Schedulability Analysis of Heterogeneous Systems for Performance Message Sequence Chart

Frank Slomka, Jürgen Zant, Lennard Lambert
University of Erlangen-Nuremberg
Department of Computer Architecture and Performance Evaluation,
Martensstr. 3, 91058 Erlangen, Germany
{slomka, lambert}@informatik.uni-erlangen.de

Abstract

Telecommunication systems are often specified in the standardized languages SDL and MSC. These languages allow only the specification of pure functional aspects. To remedy this problem we have combined the language MSC and performance aspects in Performance MSC (PMSC). From a PMSC specification a task model can be derived that includes beside computation times, periods and deadlines of tasks, also absolute start times of tasks and dependencies between tasks. This allows us to apply an extended schedulability analysis of asynchronous tasks on heterogeneous target architectures. We present the analysis technique and demonstrate with a small example, how the algorithm can be used for the real-time analysis of a cordless telephone.

1 Introduction

In the telecommunication area the specification languages SDL (Specification and Description Language, [9]) and MSC (Message Sequence Chart, [10]) are widely used. The main application of the languages is the development, formal specification, and validation of telecommunication protocols. In MSC typically use-cases are formalized while in SDL the full functional behavior is described by extended finite state machines. With this application domain in mind it is comprehensible that the main focus of SDL is not the design of real-time systems.

However, SDL and MSC are also used for the specification and the design of real-time systems [3]. Using SDL for the design of real-time systems exposes the disadvantages of SDL and MSC for applications like motor-controlling or other embedded systems. For examples SDL or MSC are defined without constructs to describe timing constraints or implementation aspects. To use SDL and MSC as languages for hardware/software codesign, we extended the language SDL with constructs to support the specification of timing, cost and resource constraints [14]. MSC is also extended by extensions to specify performance requirements [6]. The extensions to the standard languages are called SDL* and PMSC (fig. 1). With SDL* and PMSC a complete implementation design specification for mixed hardware/software systems can be defined [12].

A set of tools is being developed in Erlangen to support the automatic hardware and software implementation of SDL/MSC specified systems. To gain reasonable price of the system the configuration, e.g. the mapping

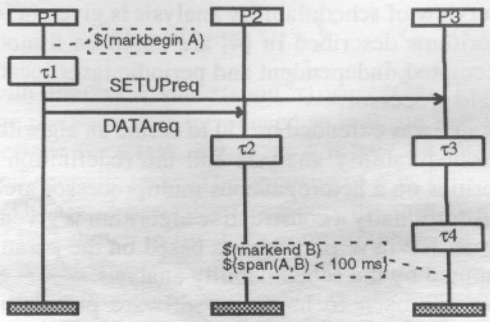

Figure 1: Example of a PMSC specification

of functional system modules, like tasks or processes, to architecture components (processors, ASICs etc.) and the scheduling strategy, is optimized at compile-time. Due to the huge state space of the system, heuristic optimization algorithms must be applied and the user can prescribe some optimization decisions manually. Thus while the timing constraints are fulfilled the total system costs are minimized. The basis of the optimization process is a model-based performance evaluator based on a PMSC specification.

PMSC follows the 3-phase-methodology [8] where the model of the system is divided into two submodels, namely the load and the machine model. The load model in PMSC describes how the load of the system is structured by an MSC, where all tasks (see fig. 1: τ_1, τ_2, τ_3, τ_4) define some load for the machines. Moreover the intensity of the load is defined by the absolute start times of an MSC, when all head tasks are enabled to start (fig. 1, τ_1). The load model of a PMSC specification can be transformed into its corresponding task graph [11]. The machine model describes the performance of the machines, their topology and the scheduling algorithms

This research has been funded by the Deutsche Forschungsgemeinschaft (DFG) under grant He 1408/4-1 and grant SFB182, B3

used. When mapping load components on machine components and giving the load components a priority, the full system model is obtained.

In the paper we discuss an algorithm for the schedulability analysis of heterogeneous systems with depencies between tasks. The algorithm calculates the worst case response times of each instance of a system task without a simulation of a complete system trace. With this technique it is possible to analyze the response times of sporadic and periodic tasks. The goal is to use this task model for an automatic derivation from a PMSC specification.

The paper is organized as follows: the next section describes the related work. In section 3 we discuss the basic ideas from literature for a codesign approach with schedulability analysis for periodic tasks. An introduction to the extended schedulability analysis is given in section 4. An example how the analysis algorithm works is given in section 5. Finally we will conclude the paper.

2 Related Work

An overview of schedulability analysis is given in [4]. The algorithms described in [4] are based on a model with preempted, independent and periodic tasks located on a single processor.

The model was extended by [1] to define an algorithm for the schedulability analysis and the redefinition of task priorities on a heterogeneous multiprocessor architecture. Additionally a constructive algorithm is given to calculate an HW/SW partitioning based on the parameters computed by the schedulability analysis.

Another approach to hardware/software partitioning based on schedulability analysis with periodic and independent tasks is described in [13]. An extension is given to include communication times to schedulability analysis, and an algorithm for HW/SW partitioning is presented.

An idea for codesign with periodic tasks or processes is found in [2]. The algorithm extends data-dominated codesign approaches by a reactive process model to support the design of heterogeneous systems.

A real-time verification based on schedulability analysis for aperiodic tasks is described in [7]. This approach calculates, whether all tasks in the model guarantee their deadlines. In contrast to [1] and [13] the algorithm only finds deadline violations and does not calculate any metric values for the next step of the optimizer. Another disadvantage to use this algorithm for HW/SW codesign, is that the algorithm only supports the verification of message-coupled systems.

3 Schedulability Analysis

Functional correctness of a real time system depends on the correctness of the implementation and on the fulfillment of the system's timing constraints. An important factor of the design of real-time systems is to find the optimal priority assignment of all tasks with the lowest implementation costs. The optimal priority assignment results in a system that holds its deadlines on the slowest possible machine. In a heterogeneous environment the relative speed of the components is not changed.

3.1 Response Time Analysis

The response time analysis is based on a simple process model (see [4]). Each task (τ) of the set of tasks (B), is completely independent and all tasks are periodic, with known periods (T). Furthermore each task has a given worst-case execution time (C) and a deadline (D) when the execution of the task has to end. So each task τ_i is characterized by a tuple (C_i, T_i, D_i, p_i) where C_i, T_i, $D_i \in \mathbb{R}^+$ are given in time units, and where $p_i \in \mathbb{N}$ is defined as the priority of the task τ_i. The priorities of the tasks are directly proportional to the priority number p. For each of these tupel we can define an abbreviation $X(\tau_i) = X_i$, $X \in \{C, T, D, p\}$. This means for example that $C(\tau_1) = C_1$ and $D(\tau_2) = D_2$.

Usually the priority of each τ is given by the deadline monotonic priority assignment algorithm which is optimal on an one-processor system. Deadline-monotonic means that the priority p is direct proportional to the reciprocal of the deadline D. Therefore the task with the shortest deadline will be scheduled with the highest priority. Defining a set of higher prior tasks $H_p(\tau) = \{\tau' \in B \mid p(\tau') > p(\tau)\}$ the response time of a task is

$$R(\tau) = \min \{t > 0 \mid \frac{R(\tau, t)}{t} = 1\} \qquad (1)$$

(see [1]), with

$$R(\tau, t) = C(\tau) + \sum_{\tau' \in H_p(\tau)} \left\lceil \frac{t}{T(\tau')} \right\rceil C(\tau') \qquad (2)$$

If $R(\tau) \leq D(\tau)$ is true for all tasks, the system will satisfy the specified timing constraints.

3.2 Codesign with Schedulability Analysis

To schedule tasks in real-time multiprocessor systems, the deadline monotonic priority assignment algorithm is not optimal. An optimal priority assignment for tasks on heterogeneous architectures is given in [1]: In contrast to the approach described above the tasks τ and τ' may run on different processors using a timed shared resource, e.g a shared memory block. Under the assumption that τ' is active on the same resource as τ the interference $I(\tau, \tau')$ is an upper bound for the time interval that τ has to wait for τ'. With a set $M = \{\mu_1, ..., \mu_n\}$, where $\mu_i \in M$ are time-shared resources of the system the interference is formally defined by:

$$I(\tau, \tau') = \sum_{M' \in U(\tau)} C^{M'}(\tau') \qquad (3)$$

with $U(\tau) = \{M' \subseteq M \mid \exists M'' \subseteq M, M' \cap M'' \neq \emptyset \wedge C^{M''}(\tau) \neq 0\}$ and $C^{M'}(\tau)$ is the time when τ is using the particular combination of the resources M'. For example, if the tasks τ and τ' are running on the same processor, the interference $I(\tau, \tau') = C(\tau')$ is a special case of equation (3). Using the interference for all tasks with a higher priority

as τ during a time interval t, the worst case-response time can be calculated by:

$$R(\tau, t) = C(\tau) + \sum_{\tau' \in H_p(\tau)} \left\lceil \frac{t}{T(\tau')} \right\rceil I(\tau, \tau') \qquad (4)$$

The calculation of the response time of a task $R(\tau)$ is calculated as in equation (1). For the calculation of the priority order of the tasks on a heterogeneous architecture, the minimal required speedup (mrs) for a task τ is defined in [1]. The mrs $S(\tau)$ of a task τ describes how much faster the system has to execute such that τ has the longest response time under the assumption that it holds its deadline. Given an initial priority order for all tasks we can calculate for every task the mrs $S(\tau)$:

$$S(\tau) = min\left\{ \frac{R(\tau, t)}{t} \middle| (0 < t \le D(\tau)) \right\} \qquad (5)$$

The optimal system has the property that max $\{S(\tau) \mid \tau \in B\}$ is minimal. As shown in [1] this metric can be also used to calculate the hardware/software partitioning of the system.

4 Extensions to Schedulability Analysis

The technique discussed in section 3.2, can only use periodic and independent tasks with different priorities to analyze the priority order of a given set B. For the analysis of PMSC specifications an analysis technique is introduced that supports absolute start times of tasks and dependencies of tasks. Note that the analysis of the execution path that results if only the worst case computation times of all tasks are considered is not sufficient for heterogeneous systems. Because of this phenomenon we have to consider best case execution times.

4.1 Extended Task Model

Since the analysis has to consider all occurrences of a task, we distinguish the set of all task instances ${}^n\tau \in B^{inst}$ and the set of task definitions $\tau \in B^{task}$. Both sets are the disjoint union of sets, that result from three different types of tasks:

- Periodic Tasks ($\bar{\tau}$) with a period T and a defined starting point $s({}^0\bar{\tau})$, when the first instance ${}^0\bar{\tau}$ of $\bar{\tau}$ starts. With this we can calculate the starting point of each instance ${}^n\bar{\tau}$ of $\bar{\tau}$ by: $s({}^n\bar{\tau}) = s({}^0\bar{\tau}) + nT$ with $n \in \mathbb{N}_0$. The set of periodic tasks is $B_{\bar{\tau}}^{task}$ and the set of instances of this tasks is $B_{\bar{\tau}}^{inst}$ (${}^{\bar{\tau}}\bar{\tau} \in B_{\bar{\tau}}^{task}$, ${}^n\bar{\tau} \in B_{\bar{\tau}}^{inst}$). The following sets are defined similar.
- Triggered Tasks ($\tilde{\tau}$): tasks which are triggered by other tasks ($\tilde{\tau} \in B_{\tilde{\tau}}^{task}$, ${}^n\tilde{\tau} \in B_{\tilde{\tau}}^{inst}$).
- Sporadic Tasks ($\hat{\tau}$), which have only one defined starting point $s({}^0\hat{\tau})$, but no period ($\hat{\tau} \in B_{\hat{\tau}}^{task}$, ${}^n\hat{\tau} \in B_{\hat{\tau}}^{inst}$).

All tasks are equipped by a best case execution time C_{best} and C_{best}^M and a best case starting time s_{best} and an ending time $e(\tau) = s(\tau) + R(\tau)$. Analogous to the interference I, a lower bound I_{best} can be computed. Additionally for each task a set N, which contains all tasks triggered by τ, is defined. From N a successor relation can be derived: $succ({}^n\tau, {}^n\tau') :\Leftrightarrow \tau' \in N(\tau)$. In the approach described in this paper we do not consider multiple predecessors of a task, thus we require: $\forall\tau$: $\forall(\tau_1,\tau_2)$ succ $(\tau_1,\tau) \wedge$ succ $(\tau_2,\tau) \Rightarrow \tau_1 = \tau_2$. To have a consistent specification it is required that the transitive clousure $succ^+$ of the relation succ is acyclic. With succ* we denote the reflexive-transitive clousure of succ and succ* $({}^n\tau) = \{{}^n\tau' \mid succ^*({}^n\tau, {}^n\tau')\}$ (see fig. 2). For each instance a root instance is given by: $root({}^n\tau) = {}^n\tau'$: $succ^*({}^n\tau', {}^n\tau) \wedge \neg\exists {}^n\tau''$: succ $({}^n\tau'', {}^n\tau')$. $s_{best}({}^n\tau)$ can now be defined formally:

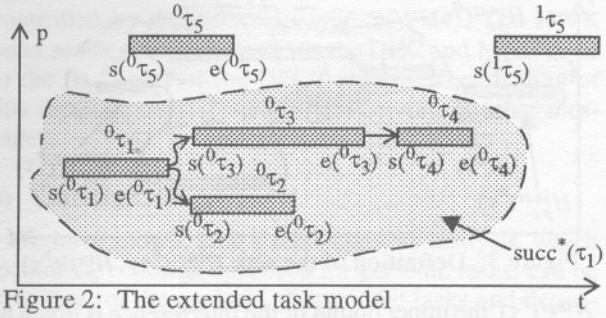

Figure 2: The extended task model

$$s_{best}({}^n\tau) = s(root({}^n\tau)) + \sum_{{}^k\tau' \in Pre({}^n\tau)} C_{best}(\tau') \qquad (6)$$

with $Pre({}^n\tau) = succ^*(root({}^n\tau))\backslash succ^*({}^n\tau)$. Furthermore the predicate $\downarrow$ is introduced to indicate that a value is defined for a task, e.g. $s({}^0\tau)\downarrow$ is true for all periodical and sporadic tasks. The opposite of $\downarrow$ is $\uparrow$.

The fig. 2 basically represents the task model of the PMSC shown in fig. 1. In comparison to fig. 1 the task τ_5 is added.

In this model communication can be modeled by communication tasks, which are a special form of triggered tasks, where the computation time depends on the systems topology and the mapping of normal tasks. It is out of the scope of this paper to add communication tasks to the model.

4.2 Analysis of the Extended Task Model

The analysis considers all occurrences of tasks in the correct timely order. If more instances start at the same absolute time, the one with the highest priority is considered first. For the task an upper bound for its execution time is calculated with a similar metric as in section 3.2. We first introduce two different sets of instances: $H_{pe}^{inst}({}^n\tau)$, $H_{pa}^{inst}({}^n\tau)$. $H_{pe}^{inst}({}^n\tau)$ contains all instances, with a higher as ${}^n\tau$ that have a starting point prior to ${}^n\tau$ and have an estimated later endpoint than the best starting point of ${}^n\tau$. $H_{pe}^{inst}({}^n\tau) = \{{}^k\tau' \in B^{inst} \mid p({}^k\tau') > p({}^n\tau) \wedge s({}^k\tau')\downarrow \wedge s({}^k\tau') \le s({}^n\tau) \wedge s_{best}({}^n\tau) < e({}^k\tau')\}$. $H_{pa}^{inst}({}^n\tau)$ contains all instances with a higher priority then ${}^n\tau$ that may start while ${}^n\tau$ is active, thus the instance ${}^k\tau$ has a worst case starting point later then ${}^n\tau$ or is a triggered instance with an unknown starting point and predecessors that allow the instance ${}^k\tau$ to start while ${}^n\tau$ is active. $H_{pa}^{inst}({}^n\tau) = \{{}^k\tau' \in B^{inst} \mid p({}^k\tau') > p({}^n\tau) \wedge ((s({}^k\tau')\downarrow \Rightarrow s({}^k\tau') > s({}^n\tau)) \wedge (s({}^k\tau')\uparrow \Rightarrow (\forall {}^k\tau'' : succ^+({}^k\tau'', {}^k\tau') \Rightarrow$

$(p(^k\tau'') > p(^n\tau) \vee I_{best}(^n\tau, ^k\tau'') = 0))))\}$. For instances $^k\tau'$ in

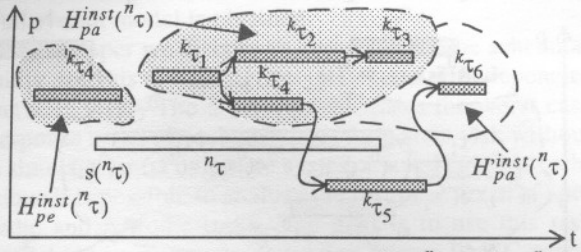

Figure 3: Definition of the sets $H_{pe}^{inst}(^n\tau)$, $H_{pa}^{inst}(^n\tau)$

$H_{pe}^{inst}(^n\tau)$ the upper bound of the interference is given by the interference of τ and τ' and the maximal overlapping of the two instances

$$\tilde{I}(^n\tau, ^k\tau') = min(I(^n\tau, ^k\tau'), e(^k\tau') - s_{best}(^n\tau)) \quad (7)$$

Both sets are shown in fig. 3, where $H_{pa}^{inst}(^n\tau)$ is valid if the best case interference of $^k\tau_5$ and $^n\tau$ do use the same resources, and $H_{pa}^{inst}(^n\tau)$ if they do not. The earliest point of preemption of the instance $^n\tau$ by another instance $^k\tau'$ after $^n\tau$ has started is given by δ:

$$\delta(^n\tau, ^k\tau') = (s_{best}(^k\tau') - s(^n\tau)) \quad (8)$$

Using the number of interrupts that may occur from $^k\tau'$ within the first t time units after $^n\tau$ has started:

$$n(^n\tau, ^k\tau', t) = max(sign(t - \delta(^n\tau, ^k\tau')), 0) \quad (9)$$

we can estimate the worst case response time of an instance $^n\tau$ which is maximally interrupted in the interval t by:

$$R(^n\tau, t) = C(^n\tau) + \sum_{^k\tau' \in H_{pe}^{inst}(^n\tau)} \tilde{I}(^n\tau, ^k\tau') + \sum_{^k\tau' \in H_{pa}^{inst}(^n\tau)} n(^n\tau, ^k\tau', t) I(^n\tau, ^k\tau') \quad (10)$$

With these equations, an algorithm for the schedulability analysis of aperiodic tasks can be formulated. First, the task with the highest priority and the earliest starting point is chosen from the set $B_a = \{^n\tau \in B^{inst} \mid s(^n\tau) \downarrow\}$. The set B_a contains all tasks with a starting point. Then the worst-case-response time of task $^n\tau$ is calculated. The starting points of all tasks $^k\tau' \in N(^n\tau)$ which are triggered by $^n\tau$ are calculated by $s(^k\tau') = e(^n\tau)$.

All tasks with starting points calculated by the equation above are now elements of the set B_a and we can calculate the worst-case-execution time of these tasks. Because in equation (10) the value t is used to determine the number of interrupts of a task, $S(^n\tau)$ can be calculate with equation (5). After calculating the worst-case-response time of an instance $^n\tau$, $^n\tau$ is eliminated from the set B_a and moved to the set B_e, which contains all tasks with calculated endpoints.

Crucial for the algorithm is the stop condition, which is calculated dynamically. Let the least common multiplier (lcm) of all periods be $T_c = lcm_{\tilde{\tau}_i \in B_{\hat{\tau}}^{inst}}(T(\tilde{\tau}_i))$ then the system must at least be analyzed till the absolute time thus all sporadic tasks are in the interval:

$$T_{must} = n_0 T_c > \max_{^i\hat{\tau} \in B_{\hat{\tau}}^{inst}} \left(s(^i\hat{\tau}) + \sum_{^i\tau \in succ^*(^i\hat{\tau})} D(^i\tau) \right) \quad (11)$$

Where $n_0 \in \mathbb{N}$ is the least smallest number that satisfies the unequation. Since from T_{must} on no sporadic tasks are in the system, the behavior is repeated every T_c time units. These cycles with period T_c may differ from each other caused by tasks that have to continue their worst case execution from earlier cycles. In the algorithm we

```
boolean test_schedulability(B) begin
  B_e := ∅; B_a := B^inst_τ̂ ; T_c= lcm(T_i); T_must = n_0 T_c;
  A_cp= ∅; B_cp := ∅;
  do forever begin // function terminates with return value
    B_a' := {^nτ ∈ B_a | ∀^kτ' ∈ B_a: s(^nτ) ≤ s(^kτ')};
    ^nτ := choose from {^nτ ∈ B_a' | ∀^kτ' ∈ B_a': p(^nτ) ≥ p(^kτ')};
    if (s(^nτ) > T_must ∧ B_cp = ∅) then
      B_cp := {^nτ ∈ B^inst | (e(^nτ)↓ ⇒ s(^nτ) < T_must < e(^nτ)) ∧
                (e(^nτ)↑ ⇒ s(root(^nτ)) < T_must)};
    R(τ) = min {t > 0 | R(τ,t)/t = 1};
    if (R(^nτ) > D(^nτ)) then return false; // system may violate D
    e(^nτ) := s(^nτ) + R(^nτ);
    B_e := B_e ∪ {^nτ}; B_a := B_a \ {^nτ};
    if (^nτ ∈ B_cp) then begin
      if (∀^kτ ∈ B_cp : e(^kτ)↓ ) then begin
        C_p(T_must) := {(τ, e(^nτ) - T_must) | ^nτ ∈ B_cp};
        if (C_p(T_must) ∈≤ A_cp) then return true;//system satisfies
        A_cp := A_cp ∪ C_p(T_must);                    //all deadlines
        T_must := T_must + T_c; B_cp := ∅;
      end if;
    end if;
    if (N(^nτ) ≠ ∅) then begin
      for each ^kτ' ∈ N(^nτ) begin
        s(^kτ') := e(^nτ);
        B_a := B_a ∪ {^kτ'};
      end for each;
    end if;
  end do forever;
end test_schedulability;
```

Figure 4: Algorithm

keep these worst case properties in the set A_{cp}. Every element in A_{cp} is a set that contains the tasks and the time intervals they overlap with the next cycle. If the algorithm has already computed a cycle with the same or stricter worst case properties the algorithm stops. If no violation of a deadline has been found, the system holds its deadlines. To compare the worst case properties of the sets in A_{cp} we define a new operator $\in_\leq : C_p \in_\leq A_{cp} :\Leftrightarrow \exists A_s \in A_{cp}: \forall x \in C_p: \exists a \in A_s: \Pi_1(x) = \Pi_1(a) \wedge \Pi_2(x) \leq \Pi_2(a)$, where Π_i is the projection on the i^{th} component of a tuple. The worst case properties of C_p are satisfied by A_{cp}, if an element in A_{cp} contains at least the same tasks as in C_p and all time intervals of the tasks in the element are greater or equal than in C_p. Since one cycle has been found with a greater interference from the previous cycle than the current cycle, the worst case response times for the cycles following the current cycle are smaller or equal then the worst case response times computed before. The algorithm is finite since no task can arbitrarily be delayed because of its deadline.

5 Example

The algorithm has been implemented in Java. In order to demonstrate how the algorithm works, we consider a cordless DECT telephone (Digital Enhanced Cordless Telephone). The DECT protocol specification [5] describes the levels one, two and three of the OSI reference

model for wireless speech and data services and supports high traffic loads. It is designed to provide large cordless PAPX installations or wireless LANs and is also available for domestic consumers.

In our example we consider a small DECT mobile part. All software-tasks of the system are running on a 8051 microcontroller (note that in this case $I(\tau, \tau') = C(\tau')$). The microcontroller is connected to an ASIC, which implements the burst mode logic for the radio interface of the telephone.

τ	p	T	D	s	C	R	S
RV	7	0.417	0.2	0.417	0.2	0.2	1
ISR1.2	6	0	0.8	0	0.4	0.4	0.5
MAC1.2	4	/	3	/0.4	1.5	1.5	0.5
ISR1.1	6	0	0.8	2.1	0.4	0.585	0.73
ISR4	6	0	0.8	2.4	0.4	0.785	0.98
MAC1.1	4	/	3	/2.69	1.5	2	0.67
APP	1	/	90	/3.19	5	29.3	0.33
DLC1.1	3	/	16	/4.69	5	7.6	0.48
SLI	5	10	1	x5	0.5	0.5	0.5
NWL1.1	2	/	50	/12.29	10	12.6	0.25
ISR1.3	6	0	0.8	10	0.4	0.6	0.75
MAC1.3	4	/	3	/10.6	1.5	1.5	0.5
ISR1.4	6	0	0.8	20	0.4	0.6	0.75
MAC1.4	4	/	3	/20.6	1.5	1.5	0.5
NWL4	2	/	50	/32.49	10	20.2	0.4
ISR2.1	6	20	0.8	c0	0.4	0.6	0.75
MAC2.1	4	/	3	/c0.6	1.5	1.5	0.5
ISR2.2	6	20	0.8	d0	0.4	0.6	0.75
MAC2.2	4	/	3	/d0.4	1.5	1.5	0.5
DLC2.2	3	/	16	/d2.1	5	5.5	0.34
DLC4	3	/	16	/52.69	5	5.5	0.34
MAC4	4	/	3	/58.19	1.5	1.5	0.5

Table 1: Tasks of a DECT telephone (all times in ms).

If the telephone receives data, the task RV (Receive), which runs on the ASIC, writes the data to the registers of the ASIC. Then the burst-mode logic interrupts the microcontroller. In dependence from the logical channel, the controller starts an interrupt service routine (ISR). The ISR has an interference I(RV, ISR) = 0.1 with the hardware task, because both tasks are using the registers of the ASIC. The ISR copies the data from the registers of the ASIC to the microcontroller's memory. In table 1 it is shown how an analysis of the receipt of a network layer (NWK) message can be done by the algorithm. First the ISR calls six times the MAC (Medium Access Control) and the DLC (Data Link Control). After the DLC received six fragments of the NWK layer message the DLC calls the NWK layer, which is used to provide services like the call-setup of a connection. In table 1 the communication between the tasks is marked by an backslash (e.g. ISR1.1 → MAC1.1 → DLC1.1 → NWK1.1) and x = 0,1,2,3...; c = 3,5,7; d = 4,6,8,...; as factors to describe the different intervals when a task has to start.

Additionally to this full synchronous tasks we add an asynchronous interrupt by the user. The ISR4 routine reads the input key, e.g. the pressing of the key *talk* by the user, and sends the data to the task APP (application), which represents the program code for the user interface. Then the application starts a call-setup controlled by the network layer (NWK4). The NWK layer sends a new message via the DLC and MAC layer to the fixed part of the DECT system. The sending of this message overlaps with the receiving of other messages.

6 Conclusion

We have shown, how a schedulability analysis, which includes a codesign approach for periodic tasks, can be extended for the analysis of aperiodic tasks and depencies between tasks. Because the algorithm allows to calculate worst-case-response times of tasks which are running on heterogeneous architectures and the task interaction for different use-case-scenarios can be specified by MSCs, the described algorithm is potentially a powerful evaluation algorithm for a system synthesis tool for SDL/MSC specified systems.

7 References

[1] J.Axelsson. Hardware/software partitioning aiming a fulfilment of real-time constraints. Journal of Systems Architecture, Vol.42, No. 6&7, December 1996

[2] T. Benner, R. Ernst. An Approach to Mixed Systems Co-Synthesis. 5th International Workshop on Hardware/Software Codesign. IEEE Computer Society Press, 1997

[3] R. Braek, O. Haugen. Engineering Real Time Systems. Prentice Hall, 1993

[4] A. Burns, A. Wellings. Real-Time Systems and Programming Languages, Addison Wesley. 2nd edition, 1996

[5] ETSI. DIN ETS 300-175, Radio Equipment and Systems (RES), Digital Enhanced Cordless Telecomunications (DECT) Common Interface. ETSI, 1993

[6] N. Faltin, L. Lambert, A. Mitschele-Thiel, F. Slomka. An Annotational Extension of Message Sequence Charts to Support Performance Engineering. In SDL'97, Time for Testing, SDL, MSC and Trends, Proceedings of the eight SDL-Forum, Elsevier Science Publishers, Sept. 1997.

[7] K. Gresser. An Event Model for Deadline Verification of Hard Real-Time Systems. IEEE Procedings 5th Euromicro Workshop on Real Time Systems, Oulu, Finland, 1993

[8] U. Herzog. Performance Evaluation and Formal Description. Advanced Computer Technology, Proc. Reliable Systems and Applications, IEEE Computer Society Press, May 1991

[9] ITU-T. Recommendation Z.100. Specification and Description Language. ITU, 1993

[10] ITU-T. Z.120, Message Sequence Chart. ITU, 1996

[11] L. Lambert. Bewertung von MSC-Spezifikationen mit Task-Graphen. Formale Beschreibungstechniken für verteilte Systeme, 7. GI/ITG-Fachgespräch, Berlin, Juni 1997.

[12] A. Mitschele-Thiel, F. Slomka. A Methodology for Hardware/Software Codesign of Real-Time Systems with SDL/MSC. Proc. International Workshop on Conjoint Systems Engineering (CONSYSE), to appear by IT Press, 1998

[13] Y. Shin, K. Choi. Enforcing Schedulability of Multi-Task Systems by Hardware-Software Codesign. 5th International Workshop on Hardware/Software Codesign, IEEE Computer Society Press, 1996

[14] S. Spitz, F. Slomka, M. Dörfel. SDL* - An Annotated Specification Language for Engineering Multimedia Communication Systems. 6th Open Workshop On High Speed Networks, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1997