Computational Analysis of Complex Real-Time Systems - FMTV 2016 Verification Challenge

Ingo Stierand, Philipp Reinkemeier, Sebastian Gerwinn, Thomas Peikenkamp
OFFIS, Oldenburg, Germany
{stierand,reinkemeier,gerwinn,peikenkamp}@offis.de

***Abstract*—Real-time scheduling analysis is an important step in safety relevant embedded system design for many application domains, such as avionics, automotive and automation. Increasing system complexity, not least due to raising automated mobility, requires constant evolution of the analysis approaches, resulting in a vital research domain.**

We like to contribute to the research by presenting a computational analysis approach, where the system model is unfolded as discrete-time state transition system. The analysis engine is tailored particularly for real-time scheduling analysis and exploits respective optimisations. We show the applicability of the approach on an industrial relevant problem, and discuss its advantages and limits.

I. Introduction

The question whether a system can deliver its functions timeliness when deployed on a hardware architecture is an integral part of the safety aspect of embedded system design for many application domains, such as avionics, automotive and automation. Real-time scheduling analysis is a well established discipline, providing a wealth of methods to verify relevant timing aspects of the deployed system. Most approaches are based on also well-established models, so called *task networks*, and differ mainly in details that reflect the focus and capabilities of the underlying mathematical method.

Although the discipline exists for several decades, publicly available benchmarks were rather rare for a long time. People sporadically came up with real-world or carefully designed artificial examples [9], [6]. Such models help the community to compare their methods, to investigate the individual advantages (and disadvantages), and to evolve the approaches.

Recently, a group of researchers came up with the idea of a verification challenge, where particularly timing analysis problems are made public, and invited all interested parties to try their approaches and to discuss the results. We believe this is a very good idea, which is proven to be an effective instrument for progress in other formal verification communities.

We would like to contribute to this effort by providing analysis for a system model that is derived from a large real-world application. The authors of [5] constructed a generator from a anonymised engine control application with thousands of functions, which can be parametrised in order to obtain appropriate benchmarks.

The work has been partially funded by the German Ministry for Education and Research (BMBF) under the funding ID 01IS14029H (AMALTHEA4public) and ID 01IS15031H (ASSUME)

The present system is given as an AMALTHEA4public[1] (A4P) model, and can be downloaded from the WATERS workshop website[2]. On this model, we apply a model-checking based analysis [7]. In contrast to other existing, more general frameworks like timed automata, our approach is based on the idea to construct a highly specialised model-checking engine particularly tailored for real-time scheduling analysis. This has been done before, e.g., with the TIMES tool [1]. Our approach however exploits discrete-time state space construction, using a variant of time darts [4] in order to reduce the footprint of state-space representation.

We briefly discuss the system model, and how we interpret it where needed, in the following section. Section III introduces the analysis approach and details how we tackle the verification challenge. Results are presented in Section IV, followed by a summarising discussion in Section V. Section VI concludes the paper.

II. System Model and its Interpretation

The system of the verification challenge consists of a multi-core processing unit with four identical cores, which are connected to a crossbar switch, and five memory banks. All system components are running at 200 MHz, resulting in a length of 5 ns for a processing cycle.

Every core is directly connected to its local memory bank. Additionally, the cores can access all other local memory banks as well as the global memory bank via the crossbar switch, however, at the cost of additional cycles. The switch, as stated in the challenge call [3], provides full connectivity. We interpret this such that no congestion occurs at the switch due to concurrent memory accesses from different cores. The switch imposes 8 cycles latency on a memory access. The challenge call further states that accesses to the memory banks are serialised using a FIFO strategy. This is also true for the local memories; all accesses from the local core as well as from other cores via the switch go to the same FIFO buffer. Every memory access takes 1 cycle.

The system consists of 21 tasks. Each task contains multiple runnables, which are executed sequentially, as the call graphs in the model indicate. All runnables consist of a similar set of runnable items, which is (1) a sequence of read accesses to various memory cells (labels), (2) execution of an instruction

[1]http://www.amalthea-project.org/
[2]https://waters2016.inria.fr/challenge/

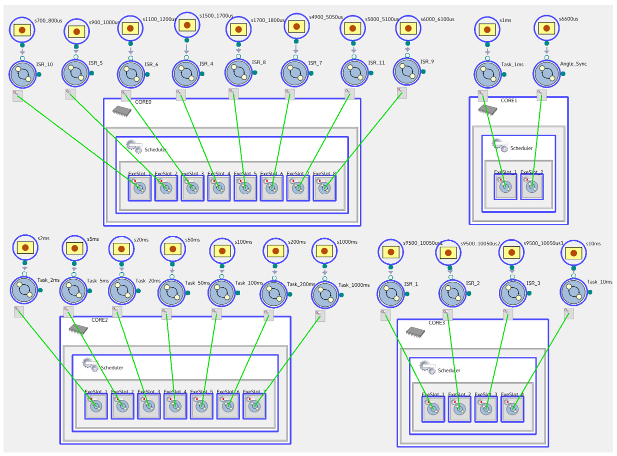


Fig. 1. Analysis Model (all four cores with associated tasks, buses and memories are omitted)

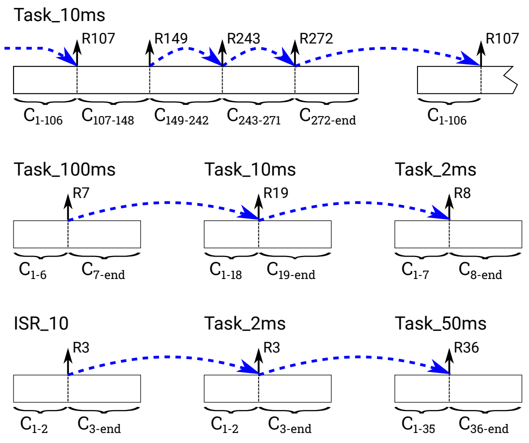


Fig. 2. Three Effect Chains of the Challenge Model

sequence abstracted by a probability distribution, and (3) a sequence of write accesses to memory cells, in this order. We interpret the runnable items as a sequence, i.e., read and write accesses to the labels are performed one after the other.

The tasks are allocated to the four cores as shown in Figure 1. Each core executes an operating system that schedules tasks according to the OSEK standard. To this end, tasks get priorities in ascending order, with 0 being the lowest priority. While these priorities are globally defined, task allocation induces unique core-local priority orders. All except five tasks are preemptively scheduled (cf. Table I). The remaining tasks are cooperative. Preemptive tasks can preempt lower priority tasks – no matter whether preemptive or cooperative – at any point in time. Cooperative tasks can be preempted by higher priority cooperative tasks only at runnable boundaries. Note that, while the A4P meta-model defines so called *schedule points* where cooperative tasks can be preempted, the authors of the challenge explicate it otherwise.

All tasks are activated by individual stimuli. The A4P modelling framework defines various kind of stimuli. Two types are used for the model, namely periodic and sporadic. According to the documentation, periodic stimuli are defined by two parameters. The *offset* defines the time of the first task activation after system initialisation. The *recurrence* parameter defines the activation period relative to the first one. Sporadic stimuli are defined by a probability distribution defining the minimum and maximum inter-arrival time for task activations. All sporadic stimuli in the model are defined by a uniform distribution with lower and upper bound. We assume, as stated by the authors of the challenge, that also for sporadic stimuli the first event occurs at time 0.

The model finally contains three effect chains as shown in Figure 2, which define data flows that can be observed in the system. All events referred to by the chains are start events of runnables. The first chain refers to a sequence of runnables that all belong to task Task_10ms, which is (names are abbreviated) {R149,R243,R272,R107}. A further inspection of the model reveals that theses runnables indeed access common memory labels; every runnable of the chain sequence (except the last) writes to a label that is read by the subsequent runnable. As the runnables of the task are executed in ascending numbering order, the resulting data flow is spread over two subsequent executions of the task, which is indicated in the top part of Figure 2.

The other two are cross-core effect chains. The second chain involves tasks Task_100ms, Task_10ms and Task_2ms, which are allocated to Core 2 and Core 3, respectively. The third chain also crosses two cores, Core 0 (ISR_10) and Core 2 (Task_2ms and Task_50ms).

The challenge states three sub-challenges. All of them ask for tight lower and upper bounds of the end-to-end latencies of the three effect chains. The first effect chain for example is asking for lower and upper bound of the time between the start events of runnables R149 and R107. The first sub-challenge states that all memory accesses shall be ignored. The other two state that memory accesses should be taken into account, which induces additional latencies due to congestions for memory accesses. Concerning the second sub-challenge, we assume that the labels are allocated to the memory banks as defined in the model. The third sub-challenge asks for an allocation of the labels such that the end-to-end latencies become minimal. We do not cope with this optimisation challenge in the present paper.

III. Analysis Approach

In order to keep analysis times and (memory) space manageable, we follow a compositional approach, where we consider the challenge as a set of separate scheduling problems. To this end, we re-model every core and its allocated tasks in terms of our analysis model as exemplified in Figure 3. The top part of the figure shows the relevant artefacts, namely *event sources* (yellow boxes), *tasks* (blue circles) and *processing units* (grey boxes). Event sources emit events according to their assigned event stream behaviour, which is defined by four parameters P^-, P^+, J and O. The time between any two subsequent events is selected non-deterministically as follows: Given time instants $t'_{i+1} \in t'_i + [P^-, P^+]$ the event source emits events at

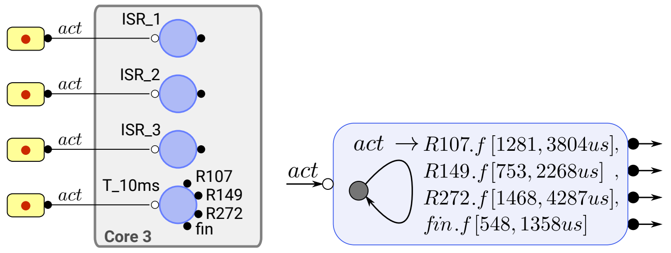

act
act
act
act
ISR_1
ISR_2
ISR_3
T_10ms
R107
R149
R272
fin
Core 3
act
$act \rightarrow R107.f\,[1281, 3804us]$,
$R149.f\,[753, 2268us]$,
$R272.f\,[1468, 4287us]$,
$fin.f\,[548, 1358us]$

Fig. 3. Analysis Model Example

time points $t_i \in t'_i + [0, J]$. Generally, the first time instant is chosen non-deterministically from the interval $[O, O + P^+]$, i.e., $t'_1 \in [O, O + P^+]$.

Event sources are hence sufficient to model periodic as well as sporadic stimuli of the challenge model. As the stimuli defined in A4P send their first event at a fixed offset (in the model always 0), we have to remove the initial non-determinism of the corresponding event sources in the analysis model. This is obtained by command line parameters of the analysis tool.

Tasks are activated by incoming events (here always act) via their input ports (small white circles), and emit events during execution via their output ports (small black circles). Task execution finishes with the last emitted event. Tasks must emit at least one event. Tasks have internal state transition systems as shown at the right part of Figure 3. Depending on the internal state of a task and the event that activates it, the tasks execution is performed according to the annotation of the corresponding transition. For example, if the task in the figure is activated by an incoming act event, it executes the corresponding transition, which is a loop at the sole task state in Figure 3. During execution, it sends event f to output port R107 after $1281 - 3804$ μs "consumed" execution time, an event to output port R149 after $753 - 2268$ μs, and so on. The task finishes execution with the last sent event.

The analysis model allows for two interpretations of the execution times annotated at a transition. For *simultaneous* transitions, the execution times for the individual output events pass simultaneously. The output order of events with overlapping execution time intervals is non-deterministic. For *sequential* transitions, the execution times pass sequentially. At the end of each execution time, the corresponding event is emitted. Only sequential transitions where used for the challenge.

The models used for analysis of the challenge have been manually constructed. For the calculation of the execution times however a simple parser tool has been implemented. While a fully automatic translation would be possible, we avoided the additional effort for the present work. The resulting analysis model is depicted in Figure 1. The minimal and maximal execution times obtained for the tasks, or task segments, from the original model by summing up the execution times of the involved runnables are depicted in Table I with descending (core-local) priority order from top to bottom. In order to enable calculation of bounds for the

TABLE I
TASK PARAMETERS (TIMES W/O MEMORY ACCESSES IN # CYCLES)

Core	Task	preempt.	min	max
0	ISR_10	yes	3.363	6.068
0	ISR_5	yes	25.825	51.636
0	ISR_6	yes	2.980	6.190
0	ISR_4	yes	33.242	73.160
0	ISR_8	yes	26.089	60.777
0	ISR_7	yes	34.678	64.974
0	ISR_11	yes	27.629	61.177
0	ISR_9	yes	35.617	74.097
1	Task_1ms	yes	50.035	152.870
1	Angle_Sync	yes	260.919	761.071
2	Task_2ms	yes	27.748	80.817
2	Task_5ms	yes	73.108	186.363
2	Task_20ms	no	721.008	2.093.688
2	Task_50ms	no	262.830	616.897
2	Task_100ms	no	625.239	1.883.595
2	Task_200ms	no	14.041	27.697
2	Task_1000ms	no	13.610	27.432
3	ISR_1	yes	3.075	7.011
3	ISR_2	yes	2.064	3.549
3	ISR_3	yes	2.424	4.787
3	Task_10ms	yes	797.773	2.342.546

end-to-end latencies, the respective tasks have been modelled using sequential transition executions as shown in Figure 2. For the first effect-chain, task Task_10ms contains a sequential transition with five execution times. The first one subsumes the execution of runnables with numbers 1 to 106 of the tasks call graph. The task contains a corresponding output port R107 at which the start event for runnable 107 can be observed. The second execution segment subsumes the execution of runnables 107 to 148, for which port R149 indicates start of runnable 149, and so on.

During modelling, we made two notable observations. First, execution times for runnables are expressed in terms of Weibull distributions, which express probabilities for particular execution times. The values in the model are no hard bounds, but define an interval with a certain probability mass, which in our case is $1 - 5 \cdot 10^{-4}$ for all runnables. The definitions imply that there is a non-zero (although potentially very small) probability for each runnable to have very large execution times, which may lead to overload situations where tasks would miss every given finite deadline. Hence, from a safety point of view, the system has to be rejected.

Secondly, we observed that the utilization of three cores (1, 2 and 3) is larger than 100%. For a simple fixed-priority scheduling, this would result in an infeasible task set that cannot be scheduled. The A4P model however defines an OSEK scheduling scheme for all cores, and a limit of one for the maximum number of activations for each task. The model hence implies (considering the OSEK specification) that for each activated task all further activations of this task are ignored until it finishes it execution.

We exploit a model-checking approach for analysing the effect chains, which is implemented in the tool RTANA$_2$ (cf. footnote 3). It is fed with an analysis model and performs a discrete-time state unfolding, resulting either in a closed state-transition system, or terminates if it detects an infeasible scheduling situation, such as a buffer overflow. After state space construction, the tool performs a path analysis in order to obtain the exact minimal and maximal latencies for the respective effect chain. For further details about the approach the reader is referred to [7].

For the verification challenge, we deal with the state-space explosion problem in three ways. First, we introduce abstractions where needed by increasing the length of discrete time slots, at which scheduling decisions occur. The effect is similar to the so-called tick scheduling [8]. Additionally, we exploit the model characteristics where possible to perform compositional analysis. Foremost, we consider the cores separately. If this does not sufficiently reduce the state space, we incorporate analytic methods to obtain response times for individual tasks. The results are fed back to the computational analysis, indeed introducing additional over-approximations. A detailed discussion of the analysis and their results is given in the following section.

IV. Results

As stated above, we took a number of measures to tackle the problem of state space explosion. Although the analysis exploits some symbolic representation of time, a main factor for the resulting memory footprint is the length of discrete time slots. With respect to the model, a suitable slot length would be $5ns$. Due to the characteristics of the challenge model with its large execution time intervals, this leads to very large state spaces. Hence, we decided to set the slot length to $1\mu s$, which indeed results in an over-approximate analysis. In order to still obtain safe approximations, we adjusted the execution times accordingly: for lower bounds we took the floor, and for upper bound the ceiling. More precisely, we calculated an interval $[l', u']$ of $1\mu s$ slots from execution time interval $[l, u]$ such that $l' = \lfloor \frac{l}{200} \rfloor$ and $u' = \lceil \frac{u}{200} \rceil$.

To further reduce analysis effort, we also constructed individual analysis models for the various sub-problems. For example, two models have been constructed for the second effect chain, where only relevant parts of the original model remain. This includes to sum up the execution times of runnables that are not relevant for the particular analysis task.

The following sections discuss the individual approaches for the sub-challenges. The analysis models and result logs are also publicly available[3].

A. Sub Challenge 1 - First Effect Chain

The first effect chain does not involve further abstraction as it involves only a single task running on Core 3. The model used for analysing respective latency bounds is depicted in Figure 3. The results in Table II for the first effect chain also show the individual task response times obtained with the analysis.

Scenarios for the results are depicted in Figure 4. The lower bound corresponds to the situation where two subsequent

[3]https://vprojects.offis.de/rtana

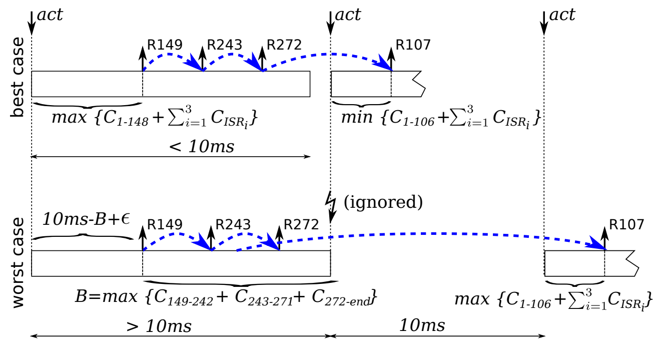

act
act
act
best case
R149
R243
R272
R107
max {$C_{1\text{-}148} + \sum_{i=1}^{3} C_{ISR_i}$}
min {$C_{1\text{-}106} + \sum_{i=1}^{3} C_{ISR_i}$}
< 10ms
(ignored)
worst case
10ms-B+ϵ
R149
R243
R272
R107
B=max {$C_{149\text{-}242} + C_{243\text{-}271} + C_{272\text{-}end}$}
max {$C_{1\text{-}106} + \sum_{i=1}^{3} C_{ISR_i}$}
> 10ms
10ms

Fig. 4. Scenarios for Sub-Challenge 1 - Effect Chain 1

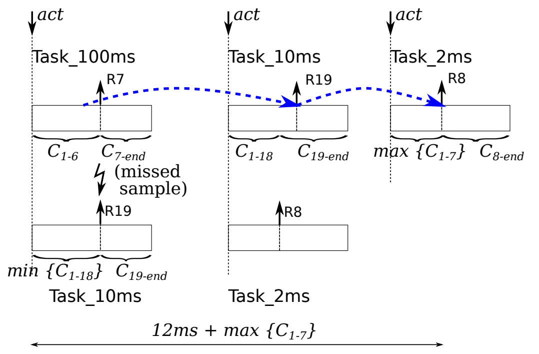
activations of task Task_10ms occur, all runnables R1 – R148 and interrupt service routines ISR_1 – ISR_3 of the first activation consume their maximum processing time, and in the second activation R1 – R106 and ISR_1 – ISR_3 consume their minimum processing time.

A scenario for the upper bound is shown at the bottom of Figure 4. Here, all runnables starting from R149 of the first activation consume their maximum execution time. The overall task execution is slightly longer than 10 ms, resulting in ignoring the subsequent task activation (OSEK task activation limit). Runnables R1 – R106 and interrupt service routines ISR_1 – ISR_3 of the third activation consume their maximum execution time as well.

B. Sub Challenge 1 - Second Effect Chain

In order to avoid state space explosion when analysing the second effect chain, we exploit (1) a compositional analysis approach in combination with an analytical analysis method implemented by pyCPA [2], and (2) a trick. While the first and last task of the effect chain are executed on Core 2, the intermediate task Task_10ms is executed on Core 3. The analysis is done in three steps. First, we obtain time bounds for execution of task Task_10ms from its activation up to the start event of runnable R19. Second, we create a 'placeholder' task Task'_10ms with execution time bounds according to the results of the first step. The task is not allocated to Core 2, causing the analysis to assume a distinct processing resource solely assigned to the task, which hence executes without any interferences. This way, the model provides a safe over-approximation for imposed data flow latencies on the effect chain. The same approach is applied to obtain time bounds for execution of task Task_100ms from its activation up to the start of runnable R7. To obtain these bounds we setup a pyCPA model with all tasks from Core 2 having a higher priority than Task_100ms. Again, we modelled a placeholder task Task'_100ms based on these results.

Concerning the „trick", cooperative scheduling as in the challenge can be considered as temporal priority inversion. The maximum length of the inversion is no longer than the highest maximum execution time among all runnables of lower-priority tasks. This time is added to the execution time of Task_100ms, which is again a safe over-approximation of the actual behaviour.



act
Task_100ms
act
Task_10ms
act
Task_2ms
R7
R19
R8
$C_{1\text{-}6}$ $C_{7\text{-}end}$
$C_{1\text{-}18}$ $C_{19\text{-}end}$
max $\{C_{1\text{-}7}\}$ $C_{8\text{-}end}$
(missed sample)
R19
R8
min $\{C_{1\text{-}18}\}$ $C_{19\text{-}end}$
Task_10ms
Task_2ms
12ms + max $\{C_{1\text{-}7}\}$

Fig. 5. Worst case scenario for Sub-Challenge 1 - Effect Chain 2

The first two analyses for the second effect chain in Table II depicts the response time bounds for Task_10ms from activation until the start of runnable R19 and for Task_100ms from activation until the start of runnable R7.

A scenario for the upper bound is shown in Figure 5. Runnables R1 – R18 of task Task_10ms consume their minimum processing time. Runnable R19 is just started before runnable R7 of task Task_100ms. So the effect influences the next start of R19 in the next job of Task_10ms. Since $min\{C_{1-18}\} > max\{C_{1-7}\}$, the job of task Task_2ms cannot sample the data of the job of task Task_10ms starting at the same time. Thus, the execution of R8 in the next job of Task_2ms samples that data. In that job of Task_2ms the runnables R1 – R7 consume their maximum processing time.

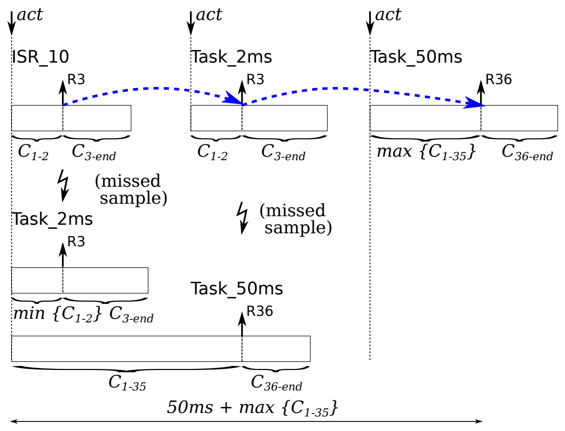
The calculated lower bound of 0 is due to the compositional approach where task dependencies are lost. Therefore, the start events of involved runnables can occur at the same time instant, and the analysis has to assume that they might occur in the order R7 $\rightarrow$ R19 $\rightarrow$ R8 with no delay inbetween.

C. Sub Challenge 1 - Third Effect Chain

For the third effect chain, we apply a similar approach as for the second one. This time we insert placeholder tasks ISR_10 and Task'_50ms. Again we use pyCPA to obtain the bounds for execution of Task_50ms up to the start of its runnable R36. ISR_10 however is the highest priority task running on Core 0. Hence, it is sufficient to model this task without a resource, but with its core execution times. Concerning the „trick", time is added to the execution time of Task_50ms instead of Task_100ms, which is the maximum execution time among all runnables of Task_100ms, Task_200ms and Task_1000ms.

A scenario for the upper bound is shown in Figure 6. Runnables R1 – R2 of task Task_2ms consume their minimum processing time. Runnable R3 is just started before runnable R3 of ISR_10. So the effect influences the next start of R3 in the next job of Task_2ms. Here again a data sample might be missed and the invocation of runnable R36 in the next job of Task_50ms results in the worst case scenario for the third effect chain. In that job of Task_50ms the runnables R1 – R36 consume their maximum processing time.

The lower bound of the effect chain is 0 for the same reasons as for the lower bound of the second effect chain.



act
ISR_10
act
Task_2ms
act
Task_50ms
R3
R3
R36
$C_{1\text{-}2}$ $C_{3\text{-}end}$
$C_{1\text{-}2}$ $C_{3\text{-}end}$
max $\{C_{1\text{-}35}\}$ $C_{36\text{-}end}$
(missed sample)
Task_2ms
(missed sample)
R3
min $\{C_{1\text{-}2}\}$ $C_{3\text{-}end}$
Task_50ms
R36
$C_{1\text{-}35}$ $C_{36\text{-}end}$
50ms + max $\{C_{1\text{-}35}\}$

Fig. 6. Scenarios for Sub-Challenge 1 - Effect Chain 3

TABLE II
RESULTS FOR SUB-CHALLENGE 1

	Latency Bound/ BCRT,WCRT	Analysis Time	Analysis Space
ISR_1	[15, 36] μs		
ISR_2	[25, 54] μs		
ISR_3	[37, 78] μs		
Task_10ms	[4.024, 11.871] μs		
Effect Chain 1	[5.105, 19.524] μs	49 s	2 GiB
Task'_10ms	[315, 831] μs	41 s	1 GiB
Task'_100ms	[99, 39.890] μs	1 s	22 MiB
Effect Chain 2	[0, 11.826] μs	87 s	400 MiB
ISR_10	[13, 25] μs	–	–
Task'_50ms	[975, 39.029] μs	1 s	22 MiB
Effect Chain 3	[0, 89.015] μs	56 s	400 MiB

D. Sub Challenge 2

The second sub-challenge states that memory accesses of runnables shall be taken into account. A comprehensive analysis would calculate exact bounds on the latencies imposed by concurrent such accesses from tasks running on different cores. As this is currently infeasible by our analysis, we have to calculate safe approximations. This is however simple, as all variables are mapped to the global memory bank. Hence every memory access can be delayed by up to three other memory accesses (from other cores). This results in an overall latency of every memory access between 9 and 12 cycles. Based on these adapted runnable execution times, the analysis then follows the same scheme as for sub-challenge 1. The results are shown in Table III

An interesting result is that the worst-case reaction time of the second effect chain becomes lower when taking memory accesses times into account. This is because the best-case execution time of runnables R1 – R18 of Task_10ms increases, resulting in a smaller time distance to the final reaction of the effect chain.

E. Probabilistic Aspects

It is worth mentioning that the results we reported in this section are only valid with a certain probability. This is due to the fact that the execution times of the different runnables are subject to random fluctuations. Within the model given

TABLE III
RESULTS FOR SUB-CHALLENGE 2

	Latency Bound/ BCRT,WCRT	Analysis Time	Analysis Space
ISR_1	[16, 37] μs		
ISR_2	[27, 56] μs		
ISR_3	[40, 82] μs		
Task_10ms	[4.247, 12.171] μs		
Effect Chain 1	[5.042, 19.782] μs	62 s	2 GiB
Task'_10ms	[332, 854] μs	51 s	1.5 GiB
Task'_100ms	[104, 99.223] μs	1 s	22 MiB
Effect Chain 2	[0, 11.811] μs	848 s	750 MiB
ISR_10	[14, 26] μs	–	–
Task'_50ms	[995, 39.628] μs	1 s	22 MiB
Effect Chain 3	[0, 89.613] μs	54 s	400 MiB

for the challenge, these fluctuations are characterised by a Weibull distribution. Specifically, the individual upper and lower bounds on the execution times, which we used in this section, mark intervals of execution times containing a probability mass of $1 - 5 \cdot 10^{-4}$. From this we can derive a lower bound on the probability that the computed bounds hold. More precisely, the computed bounds hold, if the execution times of all runnables with random execution times fall into their respective intervals. As the individual fluctuations are assumed to be independent, this probability is given by $(1-5\cdot10^{-4})^{1250}$. However, this is a rather pessimistic bound, as the latency bounds could still hold, even if one or more individual execution times lie outside of the intervals used.

V. Discussion

The AMALTHEA4public project aims at defining a comprehensive meta-model for real-time systems with focus on the automotive domain, being compliant with AUTOSAR where possible. Tasks, for example, may contain call graphs, which precisely define execution ordering of the runnables within the tasks, as well as their internal behaviour in terms of runnable items. From this point of view the model was easy to understand. However, there is still room for interpretation.

First, it was an effort to retrieve the exact semantics of stimuli. While the documentation of the A4P meta-model defines precisely the semantics of periodic stimuli, definition of sporadic stimuli is rather sloppy, and required clarification by the challenge authors.

The second obstacle was the interpretation of cooperative tasks. The OSEK standard defines various configurations, resulting in different preemption scenarios for the entire task set. It looks like the A4P meta-model either misses documentation of the chosen interpretation or some bits of information allowing to select the intended one. Furthermore, the A4P meta-model defines the particular type 'schedule point' of runnable entity in order to explicitly define code positions where cooperative tasks can be preempted. While no such entities are defined in the model, the challenge authors state that they should be implicitly assumed to exist.

In summary, it took about a day work, including reviewing documentation, to understand model semantics as precise as required for the analysis. No less time was required to set up the analysis models. The main issue here was to find suitable abstractions such that the analysis would fit into the available memory space. While it would be possible to construct a comprehensive analysis model also including memory accesses for the entire system, it was clearly impossible to get analysis results for such model in reasonable time and space. Particularly memory accesses and preemption with cooperative scheduling involved significant effort in tailoring the models. As this is indeed somehow unsatisfactory, it shows some deficiencies of the current analysis, and points towards potential directions for further improvements, such as improved combination of analytic and computational analysis.

VI. Conclusion

We presented a computational analysis approach for the verification of timing characteristics of a non-trivial model that is based on a real-world engine control application. As always with computational approaches, the analysis soon starts to suffer from state-space explosion for „interesting" system sizes. However, as the analysis engine is particularly designed for dealing with real-time scheduling problems, it already shows nice performance compared to generic model-checking approaches such as timed automata or SAT-based engines.

The model presented for the verification has some interesting properties that are hard to encode with classical analytic real-time scheduling approaches. We are convinced that computational approaches can provide valuable results in such cases. We strongly believe that this line of research is still in its infancy and has much potential for further improvements. Real-world problems such as the present are highly useful in order to find sweet spots for such evolution.

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In *In Proc. of FORMATS'03, number 2791 in LNCS*, pages 60–72. Springer-Verlag, 2003.

[2] J. Diemer, P. Axer, and R. Ernst. Compositional Performance Analysis in Python with pyCPA. In *3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2012.

[3] A. Hamann, D. Ziegenbein, S. Kramer, and M. Lukasiewycz. FMTV 2016 Verification Challenge. Robert Bosch GmbH Corporate Research, Germany.

[4] K.Y. Jørgensen, K.G. Larsen, and J. Srba. Time-Darts: A Data Structure for Verification of Closed Timed Automata. In *Proc. of the 7th International Conference on Systems Software Verification (SSV)*, volume 102 of *EPTCS*, pages 141–155. Open Publishing Association, 2012.

[5] S. Kramer, D. Ziegenbein, and A. Hamann. Real World Automotive Benchmarks For Free. In *Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS)*, 2015.

[6] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and G. Harbour. Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems. In *Proc. Conference on Embedded Software (EMSOFT)*, 2007.

[7] I. Stierand, P. Reinkemeier, T. Gezgin, and P. Bhaduri. Real-Time Scheduling Interfaces and Contracts for the Design of Distributed Embedded Systems. In *Proc. International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2013.

[8] K. W. Tindell, A. Burns, and A.J. Wellings. An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks. *Journal of Real-Time Systems*, 6(2):133–151, 1994.

[9] K.W. Tindell, A. Burns, and A.J. Wellings. Allocating hard real-time tasks: An NP-Hard problem made easy. *Real-Time Systems*, 4:145–165, 1992. 10.1007/BF00365407.