Towards a Methodology for the Quantitative Evaluation of Automotive Architectures

Patrick Popp[+] Marco Di Natale[+] Paolo Giusto[*] Sri Kanajan[+] Claudio Pinello[*,1]
General Motors Research and Development
[+]30500 Mound Road, Warren, MI 48090-9055 [*]350 Marine Parkway, RedWood Shores, CA 94065

Abstract

Architecture design is a critical stage of the Electronics/Controls/Software (ECS) -based vehicle *design flow. Traditional approaches relying on component-level design and analysis are no longer effective as they do not always allow for the quantitative evaluation of properties arising from the composition of subsystems. This paper presents a system level architecture design methodology that is supported by tools and methods for the quantitative evaluation of key metrics of interest related to timing, dependability and cost. An example of its application to a by-wire system case study is presented, and the challenges faced in its application in the context of the actual development process are discussed.*

1 Introduction

Function development in electronics/controls/software-based (ECS) vehicle architectures has traditionally been component or sub-system focused. Each complex function is deployed to an Electronic Control Unit (ECU hereafter), which is mostly autonomous. Any time a new feature is introduced, a new ECU is added into the system, leading to the following shortcomings:

- Proliferation in the number of ECUs, subsystems and busses, making the system difficult to test and validate.
- Legacy architectural decisions constrain the new features to bandwidth and memory limitations on the serial data buses and ECUs.

In recent years, there has been a shift from the *single ECU* approach towards an increased networking of control modules within application domains (e.g. Powertrain) as well as across domains (e.g. Powertrain and Chassis). This shift has been driven by an exponential increase in the number of *horizontally* integrated complex functions (e.g. Stability Control and Adaptive Cruise Control). Today, the design and implementation of in-vehicle distributed architectures require facing new challenges such as:

- The transition to a systems engineering process that handles the vehicle as a complete, integrated system
- New methodologies and tools are required to handle the increasing interdependency of many tasks with spatial distribution and parallel execution across several ECUs and the evaluation of non-functional design requirements, such as timing, dependability, cost, time to market, extensibility over the product family lifetime and scalability across the OEM's portfolio.

Furthermore, one of the biggest challenges is the lack of information when architectural decisions are committed, which makes the process of architecture selection and design extremely susceptible to the uncertainty in the requirements.

These challenges require a methodology to assist in the process of designing, evaluating, and programming automotive architectures. The evaluation must be based on qualitative and quantitative metrics both to check requirements and to assess trade-offs while enabling *late-binding* design decisions and *early verification* of them as opposed to *early-binding* decisions with *late verification*.

2 System level methodology for quantitative architecture exploration and selection

According to the typical V-cycle development process [Beck01], a system is the result of multiple refinement stages encompassing several levels of abstraction, from user requirements, to system testing and sign-off. Within this design paradigm, the verification of *functional correctness*, most often done by simulation, is the main objective today. However, complex embedded systems are also characterized by *non-functional requirements*, such as timing behavior, which includes the evaluation of latencies and jitter, and requirements for safety that may exceed even the stringent constraints of the aeronautics industry, currently estimated at a required failure rate of less than 10^{-10} failures/hour [Rus01]. Finally, a major non-functional metric is cost, and the related secondary metrics including reusability, flexibility, scalability and extensibility of the architecture artifacts.

The evaluation of architecture solutions is performed in a quantifiable manner against a set of constraints and metrics functions, classified according to a general taxonomy that identifies three main domains, namely ***timing***, ***dependability*** and ***cost***. The secondary requirements, together with a short description of their meaning and the associated metrics are summarized in Table 1.

The selection and the quantitative definition of metrics and constraints is a challenging task by itself. The identification of the main domains related to timing, dependability and cost is common to other architecture evaluation methodologies, such as the ATAM at the Carnegie Mellon SEI [Kaz00]. Other domains, including

[1] Claudio Pinello is now with Cadence Berkeley Labs.

978-3-9810801-2-4/DATE07 © 2007 EDAA

Primary	*Secondary*	*What is captured*	*Metrics*
Timing	End-to-end latency	measuring the time distance between two events (related to stability and performance)	*Milliseconds*
	Jitter	maximum delay of a periodic signal with respect to ideal reference	*Milliseconds, or % of period*
	Input coherency	time distance between two events/samples from multiple sensors observing the same object/phenomenon	*Milliseconds*
Dependability	Reliability	expectation on failure, related to warranty cost impact	Expected time between failures MTTF or fault rate (number of faults per hour)
	Availability	Percentage of uptime	MTTF/(MTTF+MTTR)
	Safety	which faults can be tolerated and which cannot. Related to fault tolerance, fail safe vs fail operational	number of components/cutset that must fail for the system to fail
Cost	Piece cost		$
	Extensibility	room for functional additions (e.g. Complement to resource utilization)	fraction of resource utilization available for future use
	Degree of Reuse	ability to design/deploy using preexisting solutions, (SW or HW components, schedules and configurations)	number of units deployed
	Scalability	suitability for a range of content level (while cost-effective)	number of products or product lines

Table 1: Definition of Primary and Secondary Metrics

energy requirements are of course relevant but are not targeted by our analysis. Furthermore, a standard definition of concepts like extensibility, reuse or scalability is still lacking and our definitions of the metrics definitions are the result of an ongoing process, far from being completed.

For timing related metrics and use cases (end to end latency verification, bus/cpu utilization, etc.), we propose the use of schedulability analysis theory and system-level simulation to provide formal evaluation of the *timing behavior at the highest possible level in the design flow* and to estimate extensibility by providing a measure of the available processor and communication time for new functions and messages in product derivatives.

Architecture options are also scored according to a *quantitative evaluation of reliability based on fault tree analysis*. Monetary cost is evaluated based on the architecture's intended product line and life cycle. These analytical methods must work in the presence of incomplete information, given that common automotive flows require the selection of the physical architecture at very early stages.

Collectively, these non-functional properties cannot be assessed based on an abstract model of the system functions alone, but they depend upon the computation platform and the implementation of the function on the underlying execution architecture, including the topology of the ECUs, the physical communication links and their scheduling or access control policies.

2.1 The Platform-based Design Methodology

The match between function and architecture is a key aspect of the design of embedded systems and the founding principle of many design methodologies such as the platform-based design [Vin02] and the Ptolemy and Metropolis frameworks [Bal03], as well as of emerging standards and recommendations, such as the UML Profile for Schedulability, Performance and Time from the Object Management Group [OMG02] and AUTOSAR.

We advocate the use of the conceptual framework of the ***Platform-based design*** methodology and the *meet-in-the-middle* approach as key enablers for the exploration of design alternatives and architecture level solutions. Platform-based design requires/entices the identification of clear abstraction layers and a design interface that allows for the separation of concerns between the refinement of the functional architecture specification and the abstractions of possible implementations. The application-layer software components are thus decoupled from changes in microcontroller hardware, ECU hardware, I/O devices, sensors, actuators, and communication links.

The basic idea is captured in Figure 1. The vertex of the two cones represents the combination of the functional model and the architecture platform. Decoupling the application-layer logic from dependencies on infrastructure-layer hardware or software enables the application-layer components to be reused without changes across multiple vehicle programs.

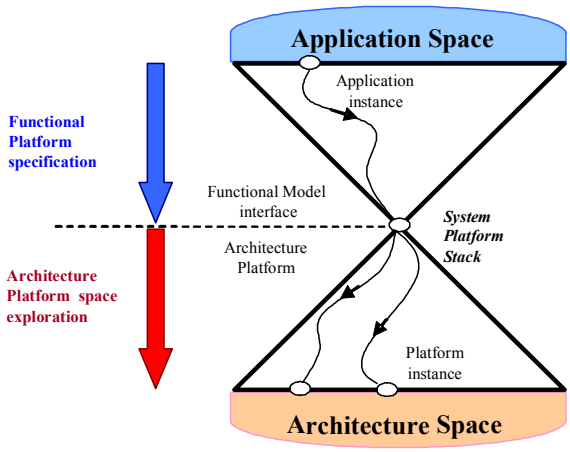

Figure 1: Platform-based design ([Vin02])

A prerequisite for the adoption of the platform-based design and of the meet-in-the middle approach is the definition of the right models and abstractions for the description of the functional platform specification and for the architecture solutions at the top and the bottom of the hourglass of Figure 1. The platform interface must be isolated from lower-level details but at the same time, it must provide enough information to allow design space exploration with a fairly accurate prediction of the properties of the final implementation. This model may include size, reliability, power consumption and timing; variables that are associated to the lower level abstraction (from the implementation platform). On the other hand, we pass constraints from higher levels of abstraction down to lower levels to satisfy the original design constraints.

Design space exploration consists of seeking the "optimal" mapping of the system platform model into the candidate execution platform instances. The mapping must be driven by a set of methods and tools providing an objective and quantitative measure of the fitness of the architecture solutions with respect to a set of feasibility constraints and optimization metric functions including those defined in Table 1.

Ideally, there could be the possibility for the automatic selection of the platform by software tools. In reality, the technology is not mature for a full synthesis of the mapping and the platform attributes and the approach that is currently viable is a what-if analysis where different options are selected as representatives of the principal platform options and evaluated according to measurable metrics.

The evaluation and selection of an architecture design requires as input a complex set of models defining the functions of the car electronic systems, that is, the application instance at the top of the hourglass of Figure 1. Similarly, a model of the available execution platform instances (at the bottom of the hourglass) is required.

2.2 Functional Models

The starting point for the definition of ECS based vehicle architecture is the specification of the set of features that the system is expected to provide. A feature is a very high level description of a system capability. The subsystem that determines the feedback force on the brake pedal of a brake-by-wire system is an example consisting of mechanical, electronic, and software parts that together emulate key aspects of the feel of the conventional hydraulic brake pedal.

The software component of each feature is further developed by control engineers who devise control algorithms fulfilling the design goals. Typically, these algorithms are captured by a hierarchical set of block diagrams produced with commercial tools for control algorithm design.

The *functional model*(s) are created from the decomposition of the feature in a hierarchical network of components encapsulating a behavior, within a provided and required interface, expressed by a set of ports or by a set of methods with the corresponding signature. This view abstracts from the details of the functional behavior and models only the interface and the communication semantics, including the specification of the activation signal for each functional block, be it a periodic activation signal, or an activation signal arriving, together with the incoming data, from one of its input ports, as the result of the computation of a predecessor block.

A function label f_i is associated to each block, which computes a set of output values o_i based on a set of inputs i_i and possibly its internal state S_i at some given time, that is, $o_i = F_i(i_i, S_i)$. Each activation instant triggers a function instance $f_{i,k}$, which conceptually executes in zero time (at this level, the design abstraction is independent from resource availability).

The functional description is further endowed with the constraints that are required. For example, timing constraints are expressed in the context of the functional architecture by adding *end-to-end deadlines* to the computation paths, *maximum jitter* requirements to any signal and *time correlation* constraints between any signal pair originating from the same functional block or providing input to a common block.

To give an example of the implications of a choice of an activation/communication model, the data communication between any two blocks activated periodically according to local, non synchronized clocks, is assumed to be *nondeterministic* in time and lossy, meaning that output values may be overwritten before having been read.

2.3 Architecture models

The model of the architecture is hierarchical and captures the *logical* topology of the car network, including the communication busses, such as CAN [CAN91] and time-triggered links, the number of processors for each ECU and the resource management policies that control the allocation of each ECU and BUS, and also the physical and geometric relationships, including abstractions for modeling wiring harnesses and connectors. At this stage, the hardware and software resources that are available for the execution of the application tasks and the resource allocation and scheduling policies must also be specified. Each RTOS provides a set of services and logical resources and has a set of parameters related to the provided scheduling policy for the ECUs. The definition of the MAC layer and the scheduling policy of the physical communication links must also be known.

2.4 System platform model and mapping

If specification of functionality aims at producing a logically correct representation of system behavior, the system platform model is where physical concurrency and resource requirements are expressed.

The system platform model(s) are a representation of the mapping process and can be of different types for different

analysis purposes, hiding unnecessary details and exporting only the necessary amount of information.

At this level, we define tasks as units of computation processed concurrently in response to environment stimuli or prompted by an internal clock. Tasks cooperate by exchanging ***messages*** and ***synchronization*** or ***activation signals*** and contend for use of the processing and communication resource(s) (e.g., processors and buses) as well as for the other resources in the system. The system platform model entities must, on one hand, be the implementation of the functional model entities and are, on the other hand, mapped onto the target hardware.

The mapping phase consists of allocating each functional block to a software task and each communication signal variable to a virtual communication object. The task activation rates must be entered as parameters of the architectural models and compliance checks are performed with the functional blocks activation rates. If more than one functional block is mapped to a task, the order of the execution must be provided during the mapping phase. The mapping of the threads and message model into the corresponding architecture model and the selection of resource management policies allows the subsequent validation against non-functional constraints.

As a result of the mapping of the platform model into the execution architecture, the entities in the functional models are put in relation with timing execution information derived by worst case execution time analysis or back-annotations extracted from physical or virtual implementation.

Given a mapping, it is possible to determine which signals are local (because the source and destination functions are deployed onto the same ECU) and which are remote, hence need to go over the network. Each communication signal is therefore mapped to a message, or to a task private variable or to a protected shared variable. Each message, in turn, is mapped to a serial data link, and the relation can be extended by mapping serial data links to harnesses, and harnesses to physical places in the car.

Conceptually, the mapping results in a restriction of the possible behaviors of the functional model after the intersection with the set of all the behaviors that are possibly allowed by the platform implementation. Therefore, not all the mappings are allowed or should be made legal. For example, a nondeterministic communication among two functional blocks can be made deterministic, and a global execution order for all the functional blocks can be defined, after mapping them into the task set, in accordance with the partial order defined by the functional model semantics.

3 A Data model for architecture exploration

Architecture exploration by definition of platform models and platform mapping can be considerably easier if the models of the system at the different abstraction levels are homogeneous.

The Architecture Exploration Tools and Methods (AETM) Data Model defines the design artifacts needed for architecture exploration and the relationships among them. It is a key enabler for an integrated tool framework aimed at supporting the concept of a virtual integration platform.

In order to favor the flexible mapping/re-mapping capabilities for fast creation of architecture alternatives and the re-use of components across different design alternatives, the data model enables the separate design capture of different abstraction layers that constitute a design: functional layer (e.g. signals and functions), software layer (e.g. software tasks and scheduling), serial data link layer (e.g. network bus scheduling), and physical layer.

The AETM design model is formally defined and represented by an XML schema, which controls the format of all files exchanged by the toolset, defines the elements of the functional and architecture level design, the mapping relationships, the annotations adding timing attributes to the design objects and the schedulability-related information. It currently allows the expression of the structural properties of the design models; but a formal definition of the semantics is still lacking and the interpretation of the model is performed by the analysis tools.

4 Quantitative what-if analysis

The procedure for architecture selection and evaluation is a what-if iterative process. First, the set of metrics and constraints that apply to the design is defined. Then, based on the designer's experience, a set of initial candidate architecture configurations is produced. These architectures are evaluated based on the methods and tools presented in the following sections. The architecture options are scored and, based on the results of quantitative analysis, a final solution can be extracted from the set as the best fit or a new set of candidate architectures, possibly, but not necessarily, produced by incremental modifications on the previously considered ones, can be selected as the new possible architecture options. The iterative process continues, until a solution is obtained.

The intervention of the designer is required in two tightly related stages of the exploration cycle. Given the set of metrics and constraints and the use cases, the designer must provide the initial set of architecture options. After the options have been scored and annotated by the analysis and simulation tools, the designer must understand the results of the analysis and select the architecture options that are the best fit to the exploration goals and (more importantly) understand the results of the analysis to add other options to the next set of architecture configurations that needs to be evaluated.

Several iterations between mapping and analysis might be performed before the final design decision. The set of analysis and synthesis methods that are currently available include:

Analysis methods

- Evaluation of end-to-end latency and schedulability against deadlines for chains of computations spanning tasks and messages scheduled with fixed priority.
- Sensitivity analysis for tasks and messages scheduled with fixed priorities and sensitivity analysis for resources scheduled with fixed priorities.
- Evaluation of message latencies in CAN bus networks.
- System level simulation of time properties and functional behaviors (based on the Metropolis engine).
- Analysis of fault probability and cutsets (conditions leading to critical faults) based on fault trees.
- Product line cost analysis.

Synthesis methods

- Automatic generation of fault trees.
- Synthesis of task and message schedules in time triggered systems - tasks are scheduled according to the OSEKTime paradigm, and message schedules are generated for Flexray networks [Fle06].
- Synthesis of the activation model for tasks and messages for minimizing end-to-end latencies with respect to the requested deadlines.
- Fault tolerance driven scheduling.

4.1 Tools framework

The tools that are currently in use for the evaluation are the following.

For *timing* we tested the use the MAST [Gon01] tools for the evaluation of the worst case response times of the tasks [Gon94] and a custom procedure implementing the analysis of the message latencies in the CAN bus according to the analysis in [Tin95] and on the refinement that takes into account the non-preemptability of the TxObjects at the bus adapter. Task and message response times are then used by an additional layer of in-house developed code implementing the computation of the worst case end-to-end latencies.

For *dependability* analysis and the synthesis of dependable architecture solutions, we tested the use of Fault Tree analysis tools and of two other tools that have been developed as the result of research work at the University of California at Berkeley, namely the SCRAPE tool and the Fault Tree Generator program.

Finally, in-house developed programs are used for the analysis of the product line *cost* of architecture solutions.

5 Case study

The architecture exploration methodology described in this paper was applied to a number of case studies, including integrated active and passive safety systems, and stability control systems. In this section, we report on the analysis of five possible options for a steer- by-wire architecture.

The baseline architecture, considered as a starting point for the analysis of the case, consists of redundant steering motors, four supervisory ECUs (S-ECU) running the control algorithms and reading/driving the interface sensors (three redundant steering angle and steering torque sensors) and actuators, and peripheral ECUs (P-ECU) controlling the steering motors and reading the sensors. The supervisory ECUs are connected by a Flexray bus and each of them has a CAN link to a motor control unit (Figure 2). The goal of the case study was to verify the capability of modeling the system and to evaluate a set of possible execution platforms according to the following metrics:

- *Utilization*: the amount of computation per unit time. Processor utilization and bus bandwidth are the two key metrics.
- *Composability*: the ability to integrate components together without loss of the original properties.
- *Reusability/Cost*: what parts of the architecture can be made common in order to leverage economies of scale, and better unit/costs with suppliers.
- *Dependability*: The degree of reliability of the architecture based on a given top event, representing a fault in the system
- *Modifiability/Scalability*: the ability to extend or modify the current architecture in terms of functionality or execution platform without causing a ripple of changes.

For each of them a range of scores between 0 and 8 was defined. For example, for the dependability metrics, the architectures were scored according to the following rules:

- 1-2 points for meeting baseline fault hypothesis assuming only permanent faults and achieving the baseline system failure rate.
- 2-4 points for meeting the baseline fault hypothesis assuming permanent, transient type fault scenarios and achieving the baseline system failure rate.
- 4-6 points for meeting the fault hypothesis requirements given an arbitrary failure mode and achieving the baseline system failure rate.
- 6-8 points for exceeding the fault hypothesis requirements given an arbitrary failure mode and achieving a significantly better system failure rate than the baseline.

This ranking is an attempt at reducing the complexity of the metrics and allowing an easy visualization of the tradeoffs of the different architecture options with respect to the multiple domains of time, dependability and cost.

Five possible options for the physical architecture have been considered.

Alternative 1 The baseline architecture.

Alternative 2 Obtained by dropping the four CAN busses connecting the four P-ECUs to the S-ECUs, and by connecting them using the FlexRay backbone.

Alternative 3 Obtained by removing one of the S-ECU units (motivated by the results of the analysis showing that the CPU utilization is very low).

Alternative 4 Obtained by substituting the Flexray bus with a triple redundant CAN system (motivated by the results showing that the FlexRay utilization is very low).

Alternative 5 To reduce the component count, in this alternative we dropped the four P-MCUs and connected the sensors and actuators directly to the three S-ECUs.

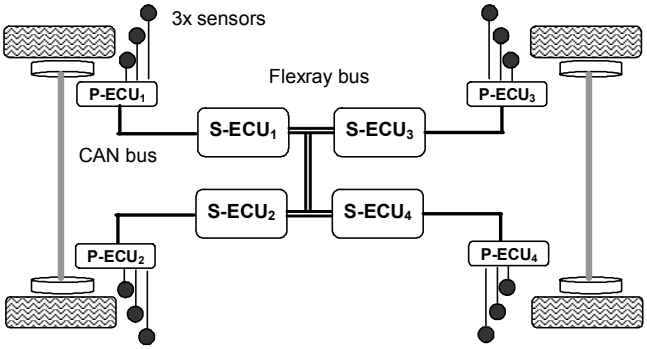

Figure 2: Baseline architecture of the case study

We used the tools and methods described in [Pin04] and [McK05] to perform the schedule synthesis, the timing analysis, and the dependability assessment. Furthermore, a qualitative assessment of cost was provided. The results of the evaluation were presented in a combined way (Figure 3).

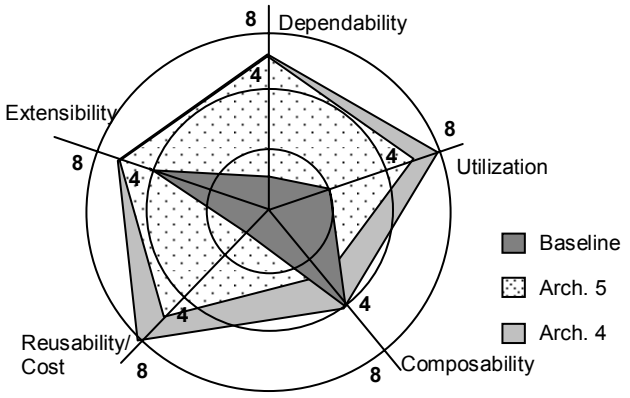

Figure 3: Evaluation of multiple metrics for the architecture alternatives.

The case study demonstrates the potential for the methodology to enable a more through exploration of the architecture space, thus improving the overall quality of the final result. This work is focused on indicating the value of a systematic, quantitative based architecture exploration methodology. Further development is required to improve the quality of the input data and the accuracy of our metric estimates. Furthermore, more testing is required before the entire methodology and the case study results can be applied into product development.

6 Challenges and Conclusions

Automotive in-vehicle ECS architectures are increasingly networked and continuously subject to change. A rigorous design methodology based on the separation of concerns is essential to evaluate architecture configurations with the goal of managing the complexity, improving design quality and reducing the time to market. This paper presents a design methodology, together with the definition of metrics for timing, dependability and cost and appropriate methods and tools, to support the architecture exploration in a quantitative manner. Main challenges remain because hardware, software and business data at the early-stages of the development cycle often do not exist or lack of the required accuracy. Sensitivity analysis can be used to some degree to cope with this uncertainty.

We would like to thank Max Chiodo, Tom Fuhrman and the ECI group, and professor Alberto Sangiovanni Vincentelli from UC Berkeley for useful discussions and feedback. Also, many thanks to Haibo Zeng, Abhijit Davare, Sampada Sonalkar and Mark McKelvin for their contributions.

7 Bibliography

[Bal03] Balarin F., Hsieh H., Lavagno L., Passerone C., Sangiovanni-Vincentelli A. and Watanabe Y., Metropolis: An Integrated Environment for Electronic System Design, IEEE Computer, April 2003.

[Bec01] Beck T., Current Trends in the Design of Automotive Electronic Systems, DATE Conference, 2001.

[CAN91] R. Bosch. CAN specification, version 2.0. Stuttgart, 1991.

[Fle06] Flexray Standard Specification available from http://www.flexray.com

[Gon01] M. Gonzalez Harbour et al., MAST: Modeling and Analysis Suite for Real Time Applications, ECRTS 2001.

[Kaz00] R. Kazman, M. Klein, P. Clements, ATAM: Method for Architecture Evaluation, Tech. Report CMU/SEI-2000-TR-004, Aug. 2000

[McK05] M. L. McKelvin Jr., G. Eirea, C. Pinello, S. Kanajan, A. Sangiovanni-Vincentelli, A Formal Approach to Fault Tree Synthesis for the Analysis of Distributed Fault Tolerant Systems, Emsoft, Jersey City, NJ, September 2005.

[OMG02] UML Profile for Schedulability, Performance and Time. OMG Adopted Specification, July, 1, 2002.

[Pin04] C. Pinello, L. Carloni, A. Sangiovanni-Vincentelli, Fault-Tolerant Deployment of Embedded Software for Cost-Sensitive Real-Time Feedback-Control Applications, DATE conference, Paris February 2004.

[Rus01] J. Rushby, A Comparison of Bus Architectures for Safety-Critical Embedded Systems, CSL Technical Report, SRI International, September 2001.

[Tin95] K. Tindell, A. Burns, and A. J. Wellings, Calculating controller area network (can) message response times, Control Eng. Practice, v. 3 n. 8, pp. 1163--1169, 1995.

[Vin02] Sangiovanni Vincentelli A. Defining Platform-based Design. EEDesign of EETimes, February 2002.

[Gon94] M. G. Harbour, M. Klein, and J. Lehoczky, `Timing analysis for fixed-priority scheduling of hard real-time systems, IEEE Transactions on Software Engineering, vol. 20, no. 1, January 1994.