Time and event triggered communication scheduling for automotive applications

ROGER JOHANSSON

CHALMERS LINDHOLMEN UNIVERSITY COLLEGE
Göteborg, Sweden 2004

Report No.17

REPORT No. 17

Time and event triggered communication scheduling for automotive applications [1]

ROGER JOHANSSON

Department of Electrical and Computer Engineering
Chalmers Lindholmen University College
roger@chl.chalmers.se

Chalmers Lindholmen University College

Göteborg, Sweden, September, 2004

[1] This work has been conducted within the Centre of Excellence CHARMEC (CHAlmers Railway MEChanics) – a VINNOVA Competence Center under the "Programme area 4, SD3. Computer control of braking systems for freight trains"

Time and event triggered communication scheduling for automotive applications

ROGER JOHANSSON

Report – Time and event triggered communication scheduling for automotive applications

Report no. 17

ISSN 1404-5001

Chalmers Lindholmen University College

P.O. Box 8873

SE-402 72 Göteborg

Sweden

Telephone + 46 31 772 1000

Chalmers Lindholmen University College

Göteborg, Sweden, September, 2004

Abstract

The CAN protocol has become the primary choice of protocol for real time communication sub systems within automotive electronics applications by a number of European car and heavy truck manufacturers. This is despite the fact that CAN, according to many opinions, is unsuitable for distributed real-time applications with hard real-time requirements due to the protocol's bus contention. This mechanism is likely to introduce jitter, which sometimes is highly undesirable regardless of application criticality. The latest protocol specification *TTCAN* (*Time Triggered CAN*), addresses exactly this problem. With the TTCAN extension the CAN protocol has matured. It now support time-triggered operation as well as priority based event triggered message handling. In this paper, we will present a method for message scheduling and analysis in real time systems using TTCAN/CAN hybrid communication systems. We have applied our model and used our scheduling method on a non-trivial set of messages designed to be representative for a real application. The message set consists of 65 different messages, divided into three classes based on their respective timing requirements. We describe our method as well as our results. Our primary objective with this paper is to provide the reader with new ideas and concepts addressing application-derived problems in real-time communication scheduling. We concentrate on heuristic methods to solve engineering problems rather than formal methods to solve general scheduling problems.

Keywords:

Hard real-time requirements, automotive electronics applications, heuristic scheduling

1 INTRODUCTION

When designing embedded systems for automotive applications a common engineering approach is to integrate as much functionality as possible into as few hardware computer nodes as possible. Consequently, a cluster of nodes is likely to exchange messages with a broad range of timing requirements via the communication bus. Over time, the Controller Area Network (CAN) protocol has become the primary choice of protocol for real time communication sub systems within automotive electronics applications by a number of European car and heavy truck manufacturers. There are currently numerous of CAN-implementations in a wide range of automotive applications in operation, and has so been, for several years. It is a common opinion that only time triggered protocols can guarantee response times and minimize message jitter delay at the same time. The Time Triggered Can (TTCAN) extension to the CAN protocol is designed to meet such requirements.

1.1 The CAN protocol

The Controller Area Network (CAN) [1] is a serial communications protocol, which supports distributed real-time control applications. CAN is a multi-master, broadcast protocol with collision detect and a resolution mechanism based on message priorities. I.e. each message on the CAN bus has a unique priority and is only transmitted from a single node on the bus. In particular CAN is used in higher layer protocol for automotive applications, e.g. SAE J1939 protocol [2] that has become a ‘de facto’ standard for commercial trucks manufacturers throughout the world. Another example is Volcano [3] used by Volvo Cars in recent car models. Thus, CAN is used in automotive electronics such as engine control modules, transmission control modules, anti brake-lock systems and anti skid systems, just to mention a few, with bit rates up to 1 Mbit/s.

It has sometimes been argued that CAN is unsuitable for dependable, distributed real-time applications due to the protocol’s bus contention. I.e. the protocol is unable to guarantee deadlines for messages transmitted on the bus [4]. Others have demonstrated that this is only a matter of how the application is implemented and that the protocol easily can be applied on the assumption that a carefully designed static schedule designed and analyzed pre run-time, is used, see for example references [5] and [6]. Right or wrong, the CAN protocol specification has evolved, from its initial release in 1986, also known as ‘Standard CAN’ through the extended protocol definition in 1991, ‘Extended CAN’, and recently a new specification where, in particularly, time triggered communication issues are addressed.

1.2 TTCAN

Time Triggered Can (TTCAN) is specified as an additional layer to the CAN standard, se reference [7]. There are two possible levels in TTCAN, level 1 provides time-triggered operation using a cycle time, and level 2 provides increased synchronization quality, global time and external clock synchronization. In both levels, timing features are based on local clocks from which each node derives its opinion of the Network Time Unit (NTU). Each node is allowed to have a slightly diverging view of NTU. Synchronization is accomplished by means of a reference message that starts a basic cycle. Thus the length of a basic cycle determines maximum allowed clock drifts in the nodes attached to a TTCAN network. When all nodes are synchronized, any message may be transmitted at a specific time slot, without competing with other messages for the bus. The loss of buss arbitration is thus avoided and any latency time or jitter becomes predictable.

Membership at application level is supported by protocol implicit atomic broadcast property. The protocol’s error handling assures that a message is either received correctly by all controllers or, in the case that at least one controller fails to receive the message; an “error frame” is transmitted by that controller, thus destroying the remaining message. The result is that no other controller will receive a correct message either. TTCAN provides hardware

supported global time while software implemented clock synchronization is required in a standard CAN system. .

With the TTCAN extension the CAN protocol has indeed matured. It now support priority based event triggered message handling as well as time-triggered operation. Furthermore, there are numerous of CAN-implementations in a wide range of real-time applications in operation, and has so been, for several years. In particular the new extension to the CAN protocol might be of interest within new as well as existing applications addressed by the automotive industries.

1.3 Related Work

There is a substantial amount of work done on CAN from different aspects and even more on real time systems and scheduling methods. Stankovic et. al. (reference [8]) provides an overview of classical scheduling results for real time systems. This is good background reading for anyone who wishes to become acquainted with this field.

Our CAN communication scheduling analysis is based on Rate Monotonic Scheduling Analysis, which was introduced by Liu and Layland in reference [9]. Sha et. al., reference [10], presents a matured framework with a generalized approach on which work in references [5] and [6] was based.

Methods that assumes a worst case scenario often tends to give overly pessimistic results. In reference [11] a method, striving at reducing overestimations is presented.

Real time communication scheduling analysis sometimes faces the problem of handling unreliable communication media. In reference [12] a communication fault hypothesis is used and a method for handling analysis in the presence of faults is presented.

Reference [13] gives an introduction to the use of TTCAN in practice. A scheduling method where jitter is minimized in TTCAN networks is presented in reference [14].

1.4 Objectives and outline

Our primary objective with this paper is to provide the reader with new ideas and concepts addressing application-derived problems in real-time communication scheduling. We concentrate on heuristic methods to solve engineering problems rather than formal methods to solve general scheduling problems. This approach should make this paper of interests particularly for experienced engineers with focus on applied research working within this area. The text may also serve as an introduction to TTCAN scheduling for anyone interested, but still not familiar to the subject.

This paper is outlined as follows; in chapter two, we review an important result from earlier efforts on the use of CAN in hard real time applications and summarize a common computational model that applies to CAN bus message scheduling. In chapter three, we introduce an extended model, which adds the concepts of Time Triggered CAN, resulting in our new “hybrid” method. In chapter four, we apply our new method on a non-trivial message set. In chapter five we summarize our results, present our conclusions and give some suggestions on future work.

2 BASIC COMPUTATIONAL MODEL

This entire chapter is based on results from previous work on fixed priority scheduling, in particular, we will review the adaptation of rate monotonic scheduling analysis for CAN based communication system found in reference [6]. Readers are supposed to have a brief knowledge of the CAN protocol properties. Readers familiar with CAN as well as work described in [6] may skip this chapter without loss of continuity.

The ”response time” of a CAN-message is considered to be the time interval from that the message was eligible for transmission until the time it was acknowledged and thus successfully received by any (i.e. all) other node (nodes) on the CAN-bus. The response time thus constitutes the total occupation of the bus for a successful transmission. The message does not have to be repeated in any sense and will thus not demand any further bus resource. Each successful transmission is also considered as a successful atomic broadcast since the CAN-protocol insures that a single acknowledge guarantees a through out correct reception of the message (otherwise an error frame would have disturbed the message). The “delivery time” of a CAN message is considered to be the time interval from that the message is delivered by an application in a node until it becomes available for other application in other nodes.

For each message we (a priori) must have established:

- A unique priority *m*, i.e. the CAN identifier.
- Length of the message *m.*
- The *m* message period time *p*.

We consider the CAN-bus to be an *indivisible* resource i.e. once allocated it cannot be shared. (This is most appropriate applied to the CAN-protocol due to its bus arbitration).

We furthermore assume a *constant transmission time*, this is also straightforward since we know each message length (number of bits in the message) as well as the CAN bus baud rate (bits/second).

A message transmission is “non-pre-emptive” i.e. once its started it’s guaranteed to complete (also follows from the CAN-protocol). The schedule is determined by CAN message identifiers a priori and we use *Fixed Priority Scheduling* analysis to compute the response time of each CAN message.

The total message delivery time D_m for a message *m* is the time from which it has been disposed by an application to a CAN controller in the sending node, until the message is available for another application in the receiving node. The message delivery time is then contributed to by:

- Time for preparing (formatting) the message for transmission on the CAN-bus.
- Queuing, i.e. waiting time due to lost bus arbitration.
- Transmission time depends on the message length and the bit rate.
- Time for unpacking (de-format) the message and notify the application in the receiver node.

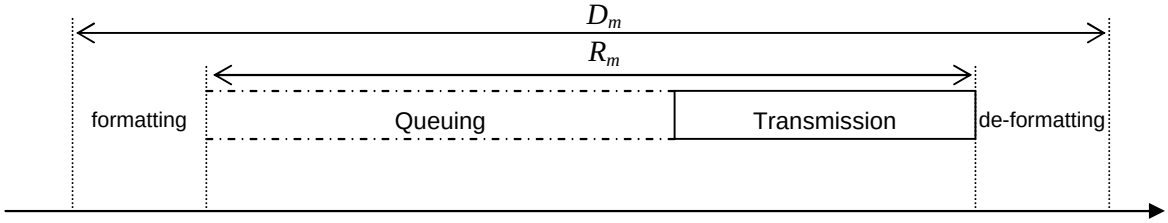

Figure 2.1: Message delivery time

While the time for preparing and unpacking normally is considered constant (depending on the actual CAN controller and operating system) and the transmission time may be calculated for each message, the queuing time depends on the actual schedule. Then, for a message *m*, in a set (schedule) of *N* periodic messages ($m = 1..N$) with period p_m and transmission time T_m the message response time R_m is bounded by:

$$R_m = Q_m + T_m$$

where:

- Q_m is a queuing time for message *m* as a result of higher priority messages transmitted and thus delayed the message *m*.
- T_m is the transmission time for message *m*.

Note that messages with $R_m > p_m$ are not guaranteed to be transmitted.

The queuing term is illustrated by the following example; assume a time interval *t*. where three messages, M1, M2 and M3 *m* =1,2 and 3, are to be transmitted from different nodes. It can be shown that the worst-case queuing time occurs if all three messages arrive (become eligible for transmission) at the same time. Further, assume that message 1 has the highest priority and message 3 has the lowest priority. Message 1 will not exhibit any delay from queuing since the CAN-bus arbitration mechanism will resolve the bus conflict in favor to message 1.

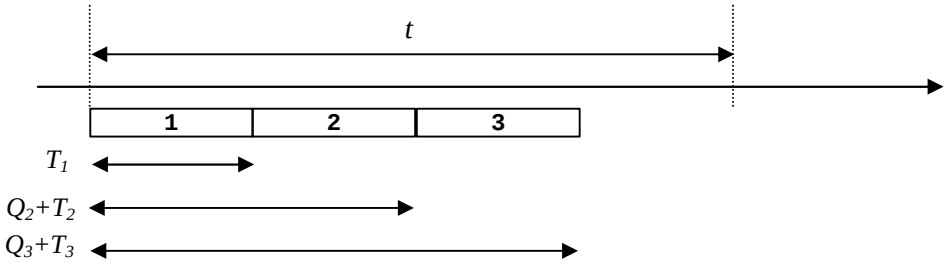

Figure 2.2: Queuing, awaiting transmission on the CAN bus

In general, the queuing for message *m* is determined by:

$$Q_m = \sum_{\forall j:hp(m)}^{1} \left\lceil \frac{Q_m}{p_j} \right\rceil T_j \qquad \textit{Equation 2.1}$$

- T_j is the transmission time for a higher priority message *j* and p_j is the period time of a higher priority message *j*.

I.e. during examining of all higher priority messages ($j \subset hp(m)$), where: $Q_1 = 0$, i.e. highest priority message can not be delayed by queuing, *Equation 2.1* will converge to the solution giving a maximum queuing time.

Due to the non pre-emptive property of a CAN message transmission, we also has to consider blocking, which causes additional queuing delay imposed from a message with *lower* priority. Consider the following situation where a high priority message suffers delay due to such blocking.

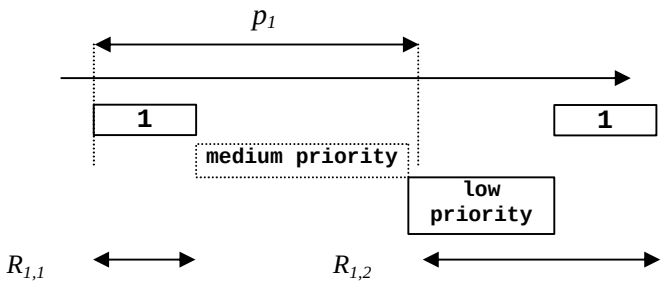

Figure 2.3: Queuing delay due to blocking

Figure 2.3 illustrates message blocking. The situation arises because of a low priority message become eligible for transmission just before release of a high priority message. Since the transmission is non pre-emptive, the entire message is transmitted and delays the message delivery time $R_{1,2}$ (second invocation of message 1 transmission). We do however observe that the high priority message can only be blocked by at most one low priority message since during the next bus arbitration; the bus will be allocated to the high priority message, which will be immediately transmitted. Hence, we extend the queuing term with a *blocking term* defined as:

$$B_m = \max(T_k) \quad \{\forall k \; lp(m) \; \} \qquad \textit{Equation 2.2}$$

I.e. the blocking term B_m for message m is obtained by examining all lower priority messages and selects the one with greatest transmission time T. We now arrive at our final expression for message transmission time:

$$R_m = B_m + T_m + Q_m \qquad \textit{Equation 2.3}$$

Where:

- B_m is the blocking time for message m as a result of interference from lower priority messages and is defined by *Equation 2.2*.
- Q_m is the queuing time for message m as a result of higher priority messages transmitted and thus delayed the message m and defined by *Equation 2.1*.
- T_m is the transmission time for message m.

We will use this result in the next chapter.

3 TTCAN COMPUTATIONAL MODEL

In this chapter, we will extend the previous computational model. We do this to provide for the implementation of time-triggered communication. Since this is an extension, we still support the earlier model and we end up with a communication subsystem, which efficiently supports time triggered operation as well as event triggered operation. We first describe the features known as time-triggered CAN. We then define different real time requirements designed as to support the semantics and requirements for a wide range of real time applications. Finally, in this chapter, we describe our scheduling method in detail.

3.1 Time Triggered CAN

Besides the features of CAN, which is a Collision Detect Collision Resolution (CD/CR) protocol, TTCAN also supports static scheduling in predefined time slots. Another enhancement with TTCAN from the CAN protocol is support for handling global time. The protocol comprises special slots devoted solely to synchronization of every TTCAN-node in the network. Time is measured with a granularity of Network Time Units (NTU) that is determined from the configuration of the actual system.

A time triggered communication schedule is always determined a priori, off line. During run-time, all nodes within the network have complete information about the schedule. Messages are periodic and there exists a *least common period* where all messages has been transmitted due to their respective periodicity. This reflects in the TTCAN *system matrix*, divided in *basic cycles*. The number of basic cycles is configuration dependent and the common period of all messages shall always correspond to the system matrix. Each basic cycle starts with a *reference message* that is used for synchronization purposes. Apart from the reference message, there are three different types of time windows in the system matrix:

- *Exclusive*, this window type is used for a statically scheduled message.
- *Arbitration*, this window type is used for ordinary CAN communication.
- *Free*, this window type is reserved for future use. This provides for simpler future integration of nodes where these windows might come to use.

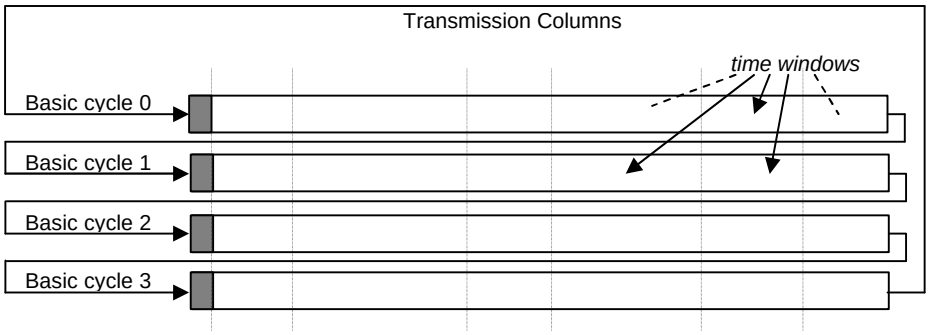

Figure 3.1 Example: TTCAN system matrix with four basic cycles

There are two possible levels of operation. Level 1 provides time triggered operation while level 2 additionally provides increased synchronization quality, global time and external clock synchronization. In both levels all timing features are based on a local time base, the local time. Ideally, all nodes in the network shall measure their local times using the same time unit (NTU), in reality a particular nodes view of NTU might differ marginally from the common view due to slight differences in local oscillators.

3.2 Overview of real time communication requirements

In this paragraph we will give some presumptions intended to clarify the terminology we are about to use. Although quite formally expressed, they should not be confused with definitions, since they lack the completeness of real definitions.

- A *task* is a sequence of ordered machine instructions carried out by a microprocessor. Each instruction takes a finite amount of time to be executed. A task may interact with other tasks by means of *message passing* or with the surrounding environment by means of input-/output- devices.
- A *message* carries information between tasks, the message initiator (sender) and message receiver(s). The message passing mechanism is responsible for delivering the message from its initiator to all intended receivers.
- A message is said to be *periodic* if it is initiated with regular equidistant time intervals.
- A message is said to be *sporadic* if it is initiated at least once, and if repeated, generally with no regular equidistant time intervals
- All messages passes through an indivisible resource, called the *bus*.
- When two or more messages contents for bus access the *arbitration* mechanism decides the order in which the messages will gain access to the bus. The order is determined by pre assigned message priorities.
- We assume that there is a global time base denoted *NTU* (Network Time Unit) which is consistent through all nodes in the network.

3.3 The Scheduling activities

The scheduling process is divided in three major phases:

- *Structuring*, the entire set of messages is analyzed. Messages are assigned to a category depending on application-derived requirements. Within two of the categories, a unique message priority is assigned to each message in the category.
- *Simulation*, a finite amount of communication cycles is simulated, and data from the simulation is assembled. This means that message response times are either calculated or estimated depending on message category.
- *Analysis*, the resulting schedule is analyzed by merging data from the simulations with criteria for a final schedule. This phase will flag for messages that are required to, but not guaranteed to meet their deadline. It will also point out potentially discriminated messages, i.e. messages that are not required to meet a certain deadline but are unlikely to be scheduled at all during run-time.

We will now turn to a more thoroughly description of these activities

Structuring

The task set is analyzed with respect to all messages passing. Depending on task criticality as well as timing requirements, the messages fall into exactly one of the following categories:

> *Hard real time* – This category is devoted for messages that exchange data between tasks with highest priorities. Although this mostly applies to periodic messages this is not always the case. A sporadic message might still fall into this category presume that it is of vital importance for the system. Each message will be assigned a pre-scheduled exclusive time window and each instance (repetition) of the message is guaranteed to get sufficient bandwidth for its requirements. Anticipated jitter is neglect able.

Firm real time – This category is for messages, which still have to be guaranteed a bounded response time but where a certain amount of jitter is tolerable. These messages are assigned high CAN priority and they are scheduled in arbitration time windows. All messages should be guaranteed to meet their deadline.

Soft real time – Messages that falls into this category have no strict timing requirements. These messages are assigned low CAN priority and they are scheduled in arbitration time windows. In most applications, where bus utilization is high, response times for these messages can only be calculated probabilistic.

The structuring activity aims at assigning each application message according to criteria above to exactly one group.

We consider a message set ***M*** where each member has the following properties:

M^{i} , message identifier *i,* which is unique in the set

M^{c}, message class, which is exactly one of *h* (hard real time), *f* (firm real time) or *s* (soft real time)

M_{p}, message period time, expressed as an integer in the global time base.

M_{e}, message delivery time, length of message expressed as an integer in the global time base.

A message is said to be *defined* when all these properties has been assigned to it.

Besides from establishing message properties it is necessary to provide means for defining relations between messages. In general the *predecessor* and *successor* relational operators shall be valid within the system matrix. As an example (See Figure 3.2 below), consider an application where a physical sensor is attached to node 1, the sensor is sampled, the value is checked in this node and then passed to two other nodes in the network (2 and 3) in message *S*. Calculation based on the sensor value is now performed in nodes 2 and 3, results are transmitted, messages *C1* and *C2*, to node 4, which establishes the final value and then transmits it to the actuator node 5, (message *A*).

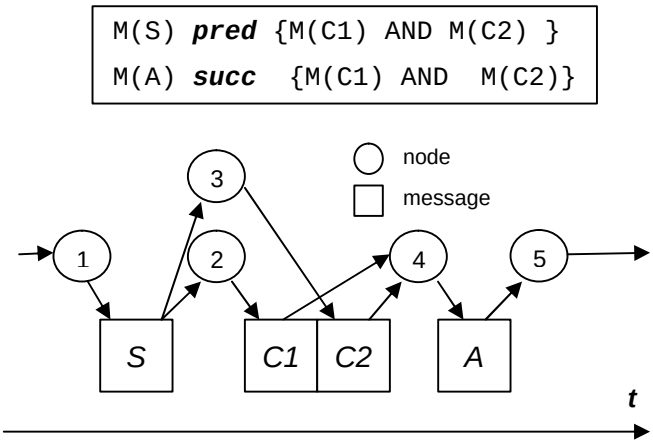

Figure 3.2: Messages ordered by relational operators

A complete analysis of the example is of course far more complex than this but we can clearly see the need for expressing interrelations between messages rather than only focus on message properties.

3.4 Allocation to M^f

From structuring we have:

$$M : \{M^h, M^f, M^s\}$$

Assume that M^h is defined.

We will now construct the Matrix Cycle for a schedule of M^h.

A feasible schedule must fulfil:

$$LCM(M^h_p) = x\ 2^n \qquad \textit{(Equation 3.1)}$$

where:

LCM denotes the Least Common Multiple period for all messages in the set.
x denotes the basic cycle length in a TTCAN schedule matrix cycle.
n denotes the TTCAN schedule basic cycle.

A feasible solution must also adhere to hardware constraints. We can not expect that a hardware implementation supports arbitrary x and n, so we stipulate that:

Equation must be solved with the following *hardware constraints*:

`Hwc1`: $1 = x = 2^y$
`Hwc2`: $0 = n = k$

`Hwc1`: The basic cycle length must never exceed 2^y NTU's. This is from the fact that any basic cycle length should be held in a hardware register, the register is limited to y bits.

`Hwc2`: The number of basic cycles must be a power of 2; this is derived from the TTCAN specification, and must never exceed 2^k, where k is a basic cycle counter held in a hardware register limited to k bits.

The maximum number of columns in a matrix cycle is determined by hardware in terms of *Triggers*. One *Trigger* is allocated to each time window. Hardware is supposed to provide a limited number of *Triggers* for each basic cycle, thus we introduce a third hardware constraint:

`Hwc3`: $Tr = m$

where m is the maximum number of possible *Triggers* during a basic cycle in the schedule.

As an example, the following hardware constraints apply to the *BOSCH TTCAN_EC* module (see reference [15]).

`Hwc1`: $1 = x = 2^{16}$
`Hwc2`: $0 = n = 6$
`Hwc3`: $Tr = 32$

There might still be several solutions to *Equation 3.1* that fulfils hardware constraints and in theory we might arbitrarily choose one. We now establish two different strategies for determination of the matrix cycle:

Strategy 1:

Minimize the number of basic cycles at the cost of an increased basic cycle length. This will require more columns in the system matrix and thus maximizing the required number of triggers. This can be suitable for small message sets and will generally produce a comprehensible schedule.

Strategy 2:

Minimize the basic cycle length at the cost of more basic cycles. This will require fewer columns, since this also distributes the use of *Triggers* through several basic cycles. Thus, improve possibility on a feasible schedule for large message sets.

EXAMPLE 3.1 *Strategy 1*

Assume we want to construct a schedule for *TTCAN_EC* and the message set $\boldsymbol{M}^h = (M^1, M^2, M^3)$ with the following properties, all expressed in Network Time Units (NTU):

$M^1_p = 1000$, $M^1_e = 168$
$M^2_p = 2000$, $M^2_e = 184$
$M^3_p = 3000$, $M^3_e = 216$

I.e. the set consists of three messages with respective period times 1000, 2000 and 3000 and with delivery times 168, 184 and 216 NTU's.

It's trivial to find:

$$\text{LCM}(M^1, M^2, M^3) = 6000.$$

so we have from *Equation 3.1*:

$$6000 = x\,2^n$$

Minimising the number of basic cycles means $n = 0$, so $x = 6000$. We establish that both `Hwc1` and `Hwc2` are fulfilled.

Total number of triggers for a set of N messages in the case of a single basic cycle can be easily calculated as:

$$\#\text{ of triggers} = \sum_{i=1}^{N} \frac{LCM(\boldsymbol{M})}{M_i}$$

in this case:

$$\#\text{ of triggers} = \frac{60}{10} + \frac{60}{20} + \frac{60}{30} = 11$$

Thus, choosing *Strategy 1* leads to a solution where we have:

1 basic cycle with 11 *Triggers*
Matrix cycle length is 6000 NTU's.

Figure 3.3 illustrates a feasible schedule using a single basic cycle. Note: time axis is *not* proportional, unused transmit columns are greyed.

Basic Cycle Triggers																	
0	168	352		1000		2000	2168		3000		3352		4000	4168		5000	
M_1	M_2	M_3		M_1		M_1	M_2		M_1		M_3		M_1	M_2		M_1	

Figure 3.3: Example 3.1 feasible schedule using a single basic cycle

End of EXAMPLE 3.1.

EXAMPLE 3.2 *Strategy 2*

Now, assuming the same preconditions as in example 3.1, applying *Strategy 2* leads us to the following problem:

Solve equation:

$$6000 = x\,2^n$$

when n is maximised (= 6) and x is an integer value less than 2^{16}.

The solution is straightforward[2]:

$n = 0$: $6000 = x\ 2^0$ ⇒ x = 6000
$n = 1$: $3000 = x\ 2^1$ ⇒ x = 3000
$n = 2$: $1500 = x\ 2^2$ Þ x = 1500
$n = 3$: $750 = x\ 2^3$ ⇒ x = 93.75
$n = 4$: $93.75 = x\ 2^4$ ⇒ x = 5.859...
$n = 5$: $5.859 = x\ 2^5$ ⇒ x = 0.1831...
…

We find that this choice yields a schedule with cycle length of 1500 NTU's using $2^2 = 4$ basic cycles.

Basic cycle	1 2 3 4	0 - 3000 -	168 - - -	352 - 3352 -	- 2000 - 5000	- 2168 - -	- - 4000 -	1000 - 4168 -		*Trigger* Information / Minimum *Triggers*
	1	M_1	M_2	M_3				M_1		4
	2				M_1	M_2				2
	3	M_1		M_3			M_1	M_2		4
	4				M_1					1

Figure 3.4: Example 3.2 feasible schedule using a minimal basic cycle length

End of EXAMPLE 3.2

Now we have shown a simple but yet efficient method for constructing a feasible matrix cycle using message properties *class*, *period* and *delivery time*. We have seen that the matrix can be constructed in several ways as long as *Equation 3.1* is satisfied. Thus, the schedule for a set of given messages, can be implemented in TTCAN in different ways still obeying the standard. The real time properties are of course consistent across different choices, however, due to hardware constraints as well as properties of remaining message sets ($M^{\mathbf{f}}$ and $M^{\mathbf{s}}$) there are still some questions to be answered relating to the actual allocation in the matrix cycle.

The actual allocation of NTU-slots to messages within the matrix cycle is arbitrary from $M^{\mathbf{h}}$ message set point of view. The TTCAN protocol brings timeliness with very small jitter. There are however, other considerations to take into account. The message set $M^{\mathbf{f}}$ comprises ordinary CAN messages exhibiting requirements on deadlines and *desires* of minimal jitter. Our choice of NTU-allocation to $M^{\mathbf{h}}$ messages will have impact on this. We can for example stuff all $M^{\mathbf{h}}$ messages in the beginning of the first cycle as we did in our previous examples; we call this *dense* allocation. The case for this choice could be a distributed control application where sensor values are gathered from different nodes and broadcasted on the bus. These values should be made public as soon as possible to minimize control delay and dense allocation of such messages should therefore be preferable.

On the other hand, this will create long intervals where all messages that use *Arbitration slots* are blocked. Since we might anticipate messages from the $M^{\mathbf{f}}$ class with deadline requirements this packing might eventually lead to violation of such requirements as illustrated by Figure 3.5 below.

[2] It is sufficient to stop iteration after determination of the case $n=3$ since: `For( A,k,n integer numbers = 0); if A` 2^{-n} `is fractional then for any integer k such that k > n ; A` 2^{-k} `is fractional`. Proof is left as a mathematical underpinning.

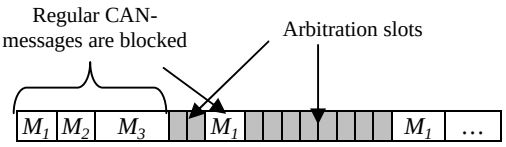

Figure 3.5: Dense allocation leads to large blocking factors

Another allocation strategy would be to spread out all time triggered messages as much as possible, we call this *sparse* allocation. This would result in shorter blocking periods, see Figure 3.6.

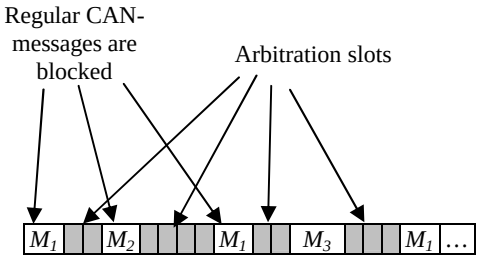

Figure 3.6: Sparse allocation of time-triggered slots

Finally, applications may impose *in-time* requirements. Again consider Figure 3.2, this example implies that there would be a demand for a distinct time offset between the message passing to guarantee that nodes have completed computations and actually delivered their messages to the controller. We call this *demanded* allocation. The current scheduler implementation implements demanded allocation by means of *release time* directives.

The choice of a good allocation strategy may be crucial for some message sets while it is of less importance in others thus it is indeed dependent on the application. Regardless of our choice of allocation strategy we need a third step, where we analyze our solution in order to find out whether or not all deadlines in the M^f set are met.

3.5 Analysis of message set M^f

TTCAN allocation activities will leave a number of arbitration slots. Interleaving of these slots depends on how the allocation was made. These allocation slots are all available for firm and soft requirements CAN messages. Since all messages in the firm class have been assigned higher priority, we concentrate our analysis on this class. The only exception is that we must include soft messages when we determine the blocking factor.

We can reuse results showed in the previous chapter with some minor modifications. Consider an arbitrary matrix cycle, after allocation of exclusive time windows we are left with a list of arbitration slots ($q_0..q_{N-1}$) where N is the number of arbitration slots, see Figure 3.7.

Basic Cycle	Grey slots are supposed to be allocated for M^h								
	NTU-slots (Columns)								
1		q_0							
2		q_1					q_2		
3		q_3		q_4			q_5		
…..		…			…		…		…
2^n		q_{N-3}					q_{N-2}		q_{N-1}

Figure 3.7: Illustration of Arbitration slots

We now introduce each exclusive time window as a *virtual message in the firm set*. This virtual message is assigned a higher priority than the highest priority message in the ordinary firm set. All virtual firm messages will have a defined start time denoting start of an exclusive window as well as completion time indicating end of the exclusive time window. Calculation of the queuing term is now done according to the following algorithm:

Algorithm 3.1 Modified queuing calculation

Exclusive time slots appears as messages with priority 0, release times that may be none zero and they are released exactly once during the analyzed time window (LCM). Ordinary messages in the firm set are arranged according to descending priority with priority 1 being the highest priority within the ordinary firm message set.

```
for each message m:
      for message j within hp(m)
            Q_Assigned = FALSE
            for all virtual messages i
                  if( Q_m + T_m ) falls within ( M_i,start , M_i,completion )
                        Q_m = M_i,completion
                        Q_Assigned = TRUE
                  endif
            endif
            if not Q_Assigned
                  Q_m = Σ_{∀j:hp(m)}^{1} ⌈ Q_m / p_j ⌉ T_j
            end
      end
end
```

End Algorithm 3.1

3.6 Analysis of message set $M^{\$}$

The messages assigned to this class are by definition less critical in the application. Analysis may therefore be relaxed. The current scheduler implementation simply reports the minimum time available for the class.

3.7 Summary

In this chapter, we have introduced an extended computational model for message scheduling and analysis in real time systems using TTCAN/CAN hybrid communication systems.

Our method requires knowledge of a complete message set intended for a common TTCAN/CAN communication bus

We applied our model and used the proposed scheduling method on a non-trivial set of messages designed to be representative for a real application. This experiment will be detailed in the next chapter.

4 A SAMPLE APPLICATION

In this chapter, we will demonstrate our scheduling method by applying it to a non-trivial set of messages. We have designed a prototype scheduling-tool *TTCST* (*TTCAN Scheduling Tool*) where we implemented most vital parts of our ideas. This tool was used for design of schedule for the case we use in this chapter, reference [16].

4.1 The application

The FAR-car described in reference [17] is an experimental mechatronical platform developed for evaluation of electronic hardware and software for distributed drive by wire concepts. The car has multiple steer mode functionality:

1. Standard 2-wheel steering, rear wheels are locked in straight forward position.
2. Four wheel parallel steering; Front and Rear wheel pairs are parallel.
3. Four wheel full steering, Front and Rear wheel pairs are synchronized to minimize steering radius.

The car also has throttle and brake functions and limited fault tolerance is implemented.

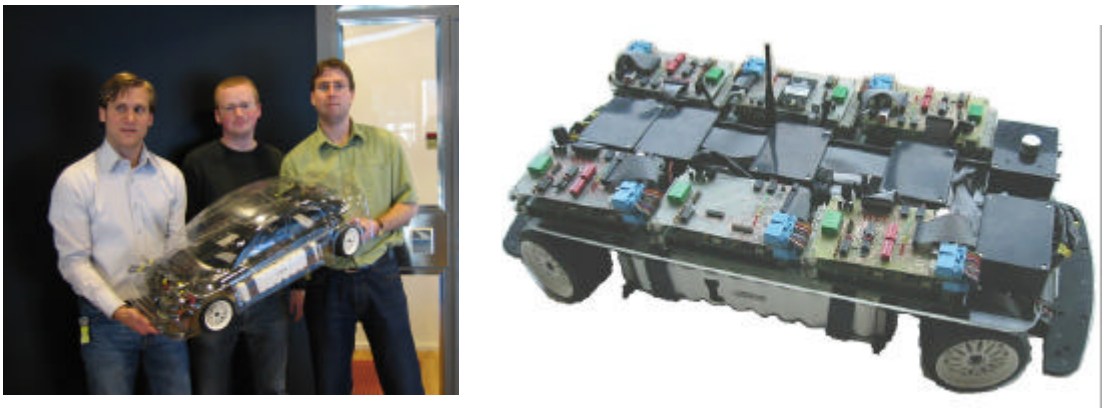

Figure 4.1: FAR-car and some of the developers (left), electronics hardware platform (right)

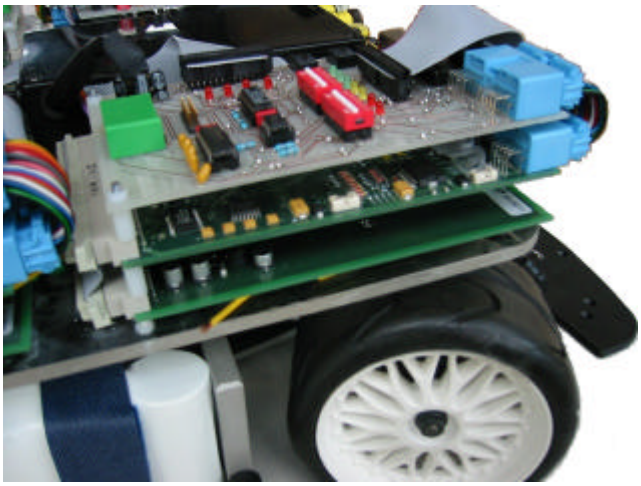

Figure 4.2: Implementation of Front Right node (3 PCB's)

4.2 Control system software

Control software is executed according to a static schedule using a very simple supervisory kernel (dispatcher). The entire schedule uses 32 ms cycle time (LCM) and the different tasks are executed according to Table 4.1.

Task No	Release time (ms)	Worst Case Execution Time (ms)	Description
1	0	1	*Synchronize*; synchronizations of TTCAN global time with kernel time.
2	1	3	*Fault detection internal*; check for consistent values in local node.
3	4	3	*Fault detection external;* check for consistency of values in the node that is monitored.
4	7	2	*Reserved 1;* planned functionality is not yet implemented.
5	9	3	*Fault detection global;* assembles error signals and communication faults.
6	12	3	*Local control external;* perform local control loop for monitored node.
7	16	4	*Global control;* main control.
8	21	5	*Read data;* sensor data acquisition.
9	23	2	*Voting;* local self checks.
10	26	3	*Reserved 2;* planned functionality is not yet implemented.
11	28	4	*Local control internal;* perform local control loop.

Table 4.1: Overview of FAR control system software

The message set

There are totally 30 unique messages associated with application data. Additional triggers are used for TTCAN reference message transmission. In the following Table 4.2 application messages has been assigned symbolic names indicating the node that transmits the message. The following symbolic names are used:

- FLx - Front Left node
- FRx – Front Right node
- RLx – Rear Left node
- RRx – Rear Right node
- HMIComx – Interface node to radio control

The first transmit column is always used for a TTCAN synchronization message (time 0), transmitted by the current time master. In the Table 4.1 this column also indicates the activities undertaken in the nodes. The matrix cycle consists of 32 basic cycle of 1 ms length. Each message is transmitted once during the matrix cycle, i.e. all messages have period time 32 ms. Digits are simply used to enumerate the messages.

Time (ms)	**0/Activity**	**250 us**	**500 us**	**750 us**
0	TTCAN Sync Message RTOS error check			
1	Fault detection internal	FL2	FR3	RL4
2		RR5	Env6	
3		HMICom7		
4	Fault detection external			
5		FL8	FR9	RL10
6		RR11		

7			FL12	FR13
8		RL14	RR15	
9	Fault detection node radio			
10				
11				
12	Local control external			
13				
14		FL16	FR17	RL18
15		RR19		
16	Global control			
17				
18				
19		FL20	FR21	RL22
20		RR23		
21	Read data from sensors		FL24	FR25
22		RL26	RR27	
23	Voting			
24				
25				
26			FL28	FR29
27		RL30	RR31	
28	Local control internal			
29				
30				
31				

Table 4.2: Process and message scheduling

From the above we conclude that the application communication scheduling is fully determined pre run-time. The prerequisites are all handled by the scheduling tool by means of definitions and directives in the input file. The following syntactic elements are used:

A *message* is introduced to *TTCST* as directive with the following syntactic form:

```
message (MSG_ID, class , period , size )
```

Message is a reserved word and:

`MSG_ID` is a unique symbolic name
`class` denotes one of h (hard), f (firm) or s (soft)
`period` is message period time, a floating point constant, in seconds
`size` is the number of bytes in the message

Ordering of messages; Messages can be ordered by using a *precedence relation* statement:

```
MSG_ID1 prec { MSG_ID2 [and MSG_ID3 ...[ and MSG_IDn ]..] }
```

Prec is a reserved word which states that MSG_ID1 always should precede the messages that are named within the “curly brackets”. The list should consist of at least one message, additional messages should be preceded with the reserved word **and**.

The prerequisites impose that we use demanded allocation of TTCAN message slots. This is accomplished by specifying release times for all messages.

Release time statement:

```
MSG_ID release ( release_time )
```

Release is a reserved word which states that message MSG_ID can not be scheduled for transmission before time *release_time* (a floating point constant) in the matrix cycle.

A complete listing of the input file used for this case can be found in Appendix A.

4.3 Class h messages schedule

Scheduler output is extensive, a complete listing can be found in Appendix B. The TTCAN messages schedule (messages in class *h*) are listed in two forms, the second form ”schedule by release” look like this:

```
-- SCHEDULE BY RELEASE------------
-- Start  .. Stop (NTU)   Start    .. Stop (us) Basic Cycle Invocation Message
   000000 .. 000048  --   00000000 .. 00000060   -- 0       -- 0    -- 'SYNC'
   000800 .. 000848  --   00001000 .. 00001060   -- 1       -- 1    -- 'SYNC'
   001000 .. 001112  --   00001250 .. 00001390   -- 1       -- 0    -- 'FL2'
   001200 .. 001312  --   00001500 .. 00001640   -- 1       -- 0    -- 'FR3'
   001400 .. 001512  --   00001750 .. 00001890   -- 1       -- 0    -- 'RL4'
   001600 .. 001648  --   00002000 .. 00002060   -- 2       -- 2    -- 'SYNC'
   001800 .. 001912  --   00002250 .. 00002390   -- 2       -- 0    -- 'RR5'
   002000 .. 002112  --   00002500 .. 00002640   -- 2       -- 0    -- 'Env6'
   002400 .. 002448  --   00003000 .. 00003060   -- 3       -- 3    -- 'SYNC'
        .....
```

Each invocation of a message is written on one line. Start and stop times are given in both NTU’s and real time (us). It is easy to verify congruence with the specification in Table 4.2.

4.4 Class f messages schedule

Although there are only time triggered messages used in the present FAR car we might expect the need for event triggered messages in a future extension. In this case study we therefore added a hypothetic message set including both firm and soft real time requirements. We only analyze the class *f* messages for schedulability. If the analysis shows that one or more messages in this class not can be guaranteed to complete, an error message will be displayed. A simulation, written to a file (see Appendix C) shows the entire schedule of class *h* and class *f* messages during a LCM. The scheduler output file contains only the first invocation of each class *f* message (see Appendix B).

Finally a schedule summary is written to the output file:

```
-- UTILISATION ------------
-- Class h messages: 3408 NTU (13%)
-- Class f messages: 11424 NTU (44%)
-- Class s messages: 8624 NTU (33%)
-- END OF SCHEDULE---
```

The summary displays percentage usage within the three message classes. Note that total utilization can be over 100% as long as the class *h* and *f* messages are schedulable. In such cases, this implies that the soft message set cannot be guaranteed to meet deadlines. The scheduler will not flag for this.

5 CONCLUSIONS

In this paper, we have demonstrated a method for handling communication scheduling in distributed control system where real-time requirements range from *hard* to *soft*. We implemented our method in a prototype tool called *TTCST* (*TTCan Scheduler Tool*). We used the tool in a case study adapted from ongoing work within a steer-by wire application. The tool, a short user description and example files is available as download from [16] or by email requisition from the author.

5.1 Discussion of work

Although straight forward in theory it turned out that that implementation of the scheduling tool became time consuming and sometimes cumbersome. Therefore the tool has not been exhaustively tested or validated by any mean.

Our choice of case for test and demonstration for the scheduling tool was based on two primary factors.

1. Although realistic input "real-world" data could have been used this would have introduced irrelevant complexity and made the case more difficult to comprehend.
2. The FAR car system has been implemented and a TTCAN communication subsystem has been programmed and tested.

Thus this work has gained a lot from experiences from the FAR car project.

5.2 Discussion of results

We do believe that our method of dividing an application message set into classes and then treat these classes according to criticality is a realistic and suitable approach for engineering purposes. By introduction of simple directives to introduce message properties, control e.g. precedence's and release times the input language of the scheduler tool further approaches the engineers' needs.

There is certainly a need for new hybrid scheduling methods today. Contemporary protocols, such as TTCAN and FlexRay supports both time triggered and event triggered message communication mechanisms. It is likely that future protocol specifications for real-time communication will follow the same path.

The method itself, although now implemented and demonstrated with TTCAN does not rely explicitly on the TTCAN protocol. Modifications in the static scheduling phase (class *h*) messages can be undertaken for adoption to another hybrid protocol.

5.3 Future work

Development of test and validation suites; as mentioned above, the scheduling tool has not been extensively tested. A validation suite is needed.

A code generator for TTCAN controllers; programming of communication controllers can be both difficult and tedious, automatic code generation for initialization of controller, message transmission and reception is highly desirable.

Interface to message databases; it would be very challenging if the tool could be automatically fed with existing applications (e.g. SAE J1939) for further evaluation.

Extension to meet further needs; a more semantic powerful input language designed to meet design engineers demands and needs.

Variants for other communication protocols; this would make the tool more general and provide means for efficient comparisons between different protocol specifications and their implementations.

Acknowledgement

The author wishes to thank Prof. Jan Torin for valuable comments and appreciated support. This work has been conducted within The Centre of Excellence CHARMEC (CHAlmers Railway MEChanics).

REFERENCES

[1] International Organization for Standardization. *ISO 11898-1: Road vehicles – Controller area network (CAN) Part1: Data link layer and physical signalling*, 2003.

[2] Society of Automotive Engineers, SAE, *Surface Vehicle Recommended Practice J1939-7x*. 1996.

[3] Rajnak, A., Ramnefors, M., *The Volcano Communication Concept*, SAE 2003-01-0114, SAE Technical Paper Series, 2002.

[4] Rushby, J. *A Comparison of Bus Architectures for Safety-Critical Embedded Systems*, CSL Technical Report, SRI International, September 2001.

[5] Tindell, K., Hansson.H and Welling A., *Analysing Real-Time Communications: Controller Area Network CAN)*, In F. Jahanian and K. Ramaritham, editors, Proc. 15 th RealTime Systems Symposium, pages 259--263. IEEE Computer Society Press, 1994.

[6] Tindell, K. , Burns, A. and Wellings, A.J. , *Calculating Controller Area Network (CAN) Message Response Time*, Control Engineering Practice, Vol. 3(8), pp. 1163-1169 (1995).

[7] International Organization for Standardization. *ISO 11898-4: Road vehicles – Controller area network (CAN) Part4: Time- triggered communication*, 2004.

[8] Stankovic, J.A. , Spuri, M. , Di Natale M. and Butazzo, G. , *Implications of Classical Scheduling Results for Real-Time Systems*, IEEE Computer, 28(6), June 1995.

[9] Liu, C. and Layland, J. , *Scheduling Algorithm for Multiprogramming in a Hard Real-time Environment*, Journal of the ACM, 20(1), pp. 41 – 61, January 1973.

[10] Sha, L. ,Rajkumar, R. ,Sathaye, S. , *Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems*. Proceedings of the IEEE, Vol 82, No 1, January 1994, pp 68-82.

[11] Nolte, T. , Hansson, H. , Norström, C. , *Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network* , In Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, May 27 - 30, Toronto, Canada, 2003, pp 200-207.

[12] Broster, I. Burns, A. Rodriguez-Navas, G. , *Probalistic Analysis of CAN with Faults*, In Proceedings of the 23rd Real-time Systems Symposium, Austin, Texas, December 3-5, 2002, pp 269-278.

[13] Hartwich, F. , Müller, B. , Führer, T. , Hugel, R. , *CAN Network with Time Triggered Communication*, In 7th iCC Proceedeings, CAN in Automation (CiA), Holland, 2000.

[14] Fonseca, J. , Coutinho, F. , Barreiros, J., *Scheduling for a TTCAN network with a stochastic optimization algorithm*, In 8th iCC Proceedeings, CAN in Automation (CiA), Las Vegas, USA, February, 2002.

[15] *TTCAN IP Module User's Manual Revision 1.6*, At: `http://www.can.bosch.com`

[16] At: `http://www.chl.chalmers.se/~roger/research/ttcst/tool.zip`

[17] Cornelsen, J. and Dahlqvist, P. *Development of a Safety Critical Mechatronical System used as Demonstrator for the GAST Project*, Master´s Degree Project MMK 2004:76 MDA 233, Department of Machine Design, Royal Institute of Technology, Stockholm, 2004.

Appendix A: Input file for case study

```
//
// far-sched.txt
// scheduler input example
// options:
// -pbc=1000 i.e basic cycle is 1000 us
// -ntu=1250 i.e. 1250 ns
// -cbt=1250 i.e. 1250 ns bit time

// Synchronisation message
message( SYNC , h , 0.001e0 , 0 )
// Declarations of messages with
// HARD real time requirements
// All period times 32 ms...

message( FL2 , h , 0.032e0 , 8 )
message( FR3 , h , 0.032e0 , 8 )
message( RL4 , h , 0.032e0 , 8 )
message( RR5 , h , 0.032e0 , 8 )
message( Env6 , h , 0.032e0 , 8 )
message( HMICom7 , h , 0.032e0 , 8 )
message( FL8 , h , 0.032e0 , 8 )
message( FR9 , h , 0.032e0 , 8 )
message( RL10 , h , 0.032e0 , 8 )
message( RR11 , h , 0.032e0 , 8 )
message( FL12 , h , 0.032e0 , 8 )
message( FR13 , h , 0.032e0 , 8 )
message( RL14 , h , 0.032e0 , 8 )
message( RR15 , h , 0.032e0 , 8 )
message( FL16 , h , 0.032e0 , 8 )
message( FR17 , h , 0.032e0 , 8 )
message( RL18 , h , 0.032e0 , 8 )
message( RR19 , h , 0.032e0 , 8 )
message( FL20 , h , 0.032e0 , 8 )
message( FR21 , h , 0.032e0 , 8 )
message( RL22 , h , 0.032e0 , 8 )
message( RR23 , h , 0.032e0 , 8 )
message( FL24 , h , 0.032e0 , 8 )
message( FR25 , h , 0.032e0 , 8 )
message( RL26 , h , 0.032e0 , 8 )
message( RR27 , h , 0.032e0 , 8 )
message( FL28 , h , 0.032e0 , 8 )
message( FR29 , h , 0.032e0 , 8 )
message( RL30 , h , 0.032e0 , 8 )
message( RR31 , h , 0.032e0 , 8 )


// Declarations of messages with
// FIRM real time requirements
message( f1 , f , 0.0025e0 , 8 )
message( f2 , f , 0.0025e0 , 8 )
message( f3 , f , 0.0025e0 , 8 )
message( f4 , f , 0.0025e0 , 8 )
message( f5 , f , 0.005e0 , 8 )
message( f6 , f , 0.005e0 , 8 )
message( f7 , f , 0.005e0 , 8 )
message( f8 , f , 0.005e0 , 8 )
message( f9 , f , 0.005e0 , 8 )
message( f10 , f , 0.005e0 , 8 )
message( f11 , f , 0.010e0 , 8 )
message( f12 , f , 0.010e0 , 8 )
message( f13 , f , 0.010e0 , 8 )
message( f14 , f , 0.010e0 , 8 )
message( f15 , f , 0.010e0 , 8 )


// Declarations of messages with
// SOFT real time requirements
message( s1 , s , 0.010e0 , 8 )
message( s2 , s , 0.010e0 , 8 )
message( s3 , s , 0.010e0 , 8 )
message( s4 , s , 0.010e0 , 8 )
message( s5 , s , 0.010e0 , 8 )
message( s6 , s , 0.010e0 , 8 )
message( s7 , s , 0.010e0 , 8 )
message( s8 , s , 0.010e0 , 8 )
message( s9 , s , 0.010e0 , 8 )
message( s10 , s , 0.010e0 , 8 )
message( s11 , s , 0.010e0 , 8 )
message( s12 , s , 0.010e0 , 8 )
message( s13 , s , 0.010e0 , 8 )
message( s14 , s , 0.010e0 , 8 )
message( s15 , s , 0.010e0 , 8 )
message( s16 , s , 0.010e0 , 8 )
message( s17 , s , 0.010e0 , 8 )
message( s18 , s , 0.010e0 , 8 )
message( s19 , s , 0.010e0 , 8 )
message( s20 , s , 0.010e0 , 8 )

// Optional section: Precedence's
SYNC pred { FL2 }
FL2 pred{ FR3}
FR3 pred{ RL4}
RL4 pred{ RR5}
RR5     pred{ Env6 }
Env6    pred{ HMICom7 }
HMICom7 pred{ FL8 }
FL8     pred{ FR9 }
FR9     pred{ RL10 }
RL10    pred{ RR11 }
RR11    pred{ FL12 }
FL12    pred{ FR13 }
FR13    pred{ RL14 }
RL14    pred{ RR15 }
RR15    pred{ FL16 }
FL16    pred{ FR17 }
FR17    pred{ RL18 }
RL18    pred{ RR19 }
RR19    pred{ FL20 }
FL20    pred{ FR21 }
FR21    pred{ RL22 }
RL22    pred{ RR23 }
RR23    pred{ FL24 }
FL24    pred{ FR25 }
FR25    pred{ RL26 }
RL26    pred{ RR27 }
RR27    pred{ FL28 }
FL28    pred{ FR29 }
FR29    pred{ RL30 }

// Optional section: Release times
FL2     release (0.001250e0)
FR3     release (0.001500e0)
RL4     release (0.001750e0)
RR5     release (0.002250e0)
Env6    release (0.002500e0)
HMICom7 release (0.003250e0)
FL8     release (0.005250e0)
FR9     release (0.005500e0)
RL10    release (0.005750e0)
RR11    release (0.006250e0)
FL12    release (0.007500e0)
FR13    release (0.007750e0)
RL14    release (0.008250e0)
RR15    release (0.008500e0)
FL16    release (0.014250e0)
FR17    release (0.014500e0)
RL18    release (0.014750e0)
RR19    release (0.015250e0)
FL20    release (0.019250e0)
FR21    release (0.019500e0)
RL22    release (0.019750e0)
RR23    release (0.020250e0)
FL24    release (0.021500e0)
FR25    release (0.021750e0)
RL26    release (0.022250e0)
RR27    release (0.022500e0)
FL28    release (0.026500e0)
FR29    release (0.026750e0)
RL30    release (0.027250e0)
RR31    release (0.027500e0)


// end of input
```

Appendix B: Scheduler output

The scheduler produces vast output; less important text is omitted here.

```
-- TTCAN Scheduler (TTCST)  Version 0.8.0
-- Compilation started: 08/18/04 17:31:04
-- Input file: 'far-sched.txt'
----------------------------
-- Option settings:
-- NTU: 1250
-- Can Bit Time: 1250
-- Basic cycle default overridden
-- END of translation, No Errors
-- Created output file 'ttcst.out'
----------------------------
-- TIMING PROPERTIES:
-- NTU (Network Time Unit) = 1250 (ns)
-- Can Bit Time            = 1250 (ns)
----------------------------
-- SCHEDULE SUMMARY
-- Message Set Hard LCM    = 25600 NTU ( 32 ms )
-- Matrix Cycle using 32 basic cycle with length = 800 NTU ( 1000 us )
-- Minimum number of triggers required during LCM = 62
```

Note: This figure accounts even for Synchronisation messages, so it is not really true. In fact, the number of triggers is 30 since each basic cycle starts with a synch-message.

```
-- Total utilisation = 97 % (Required = 100 %)
-- (Total utilisation lower than requirement)
----------------------------
-- MESSAGE SET OVERVIEW:
```

A list of all defined messages is omitted here.

```
-- SCHEDULE BY RELEASE------------
```

A list of time triggered messages, start and stop times for the entire LCM.

```
-- Start  .. Stop (NTU)   Start    .. Stop (us) Basic Cycle Invocation Message
   000000 .. 000048  --   00000000 .. 00000060   -- 0       -- 0    -- 'SYNC'
   000800 .. 000848  --   00001000 .. 00001060   -- 1       -- 1    -- 'SYNC'
   001000 .. 001112  --   00001250 .. 00001390   -- 1       -- 0    -- 'FL2'
   001200 .. 001312  --   00001500 .. 00001640   -- 1       -- 0    -- 'FR3'
   001400 .. 001512  --   00001750 .. 00001890   -- 1       -- 0    -- 'RL4'
   001600 .. 001648  --   00002000 .. 00002060   -- 2       -- 2    -- 'SYNC'
   001800 .. 001912  --   00002250 .. 00002390   -- 2       -- 0    -- 'RR5'
   002000 .. 002112  --   00002500 .. 00002640   -- 2       -- 0    -- 'Env6'
   002400 .. 002448  --   00003000 .. 00003060   -- 3       -- 3    -- 'SYNC'
   002600 .. 002712  --   00003250 .. 00003390   -- 3       -- 0    -- 'HMICom7'
   003200 .. 003248  --   00004000 .. 00004060   -- 4       -- 4    -- 'SYNC'
   004000 .. 004048  --   00005000 .. 00005060   -- 5       -- 5    -- 'SYNC'
   004200 .. 004312  --   00005250 .. 00005390   -- 5       -- 0    -- 'FL8'
   004400 .. 004512  --   00005500 .. 00005640   -- 5       -- 0    -- 'FR9'
   004600 .. 004712  --   00005750 .. 00005890   -- 5       -- 0    -- 'RL10'
   004800 .. 004848  --   00006000 .. 00006060   -- 6       -- 6    -- 'SYNC'
   005000 .. 005112  --   00006250 .. 00006390   -- 6       -- 0    -- 'RR11'
   005600 .. 005648  --   00007000 .. 00007060   -- 7       -- 7    -- 'SYNC'
   006000 .. 006112  --   00007500 .. 00007640   -- 7       -- 0    -- 'FL12'
   006200 .. 006312  --   00007750 .. 00007890   -- 7       -- 0    -- 'FR13'
   006400 .. 006448  --   00008000 .. 00008060   -- 8       -- 8    -- 'SYNC'
   006600 .. 006712  --   00008250 .. 00008390   -- 8       -- 0    -- 'RL14'
   006800 .. 006912  --   00008500 .. 00008640   -- 8       -- 0    -- 'RR15'
   007200 .. 007248  --   00009000 .. 00009060   -- 9       -- 9    -- 'SYNC'
   008000 .. 008048  --   00010000 .. 00010060   -- 10      -- 10   -- 'SYNC'
   008800 .. 008848  --   00011000 .. 00011060   -- 11      -- 11   -- 'SYNC'
   009600 .. 009648  --   00012000 .. 00012060   -- 12      -- 12   -- 'SYNC'
   010400 .. 010448  --   00013000 .. 00013060   -- 13      -- 13   -- 'SYNC'
   011200 .. 011248  --   00014000 .. 00014060   -- 14      -- 14   -- 'SYNC'
   011400 .. 011512  --   00014250 .. 00014390   -- 14      -- 0    -- 'FL16'
   011600 .. 011712  --   00014500 .. 00014640   -- 14      -- 0    -- 'FR17'
   011800 .. 011912  --   00014750 .. 00014890   -- 14      -- 0    -- 'RL18'
   012000 .. 012048  --   00015000 .. 00015060   -- 15      -- 15   -- 'SYNC'
   012200 .. 012312  --   00015250 .. 00015390   -- 15      -- 0    -- 'RR19'
   012800 .. 012848  --   00016000 .. 00016060   -- 16      -- 16   -- 'SYNC'
   013600 .. 013648  --   00017000 .. 00017060   -- 17      -- 17   -- 'SYNC'
   014400 .. 014448  --   00018000 .. 00018060   -- 18      -- 18   -- 'SYNC'
   015200 .. 015248  --   00019000 .. 00019060   -- 19      -- 19   -- 'SYNC'
   015400 .. 015512  --   00019250 .. 00019390   -- 19      -- 0    -- 'FL20'
   015600 .. 015712  --   00019500 .. 00019640   -- 19      -- 0    -- 'FR21'
   015800 .. 015912  --   00019750 .. 00019890   -- 19      -- 0    -- 'RL22'
   016000 .. 016048  --   00020000 .. 00020060   -- 20      -- 20   -- 'SYNC'
   016200 .. 016312  --   00020250 .. 00020390   -- 20      -- 0    -- 'RR23'
   016800 .. 016848  --   00021000 .. 00021060   -- 21      -- 21   -- 'SYNC'
   017200 .. 017312  --   00021500 .. 00021640   -- 21      -- 0    -- 'FL24'
```

```
   017400 .. 017512  --   00021750 .. 00021890   -- 21      -- 0    -- 'FR25'
   017600 .. 017648  --   00022000 .. 00022060   -- 22      -- 22   -- 'SYNC'
   017800 .. 017912  --   00022250 .. 00022390   -- 22      -- 0    -- 'RL26'
   018000 .. 018112  --   00022500 .. 00022640   -- 22      -- 0    -- 'RR27'
   018400 .. 018448  --   00023000 .. 00023060   -- 23      -- 23   -- 'SYNC'
   019200 .. 019248  --   00024000 .. 00024060   -- 24      -- 24   -- 'SYNC'
   020000 .. 020048  --   00025000 .. 00025060   -- 25      -- 25   -- 'SYNC'
   020800 .. 020848  --   00026000 .. 00026060   -- 26      -- 26   -- 'SYNC'
   021200 .. 021312  --   00026500 .. 00026640   -- 26      -- 0    -- 'FL28'
   021400 .. 021512  --   00026750 .. 00026890   -- 26      -- 0    -- 'FR29'
   021600 .. 021648  --   00027000 .. 00027060   -- 27      -- 27   -- 'SYNC'
   021800 .. 021912  --   00027250 .. 00027390   -- 27      -- 0    -- 'RL30'
   022000 .. 022112  --   00027500 .. 00027640   -- 27      -- 0    -- 'RR31'
   022400 .. 022448  --   00028000 .. 00028060   -- 28      -- 28   -- 'SYNC'
   023200 .. 023248  --   00029000 .. 00029060   -- 29      -- 29   -- 'SYNC'
   024000 .. 024048  --   00030000 .. 00030060   -- 30      -- 30   -- 'SYNC'
   024800 .. 024848  --   00031000 .. 00031060   -- 31      -- 31   -- 'SYNC'
-- END OF MESSAGE SET H SCHEDULE---


-- FIRM MESSAGES SCHEDULE BY PRIORITY------------
```

A list of event triggered messages, start and stop times only the first invocation.

```
-- Start  .. Stop (NTU)   Start    .. Stop (us)
   000048 .. 000160  --   00000060 .. 00000200   -- -- 'f1' (0)
   000160 .. 000272  --   00000200 .. 00000340   -- -- 'f2' (1)
   000272 .. 000384  --   00000340 .. 00000480   -- -- 'f3' (2)
   000384 .. 000496  --   00000480 .. 00000620   -- -- 'f4' (3)
   000496 .. 000608  --   00000620 .. 00000760   -- -- 'f5' (4)
   000608 .. 000720  --   00000760 .. 00000900   -- -- 'f6' (5)
   000848 .. 000960  --   00001060 .. 00001200   -- -- 'f7' (6)
   001088 .. 001200  --   00001360 .. 00001500   -- -- 'f8' (7)
   001312 .. 001424  --   00001640 .. 00001780   -- -- 'f9' (8)
   001648 .. 001760  --   00002060 .. 00002200   -- -- 'f10' (9)
   001872 .. 001984  --   00002340 .. 00002480   -- -- 'f11' (10)
   002712 .. 002824  --   00003390 .. 00003530   -- -- 'f12' (11)
   002976 .. 003088  --   00003720 .. 00003860   -- -- 'f13' (12)
   003088 .. 003200  --   00003860 .. 00004000   -- -- 'f14' (13)
   003248 .. 003360  --   00004060 .. 00004200   -- -- 'f15' (14)
-- END OF MESSAGE SET F SCHEDULE---


-- UTILISATION ------------
-- Class h messages: 3408 NTU (13%)
-- Class f messages: 11424 NTU (44%)
-- Class s messages: 8624 NTU (33%)
-- END OF SCHEDULE---
```

Appendix C: Simulation output

Simulation displays class *h* and *f* messages during the entire LCM.

```
class message invocation start-stop
h 'SYNC'(1) 0-48
f 'f1'(1) 48-160
f 'f2'(1) 160-272
f 'f3'(1) 272-384
f 'f4'(1) 384-496
f 'f5'(1) 496-608
f 'f6'(1) 608-720
h 'SYNC'(2) 800-848
f 'f7'(1) 848-960
h 'FL2'(1) 1000-1112
f 'f8'(1) 1112-1224
f 'f9'(1) 1312-1424
h 'SYNC'(3) 1600-1648
f 'f10'(1) 1648-1760
h 'RR5'(1) 1800-1912
f 'f11'(1) 1912-2024
f 'f1'(2) 2048-2160
f 'f2'(2) 2160-2272
f 'f3'(2) 2272-2384
f 'f4'(2) 2384-2496
h 'HMICom7'(1) 2600-2712
f 'f12'(1) 2712-2824
f 'f13'(1) 2976-3088
f 'f14'(1) 3088-3200
h 'SYNC'(5) 3200-3248
f 'f15'(1) 3248-3360
h 'SYNC'(6) 4000-4048
f 'f1'(3) 4048-4160
f 'f2'(3) 4160-4272
f 'f3'(3) 4272-4384
f 'f4'(3) 4384-4496
f 'f5'(2) 4496-4608
f 'f6'(2) 4608-4720
h 'SYNC'(7) 4800-4848
f 'f7'(2) 4848-4960
h 'RR11'(1) 5000-5112
f 'f8'(2) 5112-5224
f 'f9'(2) 5312-5424
h 'SYNC'(8) 5600-5648
f 'f10'(2) 5648-5760
h 'FL12'(1) 6000-6112
f 'f1'(4) 6112-6224
f 'f2'(4) 6224-6336
f 'f3'(4) 6336-6448
f 'f4'(4) 6448-6560
h 'RL14'(1) 6600-6712
h 'RR15'(1) 6800-6912
h 'SYNC'(10) 7200-7248
h 'SYNC'(11) 8000-8048
f 'f1'(5) 8048-8160
f 'f2'(5) 8160-8272
f 'f3'(5) 8272-8384
f 'f4'(5) 8384-8496
f 'f5'(3) 8496-8608
f 'f6'(3) 8608-8720
h 'SYNC'(12) 8800-8848
f 'f7'(3) 8848-8960
f 'f8'(3) 9088-9200
f 'f9'(3) 9312-9424
h 'SYNC'(13) 9600-9648
f 'f10'(3) 9648-9760
f 'f11'(2) 9872-9984
f 'f1'(6) 10048-10160
f 'f2'(6) 10160-10272
f 'f3'(6) 10272-10384
f 'f4'(6) 10384-10496
f 'f12'(2) 10712-10824
f 'f13'(2) 10976-11088
f 'f14'(2) 11088-11200
h 'SYNC'(15) 11200-11248
f 'f15'(2) 11248-11360
h 'FL16'(1) 11400-11512
h 'FR17'(1) 11600-11712
h 'RL18'(1) 11800-11912
h 'SYNC'(16) 12000-12048
f 'f1'(7) 12048-12160
f 'f2'(7) 12160-12272
f 'f3'(7) 12272-12384
f 'f4'(7) 12384-12496
f 'f5'(4) 12496-12608
f 'f6'(4) 12608-12720
h 'SYNC'(17) 12800-12848
f 'f7'(4) 12848-12960
f 'f8'(4) 13088-13200
f 'f9'(4) 13312-13424
h 'SYNC'(18) 13600-13648
f 'f10'(4) 13648-13760
f 'f1'(8) 14048-14160
f 'f2'(8) 14160-14272
f 'f3'(8) 14272-14384
f 'f4'(8) 14384-14496
h 'SYNC'(20) 15200-15248
h 'FL20'(1) 15400-15512
h 'FR21'(1) 15600-15712
h 'RL22'(1) 15800-15912
h 'SYNC'(21) 16000-16048
f 'f1'(9) 16048-16160
f 'f2'(9) 16160-16272
f 'f3'(9) 16272-16384
f 'f4'(9) 16384-16496
f 'f5'(5) 16496-16608
f 'f6'(5) 16608-16720
h 'SYNC'(22) 16800-16848
f 'f7'(5) 16848-16960
f 'f8'(5) 17088-17200
h 'FL24'(1) 17200-17312
f 'f9'(5) 17312-17424
h 'SYNC'(23) 17600-17648
f 'f10'(5) 17648-17760
h 'RL26'(1) 17800-17912
f 'f11'(3) 17912-18024
f 'f1'(10) 18048-18160
f 'f2'(10) 18160-18272
f 'f3'(10) 18272-18384
f 'f4'(10) 18384-18496
f 'f12'(3) 18712-18824
f 'f13'(3) 18976-19088
f 'f14'(3) 19088-19200
h 'SYNC'(25) 19200-19248
f 'f15'(3) 19248-19360
h 'SYNC'(26) 20000-20048
f 'f1'(11) 20048-20160
f 'f2'(11) 20160-20272
f 'f3'(11) 20272-20384
f 'f4'(11) 20384-20496
f 'f5'(6) 20496-20608
f 'f6'(6) 20608-20720
h 'SYNC'(27) 20800-20848
f 'f7'(6) 20848-20960
f 'f8'(6) 21088-21200
h 'FL28'(1) 21200-21312
f 'f9'(6) 21312-21424
h 'SYNC'(28) 21600-21648
f 'f10'(6) 21648-21760
h 'RL30'(1) 21800-21912
h 'RR31'(1) 22000-22112
f 'f1'(12) 22112-22224
f 'f2'(12) 22224-22336
f 'f3'(12) 22336-22448
f 'f4'(12) 22448-22560
h 'SYNC'(30) 23200-23248
h 'SYNC'(31) 24000-24048
f 'f1'(13) 24048-24160
f 'f2'(13) 24160-24272
f 'f3'(13) 24272-24384
f 'f4'(13) 24384-24496
f 'f5'(7) 24496-24608
f 'f6'(7) 24608-24720
h 'SYNC'(32) 24800-24848
f 'f7'(7) 24848-24960
f 'f8'(7) 25088-25200
f 'f9'(7) 25312-25424
```