New Strategies For Multisource Software on Multicore Automotive Electronic Control Unit

RINOLD WILSON P.G.(PG SCHOLAR),
ALEX GEORGE(ASSISTANT PROFESSSOR),
Department of Electrical And Electronics Engineering,
SNS College Of Engineering,
Kurumbapalayam, Coimbatore,
india
mail@:rinold777@gmail.com

Abstract- **In this paper we improve the efficiency of the multi core ECUs by using multisource software. As the demand for computing power is quickly increasing in the automotive domain, car manufacturers and tier-one suppliers are gradually introducing multicore electronic control units (ECUs) in their electronic architectures. In addition, these multicore ECUs offer new features such as higher levels of parallelism, which ease the compliance with safety requirements . These new features involve greater complexity in the design, development, and verification of the software applications. Here applying algorithms to improve the efficiency of ECUs. dynamic method is used for executing the different type of runnables. In this paper, we address the problem of sequencing numerous elementary software modules, called runables, on a limited set of identical cores. Wke show how this problem can be addressed as the following two sub problems, which cannot optimally be solved due to their algorithmic complexity:**

Partitioning the set of runnables, Building the sequencing of the runnables on each core. We then present low-complexity heuristics to partition and build sequencer tasks that execute the runnable set on each core. Finally, we globally address the scheduling problem, at the ECU level, by discussing how we can extend this approach in cases where other OS tasks are scheduled on the same cores as the sequencer tasks.

Keywords- **electronic control units (ECUs), Least crowded (LC), Least loaded (LL), Least peak(LP), Least peak sigma(LPS).**

1. Introduction

MULTISOURCE software running on the same electronic control unit (ECU) is becoming increasingly wide spread in the automotive industry. This case is one of the main reasons that car manufacturers want to reduce the number of ECUs.

One major outcome of the Automotive Open System Architecture (AUTOSAR) initiative and, more specifically, its operating system (OS) is to help car manufacturers shift from the “one function per ECU” paradigm to more centralized architecture designs by providing appropriate protection mechanisms.

Another crucial evolution in the automotive industry is that chip manufacturers are reaching the point where they can no longer cost effectively meet the increasing performance requirements through frequency scaling alone. This condition is one reason that multicore ECUs are gradually introduced in the automotive domain. The higher level of performance provided by multicore architectures may help simplify in-vehicle architectures by executing on multiple cores that the software previously run on multiple ECUs. This possible evolution toward more centralized architectures is also an opportunity for car manufacturers to decrease the number of network connections and buses. As a result, parts of the complexity will be transferred from the electrical/electronic architecture to the hardware and software architecture of the ECUs. However, static cyclic scheduling makes it easy to add functions to an existing ECU.

In practice, important architectural shifts are hindered by the carryover of ECUs and existing subnetworks, which are widely used by generalist car manufacturers. The extent to which more centralized architectures will be adopted thus remains unsure.

The introduction of multisource and multicore will induce drastic changes in the software architecture of automotive ECUs. Section II introduces the most likely scheduling choices and the literature relevant to the task scheduling in multiprocessor automotive ECUs. Then, Section III presents solutions for the scheduling of numerous software modules
when only a few OS tasks are allowed. This paper builds on the study published in [5], where it was assumed that only one sequencer task was running on each core of the ECU to schedule the runnables. In Section V, we consider how we can build several sequencer tasks while possibly scheduling other asks on the same core, and we discuss how we can globally analyze the schedulability of such systems. For clarity, "sequencing" refers to the scheduling of runnables, whereas "scheduling" is solely used for tasks.

2. Scheduling in the automotive domain

Scheduling Design Choices for Multicore ECUs

In this section, we explain and justify, particularly in light of predictability requirements, the multicore scheduling approach, which is, to the best of our knowledge, the most widely considered method in the automotive industry.

1) Partitioning Scheduling Scheme: In a multicore system, either the tasks are statically allocated to the cores or they can dynamically be distributed at runtime to balance the workload or migrate functions to increase availability. The latter approach involves complex tasks and resource interactions that are difficult to predict and validate. Thus, approaches that rely on static allocation (i.e., partitioning) and deterministic mechanisms such as periodic cyclic scheduling are more likely to be used in the automotive context, and this is the option taken within the AUTOSAR consortium. Scheduling tasks on a multiprocessor systems under the static partitioning approach has been well studied; for example, see [6]–[9]. However, the works we are aware of deal with online algorithms such as fixed priority preemptive (FPP) or earliest deadline first (EDF) and do
not consider the static cyclic scheduling of tasks.

2) Static Cyclic Scheduling: The static cyclic scheduling of elementary software modules or runnables is common, because there are usually many more runnables than the maximum number of tasks allowed by automotive operating systems such as OSEK/VDX or AUTOSAR OS. Thus, runnables must be grouped together and scheduled within a sequencer task (also
called a dispatcher task). In this paper, we focus on how we can sequence large runnable sets on multicore platforms using a
static partitioning approach. Indeed, the static task partitioning scheme is very likely to be adopted, at least, in a first step, because it is conceptually simple and provides better predictability for ECU designers compared with a global scheduling approach. We aim at developing practical algorithms whose performances can be guaranteed to build the dispatcher tasks on each core and to schedule the runnables within these dispatcher tasks to comply with sampling constraints and, as long as possible, uniformize the CPU load over time. This latter objective is, of course, important to minimize the hardware cost
and to facilitate the addition of new functions, as typically done in the incremental design process of car manufacturers. This objective is achieved by desynchronizing the runnable release
dates. Precisely, the first release date of each runnable, called its offset, is determined to uniformly spread the CPU demand over time. The configuration algorithms developed in this paper are closely related to [10] (monoprocessor scheduling of tasks with offsets) and [11] (scheduling of frames with offsets), but it is applied to multicore and goes beyond as we provide lower bounds on the performances.
However, the proprietary algorithms used in these tools can usually not be disclosed, and they are sometimes specialized for some specific usage.

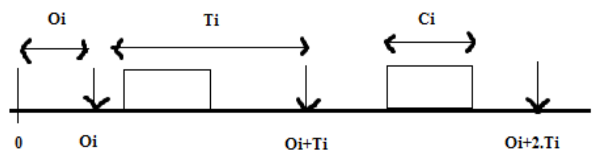


Fig. 1. Model of the runnables. After its release, an instance of a runnable has to be executed before the next instance is released (i.e., the deadline is set to the period).

Model Description

In this paper, we consider a large set of *n* periodic elementary software modules, also called runnables, that will be allocated on an ECU that consists of *m* identical cores. In practice, a runnable can be implemented as a function that is called, whenever appropriate, within the body of an OS task.

1) Runnable Characteristics: The *i*th runnable is denoted by *Ri* = (*Ci, Ti,Oi, {R}, Pi*). Quantities *Ci*, *Ti*, and *Oi* correspond, respectively, to the worst case execution time (WCET), the period (i.e., the exact time between two successive releases), and the offset of *Ri*. As shown in Fig. 1, the offset of a runnable is the release date of the first instance of that runnable, and subsequent instances are then periodically released. The choice made for the offset values has a direct influence on the repartition of the workload over time.

2) Dispatcher Task: Runnables are scheduled on their designated core using a dispatcher task or a "sequencer task," which stores the runnable activation times in a table and releases them at the right points in time. A dispatcher task is characterized by the duration of the dispatch table *Tcycle*, which is executed in a cyclic manner, and by a quantum *Ttic*, which is the duration of a slot in the table. Typically, we may have, for example, *Tcycle* = 1000 ms and *Ttic* = 5 ms. Note that *Tcycle* must be a multiple of the greatest common divisor of the runnable periods and the least common multiple (LCM) of these periods must be a multiple of *Ttic*. As a result, a dispatch table holds *Tcycle/Ttic* slots.

3) Assumptions: In this paper, we place a set of working assumptions, which, in our experience, can most often be met in current automotive applications, as follows.

• Each runnable is periodically executed strictly. As a result, the whole trajectory of the system is defined by the first activation times of the runnables (i.e., their offsets).

• The WCETs of the runnables are assumed to be small compared to *Ttic*. Typical values for the case that we consider would be 5 ms for *Ttic* and *Ci* $\leq$ 300 μs.

• All cores are identical with regard to their processing speed.

• There are no dependencies between runnables allocated on different cores. Therefore, all cores can independently be scheduled. This assumption is in line with the choices made by AUTOSAR with regard to multicore architecture.

4) Schedulability Condition: Assuming that we only consider runnable scheduling, the system is schedulable and, thus, can safely be deployed if and only if the following conditions are satisfied on each core.

1) The runnables are strictly periodically executed.

2) The initial offset of each runnable is smaller than its period.

3) The sum of the WCET of the runnables allocated in each slot does not exceed a given threshold, which is typically chosen as the duration of the slot, i.e., *Ttic*.

3. RUNNABLE SEQUENCING ALGORITHMS FOR MULTICORE PROCESSORS

In this section, we present algorithms and, when possible, derive lower bounds on their efficiency to schedule large numbers of runnables on multicore ECUs. Because automotive OSs can only handle a limited amount of OS tasks, the sequencing of runnables has to be done within dispatcher tasks. The first step of the approach is to partition the runnable sets onto different cores. The next and last step is to determine the offsets between the runnables allocated on each core to balance the load over time.

A. Building Tasks as a Bin-Packing Problem

It is assumed that the number of cores is fixed. We first distribute all the runnables on the cores. Assigning *n* tasks to *m* cores is similar to subdividing a set of *n* elements into *m* nonempty subsets. By definition, the number of possibilities for this problem is given by the Stirling number of the second kind

$$(1|m!)\sum_{i=0}^{m}(-1)^{(m-i)}\binom{m}{i}i^{n}$$

Considering that the runnables may have core allocation constraints, the cores should be distinguished. Thus, the *m*! combinations of cores must be considered. As a result, we have at most different possibilities for the partitioning problem alone. Such a complexity prevents us from an exhaustive search. For example, with *n* = 30 and *m* = 2, the search space holds more than one billion possibilities. Considering this complexity, to balance as evenly as possible the utilization of processor cores, we propose a heuristic based on the bin-packing decreasing worst-fit scheme for a fixed number of bins (where "bins" are processor cores). The heuristic is given in Algorithm 1.

B. Strategies for Sequencing Runnables

The next stage consists of building the dispatch table for the set of runnables. In the first step, it is assumed that there are no precedence constraints between the runnables and that a single sequencer table is needed per core. This latter assumption can easily be relaxed, as done in Section

1) Least Loaded Algorithm: Considering a runnable *Ri* of period *Ti*, there are (*Ti/Ttic*) possibilities for allocating this runnable (see schedulability condition 2 in Section II-B4). As a result there are *i*=1(*Ti/Ttic*) alternative schedules for the *n* runnables, and given the cost function, we are not aware of any ways of finding the optimal solution with an algorithm that does not have an exponential complexity. Considering a realistic case of 50 runnables whose period is as least twice as large as *Ttic*, we would need to evaluate a minimum of 250 possible solutions. Once again, given the complexity, we have to resort to a heuristic. Here, we adapt to the problem of sequencing runnables the LL algorithm proposed by Grenier *et al.* in [11]
for the frame offset allocation on a controller area network.
The intuition behind the heuristic is simple. At each step, we assign the next runnable to the LL slot, as described in Algorithm 2. The load of a slot is the sum of the *Ci* of the runnables *{Ri}* assigned to this slot. This algorithm is further referred to as the LL algorithm.

slot	1	2	3	4	5	6	7

load	2	4	2	3	2	4	2

For practical applications, ties at step 1 are broken using the highest WCET first, and ties at step 2a are broken by choosing the central slot of the longest sequence of consecutive slots having the minimum load. Although the latter rule for breaking ties does not have any impact on the theoretical results, which will be derived next, it helps separate load peaks, which is important from the ECU designer point of view. As an illustration, applying the LL heuristic to the set of runnables-mode voltage.

2) Upper Bound on the Peak Load: Here, we derive an upper bound on the peak load, which holds for runnable sets with harmonic periods (i.e., each period is a multiple of all smaller periods). Based on this value, we consequently derive a closed form sufficient schedulability condition. In this perspective we first point out that the slots in which a runnable *Ri* will periodically be assigned are of equal load, which is the rationale behind step 2a in Algorithm 2.

Lemma 1: Before the allocation of a runnable *Ri*, the slot allocation induced by the previously allocated runnables repeats with a period (*Ti/Ttic*).

Lemma 2: The maximum load in the LL slot is obtained for perfect load balancing, which corresponds to a constant load throughout the cycle.

Proof: Reasoning with a constant allocated load, anything else than a perfect load balancing will result in a load below the average load per slot in some slot that will eventually be chosen to allocate the runnable under consideration.

Theorem 3: On processor *k*, an upper bound on the peak load of a slot allocation is

$$PL_k = \max_{i\in\{\mathcal{R}\}_k} \left\{ C_i + \rho_k T_{tic} - \sum_{j=i}^{\mathrm{card}\{\mathcal{R}\}_k} \frac{C_j}{T_j} T_{tic} \right\}.$$

Proof: In the case of perfect load balancing, before the allocation of *R_ i*, the load of a slot is given by

$$\sum_{j\in\{\mathcal{R}\}_k}^{i-1} C_j \cdot \frac{T_{cycle}}{T_j} \cdot \frac{T_{tic}}{T_{cycle}}.$$

After the allocation of *Ri*, the load in the corresponding slot is

$$C_i + \sum_{j\in\{\mathcal{R}\}_k}^{i-1} C_j \cdot \frac{T_{tic}}{T_j}.$$

$$\textstyle\sum_{j\in\{\mathcal{R}\}_k}^{i-1} (C_j/T_j) = \rho_k - \sum_{j=i}^{\mathrm{card}\{\mathcal{R}\}_k} (C_j/T_j).$$

Consequently, the worst case peak load on processor core *k* resulting from the allocation of *Ri* in a slot is

$$PL_k^i = C_i + \rho_k T_{tic} - \sum_{j=i}^{\mathrm{card}\{\mathcal{R}\}_k} \frac{C_j}{T_j} T_{tic}.$$

4. EXPERIMENTATIONS

Here, we evaluate the ability of the algorithms to uniformize the CPU load over time and to keep on providing feasible solutions at very high load levels. For this purpose, the algorithms LL, LP, and LP$k\sigma$, described, respectively, in Sections III-B1, B4, and B5, have been implemented in the freely available software RTaW-ECU.

A. Balancing Performance

We applied the algorithms to sets of runnables that are realistic in the sense that their characteristics (i.e., period and WCET) are drawn at random from distributions derived from an existing body gateway ECU with about 200 runnables whose periods are close to harmonic (only about 5% of the runnables have no harmonic periods).

In the experiments, the duration of the slot *Ttic* is set to 5 ms, the largest WCET is 30 times the smallest WCET, and the periods are nonharmonic, chosen in {10, 20, 25, 40, 50, 100, 200, 250, 500, 1000} ms. Random dependencies between runnables are also introduced through the following three parameters.

• Interdependency ratio, i.e., the percentage of runnables that are dependent and must thus be executed on the same core, is chosen equal to 30%.

• Maximum size of the clusters of dependent runnables is equal to 4.

• Core locality constraint ratio is the percentage of runnables that are pre allocated to a given core, chosen to be equal to 30%.

The following additional parameters are used for this experimentation:

1) *C*max = 300 μs; 2) *Ttic* = 5 ms; 3) *Tcycle* = 1 s.

In addition, there are more than 4000 runnables to schedule on three cores, inducing an average load of 95% of the capacity of the ECU.

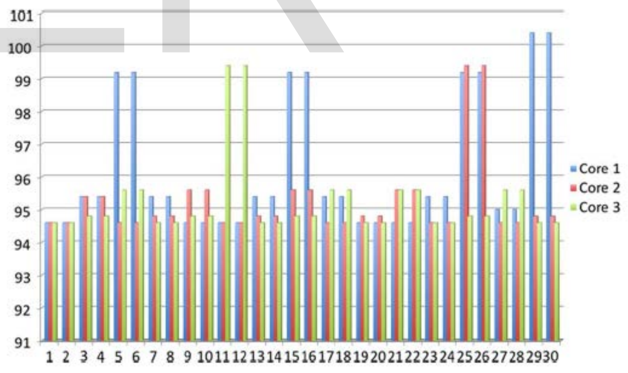


Fig2.Distribution of the load percentage over time.

The parameters have been set so that the problem is challenging, because we are above the harmonic schedulability bound, which would be 94% here.

B.Schedulability Performances and Robustness on Automotive ECUs

The goal is to assess the extent to which the schedulability bound, even if it has been derived in the harmonic case, can provide guidelines for the nonharmonic case. Precisely, we measure the success rate of the algorithms in the nonharmonic case at load levels such that feasibility would be ensured in the harmonic case. In the existing body gateway ECU, the set of task periods is close to be harmonic, because withdrawing only a few runnables ensures the harmonic property. To test the algorithms in a more difficult context, we build a "hard" nonharmonic case with more departure from the harmonic property. Precisely, the periods are now chosen in the set {10, 20, 25, 40, 50, 100, 125, 200, 125, 500, 1000} ms.

As shown in Table III, when the load is close to the harmonic schedulability bound, the algorithms remain efficient, in particular the LP, which successfully scheduled the 1000 random configurations of the test. This result suggests that the harmonic schedulability bound is also a good dimensioning criterion in the nonharmonic case. Table IV presents the results obtained at higher loads, i.e., above the harmonic schedulability bound. Precisely, sets of runnables with the maximum WCET equal to 300 and 900 μs and CPU loads equal to 95% and 97% are scheduled with LL, LP, and LP1σ.

6. CONCLUSION

Multisource software and multicore ECUs will drasticall change the electrical/electronic architectures and should enable more cost effective and more flexible automotive embedded systems. In our view, the OS protection mechanisms specified by AUTOSAR provide a sound basis for developing appropriate safety mechanisms and policies, despite the growing complexity and criticality of software functions. However, current design methodologies need to be adapted to this new context, and there is a wide range of technical problems to be solved.

Among these issues are the design of the software architectures and the scheduling of the software components, which have been considered in this paper. The set of runnable sequencing algorithms proposed in this paper aims at uniformizing the load over time and thus increases the maximum workload schedulable on the CPU. The algorithms also provide guaranteed performance levels in some specific contexts. Experimentations on realistic case studies have confirmed that the algorithms are versatile and efficient in terms of CPU usage optimization.

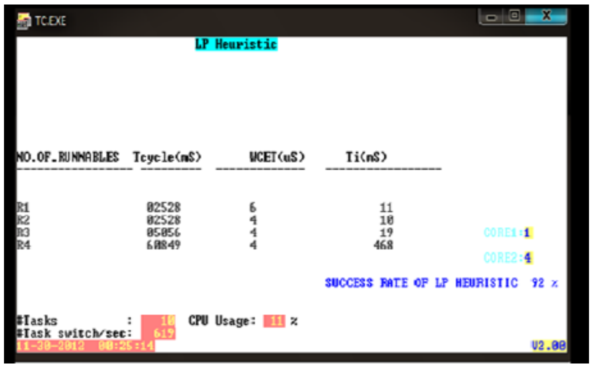

```
TC.EXE
                     LP Heuristic

NO.OF.RUNNABLES  Tcycle(mS)       WCET(uS)     Ti(mS)
---------------- ---------  ------------  ----------------

R1                02528          6            11
R2                02528          4            10
R3                05056          4            19                  CORE1:1
R4                60849          4            468
                                                                  CORE2:4
                                         SUCCESS RATE OF LP HEURISTIC  92 %

#Tasks          :    10   CPU Usage:  11 %
#Task switch/sec:   619
11-30-2012  00:25:14                                                V2.00
```

Fig 3. Outpuf of LP algorithm with efficiency 92%

We have presented practical solutions for scheduling activities according to both the static cyclic and priority-driven paradigms, as it is becoming a need in automotive multicore ECUs and other complex embedded systems with dependability requirements such as in the aerospace domain.

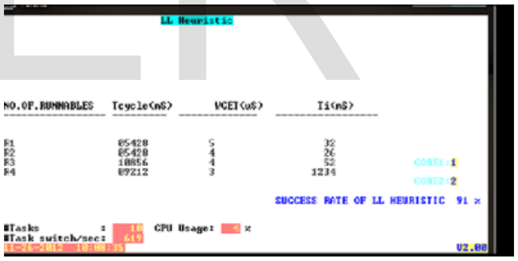

```
                     LL Heuristic

NO.OF.RUNNABLES  Tcycle(mS)       WCET(uS)       Ti(mS)
---------------- ---------  ------------  ----------------

R1                05428          5               32
R2                05428          4               26
R3                10856          4               52             CORE1:1
R4                09212          3             1234
                                                                CORE2:2
                                         SUCCESS RATE OF LL HEURISTIC  91 %

#Tasks          :    10   CPU Usage:   4 %
#Task switch/sec:   619
11-26-2012  10:00:35                                                V2.00
```

Fig 4. Output of LL algorithm with efficiency 91%

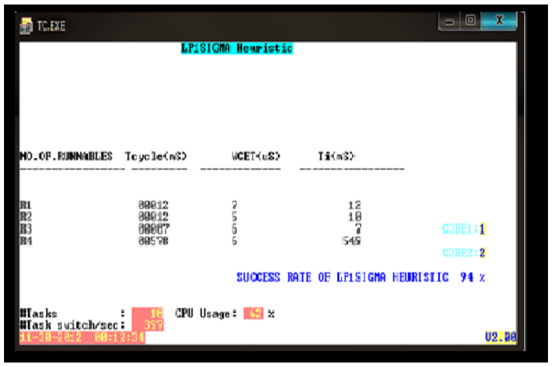
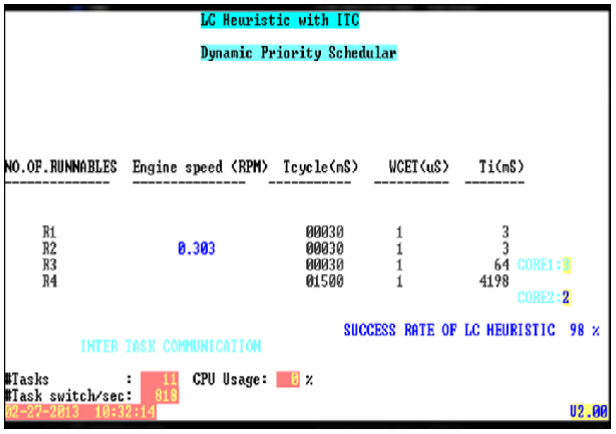

```
TC.EXE
                   LP1SIGMA Heuristic

NO.OF.RUNNABLES  Tcycle(mS)       WCET(uS)     Ti(mS)
---------------- ---------  ------------  ----------------

R1                00012          7            12
R2                00012          5            10
R3                00087          6             7             CORE1:1
R4                00578          6           545
                                                             CORE2:2
                            SUCCESS RATE OF LP1SIGMA HEURISTIC  94 %

#Tasks          :    10   CPU Usage:  62 %
#Task switch/sec:   397
11-30-2012  00:12:34                                                V2.00
```

Fig 5. Output of LP sigma algorithm with efficiency 94%



```
                      LC Heuristic with ITC

                      Dynamic Priority Schedular



NO.OF.RUNNABLES  Engine speed (RPM)  Tcycle(mS)   WCET(uS)   Ti(mS)
---------------  ------------------  ----------   --------   ------

      R1                              00030         1          3
      R2             0.303            00030         1          3
      R3                              00030         1         64 CORE1:3
      R4                              01500         1       4198
                                                                 CORE2:2
                                              SUCCESS RATE OF LC HEURISTIC  98 %
          INTER TASK COMMUNICATION

#Tasks          :   11  CPU Usage:   0 %
#Task switch/sec:  818
02-27-2013  10:32:14                                                   U2.00
```

Fig 5. Output of LC algorithm with efficiency 98%

This approach first requires the precise modeling of the data exchanges and capturing their timing constraints, for example, using the TIMMO-2-USE methodology.

REFERENCES

[1] A. Emadi, Y. Lee, and K. Rajashekara, "Power electronics and motor drives in electric, hybrid electric, and plug-in hybrid electric vehicles," *IEEE Trans. Ind. Electron.*, vol. 55, no. 6, pp. 2237–2245, Jun. 2008.

[2] F. Mapelli, D. Tarsitano, and M. Mauri, "Plug-in hybrid electric vehicle: Modeling, prototype realization, and inverter losses reduction analysis," *IEEE Trans. Ind. Electron.*, vol. 57, no. 2, pp. 598–607, Feb. 2010.

[3] D.-J. Kim, K.-H. Park, and Z. Bien, "Hierarchical longitudinal controller for rear-end collision avoidance," *IEEE Trans. Ind. Electron.*, vol. 54, no. 2, pp. 805–817, Apr. 2007.

[4] T. Bucher, C. Curio, J. Edelbrunner, C. Igel, D. Kastrup, I. Leefken, G. Lorenz, A. Steinhage, and W. von Seelen, "Image processing and behavior planning for intelligent vehicles," *IEEE Trans. Ind. Electron.*, vol. 50, no. 1, pp. 62–75, Feb. 2003.

[5] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion, "Multisource and multicore automotive ECUs—OS protection mechanisms and scheduling," in *Proc. IEEE ISIE*, 2010, pp. 3734–3741.

[6] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Trans. Comput.*, vol. 44, no. 12, pp. 1429–1442, Dec. 1995.

[7] Y. Oh and S. Son, "Fixed-priority scheduling of periodic tasks on multiprocessor systems," Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, Tech. Rep. CS-95-16, 1995.

[8] S. Lauzac, R. Melhem, and D. Mossé, "An improved rate-monotonic admission control and its applications," *IEEE Trans. Comput.*, vol. 52, no. 3, pp. 337–350, Mar. 2003.

[9] A. Karrenbauer and T. Rothvoss, "An average-case analysis for rate monotonic multiprocessor real-time scheduling," in *Proc. 17th Annu. ESA*, 2009, pp. 432–443.

[10] J. Goossens, "Scheduling of offset free systems," *Real-Time Syst.*, vol. 24, no. 2, pp. 239–258, Mar. 2003.

[11] M. Grenier, L. Havet, and N. Navet, "Pushing the limits of CAN— Scheduling frames with offsets provides a major performance boost," in *Proc. Eur. Congr. ERTS*, 2008.

[12] RealTime-at-Work, *RTaW-ECU: Static cyclic scheduling of tasks*, 2011. [Online]. Available: http://www.realtimeatwork.com

[13] AUTOSAR Consortium, AUTOSAR Release 4.0, *Specification of multicore OS architecture v1.0*, 2009.

[14] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*. New York: Dover, 1970.

[15] O. Redell and M. Törngren, "Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter," in *Proc. RTAS*, 2002, pp. 164–172.

[16] A. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE Trans. Softw. Eng.*, vol. 23, no. 10, pp. 635–645, Oct. 1997.

[17] S. Baruah, D. Chen, and A. Mok, “Static-priority scheduling of multiframe tasks,” in *Proc. 11th Euromicro Conf. Real-Time Syst.*, 1999, pp. 38–45.

[18] A. Zuhily and A. Burns, “Exact response time scheduling analysis of accumulatively monotonic multiframe real time tasks,” in *Proc. ICTAC*, 2008, pp. 410–424.