RTDT : a Static QoS Manager, RT Scheduling, HW/SW Partitioning CAD Tool

Hedi Tmar, Jean-Philippe Diguet, Abedenour Azzedine, Jean-Luc Philippe, Mohamed Abid

RTDT : a Static QoS Manager, RT Scheduling, HW/SW Partitioning CAD Tool

H.Tmar*+, J-Ph.Diguet*, A.Azzedine*, M.Abid+, J-L.Philippe*

* *LESTER Lab, University of South Britanny, France, Lorient*
+ *CES Lab, National Engineers school of Sfax, Tunisia, Sfax*

Abstract

The Hardware (HW)/Software (SW) 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 a 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.

Key words: RT system, Power model, RT scheduling, Static QoS manager, SoC.

1 Introduction

The domain of CAD tools is now crucial for System On a 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 longer the amount of transistors available on a chips but rather the way to follow up the potential they offer with reduced design delays and cost methodologies. The questions related to the the complexity of SOC are manyfold and include different issues like reliability, design delay, power and real time constraints. As previously performed

Email address: tmar@iuplo.univ-ubs.fr, jean-philippe.diguet@univ-ubs.fr, azzedine@iuplo.univ-ubs.fr, mohamed.abid@enis.rnu.tn, jean-luc.philippe@univ-ubs.fr (H.Tmar*+, J-Ph.Diguet*, A.Azzedine*, M.Abid+, J-L.Philippe*).

in other industrial domains like avionics and automobiles, the microprocessor industry is evolving towards unavoidable knowledge management methods in order to reduce design cost and delay while focusing on a few real value-added innovations [1]. In the domain of SOC, the designers are relying on reuse of reconfigurable HW or SW intellectual property blocks (IP). IP based framework for simulations is stilling 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. Our objective is to provide such a tool in the context of low power real time embedded systems.

This kind of systems is strongly used in several fields like industries, telecommunications, and avionics. These complex systems are typically reactive and real time. Their design requires a high level design tool in order to rapidly select and synthesize promising architectures. Due to a hard real time constraints, hardware implementation (e.g. IP based) of critical functions must be performed. It is then necessary to use a software/hardware codesign approach which must allow a minimal design cost and a minimum time to market. Such a systems are increasingly control-dominated and data-dependent for optimization purposes. It means that a dynamic scheduling should be theoretically used in order to cope with the load variability. However, this kind of scheduling techniques has drawbacks incompatible with embedded systems; actually dynamic scheduling can not guarantee real time constraints and requires a complex implementation. For these reasons, embedded real time operating systems (RTOS) use fixed scheduling policies that consist in a priori time slot reservation for each task. However, worst case execution time (WCET) of tasks must be considered to guarantee real time constraints. So in case of very variable execution delay, it can lead to a fall of the system performances by means of architecture over-sizing. Thus the solution which appears is no longer a real time management but rather a QoS management [4]. 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 tasks for periodic and aperiodic "soft RT" tasks. This kind of tasks respects the RT constraints with a given probability. Our main goal consists in avoiding worst case assumptions and deterministic guarantees for periodic and aperiodic tasks with real time constraints by means of probabilistic scheduling.

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

2 Problem specification

In this section, we present our global approach suite for SoC design and we place our work within the state of the art.

2.1 The proposed approach

Globally we address the partitioning of a dependent task graph over a multiprocessor architecture in a real time context. Solving this question directly is extremely complex and heuristics can offer 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 [5]. Given that fact, we have split the problem and defined a 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 composed by a Processor, Accelerators, Coprocessors and Memories (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 No real time. It uses the Radha Rathan Tool [6] to get the time constraint interval for each task;
- *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 can not be included in the partitionning/scheduling loop but can be efficiently performed over a small set of solutions.

This paper specifically addresses the second and third steps.

2.2 Related works

Our research results can be viewed in the context of two areas of related works : high level Hw/Sw partioning-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 [7]. 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 [8]. It includes multi-granularity selection in the context of performance optimization. In the context of real time scheduling, static non preemptive scheduling [9,10] is usually adopted in embedded real-time systems since dynamic scheduling can not guarantee the real time constraints and incurs a computational overhead. The objectives formulated in [11,12] are quite close from ours; the multi-rate issue is handled and preemptive scheduling is considered. It also includes the RTOS overhead but doesn't take into account preemptions due to the access to a 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 whose can be found in the domain of PC-based servers for video tracking [13] or web applications [14], propose an interesting close-loop approach for QoS and CPU Bandwidth adaptation. In the domain of mobile system, Agile [15] propose 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 [16]. The authors study how multiple voltages can be used to simultaneously satisfy hardware 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 hardware resource and coprocessor sharing between tasks.

3 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 HW/SW communication modeling.

3.1 PACM Architecture

The figure 1 presents the PACM model. Basically our architecture is built around a processor core (e.g. IP Nios), which offers a 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) software ii) software with coprocessors that can be shared with other tasks and iii) dedicated hardware.

3.2 HW/SW communications

3.2.1 General case

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

- If T_P and T_C are software, there is no need for communication task neither additional memory;
- If T_P and T_C are hardware there's no need for communication task, however a communication memory (output for T_P and input for T_C) is added;

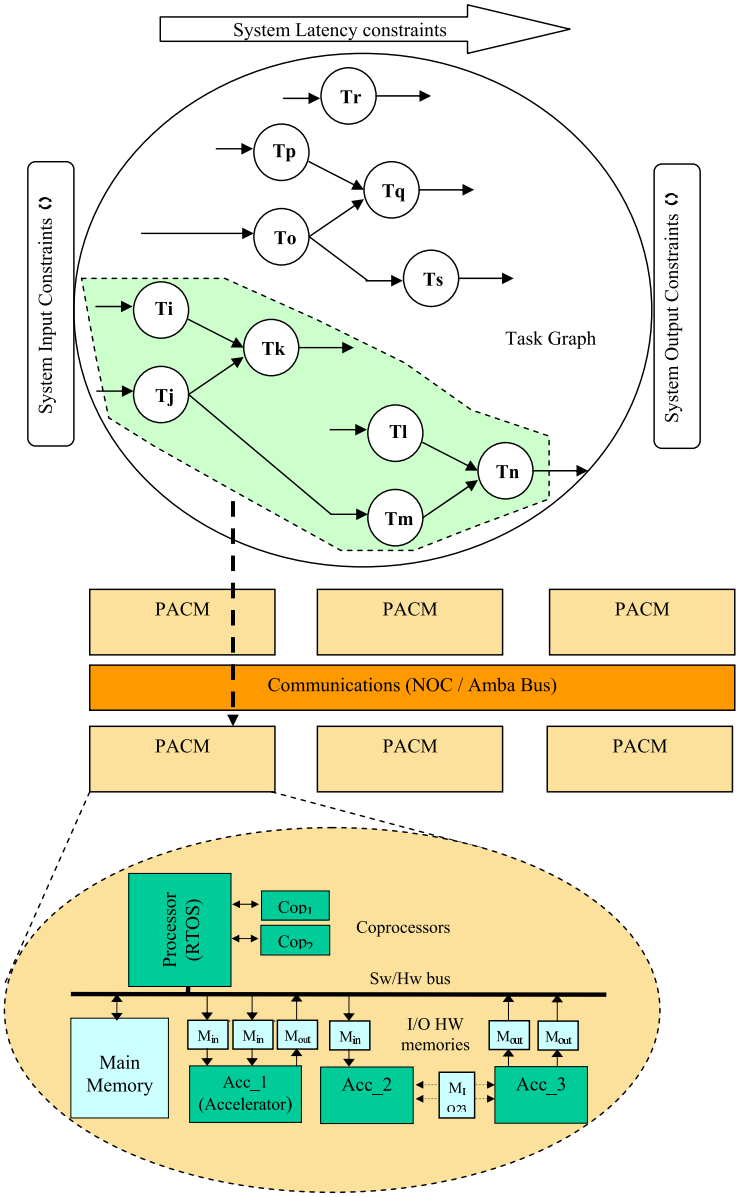
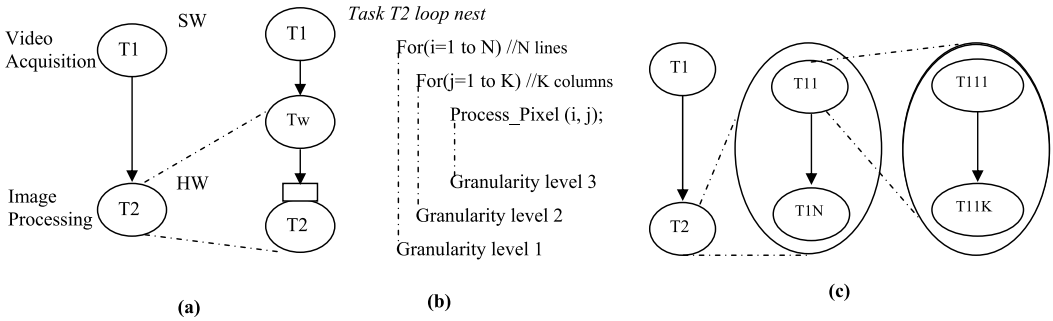


Fig. 1. Task graph assignment to the target architecture

- If T_P is software and T_C is hardware, a new communication task is created, this task writes the data produced by T_P in a new input hardware memory;
- If T_P is hardware and T_C is software, a new communication task is created, this task reads the data produced by T_P in a new output hardware memory.

3.2.2 Communication task features

The first point is the communication task period value and the amount of data to transmit. The period of the communication task T_{com} to be added 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



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

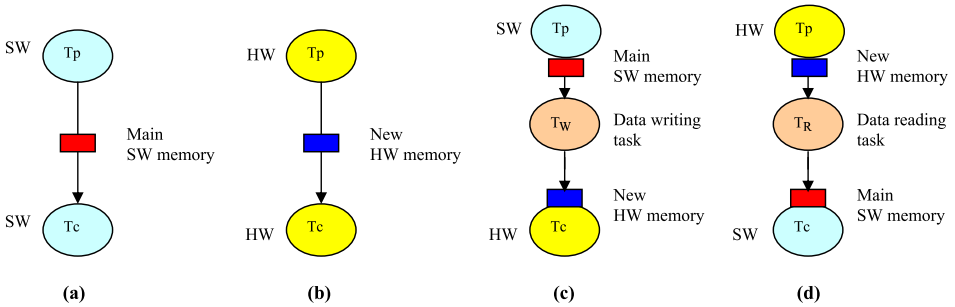


Fig. 3. Memory and Tasks implementations for HW/SW communications

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 follow :

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} the amount of data to transmit,N_{cycles_com} the number of cycle par data transfer. All these parameters are generic and can be easily specified.

3.2.3 Memory implementation

The task schedulability is computed while considering independent tasks; actually the question of task dependencies is solved by shifting release times of consumer tasks[17]. 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 point is solved by the RTOS when tasks are software, but if one of the tasks is hardware then conflicts can occur. Our framework proposes two solutions to address this point. The first one is the implementation of a critical resource with priority ceiling. The second one is the data storage from successive iterations.

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 $P_P \geq P_C$ ($P_P = n * P_C$ and $n \geq 1$)

$$MS = 2 * NDataout\_P = 2 * n * NDataIn\_C \tag{3}$$

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

$$MS = 2 * n * NDataout\_P = 2 * NDataIn\_C \tag{4}$$

3.2.4 Multiple inputs/outputs

Including memory reusing optimization during the partitioning / scheduling process incurs a unacceptable complexity gap. Moreover, memory optimization can efficiently be performed afterwards over a small set of promising solutions. So, in case of multiple dependencies, (1 producer with several consumers or 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.

4 Design space exploration & 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.

4.1 RT scheduling strategy

4.1.1 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 HPF (High priority first) 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 [18].

In a first approach we consider two kinds of tasks (we will see in section 5 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 real time constraints (RTC), and sporadic tasks with hard RTC. Like in [11], we consider the sporadic tasks as a periodic tasks with a period equals to the minimum delay between two subsequent executions; this value is provided by the Radha Ratan tool. 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 task priority is computed as the inverse of the task period. The question of multi-rate dependencies is solved by shifting the release time computation as detailed in [17].

4.1.2 Response time computation

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

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

Where :

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

The context switching overhead is the delay between the preemption of a given task and the activation of another task. The difficulty is that C_{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 coprocessors. The influence of the number of tasks is not insignificant but can be neglected compared with 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 the coprocessor is obviously related to the number of data and status registers.

4.2 Design space exploration for HW/SW partioning

4.2.1 Cost function

The cost function takes into 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. In order to eliminate units, relative costs are used to evaluate the cost value for a given schedulable solution S:

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

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 α parameter also act on the power optimization.

4.2.2 Area Cost

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

4.2.3 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 [19], 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 used 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 :

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

where : Pw_d the average dynamic power dissipated during a hyperperiod T_G and Pw_s the average static power.

4.2.4 Dynamic power/energy metric

Let Pw_d the average dynamic power dissipated during a hyperperiod T_G.

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

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

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

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

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

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

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

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 $W/\mu m^2$ as indicated in [20]).

4.2.5 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 :

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

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

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

Without HW dynamic power supply management, we obtain :

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

4.2.6 Partitioning Algorithm

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 hardware 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.4) in 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 rs to be guarantied for the server task.

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 software solution and progresses towards a complete hardware

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

Fig. 4. Schedulability test

solutions 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 Figure 4. 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 hardware 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 lower priority must be computed again. The B&B is efficient even for large graphs (100 tasks) when there are a few schedulable solutions, but it's 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.

4.3 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 Figure 5.

We have opted for the task graph defined in [6] 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 $[PminPmax]$ for each system task.

Each Task is described in a C code file, the Design Trotter framework [21] first

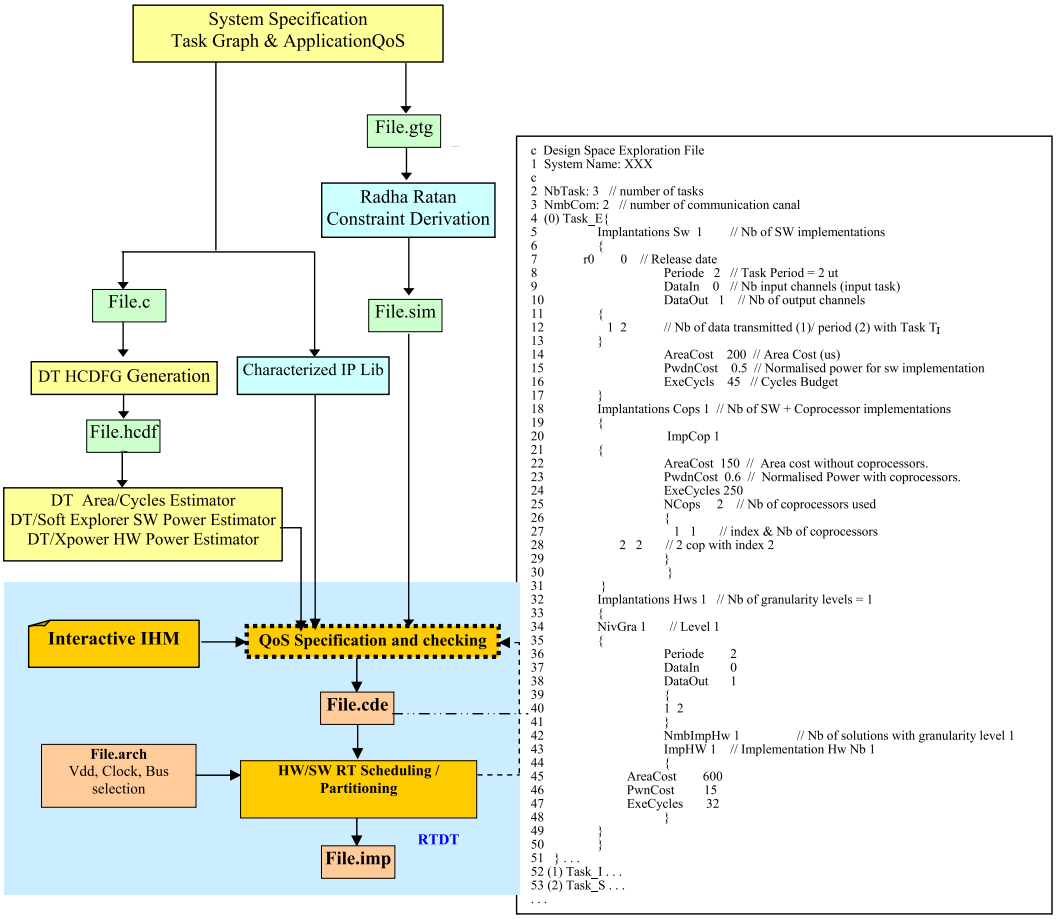


Fig. 5. RTDT Codesign Flow

generates a hierarchical data flow graph (HDFG) from which different kind of estimation can be produced like delay / area of FPGA HW components [22] or power software estimation by hierarchically combining of C/Data Flow Graph from [23]. 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.5) includes the task constraints selected and the description of all task implementations. The "file.Arch" gives the architectural parameters, like the Vdd/clock modes V_{dd}, f for hardware and software parts, the bus protocol and so on.

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

5 Static QoS manager

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

5.1 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.

5.2 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 real time constraints with a given probability. the aim is to avoid worst case assumptions and deterministic guarantees for periodic and aperiodic tasks with soft eal time constraints by means of probabilistic scheduling. Each task can be represented by a QoS array of N parameters:

$$TQoS_i[x_1, ..., x_N]$$

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

$$TQoS(i)[AQoS, RTQoS]$$

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 real time constraints 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:

$$Probability(RL - Execution - Time(i) = W(i)) = RTQoS(i)$$

Where : $RL - Execution - Time$ means the real life execution time of task i. Regarding the $RTQoS$ dimension, the designer must choose the minimun

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 exist between task application qualities according to these choices. Regarding the real time issue, a task with a $RTQoS$ equals to 1 ($W(i) = WCET(i)$) should not be delayed by another to miss its deadline.

5.3 QoS decision & generation step

In this subsection, we detail the QoS specification and checking step that is shown in figure 6. 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.5) includes the tasks constraints selected and the description of all task implementations.

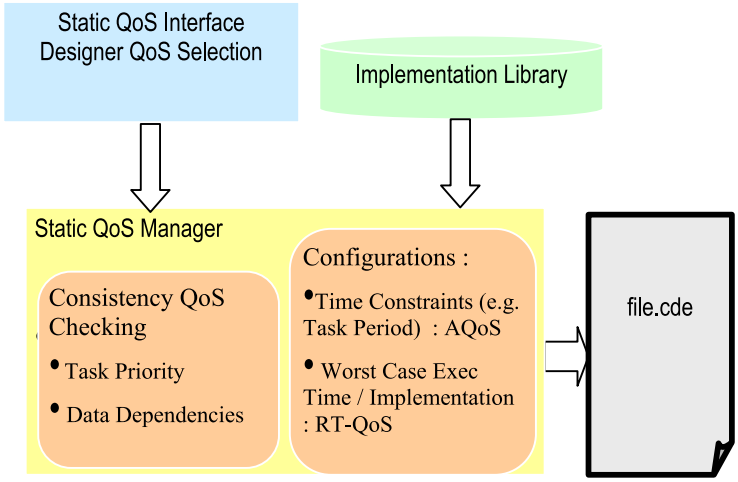


Fig. 6. QoS specification and checking

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

- T1 Data dependency
- T2 Resource sharing
- T3 Task priority

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

- R1 QoS Round down
- R2 QoS Round up
- R3 QoS Unchecked

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 T1 the designer will select R1, or R2 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 : {(T1,R1) ; (T2,R3) ; (T3,R3)}

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 4.1. Thus, the QoS checker must perform the T2 and T3 tests with rules R1 or R2 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 rule R1 or R2 must be performed. For example, if power optimization is favoured, the QoS checker is configured as follows for dimension $RTQoS$: {(T1,R3) ; (T2,R1) ; (T3,R1)}. The QoS aware codesign flow is illustrated in section 6.

5.4 Feedback scheduling analysis

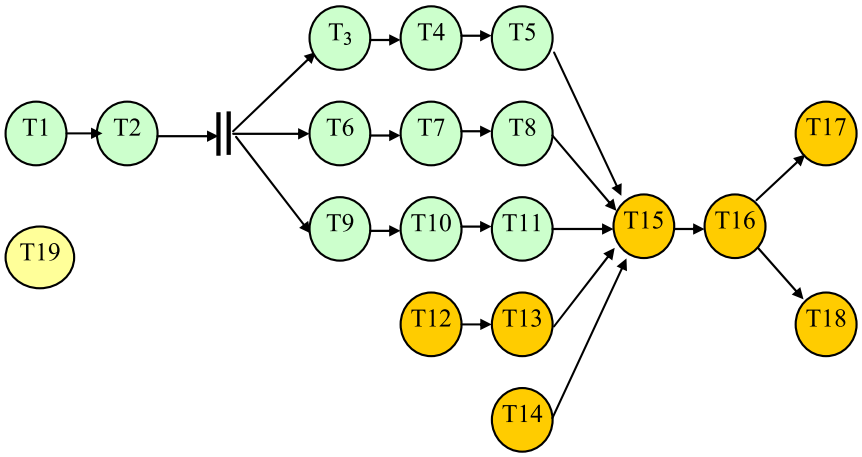
After the HW/SW RT partitioning/scheduling step, a timing analysis report is returned to the designer. This report contains the probability scheduling results. it indicates the scheduling safety rate (SSR), namely, the rate of success of the server task capacity to support the probably deadlines violations,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

$$Probability(\sum_{i \in SW} RL - Execution - Time(i) \leq T_s + \sum_{i \in SW} W(i)) = SSR$$

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 task$i$, as mentioned above.

6 Case study : a football player robot application

The case study described in Figure 7 is a football player robot application with video tasks for object detection, HF 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_{19} is the server task with the lowest priority; it includes all aperiodic software tasks with soft real time constraints or without real time constraints.



	Title	Period (ms) (min, med, max)	Data In (32 bits)	Data Out
T_{12}	HF 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}	HF Emission	20, 50, 100	20	----
T_{18}	20 Motor Command	20, 50, 100	20	----
T_{19}	Server Task	200	40	----

Fig. 7. Football player robot application

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

$$\forall i \in \{12, ..., 18\}, TQoS_i = [1, 1]$$

For tasks from 1 to 11, different tradeoffs are experimented. Three video rates are considered: 40ms (High : Pmin), 100ms (Medium : Pmedium) or 200ms (Low : Pmax).

The software RL-Execution-Time is obviously different for all these tasks. it also vary within the interval [Tmin, WCET]. For instance, the video acquisition and the Bayer interpolation delay variation are reasonably limited, but for tasks like filtering interpolation or object positioning, the gap is much more important (e.g. from 90 to 450ms for T_5). Regarding the $RTQoS$, we usually consider a Gaussian $G(T_{avg}, s)$ $RL - Execution - Time$ distribution for each of these tasks as presented in Figure 8. Four particular values can be distinguished : Tmin, Topt, Tavg, and WCET such as:

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

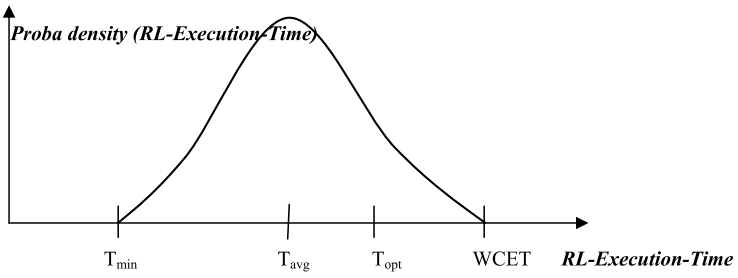


Fig. 8. RL-Execution-Time distribution

Various QoS tradeoffs have been evaluated with our codesign framework. The different solutions are detailed in Figure 9. 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 Figure 10 with a cost function tuned with α= 0.3 (area) and β=0.7 (power). We present relative values to show out the influence of QoS choices. Thus we observe in Figure 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.

Sol	1	2	3	4	5	6	7	8	9
AQoS(x)	100%	100%	100%	40%	40%	40%	20%	20%	20%
RTQoS(y)	100%	75%	50%	100%	75%	50%	100%	75%	50%
Period	Pmin	Pmin	Pmin	Pmedium	Pmedium	Pmedium	Pmax	Pmax	Pmax
Execution time	Tmax	Topt	Tavg	Tmax	Topt	Tavg	Tmax	Topt	Tavg

Fig. 9. TQoS[$AQoS$,$RTQoS$] tradeoffs

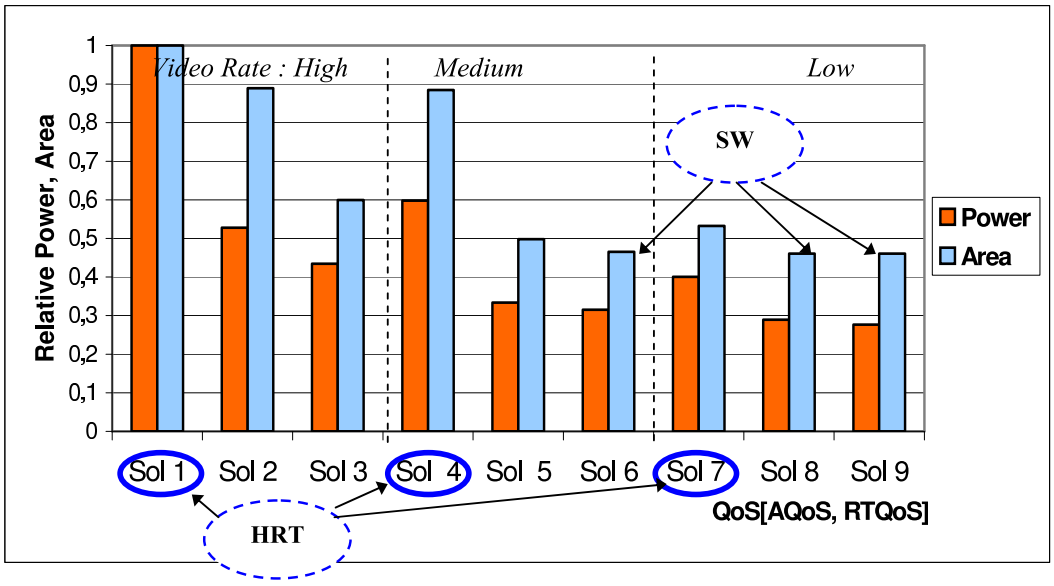


Fig. 10. QoS / Power / Area Tradeoffs

7 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-Vdd 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 steps : selection of QoS choices, HW/SW partitioning and schedulability test. Unlike the previous approach, we take into account the resources and the coprocessors sharing between tasks. 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] http://www.artist embedded.org/Overview/, IST ARTIST (2002).

[5] P.Marti, J. Fuertes, G. Fohler, K. Ramamritham, Jitter compensation for real-time control systems, in: IEEE Real-Time Systems Symposium, London, UK, 2001.

[6] A.Dasdan, Timing analysis of embedded real-time systems, Ph.D. thesis, University of Illinois, Urbana-Champaign, USA (Nov. 1999).

[7] G.Micheli, R.Ernst, W.Wolf, Readings in hardware/software codesign, Morgan Kaufman Publishers.

[8] 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.

[9] R.Gupta, Co-Synthesis of Hardware and Software for Digital Embedded Systems, Kluwer Academic Publishers, 1995.

[10] T.Y.Yen, W.Wolf, Hardware-Software Co-Synthesis of Distributed Embedded System., Kluwer Academic Publishers, 1997.

[11] 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.

[12] B.P.Dave, G.Lakshminarayana, N.K.Jha, COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems, IEEE Trans. on Software Engineering 7 (1).

[13] B.Li, K.Nahrstedt, A control-based middleware framework for quality of service adaptation, IEEE Journal on Selected Areas in Communication.

[14] C.Lu, J.Stankovic, G.Tao, S.Son, Feedback control real-time scheduling: Framework, modeling and algorithm, special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing 23 (1/2) (2002) 85–126.

[15] B.Noble, M.Satyanarayanan, D.Narayanan, J.E.Tilton, J.Flinn, K.R.Walker, Agile application-aware adaptation for mobility, in: 16^{th} ACM Symp.on Operating Systems Principles, 1997.

[16] J.L.Wong, G.Qu, M.Potkonjak, An on-line approach for power minimization in qos sensitive systems, in: ASP-DAC, 2003.

[17] A.Azzedine, J-Ph.Diguet, J-L.Philippe, Large exploration for hw/sw partitioning of multirate and aperiodic real-time systems, in: 10^{th} Int. Symp. on H/S Codesign, Estes Park, USA, 2002.

[18] M.Joseph, P.Pandya, Finding response time in a real-time system, IEEE Design and Test of Computers 29 (5) (1986) 390–395.

[19] J.A.Butts, G.Sohi, A static power model for architects, in: 33^{rd} ACM/IEEEInt. Symp. on Microarchitecture, 2000.

[20] S. I. Association, International technology roadmap for semiconductors, http://public.itrs.net/Files/2003ITRS/Home2003.htm (2003).

[21] Y. Moullec, N. Amor, J-Ph.Diguet, P.Koch, Follow-up Modelling for Wireless Personal Communication Systems, in: 7^{th} Int. Symp. on Wireless Personal Multimedia Communications, Abano, Italy, 2004.

[22] S.Bilavarn, G.Gogniat, J-L.Philippe, L.Bossuet, Fast prototyping of reconfigurable architectures from a c program, in: IEEE ISCAS, Bangkok, 2003.

[23] J.Laurent, N.Julien, E.Senn, E.Martin, Power consumption modeling and characterization of the ti c6201, IEEE Micro 23 (5).