

EVS28
KINTEX, Korea, May 3-6, 2015

System-on-Chip-based highly integrated Powertrain Control Unit for next-generation Electric Vehicles: harnessing the potential of Hybrid Embedded Platforms for Advanced Model-Based Control Algorithms

M. Dendaluze¹

Tecnalia Research and Innovation, Industry and Transport Division ¹martin.dendaluze@tecnalia.com

Abstract

In this paper, a novel highly integrated System on Chip (SoC) based control unit concept is presented, which is conceived to combine the functionality of various powertrain ECUs and to additionally enhance their control with predictions based on the real-time execution of complex mathematical models.

Such a platform has the potential for providing considerable benefits to upcoming hybrid and electric vehicles, especially for highly complex and efficient hybrid and multi-motor electric vehicles. A clear advantage is the reduction of Control Units, consequently also reducing the system complexity, communication requirements and development effort, which are directly associated to production and development costs. A controller architecture capable of exploiting the strength intrinsic to the nature of each platform type is presented, basing on the new generation of high-performance hybrid SoC platforms which combine powerful processors with cutting edge FPGAs. The parallel hardware paradigm of FPGAs enables the implementation into a single component not only of several Field Oriented Control loops for electrical motors, power management functions and vehicle-level optimizations, but also of advanced real-time predictive algorithms. The combination of so many complex tasks would not be feasible on a typical automotive microcontroller unit.

A further objective for obtaining a highly integrated solution is to establish an efficient development and prototyping process, aiming to simplify and harmonize the workflow through the whole V model, by completely basing it on model-based-design tools.

Finally, the concept demonstrator being implemented in this paper will combine a state-of-the-art high performance microprocessor and FPGA using a commercial SoC platform together with model-based software development tools. This will also fulfil the expectation of providing a topology with great migration and industrialization potential for the case of higher qualification requirements. Furthermore, it is another step towards a necessary mindset shift on control system development and integration methods for increasingly complex vehicles.

Keywords: Control Unit, System-on-Chip, Multi-Motor Electric Vehicle, Advanced Algorithms, Model-Based

1 Introduction

It is well known that modern vehicles integrate a great amount of embedded systems with millions of lines of code and over 100 Electronic Control Units (ECUs) which provide all the functionalities today's vehicles have to offer [1]. This has led to a context where up to 90% of innovations in a vehicle are said to be associated to electronics and software [2].

Sophisticated next-generation electric and hybrid vehicles imply not only further challenges associated to the mentioned complexity, but also an opportunity to develop innovative features and enhanced performance and efficiency.

For instance, multi-motor vehicles will lead to having repeated components and even more complex system architectures and control algorithms. Also, the next generation of Advanced Driver Assistance Systems (ADAS) will require not only high levels of integrity and safety -as addressed by the ISO-26262-, but also immense processing capabilities and bandwidth [3].

Embedded platforms are inevitably evolving according to these changing requirements. This has led to the appearance of new platforms offering redundancy and other protection and integrity mechanisms. Performance figures are also notably growing, as is the strengthening of a wider variety of embedded platforms apart from microcontrollers, such as FPGAs and System on Chip (SoC) devices.

This evolution is accompanied by the increasingly well-established Model-Based-Development (MBD) methodology and software development standards following this concept, as is the case for AUTOSAR. MDB is ideal for the well-known V-model development approach and provides a good way to cope the complexity and technical implications associated to the previously mentioned points.

In this paper a solution dealing with the described challenges will be pursued, targeting to reduce the ECU number in modern electric vehicles and consequently the associated complexity, energy consumption and development and material costs. This means integrating the functionality of several ECUs into a single highly integrated ECU, which will be referred to as *hiECU*. The challenge is to harmonize the increasingly demanding requirements with the permanently growing possibilities of modern vehicles by the means of upcoming hardware and software solutions. This means exploiting the processing and parallelism potential of modern embedded platforms, while

establishing a smooth and scalable development workflow will also be crucial.

2 Embedded Platforms

In this section an insight into various embedded platforms of different types, which are relevant to the topics discussed in this paper, will be provided. After covering the state-of-the-art redundant automotive microcontrollers (μC) and some interesting FPGAs, the latest generation of SoCs which combine both of the previous platform types will be treated. The basic principles, main parametrics, applicability to automotive applications and performance figures will be discussed for each platform.

2.1 Microcontrollers

Several manufacturers have developed product ranges with microcontrollers designed to suit the requirements of automotive applications. Table 1 collects the parametrics of a series of such microcontrollers which have been selected as especially relevant. The newest high-end models of each of the selected manufacturers' product ranges have been chosen, meaning that for many applications a lower variant could probably be used to obtain an optimized relation between performance, features and costs. As all of them are well equipped with ADC, PWM and communication features, this information is not further detailed. The "Spec. tools" row is related to section 3 and is used to indicate if any kind of special development tool providing platform-specific hardware abstraction support is offered (further than usual Code-Generation).

Regarding to functional safety and integrity aspects, these devices offer a series of mechanisms in order to enhance their reliability, the integrity of their operations and fault handling. Typical mechanisms include redundant dual-core lockstep operation, CRC, ECC, MPU, watchdogs, hardware self-test mechanisms and diverse other fault detection and correction mechanisms [see references in table 1].

In order also to address the challenging requirements established by the ISO-26262 (or IEC-61508), diverse design resources, tools and documentation as support for certification are provided by the manufacturers. A series of solutions to address this and more information can be found in section 3. Certifications for the selected devices are also available [4][5][6].

When discussing performance, metrics for these platforms are in the range of 240 to 600 Dhrystone MIPS (DMIPS), as indicated in Table 1. This can be

used as an indicator of the raw computing capacity the devices can provide, as the effective functional performance will greatly depend on the efficiency with which the specific application is implemented. Furthermore, DMIPS stands for a “million instructions per second” [7], which is not representative for floating point operations. Unfortunately, it is difficult to obtain metrics for the more representative Whetstone benchmark or GFLOPS values, representing floating point performance. In practice, these floating point metrics should be expected to be at least one order of magnitude below the DMIPS metrics for these platforms [8].

Table 1: Selected microcontroller parametrics

Manuf.	Infineon	Texas Instruments	Free-scale
Device	Tri-Core 299	TMS570 LC435	MPC 5744P
Core	3x TriCore	2x ARM R5F	2x e200z4
Frequency	300MHz	300MHz	200Mhz
DMIP max.	>600	500	>240
RAM	728 KB	512KB	384 KB
Flash	8 MB	4MB	2.5 MB
Spec. tools	n.a.	[9]	[10]
~Cost	tbd€ Rev	~32€	~12€
Ref	[4]	[5]	[6]

2.2 FPGAs

In contrast to the Microcontrollers’ sequential instruction execution, FPGAs follow a radically different operation paradigm based on a “matrix of configurable logic blocks” [11]. In other words, they basically are programmable hardware, meaning that each small internal element can be reconfigured for the signals to physically propagate in a determined way, instead of executing instructions per se.

FPGAs have been widely used in multimedia and communications applications and, with the significant improvement regarding to performance, features and costs that they have experienced over the last years, their presence is not only intensifying in many fields, but it is also spreading to new fields.

Important progress has been achieved in their capability, efficiency and performance in handling high precision floating point numbers [12]. In this way, FPGAs have turned to be an excellent computation acceleration solution.

Furthermore, the fact of the extremely wide range of performance and cost ranges FPGAs cover,

both rather modest conventional control applications and more advanced ADAS applications can benefit from their parallelization potential. The following table 2 summarizes the parametrics of two low-mid-range FPGAs which fall in the reasonable price range for conventional automotive applications, as well as one high performance FPGA with a notably higher cost point.

Table 2: Selected FPGA parametrics

Manuf.	Altera	Xilinx	Xilinx
Device	Cyclone IV EP4 CE40	Spartan 6 XA 6SLX25	Virtex 6 XC6VL X75T
Logic	39K LE	24K cell	74K cell
Multipliers	116	38	288
Memory	1134Kb	936Kb	5616Kb
Cost	~50€	~45€	~430€
Ref	[13]	[14]	[14]

Manufacturers have been covering considerations such as automotive temperature ranges and AEC-Q100 qualification. Both dominant companies in the FPGA market, Altera and Xilinx [15], have showed a significant effort to address standards such as the IEC-61508 and ISO-26262 and they are currently obtaining the corresponding certifications. Automotive-grade device ranges are available and also backed by diverse software solutions, similarly as for the case of microcontrollers [16][17][18]. A series of internal FPGA hardware-level mechanisms for integrity enhancement and isolation exist, like redundancy, watchdogs and segregation by safety level, amongst others [19]. It is also possible to emulate redundant processor cores running in lockstep by using SoftCores [20]. For a further insight into safety and integrity topics, please refer to section 3.

Regarding to the performance metrics, these figures are strongly application and implementation dependant, in an even higher degree than it is the case for the microcontrollers. Nevertheless, significantly higher values can be obtained with FPGAs, especially when referring to GFLOPs [12][21]. Such results require a deeper analysis and must be interpreted as a theoretically achievable maximum value and should not be taken as directly comparable. In any way, it can be clearly seen how FPGAs are capable of providing vastly higher computational performance than microcontrollers. Even for the case that due to implementation aspects the performance figures should be one or as much as two orders of magnitude lower for certain applications, the performance remains being clearly higher. [12] [22]

2.3 SoCs

The kind of SoC which is of interest to this paper are those that combine both previously mentioned architecture types: processors and FPGAs. Combining both into a single component provides a series of interesting benefits, apart from having a better way to design a control architecture which exploits the intrinsic strength of each of the platforms. This is further reinforced by a much better interconnection, which provides not only higher bandwidth, but also significant additional interfacing circuitry cost and consumption savings. This means, improved efficiency regarding to costs, energy, development effort and also board space.

It is noticeable that when next-generation ADAS applications are discussed, FPGA and SoC frequently receive a strong emphasis [3][17][23]. This illustrates not only their remarkable processing power, but also critical-application suitability.

Table 3 presents the parametrics for three SoCs, where devices in the same cost order of magnitude as the previously discussed devices.

Following the path of their FPGA platforms, Altera and Xilinx are also offering automotive-grade SoCs.

Table 3: Selected SOC's parametrics

Manuf.	Xilinx	Altera	Microsemi
Device	Zynq Z-7020	Cyclone V SoC	Smart Fusion2
FPGA	Artix-7	Cycl. V	
Logic	85K cell	110K LE	56K LE
Multipl.	220	224	72
Process.	2x ARM A9	2x ARM A9	1x ARM M3
Freq.	677MHz	800MHz	166MHz
Cost	~90€	~95€	~90€
Ref	[14]	[13]	[24]

In what respects to safety and integrity aspects, for the points regarding the FPGA the information from section 2.2 can be referred to. In what respects to the processor, comparing to the automotive microcontrollers from section 2.1, the higher clocked and application-oriented ARM Cortex-A9 core used in Altera's and Xilinx's SoCs do not include as many intrinsic mechanisms as the cited automotive microcontrollers do. Still, it is feasible to achieve high integrity levels by combining the usage of the processor with the FPGA and by using ECC and MPU mechanisms [25].

Regarding performance, many points have already been discussed in the previous two sections. Information related to the FPGA basically is directly applicable. Nevertheless, in what respects to the processor, a more effective floating point handling is to be expected -due to the application oriented instruction set of the Cortex A9- and with its higher clock rate and dual core, raw performance will be correspondingly higher [22].

3 Workflow approaches

A great diversity of solutions can be chosen and combined with each other for addressing the development and implementation of systems like the one proposed in this paper. In the following sections an insight into the most relevant approaches to cover different aspects of the development tasks in the context of this paper will be provided. Section 4.3 will then describe the specifically selected solution for this work.

3.1 Model-Based-Development

Model-Based Development -or Design- is increasingly being used in automotive applications in order to address the steadily growing complexity and requirements of modern vehicles, as mentioned in the introduction. Complex architectures can be specified and modified in a structured and visual manner. Combining this with the corresponding tools, it is excellent for addressing crucial topics for complex and safety-critical designs, such as traceability, validation and documentation. [23]

The well-known AUTOSAR standard follows the same principles, establishing a workframe that aims to provide great reconfigurability for complex ECU-based systems. Abstraction is provided through different layers which include low-level components such as hardware drivers and communication stacks. In this way, Software Components (SWC) containing the desired functionality can be developed in a hardware-independent manner and then incorporated into the software structure running typically on a microcontroller. [26][27]

Code for SWCs can be generated by tools such as Simulink -with the corresponding toolboxes [28]- or dSpace Targetlink [29]. In this way, the same block-based control diagram can serve for all the development phases starting from initial conception and simulation to final code production for different target devices, including also processor or FPGA in-the-loop simulations [28]. Legacy C code may also be integrated in AUTOSAR authoring tools offered by diverse vendors. [30][31][32]

3.2 μ C or Processor implementation

Two elementary approaches can be taken for implementing C code on a processor-based platform. The one with broadest possibilities is using a RTOS, which provides not only a considerable hardware abstraction, but also the necessary scheduling and event management mechanisms which are necessary for big production-ready systems. Nevertheless, it can be a costly solution, especially when high integration, efficiency and integrity levels are required. An even higher abstraction -and cost-solution would be using AUTOSAR Authoring tools and a corresponding RTOS. [30][31][32]

The second approach is avoiding such a RTOS by taking a traditional programming approach with a *bare-metal* software. This can result especially attractive for smaller systems and prototypes, cases in which it might result to be a more straight forward solution with the additional benefit of avoiding the RTOS's overhead. Tasks can be organized using hardware interrupts which can provide sufficient control over priorities and preemption. In principle this requires deeper low-level knowledge and coding of hardware related functions. Nevertheless, this task can be greatly facilitated by using device-specific tools such as Hardware Abstraction Layer code generators -i.e. HalCoGen for Texas Instruments [9]- or Target Support Toolboxes for Simulink -available for certain Texas Instruments and Freescale devices [28][10]-. These toolboxes open the possibility to implement and run a complete control system without writing a single line of code.

3.3 FPGA Implementation

The following implementation solutions for FPGAs will be focused on following the MBD approach, seeking abstraction and flexibility. Nevertheless, this still provides as much as two interesting approaches, each of which has two sub-variants.

The first approach is purely Simulink-Based, by means of model-based (V)HDL -or Verilog-generation. Device-agnostic as well as device-specific code can be generated with the Mathworks's HDL-coder package, basing on generic Simulink blocks. This certainly boosts the design reusability, although it is limited in part by a series of restrictions of supported blocks and functions. An alternative (V)HDL/Verilog production approach is to use special Toolboxes with device-specific blocks offered by the manufacturers. At the cost of migrating the

diagram from generic blocks to these special blocks and accepting another series of limitations, efficient and well controllable code-production can be expected with these solutions.

The second approach relies on the principle of reusing C code which can be inherited from the Processor workflow. One possibility is to emulate a processor core on the FPGA by using Softcores. Such solutions are provided for instance by Altera and Xilinx which offer their own Softcore Processors -Nios II [33] and MicroBlaze [34] respectively- and also highly capable ARM cores amongst others [35]. In certain cases this can be a convenient alternative for executing C code and avoiding having to use a SoC or dedicated microcontroller, as could possibly be the case for communication functions. Nevertheless, for computationally demanding and time-critical functions, the sequential execution will still be a major drawback.

The alternative solution for the approach of implementing C code on a FPGA, is using High Level Synthesis (HLS) tools to translate C/C++ code into (V)HDL or Verilog hardware description formats. This avoids the inconvenient seen in the previous methods as were the need to re-design models, functionality limitations and speed issues, although it might arise a concern regarding to efficiency in comparison to conventional -more tedious- FPGA programming. Xilinx offers its own tool for this task which can seamlessly be integrated into their tools [36], and third party tools capable to address other FPGAs are also available.

Finally, it must be noted that the implementation for SoC devices will not be treated separately, as this would mainly combine the considerations of both the previous platforms. When developing on a SoC, both parts can basically be treated as independent elements with very capable interfaces and interaction mechanisms among them. These can be integrated using the same vendor tools as for the FPGA implementation. This will also be seen in section 4.3.

3.4 Addressing safety-critical aspects

Many safety and integrity related solutions have already been discussed in previous sections, both regarding hardware and software mechanisms.

In order to address the integrity aspects related to the ISO-26262, adequate software development and implementation solutions are also required. It is important to keep in mind that -apart from the hardware and software mechanisms in runtime-integrity must also be ensured within the toolchain. This needs to be qualified or certifiable in order to

ensure that none of the functionalities validated on the high MBD level is incorrectly propagated to the hardware. [37]

Manufacturers address this by providing a series of resources to support certification of their products, with specific devices and tools, and the corresponding manuals, safety reports (Failure Modes, FMEDA, etc.) and other support material [see sections 3.3 and 3.4]. Obviously, a variety of third party tools are also offered on the market, as some of the mentioned in section 3.

Also RTOSs are of great relevance in this topic, as they enable the implementation of partitioning, virtualization and other mechanisms.

Safety, integrity and qualification related topics are pointed out but cannot be deeply analysed in the scope of this work. It must be certainly considered that powertrain components are subject to functional safety requirements regarding to the ISO-26262[38], as will also the ADAS systems [17]. Yet, this does not mean that every single function needs to meet such requirements. An illustrative example of this can be seen in an active ADAS based on computer vision, where the decision making functions are subject to ASIL-D, but the image processing task can be relaxed down to QM level. [17]

4 Presented solution

In this section, the highly integrated control unit design will be developed and its individual components will be explained. The reasons and criteria followed to choose the specific combination of embedded platform and development workflow will also be discussed.

4.1 Platform selection

Starting from a general image of the main characteristics of the system as a whole to be integrated, the first step is to choose a platform type. Being the objective to develop a control unit which integrates several controllers of different nature, it certainly needs to be a versatile platform offering high capacity and performance. High energy efficiency is also a positive point and good communications and interfacing capacities are a must. Suitability for automotive applications is obviously also a requirement, as well as compatibility with high-level development tools. Table 4 presents a selection matrix with an evaluation of the main criteria as discussed. The values represented by stars are based on the information from sections 2 and 3. The cells with

less stars representing insufficient or borderline capabilities are marked in grey.

Table 4: Platform Type Selection

	uC	FPGA	uC + FPGA	SoC
Speed	*	***	***	****
Capacity	***	**	****	****
Integration	**	**	**	****
Comunic.	***	**	****	****
Efficiency	**	***	**	***
Costs	****	***	**	***
Workflow	****	***	**	***

This table 4 confirms that SoCs are a convenient platform to be used, as it provides good aptitude and fits in the context discussed in the introduction. Focusing on the SoCs from Table 4 which are inside the “industry-suitable” cost order of magnitude, the devices from Altera and Xilinx are selected due to their better specifications and development resources. For the implementation of this work, in general term Altera’s and Xilinx’s products could be considered as equally valid, especially regarding to hardware. Nevertheless, Xilinx’s development tools have been considered as more convenient for this paper, especially due to the HLS, meaning that a Xilinx Zynq platform will be used, specifically the ZC702 board offered by Xilinx with a Z7020 SoC will be used. This board does not mount an automotive-grade variant, but this in practice is irrelevant to the demonstration purposes of the developed solution.

4.2 Integrated Elements

This section will present the different components that have been integrated on the selected platform, addressing on how it will be distributed among the parts of the SoC (processor or FPGA). Figure 1 shows a simplified diagram that emphasises the hierarchy of the implemented control system.

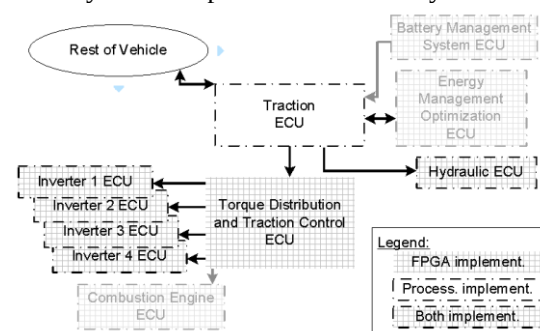


Figure 1: Conceptual architecture of proposed solution

4.2.1 Traction-ECU

In the presented scheme, the most significant output that the Traction-ECU provides to the rest of the powertrain would be the overall torque demand basing on the driver's pedal input (or cruise control's request). Logically, a significant amount of other signals -such as commands or other set-points and information- will also be provided to the rest of powertrain components. This comes from the great responsibility that the Traction-ECU actually carries in the powertrain domain. It can be seen as the central governing element supervising each of the other components and acting as an interconnecting hub. On a production street vehicle this will involve enormous amounts of logic and state machines, diagnostics and parameters to be taken into account. Obviously, this involves proprietary designs which cannot be addressed in this work. In this work a simplified control software has been set up. The control and supervision functions cover the functionality required for commissioning a working electric vehicle prototype. In such projects, additional functionalities which would usually require a dedicated ECU -for instance the traction control- are often included as a sub-function inside the Traction-ECU itself. This simplification has been avoided in this work, and the Traction-ECU is considered as a separate ECU (see section 4.2.2.) The functionalities implemented in the Traction-ECU of the demonstrator include: centralization of feedback signals, status monitoring of powertrain components, powertrain-level parameter handling, state-logic and operation modes selection, high-level diagnostics, driver input processing, information output processing, set point calculation and limitations.

The functionality has been programmed in Simulink using functions such as stateflow charts, lookup tables and conditional logic in different subsystems. They have been divided in two subfunctions according to their criticality. Using code generation, a function to be called with different priority interrupts has been created for each of the previous tasks. The cycle times have been fixed at conservative values considering the relatively slow physical time constants involved and the speed of the implemented CAN protocol: 2.5ms for the higher priority tasks function, 5ms for the lower.

Considering these timings and vast amount of logical structures and parameters, these functionalities are best suited to be implemented on a processor.

4.2.2 Torque-Distribution-ECU and Traction-Control-ECU

This ECU combines what typically could be distinguished as two independent ECUs: a Torque-Distribution-ECU and a Traction-Control-ECU. The reason is that with the enhanced functionality of the design that has been developed, an increased dependency exists among these two functions. In a simpler architecture, the Torque-Distribution-ECU would determine the optimal torque distribution without considering the applicable torque limitation (see approach A in figure 2). In such a topology, the next component in the torque set-point propagation chain would be the traction control system. This will apply a limitation to the torque output of any wheel suffering from excessive slip. This truncation of the optimized solution provided by the Torque-Distribution-ECU would notably degrade the quality of the final result. The point is that optimization algorithms typically obtain a set of nearly-optimal solutions -often referred to as Pareto front [39]-. The concern is, that an alternative nearly optimal solution might exist which would not require any traction control limitation, therefore being a better solution than an initially better solution which is later limited.

A possible solution could be the Traction Control ECU to first calculate the maximum value for each wheel and then provide this as an input to the Torque-Distribution-ECU (see approach B in figure 2). This would be convenient for a simple traction control algorithm. But an objective of this work is to enable exploiting the FPGA's great computing power to enhance the torque distribution and also the traction control by the means of advanced algorithms -involving dynamic models or machine learning-. Considering that the calculations needed for both ECUs would be similar, implementing redundant calculations in separate systems would not be an efficient solution. Therefore, combining both functions into a single device has been chosen as the best scheme (approach C in figure 2).

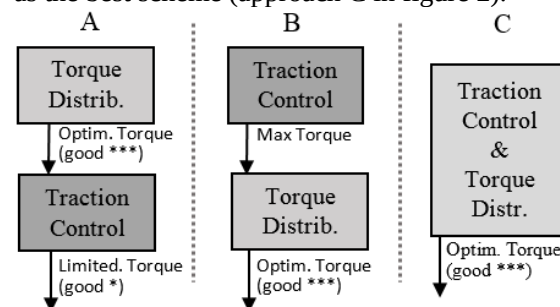


Figure 2: Different Torque Distribution and Traction-Control-ECU approaches

Regarding to the mentioned advanced algorithms for available grip and vehicle dynamics estimation, these are proposed to be implemented by the means of Neural Networks (NN). Simulation based experiments have been conducted obtaining promising results for variables such as normal force and available grip on the wheels. Nevertheless addressing this would lead to an extensive discussion which cannot be included in this paper. Still, an impression of the feasibility of such algorithmia is illustrated by implementing a NN through a Simulink block diagram following the formulation of a NARX architecture. The total size of the Neural Network of 85 Neurons.

In this way, the complete functional baseline together with the structure is provided in this work.

4.2.3 Energy-Management-Optimization-ECU

This ECU is treated as a special case in this work. Firstly, simple energy management strategies based on rules and low complexity equations would not need to be implemented in a separate ECU -they could be implemented in the Traction-ECU itself-. But -similarly to the torque distribution algorithms- elaborate modelling and optimization strategies - such as driving cycle prediction- might use machine learning techniques and require considerable amounts of computational capacity, although they have been proven to be implementable in real-time even on typical automotive microcontrollers [40]. The requirements and implications of such algorithms have been considered in the conception of this work, but will not be implemented in this first stage. In fact, in a related work, a SVN machine learning algorithm was implemented for gear shifting optimization [41]. Also in that work, the high capacity such algorithms require can be seen, as instead of implementing the whole Gearbox-ECU in a single microcontroller, it needed to be distributed in two ECUs. [41]

4.2.4 Inverter-ECU

The inverters are another component that very clearly will show the benefits of migrating the execution of their algorithmic core to an FPGA. Firstly, increasing rotational speeds -often due to motor downsizing- can represent a serious challenge to implement fast current loops on the microcontrollers, as shorter sampling times of the controller are required in order to correctly follow and control the waveforms. Secondly, the need for more processing power definitely appears when more than one electrical machine (motors and

generators) are to be controlled. This commonly is covered by having a dedicated ECU for each inverter, but could be addressed in a much more efficient manner integrating several control loops into the same ECU by exploiting the intrinsic parallelism of the FPGAs. Both points will be addressed in this subsection.

For this work, a Permanent Magnet Synchronous Motor with 6 pole pairs (N_{pp}) and a maximum speed of 7500rpm will be used. It will be driven by a two-level inverter operating at 16 KHz asymmetric PWM. This means a switching period of 62.5 μ s and control period ($T_{control}$) of 31.25 μ s.

At higher speeds the fundamental current frequency increases, meaning that the controller will get fewer samples per waveform period. In this case eq. 1 provides the number of samples, which is a good value for the Field-Oriented-Control (FOC) that is chosen to be implemented, considering that around 30-40 samples are a common practice value often taken as reference.

$$N = \frac{1/T_{control}}{N_{pp} \times f_{rot}} = \frac{1/31.25\mu s}{6 \times 7500/60 Hz} = 42.6$$

Nevertheless, this shows that sampling times can certainly be very tight and should not be increased, which can be a problem even for single motor controllers. Practice has shown how typical 32 bit microcontrollers in the 100-200Mhz range in Inverter-ECUs can struggle to execute all their tasks in the given cycle time. This issue inevitably gets accentuated when more elaborate and refined control methods are to be used, as these often require more complex algorithms.

FPGAs are capable of executing a FOC motor control faster than the cited microcontrollers. A low range FPGA can execute a standard FOC in around 2 μ s [42]. And the biggest advantage resides in their inherent parallel processing capabilities. Adding several simultaneous FOC loops in parallel will basically only increase the resources usage, but not the time.

In this work, the complete inverter functionality has been designed to be implemented in two tasks running at different cycle times. The *PWM-Task* is in charge of the current control, which means it will be running at $T_{control}$. This task includes a series of diagnostics and protection mechanisms which require fast reactions, as for instance overcurrent protections. All the necessary signal and value adaptation operations are inevitably also be included, as well as some basic and relatively lightweight functional logic.

The second task, or *Slow-Task*, has been specified to run at a conservative period of 2.5ms, to match

the Traction-ECU. It will execute a series of functions like less time-critical diagnostics, vector-control set-point calculations and limitation, as well as complete state-logic.

These tasks have been developed in Matlab-Simulink, each being modelled as a subsystem to be generated C-code from. In this case, dSpace Targetlink has been used. This controller code has been successfully implemented and tested on a motor testbench, as seen in Figure 3.

While *Fast-Task* is implemented in the FPGA, the *Slow-Task* is better suited for the processor core due to the lower timing requirements and bigger size. An alternative approach would be to distribute the *Fast-Task* itself between the two SoC parts, establishing an accelerator-like scheme, but considering it is relatively lightweight, the added complexity is not considered as worth it.



Figure 3: Demonstrator test-setup

4.2.5 Hydraulics-Pump-ECU

The implications and development for this controller basically follow the description of the Inverter-ECU from section 4.2.4 and will therefore not be discussed in order to avoid excessive redundancy. Basically, the implemented Hydraulics-Pump-ECU is a slightly simplified version of the Inverter-ECU, for instance, basing on a predefined set of operating points.

4.2.6 Combustion-Engine-ECU

The ECU in charge of controlling a hypothetical combustion engine -which is outside the field of expertise of the author- is considered in the design of the architecture, but is not included in the actual implementation. Nevertheless, for the case of a Hybrid Vehicle which contains such an engine, the presented solution could support its inclusion. In such a case, it should presumably be distributed using a similar topology as the Inverter-ECU described in section 4.2.4. **Error! Reference source not found..** The ignition control and fast loop would be inside the fast task running on the

FPGA, whereas the control logic would be in slower processor-based tasks.

4.2.7 Battery-Management-System-ECU

Similarly to the case of the Combustion-Engine-ECU from section 4.2.6, this component is considered in the design architecture but not included in the actual implementation. Although it is not an optional component as in the previous case, it has a lower integration dependency -and benefit- than the other powertrain components, and is often provided by the battery manufacturer.

4.3 Workflow and Implementation

In this chapter the final development tool setup that has been established for the desired workflow and implementation methods will be described.

A key point was to have a model-based design workflow capable of providing sufficient abstraction for a smooth integration of diverse software components on different platforms. Adequation for automotive industry requirements is important. Efficiency is also to be taken into account, but so is also keeping a reasonable amount of flexibility and code reusability. This means that a sufficiently generic baseline solution is to be provided, avoiding creating unnecessarily deep dependencies on device-specific tools.

The required criteria have been satisfied by using Mathworks Matlab-Simulink (r2014b) for the top-level control design and simulation. As alternative to Mathworks' Embedded Coder, dSpace Targetlink (v3.5/4.0) has been used for production-ready C code generation.

The SoC integration has been done with Xilinx's tools (2014.4) using the Vivado and SDK packages, for the FPGA/SoC integration and the processor's C programming, respectively. As an alternative, for certification purposes, the legacy Xilinx ISE package could have been used [14]. The conversion from Simulink to VHDL has been done using the generated C code as intermediate step, translating it to VHDL afterwards with Xilinx's HLS software. This strongly mitigates the gap between the processor and FPGA. Alternatively, third party HLS tools could be used, making this workflow equally usable for the alternative SoCs.

The SoC has been set up to run on a *bare metal* implementation, basing on a series of differently prioritized timer interrupts and interrupt requests between the processor and the FPGA. This was doable with reasonable ease thanks to the reduced hardware IO requirements and the design resources provided by the manufacturer. A single core has been used, leaving the future spare for additional

functions, supervision or redundancy. In spite of not having used any RTOS nor AUTOSAR platform, the developed functions are perfectly suitable to be included in any of both. In fact, for future evolutions of this design and for fitting bigger architectures, moving it to such an RTOS or AUTOSAR compatible system would be of interest. As already described in section 3, Simulink and Targetlink greatly facilitate this integration and such solutions are available for the discussed SoCs.

Higher optimization levels might probably be achieved by using the device specific toolboxes, but this would require the generic Simulink blocks to be replaced by vendor-specific blocks, which would eliminate the generic character of the design and require additional effort when porting the solution to another platform, or even when redistributing functions between the processor and FPGA parts of the SoC.

5 Results

The components described in section 4.2 have been integrated following the workflow established in section 4.3 and have been implemented in the Xilinx Zynq SoC platform selected in section 4.1. By loading it on the ZC702 -plus adequate conditioning circuitry- a working demonstrator has been obtained.

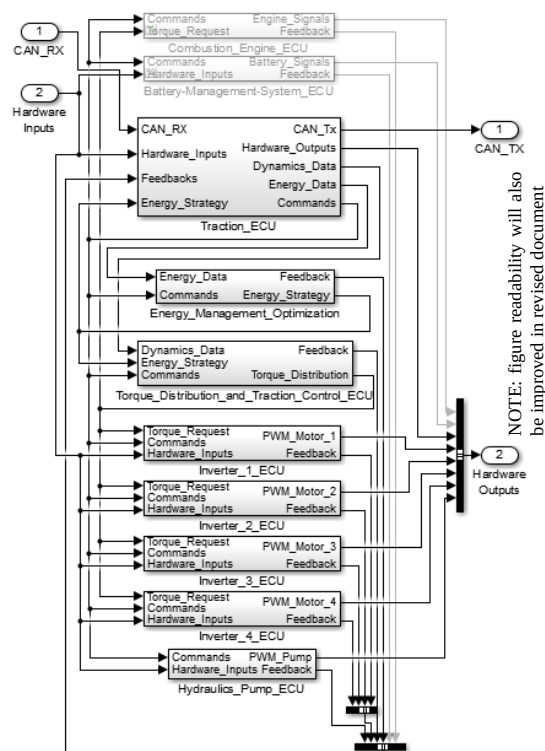


Figure 4: hiECU diagram in Simulink

Figure 4 shows the diagram of the completely integrated control system model setup in Simulink - with 4 inverter controllers- which has been used for controller development, simulation and code-generation. It provides a more detailed view of the architecture and the interaction between components can be observed. Many of the interface buses contain a very significant amount of signals and some of the subsystems unfold algorithms of great complexity.

The most illustrative results of the integrated components are summarized in Table 5 following their presentation order in section 4.2. Memory usage (of the ZC702 board) and execution time are shown for the processor part, and resource usage and latency for the FPGA.

As commented in section 3, these values depend on a variety of implementation related aspects. An important factor are the optimization settings, which have been configured to optimize the most problematic aspects for each platform. This means, optimizing for speed on the processor part, and optimizing for hardware occupation in the FPGA.

Table 5: Summary of Implementation Results

Implem.	Processor		FPGA	
	RAM/ Progr. [%]	Exec. Time [ms]	Resou. usage [%]	Latency [ms]
Traction	17 / ~0	0.12	-	-
Torq. Dist.	-	-	22%	0.015
Inverter	4x10/~0	4x0.007	4x12%	0.003
Hydraul.	8 / ~0	0.005	11%	0.003
Total	65/ <0.1	0.153	81%	n.a.

The results clearly show that memory should be no major concern on the processor part, as even the integrated memory is sufficient and DDR memory can also be used. The execution times, are sufficiently fast for the millisecond order of magnitude tasks. Regarding to the FPGA, here the speed is of no concern, but resource usage, especially the LUTs, is considerable. Nevertheless, here considerable optimization margin is still open. A significant difference is expected to be achieved by reusing algorithmic and mathematic functions of the FOC, of which five are included and synthesized independently. Furthermore, converting the FOC algorithm to fixed point should reduce resource usage by more than half [42]. And in any case, if bigger capacity is needed for integrating more components or enhancing their sophistication, the Zynq SoC can be seamlessly upgraded.

Note: Further results, diagrams and images, could not be fitted in this paper will be provided during the oral presentation at EVS28.

6 Conclusions

After providing a relevant insight into the implications of upcoming automotive control unit applications, their requirements and a series of hardware platforms and development solutions, this paper has presented a novel highly integrated Electronic Control Unit -hiECU- design.

The designed hiECU architecture has been developed as described in section 4: integrating different powertrain ECUs into a single SoC using a MBD workflow completely based on code generation. This has been implemented on a demonstrator setup with a Xilinx Zynq SoC.

Some of the most challenging aspects of the next-generation of electric vehicles have been addressed with the presented solution, discussing topics such as architecture, functionality, costs, efficiency and safety/integrity.

The presented hiECU has proven to be not only an efficient solution with great development possibilities, but it also is a highly capable system suitable to incorporate complex and highly demanding advanced control algorithms.

Results have shown the benefits of the implementation on FPGAs of time-critical control loops, which has been discussed in detail. The selected platform's strong intrinsic parallelism has shown to be very suitable for highly integrated control concepts as the one presented, as well as for multi-motor vehicles in general.

Development effort is reduced by the means of a high-level workflow and the simplification of the interfacing and communication issues by combining components with relevant dependencies into a single device.

In essence, apart from the highly efficient hiECU itself, this paper shows that the used SoC + MBD workflow solutions are adequate for powertrain control and automotive applications in general. This emphasizes that a mindset shift towards such hybrid highly integrated architectures with a strong parallel character is worth it, and that approaching their implementation with a high-level methodology is more than convenient.

7 Future Work

Keeping the established hiECU architecture, methodology and controller functions, an evolution of the presented implementation will be

done by including a RTOS in order to bring it a further step closer to a production-ready solution.

A detailed analysis of optimization aspects is planned to be performed, benchmarking performance and resource usage with different implementation approaches.

Regarding controller algorithmia, an enhanced Torque-Vectoring controller (section 4.2.2) will also be developed on. This should include vehicle dynamics models to be executed on the FPGA and the applicability of applying predictive algorithms and machine learning techniques will be researched. Furthermore, intelligent optimization controllers covering the broadest possible powertrain components will be researched.

In a parallel line, further considerations regarding to next-generation ADAS applications will also be studied.

References

- [1] Freescale, "Future Advances in Body Electronics." 2013.
- [2] A. Roychoudhury, *Embedded Systems and Software Validation*. M. Kaufmann, 2009.
- [3] K. Morris, "Driver's Ed for FPGAs," 06-Jan-2015, <http://www.eejournal.com/archives/articles/20150105-audi>, accessed on 08-Jan-2015.
- [4] "Infineon.", <http://www.infineon.com/>, accessed on 31-Jan-2015.
- [5] "Texas Instruments.", <http://www.ti.com/>.
- [6] "Freescale.", <http://www.freescale.com/>, accessed on 31-Jan-2015.
- [7] A. R. Weiss, "Dhrystone Benchmark, White Paper," *EEMBC Certification Laboratories, LLC (ECL)*, 2002.
- [8] "EEMBC.", <http://www.eembc.org>, accessed on 24-Jan-2015.
- [9] "TI - HalCoGen.", <http://www.ti.com/tool/halcogen>.
- [10] "Freescale, Motor Control Development Toolbox.", http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC_TOOLBOX, accessed on 30-Jan-2015.
- [11] "Xilinx, What is an FPGA?", <http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm>, accessed on 15-Jan-2015.
- [12] "Latest FPGAs Show Big Gains in Floating Point Performance," *HPCwire*, http://www.hpcwire.com/2012/04/16/latest_fpgas_show_big_gains_in_floating_point_performance/, accessed on 22-Jan-2015.
- [13] "Altera.", <http://www.altera.com/>, accessed on 31-Jan-2015.

- [14] "Xilinx.", <http://www.xilinx.com/>, accessed on 31-Jan-2015.
- [15] "Top FPGA Companies For 2013 - SourceTech411.", <http://sourcetech411.com/2013/04/top-fpga-companies-for-2013/>, accessed on 31-Jan-2015.
- [16] "Xilinx All Programmable Functional Safety Design Flow Solution Product Brief - safety-guidelines.pdf.", http://www.xilinx.com/publications/prod_mktg/safety-guidelines.pdf, accessed on 12-Jan-2015.
- [17] Altera, "A Safety Methodology for ADAS Designs in FPGAs." 2013.
- [18] "TÜV-Qualified FPGAs for Functional Safety Designs.", <http://www.altera.com/end-markets/industrial/functional-safety/ind-functional-safety.html>, accessed on 29-Jan-2015.
- [19] "Xilinx Isolation Design Flow.", <http://www.xilinx.com/applications/isolation-design-flow.html>, accessed on 12-Jan-2015.
- [20] Xilinx, "Spartan-6 FPGA Dual-Lockstep MicroBlaze Processor with IDF." 2012.
- [21] "Comparing Peak Floating Point Claims.", <http://www.hpcwire.com/2014/06/17/comparing-peak-floating-point-claims/>, accessed on 22-Jan-2015.
- [22] "Exploring the Floating Point Performance of Modern ARM Processors.", <http://www.anandtech.com/show/6971/exploring-the-floating-point-performance-of-modern-arm-processors>, accessed on 24-Jan-2015.
- [23] K. Kim and K. Choi, "SoC Architecture for Automobile Vision System," in *Algorithm & SoC Design for Automotive Vision Systems*, J. Kim and H. Shin, Eds. Springer Netherlands, 2014, pp. 163–195.
- [24] "Microsemi.", <http://www.microsemi.com/>, accessed on 31-Jan-2015.
- [25] Altera, "Cyclone V Hard Processor System Technical Reference Manual." 2014.
- [26] "AUTOSAR, Home.", <http://www.autosar.org/>, accessed on 30-Jan-2015.
- [27] G. Sandmann and R. Thompson, "Development of AUTOSAR software components within model-based design," SAE Technical Paper, 2008.
- [28] "Mathworks.", <http://www.mathworks.com/>, accessed on 30-Jan-2015.
- [29] "dSPACE.", <https://www.dspace.com/>, accessed on 30-Jan-2015.
- [30] "Vector.", <http://vector.com/>, accessed on 30-Jan-2015.
- [31] "ETAS.", <http://www.etas.com/>, accessed on 30-Jan-2015.
- [32] "Elektrobit.", <http://www.elektrobit.com/>.
- [33] "Altera, Nios II.", <http://www.altera.com/devices/processor/nios2/ni2-index.html>, accessed on 30-Jan-2015.
- [34] "Xilinx, Microblaze.", <http://www.xilinx.com/tools/microblaze.htm>, accessed on 30-Jan-2015.
- [35] "Altera, Embedded (soft) Processors.", <http://www.altera.com/devices/processor/additional/ipm-index.jsp>, accessed on 30-Jan-2015.
- [36] "Xilinx, HLS.", <http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html>, accessed on 30-Jan-2015.
- [37] A. Holmberg, "IAR, How to validate a build tool chain." IAR, 2013.
- [38] Hui Guo and et al., "Iso 26262 Impact on Development of Powertrain Control System," in *Proceedings of the FISITA 2012 World Automotive Congress*, SAE-China and FISITA, Eds. Springer Berlin Heidelberg, 2013, pp. 705–715.
- [39] J. Horn and et al., "A Niched Pareto Genetic Algorithm for Multiobjective Optimization," in *In Proceedings of the First IEEE Conference on Evolutionary Computation*, 1994, pp. 82–87.
- [40] M. Dendaluce and et al., "Microcontroller Implementation of a Multi Objective Genetic Algorithm for Real-Time Intelligent Control," in *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, Springer International Publishing, 2014, pp. 71–80.
- [41] M. Allende, "Advanced Shifting Control of a Two Speed Gearbox for an Electric Vehicle," presented at the EVS28, Korea, 2015.
- [42] Altera, "Optimize Motor Control Designs with an Integrated FPGA Design Flow, White Paper," 2012.

Authors



Dendaluce Jahnke, Martin: PhD. Candidate and Researcher at Tecnia Research and Innovation. Having gathered significant experience in alternative powertrain prototyping and implementation, he currently is working on different automotive applications of advanced algorithms

and their implementation on state-of-the-art platforms.