Deadline Analysis of AUTOSAR OS Periodic Tasks in the Presence of Interrupts

Yanhong Huang[1], João F. Ferreira[2,3], Guanhua He[2], Shengchao Qin[2,4,⋆], and Jifeng He[1]

[1] East China Normal University
[2] Teesside University
[3] HASLab/INESC TEC, Universidade do Minho
[4] Shenzhen University
`{yhhuang,jifeng}@sei.ecnu.edu.cn`,
`{jff,g.he,s.qin}@tees.ac.uk`

Abstract. AUTOSAR, the open and emerging global standard for automotive embedded systems, offers a timing protection mechanism to protect tasks from missing their deadlines. However, in practice, it is difficult to predict when a deadline is violated, because a task missing its deadline may be caused by unrelated tasks or by the presence of interrupts. In this paper, we propose an abstract formal model to represent AUTOSAR OS programs with timing protection. We are able to determine schedulability properties and to calculate constraints on the allowed time that interrupts can take for a given task in a given period. We implement our model in *Mathematica* and give a case study to illustrate the utility of our method. Based on the results, we believe that our work can help designers and implementors of AUTOSAR OS programs check whether their programs satisfy crucial timing properties.

Keywords: AUTOSAR, timing protection, interrupts, periodic fixed priority scheduling, real-time operating systems.

1 Introduction

The increasing complexity of automobile Electronic Control Units (ECUs) demands standards and methods that support a systematic and reliable approach to the development of automotive software systems. One of such emerging standards is AUTOSAR [1], an initiative led by major automotive OEMs, suppliers and tool vendors to standardize an automotive software architecture. It contains a list of specifications to describe the architecture, including the AUTOSAR Operating System (OS) specification which presents the essential requirements that AUTOSAR OS implementations must follow.

One of the main novelties in AUTOSAR OS is a timing protection mechanism, whose main goal is to prevent tasks from missing their deadlines. The specification suggests the configuration of certain time constraints, like bounding

⋆ Corresponding author.

the execution time of each task and interrupt service routine (ISR). However, in practice, it is difficult to predict when a deadline is violated, because a task missing its deadline may be caused by delays in unrelated tasks. The problem becomes even more challenging when interrupts are enabled, because interrupts can occur at anytime and it is difficult to estimate the time they may take.

In recent works, Bertrand et al. [9] pay attention to the AUTOSAR OS timing protection mechanism. They analyze the mechanism and compare it with other similar mechanisms used in comparable real-time operating systems. They implement the mechanism in a simulation tool and find that "smart configurations" allow better results. However, they do not provide a model that can be directly used by developers to predict deadline faults. Moreover, they do not consider the possibility of interrupts occurring. Hladik et al. [10] provide AUTOSAR OS designers with some usable analysis techniques and corresponding design guidelines. Their work is close to ours, but we focus on periodic tasks. We define a formal model for periodic tasks that can be interrupted by ISRs; our healthiness conditions can directly help the designers configuring the time constraints of periodic tasks mentioned in the specification of the AUTOSAR OS timing protection mechanism. In another work, Schwarz et al. [20] have considered the problem of interrupts occurring in OSEK/VDX OSes. They provide a static analysis for detecting data races between tasks running with different priorities as well as methods to guarantee transactional execution of procedures. However, their focus is on memory safety problems caused by interrupts, whilst our focus is on timing safety.

In this paper, we develop an abstract formal model to represent AUTOSAR OS periodic tasks and to help designers and implementors of AUTOSAR OS programs to analyze and predict deadline faults in their programs. In summary, the main contributions of our work are:

– An abstract formal model that can be used to analyze and predict deadline faults in AUTOSAR OS programs. We define two healthiness conditions, depending on whether interrupts are enabled. Developers of AUTOSAR OS programs can use these healthiness conditions to statically check whether their programs will miss any deadline.
– Based on the model that we develop, we show how to compute the time that interrupts (including sporadic tasks triggered by interrupts) are allowed to take during the execution of tasks. Given that in practice specifications usually mention time intervals (e.g., "interrupt service routines can take n ms, as long as n is between 1ms and 5ms"), rather than showing how to obtain particular times, we show how to derive general time constraints; by focusing on general constraints, developers can consider time intervals.
– To illustrate the practicality of our model, we implement it in *Mathematica* [8]. This allows the automatic calculation of time constraints for interrupts. We also analyze a simple, yet non-trivial, case study. Based on the results, we believe that the work presented in this paper can help designers and implementors of AUTOSAR OS programs check that their programs satisfy crucial timing properties.

The remainder of the paper is organized as follows: Section 2 introduces AUTOSAR OS, its timing protection mechanism, and the goals and assumptions that we have made. We conclude that section with a simple example illustrating the underlying challenges. In Section 3, we define an abstract model to represent AUTOSAR OS periodic tasks. We define two healthiness conditions that can be used to predict deadline faults in Section 4 and Section 5 respectively. Section 5 also presents how to calculate the maximum available time that interrupts can take during the execution of a given task in a given period. In Section 6, we put the method developed into practice to analyze a case study. Finally, we conclude the paper in Section 7, where we discuss related work and further directions to develop the work presented.

2 Background

In this section, we explain the main characteristics of AUTOSAR OS [1] and of its timing protection mechanism. We also present and justify the assumptions that we have made. We conclude with an example that illustrates the problem we want to tackle and may help the reader understand subsequent sections.

On AUTOSAR OS.
AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. One of the specifications included in AUTOSAR is AUTOSAR OS, which specifies AUTOSAR operating systems.

The two main identities discussed in AUTOSAR OS are tasks and interrupt service routines (ISR). Each task and ISR are statically associated with a program. There are two types of tasks: basic and extended tasks. The difference is that the AUTOSAR OS event mechanism is only applied to extended tasks. In other words, basic tasks do not block whereas extended tasks can block until a given event happens. Each task has a priority and AUTOSAR OS suggests a priority-based scheduling policy. Because two tasks cannot occupy the same resource at the same time, AUTOSAR OS prescribes a priority ceiling protocol. As a result, each resource is given a *ceiling priority* that is set at least to the highest priority of all tasks that access a resource or any of the resources linked to this resource. If a task requires a resource, and its current priority is lower than the ceiling priority of the resource, the priority of the task is raised to the ceiling priority of the resource. The priority of the task is reset to its initial priority after releasing the resource. Regarding ISRs, there are also two categories: category 1 and category 2. The main difference is that ISRs of category 1 (ISR1) cannot be controlled by the kernel, while ISRs of category 2 (ISR2) are similar to basic tasks. Contrary to ISR2, ISR1 cannot use any operating system services. That means ISR2 may activate tasks.

On AUTOSAR OS Timing Protection Mechanism.
AUTOSAR is derived from OSEK/VDX [2], an open and widely used industry standard for automotive embedded systems that does not offer any protection mechanism. One of the novelties is that AUTOSAR extends OSEK/VDX

with memory and timing protection mechanisms. Before the publication of AUTOSAR, two extensions were proposed to equip OSEK/VDX with timing protection: OSEK-time and HiS OSEK. Both extensions used deadline monitoring, which allows recovery when a failure is detected. However, deadline monitoring is insufficient to correctly identify what caused a deadline fault. As explained in the AUTOSAR OS specification, a deadline can be violated due to a deadline fault introduced by an unrelated task or by the interference of an interrupt. The fault in this case lies with the unrelated task or interrupt and will propagate through the system until a task misses its deadline. A task that misses a deadline is therefore not necessarily the task that has failed at runtime; it is simply the earliest point at which a timing fault is detected.

The AUTOSAR OS timing protection mechanism protects tasks and interrupt service routines of category 2. It suggests three ways of preventing timing faults:

1. bound the execution time of each task and ISR2;
2. bound the locking time (e.g., the time that resources are held by tasks);
3. guarantee inter-arrival time (e.g., the time for successive activation of tasks and ISR2s).

In practice, these bounds are set statically. At runtime, the bounds are used by the kernel to control the execution of tasks and ISRs. To give a concrete example, in *Arctic Core* [3], the leading open-source implementation of AUTOSAR OS, each task and ISR2 control block have a reference to a *OsTimingProtectionType* structure where the bounds are defined:

```
typedef struct OsTimingProtection {
    // ROM, worst case execution budget in ns
    uint64  executionBudget;
    // ROM, the frame in ns that timelimit may
    // execute in.
    uint64 timeFrame;
    // ROM, time in ns that the task/isr may
    // with a timeframe.
    uint64 timeLimit;
    // ROM, resource/interrupt locktimes
    OsLockingtimeType *lockingTime;
} OsTimingProtectionType;
```

Note the reference to “ROM” in the comments, meaning that these values should be set initially by the programmer and never changed during execution.

On Our Goals and Assumptions.
In this paper, we focus on the timing protection mechanism introduced in AUTOSAR OS. The specification gives the three suggestions above, but it does not mention how to set the time constraints. The goal is to define an abstract formal model to represent AUTOSAR OS periodic tasks and to help designers of AUTOSAR OS programs to analyze and predict deadline faults in their programs. Based on the model, we present how to calculate time constraints on

the allowed time that interrupts can take for a given task in a given period. We introduce a model heavily inspired by an implementation provided by the company *iSoft* (iSoft Infrastructure Software CO.), so we make the assumptions described below.

Assumption 1: In AUTOSAR OS, the tasks that have deadlines are usually periodic tasks (periodicity is achieved by a mechanism called *Schedule Tables*). As a result, all the tasks we discuss in this paper are periodic tasks with a given period. We also assume each task's deadline equals its own period and that all the tasks are ready at the very beginning. Moreover, we assume that the priority and the worst case execution time (WCET) of all tasks are known. The calculation of WCETs is out of the scope of this paper (for details on methods and tools to calculate WCETs, we refer the reader to [4]).

Assumption 2: We assume that any task activated by ISR2 will execute and terminate before ISR2 concludes. When ISR2 terminates, the preempted task is resumed. We consider the total time taken by ISR2 as time spent by interrupts, even if it was running a sporadic task.

Assumption 3: We do not consider AUTOSAR OS resources in this paper, so we assume that the priority of tasks will not change during execution. In other words, we do not discuss the locking time introduced in the timing protection specification.

We now give an example that illustrates the sort of system we will discuss and the sort of problem we want to solve in this paper.

Example 1. Suppose that there are three tasks in the system: A, B, and C; the characteristics of each task, like their priority and deadline (which is the same as their period), are configured statically as shown in Table 1.

Table 1. Properties of the tasks shown in Example 1

Task	**Priority**	**Execution time**	**Deadline (same as Period)**
A	3	1	5
B	2	3	10
C	1	5	15

Figure 1 presents the execution of the three tasks when no interrupts happen. Assuming a fixed priority preemptive scheduling policy and assuming higher priority tasks run before lower priority tasks, task A runs at the very beginning, then B runs, and C starts at the fifth time unit. Because the period of A is 5, task C is preempted and control passes to task A at the sixth time unit. C resumes after the execution of A. In this example, all the tasks meet their own deadlines.

Now we consider an example where interrupts occur during the execution of tasks. We assume the existence of an interrupt called *Isr* that costs one time unit and that can happen at any time. To help with the presentation, in this

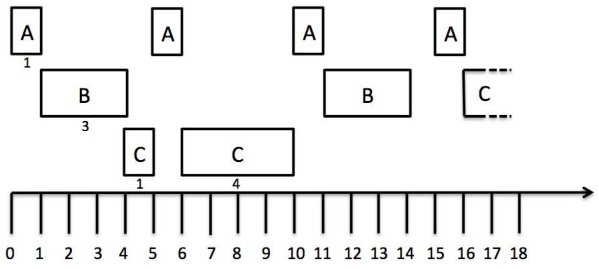

Fig. 1. The execution of tasks without interrupts

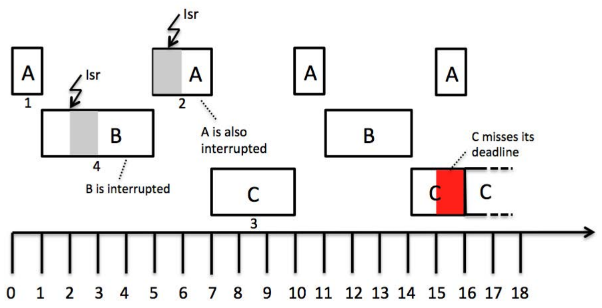

Fig. 2. The execution of tasks with interrupts

example, we assume that *Isr* happens during the execution of both A and B. Figure 2 shows that task C cannot execute until the eighth time unit because the higher priority tasks A and B finish later due to the time taken by *Isr*. As the figure shows, the late start of C leads to a timing fault, as C does not meet its deadline. This example shows that although A and B are interrupted, both of them still meet their deadlines. However, C misses its deadline even though it was never interrupted.

In order to avoid the deadline errors mentioned above, the AUTOSAR OS specification advices to bound the execution time of each task, which is actually the sum of the time taken by task own execution and interrupts execution. Assuming that the worst case execution time of tasks are given (assumption 1), we only need to bound the maximum allowed time taken by interrupts for each task in its given period. For example, to guarantee that the tasks shown meet all their deadlines, task A cannot be interrupted for more than 1 time units, and tasks B and C cannot be interrupted at the same time.

3 Tasks Model

In this section, we present an abstract language to specify AUTOSAR OS periodic tasks that can be interrupted. Later, we also define two healthiness conditions that allow to determine when programs can safely execute without any task missing its deadline. Altogether, we can see our formalization as a model that allows the analysis of the AUTOSAR OS timing protection mechanism in the presence of interrupts.

In our model, a *system* is composed by three components: a set of periodic tasks, a set of ISRs, and a list of functions providing static information. We use the notation $\mathsf{Sys} ::= \{\mathsf{Prog}, \mathsf{ISR}, \mathsf{Funs}\}$ to describe a system, where Prog and ISR denote a set of periodic tasks and a set of ISRs respectively, and Funs represents information about tasks and ISRs. A system with m tasks and n ISRs can be represented as:

$$\mathsf{Prog} ::= \{T_1, T_2, ..., T_m\}$$

$$\mathsf{ISR} ::= \{I_1, I_2, ..., I_n\}$$

Each task has its own priority and period which are configured statically and do not change during execution. Recall that the tasks we consider are periodic and their deadlines are equal to their periods (assumption 1). Each ISR also has a priority, which we assume to be always higher than all the tasks' priorities, so that ISRs can interrupt any running task. In our model, we allow a higher priority ISR to interrupt a lower priority ISR. Moreover, we also assume that the worst case execution time of both tasks and ISRs are given.

To facilitate the expression of properties in our model, we make use of the following functions Funs (we assume that T is an element of Prog and I is an element of ISR):

- **Priority Function**.
  $Pr : T \cup I \to \mathbb{N}$ is used to get the priority of a given task or ISR. Different tasks can have the same priority, but no task can have a higher priority than that of an ISR.
- **Deadline Function**.
  $De : T \to \mathbb{N}$ is used to get the deadline (or period) of a given task. Furthermore, we assume that task T_i in its k^{th} period is ready at time $(k-1) \times De(T_i)$, where $k \in \mathbb{N}^+$, and it must finish before $k \times De(T_i)$.
- **Worst Case Execution Time Function**.
  $ET : T \cup I \to \mathbb{N}$ is used to get the worst case execution time of a given task or ISR. We assume the worst case execution time analysis has been already done, so it is not considered in this paper.
- **Maximum Interrupt Time Function**.
  $IT : T \to \mathbb{N}$ is used to get the maximum allowed time taken by interrupts for a given task. It is the total time that the system allows ISRs to take during one period of a given task.

All systems should define the four functions above. But as we mentioned in the previous section, it is difficult to set a reasonable value for the maximum interrupt time. One of our goals in this paper is to help the designers set this value, so we allow the designers to omit this value, and we can calculate that based on the other three values. Of course, if the designers have given all of them, we can also determine whether the maximum interrupt time is suitable to make all tasks meet their deadlines.

Example 2. We use our language to describe the system mentioned in Example 1. There are three tasks: A, B and C, so we set Prog as $\mathsf{Prog} ::= \{A, B, C\}$. There is only one ISR, so we set ISR as $\mathsf{ISR} ::= \{Isr\}$. The whole system is represented as $\mathsf{Sys} ::= \{\mathsf{Prog}, \mathsf{ISR}, \mathsf{Funs}\}$.

Now we define the priority function so that $Pr(A) = 3$, $Pr(B) = 2$, $Pr(C) = 1$, and $Pr(Isr) = 4$; the deadline function is defined as $De(A) = 5$, $De(B) = 10$, $De(C) = 15$; finally, the worst case execution time function is defined as $ET(A) = 1$, $ET(B) = 3$, $ET(C) = 5$ and $ET(Isr) = 1$.

In this example, the maximum allowed time taken by interrupts for each task, $IT(T_i)$, is not given. Later, in Section 5, we show how the function IT can be defined.

4 Timing Protection without Interrupts

In this section, under the assumption that interrupts are disabled, we show conditions under which a collection of tasks can safely execute without missing deadlines.

Given that tasks are periodic, we are interested in evaluating the behaviors of tasks in their shortest repeating cycle[1], which is given by the least common multiple of the periods of all tasks. Given a program Prog, we denote the shortest repeating cycle of its tasks as $\mathsf{lcm}(\mathsf{Prog})$. In Example 2, we have $\mathsf{lcm}(\mathsf{Prog})$=30, because the periods of A, B, and C are, respectively, 5, 10, and 15. In other words, the executing pattern shown in Figure 1 repeats itself after 30 time units.

To ensure that a system can run safely without any task missing its deadline, the requirement that we need to guarantee is:

Requirement. *Each task T_i is expected to have a complete execution in each period from $(k{-}1){\times}De(T_i)$ to $k{\times}De(T_i)$, for all $k \in \mathbb{N}^+$.*

For instance, in Example 1, we want to guarantee that task C starts running and terminates within the periods from $(k{-}1){\times}15$ to $k{\times}15$, for all $k \in \mathbb{N}^+$. However, in the scenario shown in Figure 2 this property is violated because C is not able to terminate within its first period (from 0 to 15).

To formalize the requirement above we start by analyzing the time available for a given task to execute during a given period. Given that our model assumes a fixed priority preemptive scheduling[2], to calculate the time available for task T_i during its k^{th} period (denoted by $\mathbf{AT}_k(T_i)$), we subtract from T_i's deadline the amount of time units taken by all the tasks that have a higher priority than T_i. There is a special case that needs to be considered: if a task T_i has several higher priority tasks with the same period, we combine these tasks

[1] The shortest repeating cycle is the smallest amount of time after which the executing pattern repeats itself (when interrupts are disabled).

[2] A fixed priority preemptive scheduler always chooses the highest priority task that is ready to execute.

into one task when calculating the time available for T_i. For example, suppose that tasks T_{j_1},..., T_{j_n} have the same period, i.e., $De(T_{j_1})$=...=$De(T_{j_n})$. When calculating the time available for task T_i whose priority is lower than all of those tasks T_{j_1},..., T_{j_n}, we define a new task T_j with period $De(T_j)$=$De(T_{j_1})$, priority $Pr(T_j)$=$min(Pr(T_{j_1}), ..., Pr(T_{j_n}))$ and the worst case execution time $ET(T_j)$=$ET(T_{j_1})$+...+$ET(T_{j_n})$. Assuming that $\mathbf{TT}(T_j, l)$ represents the total time taken by task T_j up to the time unit l, we define $\mathbf{AT}_k(T_i)$ as follows:

$$\mathbf{AT}_k(T_i) =_{df} De(T_i) - \sum_{\substack{Pr(T_i) \leq Pr(T_j) \\ \wedge\ j \neq i}} (\mathbf{TT}(T_j, k \times De(T_i)) - \mathbf{TT}(T_j, (k-1) \times De(T_i)))$$

Clearly, the value of $\mathbf{TT}(T_j, k \times De(T_i)) - \mathbf{TT}(T_j, (k-1) \times De(T_i))$ corresponds to the time taken by task T_j between the time unit $(k-1) \times De(T_i)$ and the time unit $k \times De(T_i)$; in other words, it corresponds to the time taken by task T_j during the k^{th} period of T_i. To define $\mathbf{TT}(T_j, l)$ we first observe that the number of times that T_j executes up to time unit l is at least $\lfloor \frac{l}{De(T_j)} \rfloor$. As a result, the total time that T_j takes up to time unit l is at least $\lfloor \frac{l}{De(T_j)} \rfloor \times ET(T_j)$. To determine the exact amount of time, we need to analyze the value of $l \bmod De(T_j)$: if it is long enough for T_j to execute, then T_j executes once; otherwise, T_j will run for $l \bmod De(T_j)$ time units. Put more formally, we have:

$$\mathbf{TT}(T_j, l) =_{df} \begin{cases} \lfloor \frac{l}{De(T_j)} \rfloor \times ET(T_j) + ET(T_j) & \text{if } l \bmod De(T_j) \geq ET(T_j) \\ \lfloor \frac{l}{De(T_j)} \rfloor \times ET(T_j) + l \bmod De(T_j) & \text{if } l \bmod De(T_j) < ET(T_j) \end{cases}$$

Example 3. Based on Example 2, we show how to calculate each task's available time in each period in the shortest repeating cycle. The shortest repeating cycle is 30 time units, task A will execute 6 times, task B will execute 3 times, and task C will execute 2 times.

$$\begin{aligned}
&AT_i(A) = 5 - 0 = 5 \qquad \text{where } i \in \{1,2,3,4,5,6\} \\
&AT_1(B) = 10 - \lfloor \tfrac{10 \times 1}{5} \rfloor \times 1 = 10 - 2 = 8 \\
&AT_2(B) = 10 - (\lfloor \tfrac{10 \times 2}{5} \rfloor \times 1 - \lfloor \tfrac{10 \times 1}{5} \rfloor \times 1) = 10 - (4 - 2) = 8 \\
&AT_3(B) = 10 - (\lfloor \tfrac{10 \times 3}{5} \rfloor \times 1 - \lfloor \tfrac{10 \times 2}{5} \rfloor \times 1) = 10 - (8 - 6) = 8 \\
&AT_1(C) = 15 - (\lfloor \tfrac{15 \times 1}{5} \rfloor \times 1 + \lfloor \tfrac{15 \times 1}{10} \rfloor \times 3 + 3) = 15 - (3 + 6) = 6 \\
&AT_2(C) = 15 - ((\lfloor \tfrac{15 \times 2}{5} \rfloor \times 1 - \lfloor \tfrac{15 \times 1}{5} \rfloor \times 1) + (\lfloor \tfrac{15 \times 2}{10} \rfloor \times 3 - (\lfloor \tfrac{15 \times 1}{10} \rfloor \times 3 + 3))) \\
&\qquad\quad = 15 - ((6 - 3) + (9 - 6)) = 15 - (3 + 3) = 9
\end{aligned}$$

Example 4. We use this example to explain the special issue we mentioned before. A system has three tasks, two of them have a same period. We set Prog ::= $\{T_1, T_2, T_3\}$. The priority function is defined as $Pr(T_1) = 3$, $Pr(T_2) = 2$, and $Pr(T_3) = 1$. The deadline function is defined as $De(T_1) = De(T_2) = 8$ and $De(T_3) = 10$. The worst case execution time is defined as $ET(T_1) = 2$, $ET(T_2) = 2$, and $ET(T_3) = 4$. When calculating $AT(T_3)$, we combine T_1 and T_2 into T_{12} which $De(T_{12}) = 8$, $Pr(T_{12}) = 2$, and $ET(T_{12}) = 4$. Hence, i.e., the time is available for task T_3 in its first period: $AT_1(T_3) = 10-(\lfloor\frac{10\times1}{8}\rfloor\times4+10\,mod\,8) = 10-(4+2) = 4$.

Now that we have a formal definition for $\mathbf{AT}_k(T_i)$, we can use it to formalize conditions under which a collection of tasks can safely execute without any deadline being missed. We first show how to formalize the situation where no interrupts can occur; we then extend it to the situation where interrupts are enabled.

Definition 1 (Healthiness $\mathcal{H}_1$). We say that a system is in $\mathcal{H}_1$ if in the absence of interrupts, all the tasks meet their respective deadlines in their shortest repeating cycle. Formally, given a system Sys, containing m tasks $T_1, \ldots, T_m$, we define the healthiness predicate $\mathcal{H}_1$:

$$\mathcal{H}_1(\mathsf{Sys}) =_{df} \forall i{\in}\{1,...,m\}, k{\in}\{1,..., \frac{\mathsf{lcm(Prog)}}{De(T_i)}\} \bullet \mathbf{AT}_k(T_i) \geq ET(T_i)$$

Using the values calculated in Example 3, we can conclude that the system described in Example 1 is an $\mathcal{H}_1$ system.

5 Timing Protection with Interrupts

In the previous section, we have defined $\mathcal{H}_1$ systems, which are systems where no deadlines are missed, as long as there are no interrupts. In this section, we study the case when interrupts can occur.

To consider interrupts, we redefine the functions shown in the previous section to include the time taken by interrupts. First, we define $\mathbf{ATI}_k(T_i)$, which represents the time available for a given task T_i during its k^{th} period when interrupts are allowed to occur[3].

$$\mathbf{ATI}_k(T_i) =_{df} De(T_i) - \sum_{\substack{Pr(T_i)\leq Pr(T_j) \\ \wedge\ j\neq i}} (\mathbf{TTI}(T_j, k{\times}De(T_i)) - \mathbf{TTI}(T_j, (k{-}1){\times}De(T_i)))$$

where $\mathbf{TTI}(T_j, l)$ represents the total time taken by task T_j and by interrupts up to time unit l. Following the discussion in the previous section, we observe that the value of $\mathbf{TTI}(T_j, k{\times}De(T_i)) - \mathbf{TTI}(T_j, (k{-}1){\times}De(T_i))$ corresponds to the time taken by task T_j and by interrupts during the k^{th} period of T_i. The function $\mathbf{TTI}$ is defined below.

Functions $\mathbf{ATI}$ and $\mathbf{TTI}$ are similar to functions $\mathbf{AT}$ and $\mathbf{TT}$. The difference is that the former two consider the execution time $ET(T)$ and the maximum allowed time for interrupts $IT(T)$, while the latter two only consider the execution time $ET(T)$. Note that the sum of the worst case execution time of a task and the maximum allowed time for interrupts for that task is the actual bounding execution time mentioned in AUTOSAR OS timing protection mechanism.

[3] The special case discussed in the previous section should be also considered here.

$$\mathbf{TTI}(T_j, l) =_{df} \begin{cases} \lfloor \frac{l}{De(T_j)} \rfloor \times (ET(T_j) + IT(T_j)) + (ET(T_j) + IT(T_j)) \\ \qquad \text{if } \; l \; \mathsf{mod} \; De(T_j) \geq (ET(T_j) + IT(T_j)) \\ \lfloor \frac{l}{De(T_j)} \rfloor \times (ET(T_j) + IT(T_j)) + l \; \mathsf{mod} \; De(T_j) \\ \qquad \text{if } \; l \; \mathsf{mod} \; De(T_j) < (ET(T_j) + IT(T_j)) \end{cases}$$

The time available for a given task may be different in its different periods; we use the notation $\mathbf{minATI}(T_i)$ to denote the minimum time available for task T_i in the repeating cycle of the system:

$$\mathbf{minATI}(T_i) = \mathbf{min}(\mathbf{ATI}_1(T_i), \ldots, \mathbf{ATI}_k(T_i)), \quad \textit{where} \;\; k = \frac{\mathsf{lcm}(\mathsf{Prog})}{De(T_i)}$$

We now use **minATI** to extend Definition 1 for the case where interrupts can occur.

Definition 2 (Healthiness $\mathcal{H}_2$). We say that a system is in $\mathcal{H}_2$ if (1) a system is in $\mathcal{H}_1$ and (2) when in the presence of interrupts, the system guarantees that all tasks can meet their deadlines. Formally, given a system as described in Definition 1, we define the healthiness predicate $\mathcal{H}_2$:

$$\mathcal{H}_2(\mathsf{Sys}) =_{df} \Big(\mathcal{H}_1(\mathsf{Sys}) \;\wedge \forall i \in \{1, ..., m\} \bullet \mathbf{minATI}(T_i) \geq ET(T_i) + IT(T_i) \Big)$$

$\mathcal{H}_2$ systems guarantee that all tasks will meet their deadlines even if interrupts occur. According to the definition above, healthiness condition $\mathcal{H}_1$ is subsumed by $\mathcal{H}_2$, meaning that an $\mathcal{H}_2$ system must be an $\mathcal{H}_1$ system. In other words, when a system wants to guarantee all tasks meet their deadlines in the presence of interrupts, the system must guarantee all tasks meet their deadlines in the absence of interrupts at first. However, we should note that in practice, it is difficult to set a reasonable maximum allowed time for each task to be interrupted, because we always have to consider the whole system. We can overcome this difficulty by observing that Definition 2 gives us a system of inequations that can be used to calculate the values for function IT:

$$\forall i \in \{1, ..., m\} \bullet \;\; IT(T_i) \leq \mathbf{minATI}(T_i) - ET(T_i) \tag{1}$$

We will use this system of inequations in the next section.

As a concluding remark, we observe that the function IT can be defined according to different requirements. For example, three cases are suggested as below:

- **Case 1**. The system allows only one task to be interrupted at runtime. For example, task T_i is the only task that can be interrupted; the value of IT for other tasks is set to 0: $\forall j, j{\neq}i \bullet IT(T_j) = 0$.
- **Case 2**. The system allows every task to be interrupted the same amount of time, so we define $\forall i \bullet IT(T_i) = t$, for some t. One implementation of AUTOSAR OS developed by *iSoft* adopts this kind of timing protection mechanism.

– **Case 3**. The system allows every task to be interrupted, but distributes them into different groups according to the amount of time allowed for interrupts. Every two tasks in the same group have the same allowed time for interrupts, i.e., if task T_i and task T_j are in the same group, then $IT(T_i) = IT(T_j)$.

To illustrate how IT can be used, we give an example based on Example 3.

Example 5. Based on Example 3, we assume task C is the only task that can be interrupted (like case 1 above), so we set $IT(A) = IT(B) = 0$. In this case, $\forall k \bullet \ \mathbf{ATI}_k(C) = \mathbf{AT}_k(C)$ and $\mathbf{minATI}(C) = \mathbf{min}(6, 9) = 6$. So $IT(C) \leq (6 - 5) = 1$. In fact, looking at Figure 1, we see that if C is interrupted for more than one time unit, task C will miss its deadline.

6 A Case Study

In this section, we use a more complex example to illustrate the utility of our method. The tricky part of the example is that the deadline of a lower priority task is less than that of a higher priority task. Moreover, the higher priority task may not finish in one period of lower priority task, but in two, which makes it difficult to predict the *healthiness* of the system.

Implementation.
We have implemented all the functions and healthiness conditions in *Mathematica* [8] and evaluated our method. The user can define a system using the prototype; when the priority, deadline and worst case execution time of all the tasks of a program are given, the prototype can determine whether a system is in $\mathcal{H}_1$. If the maximum allowed time taken by interrupts for all the tasks is given as additional input (i.e., function IT), our prototype can determine whether a system is in $\mathcal{H}_2$. Otherwise, it can calculate a set of time constraints to help the

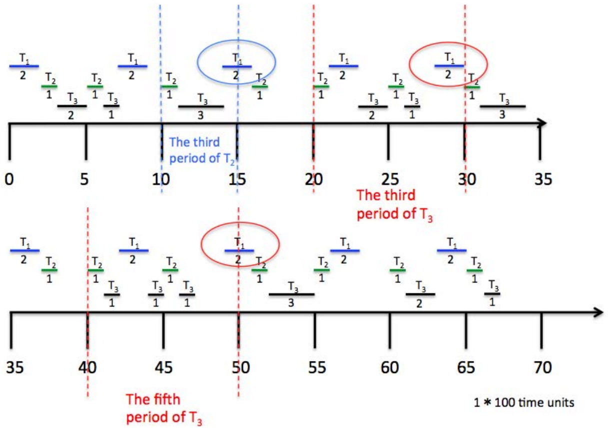

Fig. 3. The scheduling of tasks in case study

designers configure the maximum allowed time for interrupts (according to the system of inequations (1) shown in Section 5).

Example.
The example we use is shown in Table 2. We present the scheduling of tasks without interrupts in Figure 3. In the example, the priority of task T_2 is lower than that of T_1, but the deadline of T_2 is less than that of T_1. In the third period of task T_2 and in the fifth period of task T_3, T_1 starts at the last time unit, and does not finish in one period of the lower priority tasks. In the third period of task T_3, T_1 finishes its execution in one period of T_3. But when interrupts are enabled, T_1 may finish in next period of T_3. Moreover, we can see that the time available for the same task in different periods is different. That is why we should consider the shortest repeating cycle.

Table 2. Properties of the tasks shown in the case study

Task	Priority	Execution time	Deadline (same as Period)
T_1	3	200	700
T_2	2	100	500
T_3	1	300	1000

$$
\begin{aligned}
&AT_i(T_1) > 700 - 0 = 700\\
&AT_i(T_1) > ET(T_1) \text{ where } i \in \{1, ..., 10\}\\
&AT_1(T_2) = 500 - 200 = 300\\
&AT_1(T_2) > ET(T_2)\\
&AT_2(T_2) = 500 - (\lfloor \tfrac{500*2}{700} \rfloor * 200 + 200 - 200) = 500 - (400 - 200) = 300\\
&AT_2(T_2) > ET(T_2)\\
&AT_3(T_2) = 500 - ((\lfloor \tfrac{500*3}{700} \rfloor * 200 + 500 * 3 \bmod 700 - (\lfloor \tfrac{500*2}{700} \rfloor * 200 + 200))\\
&\qquad\quad = 500 - ((400 + 100) - 400) = 400\\
&AT_3(T_2) > ET(T_2)\\
&AT_4(T_2) = 500 - (\lfloor \tfrac{500*4}{700} \rfloor * 200 + 200 - (\lfloor \tfrac{500*3}{700} \rfloor * 200 + 500 * 3 \bmod 700))\\
&\qquad\quad = 500 - (600 - 500) = 400\\
&AT_4(T_2) > ET(T_2)\\
&\qquad\quad \ldots\\
&AT_1(T_3) = 1000 - (\lfloor \tfrac{1000*1}{700} \rfloor * 200 + 200 + \lfloor \tfrac{1000*1}{500} \rfloor * 100) = 1000 - (400 + 200) = 400\\
&AT_1(T_3) > ET(T_3)\\
&\qquad\quad \ldots\\
&AT_6(T_3) = 1000 - ((\lfloor \tfrac{1000*6}{700} \rfloor * 200 + 200 - (\lfloor \tfrac{1000*5}{700} \rfloor * 200\\
&\qquad\quad +1000 * 5 \bmod 700)) + (\lfloor \tfrac{1000*6}{500} \rfloor * 100 - \lfloor \tfrac{1000*5}{500} \rfloor * 100))\\
&\qquad\quad = 1000 - ((1800 - 1500) + (1200 - 1000)) = 1000 - (300 + 200) = 500\\
&AT_6(T_3) > ET(T_3)\\
&AT_7(T_3) = 1000 - ((\lfloor \tfrac{1000*7}{700} \rfloor * 200 - (\lfloor \tfrac{1000*6}{700} \rfloor * 200 + 200)) + (\lfloor \tfrac{1000*7}{500} \rfloor * 100 - \lfloor \tfrac{1000*6}{500} \rfloor * 100))\\
&\qquad\quad = 1000 - ((2000 - 1800) + (1400 - 1200)) = 1000 - (200 + 200) = 600\\
&AT_7(T_3) > ET(T_3)
\end{aligned}
$$

Fig. 4. The calculation details for the case study

Healthiness $\mathcal{H}_1$. To evaluate whether this example is an $\mathcal{H}_1$ system, we have to make sure all the tasks can meet their deadlines when there are no interrupts (according to Definition 1). Using our implementation in *Mathematica*, we conclude that this system is in $\mathcal{H}_1$. We also list some details in Figure 4 to help the reader understand our method.

The shortest repeating cycle of this example is 7000 time units. So we should consider 10 periods of task T_1 (because the period is 700), 14 periods of task T_2 (because the period is 500), and 7 periods of T_3 (because the period is 1000). Here, we only list a few.

Healthiness $\mathcal{H}_2$. We calculate the maximum allowed time taken by interrupts for each task by following the constraints given in the previous section. More specifically, the values of $IT(T_1)$, $IT(T_2)$ and $IT(T_3)$ should satisfy the inequality $2IT(T_1) + 2IT(T_2) + IT(T_3) \leq 100$ (calculated by the implementation in *Mathematica*). We can use this inequality to set the interrupt time $IT(T_i)$ for task T_i. For example, if we want to make the maximum allowed time that interrupts can take the same for all tasks (as described in Case 2 above), we can set $IT(T_1){=}IT(T_2){=}IT(T_3){=}20$ at most. Moreover, if the system has already set the function IT, we can use this inequality to evaluate whether this system is in $\mathcal{H}_2$.

7 Related Work and Conclusion

Related Work.
Many researchers have done much work on fixed priority scheduling of periodic tasks. The problem of scheduling periodic tasks with hard deadlines on a uniprocessor was first studied by Liu and Layland in 1973 [11]. Later, Lehoczky developed an exact schedulability criterion for the fixed priority scheduling of periodic tasks with arbitrary deadlines [12]. Harbour et al. presented a generalized model of fixed priority scheduling of periodic tasks where each task's execution priority may vary [13]. Katcher et al. developed scheduling models for four generic scheduler implementations that represent the spectrum of implementations found in real-time kernels [14]. In many respects, there is an overlap of concerns between these works and this paper. However, the focus and novelty of our work is on estimating the available time for interrupts that can occur in periodic AUTOSAR OS programs.

A related line of research is presented in [15,16,17,18], where timed automata are used to analyze different problems about scheduling. We plan to reuse parts of these works to extend our model with support for AUTOSAR resources (see discussion on future work below).

There has also been a considerable amount of work on interrupt-based programs. For example, the works [5,20,21] deal with the analysis and verification of memory safety properties. More related to this paper is the work presented in [19], where a tool for deadline analysis of interrupt-driven Z86-based software is presented. Other works give advice and introduce models to guarantee that interrupts do not cause timing faults: in [6], the number of the interruptions during certain time intervals is limited; in [7], tasks and interrupts are integrated to provide predictable execution times in real-time systems.

The work presented in this paper can be seen as a continuation of our previous work. In [22], we have developed a formal model of interrupt-based programs from a probabilistic perspective: we designed a probabilistic operational semantics able to capture the potential properties, and specified the time constraint of interrupt programs. In [23], we have analyzed and verified ORIENTAIS, an operating system based on OSEK/VDX and developed by *iSoft*.

Conclusion.
This paper proposes a simple and abstract formal model to represent AUTOSAR OS programs that need to ensure certain timing properties. The model can be used to predict if a given periodic task will miss its deadline. We present two healthiness conditions: condition $\mathcal{H}_1$ is used to check if tasks meet their deadlines when interrupts are disabled; condition $\mathcal{H}_2$ is used to check if tasks meet their deadlines when interrupts are enabled. To have a definitive answer to the question "is a given system an $\mathcal{H}_2$ system?", we need to know what is the time taken by interrupts, so we use *Mathematica* to compute time constraints for interrupts. Rather than assuming specific times for interrupts, we are able to give answers of the type "the given system is an $\mathcal{H}_2$ system, provided that task T_1 is not interrupted for more than 5 time units *and* task T_2 is not interrupted for more than 3 time units". It is worth saying that, although our focus is on interrupt-based programs, the timing constraints developed are general enough to be used for other purposes. For example, the theory applies if instead of interrupts we consider delays; saying that a given task can be interrupted for at most 5 time units is the same as saying that the task can be delayed by at most 5 time units.

We believe that the work presented in this paper can help designers and implementors of AUTOSAR OS programs check that their programs satisfy crucial timing properties.

As for future work, we plan to extend the model to consider the locking time of resources mentioned in the AUTOSAR OS timing protection specification. The difficulty of adding resources in the model is related with the *priority ceiling protocol*, which causes the priority of tasks to change during execution. The extended model should be able to cope with dynamic changes of priority (i.e., functions like **AT** and **ATI** will need to be redefined). We also plan to generalize the model so that periods and deadlines do not coincide. Finally, we will construct software tools that automatically construct models from the source code of AUTOSAR OS programs. We envisage these tools to be integrated into tool chains specifically designed for AUTOSAR OS developers.

Acknowledgments. This work was partly supported by the Danish National Research Foundation and the National Natural Science Foundation of China (Grant No. 61061130541) for the Danish-Chinese Center for Cyber Physical Systems. And, it is also supported by National Basic Research Program of China (Grant No. 2011CB302904), National High Technology Research and Development Program of China (Grant Nos. 2011AA010101 and 2012AA011205).

The majority of the work reported in this paper was done while Yanhong Huang was a visiting researcher at Teesside University (funded by the Teesside University Research Fund). The support of Teesside University is gratefully acknowledged.

References

1. AUTOSAR. Specification of Operating System V3.1.1 R3.1 Rev 0002 (2012), `http://www.autosar.org/` (last accessed: July 1, 2013)
2. OSEK/VDX, `http://www.osek-vdx.org/` (last accessed: July 1, 2013)
3. Arctic Core — the open-source AUTOSAR embedded platform, `http://www.arccore.com/` (last accessed: July 1, 2013)
4. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. 7(3) (2008)
5. Tuch, H.: Formal Memory Models for Verifying C Systems Code. Ph.D. Thesis. University of NSW, Australia (2008)
6. Regehr, J., Reid, A., Webb, K.: Eliminating stack overflow by abstract interpretation. In: EMSOFT (2003)
7. Leyva-del-Foyo, L.E., Mejia-Alvarez, P., de Niz, D.: Predictable Interrupt Management for Real Time Kernels over conventional PC Hardware. In: RTAS (2006)
8. Wolfram Research, Inc., Mathematica, Version 8.0, Champaign, IL (2010).
9. Bertrand, D., Faucou, S., Trinquet, Y.: An analysis of the AUTOSAR OS timing protection mechanism. In: ETFA (2009)
10. Hladik, P.E., Deplanche, A.M., Faucou, S., Trinquet, Y.: Adequacy between AUTOSAR OS specification and real-time scheduling theory. In: SIES (2007)
11. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Jounal of the Assocaition for Computing Macheinery 20(1) (1973)
12. Lehoczky, J.P.: Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: RTSS (1990)
13. Harbour, M.G., Klein, M.H., Lehoczky, J.P.: Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority. In: RTSS (1991)
14. Katcher, D.I., Arakawa, H., Strosnider, J.K.: Engineering and analysis of fixed priority schedulers. IEEE Transactions on Software Engineering (1993)
15. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Wang, Y.: TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 60–72. Springer, Heidelberg (2004)
16. Fersman, E., Wang, Y.: A Generic Approach to Schedulability Analysis of Real Time Tasks. Nordic Journal of Computing 11(2) (2004)
17. Krcal, P., Wang, Y.: Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 236–250. Springer, Heidelberg (2004)
18. Fersman, E., Mokrushin, L., Pettersson, P., Wang, Y.: Schedulability Analysis of Fixed-Priority Systems Using Timed Automata. Journal of Theoretical Computer Science 354(2) (2006)
19. Brylow, D., Palsberg, J.: Deadline Analysis of Interrupt-Driven Software. IEEE Transactions on Software Engineering (2004)
20. Schwarz, M.D., Seidl, H., Vojdani, V., Lammich, P., Muller-Olm, M.: Static analysis of interrupt-driven programs synchronized via the priority ceiling protocol. In: POPL (2011)

21. Feng, X., Shao, Z., Guo, Y., Dong, Y.: Certifying Low-Level Programs with Hardware Interrupts and Preemptive Threads. J. Autom. Reasoning 42(2-4) (2009)
22. Zhao, Y., Huang, Y., He, J., Liu, S.: Formal Model of Interrupt Program from a Probabilistic Perspective. In: ICECCS (2011)
23. Shi, J., Zhu, H., He, J., Fang, H., Huang, Y., Zhang, X.: ORIENTAIS: Formal Verified OSEK/VDX Real-Time Operating System. In: ICECCS (2012)