Mastering Timing Challenges for the Design of Multi-Mode Applications on Multi-Core Real-Time Embedded Systems

Mircea Negrean and Rolf Ernst
Institute of Computer and Network Engineering
Technische Universität Braunschweig
D-38106, Braunschweig, Germany
{negrean,ernst}@ida.ing.tu-bs.de

Simon Schliecker
Symtavision GmbH
Frankfurter Str. 3 C
D-38122, Braunschweig, Germany
schliecker@symtavision.com

Abstract Driven by the increasing demand for computational power and by the rising applications' complexity in various embedded application domains, multi-core solutions emerge as the predominant platform for embedded real-time applications. In this context, designers have to face new challenges generated by the need to accommodate applications with complex timing behaviour, e.g. multi-mode applications that can switch between different operational modes at runtime. Consequently the availability of appropriate timing analysis methods for the prediction of the timing behaviour is essential for the design of multi-core real-time systems.

Relying on current automotive practice and on related work of the real-time research community we explain and exemplify how tool supported formal analysis methods can be applied to current and upcoming industrial design. We present recent progress in the field of scheduling analysis methods and discuss challenges and new options in the design of multi-mode applications on multi-core real-time systems.

Keywords Schedulability analysis, timing analysis, multi-mode applications, multi-core, real-time

1. Introduction

Multi-core architectures

Driven by power constraints, cost-efficiency and performance requirements, embedded system designs adopt the trend towards multi-core architectures (e.g. [1]). The new multi-core processors are aimed to host the significant increase in the computational workload of next generations embedded applications on as few processors as possible. By using powerful multi-core processors, it will be possible to integrate multiple functionalities into a single chip or to parallelize complex computations over multiple cores, e.g. in relatively high-performance domains such as engine control or advanced driver assistance systems.

Some of these complex functions consist of tasks that exhibit a dynamic behaviour at runtime, e.g. engine speed synchronous tasks in the automotive power train controllers. Engine-synchronous tasks are periodic tasks whose recurrence depends on the camshaft and crankshaft positions that vary with the engine speed. The variable recurrence of these tasks at runtime leads to a continuous change in the configurations that have to be taken into account for the OS schedule on the processors and leads to a multi-mode behaviour of the entire system.

The multi-mode behaviour of such systems is, however, much more complex. The load associated with the engine-synchronous tasks increases with the engine speed due to the higher rate of activations. If no measures are in place, the load on the controllers may eventually increase above a critical value making the system unschedulable and the engine control inefficient or even unstable. Furthermore, when multi-mode applications, potentially consisting of communicating tasks, have to be accommodated on multi-core platforms the timing behaviour of the entire system is challenged and requires increased attention.

Analysis methods and tools

Consequently, appropriate methods for the prediction of the timing behaviour are required for the design of reliable multi-core real-time systems. Scheduling and performance analysis techniques aim to help designers in detecting infeasible system configurations in the early design phases and therewith to prevent costly design mistakes. Currently, sophisticated tools (e.g. aiT [2], SymTA/S [3]) for timing analysis at the code-level, controller-level and networked system-level are becoming state-of-the-art for efficient timing verification. However, in the perspective of advanced multi-core environments the methodologies have to be extended to face the arising challenges.

Contribution

In this paper, we address this problem and show how tool supported analysis methods can be applied to the current and upcoming industrial design. We highlight the implications of accommodating multi-mode applications on multi-core systems and identify the need for enhanced solutions for timing and performance design. The analysis approaches proposed so far do not suit real-time systems which accommodate tasks with angular

recurrence, as can be found in automotive power train controllers. We identify similarities between the problem of scheduling real-time applications which accommodate tasks with angular recurrence and the problem of scheduling multi-mode applications. Relying on recent progress in the field of scheduling analysis for multi-mode applications, we propose new options for the design and analysis of complex multi-core real-time embedded systems.

2. Case study description

In what follows, we introduce an automotive specific use case in order to explain and exemplify how tool supported formal analysis methods are used in the current automotive practice and how these methods can be also applied to the foreseeable automotive multi-core setups.

The setup depicted in Figure 1 corresponds to a partitioned multiprocessor system where the two cores are independently scheduled according to a fixed-priority scheduler (e.g. OSEK/VDX [4]). The system consists of several periodic tasks and some high priority "engine-synchronous" tasks that measure the current engine state and control actuators such as fuel injection several times per engine rotation. All engine synchronous tasks are mapped to Core2. In this way we consider a simplified configuration of a current power train application (i.e. engine control) and envisage a possible implementation on the future multi-core architectures.

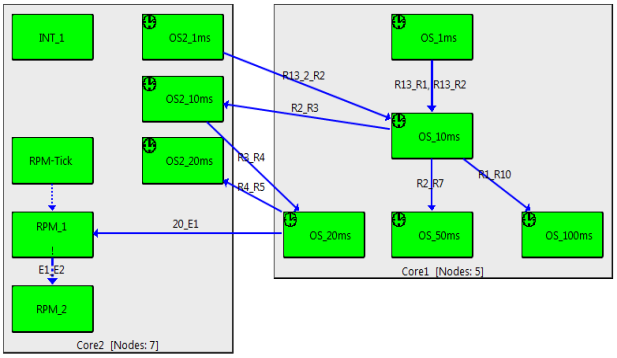

Figure 1 Dual-Core processor with inter-core communication.

Our assumption is that the only degree of variance in this system is given by the engine speed, measured in revolutions per minute (rpm). Thus, depending on the engine speed the activation of the engine-synchronous tasks will vary. For example, an engine-synchronous task that is activated each 360^0 (i.e. each 1 rotation) at 1000rpm has a recurrence of 60ms and 10ms at 6000rpm.

With increasing engine speed, the load increases due to the higher rate of activations. Figure 2 illustrates the load situation for the dual-core system in Figure 1.

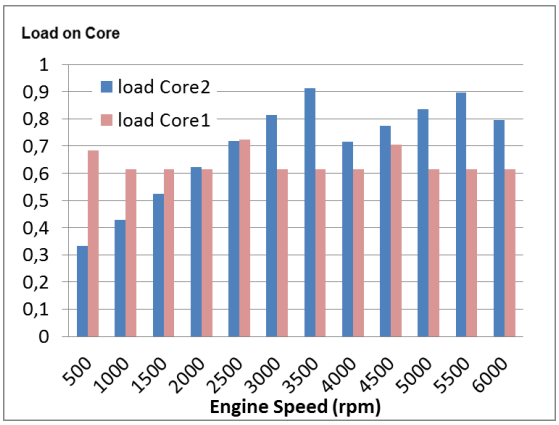

Figure 2 Workload to be processed on each core. Increasing engine speed leads to higher rate of activation for engine-synchronous tasks. In addition task modes lead to varied workload. Critical load on Core 2 is reached around 3500 and 5500 rpm.

2.1. Impact of mode changes on processor load

If no further measures are in place, the load may eventually increase above a critical value, making the system unschedulable and the engine control inefficient or even unstable. Different solutions are therefore applied in order to reduce the workload at high engine speeds. For example, some tasks are changing their behaviour at higher engine speeds, computing only rough control values (i.e. tasks reduce their execution times). Furthermore, other tasks are aborted when the engine speed reaches a critical value. In this way, automotive power train controllers resort to mode change mechanisms in order to perform at a high quality for low engine speeds and to adequately operate also under high engine speeds.

This behaviour can be reflected in a scheduling model of the system. Let's assume that starting e.g. at 3500rpm, the engine-synchronous tasks change their behaviour and some internal functions are completely shut-off (e.g. "high-quality mode" to "medium quality mode"). Later e.g. at 5500rpm, additional functions are turned off (e.g. "low quality mode"). In addition, some functions provided by the periodic tasks are reactive to the current processor load. This task flexibility ensures that the schedulability is maintained through the complete spectrum of operating conditions.

The diagram in Figure 2 shows the task workload that has to be serviced by each core, ignoring any inter-core communication effects. For each possible engine speed the load is computed based on the

- Period and execution time for the periodic tasks, where the execution time can be a function of the engine speed and the
- Rate of activation and execution time for the engine-synchronous tasks, where the rate of activation *is* and the execution time *can be* a function of the engine speed.

One can see the effect of changed task behaviour at engine speeds 3500rpm and 5500rpm, where task RPM_2 is put into a reduced quality mode. In addition, also some periodic tasks in this example have increased execution time requirements at certain engine speeds (which leads to a slight load variation on Core1 at 2500 and 4500 rpm).

This diagram can be easily produced with a model based scheduling analysis tool (such as SymTA/S [3]). Each engine speed can be seen as a different scenario, in which the corresponding model parameters are set (for example through scripting or a dedicated user interface). For large range of engine speeds the granularity of 500rpm will be sufficient, because the behaviour can be assumed to be largely monotonous with respect to increasing engine speed. In addition, any engine speed at which a task behaviour changes should be included in the experiments.

2.2. Suggested procedure for analysis

Simple case adapted from single-core process

Of course the processor load alone is not a sufficient metric to show the schedulability of the system, because it does not allow for checking the adherence to typical timing constraints such as latency. For example, a latency constraint may be associated with task RPM_2 which requires the task to recompute a control parameter for the injection no later than 1ms after it has been activated.

Also here, performance analysis has to be performed for all potential run-time scenarios. With the scripting support in model-based analysis tools such as SymTA/S [3] this can be easily achieved today. Figure 3 shows the results of a series of experiments capturing the timing conditions for different engine speeds. The worst-case response times have been computed using the busy window technique provided by the tool mentioned above. The graph shows that in the given setup:

- The engine synchronous task RPM_2 experiences a decreased response time at higher engine speeds. This is due to the skipping of functions at engine speeds above 3500 and 5500 rpm.
- A lower priority periodic task (called OS2_20ms) has significant variance in its worst-case response time: Generally its response time will increase at higher engine speeds (due to increased number of interrupts by engine synchronous tasks), but this effect is compensated by the choice of dropping functions. In effect it can be seen that task OS2_20ms has its peak response time at 3500rpm.

This kind of diagram allows the verification of global timing constraints via the quick identification of critical scenarios. Such scenarios can then be investigated in more detail. In addition the diagram highlights scenarios which are overly safe, i.e. where functions are maybe dropped without need.

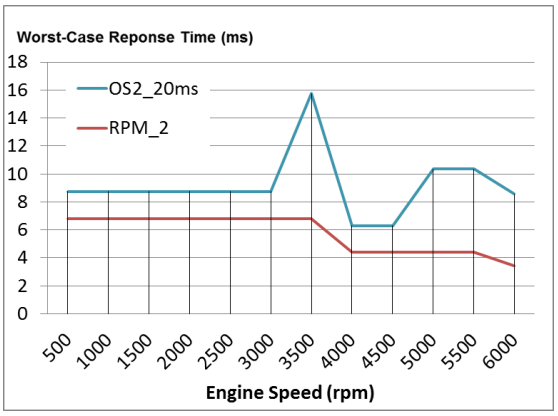

Figure 3 Response times of task RPM_2 and OS2_20ms at different engine speeds.

Complex case for future multi-core applications

As previously discussed, a simple and fast way to treat critical situations in the industrial practice (e.g. in case of increased load at high engine speeds) is to just abort some of the processed system functions in order to permit the safe execution of critical functions. However, even if by implementing severe solutions the safe system functionality can be ensured (e.g. by aborting tasks or by reducing their computational requirements), the resulting service degradation is not convenient and will become unacceptable with the increasing requirements for lower emissions and continued demands for improved fuel economy.

When functions are suddenly aborted, just like in case of processor interrupts, data are lost. Instead of executing such sharp transitions, gradual mode transitions are preferable such that functions can be continued or resumed later.

In what follows we provide an overview on recent progress in the field of scheduling analysis for multi-mode applications. Relying on this, in Section 4 we will further discuss new options for the design of multi-core real-time systems, options that could be applied in order to avoid the service degradation resulting from the overly pessimistic measures applied in the current practice.

3. Multi-mode systems

Similar to the automotive control systems, other real-time systems that can change their functionality over time and execute in different operational modes can be found in different application domains, for example in safety-critical avionic or in multimedia smart devices. Such systems may have to adapt their behaviour during runtime to changing conditions in the environment or in the system itself. To properly handle this behaviour, operational modes and mode change protocols which control the transition between the modes have to be defined.

3.1. Real-time system model

For the next explanations we abstract the dual-core system in Figure 1 and further refer to the system depicted in Figure 4.

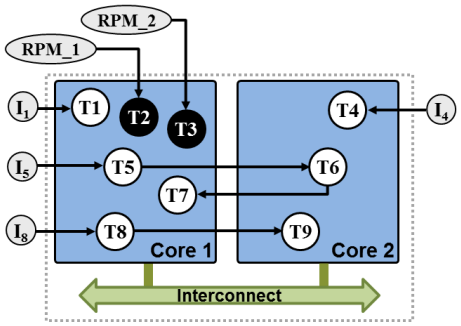

Figure 4 Illustration of a dual-core processor with inter-core communication.

Classic system model in the real-time research

In literature, real-time systems are usually modelled as a set of tasks T = {T1, T2, …, Tn} which are mapped and executed on a set of processing (CPUs) and communication (Busses) resources. Tasks have priorities allocated and their execution is performed according to a scheduling policy. As exemplified above, in the current automotive practice tasks are statically mapped on the processors and are scheduled according to a fixed-priority scheduling, i.e. according to the OSEK/VDX specification [4].

The activation of a task is triggered by an activating event, which may be the result of timer expiration, an external or internal interrupt (I1, I4, I5 and I8 in Figure 4 represent the event sources at the task input), or the result of another task being finished.

In general, tasks are characterized by their execution times, their activation periods and their deadlines, which may be smaller, equal, or larger than the periods. These parameters are expressed in time units.

Extended system model

Engine-synchronous tasks are a special type of periodic tasks, which have been neglected so far by the real-time research community. The recurrences of the engine-synchronous tasks are expressed in engine angle degree rather than time. For example, let's assume that task T2 in Figure 4 is activated each 90^0 (i.e. four time in each rotation) and task T3 each 360^0. In order to obtain a system-wide unique time base, for each fixed engine speed at which an engine-synchronous task is to be specified, the angular recurrence has be transformed in time units. The activation periods of the engine-synchronous tasks T2 and T3 at different fixed engine speed values are given in Table 1.

Table 1 Parameters of the engine-synchronous tasks

Period	Time units (ms) at constant engine-speeds (RPM)										
	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000
P_2	15	10	7.5	6	5	4.28	3.75	3.33	3	2.75	2.5
P_3	60	40	30	24	20	17.14	15	13.33	12	10.9	10

In order to capture more exactly the behaviour of engine-synchronous tasks, the time duration an engine needs to accelerate or decelerate between two discrete engine speed values rpm1 and rpm2 can be modelled with a time interval Δt(rpm1, rpm2) (see Figure 5).

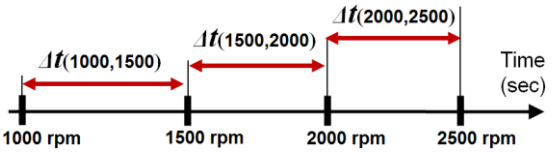

Figure 5 Time intervals between two constant engine-speed values.

In practice, these time intervals depend on the gear, on the current cruising speed and the driving behaviour, and can be obtained e.g. by analysing the acceleration behaviour of a car using test benches [11] or by relying on real field tests. The results in [11] indicate for a particular setup an acceleration phase of 20sec from 1000rpm to 3000rpm in the 4th gear and of 35sec in the 5th gear. Figure 6 depicts a possible scenario during acceleration.

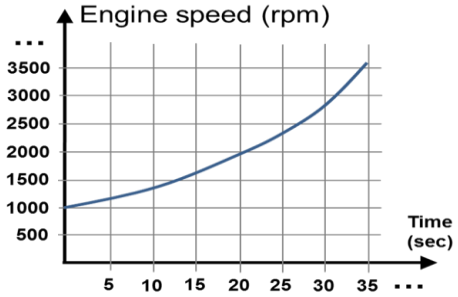

Figure 6 Example of engine-speed variation over time during an acceleration phase.

In order to handle the worst-case timing behaviour, the smallest time intervals, during acceleration and deceleration, between discrete engine speed values (i.e. the highest activation rate) have to be assumed.

3.2. Mode change model

The dual-core system in Figure 4 accommodates several independent periodic tasks, two distributed applications consisting of communicating tasks and two engine-synchronous tasks. Task priorities are indicated by their indices, where the lowest numerical index indicates the highest priority.

In case such systems may execute in different operational modes these can be specified by a finite set M = {M1, M2, …, Mx}. Each mode Mi (Mi ϵ M) is characterized by a different behaviour and is associated with a specific set of tasks (a subset of T) together with its timing properties, e.g. task execution times, priorities and deadlines.

In response to Mode Change Requests (MCR), initiated by the environment or by system internal events, multi-mode systems will experience transitions from an old operational mode (e.g. M1) characterized by

a set of functionalities, to a new operational mode (e.g. M2) characterized by a different or a changed set of functionalities. For example, assume that at runtime a mode change request (MCR) triggers a transition from an operational mode M1 to an operational mode M2 such that task T1 will be removed from Core1 and tasks T5, T6 and T7 will be added on Core1 and Core2.

In literature, the tasks executing in a multi-mode system are categorized depending on their behaviour when a mode change request (MCR) occurs as follows:

(i) *old tasks*, which are immediately aborted when a MCR occurs;
(ii) *finished or completed tasks*, which are present in the old execution mode, but not in the new one. These tasks are allowed to finish their execution during the transition phase which follows the MCR;
(iii) *new or added tasks*, which are either introduced for the first time after the MCR or represent a modified version of old tasks, e.g tasks that change their parameters - execution time or activating event model;
(iv) *unchanged tasks* which are present in both configurations and remain unchanged in their parameters during the transitions.

3.3. Mode change protocols

To control the transition between operational modes designers can opt for *synchronous* or *asynchronous protocols* [12]. The current AUTOSAR specifications related to the mode-management topic indicate the support for the same two types of mode change protocols [5,6,7].

Synchronous and asynchronous protocols differ in the way they handle new and finished tasks during the transition phase which follows a MCR.

Synchronous protocols, as opposed to asynchronous protocols, do not allow new mode tasks to be released until all finished tasks have completed their last activation corresponding to the old mode. In this way synchronous protocols ensure isolation between the execution of mode specific functionalities. Synchronous protocols do not require specific schedulability analysis for the transition phase. However, delaying the start of the new mode applications is not always suitable as this could counter the timing of new mode actions which must be performed as soon as possible (e.g. when switching to an emergency mode).

Asynchronous protocols overcome the limitation of synchronous protocols and allow functions of the new mode to be started simultaneously to the old mode functions. However, increased attention is required, as the execution of functions of both modes generates an increased workload during the transition phase, fact that can lead to timing violations [9,12,13]. Therefore, asynchronous protocols require specific schedulability analysis.

3.4. Challenges and analysis methods

For the system depicted in Figure 4 we assume the mode change from an operational mode M1 to an operational mode M2 such that task T1 will be removed from Core1 (i.e. T1 is a *finished task*) and tasks T5, T6 and T7 will be added to Core1 and Core2 (i.e. T5, T6, T7 are *added tasks*). The rest of the tasks, T4, T8 and T9 represent *unchanged tasks* and execute independent of the mode change.

In case an asynchronous mode change protocol is employed, tasks which belong to both modes will coexist in the system, i.e. all the tasks depicted in Figure 4. The overlapping execution of these tasks generates an increased load, which translates into an increase of the tasks' worst-case response times (WCRTs) - Figure 7.

Relying on different assumptions, several mode change protocols and dedicated analysis methods have been developed by the real-time research community for handling mode transitions in multi-mode single- and multi-processor systems (e.g. [9,12,13,14]). These allow computing task worst-case response times (WCRTs) in each individual operational mode and during the transition phases between every two modes. Figure 7 depicts the worst-case response time behaviour for the lower priority tasks T8 and T9.

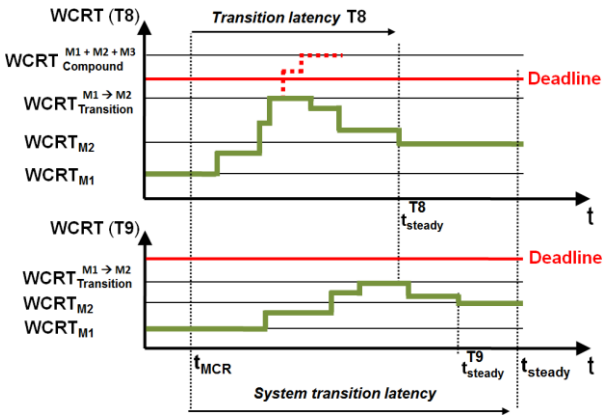

Figure 7 Tasks' worst-case response times. Illustration of a possible settling behaviour as effect of a mode change.

However, most of the existing solutions assume that systems consist of independent tasks, i.e. there is no communication and no precedence constraints between tasks. In [9] and [10] it was shown that in case of distributed applications, the initiation of a mode change has not only a local effect but also impacts the timing of tasks executing on other processors. The mode change leads to a change in the execution and communication demand on a processor (e.g. Core1). As there are tasks which communicate across the processors (e.g. tasks T8 and T9), the transient timing behaviour of tasks on a processor during the transition phase will propagate to the interconnected tasks and will impact the timing of the tasks executing on other processors. This transient

effect, initiated on a core may occur on other core long time after the mode change was performed [9,10]. Therefore, computing only the WCRTs in each individual mode and during every transition between two modes is not enough. The duration of the transition phases has to be computed and considered at design time.

In the example above we have considered that the system could switch only between two modes. However, complex real-time systems might have to switch between multiple modes at runtime. In that case, if a mode change request would trigger the transition from a mode M2 to a mode M3 before the transition phase from mode M1 to mode M2 has been completed, tasks of the three modes (i.e. M1, M2 and M3) could potentially execute at the same time on the cores. In this case the tasks' worst-case response times could increase over the deadlines. This is illustrated for task T8 and T9 in Figure 7 (see dashed red line). Therefore, the system transition latency, i.e. the duration of the system-wide transition phases between two modes, has to be computed in order to avoid the overlap of multiple mode changes that can cause violation of the timing constraints.

A solution that allows deriving mode change transition latencies of multi-mode distributed applications was proposed in [10].

4. New design options for multi-mode applications on multi-core real-time systems

With the advent of multi-core designs, the platform engineer is confronted with different problems, such as how to partition the tasks and runnables efficiently over the complete spectrum of operating conditions, and how to organize data handovers between cores. In addition, as motivated in Section 2, mode changes have to be given better attention in multi-core setups.

With respect to the considered automotive case study, in Section 2 we have shown that the methodology available for timing and performance design can be applied to real-time systems which accommodate tasks with angular recurrence. However, these methods only suit the current automotive practice, where overly pessimistic measures are applies in order to permit the safe system functionality in critical situations (e.g. by aborting tasks or by reducing their computational requirements in case of increased processor load at high engine speeds).

Based on current research results (discussed in Section 3) we propose to map the problem of scheduling real-time applications which accommodate tasks with angular recurrence to the problem of scheduling multi-mode applications on multi-core real-time systems. Relying on this, we next discuss new options for the design and analysis of future multi-core real-time embedded systems.

Design options for multi-core real-time systems with engine-synchronous tasks

In the present example, the task modes are selected based on engine speed. This implies that at a threshold speed (e.g. a speed where the system switches from a high quality to a low quality mode), the system can be in one of two modes, depending on whether the vehicle is accelerating or decelerating. Both situations need to be considered in order to identify the most critical scenario, and to choose threshold values correspondingly. With respect to our case study, the question is:

What are the engine speeds at which mode changes have to be initiated such that (i) the impact on the system's timing is minimum and (ii) the timing constraints are certainly met on all cores?

This question was relatively easy to answer for single-core setups. But, mode changes in multi-core systems imply a more complex behaviour where the load change during execution is not necessarily monotonous.

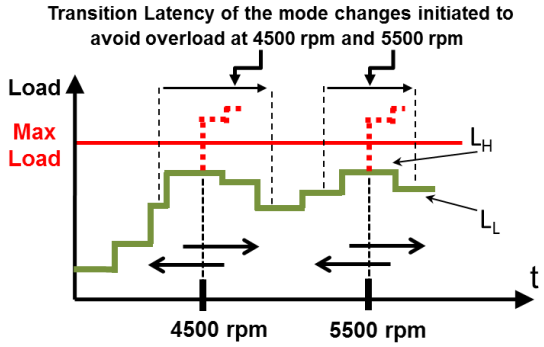

Figure 8 Multiple mode changes in order to avoid overload at different RPM values.

Thus, an analysis is required that allows to quantify the mode change latency and the peak load and task response times during all transitions (illustration in Figure 8). Based on these values, the threshold rpm values can be computed:

(i) When *accelerating*, the mode change has to be initiated (i.e. trigger a mode change request MCR) in sufficient time before the engine speed imposes a non-schedulable situation at a critical point CP rpm. For example, assume task T5, T6 and T7 in Figure 4 have to be dropped-off at 4500rpm (and similarly T8 and T9 at 5500rpm) in order to avoid an overload situation e.g. at the critical point 4600rpm (5600rpm respectively). Instead of just dropping-off these tasks a controlled removal of them should be initiated in enough time before the system reaches a non-schedulable situation.

The *mode change transition latencies* (see [10]), corresponding to the mode changes that consist of stopping tasks, have to be computed for different assumptions regarding the moment of triggering the MCR. The calculation can be performed with the analysis method in [10].

For each critical point CP rpm, the engine speed X rpm (X < CP) will be identified such that the duration of the mode change during acceleration initiated at X rpm (i.e. the mode change transition latency denoted here with $L_A(X)$) is less than the time the engine needs to accelerate from X rpm to CP rpm (i.e. $\Delta t(X,CP)$ - see extended system model in Section 3.1).

(ii) When *decelerating*, the mode change that aims at restarting tasks can only be initiated when the engine speed indicates sufficient headroom in order to allow successful scheduling also during the mode change transition. When decelerating from 5500rpm to 4500rpm a change from a low level load (L_L) to a high level load (L_H) is performed. The previously dropped tasks could be restarted "too early" to each other, fact that would lead to an overlap of multiple mode changes. As indicated in Section 3 this could lead to an overload situation. Thus, the mode change transition latencies have to be calculated (e.g. with [10]) for each required mode change during deceleration.

Furthermore, when decelerating, a mode change that consists of restarting tasks should be initiated only if there is sufficient headroom in order to allow the successful scheduling also in case of a sudden acceleration, i.e. if there is enough time to remove again the tasks before the critical points. Thus, the engine speed Y rpm (Y < X) has to be identified such that the duration of the mode change during deceleration initiated at Y rpm (i.e. $L_D(Y)$) is less than the time the engine needs to suddenly accelerate from Y rpm to X rpm (i.e. $L_D(Y) = L_A(Y) < \Delta t(Y,X)$).

For each critical point CP rpm the timing constraints will not be violated if the system is designed such that mode change transition latencies exhibit a hysteresis around an engine speed X rpm (X < CP) that can be identified as indicated above at (i) and (ii). An example is illustrated in Figure 9 and 10.

Due to mechanical characteristics (e.g. flywheel inertia) the time an engine needs to accelerate or decelerate is large (see Figures 6 and 10) in comparison to the execution time of the functions on the engine control units. Thus, if fast mode changes can be guaranteed, mode change protocols can be employed in order to avoid the service degradation resulting from the overly pessimistic measures applied in the current practice.

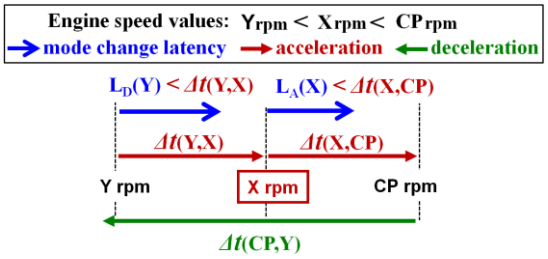

Figure 9 Mode changes shall be initiated at Xrpm during acceleration and at Yrpm during deceleration in order to avoid a non-schedulable situation at CPrpm.

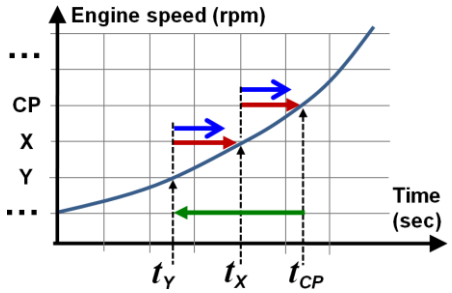

Figure 10 Complex mode changes are possible if there is enough headroom for mode change transition latencies.

However, as shown in Section 3, the timing behaviour of complex multi-mode applications on multi-core processors requires increased attention. Analysis methods have to be used for computing mode change transition latencies in order to ensure the safe system functionality.

5. Conclusion

In this paper we highlighted the implications of multi-mode applications and scenario-dependent behaviour on the system's timing. An automotive specific use case was presented that demonstrates the need for tool supported timing analysis methods. We provided an overview on recent progress in the field of scheduling analysis. Current analysis methods allow taking into account the mode transition latency and thus enable the safe provisioning of multi-mode applications in multi-core environments in the future.

Even though the explanations rely on automotive specific use cases the challenges highlighted here apply, in specific variations, also to other application domains e.g. avionics.

References

[1] Freescale, "Rationale for Multicore Architecture in Auto Apps," *Freescale Technology Forum*, June 2011. *[Online]* http://www.freescale.com/files/training_pdf/WBNR_FTF11_AUT_F0166.pdf

[2] AbsInt, aiT WCET Analyser, *[Online]* http://www.absint.com/ait/

[3] Symtavision GmbH, SymTA/S tool. *[Online]* http://symtavision.com/symtas.html

[4] OSEK Consortium, "OSEK OS Specification v2.2.3," *[Online]* http://www.osekvdx.org/, February 2005.

[5] AUTOSAR GbR, "Specification of RTE v3.0.0", *[Online]* http://www.autosar.org/

[6] AUTOSAR GbR, "Virtual Functional Bus v2.0.0", *[Online]* http://www.autosar.org/

[7] AUTOSAR GbR, "Specification of Basic Software Mode Manager v1.1.0" *[Online]* http://www.autosar.org/

[8] AUTOSAR GbR, "Specification of Multi-Core OS Architecture v1.0.0," *[Online]* http://www.autosar.org/

[9] R. Henia and R. Ernst, "Scenario Aware Analysis for Complex Event Models and Distributed Systems", *In Proc. of the Real-Time Systems Symposium*, Dec. 2007.

[10] M. Negrean, M. Neukirchner, S.Stein, S.Schliecker and R. Ernst, “Bounding Mode Change Transition Latencies for Multi-Mode Real-Time Distributed Applications“, 1*6th IEEE Conf. on Emerging Technologies and Factory Automation*, Sept. 2011.

[11] P. Podevin, G. Descombes, P. Marez, and F. Dubois, “A study of turbocharged Diesel engine during sudden acceleration. Set up and exploitation of a specific test rig.” *in Internal Combustion Engine Division of ASME*, Oct. 1999.

[12] J. Real and A. Crespo, “Mode Change Protocols for Real Time Systems: A Survey and a New Proposal”. *Real-Time Systems*, 26(2):161–197, March 2004.

[13] K. W. Tindell, A. Burns, and A. J. Wellings, “Mode Changes in Priority Pre-emptively Scheduled Systems,” *in Proc. of the Real-Time Systems Symposium*, 1992, pp. 100–109.

[14] P. Yomsi, V. Nelis and J. Goossens, “Scheduling Multi-Mode Real-Time Systems upon Uniform Multiprocessor Platforms”, *15th IEEE Conference on Emerging Technologies and Factory Automation*, Sept. 2010.