Automatic Code Generation and Platform Based Design Methodology: An Engine Management System Design Case Study

Alberto Ferrari
PARADES EEIG

Giovanni Gaviani, Giacomo Gentile, Monti Stefano and Luigi Romagnoli
Magneti Marelli Powertrain

Michael Beine
dSPACE

Reprinted From: Software/Hardware Systems (SP-1924)
Automatic Code Generation and Platform Based Design Methodology: An Engine Management System Design Case Study

Alberto Ferrari
PARADES EEIG

Giovanni Gaviani, Giacomo Gentile, Monti Stefano and Luigi Romagnoli
Magneti Marelli Powertrain

Michael Beine
dSPACE

Copyright © 2005 SAE International

ABSTRACT

The design of a complex real-time embedded system requires the specification of its functionality, the design of the hardware and software architectures, the implementation of hardware and software components and finally the system validation. The designer, starting from the specification, refines the solution trying to minimize the system cost while satisfying functional and non functional requirements. The automatic code generation from models and the introduction of the platform-based design methodology can drastically improve the design efficiency of the software partition, while maintaining acceptable the cost overhead of the final system. In this approach, both top-down and bottom-up aspects are considered and solutions are found by a meet-in-the-middle approach that couples model refinement and platform modeling. In more details, given a model of the implementation platform, which describes the available services and data types, the algorithms captured by models are refined and then automatically translated to software components. These components are integrated with handwritten (e.g. legacy) software modules together with the software platform. A final validation phase on the real target is performed to finally validate the functionality and to guarantee that the performance constraints are met.

In the future, the platform based design methodology will allow an easy accommodation of the new automotive software architecture standard promoted by the AUTOSAR consortium.

INTRODUCTION

The design of an engine management system is a very challenging problem in automotive electronics because of the complexity of the functions to be implemented and the real-time and cost constraints. In brief, an engine management system controls a combustion engine, or more recently a hybrid engine (i.e. combustion engine coupled with an electrical motor), to offer appropriate driving performance (e.g. drivability, comfort, and safety) while minimizing fuel consumption and pollutant emissions. The behavior of the controlled system is achieved by actuating several control inputs, such as throttle position, fuel injection and spark ignition that are tightly synchronized with the rotation of mechanical components (e.g. crankshaft and camshaft). Hence, several control algorithms must be designed to correctly control those inputs. Recently, the implementation of these algorithms has been migrated from hardware to software due to the always shrinking time-to-market, the continuously changing specifications and the high hardware implementation cost. Hence, several control algorithms must be executed by one or more computing platforms with hard real-time constraints.

The complexity of the nowadays control algorithms and the tight design dependency between the plant (engine), provided by the car maker, the electrical control unit (ECU), provided by the sub-system maker, and the hardware and software components (CPU, DSP and
RTOS) are such that there is the need of an integrated design chain and of a common standardized platform to fully explore the design space and better utilize the available technologies. The integrated design chain should be supported by a common design methodology and design flow, with a shared understanding of specifications and implementation constraints between all the actors of the design.

In the automotive domain, there are initiatives mainly focused on common standardized computing platforms. In Europe the AUTOSAR (15) initiative is aimed to define a common software platform to enable a sharing of software components and an easy software integration. Similar initiative has been taken in Japan by the automotive industry with the JASPAR initiative. In our understanding these initiatives will provide an important breakthrough in the design of automotive controllers and coupled with an integrated design chain will provide the backbone for the design and implementation of the next generation engine management systems.

In this paper we present the model-based design methodology that has been introduced in Magneti Marelli Powertrain and a gasoline direct injection case study (Ferrari, et al., 2004). The design methodology is based on a meet-in-the-middle (Vincentelli and Ferrari, 1999) approach with a defined set of abstraction layers and successive refinements (Balluchi, et al., 2002).

The requirements of the highest level of abstraction, the system, are expressed in terms of functionality and performance indexes. These requirements are captured by executable models and are shared between car makers and sub system makers, drastically reducing ambiguity, i.e. possible interpretation errors. These models are then refined down to implementation.

The design of control algorithms is a fundamental part of the design flow. It starts from a functional specification and ends up with a detailed description of the algorithms. In the model-based design methodology, the part of the control algorithm that is mapped to the software partition is automatically translated from a model representation to a set of software components. The software architecture of the application will accommodate and compose together those software components such that the real-time requirements are met. In the proposed design flow, the control algorithms are captured using the MATLAB/Simulink (The Mathworks, 2002) design environment and the automatic translation of the model to C-language code is performed with the production code generator TargetLink (dSPACE, 2002), in the sequel called model compiler and described in one of the following sections.

To correctly accommodate automatically and manually generated software components, a software architecture composed of different layers has been totally specified and implemented. The layer closest to the hardware is the basic input output system (BIOS). The upper layer is containing the device drivers that encapsulate electrical drivers of sensors and actuators. The RTOS and communication services are common to all layers. All these layers implement the software platform that supports the application software. The latter has structural and semantic aspects that allow the correct integration of the software components implementing the entire set of control algorithms.

To obtain the final implementation of the power-train controller, several players belonging to different organizations within a company and/or different companies have to cooperate during the design of each component and of the entire system: car makers must provide and share controller specifications, plant models and calibration sets, silicon suppliers must provide performance models of the micro-controllers. A common design methodology and tool chain is the key of success in coping with the design complexity and constraints.

The paper is organized as follows: the first part describes the methodology framework defined by Magneti Marelli Powertrain. The second part describes the TargetLink environment by dSPACE. The third part is dedicated to the description of a real design case. The fourth one indicates future work and conclusions.

**DESIGN METHODOLOGY**

The basic tenets of the Platform-based Design Methodology as exposed in (Vincentelli and Ferrari, 1999) are:

- Regarding design as a “meeting-in-the-middle process” where successive refinements of specifications meet with abstractions of potential implementations;
- The identification of precisely defined layers where the refinement and abstraction process take place.

The layers then support designs built upon them isolating from lower-level details but letting enough information transpire about lower levels of abstraction to allow design space exploration with a fairly accurate prediction of the properties of the final implementation. The information should be incorporated in appropriate parameters that annotate design choices at the present layer of abstraction. These layers of abstraction are called Platforms. In this paper, a platform is defined to be an abstraction layer in the design flow that facilitates a number of possible refinements into a subsequent abstraction layer (platform) in the design flow. The abstraction layer contains several possible design solutions but limits the design exploration space. During the design process, at every step we choose a platform instance in the platform space. Every pair of platforms, the tools and methods that are used to map the upper layer of abstraction into the lower level one is a platform stack. Key to the application of the design principle is the careful definition of the platform layers. Platforms can be defined at several points of the design process. Some
levels of abstractions are more important than others in the overall design trade-off space. In particular, the articulation point between system definition and implementation is a critical one for design quality and time.

In the proposed approach, five main levels of abstraction are identified: system level, function level, operation level, architecture level, and component level (Balluchi et al., 2002). Figure 1 shows the design methodology and the five level of abstraction. In each single design step (Figure 2), the platform is abstracted and captured by a platform description (platform model). The function requirements at each level of abstraction are captured (functional model) and mapped to the selected platform to analyse performances and verify the design step. A synthesis step is then performed to generate the particular instance of the components of the platforms.

![Figure 1 Platform-based design methodology for automotive](image)

The five levels of the platform stack are described in the sequel.

System: car manufactures define the specifications of power-train control systems in terms of desired performances of the vehicle in response to driver’s commands. Additional requested specifications, defined by governments or car manufacturers associations, are concerned with fuel consumption, noise and tail pipe emissions. At the system level, the given specifications are analyzed and expressed in an analytical formalism. Specifications have to be clearly stated and negotiated between customer and supplier to make sure that they are realizable within the budget and time allowed for completing the design.

Functions: the design of the functionality to be realized by the control system to meet the system specifications described above is very complex. A good quality of the design is obtained by decomposing the system into interacting sub-systems, referred to as functions. The decomposition allows designers to address the complexity if it leads to a design process that can be carried out as independently as possible for each component. The structure of the functions is the model of the platform at this level of abstraction. System specifications are spread out among the functional components so that the composition of the behaviors of the components is guaranteed to meet the requested objectives and constraints. The output of the functional level design is a desired behavior for each function.

Operation: at the operation level, the desired behaviors have to be obtained, satisfying also some local objectives and constraints. Solutions are expressed in terms of basic building blocks, called operations. In a first design attempt, for each function, control strategies achieving the given specifications are devised and captured with executive models. The control strategies operate on variables that are measured on the physical domain and produce values of variables that act on the physical domain. Then, each control strategy is refined by introducing chains of elementary operations, so that the set of all solutions can be integrated in a unique operations network.

Architecture and Electronic System Mapping: the design step at the architectural level produces a mapping between the behavior that the system must realize (operations) and the platform representing the chosen system architecture, i.e., an interconnection of mechanical and electrical components (e.g., sensors, actuators, microprocessors and ASICs). The set of components either are available in a library of existing parts or must be designed ex novo. This architecture and component-selection task is the subject of intense research by the system design community.

![Figure 2 Platform based design methodology](image)

The integration of the components and its validation is performed as done in the classical V cycle design methodology (with usage of models for virtual plants).

At the end of the integration process, the control algorithms are tuned adjusting the control parameters (i.e. model parameters) and final validation of the plant assumptions is carried out. This task is performed in collaboration with car makers and provides back to the control designers fundamental information of the actual
behavior of the system and assumptions made during the design.

In the past, the correctness of the algorithms and of the final implementation was validated by prototypes of the target ECU employed during the entire design. The algorithms were frequently described only on documents and C language and the physical validation was detecting, algorithms, coding and architectural errors. This is fundamentally different from the current methodology. In this case the correctness of the final implementation is strictly connected to the correctness of the control algorithms (operations), captured by models from which the implementation is automatically derived. The use of prototype ECUs is limited to the algorithm exploration phase to validate the assumption on the plant and the correctness of the control algorithms. Prototypes are not used anymore to explore the control algorithm and to validate the algorithm implementation. Moreover, since in the past the algorithm exploration and coding validation were carried out mainly at the software level, the final specification of the control algorithm was known only at C level, stability and other analyses were not carried out and the final solution was very difficult to be reused. Instead, in the proposed model-based design methodology, validation starts as soon as the designer conceives the controllers with the use of complex models of the plant, describing the engine, driveline and the driver, or in a simpler way set of recorded input/output traces. If a control algorithm is subject to changes, the model is first modified and then the new software is generated, resulting in a natural synchronization between the model representation of the control algorithm and its implementation. This methodology shift drastically reduces the use of prototypes of target ECUs, resulting in a strong reduction of design time and cost which we can account for around 30-40% in our case study.

MODEL REFINEMENT AND SOFTWARE PLATFORM

To support the methodology, the tool chain must handle the refinement of components from one level of abstraction to another one. Ideally, it should be possible to support, in the same design framework, the refinement of system requirements to control algorithms, and then to software or hardware components. If we consider data refinements, system specification and control algorithms might be provided in floating point notation of quantities, e.g. the quantity of fuel injected in the cylinder, while at the software level this data might be refined to 16 bits fixed point representation.

In the design process, different actors (working even in different companies) will interpret the data in different ways: as floating point or fixed point notation. For example, car maker will provide requirements in floating point notation and will perform calibrations and measurements in a coherent manner, but at the implementation level the power-train controller executes operation only in fixed point data, hence software engineers have to manage software in fixed point notation.

Not a single tool today in the market easily supports refinement in a general meaning. The more advance research project on this topic is the Metropolis project at the University of Berkeley, see (Burch, et al., 2002) and (The Metropolis Project).

In our approach, the algorithm specifications are captured in Simulink and are refined with different models, expressing different level of details, consistently linked by a configuration management system. The verification of the refinement is obtained only via functional simulation.

TARGETLINK: FROM MODEL TO C CODE

The translation of Simulink/Stateflow models to C code is performed by the production code generator TargetLink from dSPACE (Hanselmann, et al. 1999). This model compiler creates the software modules (component level) from refined algorithms as described in the previous sections. Code efficiency is one of the primary requirements for a model compiler. Another important necessity is process efficiency and flexibility. This term refers to the capability of the tool to support and adjust to an existing development process and how safe it can be used within that process. The fulfillment of these and several other requirements make TargetLink a tool that is highly integrateable within the environment of model-based design methodology.

USER-GUIDED MODEL REFINEMENT

As outlined in the previous section, the center focus of working with a model compiler is model refinement. The Simulink and Stateflow models represent the complete functional specification of an algorithm. This specification has to be prepared for code generation. This basically means that data for implementation have to be added to each block and each subsystem of the model specification.

If micro-controllers with fixed-point arithmetic are being used, then variable scaling is the inevitable first step of the refinement process. TargetLink supports the user by providing comprehensive scaling options including automatic scaling support. Automatic scaling helps the user to quickly move from floating point to a fixed point implementation which can then be further refined throughout the process. The user can select one of two different approaches: Automatic scaling calculation based on signal ranges recorded during simulation or scaling derived from value ranges computed based on worst-case assumptions, also referred to as worst-case auto scaling (TargetLink Production Code Generation Guide).
Further data for variable definitions and declarations have to be specified, such as names, storage classes, scope and other attributes. Function and task partitioning takes place, and some model optimizations can be carried out to support efficient code generation. The designer has different options for entering the code generation related data. One of these is based on a data dictionary: a common storage location for data objects that are being referenced in models and used for code generation.

Data dictionaries typically store data objects such as the definition of global variables, OS messages, function interfaces and macros. In traditional manual programming, such data is directly coded in C. This is why data dictionaries are seldom used in manual programming. However, together with a model-based design methodology, data dictionaries are very beneficial. Their major advantages are:

- a common project data source for large ECU projects;
- support for multi-model projects, allowing projects to be spread among different models, with their shared data objects stored in one location;
- protection of intellectual property by a systematic separation of the model-based algorithm specification from the data dictionary-based implementation specification;
- variant handling by supporting multiple values for a single property or by switching complete data dictionaries or branches of one data dictionary in the background.

A dialog-guided model refinement process, utilizing the user interfaces as described above, relieves the implementation specialist from a lot of tedious detailed work. He still specifies the implementation on a bit-accurate level, but does not write C code any longer. This reduces implementation errors and significantly increases software quality.

CONFIGURABILITY OF THE CODE GENERATION PROCESS

Model refinements are not just limited to data typing and fixed-point scaling. Properties that directly impact programming language aspects of the generated code are equally important when code is to be implemented on a production ECU:

- naming conventions have to be followed,
- variables have to be properly declared and put into the right memory sections,
- there need to be efficient ways to link to external code,
- the code output format should comply to company-specific standards.

Generated code for real production projects will always have to interface external code, specifically to components of the lower software layers or to the proven legacy code of the application layer. TargetLink has a wide variety of specification means on the block diagram level to easily interface with non-generated code. In particular, these are:

- inclusion of existing header files,
- use of external global variables,
- use of externally defined macros,
- call to imported functions,
- call to access functions or macros,
- definition of Custom Code blocks which contain hand-written C code.

Related to this is TargetLink’s full support of the OSEK/VDX operating system, see (OSEK/VDX, 2001) and (Thomsen, 2002). TargetLink provides an extended library of special OSEK blocks which make the operation of system objects, such as tasks, alarms or critical sections, available at the block diagram level.

The code can be generated in a format that exactly matches company-specific C code templates. Code output formatting is possible through XML and a XSLT style sheet. This allows the user to define the format of file and function headers, the format of code comments and the inclusion of specific header files. Furthermore, TargetLink-generated code complies with the MISRA C standard, see (MISRA, 1998) and (Thomsen, 2002).

The link between the code and the calibration tools is based on parameter description files which are standardized by the ASAM-MCD 2MC standard (formerly called ASAP2), see (ASAM-MCD 2MC, 2000). Modern code generators can all generate this format. Should there be a calibration system in use which applies a proprietary standard, then the user can write his own export filter. This can be done based on detailed information on the generated code which can be accessed via the TargetLink API. Alternatively the TargetLink-generated ASAP2 file could be post-processed within the MATLAB environment to any other format.

SIMULATION-BASED TESTING

Code generators and simulation environments complement each other in an almost symbiotic relationship. An integrated environment, such as Simulink together with TargetLink, allows a variety of significant development tasks to be completed. Simulation results are used for

- automatic fixed-point scaling,
- code testing and verification,
- benchmarking.

Although code generators work virtually flawlessly in comparison to manual programming, the generated code still needs to be tested. The strength of an integrated environment is that code tests are performed in the same environment that was used to specify the underlying simulation model. Functional identity is
achieved when simulation results match. The validity of tests is documented by code coverage measurement. TargetLink provides the environment for a three-step verification process, which shows that the model and the generated software components have identical behaviour.

The first step of this verification process is called model-in-the-loop simulation (MIL). It captures the specified behavior of the model by recording block output and block state data to an internal data server. The minimum and maximum values can be used for the automatic scaling of fixed-point data types mentioned before. The traces from MIL simulation are the basis for the subsequent steps.

Software-in-the-loop simulation (SIL) is the next step. Code is generated and compiled with a host compiler and executed in the same simulation environment.

![Figure 2: MIL, SIL and PIL simulation modes: a three-step process to verify generated code](image)

Code that runs correctly on the PC can still cause trouble on the target processor. Therefore, the final checks need to be done with processor-in-the-loop simulation (PIL). An off-the-shelf evaluation board equipped with the target processor is connected to the host PC; the generated code is compiled with the target compiler and downloaded to the evaluation board. TargetLink manages communication between the host PC and the processor board.

Performing MIL, SIL, PIL simulation and switching between the different simulation modes is completely automated and does not require any user interaction. In all supported simulation modes TargetLink allows signal traces of block outputs to be logged. These signal traces can be saved and plotted on top of each other, thus providing direct visual feedback and allowing further analysis. This is especially helpful for inspecting quantization effects and verifying the float-to-fixed point transformation.

If plots from the PIL simulation deviate from those in the SIL simulation, then the most likely cause is a problem with the target compiler, or in rare cases a problem with the processor. Both, SIL and PIL simulation, directly support code coverage analysis. The user can select between statement or decision coverage. If the plots match each other and a sufficient coverage of the generated code has been achieved, then the behaviour of the generated software component is with a high level of certainty equivalent with the behaviour of the specification model. This three-step simulation approach is easy, intuitive and quick – and, as a result, it is a safe testing method.

PIL simulation can also be used to profile the generated code and to further refine the implementation. During simulation, TargetLink automatically measures the execution time and stack consumption of the generated C functions directly on the target processor. Furthermore, code summaries list the RAM and ROM usage detailed for each function. These features allow the user to quickly try out implementation options, immediately measure the impact of the change on the generated code, and make logical implementation decisions for the most efficient implementation of a software component.

**GASOLINE DIRECT INJECTION CASE STUDY**

The model-based methodology has been applied to the gasoline direct injection (GDI) engine control. The most innovative concept of a GDI engine that requires new control algorithm is the ability to inject the gasoline directly in the combustion chamber through an injector. This capability removes the restriction of introducing fuel into the combustion chamber only when induction valves are open, and as a result a GDI engine has better performance and fuel economy and less pollution than traditional gasoline one. The complexity of a GDI engine resides to the need of a more precise control on the fuel-air mixture and combustion. In particular, the system differs from traditional one for the presence of a high-pressure fuel pump (to inject fuel directly into the cylinder), injectors that support a high pressure flux of gasoline and generate adapted spray pattern (Pontoppidan and Gaviani, 1997) an intake port that generates the desired vortex in the combustion chamber, a more complex treatment of exhaust gas. The engine runs with two different independent combustion modes: homogeneous and stratified. The former being the traditional combustion mode, the latter presenting a non homogeneous air to fuel ratio (A/R) in the combustion chamber.

The high complexity and the presence of innovative control algorithms make this system a perfect case study. Moreover, the strong dependency between the design of the combustion chamber and the design of the combustion control algorithm requires a deep analysis (prior implementation) and a strong interaction, with exchange of models, between car makers and sub system suppliers. The most important component of a GDI engine management system are: the air control by electronic throttle (DBW), variable valve timing or exhaust gas recirculation (EGR), self diagnosis for sensors and actuators and for emission regulations (EOBD/OBDII), safety control, exhaust emission control with Lambda sensor and linear lambda sensor to control the A/R, NoX sensor for NoX trap control, high pressure injector control.
Starting from the car maker requirements, the system has been decomposed and refined into 125 operations decomposed as shown in Table 1.

<table>
<thead>
<tr>
<th>Components</th>
<th>SLOC</th>
<th>%Model Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLATFORM</td>
<td>26</td>
<td>0%</td>
</tr>
<tr>
<td>APPLICATION</td>
<td>86 with MC</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>13 HC</td>
<td></td>
</tr>
</tbody>
</table>

A set of 86 operations has been completely modeled and automatically translated to C code via TargetLink, resulting in 94% of the total application code. The application component accounts for more than 2/3 of the total lines of code, while the remaining part is related to the software platform. As expected the system design cycle has been reduced compared to the traditional approach by 20-40%. However, the time of the first design cycle was comparable (or even longer) of the traditional one. This was mainly due to the complexity to harness the design process and build the modeling library. The subsequent design cycles have been drastically faster and the final number of design cycles has been reduced. If we consider the number of software engineers involved in the production of the code, the productivity increased by a factor of 4, in terms of line of source code per hour (SLOC/hour). This improvement has been confirmed also in design cycles with strong modification of functional requirements. As a result of this increased software productivity, the ratio between the number of software engineers and control engineers in the design team was lower than in previous product developments.

The model compiler has been applied only to models that are mapped to the application partition. This partition contains mainly new control strategies and some legacy software components implemented by hand. The former have high level of reworks due to the strong interaction with car makers to finalize the requirements and are subject to several design cycles during the system development. The reuse of these components across different products might not be very high.

The software platform defined in the design methodology is instead shared among several products and the variation of the constituting components follows the variation of the hardware components. The platform code has been written by hand since it is highly efficient code and its functionality it is not typically subject to variation during the development time. The strong separation between application and platform and the enforcement of a software architecture also for the platform have been instrumental to manage a variety of ECU configurations and different hardware platforms. In particular, the number of cylinders has been parameterized and can vary from 2 to 6. At the same time, the platform handles the different lists of sensors and actuators needed to support the different engine configurations.

The resulting encapsulation of the hardware platform has been proven by the variations of the set of custom ICs, during the design time that did not require any modification of the control algorithms.

In details, the platform has allowed to manage up to 7 different types of engines and ECUs belonging to 2 different car makers without modifying the control models.

This flexibility is the result of the adopted methodology that encapsulates these variants with the minimum amount of software differentiation. In particular, all the hardware and engine configuration variants have been captured in the lower level of the layered software architecture, respectively BIOS and device drivers, while the software application has been composed with the automatically generated or hand written software components. This flexibility introduced by the software layering has also encapsulated the evolution of the ECU from the first hardware prototype (A) to the start of production.

The model based software components have been also delivered to multi-point injection engine managements, today in productions.

**CONCLUSION AND FUTURE WORKS**

The methodology described in this paper has shown in the years of use in the GDI product development its validity and the maturity level of the tools. The application to a real product has shown the improvement of the time-to-market and the capability to cope with the complexity of modern power-train controllers. The TargetLink model compiler has been instrumental in implementing our model-based design methodology.

To further improve the cost reduction, a tremendous effort in modeling power-train physical processes and other electro-mechanical components (sensors and actuators) will be required. We feel that in the future, the creation and use of plant models will play a strategic role in the automotive domain.

Other improvements must be done to better cover some important design aspects, such as requirements tracking at the model level, unified framework for refinement and model protection to support the exchange of intellectual properties between car makers and subsystem makers.

An important problem of future investigations is the capability to support the development of ECUs based on models captured with different semantics and tool environments (ASCET-SD).
In the future we expect:

- to have more data related to the process to quantify the advantages of the approach in large scale development;
- to start a formalization of architectural aspects, such as the description of the software platform with architectural description language and UML, based on the AUTOSAR standard;
- to improve the integration in the design chain.

In the near future, we plan to extend the use of the model-based design methodology to other power-train applications and to exploit the new coming features of the new releases of TargetLink.

Finally, the application of the model-based design methodology is expected to drastically decrease the time-to-market of new power-train controllers. In conclusion, the definition of a common design methodology and tool chain is the key of success in coping with the complexity and constraints.

ACKNOWLEDGMENT

We would like to thank Alberto Sangiovanni-Vincentelli for the foundation of the platform-based design methodology and A. Balluchi from PARADES for the main conception and contribution to the methodology. Cesare Pancotti, Giovanni Stara, Giovanni Reggiani, Walter Nesci and Paolo Marceca from Magneti Marelli Powertrain for their contribution to the GDI project and Software Architecture.

REFERENCES

8. TheMetropolisProject
http://www.gigascale.org/metropolis
10. OSEK/VDX Operating System (2001), Version 2.2
15. AUTOSAR, www.autosar.org
17. ASCET-SD, www.etas.de

CONTACT

Alberto Ferrari
PARADES EEIG
Via San Pantaleo, 66
00185 Roma, Italy
mailto:aferrari@parades.rm.cnr.it

Michael Beine
dSPACE GmbH – Product Manager TargetLink
Technologiepark 25
33100 Paderborn, Germany
mailto:mbeine@dspace.de

Giovanni Gaviani
Magneti Marelli Powertrain – R&D Director
Via Timavo 33
40100 Bologna
Italy
mailto: Giovanni.Gaviani@bologna.marelli.it

Giacomo Gentile
Magneti Marelli Powertrain – Design Methodologies
Via Timavo 33
40100 Bologna
Italy
mailto: Giacomo.Gentile@bologna.marelli.it

Stefano Monti
Magneti Marelli Powertrain – Sw Architectures
Via Timavo 33
40100 Bologna
Italy
mailto: Stefano.Monti@bologna.marelli.it

Luigi Romagnoli
Magneti Marelli Powertrain – Sw Architectures
Via Timavo 33
40100 Bologna
Italy
mailto: Luigi.Romagnoli@bologna.marelli.it