Hardware and Software Development Framework for Embedded Automotive Electronics

P Mortara

Hardware and Software Development Framework for Embedded Automotive Electronics

P. Mortara

Ixfin Magneti Marelli Electronic Systems
10078 Venaria Reale – Torino – Italy

Abstract

Electronics in Automotive Applications is constantly growing with an increasing quantity and complexity of their related embedded software. Multiplexed networked architectures give the possibility to implement different distributes solutions thus allowing optimized partitioning.
This paper has the aim to describe the Hardware and Software Development Framework conceived and realized in Ixfin Magneti Marelli Electronic Systems in order to improve and optimize design and prototypes implementation of distributed electronics in the area of body applications.
Quick function re-allocation, solutions reuse, customer function integration, time to market reduction and R&D costs containment were the main objectives of this project.
The document starts with a brief overview of the development process introducing the concept of the platform approach.
Then both the Hardware and in particular the Software development framework are described and discussed in details.
Some examples of real applications of the framework, performance indexes and balance metrics of the obtained results are presented in the conclusion.

1. INTRODUCTION

Electronics in Automotive Application will grow strongly in the future years. Concerning Body Car Applications (body computer, doors / light / seats management, anti-theft, climate control..) starting from new products generation up to 2005-2006 is also expected that Multiplex Systems will gradually move to a Sub-Systems integration increasing in the meantime the number and the complexity of functionalities.
Dialing with such a scenario, instead of an ECUs modules development approach, a 360° System vision is necessary. This will be useful to assure flexibility in hardware and software design, to grant possibility of quick function re-allocation, reuse of solution, reducing time to market and containing R&D costs. Functional distribution on different nodes allows the optimization of computing power in the Multiplex Systems: algorithms, strategies, and high level functions do not necessary need to be resident in a specific ECU. Optimal function partitioning improves flexibility in the multiplex system definition, from low end up to high end configuration.

This article has not been conceived with the aim to discuss engineering principles, but to illustrate with a real fact description the concrete example of their application in the R&D center of an industrial automotive electronic component supplier.
The paper is principally focused on the hardware and software development framework for prototyping platform based design.
Nevertheless for more clearness a very short overview of the process and the others main development stages are recalled in Chapter 3
More details on system design and software engineering concepts can be found in the articles listed in the bibliographic reference.

2. PLATFORM CONCEPTS

Following the above considerations Ixfin Magneti Marelli Electronic System has defined a Development Process based on Platform approach with these main objectives:

- To anticipate the needs for functionality of customers
- To anticipate technological and methodological steps aiming to take competitive differentials and to improve product functionality
- To develop on the shelf available competitive solutions

A product Platform is:

- A defined Hardware and Software System Architecture

- A library of standard configurable Hardware and Software validated components
- A set of test solutions, modules and tools
- A defined industrial process
- A workbench useful for the development of the first functional prototypes of all applications

3. DEVELOPMENT PROCESS

The "Y" graph of the System Design approach is synthetically represented in Fig. 1.
Modeling and Simulation, Architectures and Components selection and the different possible instances mapped solution provide the input to the Hardware and Software develepment framework for prototype implementation.

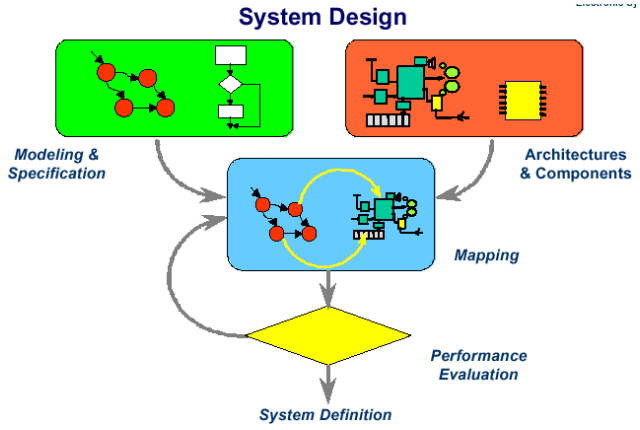

Fig. 1 System Design – "Y" flow graph

3.1. MODELING & SIMULATION

This phase uses in input the set of functional requirements coming from different customers as well as internally defined preliminary specifications for new innovative functionalities. This represents the functional system perimeter that will address the platform development approach.
With the modeling and simulation activity performed with appropriate tools (continuous and discrete modeling, network simulation) preliminary different possible distributed architectures are verified.

3.2. ARCHITECTURES & COMPONENTS - MAPPING

To allow preliminary individuation of the optimal hardware architecture and components able to satisfy the requirements characterized in the previous analysis stage, a dedicated internal company tool called F_DIAMOND has been designed.
F_DIAMOND is the acronyms for Function Deployment on Instances and Architecture Mapping Of Device.
It's essentially a software database; it has been conceived to collect technical, economics and availability time information of new devices offered by semiconductor market for automotive applications. Added to the database feature there are some special functions that can help designers to:

- Define a set of hardware requirements (elementary resources)
- Find the available devices into the database that satisfy partially or totally the requirements
- Choose a set of devices (one or more) that can perform the required functions
- Compare different solutions in terms of performance, cost and timing.
- Collect and summarize the results in standard format report files

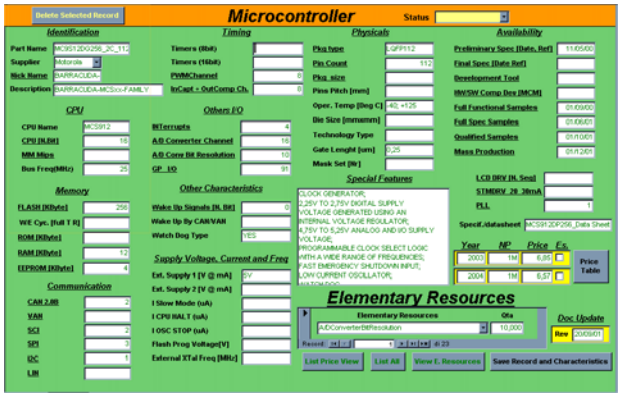

Fig. 2 Database Components Library

The main goal of the database, was to create a supporting tool compatible to the "Y development flow " of system architecture, described in Fig. 1, in order to evaluate alternative electronic architectures.

F-DIAMOND provides to:

- Organize component's information offered by the market in a database. In Fig. 2 an example of the GUI for micro-controllers data entry is represented.

summary previous next

- Support the definition of the electronic resources required by the platform, starting from the functional requirements.

- Give the support to associate these components in architectures fulfilling the Platform's requirements. This means to help designer searching components and collect them in a kit (named instance) that satisfies the requirements (instance creations) (Fig. 3).

- Give support for comparison evaluations of alternative instance solutions regarding technical performance, cost and availability.

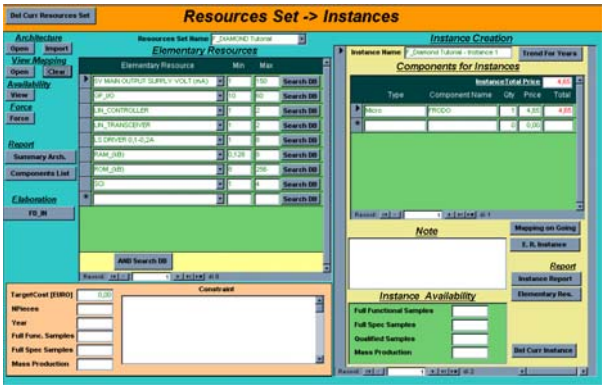

Fig. 3 Create Instances

4. HOW (HARDWARE OPEN WORKBENCH)

.

The HOW (Hardware Open Workbench) represented in Fig. 4, as been designed with the aim to support in an adequate way the hardware prototype development with the following main objectives:

- Quick implementation of new functions
- Easy & quick function re-allocation / partitioning
- Test / Verification of different system architectures
- Test / Verification of different components/circuit solutions
- Fast reusability in several projects and the possibility to share circuit solution between different teams
- Direct use of hardware circuit solutions just re-composing the electrical schematic, and build-up the definitive Layout
- Use of the software platform running on workbench as it is on the product

4.1. BIG BOARD

The Hardware part of the development framework is based on a 590 x 470 mm four layers Big Boards PCB implementing the basic layout interconnection between every plug-in connectors (Logic and FPGA flat cables connectors, Function Boards connectors, external connectors...).
This is the fixed part of the Workbench that remain the same for any implementation.
Big Boards may represent what in the actual Body Multiplex Systems are normally called *Body Computer* and *Body Nodes* (doors, roof, trunk..etc). Big Board hosts connectors based Logic and Function Plug-in interconnected by a FPGA (for logic signals) and manual jumpers (for analog signals). The interface to the externals sensors/actuators implemented in the laboratory Bench Simulator or in the Car is provided with dedicated connectors. Big Board has been designed in order to host up to 21 Function Boards Plug-In.

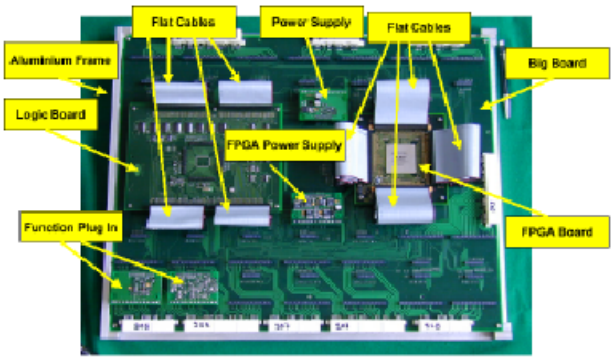

Fig. 4 HOW (Hardware Open Workbench)

4.2. LOGIC BOARD

The Logic Board basically host the micro-controller and his peripherals: Mux, glue logics..etc (Fig. 5)
The design approach has been addressed in order to support the most complex I/O features for futures body applications. All Body Computer actual and possible futures functionalities as well as Body Nodes (doors,trunk etc.) and Climate Control has been considered. The obtained I/O features increased of 20% more or less, represent what we've called SLB (Standard Logic Bus).

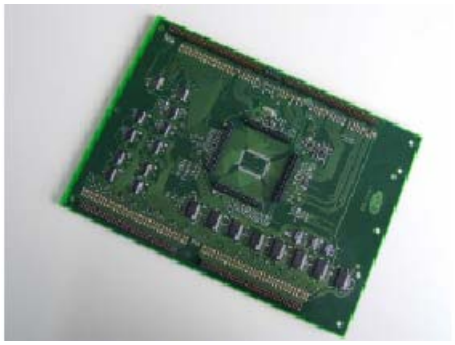

Fig. 5 Logic Board

The basic assumption is that any logic architecture for platforms Nodes will use the same SLB; the quantity of I/O features present on LSB will depend of course from the complexity of the particular Node.
Completely different logic architectures, starting for instance from the smallest Node to the most complex Body Computer, based on different micro-controller, can be implemented on the base of the same board boundary.

Fig. 6 FPGA Board

4.3. FPGA BOARD

The FPGA Board implemented on a 150x150 mm PCB (Fig. 6) has the aim to provide the mono-directional digital I/O connection between the Logic Board and Plugs-In Boards. This is designed using a BGA 600 FPGA giving a flexible solution for easy and quick Logic and Functions reallocation. In consideration of the high number of signals, the connection to the Big Board is implemented using 8 Flat Cables. For more stable environment such as final platforms to be released to the Application team this Board may be removed and substituted with direct connections realized using a Wire Wrapped Harness: this is also a cheaper solution respect the more flexible but also high expensive FPGA approach.
The FPGA used in the body platform supports the in-system programmability (ISP).
Configuration data are downloaded from the PC parallel port via a download cable.

4.4. FUNCTION BOARDS

Function Boards implements the modularly Plug-In concept.
They host the hardware features to be implemented around the Logic core.
Each Function Board from the structural point of view is designed in the same way and it is basically composed by a 80 x 50 mm PCB, 14 pin Connector for the external I/O and 20 pin Connector for the FCB.
All functionalities are implemented using this predefined scheme and grouped in an homogeneous way. In case the functionality exceeds the capability of the perimeter, it will be splitted in more Plug-in.
Some examples of Function Boards Plug-in are: Analog Input, Digital Input (Fig. 7), Relais Driver, SmartMOS Driver, Serial Line Transceiver (CAN/LIN Driver), RF modules…etc.

Fig. 7 Function Board Plug-In (Digital Inputs)

4.5. SUB-SYSTEM NODES

Sub-System Nodes host the smallest device oriented functionality that take part for example in the sub-bus LIN network.
Also in this case such a standard general purpose hardware architecture has been designed. (Fig. 8)
Both Master / Slave LIN Nodes have been designed around a low cost 8 bit micro-controller on a quite small PCB.
The Master LIN node also provide the Gateway function with the CAN.
Using this approach, different functions, from window power lift to the seat management for instance, can be quickly implemented on the same common base.
This implementation allow to tune both Hardware/Software functionality and should be the output for a custom IC / smart components design specifications.

Fig. 8 LIN Node

5. SOFTWARE ARCHITECTURE

Real Time Operating System and software Layered Architectures are well know and used in the company since more than ten years when we start in mass production with our first 32bit Engine Management Control Unit ([8])
Nevertheless the architectures were continuously refined, taking into account the evolution in the automotive standards and in the hardware and software components.
The today main scheme of the Software Architecture looks like the Fig 9.

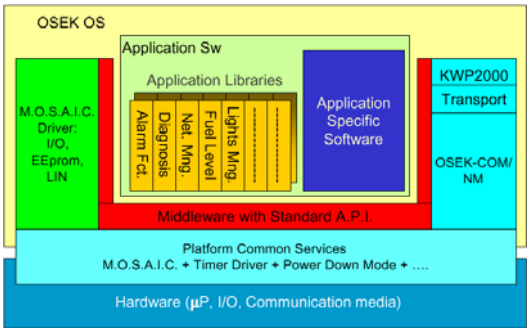

Fig. 9 Software Architecture

The low level is composed by the basic I/O modules drivers for analog, digital and frequency signal management. As discussed in [2] this part represent the HAL (Hardware Abstraction Layer) that include all the micro-controller and the Hardware devices dependent Software.
Basic functions are integrated for the most part in a developed in house environment called MOSAIC (Modular Open Software Architecture, Integrated and Configurable).
The skeleton of the Software Architecture is completed with the OSEK compliant Communication Components and the Operating System normally acquired from third parts software suppliers.
The middleware with their internal standard A.P.I. provide the mean for interaction with the high level application functions.

5.1. THE SOFTWARE LAYER STRUCTURE

The Components and their related sub-modules of the Software Architecture are mapped in 5 Layers (Fig. 10).

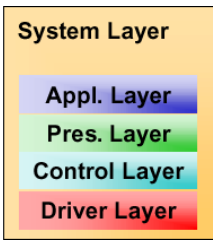

Fig. 10 Software Platform Layers

- **Driver Layer:** provides the access to the internal micro-controller resources and peripherals
- **Control Layer:** provides the control of ECU elements, management of physical values and platform services such as the generation of time-base or the transmission of one byte to the serial communication interface.
- **Presentation Layer:** provides standard access to process data.
- **Application Layer:** provides application dependent function and some particular configurable services, such as gateway function, periodic and/or event messages transmission.
-
- **System Layer:** provides management of the overall software. Operating system, scheduler, boot software is here included. It provides the declaration of common definition for all platform modules.

Each Layer communicates with the others by means of internally defined standard API.

5.2. OSEK COMPONENTS

Fully configurable and OSEK compliant real time Operating System and CAN driver with Keyword 2000 communication protocol and Transport Protocol are the only third part software components used in the Gen. 2 of our Platform Architecture.

The choice of the Operating System Conformance Class depends of the characteristics/complexity of the node, but basically for Body Applications the most used are BCC2 and ECC1.
KWP2000 and Diagnostics Services are usually specialized in the respect of the OEM customers requirement and implemented by the third part software supplier itself.
The same for the Network Management even if for some car manufacturer the component is directly implemented in the Application Layer following the customer specification.
The OSEK specifications was defined with the aim to improve re-usability / portability of the embedded software components. Nevertheless OSEK concerns the RTOS and part of the network communication system, but does not take in charge other input/output system interface with the hardware subsystems. Thus the software implementation only based around OSEK environment is not enough to enable the completely re-use and portability of software components.
This is the reason why the environment called MOSAIC in which a specific API abstracts the interface between the application software and the system platform has been conceived and internally designed.
In the following an essential overview of this environment is provided. More details can be found in [2].

5.3. MOSAIC

MOSAIC take in charge the software components that are not provided by the OSEK/VDX standards. Basically implements the abstraction layer from the logic boundary hardware architecture (Fig.11) such as the internal micro-controller peripheral devices (A/D converter, digital and frequency I/O channels etc.) as well as the external dependencies, typically sensors and actuators management.
MOSAIC software pieces are implemented like a components libraries that can be configured and specialized using the MOSAIC manager. All I/Os of the same type (e.g., analog inputs, PWM inputs, digital inputs, digital outputs) are grouped together and managed by a specific software component called the *I/O engine*.

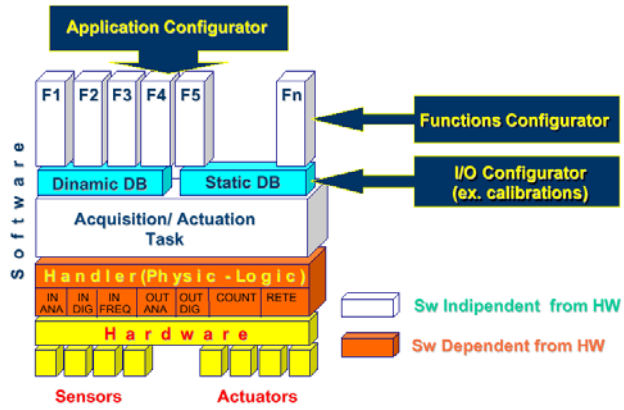

Fig 11 MOSAIC Architecture

The main configuration scheme is represented in Fig. 12.
All configuration parameters are stored and maintained in a relational database.
The database contains components data describing the hardware platform and all sensors/actuators of the system and application specific data of the I/Os configuration for a specific control unit on the vehicle network .
The configuration Manager provides the interface between the user and the database allowing the data introduction and automatically implementing validity and consistency checks.
By means of dedicated GUI sessions it is used to introduce I/O hardware and software informations. Hardware information includes component selection (micro-controllers, analog and digital multiplexers, power devices..) resource allocation including pin assignment, I/O ports, registers and memory map.
Software information includes all the parameters necessary for the acquisition or actuation of the I/O as well as for the intermediate treatments defined at the API level.
For instance linearization tables for A/D treatments can be introduced choosing between several structure type: linear, step with hysteresis, dynamic and others (Fig.13).
The configuration manager today supports I/O modules such as:

- Analog inputs;
- Digital inputs;
- Digital outputs (including power devices and PWM);
- Frequency inputs;
- LIN;
- Stepper and DC Motor controllers
- EEPROM Driver

The complete hardware and software informations set introduced with the Configuration Manager represents the configuration characteristics of a node. Several nodes can be created, modified and managed with different versions.

A new node configuration can be created copying and modifying from an existing one thus reducing the effort.
When the configuration phase is terminated the MOSAIC environment will be able to provide the code generation.
In according to the rules defined in each I/O engine in the template module, the configuration data are extracted by the parser from the data base.
The configuration files to be included and linked with the I/O engine modules and application software will then automatically created by the code generator that will provide with the configuration data insertion.
The specification of the implemented module can be automatically produced by the documentation generator that reports all the information contained in the database in a formatted document.
The overhead introduced by this configurable layered environment (5%-20% for ROM, below 5% for RAM and less than 10% for CPU load) is tiny respect the acquired benefits.
The effort needed for a complex node configuration is today evaluated in hours instead of weeks for manually written coding approach.
Only few minutes are required to make changes due to significant hardware modifications.
.

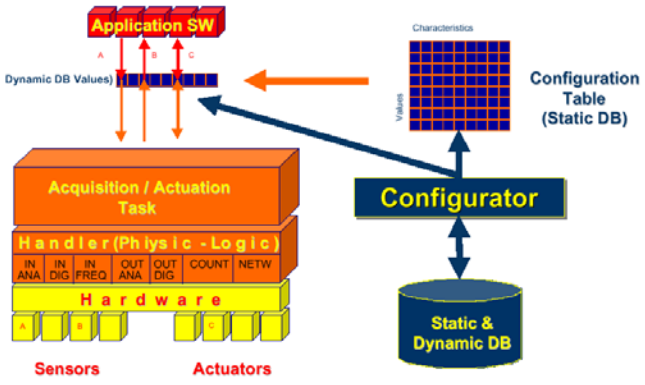

Fig 12 MOSAIC Configuration

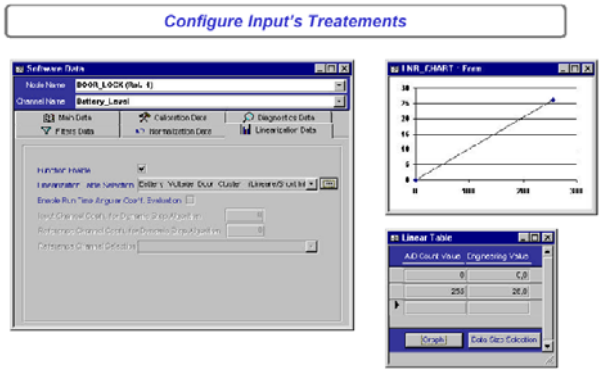

Fig. 13 MOSAIC – GUI Example

5.4. APPLICATION COMPONENTS

Application Components are located in the Application Layer of the software Architecture and implements the high level algorithms/function computation . The communication with the lower Layer is assured by the use of internal standard API.
Examples of Function Components are for instance the Fuel Level Calculation, the internal Light management, the door lock/unlock high level strategy management etc.

5.5. EXAMPLE OF A COMPLETE FUNCTION

In the Fig. 14 the example on how a complete UHF receiver function is mapped in the four layers of the software architecture is illustrated.
Using this layered structure, for instance in changing the remote transmission device with a new one that use a different data encapsulation, only the Presentation Layer of the receiver function need to be changed, while substituting the receiver components both the Driver and Control Layer have to be modified; if this two level of the software receiver management are implemented in the MOSAIC environment the modification can be done only using the configurator thus avoiding any direct modification of the source code.
Any changes on the upper level strategies control will require interventions in the Application Layer only.

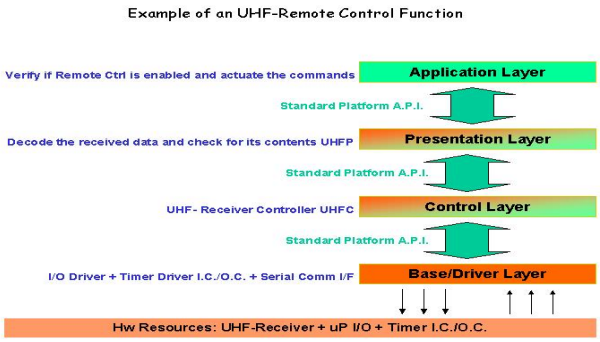

Fig. 14 UHF Remote Control Function

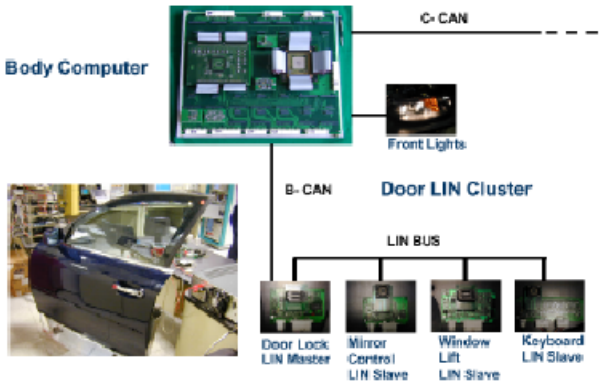

Fig. 15 The Body Platform

6. EXAMPLE OF REAL APPLICATION

The Body Platform Gen 2 was the first implementation of the integrated hardware and software development framework. It has started following the system design approach illustrated in Chapter 3 and has been based on an extended functional perimeter that include "standard Body functions" (ex. light, widows, doors management…) in different OEMs visions as well as new expected but at that time, not yet well defined new features (ex. Tire Pressure Monitoring System, enhanced Passive and Remote Keyless Enrty, doors, trunk and roof clusters sub-systems ..)

6.1. THE BODY PLATFORM

Hardware Open Workbench for both Central ECUs (i.e. Body Computer) and Sub-system cluster node was equipped with a wide range of function plug-in covering the defined functional perimeter. Following criteria and tools described in 3.2 several components solutions and relative instances was created as a input for functions plug-in design.
On the base of the software architecture features described in Chapter 5 a lot of different software components have been designed and swiftly configured using MOSAIC .
The body platform hardware environment looks like the picture represented in Fig 15.
Beside a Central ECU Node a complete Door Cluster has been be implemented including:

- Window Lift (Master LIN node)
- Mirror control (Slave LIN node)
- Door Lock control (Slave LIN node)
- Keyboard control (Slave LIN node)

This prototyping development phase has allowed the final concrete verification on both laboratory bench and in the car the different solutions identified during the previous Modeling and Simulation stage.
For instance the complete door management has been really tested in two different solutions:

- the functions implemented in the central ECU
- the functions splitted in the LIN sub-system door cluster

The strong word "*final*" means that the hardware circuitry solution is the concrete one (make a real ECU only needs to join schematics and build the layout for PCB) as well as the software platform that will remain exactly then same in the final product.
As illustrated in Fig 16-17 excluding the definitive characterization, EMI/EMC Test .., the most part of the hardware and software validation can be anticipated in a early phase.
Moreover the Software architecture will easily allow customer function integration in the application development phase.
The use of the hardware and software framework allows to save time money and improve quality for new developments.
Hardware solution circuitry (Logic boards, Function plug-in) can be reused and physically shared among different development teams also located in different sites. Obviously is the same for software: the configuration feature provided by the OSEK components and in particular by the MOSAIC environment, beside time and money savings, will drastically reduce the possibility of errors introduction of manual coding, thus improving quality and reliability. Moreover the well structured layered approach easily allow the function application integration as well as their partial modification.

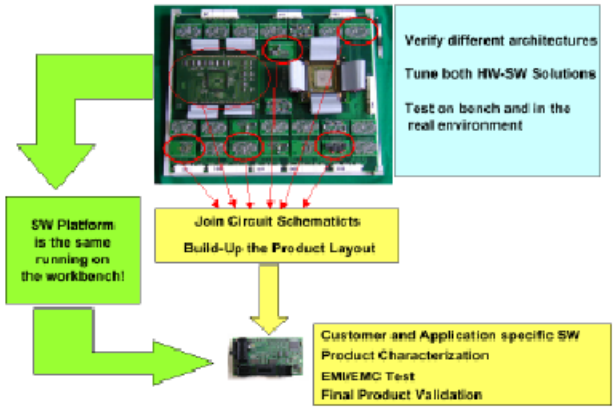

Fig. 16 "HOW" to build the product – Central Nodes

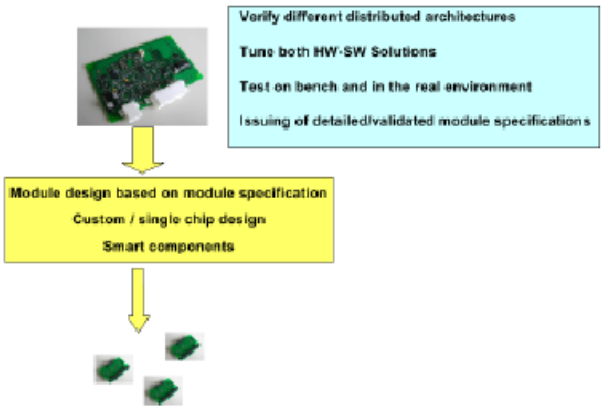

Fig. 17 "HOW" to build the product – Sub-System Nodes

7. BALANCE METRICS

As an example of the benefits of the application of the framework, the results of a porting activity due to the change of a micro-controller in the Body Platform will be described in the following.

A favorable contract with a silicon supplier entailing the use of a new 16 bit micro-controller on Body Electronics Applications was signed during the product development.
An hardware and software porting activity to *micro-controller B* from the existent Body Platform based on a *micro-controller A* was consequently swiftly addressed.

Due to the advanced development phase, time constraint was very tight, and a detailed aggressive plan has been drown up with the silicon supplier that was involved in the porting activities and development tools reintegration.

Excluding the third part software components (OSEK OS and COMM) the amount of implemented software at the moment of the micro-controller change was 46 modules for a total of 23452 pure line of code (without comments and empty lines).

The actors in the porting activities were:

1. The Silicon Supplier

   - To assure the HAL MOSAIC modules porting and some others low level drivers, *the micro-controller B* development environment and tools support.
   - To support Ixfin Magneti Marelli for the software integration/verification till end of platform delivery.

2. Third part Software Components Supplier

- To assure the availability of the complete software suite (OSEK OS,COMM,TP & KWP2000) for the specific *micro-controller B*

3. Ixfin Magneti Marelli

- To assure the porting of all the above layers and the complete software integration/verification
- To integrate the third part software components with remote support of the supplier.

From the hardware point of view, using the Open Workbench only few days were spent to redesign the Logic Board substituting the CPU and some glue logic components. Interconnection with the plugs-in was made with jumpers and a swift FPGA reprogramming.

Related to the Software only the 50% of modules required modification. The remaining 23 modules have been reused as their was just re-compiling the C source code for the *micro-controller B.*
The percentage of modified code lines respect the total (23452) was 28%.
We must point out that the micro-controller B was a completely new one and the low level drivers of the MOSAIC were not yet available in the library.
So, in terms of effort a significant part were related to the HAL modules of MOSAIC (that includes the hardware depended software and low level drivers) → 35%, one serial communication driver that has been rewritten in

software due to the absence of one more UART on the new micro-controller →10%, third parts modules integration/adaptation → 15%, and the remaining effort was spent for general environment troubleshooting activities (Emulators,Compilers,…).
The porting activity begun in the second half of July 2001 and has been successfully finished in less than two months.
This include the complete software integration and verification at laboratory bench.
The structured layered architecture of the Body Platform has allowed a good job partitioning and the possibility to work in autonomous way in different separate sites.
Despite the fact that Platform Software, at that stage of development didn't yet include all the application and a great part was quite near to the hardware dependencies, the 50% of totals modules didn't need any modification and have been reused as their was just re-compiling the source C code.
The remaining porting of application functions to complete all the Body features has not increased significantly the effort already spent.
The effort spent for pure software porting was drastically reduced: more than 60% less respect an equivalent less structured and not Platform based previous porting.
Now the availability of many micro-controllers abstraction layer integrated in MOSAIC allow us to gain another great reduction in development time.
As asserted in [2], the time needed to implement changes for different architectures is now really measured in days.

8. CONCLUSIONS

In this paper the development process and the Hardware and Software Development Framework conceived and realized in Ixfin Magneti Marelli Electronic Systems in order to improve and optimize design and prototypes implementation of distributed electronics in the area of body applications have been presented.
This is a concrete example of the application of several engineering techniques and in particular the platform approach described in [2] ..
The application of the framework dials with new complex scenarios in automotive electronics in which instead an ECUs modules development approach, a 360° System vision is necessary and a rational function partitioning is a must to improve flexibility in the multiplex system definition, from low end up to high end configuration.
Moreover quick function re-allocation, solutions reuse, customer function integration, time to market reduction and R&D costs containment take significant benefit with the platform approach implemented in the framework.
As illustrated in the examples of application in the area of Body Electronics, technical results was very good. The use of the HOW give the advantage of rapid prototyping environment but is very close to the final product : circuit solution and software remain exactly the same.
Finally, the balance metric example of the porting activity illustrated at the end confirm the validity of the approach.

9. ACKNOWLEDGMENTS

Thanks to Piero De La Pierre and Franco Salerno for their contributions, and to Jean Lopez, Roberto Bettini and Antonino Damiano that have constantly supported and inspired the work.

10. REFERENCES

[1] A. Ferrari and A. Sangiovanni-Vincentelli,- System Design: Traditional Concepts and New Paradigms, Proceedings of the 1999 Int. Conf. On Comp. Des., Austin, Oct. 1999
[2] R. Bettini, A. Damiano, A. Ferrari, A. Sangiovanni-Vincentelli, L. Tonelli - A Configurable Software Platform for Embedded Systems – In DATE 2001 – Munich – March 2001
[3] Wolfang Pree, Component-Based Software Development – A New Paradigm in Software Engineering?, Software-Concepts and Tools, Spring-Verlag, 1997
[4] C. J. Hagen and G. Brouwers, Reducing Software Life-Cycle Costs by Developing Configurable Software, Aerospace and Electronics Conference, 1994. NAECON 1994., Proceedings of the IEEE 1994 National , 1994 , Page(s): 1182 -1187 vol.2
[5] S.M.Wheater and M.C. Little, The Design and Implementation of a Framework for Configurable Software, Configurable Distributed Systems, 1996. Proceedings., Third International Conference on , 1996 , Page(s): 136 –143
[6] OSEK/VDX System Generation, OSEK Implementation Language (OIL), Version 2.4.1, Jan 30th 2003
[7] OSEK Implementation Language - Concept and Future Perspectives of OIL, Goser, Janz, Schimpf – OSEK/VDX Open Systems in Automotive Networks, 3rd Internation Workshop, February 2-3, 2000 - Bad Homburg
[8] P. Mortara , G. Mercalli - A New Automotive Real-Time Engine Control System with 32 bit Micro_Controller -paper no. 945002 - FISITA CONGRESS October 17-21, 1994 Beijing (China).
[9] E. Dilger , L.Å. Johansson , H. Kopetz , M. Krug , P. Lidén , G. McCall , P. Mortara , B. Müller , U. Panizza , S. Poledna , A.V. Schedl , J. Söderberg , M. Strömberg , T. Thurner TOWARDS AN ARCHITECTURE FOR SAFETY RELATED FAULT TOLERANT SYSTEMS IN VEHICLES – ESREL European Conference on Safety and Reliability – LISBON 1997 .
[10] P. Mortara , A. Damiano - Problematiche Software nei Sistemi Elettronici per Applicazioni Automotive – Alta Frequenza – Rivista di Elettronica AEI – Vol..7 N.3 – Maggio-Giugno 1995.
[11] P. Mortara , A. Borin, G.M. Timossi - Sistemi Elettronici Distribuiti per Applicazioni Automotive – Alta Frequenza – Rivista di Elettronica AEI – Vol..10 N.5 – Settembre-Ottobre 1998.
[12] P. Mortara , S. Giuffrè, G. Maffei - Sistemi di Sicurezza in Architettura Distribuita per Applicazioni Automotive – Alta Frequenza – Rivista di Elettronica AEI – Vol..10 N.5 – Settembre-Ottobre 1998.