Schedulability analysis and optimisation for the synthesis of multi-cluster distributed embedded systems

P. Pop, P. Eles and Z. Peng

Abstract: An approach to schedulability analysis for the synthesis of multi-cluster distributed embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways, is presented. A buffer size and worst case queuing delay analysis for the gateways, responsible for routing inter-cluster traffic, is also proposed. Optimisation heuristics for the priority assignment and synthesis of bus access parameters aimed at producing a schedulable system with minimal buffer needs have been proposed. Extensive experiments and a real-life example show the efficiency of the approaches.

1 Introduction

Depending on the particular application, an embedded system has certain requirements for performance, cost, dependability, size etc. For hard real-time applications the timing requirements are extremely important. Hence, in order to function correctly, an embedded system implementing such an application has to meet its deadlines.

Process scheduling and schedulability analysis have been intensively studied for the past decades [1, 2]. There are two basic approaches for handling tasks in real-time applications [3]. In the event-triggered approach (ET), activities are initiated whenever a particular event is noted. In the time-triggered (TT) approach, activities are initiated at predetermined points in time. There has been a long debate in the real-time and embedded systems communities concerning the advantages of each approach [3–5]. Several aspects have been considered in favour of one or the other approach, such as flexibility, predictability, jitter control, processor utilisation, testability etc.

The same duality is reflected at the level of the communication infrastructure, where communication activities can be triggered either dynamically, in response to an event, as with the controller area network (CAN) bus [6], or statically, at predetermined moments in time, as in the case of time-division multiple access (TDMA) protocols and, in particular, the time-triggered protocol (TTP) [3].

A few approaches have been proposed for the schedulability analysis of distributed real-time systems, taking into consideration both process and communication scheduling. In [7, 8] Tindell *et al.* provided a framework for the analysis of ET process sets interconnected through an infrastructure based on either the CAN protocol or a generic TDMA protocol. In [9, 10] we have developed an analysis allowing for either TT or ET process sets communicating over the TTP.

An interesting comparison of the ET and TT approaches, from a more industrial, in particular automotive, perspective, can be found in [11]. The conclusion there is that one has to choose the right approach depending on the particularities of the processes. This means not only that there is no single 'best' approach to be used, but also that inside a certain application the two approaches can be used together, some tasks being TT and others ET. The fact that such an approach is suitable for automotive applications is demonstrated by the following two trends, which are currently considered to be of foremost importance not only for the automotive industry, but also for other categories of industrial applications:

1 The development of bus protocols that support both static and dynamic communication [12]. This allows for ET and TT processes to share the same processor as well as dynamic and static communications to share the same bus. In [13] we have addressed the problem of timing analysis for such systems.
2 Complex systems are designed as interconnected clusters of processors. Each such cluster can be either TT or ET. In a time-triggered cluster (TTC), processes and messages are scheduled according to a static cyclic policy, with the bus implementing the TTP. In an event-triggered cluster (ETC), the processes are scheduled according to a priority based pre-emptive approach, while messages are transmitted using the priority-based CAN protocol. Depending on their particular nature, certain parts of an application can be mapped on processors belonging to an ETC or a TTC. The critical element of such an architecture is the gateway, which is a node connected to both types of clusters, and is responsible for the inter-cluster routing of hard real-time traffic.

In this paper we propose an approach to schedulability analysis for the synthesis of multi-cluster distributed embedded systems, including also buffer need analysis and worst case queuing delays of inter-cluster traffic. We have also developed optimisation heuristics for the synthesis of bus access parameters as well as process and

© IEE, 2003

IEE Proceedings online no. 20030829
doi: 10.1049/ip-cdt:20030829
Paper received 16th May 2003

The authors are with the Department of Computer and Information Science, Linköping University, SE-581 83, Linköping, Sweden

message priorities aimed at producing a schedulable system such that buffer sizes are minimised.

Efficient implementation of new, highly sophisticated automotive applications entails the use of TT process sets together with ET sets implemented on top of complex distributed architectures. In this context, this paper is the first to address the analysis and optimisation of heterogeneous TT and ET systems implemented on multi-cluster embedded networks.

2 Application model and system architecture

2.1 Application model

We model an application Γ as a set of process graphs $G_i \in \Gamma$ (Fig. 1). Nodes in the graph represent processes and arcs represent dependency between the connected processes. The communication time between processes mapped on the same processor is considered to be part of the process worst case execution time and is not modelled explicitly. Communication between processes mapped to different processors is preformed by message passing over the buses and, if needed, through the gateway. Such message passing is modelled as a communication process inserted on the arc connecting the sender and the receiver process (the black dots in Fig. 1).

Each process P_i is mapped on a processor $processor_{P_i}$ (mapping represented by shading in Fig. 1), and has a worst case execution time C_i on that processor (depicted to the left of each node). The size of each message is known (in bytes, indicated to its left), and also its period, which is identical to that of the sender process. Processes and messages activated based on events also have a uniquely assigned priority, $priority_{P_i}$ for processes and $priority_{m_i}$ for messages.

All processes and messages belonging to a process graph G_i have the same period $T_i = T_{G_i}$, which is the period of the process graph. A deadline $D_{G_i} \leq T_{G_i}$ is imposed on each process graph G_i. Deadlines can also be placed locally on processes. If communicating processes are of different periods, they are combined into a hypergraph capturing all process activations for the hyperperiod (lowest common multiple of all periods).

2.2 Hardware architecture

We consider architectures consisting of several clusters, interconnected by gateways (Fig. 2 depicts a two-cluster example). A cluster is composed of nodes which share a broadcast communication channel. Every node consists, among others, of a communication controller and a CPU. The gateways, connected to both types of clusters, have two communication controllers, for TTP and CAN. The communication controllers implement the protocol services, and run independently of the node's CPU. Communication with the CPU is performed through a *message base interface* (MBI), which is usually implemented as a dual ported RAM (Fig. 3).

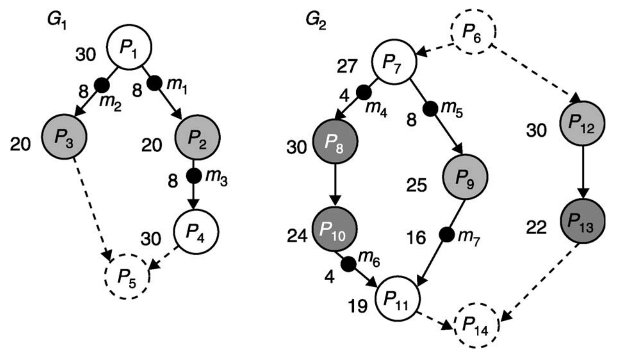


Fig. 1 *Application model example*

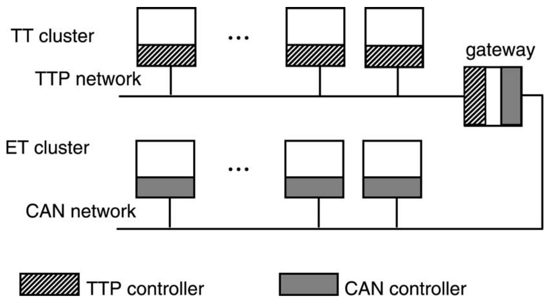


Fig. 2 *System architecture example*

Communication between the nodes on a TTC is based on the TTP [3]. The bus access scheme is TDMA, where each node N_i, including the gateway node, can transmit only during a predetermined time interval, the so-called TDMA slot S_i. In such a slot, a node can send several messages packed in a frame. A sequence of slots corresponding to all the nodes in the TTC is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table for the TTP controller, which has to know when to send/receive a frame to/from the communication channel.

On an ETC the CAN [6] protocol is used for communication. The CAN bus is a priority bus that employs a collision avoidance mechanism, whereby the node that transmits the message with the highest priority wins the contention. Message priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

2.3 Software architecture

A real-time kernel is responsible for the activation of processes and transmission of messages on each node. On a TTC, the processes are activated based on the local schedule tables, and messages are transmitted according to the MEDL. On an ETC, we have a scheduler that decides on the activation of ready processes and transmission of messages, based on their priorities.

Figure 3 illustrates the message passing mechanism. Here we concentrate on the communication between processes located on different clusters. For message passing details within a TTC the reader is directed to [14], while the infrastructure needed for communications on an ETC has been detailed in [7].

Consider the example in Fig. 3, where the process graph G_1 from Fig. 1 is mapped on to the two clusters. Processes P_1 and P_4 are mapped on node N_1 of the TTC, while P_2 and P_3 are mapped on node N_2 of the ETC. Process P_1 sends messages m_1 and m_2 to processes P_2 and P_3, respectively, while P_2 sends message m_3 to P_4.

The transmission of messages from the TTC to the ETC takes place in the following way (see Fig. 3). P_1, which is statically scheduled, is activated according to the schedule table, and when it finishes it calls the `send` kernel function in order to send m_1 and m_2, indicated in the Figure by a circled number ①. Messages m_1 and m_2 have to be sent from node N_1 to node N_2. At a certain time, known from the schedule table, the kernel transfers m_1 and m_2 to the TTP controller by packaging them into a frame in the MBI.

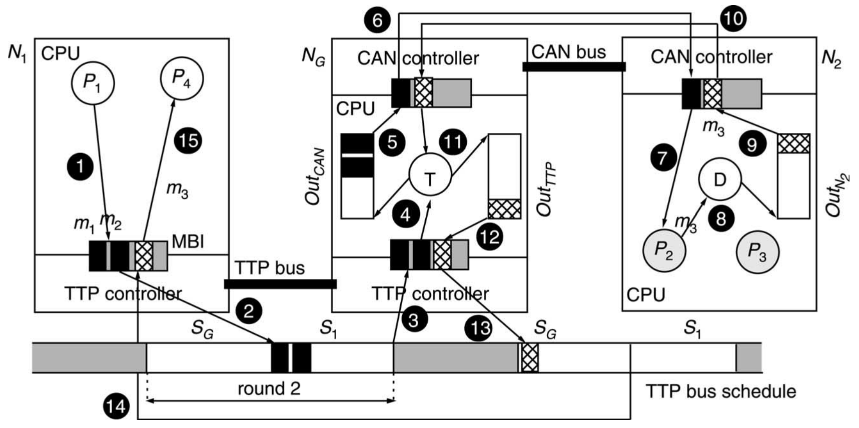


Fig. 3 *Message passing example*

Later on, the TTP controller knows from its MEDL when it has to take the frame from the MBI, in order to broadcast it on the bus. In our example, the timing information in the schedule table of the kernel and the MEDL is determined in such a way that the broadcasting of the frame is done in the slot S_1 of round two ②. The TTP controller of the gateway node N_G knows from its MEDL that it has to read a frame from slot S_1 of a round two and to transfer it into its MBI ③. Invoked periodically, having the highest priority on node N_G, and with a period which guarantees that no messages are lost, the gateway process T copies messages m_1 and m_2 from the MBI to the TTP-to-CAN priority-ordered message queue Out_{CAN} ④. The highest priority message in the queue, in our case m_1, will tentatively be broadcast on the CAN bus ⑤. Whenever message m_1 is the highest priority message on the CAN bus, it will be successfully broadcast and will be received by the interested nodes, in our case node N_2 ⑥. The CAN communication controller of node N_2 receiving m_1 will copy it in the transfer buffer between the controller and the CPU, and raise an interrupt that will activate a delivery process, responsible for activating the corresponding receiving process, in our case P_2, and hand over message m_1 that finally arrives at the destination ⑦.

Message m_3 (depicted in Fig. 3 as a hashed rectangle) sent by process P_2 from the ETC will be transmitted to process P_4 on the TTC. The transmission starts when P_2 calls its `send` function and enqueues m_3 in the priority-ordered Out_{N_2} queue ⑧. When m_3 has the highest priority on the bus, it will be removed from the queue ⑨ and broadcast on the CAN bus ⑩, arriving at the gateway's CAN controller where it raises an interrupt. Based on this interrupt, the gateway transfer process T is activated, and m_3 is placed in the Out_{TTP} FIFO queue ⑪. The gateway node N_G is only able to broadcast on the TTC in the slot S_G of the TDMA rounds circulating on the TTP bus. According to the MEDL of the gateway, a set of messages not exceeding $size_{S_G}$ of the slot S_G will be removed from the front of the Out_{TTP} queue in every round, and packed in the S_G slot ⑫. Once the frame is broadcast ⑬ it will arrive at node N_1 ⑭, where all the messages in the frame will be copied in the input buffers of the destination processes ⑮. Process P_4 is activated according to the schedule table, which has to be constructed so that it accounts for the worst-case communication delay of message m_3, bounded by the analysis in Section 4, and thus when P_4 starts executing it will find m_3 in its input buffer.

As part of our timing analysis and synthesis approach, all the local schedule tables and MEDLs are generated on the TTC, and the message and process priorities for the activities are generated on the ETC, as well as buffer sizes and bus configurations so that the global system is schedulable.

3 Problem formulation

As input to our problem we have an application Γ given as a set of process graphs mapped on an architecture consisting of a TTC and an ETC interconnected through a gateway.

We are interested first in finding a system configuration denoted by a 3-tuple $\Psi = \langle \phi, \beta, \pi \rangle$ such that the application Γ is schedulable. Determining a system configuration Ψ means deciding on:

- The set ϕ of the offsets corresponding to each process and message in the system (see Section 4). The offsets of processes and messages on the TTC practically represent the local schedule tables and MEDLs.
- The TTC bus configuration β, indicating the sequence and size of the slots in a TDMA round on the TTC.
- The priorities of the processes and messages on the ETC, captured by π.

Once a configuration leading to a schedulable application is found, we are interested in finding a system configuration that minimises the total queue sizes needed to run a schedulable application. The approach presented in this paper can be easily extended to cluster configurations where there are several ETCs and TTCs interconnected by gateways.

Consider the example in Fig. 4, where there is shown the process graph G_1 from Fig. 1 mapped on the two-cluster system, as indicated in Fig. 3. In the system configuration of Fig. 4*a* we consider that, on the TTP bus, the gateway transmits in the first slot (S_G) of the TDMA round, while node N_1 transmits in the second slot (S_1). The priorities inside the ETC have been set such that $priority_{m_1} > priority_{m_2}$ and $priority_{P_3} > priority_{P_2}$. In such a setting, G_1 will miss its deadline, which was set at 200 ms. However, changing the system configuration as in Fig. 4*b*, so that slot S_1 of N_1 comes first, we are able to send m_1 and m_2 sooner, and thus reduce the response time and meet the deadline. The response times and resource usage do not, of course, depend only on the TDMA configuration. In Fig. 4*c*, for example, we have modified the priorities of P_2 and P_3 so that P_2 is the higher priority process. In such a situation, P_2 is not interrupted when the delivery of message m_2 is supposed to activate P_3 and, thus, eliminating the interference, we are able to meet the deadline, even with the TTP bus configuration of Fig. 4*a*.

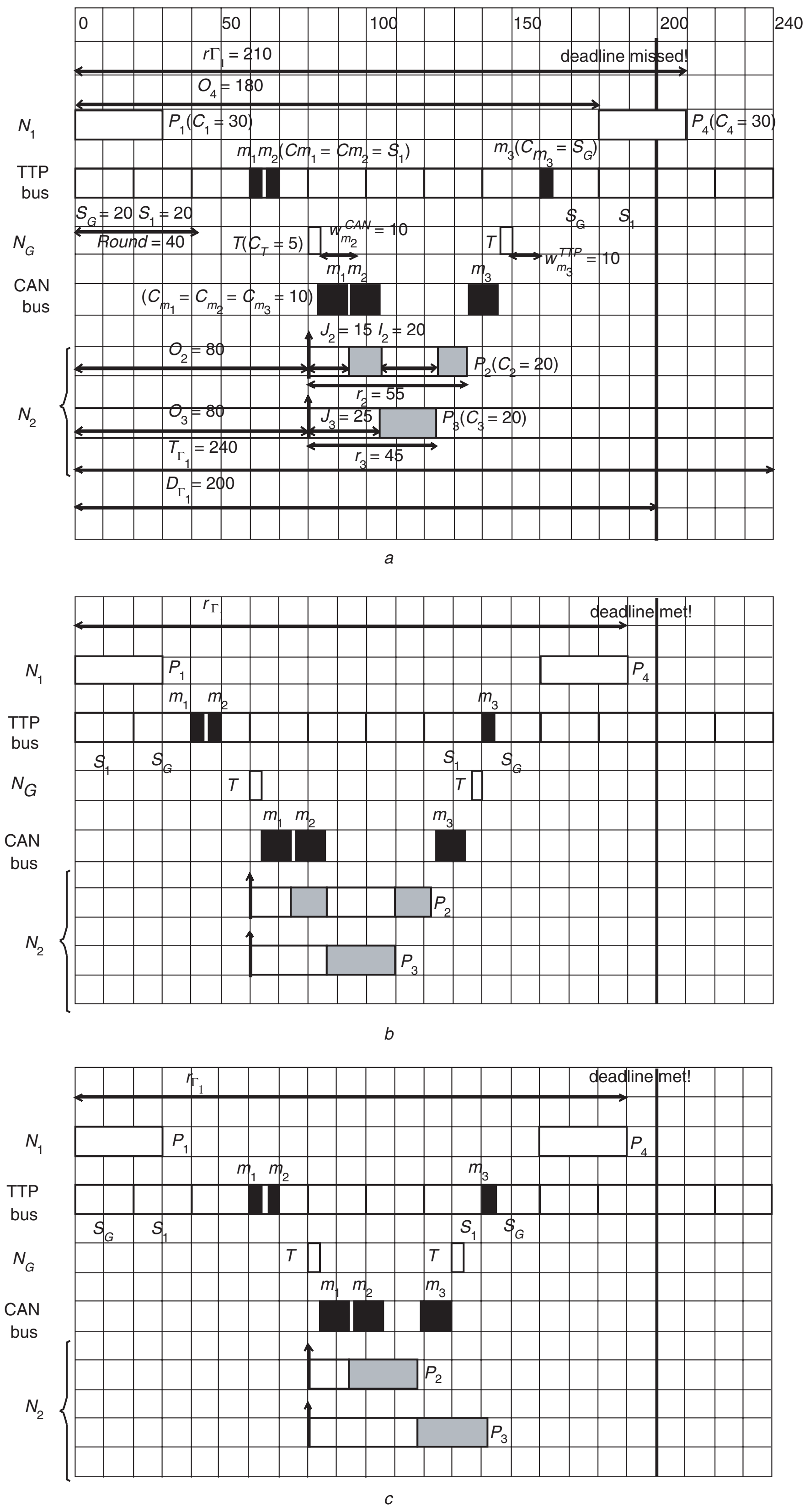


Fig. 4 *Scheduling examples*
a G_1 misses its deadline
b S_1 is the first slot, m_1, m_2 are sent sooner, G_1 meets its deadline
c P_2 is the high priority process on N_2, G_1 meets its deadline

4 Multi-cluster scheduling

In this Section we propose an analysis for hard real-time applications mapped on multi-cluster systems. The aim of such an analysis is to find out if a system is schedulable, i.e. if all the timing constraints are met. In addition, we are also interested in bounding the queue sizes.

In a TTC an application is schedulable if it is possible to build a schedule table such that the timing requirements are satisfied. In a ETC, the answer to whether or not a system is schedulable is given by a *schedulability analysis*.

In this paper, for the ETC we use a *response time analysis*, where the schedulability test consists of the comparison between the worst-case response time r_i of a process P_i and its deadline D_i. Response time analysis of data dependent processes with static priority pre-emptive scheduling has been proposed in [15–17], and has been also extended to consider the CAN protocol [7]. The authors use the concept of *offset* to handle data dependencies. Thus each process P_i is characterised by an offset O_i, measured from the start of the process graph, that indicates the earliest possible start time of P_i. For example, in Fig. 4*a*, $O_2 = 80$, as process P_2 cannot start before receiving m_1, which is available at the end of slot S_1 in round two. The same is true for messages, their offset indicating the earliest possible transmission time.

Determining the schedulability of an application mapped on a multi-cluster system cannot be addressed separately for each type of cluster, as the inter-cluster communication creates a circular dependency: the static schedules determined for the TTC influence through the offsets the response times of the processes on the ETC, which in their turn influence the schedule table construction in the TTC. In Fig. 4*a* placing m_1 and m_2 in the same slot leads to equal offsets for P_2 and P_3. Because of this, P_3 will interfere with P_2 (which would not be the case if m_2 sent to P_3 to be scheduled in round four) and thus the placement of P_4 in the schedule table has to be accordingly delayed to guarantee the arrival of m_3.

In our response time analysis we consider the influence between the two clusters by making the following observations:

- The start time of process P_i in a schedule table on the TTC is its offset O_i.
- The worst-case response time r_i of a TT process is its worst-case execution time, i.e. $r_i = C_i$ (TT processes are not pre-emptable).
- The response times of the messages exchanged between two clusters have to be calculated according to the schedulability analysis described in Section 4.1.
- The offsets have to be set by a scheduling algorithm so that the precedence relationships are preserved. This means that, if process P_B depends on process P_A, the following condition must hold: $O_B \geq O_A + r_A$. Note that, for the processes on a TTC receiving messages from the ETC, this translates to setting the start times of the processes so that a process is not activated before the worst-case arrival time of the message from the ETC. In general, offsets on the TTC are set such that all the necessary messages are present at the process invocation.

```
MultiClusterScheduling($\Gamma$, $\beta$, $\pi$)
1   -- assign initial values to offsets
2   for each $O_i \in \phi$ do
3   $O_i$ = initial value
4   end for
5
6   -- iteratively improve the offsets and response times
7   repeat
8       -- determine the response times based on
9       -- the current values for the offsets
10      $\rho$ = ResponseTimeAnalysis($\Gamma$, $\phi$, $\pi$)
11      -- determine the offsets based on
12      -- the current values for the response times
13      $\phi$ = StaticScheduling($\Gamma$, $\rho$, $\beta$)
14  until $\phi$ not changed
15
16  return $\phi$, $\rho$
end MultiClusterScheduling
```

Fig. 5 *MultiClusterScheduling algorithm*

The `MultiClusterScheduling` algorithm in Fig. 5 receives as input the application Γ, the TTC bus configuration β and the ET process and message priorities π, and produces the offsets ϕ and response times ρ. The algorithm starts by assigning to all offsets an initial value obtained by a static scheduling algorithm applied on the TTC without considering the influence from the ETC (lines 2–4). The response times of all processes and messages in the ETC are then calculated according to the analysis in Section 4.1 by using the `ResponseTimeAnalysis` function (line 10). Based on the response times, offsets of the TT processes can be defined such that all messages received from the ETC cluster are present at process invocation. Considering these offsets as constraints, a static scheduling algorithm can derive the schedule tables and MEDLs of the TTC cluster (line 13). For this purpose we use the list scheduling based approach presented in [9]. Once new values have been determined for the offsets, they are fed back to the response time calculation function, thus obtaining new, tighter (i.e. smaller, less pessimistic) values for the worst-case response times. The algorithm stops when the response times cannot be further tightened and consequently the offsets remain unchanged. Termination is guaranteed if processor and bus loads are smaller than 100% (see Section 4.2) and deadlines are smaller than the periods.

4.1 Schedulability and resource analysis

The analysis in this Section is used in the `ResponseTi-meAnalysis` function in order to determine the response times for processes and messages on the ETC. It receives as input the application Γ, the offsets ϕ and the priorities π, and it produces the set ρ of worst case response times.

We have extended the framework provided by [7, 16] for an ETC. Thus the response time of a process P_i on the ETC is:

$$r_i = J_i + w_i + C_i \tag{1}$$

where J_i is the *jitter* of process P_i (the worst-case delay between the activation of the process and the start of its execution), and C_i is its worst-case execution time. The *interference* w_i from other processes running on the same processor is given by:

$$w_i = B_i + \sum_{\forall p_j \in hp(P_i)} \left\lceil \frac{w_i + J_j - O_{ij}}{T_j} \right\rceil_0 C_j \tag{2}$$

In (2), the blocking factor B_i represents interference from lower priority processes that are in their critical section and cannot be interrupted. The second term captures the interference from higher priority processes $P_j \in hp(P_i)$, where O_{ij} is a positive value representing the relative offset of process P_j to P_i. The $\lceil x \rceil_0$ operator is the positive ceiling, which returns the smallest integer greater than x, or 0 if x is negative.

The same analysis (1) and (2) can be applied for messages on the CAN bus:

$$r_m = J_m + w_m + C_m \tag{3}$$

where J_m is the jitter of message m, which in the worst case is equal to the response time $r_{S(m)}$ of the sender process $P_{S(m)}$, w_m is the *worst-case queuing delay* experienced by m at the communication controller, and C_m is the worst-case time it takes for a message m to reach the destination controller. In CAN, C_m depends on the frame configuration and message size s_m, while in TTP it is equal to the slot size where m is transmitted.

The response time analysis for processes and messages are combined by realising that the jitter of a destination process depends on the communication delay between sending and receiving a message. Thus, for a process $P_{D(m)}$ that receives a message m from a sender process $P_{S(m)}$, the release jitter is $J_{D(m)} = r_m$.

The worst-case queueing delay for a message (w_m in (3)) is calculated differently depending on the type of message passing employed:

1 From an ETC node to another ETC node (in which case $w_m^{N_i}$ represents the worst-case time a message m has to spend in the Out_{N_i} queue on ETC node N_i). An example of such a message is m_3 in Fig. 4*a*, which is sent from the ETC node N_3 to the gateway node N_G.
2 From a TTC node to an ETC node (w_m^{CAN} is the worst-case time a message m has to spend in the Out_{CAN} queue). In Fig. 4*a*, message m_1 is sent from the TTC node N_1 to the ETC node N_2.
3 From an ETC node to a TTC node (where w_m^{TTP} captures the time m has to spend in the Out_{TTP} queue). Such message passing happens in Fig. 4*a*, where message m_3 is sent from the ETC node N_3 to the TTC node N_1 through the gateway node N_G where it has to wait for a time w_m^{TTP} in the Out_{TTP} queue.

The messages sent from a TTC node to another TTC node are taken into account when determining the offsets (`StaticScheduling`, Fig. 5), and hence are not involved directly in the ETC analysis.

We next show how the worst queueing delays and the bounds on the queue sizes are calculated for each of the previous three cases.

4.1.1 From ETC to ETC and from TTC to ETC:

The analyses for $w_m^{N_i}$ and w_m^{CAN} are similar. Once m is the highest priority message in the Out_{CAN} queue, it will be sent by the gateway's CAN controller as a regular CAN message, therefore the same equation for w_m can be used:

$$w_m = B_m + \sum_{\forall m_j \in hp(m)} \left\lceil \frac{w_m + J_j - O_{mj}}{T_j} \right\rceil_0 C_j \tag{4}$$

Intuition tells us that m has to wait, in the worst case, first for the largest lower priority message that is just being transmitted (B_m) as well as for the higher priority $m_j \in hp(m)$ messages that have to be transmitted ahead of m (the second term). In the worst case, the time it takes for the largest lower priority message $m_k \in lp(m)$ to be transmitted to its destination is:

$$B_m = \max_{\forall m_k \in lp(m)} (C_k). \tag{5}$$

Note that in our case, $lp(m)$ and $hp(m)$ also include messages produced by the gateway node, transferred from the TTC to the ETC.

We are also interested to bound the size s_m^{CAN} of the Out_{CAN} and $s_m^{N_i}$ of the Out_{N_i} queue. In the worst case, message m, all the messages with higher priority than m will be in the queue, awaiting transmission. Summing up their sizes, and finding out what is the most critical instant, we get the worst-case queue size:

$$s_{Out} = \max_{\forall m} \left(s_m + \sum_{\forall m_j \in hp(m)} \left\lceil \frac{w_m + J_j - O_{mj}}{T_j} \right\rceil_0 s_j \right) \tag{6}$$

where s_m and s_j are the sizes of message m and m_j, respectively.

4.1.2 From ETC to TTC:

The time a message m has to spend in the Out_{TTP} queue in the worst case depends on the total size of messages queued ahead of m (Out_{TTP} is a FIFO queue), the size S_G of the gateway slot responsible for carrying the CAN messages on the TTP bus, and the frequency T_{TDMA} with which this slot S_G is circulating on the bus:

$$w_m^{TTP} = B_m + \left\lceil \frac{S_m + I_m}{S_G} \right\rceil T_{TDMA} \tag{7}$$

where I_m is the total size of the messages queued ahead of m. Those messages $m_j \in hp(m)$ are ahead of m, which have been sent from the ETC to the TTC, and have higher priority than m:

$$I_m = \sum_{\forall m_j \in hp(m)} \left\lceil \frac{w_m^{TTP} + J_m - O_{mj}}{T_j} \right\rceil_0 s_j \tag{8}$$

where the message jitter J_m is in the worst case the response time of the sender process, $J_m = r_{S(m)}$.

The blocking factor B_m is the time interval in which m cannot be transmitted because the slot S_G of the TDMA round has not arrived yet, and is determined as:

$$T_{TDMA} - O_m \bmod T_{TDMA} + O_{S_G} \tag{9}$$

where O_{S_G} is the offset of the gateway slot in a TDMA round.

Determining the size of the queue needed to accommodate the worst case burst of messages sent from the CAN cluster is done by finding out the worst instant of the following sum:

$$s_{Out}^{TTP} = \max_{\forall m} (s_m + I_m) \tag{10}$$

4.2 Response time analysis example

Figure 6 presents the equations for the system in Fig. 4*a*. The jitter of P_2 depends on the response time of the gateway transfer process T and the response time of message m_1, $J_2 = r_{m_1}$. Similarly, $J_3 = r_{m_2}$. We have noted that $J_{m_1} = J_{m_2} = r_T$. The response time r_{m_3} denotes the response time of m_3 sent from process P_2 to the gateway process T, while $r_{m_3'}$ is the response time of the same message m_3 sent now from T to P_4.

The equations are recurrent, and they will converge if the processor and bus utilisation are under 100% [8].

$$r_2 = J_2 + w_2 + C_2,\ w_2 = B_2 + \left\lceil \frac{w_2 + J_3 - O_{2,3}}{T} \right\rceil C_3$$

$$r_3 = J_3 + w_3 + C_3,\ w_3 = B_3$$

$$r_{m_1} = J_{m_1} + w_{m_1}^{CAN} + C_{m_1},\ w_{m_1}^{CAN} = B_{m_1}$$

$$r_{m_2} = J_{m_2} + w_{m_2}^{CAN} + C_{m_2},\ w_{m_2}^{CAN} = B_{m_2} + \left\lceil \frac{w_{m_2} + J_{m_1} - O_{m_2,m_1}}{T} \right\rceil C_3$$

$$r_{m_3} = J_{m_3} + w_{m_3}^{N_2} + C_{m_3}$$

$$r_{m_3'} = J_{m_3'} + w_{m_3'}^{TTP} + C_{m_3'},\ w_{m_3'}^{TTP} = B_{m_3'} + \left\lceil \frac{S_{m_3}}{T} \right\rceil T_{TDMA},$$

$$B_{m_3'} = T_{TDMA}\, O_{m_3} \bmod T_{TDMA} + O_{S_G}$$

Fig. 6 *Response time analysis example*

Considering a TDMA round of 40 ms, with two slots each of 20 ms as in Fig. 4*a*, $r_T = 5$ ms, 10 ms for the transmission times on CAN for m_1 and m_2, and using the offsets in the Figure, the equations will converge to the values indicated in Fig. 4*a* (all values are in milliseconds). Thus, the response time of graph G_1 will be $r_{G_1} = O_4 + r_4 = 210$, which is greater than $D_{G_1} = 200$, hence the system is not schedulable.

5 Scheduling and optimisation strategy

Once we have a technique to determine if a system is schedulable, we can concentrate on optimising the total queue sizes. Our problem is to synthesise a system configuration ψ such that the application is schedulable, i.e. the condition [Note 1]

$$r_{G_j} \leq D_{G_j}, \quad \forall G_j \in \Gamma_i \tag{11}$$

holds, and the total queue size s_{total} is minimised [Note 2]:

$$s_{total} = s_{Out}^{CAN} + s_{Out}^{TTP} + \sum_{\forall (N_i \in ETC)} s_{Out}^{N_i} \tag{12}$$

Such an optimisation problem is NP-complete, hence obtaining the optimal solution is not feasible. We propose a resource optimisation strategy based on a hill-climb heuristic that uses an intelligent set of initial solutions in order to explore the design space efficiently.

5.1 Scheduling and buffer optimisation heuristic

The optimisation heuristic is outlined in Fig. 7. The basic idea of the `OptimizeResources` heuristic is to find, as a first step, a solution with the smallest possible response times, without considering the buffer sizes, in the hope of finding a schedulable system. This is achieved through the `OptimizeSchedule` function. Then, a hill-climbing heuristic iteratively performs moves intended to minimise the total buffer size while keeping the resultant system schedulable.

The `OptimizeSchedule` function outlined in Fig. 8 is a greedy approach that determines an ordering of the slots and their lengths, as well as priorities of messages and processes in the ETC, so that the degree of schedulability of the application is maximised. The *degree of schedulability* [10] is calculated as:

$$\delta_\Gamma = \begin{cases} f_1 = \sum_{i=1}^{n} max(0, R_i - D_i), & \text{if } f_1 > 0 \\ f_2 = \sum_{i=1}^{n} (R_i - D_i), & \text{if } f_1 = 0 \end{cases}$$

where n is the number of process graphs in the application. If the application is not schedulable, the term f_1 will be positive, and in this case the cost function is equal to f_1. However, if the process set is schedulable, $f_1 = 0$ and we use f_2 as a cost function, as it is able to differentiate between two alternatives, both leading to a schedulable process set.

Note 1: The worst-case response time for a process graph G_i is calculated based on its sink node as $r_{G_i} = O_{sink} + r_{sink}$. If local deadlines are imposed, they will also have to be tested in the schedulability condition.

Note 2: In the TTC, the synchronisation between processes and the TDMA bus configuration is solved through the proper synthesis of schedule tables, hence no output queues are needed. Input buffers on both TTC and ETC nodes are local to processes. There is one buffer per input message and each buffer can store one message instance (see explanation to Fig. 3).

```
**OptimizeResources**($\Gamma$)

-- Step 1: try to find a schedulable system
*seed_solutions* = OptimizeSchedule($\Gamma$)
-- if no schedulable configuration has been found,
-- modify mapping and/or architecture
**if** $\Gamma$ is not schedulable for $\psi_{best}$ **then**
    modify mapping
    **go to** Step 1
**end if**


-- Step 2: try to reduce the resource need, minimize $s_{total}$
**for each** $\psi$ in *seed_solutions* **do**
    **repeat**
        -- find moves with highest potential to minimize $s_{total}$
        *move_set* = GenerateNeighbors($\psi$)
        -- select move which minimizes $s_{total}$
        -- and does not result in an un-schedulable system
        *move* = SelectMove(*move_set*)
        Perform(*move*)
    **until** $s_{total}$ has not changed **or** limit reached
**end for**

return system configuration $\psi$, queue sizes
**end** OptimizeResources
```

Fig. 7 *OptimizeResources Algorithm*

For a given set of optimisation parameters leading to a schedulable process set, a smaller f_2 means that we have improved the response times of the processes.

As an initial TTC bus configuration β, `OptimizeSchedule` assigns in order nodes to the slots and fixes the slot length to the minimal allowed value, which is equal to the length of the largest message generated by a process assigned to N_i, $\mathtt{S_i} = \langle \mathtt{N_i}, \mathtt{size_{smallest}} \rangle$ (line 5 in Fig. 8). Then the algorithm starts with the first slot (line 8) and tries to find the node which, when transmitting in this slot, will maximise the degree of schedulability δ_Γ (lines 9–33).

Simultaneously with searching for the right node to be assigned to the slot, the algorithm looks for the optimal slot length. Once a node is selected for the first slot and a slot length fixed ($\mathtt{S_i} = \mathtt{S_{best}}$, line 32), the algorithm continues with the next slots, trying to assign nodes (and to fix slot lengths) from those nodes that have not yet been assigned. When calculating the length of a certain slot we consider the feedback from the `MultiClusterScheduling` algorithm, which recommends slot sizes to be tried out. Before starting the actual optimisation process for the bus access scheme, a scheduling of the initial solution is performed, which generates the recommended slot lengths. We refer the reader to [9] for details concerning the generation of the recommended slot lengths.

In the `OptimizeSchedule` function the degree of schedulability δ_Γ is calculated based on the response times produced by the `MultiClusterScheduling` algorithm (line 18). For the priorities used in the response time calculation we use the 'heuristic optimised priority assignment' (HOPA) approach (line 15) from [18], where priorities for processes and messages in a distributed real-time system are determined, using knowledge of the factors that influence the timing behaviour, so that the degree of schedulability is improved.

The `OptimizeSchedule` function also records the best solutions in terms of δ_Γ and s_{total} in the `seed_solutions` list in order to be used as the starting point for the second step of our `OptimizeResources` heuristic.

Once a schedulable system is obtained, our goal is to minimise the buffer space. Our design space exploration in

```
**OptimizeSchedule($\Gamma$)**
-- given an application $\Gamma$ produces the configuration $\psi = <\Phi, \beta, \pi>$
-- leading to the smallest $\delta_\Gamma$

-- start by determining an initial TTC bus configuration $\beta$
for each slot $S_i \in \beta$ do $S_i = <N_i, size_{smallest}>$ end for

-- find the best allocation of slots, the TDMA slot sequence
for each slot $S_i \in \beta$ do
    for each node $N_j \in TTC$ do
        -- allocate $N_j$ tentatively to $S_i$, $N_i$ gets slot $S_j$
        $S_i = <N_j, size_{Sj}>$; $S_j = <N_i, size_{Si}>$
        -- determine best size for slot $S_i$
        for each slot $size \in recomended\_lengths(S_i)$ do
            -- calculate the priorities according to HOPA heuristic
            $\pi$ = HOPA
            -- determine the offsets $\phi$, thus obtaining a complete system configuration $\psi$
            $S_i = <N_j, size>$
            $\phi$ = MultiClusterScheduling($\Gamma$, $\beta$, $\pi$)
            $\psi_{current} = <\phi, \beta, \pi>$
            -- remember the best configuration so far, add it to the seed configurations
            if $\delta_\Gamma(\psi_{current})$ is best so far then
                $\psi_{best} = \psi_{current}$
                $S_{best} = S_i$;
                add $\psi_{best}$ to seed_solutions
            end if
            determine $s_{total}$ for $\psi_{current}$
            if $s_{total}$ is best so far and $\Gamma$ is schedulable
            then add $\psi_{current}$ to seed_solutions end if
        end for
    end for
    -- make binding permanent, use the $S_{best}$ corresponding to $\psi_{best}$
    if a $S_{best}$ exists then $S_i = S_{best}$ end if
end for

return $\psi_{best}$, $\delta_\Gamma(\psi_{best})$, seed_solutions
end OptimizeSchedule
```

Fig. 8 *OptimizeSchedule Algorithm*

the second step of `OptimizeResources` (lines 12–22 in Fig. 7) is based on successive design transformations (generating the neighbours of a solution) called *moves*. For our heuristics, we consider the following types of moves:

- moving a process or a message belonging to the TTC inside its [*ASAP*, *ALAP*] interval calculated based on the current values for the offsets and response times
- swapping the priorities of two messages transmitted on the ETC, or of two processes mapped on the ETC
- increasing or decreasing the size of a TDMA slot with a certain value
- swapping two slots inside a TDMA round.

The second step of the `OptimizeResources` heuristic starts from the seed solutions (line 13) produced in the previous step, and iteratively preforms moves to reduce the total buffer size, s_{total} (12). The heuristic tries to improve on the total queue sizes, without producing unschedulable systems. The neighbours of the current solution are generated in the `GenerateNeighbours` functions (line 16), and the move with the smallest s_{total} is selected using the `SelectMove` function (line 19). Finally the move is performed, and the loop reiterates. The iterative process ends when there is no improvement achieved on s_{total}, or a limit imposed on the number of iterations has been reached (line 21).

To improve the chances to find good values for s_{total}, the algorithm has to be executed several times, starting with a different initial solution. The intelligence of our `OptimizeResources` heuristic lies in the selection of the initial solutions, recorded in the `seed_solutions` list. The list is generated by the `OptimizeSchedule` function, which records the best solutions in terms of δ_Γ and s_{total}. Seeding the hill-climbing heuristic with several solutions of small s_{total} will guarantee that the local optima are quickly found. However, during our experiments, we have observed that another good set of seed solutions are those that have a high degree of schedulability δ_Γ. Starting from a highly schedulable system will permit more iterations until the system degrades to an unschedulable configuration, thus the exploration of the design space is more efficient.

6 Experimental results

For evaluation of our algorithms we first used process graphs generated for experimental purpose. We considered two-cluster architectures consisting of 2, 4, 6, 8 and 10 nodes, half on the TTC and the other half on the ETC, interconnected by a gateway. Forty processes were assigned to each node, resulting in applications of 80, 160, 240, 320 and 400 processes. Message sizes were randomly chosen between 8 and 32 bytes. Thirty examples were generated for each application dimension; thus a total of 150 applications were used for experimental evaluation. Worst-case execution times and message lengths were assigned randomly using both uniform and exponential distribution. All experiments were run on a Sun Ultra 10 computer.

To provide a basis for the evaluation of our heuristics, we have developed two simulated annealing (SA) based algorithms. Both are based on the moves presented in the previous section. The first one, named SA schedule (SAS), was set to preform moves such that δ_Γ is minimised. The second one, SA resources (SAR), uses s_{total} as the cost function to be minimised. Very long and expensive runs have been performed with each of the SA algorithms, and the best ever solution produced has been considered to be close to the optimum value.

The first result concerns the ability of our heuristics to produce schedulable solutions. We have compared the degree of schedulability δ_Γ obtained from our `OptimizeSchedule` (OS) heuristic (Fig. 8) with the near-optimal values obtained by SAS. Figure 9 presents the average percentage deviation of the degree of schedulability produced by OS from the near-optimal values obtained with SAS. Together with OS, a straightforward approach (SF) is presented. For SF we considered a TTC bus configuration consisting of a straightforward ascending order of allocation of the nodes to the TDMA slots; the slot lengths were selected to accommodate the largest message sent by the respective node, and the scheduling has been performed by the `MultiClusterScheduling` algorithm (Fig. 5).

Figure 9 shows that, when considering the optimisation of the access to the communication channel, and of priorities, the degree of schedulability improves dramatically compared with the straightforward approach. The greedy heuristic `OptimizeSchedule` performs well for all the graph dimensions, having run-times that are more than two orders of magnitude smaller than with SAS. In the Figure, only the examples in which all the algorithms have obtained schedulable systems were presented. The SF approach failed to find a schedulable system in 26 out of the total 150 applications.

Next, we are interested in evaluating the heuristics for minimising the buffer sizes needed to run a schedulable application. We compare the total buffer need s_{total} obtained by the `OptimizeResources` (OR) function with the near-optimal values obtained when using simulated annealing, this time with the cost function s_{total}. To find out how relevant the buffer optimisation problem is, we have compared these results with the s_{total} obtained by the OS

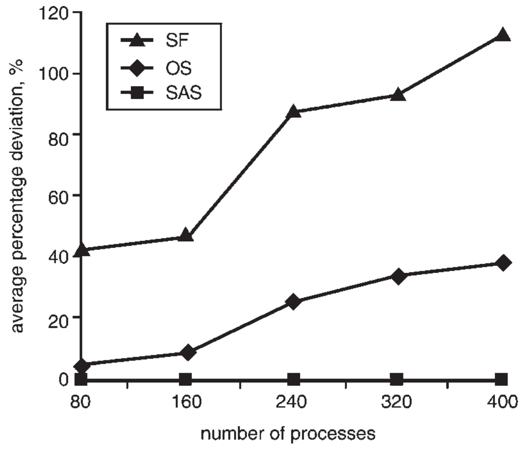
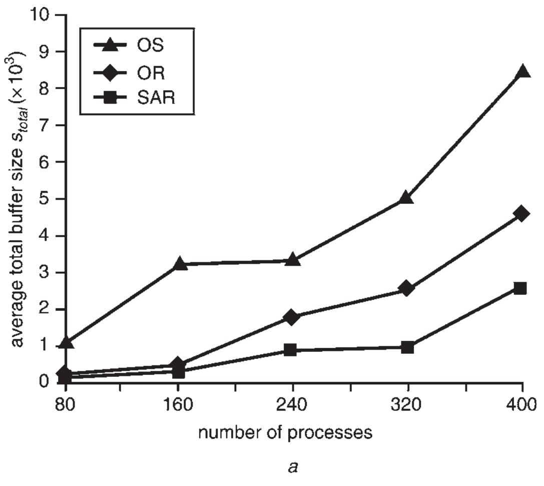


Fig. 9 *Comparison of the scheduling optimisation heuristics*



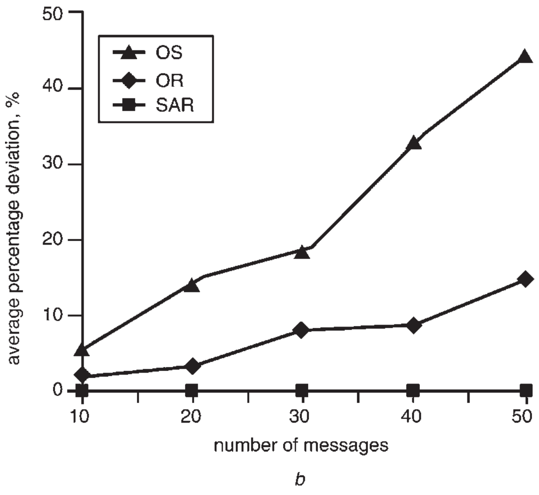


Fig. 10 *Comparison of the buffer size minimisation heuristics*
a Bounds on total buffer size obtained with OS, OR, SAS
b Percentage deviations for OS, OR from SAR

approach, which is concerned only with obtaining a schedulable system, without any other concern. As shown in Fig. 10*a*, OR is able to find schedulable systems with a buffer need half that of the solutions produced with OS. The quality of the solutions obtained by OR is also comparable with the one obtained with simulated annealing (SAR).

Another important aspect of our experiments was to determine the difficulty of resource minimisation as the number of messages exchanged over the gateway increases. For this, we generated applications of 160 processes with 10, 20, 30, 40 and 50 messages exchanged between the TTC and ETC clusters. Thirty applications were generated for each number of messages. Fig. 10*b* shows the average percentage deviation of the buffer sizes obtained with OR and OS from the near-optimal results obtained by SAR. As the number of inter-cluster messages increases, the problem becomes more complex. The OS approach degrades very fast, in terms of buffer sizes, while the OR approach is able to find good quality results even for intense inter-cluster traffic.

When deciding on which heuristic to use for design space exploration or system synthesis, an important issue is the execution time. On average, our optimisation heuristics needed a couple of minutes to produce results, and hence they can be used inside a design space exploration loop. Although the simulated annealing approaches (SAS and SAR) had an execution time of up to three hours, they produced new-optimal results, and hence are suited for the final stages of the synthesis process.

Finally, we considered a real-life example implementing a vehicle cruise controller. The process graph that models the cruise controller has 40 processes, and it was mapped on to an architecture consisting of a TTC and an ETC, each with two nodes, interconnected by a gateway. The 'speedup' part of the model has been mapped on the ETC while the other processes were mapped on the TTC. We considered one mode of operation with a deadline of 250 ms. The straightforward approach SF produced an end-to-end response time of 320 ms, greater than the deadline, while both the OS and SAS heuristics produced a schedulable system with a worst-case response time of 185 ms. The total buffer need of the solution determined by OS was 1020 bytes. After optimisation with OR a still schedulable solution with a buffer need reduced by 24% has been generated, which is only 6% worse than the solution produced with SAR.

7 Conclusions

This paper has presented an approach to schedulability analysis for the synthesis of multi-cluster distributed embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. The main contribution is the development of a schedulability analysis for such systems, including determining the worst-case queuing delays at the gateway and the bounds on the buffer size needed for running a schedulable system.

Optimisation heuristics for system synthesis have been proposed, together with simulated annealing approaches tuned to find near-optimal results. The first heuristic, OS, was concerned with obtaining a schedulable system by maximising the degree of schedulability. Our second heuristic, OR, was aimed at producing schedulable systems with a minimal required buffer size.

8 Acknowledgment

The authors are grateful to the industrial partners at Volvo Technological Development, Gothenburg, Sweden, for their close involvement and thoughtful feedback during this work.

9 References

1 Audsley, N.C., Burns, A., Davis, R.I., Tindell, K.W., and Wellings, A.J.: 'Fixed priority pre-emptive scheduling: an historical perspective', *Real-Time Syst.*, 1995, **8**, pp. 173–198
2 Balarin, F., Lavagno, L., Murthy, P., and Sangiovanni-Vincentelli, A.: 'Scheduling for embedded real-time systems', *IEEE Des. Test Comput.*, 1998, **15**, (1), pp. 71–82
3 Kopetz, H.: 'Real-time systems—design principles for distributed embedded applications' (Kluwer Academic Publishers, Norwell, MA, 1997)
4 Audsley, N., Tindell, K., and Burns, A.: 'The end of line for static cyclic scheduling?'. Proc. 5th Euromicro Workshop on Real-Time Systems, Oulu, Finland, 22–24 June 1993, pp. 36–41
5 Xu, J., and Parnas, D.L.: 'On satisfying timing constraints in hard-real-time systems', *IEEE Trans. Softw. Eng.*, 1993, **19**, (1), pp. 70–84
6 'CAN Specification Version 2.0', Robert Bosch GmbH, 1991
7 Tindell, K., Burns, A., and Wellings, A.J.: 'Calculating CAN message response times', *Control Eng. Pract.*, 1995, **3**, (8), pp. 1163–1169
8 Tindell, K., and Clark, J.: 'Holistic schedulability analysis for distributed hard real-time systems', *Microprocess. Microprogr.*, 1994, **50**, (2–3), pp. 117–134
9 Eles, P., Doboli, A., Pop, P., and Peng, Z.: 'Scheduling with bus access optimization for distributed embedded systems', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2000, **8**, (5), pp. 472–491
10 Pop, P., Eles, P., and Peng, Z.: 'Bus access optimization for distributed embedded systems based on schedulability analysis'. Proc. Design, Automation and Test in Europe Conf., Paris, France, 27–30 March 2000, pp. 567–574
11 Lönn, H., and Axelsson, J.: 'A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications'. Proc. 11th Euromicro Conf. on Real-Time Systems, York, UK, 9–11 June 1999, pp. 142–149
12 'FlexRay Requirements Specification', http://www.flexray-group.com/
13 Pop, T., Eles, P., and Peng, Z.: 'Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems'. Int. Symp. on Hardware/Software Codesign, Estes Park, CO, 6–8 May 2002, pp. 187–192
14 Pop, P., Eles, P., and Peng, Z.: 'Scheduling with optimized communication for time-triggered embedded systems'. Proc. 7th Int. Workshop on Hardware/Software Codesign, Rome, Italy, 3–5 May 1999, pp. 178–182
15 Palencia, J.C., and González Harbour, M.: 'Schedulability analysis for tasks with static and dynamic offsets'. Proc. 19th IEEE Real-Time Systems Symp., Madrid, Spain, 2–4 December 1998, pp. 26–37
16 Yen, T.Y., and Wolf, W.: 'Hardware-software co-synthesis of distributed embedded systems' (Kluwer Academic Publishers, Norwell, MA, 1997)
17 Tindell, K.: 'Adding time-offsets to schedulability analysis'. Technical Report Number YCS-94-221, Department of Computer Science, University of York, 1994
18 Gutiérrez García, J.J., and González Harbour, M.: 'Optimized priority assignment for tasks and messages in distributed hard real-time systems'. Proc. 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, CA, April 1995, pp. 124–132