IPP-HURRAY! Research Group

http://www.hurray.isep.ipp.pt

Polytechnic Institute of Porto
School of Engineering (ISEP-IPP)

Designing Real-Time Systems Based on Mono-Master Profibus-DP Networks

Salvatore MONFORTE
Mário ALVES
Francisco VASQUES (FEUP)
Eduardo TOVAR

HURRAY-TR-0013
July 2000

relatório técnico

technical report

THIS WORK IS PARTIALLY SUPPORTED BY THE EUROPEAN COMMISSION UNDER PROJECT IST-1999-11316 R-FIELDBUS

Designing Real-Time Systems Based on Mono-Master Profibus-DP Networks

Salvatore MONFORTE*, Mário ALVES, Eduardo TOVAR
IPP-HURRAY! Research Group
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340502 Fax: +351.22.8321159
E-mail: { salvo@dei, malves@dee, emt@dei}.isep.ipp.pt
http://www.hurray.isep.ipp.pt

Francisco VASQUES
Departamento de Engenharia Mecânica e Gestão Industrial
Faculdade de Engenharia da Universidade do Porto
Rua dos Bragas
4099 Porto Codex
Portugal
Tel.: +351.22.2041751 Fax: +351.22.2059125
E-mail: vasques@fe.up.pt

* On leave from the
Institute of Computer Science and Telecommunications (IIT)
V.le A.Doria, 6
95125 Catania
Italy
Tel.:+39 095 738 2362, 7 Fax: :+39 095 738 2397
E-mail: smonforte@iit.unict.it

Abstract:

Profibus networks are widely used as the communication infrastructure for supporting distributed computer-controlled applications. Most of the times, these applications impose strict real-time requirements. Profibus-DP has gradually become the preferred Profibus application profile. It is usually implemented as a mono-master Profibus network, and is optimised for speed and efficiency. The aim of this paper is to analyse the real-time behaviour of this class of Profibus networks. Importantly, we develop a new methodology for evaluating the worst-case message response time in systems where high-priority and cyclic low-priority Profibus traffic coexist. The proposed analysis constitutes a powerful tool to guarantee prior to runtime the real-time behaviour of a distributed computer-controlled system based on a Profibus network, where the real-time traffic is supported either by high-priority or by cyclic poll Profibus messages.

DESIGNING REAL-TIME SYSTEMS BASED ON MONO-MASTER PROFIBUS-DP NETWORKS [1]

Salvatore Monforte[‡], **Mário Alves**[†], **Francisco Vasques**[ξ], **Eduardo Tovar**[†]

[‡] *IPP-HURRAY Group, on leave from Dept. of Computer Science and Telecommunications, University of Catania, Italy, E-mail: smonforte@iit.unict.it*
[†] *IPP-HURRAY Group, Polytechnic Institute of Porto, Portugal, E-mail: {malves@dee, emt@dei}.isep.ipp.pt*
[ξ] *DEMEGI-FEUP, University of Porto, Portugal, E-mail: vasques@fe.up.pt*

Abstract: Profibus networks are widely used as the communication infrastructure for supporting distributed computer-controlled applications. Most of the times, these applications impose strict real-time requirements. Profibus-DP has gradually become the preferred Profibus application profile. It is usually implemented as a mono-master Profibus network, and is optimised for speed and efficiency. The aim of this paper is to analyse the real-time behaviour of this class of Profibus networks. Importantly, we develop a new methodology for evaluating the worst-case message response time in systems where high-priority and cyclic low-priority Profibus traffic coexist. The proposed analysis constitutes a powerful tool to guarantee prior to runtime the real-time behaviour of a distributed computer-controlled system based on a Profibus network, where the real-time traffic is supported either by high-priority or by cyclic poll Profibus messages.

Keywords: Fieldbus Networks, Real-time Communication.

1. INTRODUCTION

A computer-controlled system can be decomposed into a set of three subsystems: the controlled object; the computer system; and the human operator (Kopetz, 1997). Collectively, the controlled object and the human operator can be referred to as the environment of the computer system.

The role of the computer system is to react to stimuli from the controlled object or the operator. Basically, the computer system should be able to receive, via the instrumentation interface, information about the status of the controlled object, compute new commands according to the references provided by the man-machine interface, and transmit those new commands to the actuators, also via the instrumentation interface. A computer-controlled system can have a centralised architecture, where the field devices (e.g., sensors and actuators) are connected to the computer system via point-to-point links. However, there are several advantages if a field level communication network is used as a replacement for the point-to-point links. The main advantage is an economical one. Naturally, the use of a bus brings other important advantages, such as easier installation and maintenance, easier detection and localisation of cable faults, and easier expansion due to the modular nature of the network. The ability to support distributed control algorithms is another advantage achievable by the use of field level networks.

Typically, a field level network is a broadcast network, where several network nodes share a common communication channel. Messages are transmitted from a source node to a destination node via the shared communication medium. A major problem occurs when at least two nodes attempt to send messages via the shared medium at about the same time. This

[1] This work was partially supported by the European Commission under the project R-FIELDBUS (IST-1999-11316), by FLAD under the project SISTER (471/97), and by IDEMEC.

problem is solved by a medium access control (MAC) mechanism.

Most computer-controlled systems are also real-time systems. In general, the issue of guaranteeing real-time requirements is one of checking, prior to run-time, the feasibility of the system's task set; that is, checking if the worst-case execution time of the tasks is smaller than its admissible response time.

In distributed computer-controlled systems, where some of the application tasks are also communicating tasks, it is of paramount importance the evaluation of the messages' response time, since this response time is one of the components of the end-to-end communication latencies.

Therefore, a potential leap towards the use of field level communication networks (fieldbus) in time-critical applications lies in the evaluation of its temporal behaviour.

Profibus (Profibus, 1996) is a well-known fieldbus network that distinguishes between two types of devices - masters and slaves - and supports both mono-master and multi-master systems. A Profibus master is a network device that can send a message on its own initiative, once it gains the right to access the bus. Profibus slaves are devices that may only acknowledge or respond to requests from masters. Generally, they are peripherals such as I/O devices, valves, drives etc.

A widely-used class of Profibus networks is the Profibus-DP (Profibus-DP, 2000) profile. It is generally implemented as a mono-master Profibus network, optimised for speed and efficiency, since it does not implement all the usual communication layers' protocol.

The aim of this paper is to analyse the real-time behaviour of this class of Profibus networks. Importantly, we develop a new methodology for evaluating the worst-case message response time in systems where high-priority and cyclic low-priority Profibus traffic coexist. The proposed analysis constitutes a powerful tool to guarantee prior to runtime the real-time behaviour of a distributed computer-controlled system based on a Profibus network, where the real-time traffic is supported either by high-priority or by cyclic poll Profibus messages.

In (Vasques and Juanole, 1994) the authors provide a real-time analysis of Profibus messages. However, do not consider that Profibus message requests are queued in a FCFS (First-Come-First-Served) queue. Further more, their analysis does not provide any estimation of the worst-case response time of each individual message. In (Tovar and Vasques, 1999a; Tovar and Vasques, 1999b), the authors develop a response time analysis. However, this is intended for multi-master systems and if applied to the mono-master system it would lead to very pessimistic results, since the authors consider always the worst-case token rotation time. Moreover, none of these works, consider the evaluation of response time guarantees for the cyclic poll Profibus messages. This type of messages was analysed in (Li and Stoeckli, 1993). In this approach, message deadlines are guaranteed since the token cycle time is bounded. The major drawback of this approach is that, in order to evaluate the token cycle time, neither high-priority traffic nor low-priority traffic (other than cyclic traffic) is allowed. This is very restrictive in terms of using Profibus to support real-time distributed computer-controlled applications.

The remainder of this paper is organised as follows. In Section 2 we give a brief description of the main Profibus characteristics. In Section 3 we introduce the network and message models used in the proposed real-time analysis. In Section 4 we propose a methodology to evaluate the Profibus temporal properties, which will be used as a basis for the response time analysis performed in Section 5. In Section 6 a numerical example is given and a realistic scenario of an industrial computer-controlled system is analysed, demonstrating the interest of the proposed analysis. Finally, in Section 7 we draw some conclusions.

2. BASIC CONCEPTS OF PROFIBUS

2.1. Message Cycle

The MAC protocol of Profibus is a simplified version of the timed token protocol (Grow, 1982). The bus access is based on a hybrid method where masters use a token-passing procedure to grant the bus access and a master-slave procedure to communicate with slave stations.

An important Profibus concept is the *Message Cycle*, which comprises the *Action Frame* sent by the *initiator* (always a master) and the associated A*cknowledge* or R*esponse Frame* sent by the *responder*. Profibus allows distinguishing between high-priority, cyclic low-priority (execution of the requests contained in the poll-list) and acyclic low-priority messages. Once the action frame has been transmitted, the initiator waits for the response during a Slot Time (T_{SL}). If a response is not received within T_{SL}, the initiator will try again up to a number of *max_retry_limit* retries.

Profibus provides a service to poll a list of sensors and actuators, by means of a pre-defined sequence of requests. This sequence is called the *Poll List*. The processing of all the *Poll List* entries is said to be a *Poll Cycle*. The *Poll Cycle* duration depends on the length of each message cycle, on the number of message cycles processed at each token arrival and on the token rotation time. Hence, it is obvious that a *Poll Cycle* may last for several token-holding periods. If the *Poll Cycle* is completed within a token holding period, the next *Poll Cycle* may only start at the next receipt of the token. Otherwise, the *Poll List* is processed in segments, without inserting acyclic low-priority message cycles.

2.2. Token Transmission and Reception

The token is passed between masters in ascending order of addresses. To close the logical ring, the master with the highest address passes the token back to the master with the smallest one. In the case of Profibus mono-master networks (Profibus-DP, 2000), the station just passes the token to itself. The advantage of preserving the same token-passing procedure is that they allow for an unambiguous scheduling of different classes of traffic (high-priority and cyclic/acyclic low-priority), preserving all the properties found in multi-master Profibus networks.

2.3. Message Dispatching

At token reception, the period during which the master station is allowed to perform message cycles (*Token Holding Time)* is computed as $T_{TH} = T_{TR} - T_{RR}$, where T_{RR} (Real Rotation Time) is the time between two consecutive token arrivals and T_{TR} (Target Rotation Time) is the expected time for a token cycle.

When a master station receives the token, it processes at least one high-priority message (even if $T_{TH} < 0$). After that, the other pending high-priority message cycles are processed if and while $T_{TH} \geq 0$. It should be pointed out that once a message cycle is started, it is always completed, including any retry (retries), even if meanwhile T_{TH} gets smaller than 0.

The processing of the Poll List is only started after all requested high-priority message cycles have been processed. After each complete Poll Cycle (all entries of the Poll List processed), the requested low-priority message cycles are performed in turn. A new Poll Cycle starts at the next receipt of the token.

3. NETWORK AND MESSAGE MODELS

Requests for message cycles are placed in high-priority, cyclic low-priority or acyclic low-priority outgoing queues. Let $Sh_i^k = (Ch_i^k, Dh_i^k, Th_i^k)$, $Sc_i^k = (Cc_i^k, Dc_i^k, Tc_i^k)$ and $Sa_i^k = (Ca_i^k, Da_i^k, Ta_i^k)$ be high-priority, cyclic and acyclic low-priority message streams in master k, respectively. A message stream is a temporal sequence of requests for message cycles concerning, for instance, the remote reading of a specific process variable.

Ch_i^k, Cc_i^k and Ca_i^k are maximum message cycle duration for a request of message stream Sh_i^k, Sc_i^k and Sa_i^k, respectively. This duration includes the time needed to transmit the request frame and completely receive the related response, and also the time needed to perform the allowed number of message retries. Th_i^k and Tc_i^k are the periodicity of streams Sh_i^k and Sc_i^k requests, respectively. We assume that this periodicity is the minimum interval between two consecutive arrivals of the related requests to the outgoing queue. Dh_i^k and Dc_i^k are the relative deadline of the related message cycle; that is, the maximum admissible time interval between the instant when the message request is placed in the outgoing queue and the instant when the related response is completely received at the master's incoming queue. Finally, nh^k and nc^k are the number of high-priority and cyclic low-priority message streams, respectively. We also consider that there is just one acyclic low-priority message stream per station.

4. TIMING PROPERTIES OF PROFIBUS

4.1. Token Cycle Properties

In this section, some token cycle properties will be analysed, for the case of Profibus mono-master networks. Namely it will be proved that the real token rotation time T_{RR} is generally smaller than T_{TR} in spite of knowing that once a message cycle is started, it is always completed. More formally, let us introduce the following definitions.

Definition 1 – Overrun – We define an overrun as the occurrence of a T_{TH} expiration while a message cycle is being processed.

Definition 2 – Overrun Window – We define an overrun window as the time window during which T_{TH} is exceeded due to the completion of a message cycle, added with the subsequent token passing interval.

Definition 3 – Late Token – A token is defined as being late if, at its arrival, the real token rotation T_{RR} is greater than the target token rotation time T_{TR}.

A late token arrival implies that at most one high-priority message can be processed by the related master station. It should also be noted that an overrun in a given token arrival, does not usually imply a late token on the next arrival.

Let us denote $A(l)$ as the token arrival instant for the l^{th} token visit. At the time instant $A(l)$, T^l_{TH} is assigned with the value $T_{TR} - T^{l-1}_{RR}$. Therefore, there will be a late token arrival only if $T^{l-1}_{RR} > T_{TR}$. Note that the real token rotation time is measured between token arrivals.

As depicted in Fig. 1, it is clear that the token is neither late in the l^{th} token visit nor in the $(l+1)^{th}$ visit, after the one where an overrun has occurred (as $T_{TR} > T^l_{RR}$). However, in some particular conditions the token can be late.

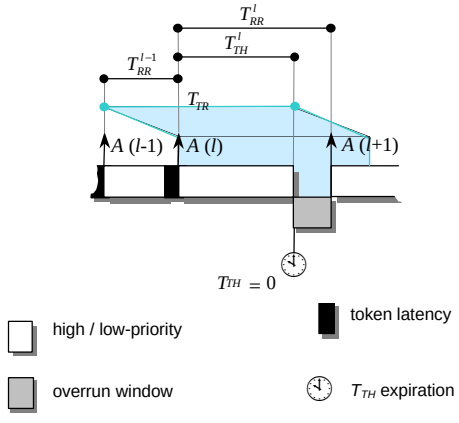

Fig. 1. Overrun and late token.

Theorem 1 – In a mono-master Profibus system, if in the l^{th} token visit an overrun occurs, then there will be a late token arrival if and only if the overrun window is greater than the value of T^{l-1}_{RR}.

Proof
Let us assume that in the l^{th} token visit an overrun occurs, and let w be the overrun window. As shown in Fig. 1, $T^{l}_{RR} = T^{l}_{TH} + w$. As $T^{l}_{TH} = T_{TR} - T^{l-1}_{RR}$, then:

$$T^{l}_{RR} = T_{TR} - T^{l-1}_{RR} + w \qquad (1)$$

We must prove that the token will be late at $(l + 1)^{th}$ token arrival, that is, $T^{l}_{RR} > T_{TR}$ if and only if $w > T^{l-1}_{RR}$. Since all quantities involved in expression (1) are positive, if $T^{l}_{RR} > T_{TR}$ then $0 < T^{l}_{RR} - T_{TR} = -T^{l-1}_{RR} + w \rightarrow T^{l}_{RR} < w$; vice-versa if $w > T^{l-1}_{RR}$ then obviously follows $T^{l}_{RR} > T_{TR}$. Hence:

$$T^{l}_{RR} > T_{TR} \Leftrightarrow w > T^{l-1}_{RR}$$

That is, a late token arrives at the $(l + 1)^{th}$ token visit if and only if the length of the overrun window is greater than the value of T_{RR} computed at the beginning of the l^{th} cycle, that is, is greater that T^{l-1}_{RR}. ❑

Corollary 1 – A low-priority overrun induces a late token if and only if in the previous token visit no low-priority and at most one high-priority message has been processed.

Proof
For a low-priority overrun the length of the longest overrun window is $w = Cl_{max} + t$, where $Cl_{max} > Ch_{max}$.

If in the previous token visit, no low-priority messages and at most one high-priority message were processed, at the token arrival $T_{RR} \leq Ch_{max} + t$, that is smaller than w and thus a late token will be induced on next visit. ❑

4.2. Basic Response Time Analysis

In (Liu and Layland, 1973), the author introduced the concept of critical instant as being the time instant at which a request for a given task has the longest response time, that is, the longest time interval till the end of the response for that request. Moreover, a critical instant for any task occurs whenever the task is requested *simultaneously* with requests for all higher priority tasks (Liu and Layland, 1973).

In a Profibus network, we must consider that, due to the FCFS behaviour of the outgoing queues, a given message request can be delayed by requests from all the other message streams (contrarily to the task scheduling case where it would be delayed only by high priority message stream requests). Therefore, a critical instant will occur when, for a given priority (i.e. high- or acyclic low-priority) every message stream simultaneously issue a message request.

Moreover, due to the non pre-emptive context of messages processing, high-priority request may suffer some additional delay before starting being processed. Let us introduce the following definitions:

Definition 4 – Profibus Critical Instant – Considering that requests for all high-priority, cyclic and acyclic low-priority message streams are simultaneously placed on the respective outgoing queues. We define a Profibus critical instant as the time instant at which a request for a given message stream has the longest response time.

Definition 5 – Initial Blocking – We define the initial blocking as the delay that the first request made at the critical instant may suffer until starting to be processed.

Definition 6 – Critical Load – We define the critical load for a given priority class, as the time interval between a critical instant and the time instant when the last request (made at the critical instant) for that priority class has been completely processed.

As far as the evaluation of the worst-case response time is concerned, two factors must be taken into account: the initial blocking and the high-priority critical load. The worst-case response time for both high-priority and low-priority messages stream made at the critical instant results from the simultaneous occurrence of:

1. the longest initial blocking, that is, the first high-priority request suffers the longest possible delay before being processed;
2. the longest high-priority critical load, that is, it takes the maximum number of token visits to process all high-priority requests.

The following two theorems prove that the simultaneous occurrence of both conditions leads to the worst-case response time for the last message request to be processed.

Theorem 2 – The longest initial blocking occurs when all requests are issued simultaneously with the occurrence of a low-priority overrun.

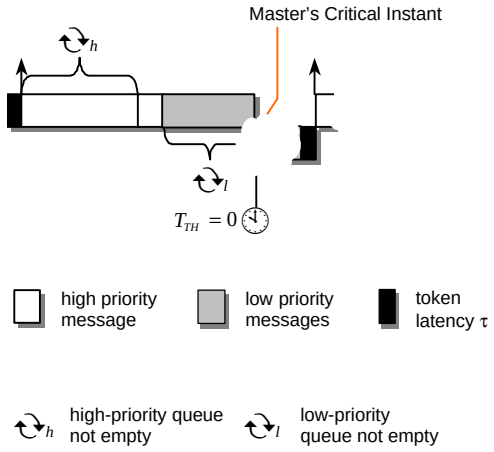

Fig. 2. Master's Critical Instant.

Proof of Theorem 2
Considering that the initial blocking depends on the position of the critical instant itself, the theorem is proved if a shifting of its position leads either to a deadline violation or to a smaller blocking.

Referring to Fig. 2, if the position of the critical instant is anticipated (shifted to the left), this leads to a deadline violation, since it means that requests were issued while the queue was not empty (either for low-priority or high-priority).

Finally, if the critical instant is postponed (shifted to the right), this leads to a smaller blocking. ❑

Corollary 2 – The longest initial blocking is $B = Cl_{max} + t$.

Proof
From Fig. 2, it is clear that the longest low-priority overrun is $Cl_{max} + t$. ❑

The length of the high-priority critical load interval depends not only on the occurrence of the longest initial blocking, but also on the traffic processed on the previous token cycle.

Theorem 3 – A Profibus critical instant occurs when the longest initial blocking is preceded by a token visit where no low-priority and at most one high-priority message is processed.

Proof
As depicted in Fig. 3, a late token arriving after a Profibus critical instant leads to a longer high-priority critical load, since the second high-priority request to be processed will suffer a delay of t. Taking into account that the maximum initial blocking is equal to the longest low-priority overrun window (Corollary 2); and in view of Corollary 1, after the critical instant a late token arrives if no low-priority message and at most one high-priority message is processed in the previous token visit. ❑

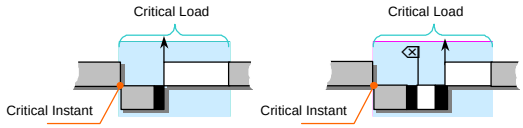

Fig. 3. Late token and high-priority critical load.

Corollary 3 – The maximum high-priority load interval leads to the worst-case response time for both cyclic and acyclic low-priority message streams.

Proof
Considering that low-priority message cycles are performed in turn only after all high-priority message have been processed, its is clear that the maximum high-priority load interval leads to the maximum span of time between the critical instant and the time instant at which the last low-priority message (either cyclic or acyclic) is processed; and thus to the worst-case response time evaluation. ❑

4.3. *Processing of High Priority Messages*

Since all the possible requests are issued at the critical instant and due to the non pre-emptive context of Profibus, a master may need several token visits to process all high-priority messages, before processing any low-priority request. Thus, there will be a well-defined pattern when processing all those requests (Fig. 4).

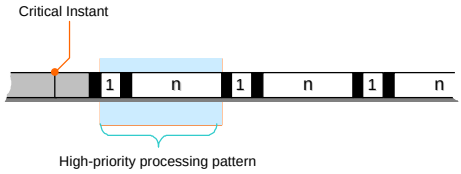

Fig. 4. The 1-*n* processing pattern.

This processing pattern is characterised by a late token arrival, where just *one* high-priority message is processed, followed by an early token arrival, where *n* high-priority messages are processed.

Theorem 4 – The occurrence of a Profibus critical instant induces a 1-*n* processing pattern for high-priority messages.

Proof
From the analysis, it follows that after the occurrence of a critical instant, a late token arrives. Thus, as depicted in Fig. 5, only one high-priority message can be processed (❶) and on next token visit T_{TH} is assigned with a maximum values equal to (T_{TR} - Ch_{max} - t). Therefore, if the number of high-priority requests issued at the critical instant is greater than the number of high-priority messages which can be processed

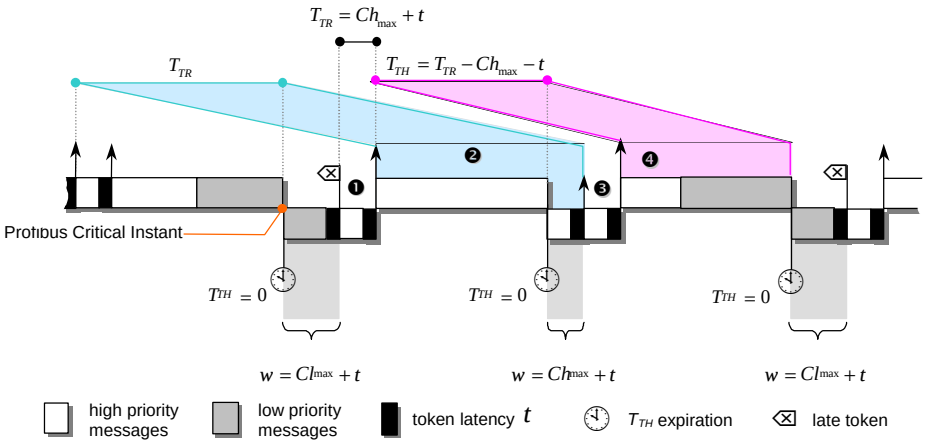

Fig. 5. Evaluation of the 1-*n* high-priority processing pattern.

during T_{TH}, then a high-priority overrun occurs, and, in the worst case, the length of the overrun window is (Ch_{max} + t). Consequently, the token real rotation (T_{RR}) equals the target token rotation time (T_{TR}) (❷), and on next token visit only one message cycle will be carried out (❸). The 1-n is clearly defined, where (n -1) messages are processed before the expiration of T_{TH} and the last of the n messages is processed in overrun. The maximum time spent processing these n messages is equal to (T_{TR} - Ch_{max} - t) for processing (n -1) messages, plus (Ch_{max} + t) to process the message in overrun, that is, is equal to T_{TR}. It is also clear that if:

- the smallest token cycle always comprises one high-priority message cycle; and
- in a token cycle where at least one low-priority message is processed the token is never released before the expiration of the token holding time timer (T_{TH});

then, in the last token cycle where there are still high-priority request pending, the token holding timer is assigned with $T_{TH} = T_{TR} - Ch_{max} - t$ (❹), and this occurs after a token cycle which comprise no low-priority and exactly one high-priority message cycle (❶, ❸). Moreover, if an overrun of T_{TH} occurs then in the next token cycle no low-priority and at most one high-priority message can be processed (❺). ❑

5. WORST-CASE RESPONSE TIME

5.1. *High-priority Message Streams*

The worst-case response time for high-priority message can be computed taking into account the following 4 components:

1. the initial blocking;
2. the time spent processing requests in token visits where n high-priority messages can be processed, plus the time spent to pass the token, that is, $\lfloor nh/n \rfloor \times T_{TR}$;
3. the time spent processing requests in token visits where just *1* high-priority message can be processed, plus the time spent to pass the token, that is, $\lfloor nh/n \rfloor \times (Ch_{max} + t)$;
4. a component Y_h, which is related to the finishing of the 1-n processing pattern. We consider that at the end of the last complete cycle of n messages, there are three possible cases:
   a) there are <u>no</u> more pending requests, and thus the computation of the response time ends before releasing the token in the previous token cycle;
   b) there is <u>just one</u> pending request, and thus the time needed to process the pending requests is exactly Ch_{max};
   c) there are <u>more than one</u> pending request, and thus one more token cycle is needed to process the pending requests.

Hence, the worst-case response time for Profibus high-priority messages is:

$$R_h = B + \lfloor nh/(n+1) \rfloor \cdot (T_{TR} + Ch_{\max} + t) + y_h \qquad (2)$$

where:

$$y_h = \begin{cases} -t, \; if \;\; nh = \lfloor nh/(n+1) \rfloor \cdot (n+1) \\ Ch_{\max}, \; if \;\; nh = \lfloor nh/(n+1) \rfloor \cdot (n+1) + 1 \\ (nh - \lfloor nh/(n+1) \rfloor \cdot (n+1)) \cdot Ch_{\max} + t, otherwise \end{cases}$$

and

$$n = \lfloor (T_{TR} - Ch_{\max} - t)/Ch_{\max} \rfloor + 1 = \lfloor (T_{TR} - t)/Ch_{\max} \rfloor$$

5.2. *Cyclic Low-Priority Message Streams*

The processing of low-priority requests issued at the critical instant, is alternated by periods where new (those requested during the processing of low-priority messages) high-priority message requests are carried out (Fig. 6).

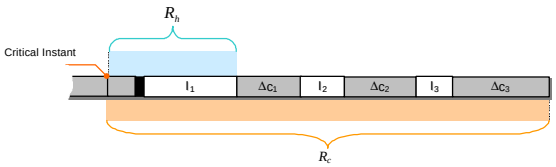

Fig. 6. Interleaving of interference intervals and cyclic processing intervals.

<u>Definition 7</u> – Interference Interval – We define an interference interval I_i as the i^{th} time window during which only high-priority messages are processed.

<u>Definition 8</u> – Cyclic Processing Interval – We define a cyclic processing interval DC_i as the i^{th} time window during which low-priority traffic is processed.

It is clear that considering both cyclic processing intervals and interference intervals it is possible to evaluate the worst-case response time for cyclic-low priority message streams.

The length of the i^{th} cyclic processing interval can be computed as the difference between the value of $T_{TR} = T_{TR} - Ch_{max} - t$ and the maximum amount of time used to process the remaining high-priority messages, that is:

$$\Delta c_i = (T_{TR} - Ch_{\max} - t) - \\ - \max\left(0, \left(n_i^h - \lfloor n_i^j/(n+1) \rfloor \cdot (n+1)\right) - 1\right) \cdot Ch_{\max} + Cl_{\max} + t$$

where n_i^h is the number of high-priority messages processed in I_i.

It should be noticed that every high-priority request which arrives within DC_i will be pending on next token cycle (start of I_i.). Moreover, there is a mutual dependence between the evaluation of the number n_i^h of high-priority messages processed in the i^{th} interference interval and the length of this interval itself:

$$\sum_{j=1}^{nh} \left\lfloor \left(I_i + \sum_{m=1}^{i-1} (I_m + \Delta c_m) + B \right) \Big/ Th_j \right\rfloor \qquad (3)$$

The interference imposed by the processing of n_i^h high-priority messages is:

$$I_i = \left\lfloor \frac{n_i^h}{n+1} \right\rfloor \cdot (T_{TR} + Ch_{\max} + t) + Ch_{\max} + t + \\ + \max\left(0, \left(nh_i - \left\lfloor \frac{n_i^h}{n+1} \right\rfloor \cdot (n+1) \right) - 1 \right) \cdot Ch_{\max}$$

Finally the worst-case response time for cyclic low-priority message streams can be expressed as follows:

$$R_c = B + \sum_{i=1}^{m-1} (\Delta c_i + I_i) + I_m + \left(nc - \sum_{i=1}^{m-1} n_i^c \right) \cdot Cl_{\max} \qquad (4)$$

where $n_i^c = \lfloor (DC_i - t) / Cl_{max} \rfloor$ is the number of cyclic low-priority messages in DC_i and $m = \min\{n \in \aleph: \sum_{i=1,\ldots,n} n_i^c \geq nc\}$.

6. NUMERICAL EXAMPLE

In this section a realistic scenario for an industrial computer-controlled system is utilised to demonstrate the interest of the proposed analysis.

Consider a mono-master Profibus network in an industrial environment, where a decentralised computer-controlled system integrates video message streams, computer-generated audio messages and control-related message streams. The supported application controls an assembly production line where parts must be assembled and checked. Both completeness tests (i.e. check if all parts are complete and in position) and dimension tests (i.e. verifying if parts are within the prescribed tolerance for diameters, distances, radii, angles etc.) are required for the assembly process. An intelligent stand-alone vision systems is used to directly evaluate images according to the stored testing program. Thus, only the data resulting from the video inspection operation will be sent through the Profibus interface.

The control-related message streams, which interconnect sensors and actuators to controllers, are mapped on the high-priority message streams. Table 1, summarises the characterisation of the high-priority message streams considered in this example, where set *x* represents a set of message streams with the same periodicity.

Table 1 Summary of high-priority message streams

	Set 1	Set 2	Set 3	Set 4
nh=20	3	5	7	5
Th_i	20 *ms*	25 *ms*	50 *ms*	60 *ms*

In order to meet realistic requirements, a value for the Slot Time (T_{SL}) and for the target token rotation time (T_{TR}) has been fixed to 100μs and 8ms, respectively. Moreover, a 1.5Mbps data transfer rate and no retry (r = 0) are considered in the example. Therefore, the bit period is equal to 0.667μs and the computation for the token-passing latency, yields: $t = 3 \times (T_{TF} + T_{SL}) = 3 \times (22 + 100) = 0.366$μs, where T_{TF} is the token frame length and 3 is the maximum number of retries predefined for the case of the token frame.

Concerning message streams to support the video inspection capabilities, we assume that the maximum data size for the result of an image evaluation is 246 bytes (equal to the maximum length for cyclic low-priority messages) and the image shutter rate is 20 picture/s. These requirements leads to polling each camera device every 50ms.

Concerning the multimedia services, which are also mapped over cyclic low-priority message streams, we assume 128Kbps as the bandwidth requirement. Hence, for a multimedia video stream application with 352x288 frame format at 10 frames/s, each video device must be polled every 15ms.

Table 2 summarises the cyclic low-priority message streams characterisation considering the utilisation of both 5 intelligent Profibus cameras for part inspection and 2 surveillance video cameras.

Table 2 Summary of cyclic message streams

	Set 1	Set 2
nc=7	2	5
Tc_i	15 *ms*	50 *ms*

Table 3 Message cycles length

	bytes	length
Ch_{max}	20 bytes	0.433*ms*
Cl_{max}	246 bytes	1.569*ms*

Considering the length for cyclic low-priority message cycle presented in Table 3, the maximum initial blocking is: $B = Cl_{max} + t = 1.935$ms. Thus, the number of high-priority message processed in a 1-*n* processing pattern is: $n + 1 = \lfloor (T_{TR}+Ch_{max} - t) / Ch_{max} \rfloor = \lfloor 18.617 \rfloor = 18$. Therefore, the worst-case response time for high-priority message streams is:

$$R_h = B + \left\lfloor \frac{nh}{n+1} \right\rfloor \cdot (T_{TR} + Ch_{\max} + t) + y_h = \\ = 1.935\ ms + \left\lfloor \frac{20}{18} \right\rfloor \times 8.799\ ms + 1.232\ ms = 11.967\ ms$$

Thus, the high-priority message stream set is schedulable as $R_h < 15\text{ms} = \min\{Th_i\}$.

Let us now compute the worst-case response time for cyclic low-priority message streams. The first interference interval is the one imposed by processing the nh high-priority requests made at the critical instant:

$$I_1 = \left\lfloor \frac{nh}{n+1} \right\rfloor \cdot (T_{TR} + Ch_{\max} + t) + Ch_{\max} + t + \\ + \max\left(0, \left(nh - \left\lfloor \frac{nh}{n+1} \right\rfloor \cdot (n+1) \right) - 1 \right) \cdot Ch_{\max} = \\ = 10.032 ms$$

The length of the first cyclic processing interval is:

$$\Delta c_1 = (T_{TR} - Ch_{\max} - t) - \\ - \max\left(0, \left(nh - \left\lfloor \frac{nh}{n+1} \right\rfloor \cdot (n+1)\right) - 1\right) \cdot Ch_{\max} + \\ + Cl_{\max} + t = 8.703\ ms$$

The number of cyclic low-priority requests that can be accomplished in DC_i is then:

$$n_1^c = \left\lfloor \frac{\Delta c_1 - t}{Cl_{\max}} \right\rfloor = \left\lfloor \frac{8.703\ ms - 0.366\ ms}{1.569\ ms} \right\rfloor = \lfloor 5.312 \rfloor = 5$$

Hence, there are still 2 cyclic pending requests. It should be reminded that there is a mutual dependence between the evaluation of the number n_i^h of high-priority messages processed in the i^{th} interference interval and the length of the interval itself. Hence, in order to compute n_2^h we need to know the value of I_2 and vice versa. From equation (3), it follows that:

$$n_2^h = nh + \sum_{j=1}^{nh} \left\lfloor \frac{I_2 + (I_1 + \Delta c_1) + B}{Th_j} \right\rfloor - nh = \\ = \sum_{j=1}^{nh} \left\lfloor \frac{I_2 + 20.67\ ms}{Th_j} \right\rfloor$$

Hence:

$$I_2 = \left\lfloor \frac{n_2^h}{18} \right\rfloor \times 8.799\ ms + 0.799\ ms + \\ + \max\left(0, \left(n_2^h - \left\lfloor \frac{n_2^h}{18} \right\rfloor \times 18\right) - 1\right) \times 0.433\ ms$$

The easiest way to solve such dependence is to form a recurrence relationship:

$$n_2^{h[m+1]} = \sum_{j=1}^{nh} \left\lfloor \frac{\left\lfloor \frac{n_2^{h[m]}}{18} \right\rfloor \times 8.79 + 0.79 + \Theta \times 0.43 + 20.67}{Th_j} \right\rfloor,$$

$$\Theta = \max\left(0, \left(n_2^{h[m]} - \left\lfloor \frac{n_2^{h[m]}}{18} \right\rfloor \times 18\right) - 1\right)$$

It follows that $n_2^{h[0]}=0$, $n_2^{h[1]}=3$ and $n_2^{h[2]}=3$. The iteration stops since $n_2^{h[1]}=n_2^{h[2]}$ and thus, $I_2 = 0.799 + 2 \times 0.433 = 1.666$ms. The length of the second cyclic processing interval is $DC_2 = (T_{TR} - Ch_{max} - \tau) + 2 \times Ch_{max} + Cl_{max} + t = 8.269$ms, which allows the processing of a number of cyclic low-priority messages is given by: $n_2^c = \lfloor (DC_2 - t) / Cl_{max} \rfloor = \lfloor (8.269 - 0.366) / 1.569 \rfloor = \lfloor 5.036 \rfloor = 5$. Therefore, considering that $nc^c < n_1^c + n_2^c$ the computation of worst-case response time for cyclic low-priority message streams is over and yields:

$$R_c = B + \Delta c_1 + I_1 + I_2 + (nc - n_1^c) \cdot Cl_{\max} = \\ = 20.67\ ms + 1.666\ ms + 3.139\ ms = 25.475\ ms$$

7. CONCLUSIONS

Profibus networks are often used as the communication infrastructure for supporting distributed computer-controlled applications. Generally, these applications impose strict real-time requirements. A potential leap towards the use of Profibus in time-critical applications lies in the evaluation of its temporal behaviour.

A widely-used class of Profibus networks is the Profibus-DP profile, usually implemented as a mono-master Profibus network. In this paper we analysed the real-time behaviour of this class of Profibus networks. Importantly, we developed a new methodology for evaluating the worst-case message response time in systems where high-priority and cyclic low-priority Profibus traffic coexist. The results presented in this paper constitute an important advance concerning the state-of-the-art in the sense that previous relevant works neither could be applied to the mono-master case nor considered the response time analysis for systems supporting both high priority and cyclic poll Profibus messages.

Work is now being carried out in order to extend the approach followed in this paper to multi-master Profibus networks.

8. REFERENCES

Grow, R. (1982). A Timed Token Protocol for Local Area Networks. In: *Proceedings of Electro'82*, Token Access Protocols, paper 17/3.

Kopetz, H. (1997). Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers.

Li, M. and L. Stoeckli (1993). The Time Characteristics of Cyclic Service in Profibus. In: *Proceedings of the EURISCON'94*, Vol. 3, pp. 1781-1786.

Liu, C. and J. Layland (1973). Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment. In: *Journal of the Association for Computer Machinery*, (20)1, pp. 46-61.

Profibus (1996). General Purpose Field Communication System, Volume 2, EN 50170.

Profibus-DP (2000). Profibus Technical Overview. http://www.profibus.com.

Tovar, E. and F. Vasques (1999a). Cycle Time Properties of the PROFIBUS Timed Token Protocol. In: *Computer Communications*, Elsevier Science, 22(13), pp. 1206-1216.

Tovar, E. and F. Vasques (1999a). Real-Time Fieldbus Communications Using PROFIBUS Networks. In: *IEEE Transactions on Industrial Electronics*, 46(6), pp. 1241-1251.

Vasques, F. and G. Juanole (1994). Pre-run-time Schedulability Analysis in Fieldbus Networks. In Proceedings of IECON'94, Bologna, Italy, pp. 1200-1204.