Design Methodology for Multifunction Vehicle Bus Devices

JAIME JIMÉNEZ, JAGOBA ARIAS, JON ANDREU, CARLOS CUADRADO, IÑIGO KORTABARRIA
Departamento de Electrónica y Telecomunicaciones
Universidad del País Vasco
Alameda Urquijo s/n, 48.013 Bilbao
ESPAÑA
http://det.bi.ehu.es/~apert

Abstract: - The specifications analysis for an MVB (Multifunction Vehicle Bus) bus administrator showed that system-on-a-chip strategies should be adopted to cope with its great complexity. Particularly, a new hardware/software codesign methodology has been followed. Its main concept is that the MVB devices (the bus administrator itself and the less complex devices) constitute a "progressive family". This notion establishes a functional partition that helps to generate the behavioral design: 14 operational blocks and a special memory for the communication data. The whole architecture has been coded in SystemC, not only for verification purposes, but also to set the start point in hardware/software partitioning. After validating this executable description by simulation, estimations about cost and performance in both, hardware and software, have been made. From these, an optimum hardware-software architecture has been obtained. As a result, the electronic platform for the Master device has been generated on an FPGA.

Key-Words: - Electronic Design methodology, hardware/software partitioning, Train Communication Network, Multifunction Vehicle Bus.

1 Introduction

In order to cope with designing a complex digital system [1-2], some strategies are:

- Abstraction and top-down design [3-4].
- Architecture exploration [5-6].
- Core-based system, on-chip bus and design for reuse [7-8].
- Platform-based design [2, 9-10].
- Hardware/software codesign [11-12].

Fig. 1 shows an electronic design flow including hardware/software codesign steps [13]: cosimulation, system description in a unified approach language, hardware/software partitioning and estimation.

The traditional design flow for electronic complex systems including an embedded microprocessor encouraged to perform as soon as possible the hardware/ software partition [14], so that behavior parts implemented in hardware and in software were established in the beginning of the creation process and no interaction beneficiated both parts. This methodology has been overcome by the codesign approach, which performs the hardware/software partition as late as possible in the design flow [11-12].

However, that partitioning step is based on a complete functional description of the system. That is, a behavioral arrangement of operational blocks interacting among them must be created before that partition. Designers have usually generated this functional design from the specifications in a quite straightforward way but always guided by their experience. Few criteria have been provided in order to perform the behavioral description in a systematic way.

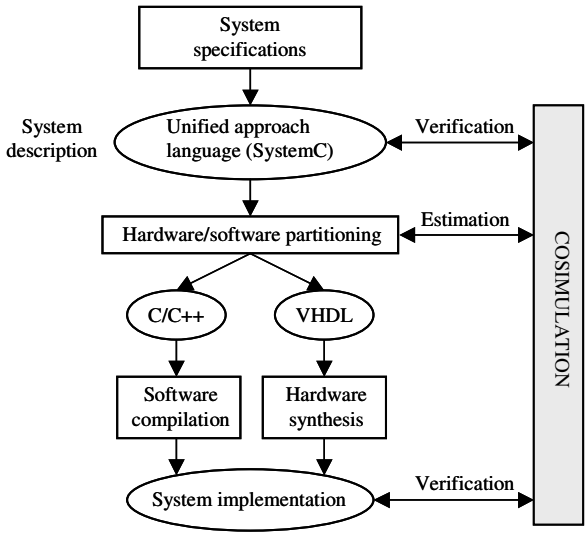

Fig. 1. Design flow for an electronic system

In addition, no guide has been reported to establish the functional blocks in the specific case of a set of devices composing an "ordered family", as it

will be presented in this paper. Such a progressive device family lets determine some first order partitions that will be useful to accomplish the functional description. For that purpose, a unified representation of the whole system behavior is used to model functional blocks, regardless of their final implementation, hardware or software [14]. In fact, several design teams have proposed the platform SystemC in order to use the C/C++ programming language as an efficient way to describe the highest level model [5, 15].

The remainder of this paper is organized as follows. Section 2 describes the Train Communication Network (TCN) and give details about the Multifunction Vehicle Bus (MVB). Section 3 explains which methodology has been followed in the design. The functional architecture to realize the bus administrator behavior, how all the modules can be interconnected and the executable model in SystemC are presented in Section 4, including some details about the software files arrangement. Section 5 is concerned with the hardware/software partitioning, the cost function and algorithm used. The results and the final partition obtained are mentioned in Section 6. Finally, some conclusions are presented in Section 7.

2 The Multifunction Vehicle Bus (MVB)

The general architecture of the Train Communication Network (TCN) includes two bus types (Fig. 2) [16]:

•MVB (Multifunction Vehicle Bus), which is used for attaching the electronic equipment inside a train vehicle.

•WTB (Wired Train Bus), which is used for interconnecting the different vehicles of a train.

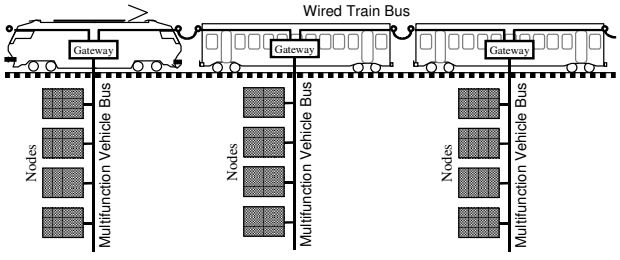

Fig. 2. Train Bus and Vehicle Bus

Both bus types have a Master-Slave architecture to control the access to the network. The Master is a device which spontaneously sends information, a Master_Frame, to a number of Slave devices. It may give a Slave the permission to transmit one Slave_Frame only within a limited time. In MVB bus, different classes of devices can be used in order to carry out various functions and services both for the vehicle and for the bus itself. Table 1 shows all the MVB device classes and the capabilities offered by each one. For instance, class 1 devices play the Slave role, receive Master and Slave Frames and, when polled, send a Slave Frame.

This table reveals that the MVB devices constitute a "progressive family", i.e., each one has all the capabilities of the previous class and, in addition, some other ones. This notion will be exploited to create the functional design of the class 4 device, the bus administrator.

Table 1. MVB device classes and corresponding capabilities

	Device_Status	Process_Data	Message_Data	User_Configurable	User_Programable	Bus_Administrator	TCN_Gateway
Class 0							
Class 1	●	●					
Class 2	●	●	●	●			
Class 3	●	●	●	●	●		
Class 4	●	●	●	●	○	●	
Class 5	●	●	●	●	○	○	●

●: Compulsory cap. ○: Optional cap.

2.1 The Master device

One bus administrator must play the Master role in the MVB network. It is in charge of periodically polling all the Slave devices in order to:

• Give permission to interchange Process Data.

• Know their Device Status and capability to be the new Master.

• Request some Message Data.

In addition, this bus administrator will inquire whether there is any unresolved event and try to transfer the Mastership to another ready bus administrator.

3 The Design Methodology

Our proposed design methodology is depicted in fig. 3. Its two main innovations are that the start point is the specifications of the whole family, instead of a particular device; and, second, a behavioral platform

for the most complex device is designed not only from functional considerations but also from the specific family framework –the "progression" notion.

This latter concept means that any device of the set offers all the capabilities of the previous less complex one in the family, plus some additional functions (fig. 4). This is the case in the MVB node classes, as table 1 shows. Such a feature is exploited in the following way: the additional capabilities of a device, not common to the simpler ones, must be assigned to functional blocks isolated from the modules that constitute the less complex devices. For instance, the functional blocks required to perform the Message data capability cannot be none of those ones used to implement the Process data, as the former is particular to class 2 devices and the latter is common to class 1, too.

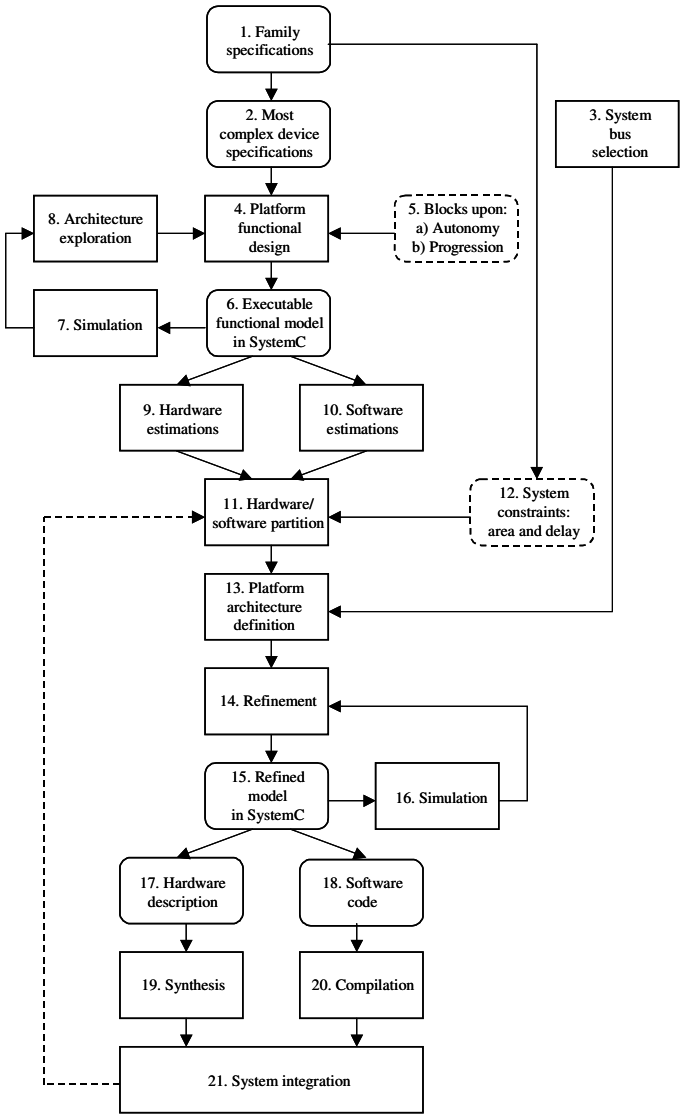

Fig. 3. Design methodology

So, an initial, basic but useful partition of the whole behavior is obtained from these family considerations. This lets the designer start the functional design not from the scratch, but from a first arrangement. In this way, our approach allows to generate a simpler device from a more complex one by just deleting the spare blocks in the functional design and repeating the subsequent steps only for the remaining ones. It must be taken into account that this last implementation process had been already performed for the complex device, so it will be straightforward, since all the functional modules will be implemented in circuits attached to the on-chip bus by means of a particular interface –the Wishbone bus, in our case. So, removing a block means disconnecting that module from the on-chip bus.

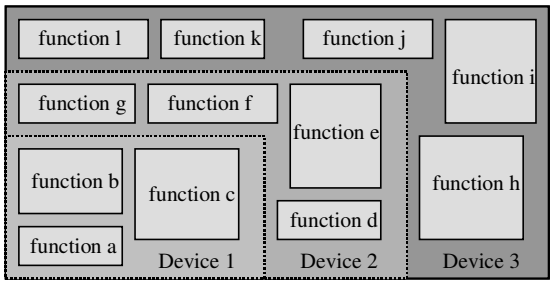

Fig. 4. Functions in a progressive family

Starting the design from the most complex device, before creating the simpler ones, is a crucial point because of the consequences on the design of the common blocks. These ones would be different if created specifically for a simpler device, so they could not be used for the more complex one. However, the reverse way is feasible indeed: a block designed for a complex device can compose a simpler one, just by fixing some inputs and ignoring some outputs, if needed. The remaining steps in the design flow of the fig. 3 are quite conventional. Nevertheless, they will be illustrated in next sections.

4. The Functional Model in SystemC

A behavioral architecture was deduced for the MVB master device, following both ideas previously introduced. In this way, all the tasks concerned with it were bound to a specific and well defined functional module. This process resulted in the blocks listed in fig. 5, where it can be seen that tasks of a capability are always independent from those ones of a simpler class –Message data and Message data polling, for example. It must be emphasized that, for the moment, 9 of these 17 blocks may be implemented in hardware or in software.

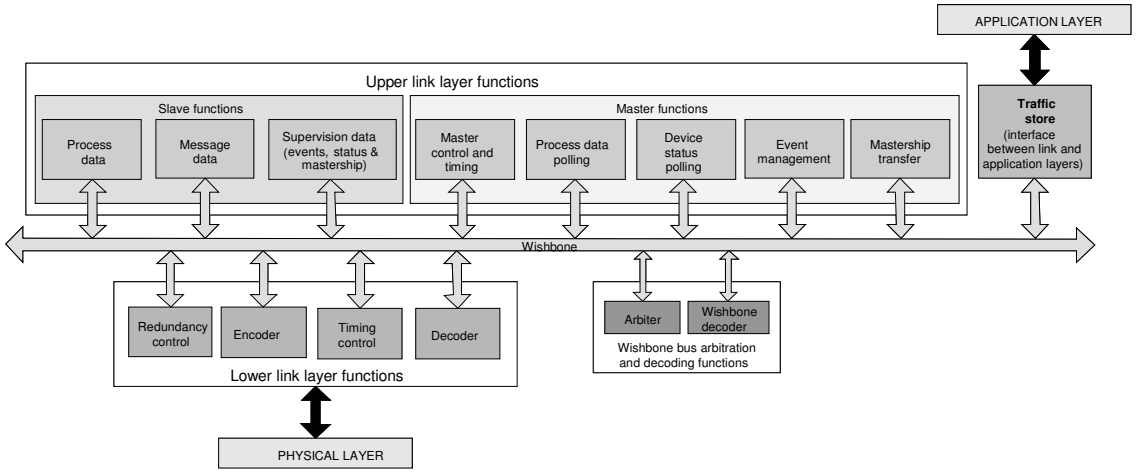

Fig. 5. Functional block diagram of the MVB link layer

The encoder, decoder, redundancy control, timing control, Master control and timing, Traffic Store, (Wishbone) arbiter and Wishbone decoder have to be implemented in hardware by nature. In section 6 the final partition between hardware and software will be detailed.

In order to cope with the great complexity of the class 4 device, a core-based design strategy was adopted [7]. So that all the functional blocks had to be provided with the particular interface to a specific on-chip bus, the Wishbone. In this way, once all the electronic modules were produced, generating the whole system would be just a matter of attaching all of them to the on-chip bus. This is a natural process to design a system-on-a-chip.

4.1 The File Arrangement of the Executable SystemC Model

Apart from the SystemC libraries, the software arrangement of all the files needed to simulate a real MVB bus scenario is represented in fig. 6. The file class_4_n.h collects all the modules indicated by fig. 5 for the n-th bus administrator. In this particular example, an MVB bus is composed of two class 4 devices and a monitor entity to display useful information about the network working. The main.cpp file collects all the previous entities and when executed by a common C++ compiler produces the results in graphical and text files. In addition, display messages are shown by the SystemC kernel.

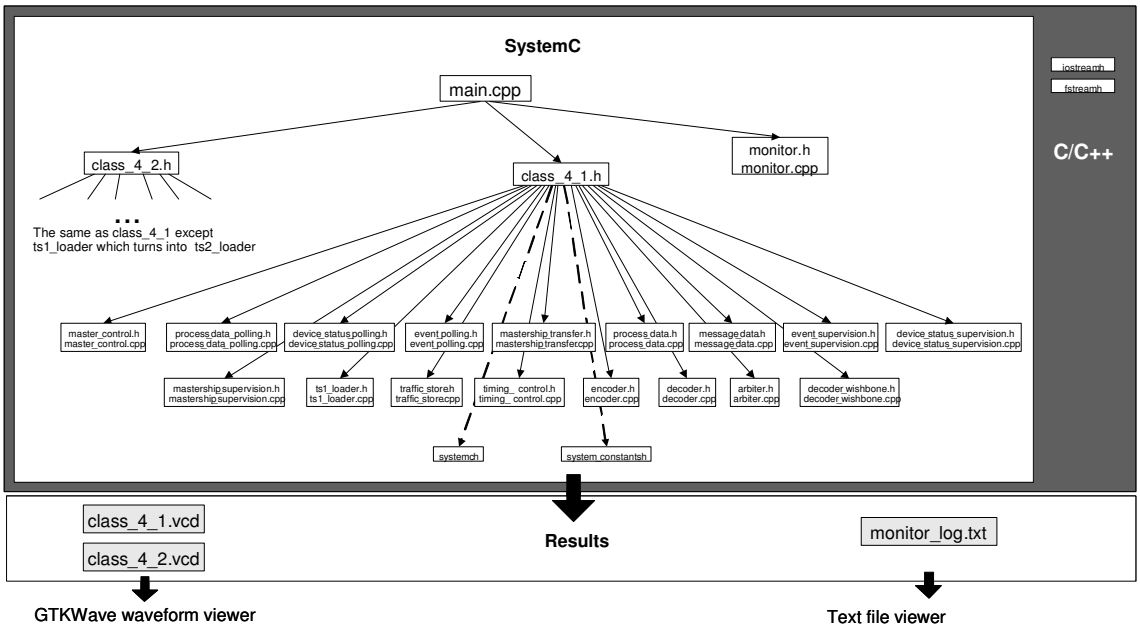

Fig. 6. File structure of the bus administrator model and test workbench

5. Hardware/Software Partition

After validating the functional design, we proceeded to the hardware/software partitioning, i.e., determine which functional blocks would be turned into specific electronic circuits and which ones into routines in one of the general purpose processors. For this object, estimations were done about silicon area and performance when implemented in hardware, and about program memory occupancy and running time when in software. From them, the following algorithm, reported by Lee, Hsiung, and Chen, was performed to partition the system [12].

First of all, an horizontal one-dimensional array named "Movable Linear Array" (MLA) (fig. 7) is used to store objects that can be implemented either as hardware or software. Each component in a system under partition is associated with a metric called Cost-Performance Difference (CPD) ratio, which takes into account both the physical resources consumed by synthesizing it and the associated processing time. The latter includes not only the operation time but also the communication one because of the on-chip bus.

A great value of the CPD ratio means that the software implementation is more effective than the hardware one and vice versa. All modules are sorted in an ascending order of their CPD ratios and placed in the MLA from left to right. The partitioning method sets the initial divider somewhere around the middle of the sorted sequence of objects in the MLA. As fig. 7 shows, all objects to the right of the divider, including itself, are implemented in software and the rest, at the left of the divider, are implemented in hardware.

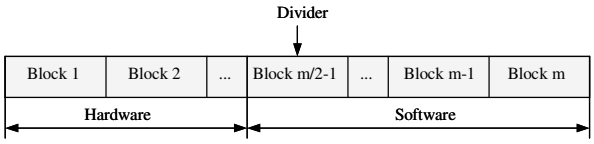

Fig. 7. Movable linear array (MLA) and divider

6. Resulting Partition

Table 2 indicates the CPD values of all the blocks in the MLA and their places in the array. These have been obtained from estimations about how many logic cells or how much program memory is needed to implement every module and about how fast they run in each case. Initially the module in the 4th place of the array was chosen as the divider, so that the 3 first functions were to be implemented in hardware and the others in software.

However, the initial partition obtained must be tested for feasibility under the given system constraints on cost and performance. Four cases are encountered during feasibility testing. First, if the performance specifications are satisfied but cost ones are not, then the software part has to be increased by selecting a new divider towards the left of the current one along the linear array of sorted objects. Second, if the cost constraints are satisfied but performance ones are not, then the hardware part must be increased by selecting a new divider towards the right of the current one along the MLA. Third, if both cost and performance specifications are satisfied, then, depending on whether preference is given to minimizing cost or to maximizing performance, we move towards the left or right, respectively. Finally, if either both cost and performance specifications are not satisfied or one of them cannot be satisfied, then no feasible partition can be found for the given system under the given constraints.

Table 2. Functional blocks ordered in the Movable Linear Array (MLA)

Module	CPD	MLA place
Process data	140	1
Message data	146	2
Event supervision	352	3
Device status supervision	378	4
Mastership supervision	389	5
Device status polling	550	6
Mastership transfer	620	7
Event management	1637	8
Process data polling	1825	9

In the case of the bus administrator, there was no specific constraint about hardware cost, as it depended upon the final selected electronic platform. Nevertheless, the standard imposes two timing constraints; first, the response time for a Slave module to send the answering Slave Frame from the moment in which the requesting Master Frame was decoded cannot be greater than 6 ms. Second, the Master cannot send a Master Frame later than 1.3 ms from the previous one.

In order to cope with these two constraints at no risk, two more iterations had to be realized from the initial partition, towards the right. In this way, the final partition showed in table 3 was established.

7 Conclusions

The MVB devices constitute an ordered or "progressive" family. This feature lets determine some first order functional partitions that will be useful to accomplish the behavioral design of the most complex device. The additional capabilities of a

device, not common to the simpler ones, must be assigned to functional blocks isolated from the modules that constitute the less complex devices.

Table 3. Hardware/software partition

Module	Implementation
Process data	HW
Message data	HW
Event supervision	HW
Device status supervision	HW
Mastership supervision	HW
Process data polling	SW
Device status polling	SW
Mastership transfer	SW
Event management	SW

HW: Hardware SW: Software

Our approach allows to generate a simpler device from a more complex one by just deleting the spare blocks in the functional design and repeating the subsequent steps only for the remaining ones. The functional design for the MVB Master device could be arranged in 14 operational blocks. So that system-on-a-chip strategies are needed to design such a complex device. A core-based architecture, modules attached to an on-chip internal bus and codesign philosophies have been followed.

The whole architecture has been coded in SystemC. After validating this executable description by simulation, the hardware/software partition has been performed following a specific algorithm. Estimations about silicon area consumed, hardware response time, program memory occupied and software execution time have been made in order to calculate a cost function for each functional block: the cost-performance difference. As a result, the electronic platform for the Master device has been generated on an FPGA. The final implementation contains a soft processor as the main component, a ROM memory, a RAM one, some internal registers and the Traffic Store.

References:

[1] R. Rajsuman, *System–on-a-chip. Design and test*, Artech House Publishers, 2000, pp. 9-12, 34-36.

[2] R. E. Bryant, "Limitations and Challenges of Computer-Aided Design Technology for CMOS VLSI", *Proceedings of the IEEE*, vol. 89, n. 3, pp. 341-363, Mar. 2001.

[3] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-level synthesis. Introduction to chip and system design*, Kluwer Academic Publishers, 1992, pp. 313-314.

[4] A. Tsuchiya, T. Shiota, and S. Kawashima, "A 0.9 V low-power 16-bit DSP based on a top-down design methodology", *Fujitsu Scientific and Technical Journal*, vol. 36, n. 1, pp. 63-71, Jun. 2000.

[5] P. Lieverse, P. Van der Wolf, K. Vissers, and E. Deprettere, "A methodology for architecture exploration of heterogeneous signal processing systems", *Journal of VLSI signal processing*, vol. 29, n. 3, pp. 197-207, 2001.

[6] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, P. van der Wolf, and E. F. Deprettere, "Exploring embedded-systems architectures with Artemis", *Computer*, pp. 57-63, Nov. 2001.

[7] R. K. Gupta and Y. Zorian, "Introducing Core-Based System Design", *IEEE Design and Test of Computers*, vol. 14, n. 4, pp. 15-25, 1997.

[8] A. M. Rincon, G. Cherichetti, J. A. Monzel, D. R. Stauffer and M.T. Trick, "Core Design and System-on-a-Chip Integration", *IEEE Design and Test of Computers*, vol. 14, n. 4, pp. 26-35, 1997.

[9] J. A. J. Leijten, J. L. Van Meerbergen, A. H. Timmer, and J. A. G. Jess, "Prophid: a platform-based design method". *Design Automation for Embedded Systems*, vol. 6, pp. 5-37, 2000.

[10] D. MacMillen, M. Butts, R. Camposanto, D. Hill, and T. W. Williams, "An industrial view of Electronic Design Automation", *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 19, n. 12, pp. 1428-1448, Dec. 2000.

[11] R. Ernst, "Codesign of embedded systems: Status and trends", *IEEE Design & Test of Computers*, vol. 15, n. 2, pp. 45-54, 1998.

[12] T. Y. Lee, P. A. Hsiung, and S. J. Chen, "DESC: a hardware-software codesign methodology for distributed embedded systems", *IEICE transactions on information and systems*, vol. E84-D, n. 3, pp. 326-339, 2001.

[13] Y. Kim, et al. "An integrated cosimulation environment for heterogeneous systems prototyping", *Design Automation for Embedded Systems*, vol. 3, pp. 163-186, 1998.

[14] S. Kumar, J. H. Aylor, B. W. Johnson, and WM. A. Wulf, *The codesign of embedded systems. A unified hardware/software representation.* S. Kumar, Ed. Kluwer Academic Publishers, 1996, pp. 2-5, 39-50.

[15] V. Carchiolo, M. Malgeri, and G. Mangioni, "Hardware/software synthesis of formal specifications in codesign of embedded systems", *ACM transactions on design automation of electronic systems*, vol. 5, n. 3, pp. 399-432, 2000.

[16] H. Kirrmann, and P. A. Zuber, "The IEC/IEEE Train Communication Network", *IEEE Micro*, vol. 21, n. 2, pp. 81-85, 87-92, Mar.-Apr. 2001.