FMTV 2016: Where is the Actual Challenge?

Alessio Balsini, Alessandra Melani, Pasquale Buonocunto, Marco Di Natale
Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: {alessio.balsini, alessandra.melani, pasquale.buonocunto, marco.dinatale}@sssup.it

***Abstract*—The FMTV challenge has been formulated and proposed to research groups as a case study and benchmark to compare different analysis methods for real-time multicore fuel injection applications. The nature of the problem is clear enough and the challenge can be likely met by a set of conventional analysis techniques (at least at the current level of description). However, the formulation of the problem and its practical solution are more than likely to reveal a number of additional issues that go from the model of the application, to analysis techniques that consider with much better precision the details of the HW platform, to the need for synthesis and optimization methods.**

I. Introduction

The FMTV 2016 challenge consists of a timing analysis problem in which the AUTOSAR model of a set of cooperating tasks in a fuel injection application is deployed onto a 4-core platform. The objective of the challenge is to apply different analysis methods (worst-case, simulation-based and possibly stochastic) to models of the system with an increasing level of accuracy with respect to the memory placement of communication variables. At the simplest level, memory access times are simply neglected; next, different access times are assumed under the hypothesis of global or local memory allocation; and, finally, the problem of optimizing the placement of the memory items is presented.

True to the spirit of the description, we tackled the objectives of the challenge in a sequence, and because of timing constraints, at the time of this submission, only the results for some of the early activities were available. However, we believe that in the case of this challenge, the experience gathered along the path is at least as valuable as the final solution, and we found several issues that are worth discussing, beyond the presentation of the tool architecture that was used to derive the solution and the hard data that we computed as a result of the analysis.

From the architecture standpoint, we attempted two solutions to the problem: to simulate the time behavior using a scheduling simulator that was previously available at our laboratory, and to analyze the task-set for its worst-case behavior, using a set of formulas derived from the problem description and obtained by adaptation of classical results.

We provide the results of these two analysis methods (with an additional discussion on how to tackle the memory access time problem), but we also believe several issues are worth discussing. Among those:

The **definition of response times when the system contains chains of tasks or runnables communicating asynchronously**. The challenge refers to a set of definitions (reactive and age) for which an application-level justification is not clear enough and for which (despite being formally presented in [1]) a solution in analytical closed form or as an algorithm has never been presented and validated in a peer-reviewed paper.

Next, while the challenge has the merit of restoring to the foreground the consideration of hardware features and issues, its **description of the HW architecture details** is still incomplete and simplistic. For example, the FIFO arbiter controlling accesses to shared memory is likely to be integrated within the crossbar or possibly placed after it, but this information can only be guessed and would affect the access times to memory.

Finally, and most important, the problem probably placed too much emphasis on the analysis part and seems to neglect **the runnables placement problem**, which is most likely the most relevant design issue for a system like this.

II. System Model and Notation

The challenge model is in large part compliant with the AUTOSAR metamodel and adopts from it definitions and most of the semantics for activation and communication of functions (runnables in AUTOSAR). An attempt at the formal characterization of the challenge model is the following.

A task τ_i is composed of an ordered sequence of n_i runnables $\rho_{i,1}, \ldots, \rho_{i,n_i}$, each of which has its execution time defined as a statistical distribution $\mathcal{C}_i$, which is defined as a truncated Weibull distribution for most if not all the runnables in the model. For the purpose of worst-case analysis, the worst-case execution time (WCET) $C_{i,j}$ and a best-case execution time $c_{i,j}$ may be computed from the distribution $\mathcal{C}_i$.

The scheduling of each task is also controlled by its scheduling mode (cooperative or preemptive) and its priority π_i, with preemptive tasks having higher priority than cooperative tasks, and cooperative tasks only preempting each other at runnable boundaries.

The model also defines deadlines that apply to tasks and task chains. For tasks, deadlines bound the worst case completion time with respect to the activation and match the common definition of a relative deadline D_i. Also, all tasks are assumed to be periodic or sporadic, with a period or a minimum inter-arrival time T_i. When applicable, relative deadlines are constrained to be smaller than or equal to periods, i.e., $D_i \leq T_i$. In the end, we assume each task is defined by a tuple (C_i, c_i, D_i, T_i), where $C_i = \sum_{j=1}^{n_i} C_{i,j}$, $c_i = \sum_{j=1}^{n_i} c_{i,j}$.

We denote as $R_{i,j}$ the worst-case response time of the jth runnable of task τ_i, while $r_{i,j}$ denotes its best-case response time. $hp^P(i)$ and $hp^C(i)$ denote the set of preemptive and cooperative tasks, respectively, having priority greater than τ_i. We denote as $hp(i) = hp^P(i) \cup hp^C(i)$ the union of the two disjoint sets.

As for end-to-end chains, the assumed model is based on the asynchronous propagation of information by means of shared data variables. These variables (labels in the model) are read and written by the runnables.

Figure 1 illustrates the three effect chains that are analyzed in the context of the challenge. Note that, in the third chain, we replaced Label 2197 with Label 646 to fix a mistake in the model (Label 2197 is not read nor written by the last two runnables in the chain, while Label 646 is the only one that satisfies the read/write relation imposed by the chain).

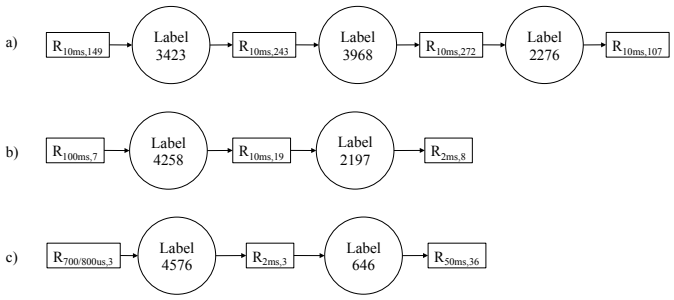

Fig. 1: Effect chains in the model.

The following semantics have been considered for end-to-end latency calculation (from [1]):

- *Last-to-First* (L2F): it considers the delay between the last input that is not overwritten until the first output generated with the same input;
- *First-to-First* (F2F) or *Reactive*: it considers the delay between the first input that may be overwritten until the first output generated with the next different input;
- *Last-to-Last* (L2L) or *Maximum Age*: it considers the delay between the last input that is not overwritten until the last output, considering duplicates.

The problem with this definition is that it is hardly formal, and even in the original reference there seems to be no single point in which a formal definition appears. Hence, we used the following definitions.

Assume a chain of periodic communicating runnables $\Gamma = \{\rho_1, \rho_2, \dots \rho_n\}$. Also, assume $a_{j,h}$ denotes the h-th activation of runnable ρ_j, $f_{j,h}$ its finishing time, and $I_{j,h}$, $O_{j,h}$ are the sets of input and output values that are respectively read from and written to the labels accessed by the h-th instance of ρ_j.

Then, the L2F latency of the chain Γ is the maximum value $f_{n,r} - a_{1,p}$ (finishing time of the r-th instance of ρ_n minus the activation time of the p-th instance of ρ_1), such that for some p, q, r:

$$\forall i = 1, \dots, (n-2) \;\; O_{i,p} = I_{i+1,q} \;\text{ and }\; O_{i+1,q} = I_{i+2,r} \\ \text{and } \; I_{i+1,q} \neq I_{i+1,q-1} \;\text{ and }\; I_{i+2,r} \neq I_{i+2,r-1}.$$

Similarly, the F2F latency of the chain Γ is the time interval between the latest $a_{1,p}$ and the earliest $f_{n,r+1}$ such that for some p, q, r:

$$\forall i = 1, \dots, (n-2) \;\; O_{i,p} = I_{i+1,q} \;\text{ and }\; O_{i+1,q} = I_{i+2,r} \\ \text{and } \; I_{i+1,q} \neq I_{i+1,q+1} \;\text{ and }\; I_{i+2,r} \neq I_{i+2,r+1}.$$

Finally, the L2L latency of the chain Γ is the maximum value $f_{n,r} - a_{1,p}$ (finishing time of the r-th instance of ρ_n minus the activation time of the p-th instance of ρ_1), such that for some p, q, r:

$$\forall i = 1, \dots, (n-2) \;\; O_{i,p} = I_{i+1,q} \;\text{ and }\; O_{i+1,q} = I_{i+2,r}.$$

Figures 2 and 3 exemplify the definitions in the case of undersampling and oversampling effects, respectively. In particular, referring to the chain $\{\rho_1, \rho_2, \rho_3\}$ in Figure 2, the end-to-end delay by the L2F semantics corresponds to the time interval between the activation $a_{1,1}$ and the finishing time of the runnable activated at time $a_{3,1}$; the end-to-end delay by F2F corresponds to the time interval $[a_{1,1}, f_{3,2}]$. By L2L, it is measured as for the L2F semantics (i.e., $f_{3,1} - a_{1,1}$). In case of oversampling (Figure 3), the end-to-end delay can be measured by the L2F semantics as $f_{3,2} - a_{1,1}$; by F2F it is $f_{3,5} - a_{1,1}$, while the L2L semantics accounts for the same data read by multiple runnable instances (e.g., in the time interval $f_{3,4} - a_{1,1}$).

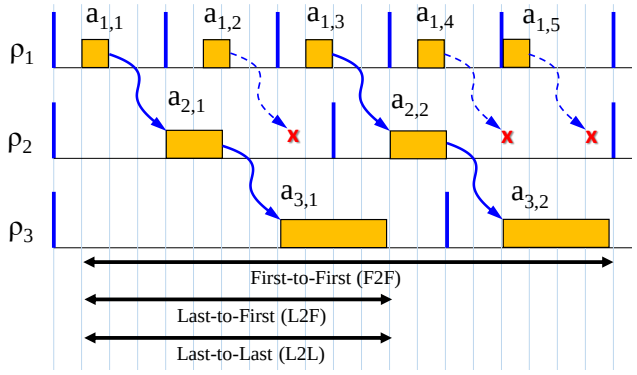

Fig. 2: End-to-end delay in the case of undersampling.

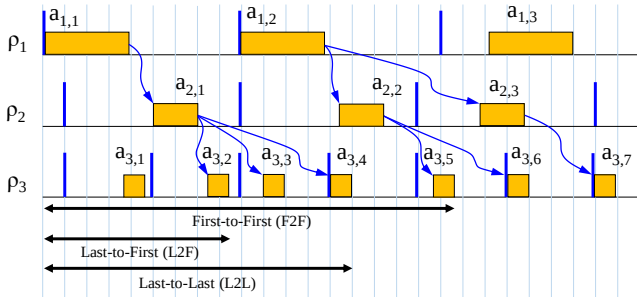

Fig. 3: End-to-end delay in the case of oversampling.

The definition ambiguity leaves open a fundamental issue. **What is the actual meaning and relevance (in application terms) of such definitions**?

III. Worst-case Latency Analysis

This section discusses the analytical approach to compute the worst-case response times for tasks and chains, with and without consideration of the timing for the access to (shared and local) memory.

A. Analysis without memory access times

For any *preemptive* task, the worst-case response time of runnable $\rho_{i,j}$ is given by the fixed point iteration of the following formula (starting with $R^0_{i,j} = \sum_{h=1}^{j} C_{i,h}$):

$$R_{i,j} = \sum_{h=1}^{j} C_{i,h} + \sum_{k \in hp(i)} \left\lceil \frac{R_{i,j}}{T_k} \right\rceil C_k. \qquad (1)$$

The above formula quantifies the higher-priority interference suffered by $\rho_{i,j}$ by considering the synchronous periodic arrivals of higher-priority tasks.

For *cooperative* tasks, the worst-case response time needs to consider also the blocking time by lower-priority cooperative runnables and the fact that the last runnable does not suffer any preemption by higher-priority cooperative tasks once it has started executing. In addition, by analogy with the limited preemptive scheduling with fixed preemption points [2], it is not enough to compute the response time of the first job after the critical instant. In particular, the computation must be carried out for all jobs $s \in [1, K_i]$ falling within the so called Level-i Active Period L_i, such that $K_i = \left\lceil \frac{L_i}{T_i} \right\rceil$. Therefore, in case of a cooperative task τ_i, we can compute the worst-case finishing time of the sth job of $\rho_{i,j}$ by the fixed point iteration of the following formula:

$$f_{i,j}^s = \sum_{h=1}^{j} C_{i,h} + B_{i,j} + (s-1)C_i + \sum_{k \in hp^P(i)} \left\lceil \frac{f_{i,j}^s}{T_k} \right\rceil C_k + \sum_{k \in hp^C(i)} \left(\left\lfloor \frac{f_{i,j}^s - C_{i,j}}{T_k} \right\rfloor + 1 \right) C_k,$$

where

$$B_{i,j} = \max_{\substack{q \in lp^C(i) \\ h=1,\ldots,n_q}} C_{q,h}$$

represents the maximum blocking time imposed by lower-priority cooperative tasks.

Then, the worst-case response time of $\rho_{i,j}$ can be computed as:

$$R_{i,j} = \max_{s \in [1,K_i]} f_{i,j}^s - (s-1)T_i. \tag{2}$$

Worst-case start time computation. Another quantity of interest for the end-to-end latency computation is the worst-case start time $S_{i,j}$ of runnable $\rho_{i,j}$. The calculation is the same for both the case of preemptive and cooperative tasks, and is given by:

$$S_{i,j} = \epsilon + \sum_{h=1}^{j-1} C_{i,h} + \sum_{k \in hp(i)} \left\lceil \frac{S_{i,j}}{T_k} \right\rceil C_k, \tag{3}$$

where ϵ is an arbitrarily small constant.

Best-case response time computation.

For preemptive tasks, the best-case response time of runnable $\rho_{i,j}$ is [3]:

$$r_{i,j} = \sum_{h=1}^{j} c_{i,h} + \sum_{k \in hp(i)} \left(\left\lceil \frac{r_{i,j}}{T_k} \right\rceil - 1 \right) c_k. \tag{4}$$

For cooperative tasks, a lower-bound on the best-case response time can be computed by considering a zero blocking-time from lower-priority tasks and the minimum amount of interference from higher-priority tasks [3], [4]:

$$r_{i,j} = \sum_{h=1}^{j} c_{i,h} + \sum_{k \in hp^P(i)} \left(\left\lceil \frac{r_{i,j}}{T_k} \right\rceil - 1 \right) c_k + \sum_{k \in hp^C(i)} \left\lfloor \frac{r_{i,j} - c_{i,j}}{T_k} \right\rfloor c_k. \tag{5}$$

B. End-to-end Latency Calculation

The end-to-end latencies have been computed according to the semantics reported in Section II. For each chain, we first compute the end-to-end latency by the Last-to-First (L2F) semantics, and then extend it to obtain the latencies by the F2F and L2L semantics.

Last-to-First semantics. The end-to-end latency of chain $\rho_1, \ldots, \rho_N$ according to the L2F semantics can be computed as:

$$\sum_{i=1}^{N-1} (R_i + \min(T_{i+1} - r_{i+1}, T_i)) + R_N. \tag{6}$$

First-to-First semantics. With respect to the L2F semantics, in the F2F semantics we need to add one cycle delay for the first runnable in the chain, in order to consider the previous input. Therefore, the end-to-end latency of chain $\rho_1, \ldots, \rho_N$ according to the F2F semantics can be computed as:

$$T_1 + \sum_{i=1}^{N-1} (R_i + \min(T_{i+1} - r_{i+1}, T_i)) + R_N. \tag{7}$$

Additionally, the F2F semantics considers previous inputs that are overwritten. In order to compute how many times in the worst case an input is overwritten between consecutive stages of the chain (i.e., between runnables ρ_i and ρ_{i+1}), we need to find the largest possible integer $\overline{n} \geq 1$ that satisfies:

$$T_{i+1} + S_{i+1} - r_{i+1} \geq \overline{n} T_i + r_i - R_i. \tag{8}$$

This relation guarantees that the longest interval between two consecutive reads is greater than the shortest interval between $\overline{n}$ consecutive writes. If the above relation holds (i.e., input overwriting takes place), we compute the end-to-end latency of chain $\rho_1, \ldots, \rho_N$ as:

$$T_1 + \sum_{i=1}^{N-1} (R_i + \overline{n} T_i) + R_N. \tag{9}$$

Last-to-Last semantics. With respect to the L2F semantics, the L2L also considers subsequent outputs that are overwritten. In order to compute how many times in the worst case an output is overwritten between consecutive stages of the chain, we need to find the largest possible integer $\widehat{n} \geq 1$ that satisfies:

$$T_i - r_i + R_i \geq \widehat{n} T_{i+1} - r_{i+1} + S_{i+1}. \tag{10}$$

This relation guarantees that the longest interval between two consecutive writes is greater than the shortest interval to perform $\widehat{n}$ consecutive reads. If the above relation holds (i.e., output overwriting takes place), we compute the end-to-end latency as:

$$\sum_{i=1}^{N-1} (R_i + \widehat{n} T_{i+1} - r_{i+1}) + R_N. \tag{11}$$

Otherwise, the end-to-end latency by the L2L semantics is as the one obtained under the L2F semantics.

C. Analysis with memory access and arbitration times

In the proposed model, the four cores contend for access to a shared global memory (GRAM) with FIFO arbitration. Each read/write access to GRAM costs 9 cycles (there is no caching effect). Therefore, in the worst case each memory access might get blocked by pending accesses from other cores, i.e., each access can be delayed for $9(m-1) = 27$ cycles. Adding up the memory access cost for the current request, we obtain a worst-case memory-access penalty of 36 clock cycles. By exploiting the knowledge of how many labels are read/written by each runnable, we can compute the worst-case memory access latency for its read/write phases.

In the best case, memory accesses do not experience any delays from other cores, leading to a best-case memory-access time of 9 clock cycles. Accordingly, we can compute the best-case memory access latency for the read/write phases.

Such values need to be added to the execution time of each runnable, to which the analysis described in Section III-A can be applied identically.

The worst-case estimate of $9(m-1)$ cycles implies that the 9 cycles access cost is repeatedly applied on each FIFO access, which is most likely a pessimistic estimate given the lack of detailed information on the HW (memory) configuration. Careful consideration of the memory access costs **require a model of the execution HW more detailed than what is typically available in scheduling analysis papers**.

D. End-to-end Latency Calculation

The end-to-end latency calculation can be performed as described in Section III-B, with the following differences.

Last-to-First semantics. Equation (6) is replaced by:

$$\sum_{i=1}^{N-1}(R_i - r_{i+1}^{read} + \min(T_{i+1}, T_i)) + R_N, \tag{12}$$

where r_i^{read} denotes the best-case response time of the read phase of ρ_i.

First-to-First semantics. (8) is replaced by:

$$T_{i+1} + S_{i+1} - r_{i+1}^{read} \geq \overline{n}T_i + r_i - R_i. \tag{13}$$

Last-to-Last semantics. (10) is replaced by:

$$T_i - r_i + R_i \geq \widehat{n}T_{i+1} - r_{i+1}^{read} + R_{i+1}^{read}, \tag{14}$$

where R_i^{read} denotes the worst-case response time of the read phase of ρ_i, which can be computed similarly as in Section III-A.

E. Experimental Evaluation

In order to make the system analyzable, the WCETs of those tasks that were not deemed schedulable by our analysis were scaled down by considering the largest scaling factor $\sigma \in (0, 1]$ that guarantees schedulability. In particular, starting from $\sigma = 1$, WCETs are iteratively scaled down in steps of 0.01 until the system becomes schedulable by the proposed analysis. Table I reports the scaling factor σ for each task, and the scaling factor σ^M obtained when memory access and arbitration are accounted for. The analytical approach described in Section III has been implemented in C++, and the code is fully available online [5].

TABLE I: Scaling factors.

Task	Core	σ	σ^{M}	Task	Core	σ	σ^{M}
ISR10	0	1	1	5ms	2	1	1
ISR5	0	1	1	20ms	2	1	1
ISR6	0	1	1	50ms	2	1	0.52
ISR4	0	1	1	100ms	2	0.28	0.12
ISR8	0	1	1	200ms	2	0.49	0.78
ISR7	0	1	1	1000ms	2	0.18	0.15
ISR11	0	1	1	ISR1	3	1	1
ISR9	0	0.58	0.29	ISR2	3	1	1
1ms	1	1	1	ISR3	3	1	1
Angle Sync	1	0.37	0.26	10ms	3	0.84	0.78
2ms	2	1	1				

1) Effect Chain 1: In the effect chain 1: (i) all runnables belong to the same task (*Task_10ms*, allocated to core 3), hence all runnables are bound to the same rate; (ii) there is backward communication between the third and the fourth runnable, which implies a one cycle delay until the last datum is read. Therefore, the worst-case end-to-end latency of this effect chain by L2F can be computed as:

$$L_1^{L2F} = T_{10ms} + R_{10ms,107} = 13376\ \mu s. \tag{15}$$

Given that all runnables belong to the same task, this result is valid also when considering the L2L semantics. As for the F2F semantics, the analysis needs to consider a one cycle delay for the first runnable, that is:

$$L_1^{F2F} = 2T_{10ms} + R_{10ms,107} = 23376\ \mu s. \tag{16}$$

2) Effect Chain 2: Unlike the previous chain, runnables in this chain belong to different tasks with different rates. In this case, the end-to-end latency calculation should also consider the over-sampling effect between pairs of consecutive runnables. By the L2F semantics, applying Equation (6), we obtain:

$$\begin{aligned} L_2^{L2F} = R_{100ms,7} + \min(T_{10ms} - r_{10ms,19}, T_{100ms}) \\ + R_{10ms,19} + \min(T_{2ms} - r_{2ms,8}, T_{10ms}) \\ + R_{2ms,8} = 52222\ \mu s \end{aligned}$$

As for the F2F semantics, due to the over-sampling effect, there are no input overwritings (Condition (8) is never verified), hence the end-to-end latency is simply given by:

$$L_2^{F2F} = L_2^{L2F} + T_{100ms} = 152222\ \mu s.$$

Finally, the end-to-end latency computation for the L2L semantics requires to verify Condition (10) for any pair of consecutive runnables. In this case, we obtain $\widehat{n} = 13$ for the first stage and $\widehat{n} = 5$ for the second stage, which yields:

$$\begin{aligned} L_2^{L2L} = R_{100ms,7} + 13 \cdot T_{10ms} - r_{10ms,19} + R_{10ms,19} \\ + 5 \cdot T_{2ms} - r_{2ms,8} + R_{2ms,8} = 180222\ \mu s. \end{aligned}$$

3) Effect Chain 3: Also in this case, runnables belong to different tasks with different rates. Task periods have increasing values, leading to an under-sampling effect.

By the L2F semantics, applying Equation (6), we obtain:

$$\begin{aligned} L_3^{L2F} = R_{700/800us,3} + \min(T_{2ms} - r_{2ms,3}, T_{700/800us}) + \\ R_{2ms,3} + \min(T_{50ms} - r_{50ms,36}, T_{2ms}) + R_{50ms,36} = 41953\ \mu s \end{aligned}$$

Due to the sporadic nature of the first runnable, we assume $T_{700/800us} = 800\ \mu s$ in order to maximize latency.

The end-to-end latency by the F2F semantics requires to add one cycle delay with respect to L2F and to verify Condition (8) for any pair of consecutive runnables. In this case, we obtain $\overline{n} = 2$ for the first stage[1] and $\overline{n} = 43$ for the second stage, which yields:

$$L_3^{F2F} = T_{700/800us} + 2 \cdot T_{700/800us} + R_{700/800us,3} + 43 \cdot T_{2ms} + R_{2ms,3} + R_{50ms,36} = 127553\ \mu s.$$

Finally, the end-to-end latency for the L2L semantics is equal to the L2F case, because no output is overwritten due to the under-sampling effect.

Similar calculations are performed to compute end-to-end latencies accounting for memory effects, as described in Section III-C.

Table II summarizes the obtained end-to-end latencies calculated according to the different semantics adopted, for each of the two challenges.

TABLE II: End-to-end latency upper bounds (μsec) for the first (I) and second (II) challenge.

Chain	L2F I	L2F II	F2F I	F2F II	L2L I	L2L II
1	13376	13383	23376	23383	13376	13383
2	52222	52796	152222	152796	180222	180796
3	41953	42448	127553	130040	41953	43248

IV. Model Simulator

The analysis by simulation of the challenge model has been performed by a purposely developed extension [6] [7] to the C++ RTSIM [8] scheduling simulator.

A. Data Acquisition

The (engine control) application model that is the subject of the challenge is defined by an XML file that can be parsed to obtain the model data. The model information is then stored in data structures internal to the simulator C++ classes.

Some of the model information requires a preliminary elaboration, such as the execution time that is represented by parameters of a Weibull distributions: the lower bound (b), the upper bound (B), the mean (η), and the probability of having values greater than the upperbound (ρ). Those parameters must be converted to compute the standard Weibull parameters: scale (λ) and shape (k). The transformation has been performed considering that the cumulative distribution function (CDF), given an uniformly distributed random variable x, is null for $x < 0$ and for $x \geq 0$ is defined as $CDF(x) = 1 - e^{-(x/\lambda)^k}$. By considering that for $x = B - b$, it is possible to obtain $CDF(B-b) = 1 - \rho$, and after performing some substitution it is possible to define $\lambda = \frac{\sqrt[k]{-\ln \rho}}{B-b}$. The mean value of a Weibull distribution is calculated as $\eta = \lambda \Gamma\left(1 + \frac{1}{k}\right)$, and, by substituting the first result in the second equation, we obtain $\frac{\sqrt[k]{-\ln \rho}}{B-b}\Gamma\left(1+\frac{1}{k}\right) - \eta = 0$.

The approach followed by the simulator described in this paper to obtain an approximation of the k parameter is to minimize the absolute error of the previously described function

$$\min_{k>0} \left| \frac{\sqrt[k]{-\ln \rho}}{B-b}\Gamma\left(1+\frac{1}{k}\right) - \eta \right|. \quad (17)$$

The function minimum is obtained by using the GNU Scientific Library [9].

B. Cores, Kernels and Schedulers

In RTSIM the main entity for scheduling simulations is the Kernel. Each Kernel has an associated Core. Once a Kernel is instantiated, the programmer assigns a Scheduler to it. Among the different available schedulers, the one used for the challenge is the fixed priority scheduler. In RTSIM, partitioned multi-core scheduling is obtained by instantiating multiple Kernel objects, one for each core.

C. Tasks

Each task τ_i in RTSIM is defined by its parameters: the activation time of first job ($a_{i,0}$), its relative deadline (D_i), its period or minimum inter-arrival time (T_i), and the sequence of instructions it executes, each defined by an execution time specification (deterministic or random). For any periodic task, the activation time of each job is computed by adding T_i to its last activation time. For sporadic tasks, a random value in the range $[T_i, T_i^{max}]$ is added to the last activation time, where T_i^{max} denotes the maximum inter-arrival time of τ_i. Relative deadlines are set equal to T_i. As for the job instructions, each task executes a sequence of runnables. According to the RTSIM syntax, we defined a new instruction "runnable(runnableName)", and the code of each task is of the form

```
runnable(r1);runnable(r2);...runnable(rN);
```

D. Runnables

Cooperative tasks preempt lower priority cooperative tasks only at runnable borders, while higher priority preemptive task can preempt any lower priority task and runnable. In the case of cooperative tasks, preemption within runnables is prevented by locking and unlocking a core-specific mutex dedicated to cooperative tasks before and after calling a runnable. The resulting job code for a cooperative task is:

```
...lock(muxC);runnable(rX);unlock(muxC)...
```

When a job calls a runnable instruction, the operations performed, in order, are the following: updating end-to-end statistics associated to labels reading events, virtually executing the runnable computations, updating end-to-end statistics associated to labels writing events.

E. Results

All the simulation runs performed for the challenge system produced the following: (i) Complete traces of the task scheduling events; (ii) F2F and L2L end-to-end delays of each chain; (iii) Response times of all runnables involved in each chain. The system simulation was performed collecting sample runs for different initial offsets of the tasks. For periodic tasks, the initial offsets are uniformly selected in the interval $[0, T_i]$, while for sporadic tasks they are chosen in $[0, T_i^{max}]$. The execution of the tasks has been simulated for a total virtual time of one hour. The simulation required 28 minutes and 50

[1]This calculation considers $T_{700/800us} = 800\ \mu s$, since this value maximizes the latency of the given effect chain.

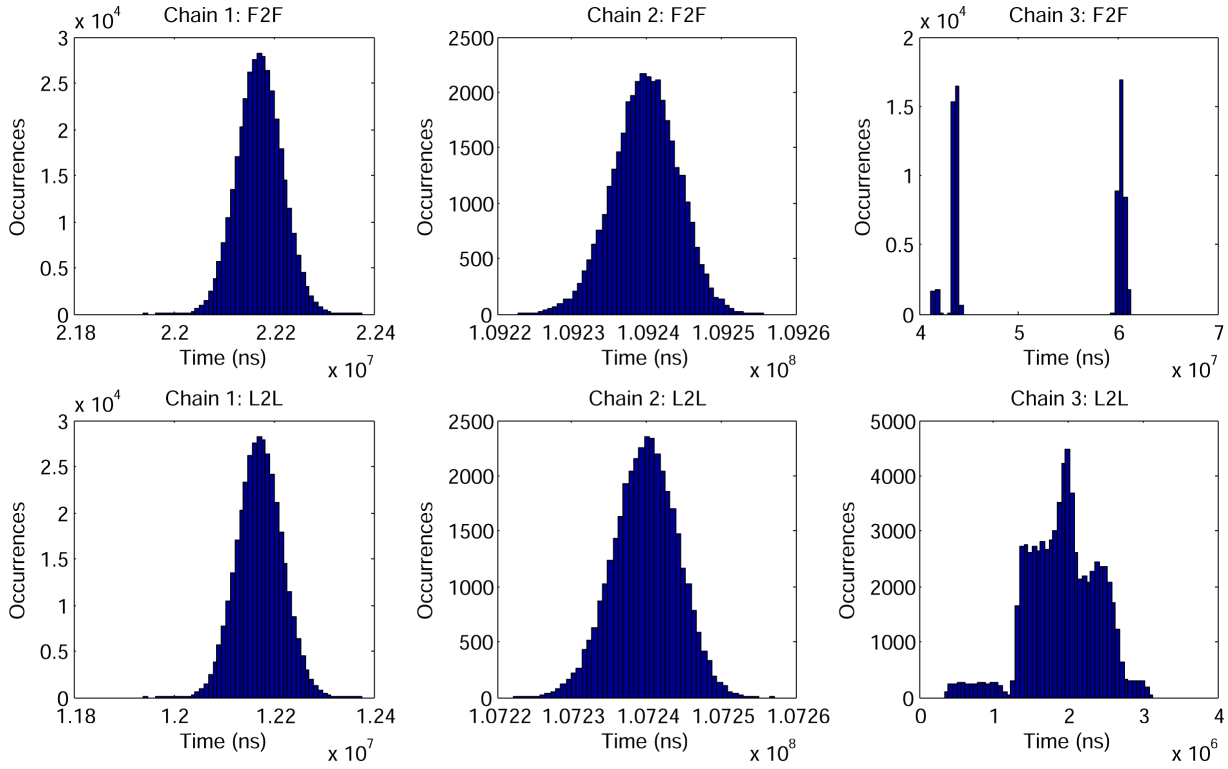

Fig. 4: End-to-end delays obtained by simulation.

seconds on a system with an *Intel i7-2630QM* core running at 2 GHz and 8 GB of DDR3 RAM running at 1333 MHz.

Figure 4 represents the distribution of the F2F and L2L latencies for each chain. For the first chain, the maximum end-to-end delays measured by simulation are 22377 μs for F2F and 12377 μs for L2L. For the second chain, the maximum end-to-end delays measured by simulation are 109.26 ms for F2F and 107.26 ms for L2L. In the end, the simulation returns 61324 μs for F2F and 3139.4 μs for L2L as maximum end-to-end delays for the third chain.

Additionally, Table III establishes a comparison between the worst-case response times of the runnables by the worst-case latency analysis of Section III (WCRT), which takes into account the scaling factors computed in Table I to guarantee schedulability, and the maximum response times observed during our simulations (SIM).

TABLE III: Worst-case response times (μsec) for the first (I) and second (II) challenge.

Runnable	WCRT I	WCRT II	SIM I
$R_{10ms,149}$	5176	5144	3556
$R_{10ms,243}$	7919	7903	5431
$R_{10ms,272}$	8896	8879	6139
$R_{10ms,107}$	3376	3383	2377
$R_{100ms,7}$	39647	39865	6992
$R_{10ms,19}$	770	781	577
$R_{2ms,8}$	142	150	122
$R_{700/800us,3}$	30	33	27
$R_{2ms,3}$	49	53	46
$R_{50ms,36}$	39074	39562	11151

The evaluation of the memory access costs rquires further extensions to the simulation engine that could not be completed in time for this paper.

V. Conclusions

In this paper, we proposed two solutions for the timing verification problem of the FMTV challenge. The first approach builds a mathematical model of the system and calculates worst-case latencies by adaptation of existing response time analysis techniques. Upper bounds on the end-to-end latencies are derived by first ignoring and then including memory access times. Then, a simulator of the given AUTOSAR model has been built on RTSIM to compute end-to-end latencies of the selected effect chains.

References

[1] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *CRTS*, 2008.

[2] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. A survey." *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, 2013.

[3] R. Bril, "Existing worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption is too optimistic," *CS-Report 06*, vol. 5, 2006.

[4] R. Bril and W. Verhaegh, "Towards best-case response times of real-time tasks under fixed-priority scheduling with deferred preemption," in *ECRTS, WiP session*, 2005, pp. 17–20.

[5] *A C++ implementation of schedulability analysis and end-to-end latency calculation for WATERS Challenge 2016*, http://retis.sssup.it/%7Eal.melani/downloads/FMTV-analysis.zip, 2016.

[6] "MetaSim2.0 event-based simulator," https://github.com/balsini/metasim2.0, accessed: May 17, 2016.

[7] "RTSIM real-time system simulator extended for waters challenge 2016," https://github.com/balsini/waters/, accessed: Branch 2016.

[8] "RTSIM real-time system simulator," http://rtsim.sssup.it/, accessed: Version 2.0.

[9] "GSL gnu scientific library," https://www.gnu.org/software/gsl/, accessed: Version 1.16.