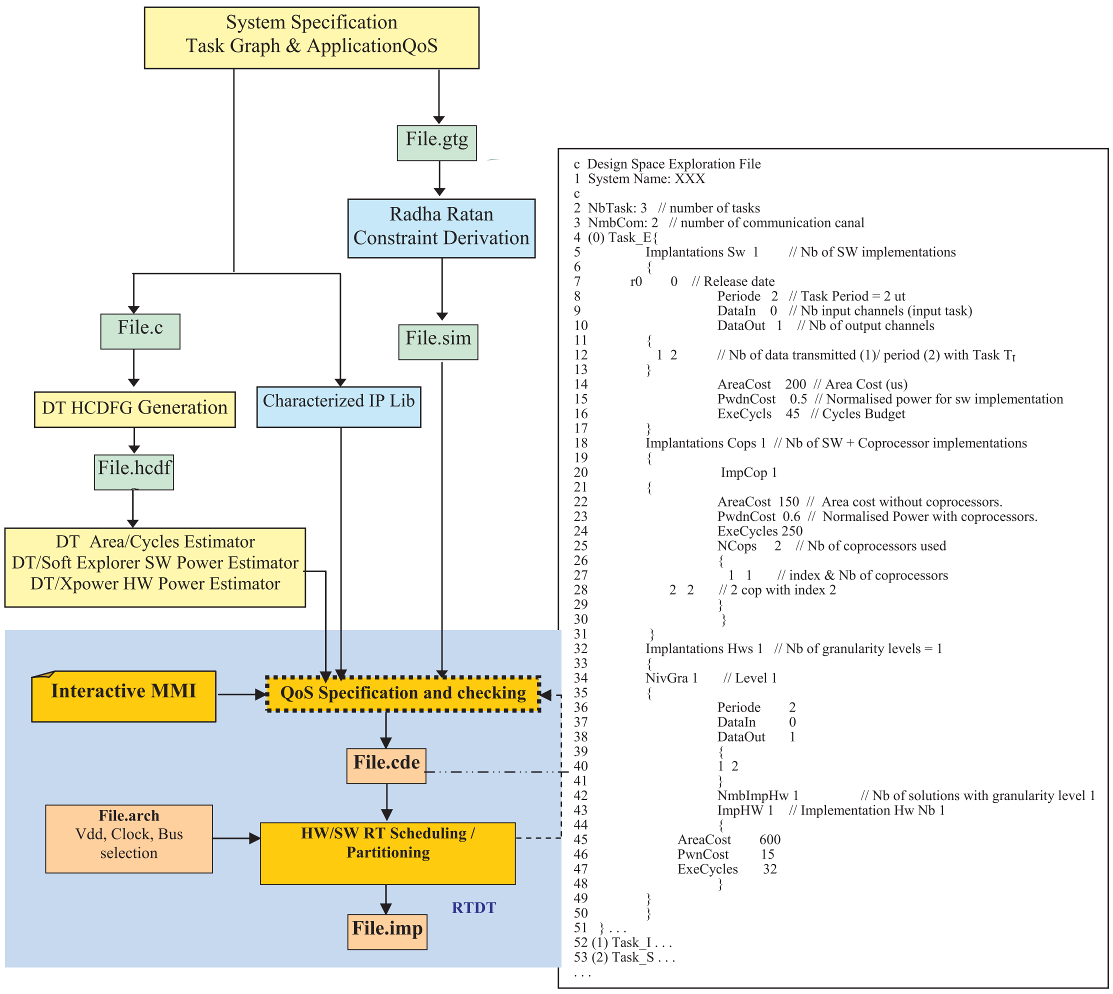
6.1. *Context definition*

One of the major issues in real time embedded systems is the question of task execution time which can vary depending on data and on environment events The second point is the question of periodic and aperiodic tasks with very versatile inter-iteration delays. In such uncertain context, the choice of the worst case can lead to very costly and oversized implementations. As systems are growing in complexity, this overestimation become unacceptable and new method must be considered.

6.2. *Model*

We propose to insert a new step within the codesign flow. This step is based on a QoS model and produces the specification file according to the designer choices. Thus, we add a new kind of tasks for periodic and aperiodic soft real time tasks. This category of tasks respects the RTCs with a given probability, the aim is to avoid worst case assumptions and deterministic guarantees for periodic and aperiodic tasks with soft RTCs by means of probabilistic scheduling. Each task can be represented by a QoS array of N parameters:

$\mathrm{TQoS}_i[x_1, \ldots, x_N]$,



```
c  Design Space Exploration File
   System Name: XXX
c
   NbTask: 3   // number of tasks
   NmbCom: 2   // number of communication canal
   (0) Task_E{
               Implantations Sw  1        // Nb of SW implementations
               {
           r0       0    // Release date
                          Periode   2   // Task Period = 2 ut
                          DataIn    0   // Nb input channels (input task)
                          DataOut   1    // Nb of output channels
               {
                 1  2         // Nb of data transmitted (1)/ period (2) with Task T_I
               }
                          AreaCost    200  // Area Cost (us)
                          PwdnCost    0.5  // Normalised power for sw implementation
                          ExeCycls    45   // Cycles Budget
               }
               Implantations Cops 1  // Nb of SW + Coprocessor implementations
               {
                           ImpCop 1
               {
                          AreaCost  150  //  Area cost without coprocessors.
                          PwdnCost  0.6  //  Normalised Power with coprocessors.
                          ExeCycles 250
                          NCops     2    // Nb of coprocessors used
                          {
                            1   1       // index & Nb of coprocessors
                   2   2      // 2 cop with index 2
                          }
                           }
                }
               Implantations Hws 1   // Nb of granularity levels = 1
               {
               NivGra 1       // Level 1
               {
                          Periode       2
                          DataIn        0
                          DataOut       1
                          {
                          1  2
                          }
                          NmbImpHw 1                  // Nb of solutions with granularity level 1
                          ImpHW 1    // Implementation Hw Nb 1
                          {
                    AreaCost       600
                    PwnCost        15
                    ExeCycles      32
                          }
               }
               }
     } . . .
   (1) Task_I . . .
   (2) Task_S . . .
. . .
```

Fig. 6. RTDT codesign flow.

where x_i is a ratio representing different aspects of QoS measurement. In this paper we consider two dimensions:

$$\mathrm{TQoS}(i)[\mathrm{AQoS}, \mathrm{RTQoS}].$$

The first term AQoS represents the QoS specific to the task, namely the application quality. For instance, it can be a data rate for network management task or a number of bits for pixel coding. The second term RTQoS is related to the RTCs and means the minimum ratio of deadlines that must be met. It means that the execution time $W(i)$ considered for task i during RT analysis is such as

$$\mathrm{Probability}(\mathrm{RL} - \mathrm{Execution} - \mathrm{Time}(i) \leqslant W(i)) \\ = \mathrm{RTQoS}(i),$$

where RL−Execution−Time means the real life execution time of task i. Regarding the RTQoS dimension, the designer must choose the minimum ratio of deadline that must be met for each task. According to the appropriate probability law, the correspondent execution time is computed and considered. For example, if the RTQoS(i) is set to 1, then we consider the WCET(i) for task i. The QoS task choices are usually not independent and the QoS specification step must check the relation that exists between task application qualities according to these choices. Regarding the real time issue, a task with a RTQoS equals to 1 ($W(i) = \mathrm{WCET}(i)$) should not be delayed by another to miss its deadline.

6.3. QoS decision and generation step

In this subsection, we detail the QoS specification and checking step that is shown in Fig. 7. The entry of that step is a tasks configurations file, in which we save all the possible versions for each task. As mentioned above, a version or a QoS task choice of a task is an application quality/$W(i)$ couple. By combining estimation data from the implementation library and the designer choices, a new file is generated. This “file.cde” (see Fig. 6) includes the tasks constraints selected and the description of all task implementations.

The QoS coherence checker tests three cases that can lead to a QoS inconsistency:

- Test 1: Data dependency
- Test 2: Resource sharing
- Test 3: Task priority

For each QoS dimension *i*, the designer can select the exclusive rule that he wants to be applied regarding QoS homogenization:

- Rule l: QoS Round down
- Rule 2: QoS Round up
- Rule 3: QoS Unchecked

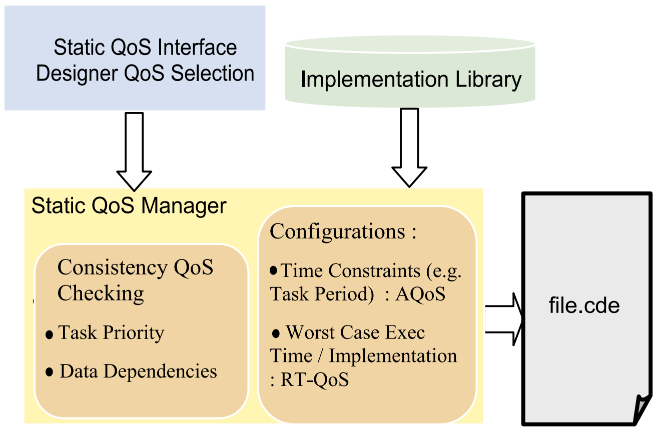
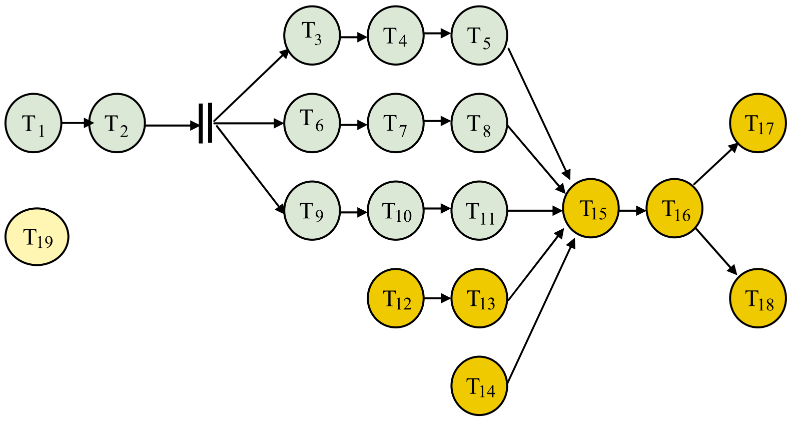


Fig. 7. QoS specification and checking.

If we consider AQoS as a minimum data-rate, then the designer must configure the QoS checker in order to verify QoS homogenization. Namely, for Test1 the designer will select Rule1, or Rule2 if he wants to favour power optimization or application quality respectively. In that particular case, the task priority and resource sharing tests can be ignored since they do not influence the AQoS. For example, if power optimization is favoured, the QoS checker is configured as follows for AQoS dimension: {(Test1,Rule1); (Test2,Rule3); (Test3,Rule3)}

The RTQoS dimension is only influenced by priority assignment and resource-sharing tests since the data dependency question is solved during the RT scheduling analysis as explained in Section 5.1. Thus, the QoS checker must perform the Test2 and Test3 with Rulel or Rule2 depending on the designer choices. Besides these test/rule specifications, the QoS checker must verify that, for all couples of tasks T_i and T_j, if priority(T_i)$<$priority(T_j), then RTQoS(T_i)$<$RTQoS(T_j). Similarly, tasks that share resources must have the same RTQoS. If these tests are not valid, then Rule1 or Rule2 must be performed. For example, if power optimization is favoured, the QoS checker is configured as follows for dimension RTQoS:



	Title	Period (ms) (min, med, max)	Data In (32 bits)	Data Out
T_{12}	wireless Acquisition	20, 50, 100	----	6
T_{13}	Decryption	20, 50, 100	6	6
T_{14}	Sensor Acquisition	20, 50, 100	----	20
T_1	Video Acquisition	40, 100, 200	----	2048
T_2	Bayer Interpolation	40, 100, 200	2048	2048
T_3, T_6, T_9	Threshold (Red, Green, Blue)	Granularity dependent : (40, 40/256), (100, 100/256), (200, 200/256)	2048	2048
T_4, T_7, T_{10}	Filtering (Red, Green, Blue)	Granularity dependent : (40, 40/256) (100, 100/256), (200, 200/256)	2048	2048
T_5, T_8, T_{11}	Object position computation (Red, Green, Blue)	Granularity dependent : (40, 100, 200)	2048	6
T_{15}	Data Merging	20, 50, 100	44	16
T_{16}	Trajectory computation	20, 50, 100	16	20
T_{17}	wireless Emission	20, 50, 100	20	----
T_{18}	20 Motor Command	20, 50, 100	20	----
T_{19}	Server Task	200	40	----

Fig. 8. Football player robot application.

{(Test1,Rule3); (Test2,Rule1); (Test3,Rule1)}. The QoS aware codesign flow is illustrated in Section 7.

6.4. Feedback scheduling analysis

After the HW/SW RT partitioning/scheduling step, a timing analysis report is returned to the designer. This report contains the probability scheduling results, it indicates the scheduling safety rate (SSR), namely, the rate of success of the server task capacity to support the probably deadlines violations, such as

$$\text{Probability}\left(\sum_{i\in \text{SW}} \text{RL}-\text{Execution}-\text{Time}(i) \leqslant \left(T_s + \sum_{i\in \text{SW}} W(i)\right)\right) = \text{SSR},$$

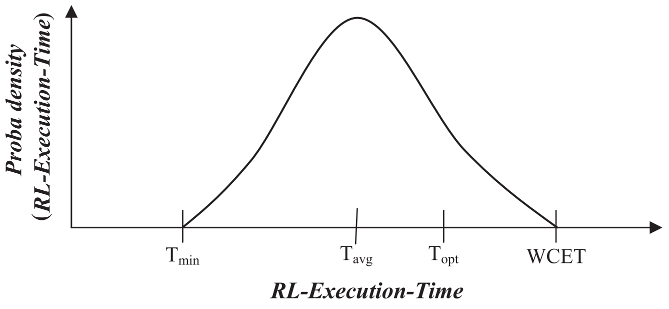


Fig. 9. RL-execution-time distribution.

where T_s is the server task delay, RL−Execution−Time(i) is the real life execution time, and $W(i)$ is the execution time considered for SW task$i$, as mentioned above.

7. Case study: a football player robot application

The case study described in Fig. 8 is a football player robot application with video tasks for object detection, wireless communications for message exchanging with other devices, motors controls, sensor acquisition, image processing and decision computation. Various HW with different granularities, SW and SW with coprocessor implementations are considered for the set of tasks. Note that T_w is the server task with the lowest priority; it includes all aperiodic SW tasks with soft RTCs or without RTCs.

Regarding the period values, the video tasks $T_1,\ldots,T_{11}$ have lower priorities than the other remaining tasks $T_{12},\ldots,T_{18}$. We consider that all tasks from 12 to 18 are hard real time namely:

$$\forall i \in \{12,\ldots,18\}, \text{TQoS}_i = [1,1].$$

For tasks from 1 to 11, different tradeoffs are experimented. Three video rates are considered: 40 ms (High: $P_{\min}$), 100 ms (Medium: P_{med}) or 200 ms (Low: $P_{\max}$).

The SW RL-Execution-Time is obviously different for all these tasks, it also vary within the interval [$T_{\min}$, WCET]. For instance, the video acquisition and the Bayer interpolation delay variation are reasonably limited, but

Table 1
TQS [AQoS, RTQoS] tradeoffs

Sol	1	2	3	4	5	6	7	8	9
APQoS(x)	100%	100%	100%	40%	40%	40%	20%	20%	20%
RTQoS(y)	100%	75%	50%	100%	75%	50%	100%	75%	50%
Period	$P_{\min}$	$P_{\min}$	$P_{\min}$	P_{med}	P_{med}	P_{med}	$P_{\max}$	$P_{\max}$	$P_{\max}$
Exec time	$T_{\max}$	T_{opt}	T_{avg}	$T_{\max}$	T_{opt}	T_{avg}	$T_{\max}$	T_{opt}	T_{avg}

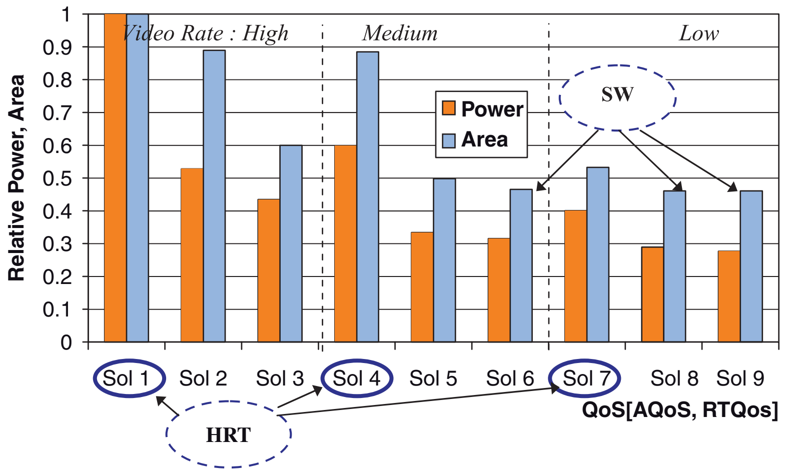


Fig. 10. QoS/power/area tradeoffs.

Table 2
QoS tasks configuration

Task_i	T_1	T_3	T_5	T_7	T_9	T_{11}
TQoS_i[AQoS,RTQoS]	[1, 0.6]	[1, 0.8]	[1, 0.9]	[1, 0.7]	[1, 0.65]	[1, 0.9]

for tasks like filtering interpolation or object positioning, the gap is much more important (e.g. from 90 to 450 ms for T_5). Regarding the RTQoS, we usually consider a Gaussian $G(T_{\text{avg}}, s)$ RL−Execution−Time distribution for each of these tasks as presented in Fig. 9. Four particular values can be distinguished: $T_{\min}$, T_{opt}, T_{avg}, and WCET such as

- Probability (RL−Execution−Time$<T_{\text{avg}}$) = 0, 50
- Probability (RL−Execution−Time$<T_{\text{opt}}$) = 0, 75
- Probability (RL−Execution−Time$<$WCET) = 1

Various QoS tradeoffs have been evaluated with our codesign framework. The different solutions are detailed in Table 1. The case 1, 4 and 7 correspond to hard real time conditions (RTQoS = 1) with three different video data rates (AQoS = 1; 0.4; 0.2).

The results are presented in Fig. 10 with a cost function tuned with $\alpha = 0.3$ (area) and $\beta = 0.7$ (power). We present relative values to show out the influence of QoS choices. Thus we observe in Fig. 10 that the power and the area costs can be efficiently reduced when the QoS constraints are relaxed. For instance, by reducing the video data rate, we observe that 40% of power reduction can be obtained for a medium quality. Another point is the cost of hard RT (HRT), actually if a soft RT (SRT) is used and tune to 75% of the WCET we note that meaningful power and area savings are achieved.

Suppose that the designer wants to optimize the power consumption, so he configures the QoS cheker for AQoS and RTQoS dimensions, respectively, as follows:

{(Test1,Rule1); (Test2,Rule3); (Test3,Rule3)}, {(Test1,Rule3); (Test2,Rule1); (Test3,Rule1)}. The QoS configurations for football player robot application tasks are presented in Table 2. We assume that the QoS array is set to [1,1] for all remaining tasks. The server task delay is set to 0.017 ms. However, the goal here is mainly the illustration of the feedback analysis. The solution obtained following the partitioning/scheduling step shows that tasks $T_1, T_3, T_5, T_7, T_9, T_{15}, T_{16}, T_{19}$ are assigned to the processor. For this solution, eight communication tasks are created and also assigned to the processor. The rate of success of the server task capacity to support the probably deadlines violations, so-called SSR generated for this solution, equals to 0.97.

8. Conclusion

The design space related to embedded systems is extremely large, it involves functional specification decisions, implementations choices including SW, HW with various granularities and coprocessors choices and also low level Clock frequency-V_{dd} couple alternatives, moreover it requires a complex real-time analysis. A tool is required to handle the problem complexity but this tool must be controllable by the designer in an interactive way. In this paper, a HW/SW co-synthesis framework is proposed for multitask RT embedded systems. The proposed iterative co-synthesis procedure with user interaction consists of three main steps: selection of QoS choices, HW/SW partitioning and schedulability test. Unlike current methodologies for RT SoC that are based on worst case design approach, and when dealing with QoS driven applications, a probabilistic approch for schedulability analysis of fixed priority driven preemptive SW system tasks with uncertain computation time is proposed. We have shown that the proposed approach for QoS management performs well with nine QoS versions. It can lead to 40% of power reduction by reducing the video data rate. The proposed approach for static QoS management can be used as a starting point for the development of a dynamic QoS manager which is the subject of our future work.

References

[1] M.J. Bass, M. Christensen, Customization and speed-to-market will drive the industry from the bottom up, IEEE Spectrum 39 (4).
[2] The soclib project, ⟨http://soclib.lip6.fr/⟩ (2004).
[3] Cadence virtual component co-design (vcc), ⟨http://www.cadence.com⟩ (2004).
[4] P. Eles, Z. Peng, K. Kuchcinski, A. Doboli, System level hardware/software partitioning based on simulated annealing and tabu search, Kluwer Journal on Design Automation for Embedded Systems 2 (1) (1997) 5–32.
[5] R. Gupta, Co-Synthesis of Hardware and Software for Digital Embedded Systems, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.
[6] T.Y. Yen, W. Wolf, HardwareSoftware Co-Synthesis of Distributed Embedded System, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.
[7] J. Hou, W. Wolf, Process partitioning for distributed embedded systems, in: Fourth International Workshop on H/S Codesign, 1996.
[8] A. Burns, G. Bernat, I. Broster, A probabilistic framework for schedulability analysis, in: Third International Workshop on Embedded Software, 2003.
[9] S. Monolache, P. Eles, Z. Peng, Memory and time efficient schedulability analysi of task sets with stochastic execution times, in: 13th Euromicro Conference on Real-Time Systems, 2001.
[10] A. Leulseged, N. Nissanke, Probabilistic analysis of multiprocessor scheduling of tasks with uncertain parameters, in: Ninth International Conference on Real-Time Embedded Computing Systems and Application, 2003.
[11] A. Atlas, A. Bestravros, Statistical rate mnotonic scheduling, in: 19th IEEE Real-Time Systems Symposium, 1998.

[12] I. Broster, A. Burns, Random arrivals in fixed priority analysis, in: First International on Probabilistic Analysis Techniques for Real-time Embedded Systems, 2004.
[13] L. David, I. Puaut, Statistic determination of probabilistic execution times, in: 16th Euromicro Conference on Real-Time Systems, 2004.
[14] ⟨http://www.artistembedded.org/Overview/⟩, 1ST ARTIST (2002).
[15] G. Micheli, R. Ernst, W. Wolf, Readings in hardware/software codesign, Morgan Kaufman Publishers, 2004.
[16] P. Dave, N.K. Jha, CASPER: Concurrent hardware-software co-synthesis of hard real-time aperiodic specification of embedded system architectures, in: Design, Automation & Test in Europe Conf, Paris, France, 1998.
[17] B.P. Dave, G. Lakshminarayana, N.K. Jha, COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems, IEEE Trans. VLSI Sys. 7 (1) (1999) 92–104.
[18] B. Li, K. Nahrstedt, A control-based middleware framework for quality of service adaptation, IEEE J. Select. Area Commun, September 1999.
[19] C. Lu, J. Stankovic, G. Tao, S. Son, Feedback control real-time scheduling: framework, modeling and algorithm, J. Control-Theor. Approach. Real-Time Comput. (special issue of RT Systems) 23 (1/2) (2002) 85–126.
[20] B. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, K.R. Walker, Agile application-aware adaptation for mobility, in: 16 ACM Symposium on Operating Systems Principles, 1997.
[21] J.L. Wong, G. Qu, M. Potkonjak, An on-line approach for power minimization in QoS sensitive systems, in: ASP-DAC, 2003.
[22] P. Marti, J. Fuertes, G. Foliler, K. Ramamritham, Jitter compensation for real-time control systems, in: IEEE Real-Time Systems Symposium, London, UK, 2001.
[23] A. Dasdan, Timing analysis of embedded real-time systems, Ph.D. Thesis, University of Illinois, Urbana-Champaign, USA, November 1999.
[24] A. Azzedine, J.-Ph. Diguet, J.-L. Philippe, Large exploration for hw/sw partitioning of multirate and aperiodic real-time systems, in: 10th International Symposium on H/S Codesign, Estes Park, USA, 2002.
[25] M. Joseph, P. Pandya, Finding response time in a real-time system, IEEE Design Test Comput. 29 (5) (1986) 390–395.
[26] J.A. Butts, G. Sohi, A static power model for architects, in: 33rd AcM/lEEEInt. Symposium on Microarchitecture, 2000.
[27] S.I. Association, International technology roadmap for semiconductors, ⟨http://public.itrs.net/Files/2003ITRS/Home2003.litm⟩ 2003.
[28] Y. Moullec, N. Amor, J.-Ph. Diguet, P. Koch, Follow-up Modelling for Wireless Personal Communication Systems, in: Seventh International Symposium on Wireless Personal Multimedia Communications, Abano, Italy, 2004.
[29] S. Bilavarn, G. Gogniat, J.-L. Philippe, L. Bossuet, Fast prototyping of reconfigurable architectures from a c program, in: IEEE ISCAS, Bangkok, 2003.
[30] J. Laurent, N. Julien, E. Senn, E. Martin, Power consumption modeling and characterization of the ti c6201, IEEE Micro 23(5).