2014-32-0070
20149070
Published 11/11/2014

doi:10.4271/2014-32-0070
saepcelec.saejournals.org

Towards an Open Source Framework for Small Engine Controls Development

Paolo Gai, Francesco Esposito, and Riccardo Schiavi
Evidence Srl

Marco Di Natale
Scuola Superiore S. Anna

Claudio Diglio, Michele Pagano, Carlo Camicia, and Luca Carmignani
Piaggio & C SpA

ABSTRACT

The paper describes the components of an envisioned open source framework that supports several stages in the model-based development of two- and three-wheelers software controls. The proposed solution supports the runtime execution on an OSEK-compatible [8] real-time operating system for multicore platforms. The framework consists of a modeling and simulation tool (including hierarchical state machines) and a code generator for the development of the functional model of controls and the definition of their task implementation; an OSEK/AUTOSAR operating system and device driver stack; OS and I/O configuration tools. The platform has been released open-source under an industry-friendly license. Our framework is currently in use for the development of innovative two-three wheelers control systems at Piaggio. In this paper we describe the experience matured in the application development, the benefits and current limitations of the approach. In particular, the result of this study has been a demonstrator platform which includes a BLDC motor control written using the proposed framework.

CITATION: Gai, P., Esposito, F., Schiavi, R., Di Natale, M. et al., "Towards an Open Source Framework for Small Engine Controls Development," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 8(1):2015, doi:10.4271/2014-32-0070.

INTRODUCTION

The design and development of the electronic and control software architecture for the next generation of two wheelers is one of the major challenges in the development of small engine products.

The need for optimal fuel consumption and emission control, paired with strict safety requirements is pushing manufacturers to reconsider their electronic architecture, adding complex functionality with the risk of incurring significant costs for the integration of new systems and the possible redefinition of the electronics architecture.

These challenges have already been tackled twenty years ago by the automotive manufacturers, generating a set of standards in an attempt to improve productivity and quality by standardization, interoperability and competition on functional content. The result has been the development of distributed software architectures based on the CAN bus and the OSEK/VDX [8] standard, and more recently the AUTOSAR [4] infrastructure. A set of automotive standards are today available and can be used as an inspiration as they share common goals such as portability, reusability, and limited footprint. Unfortunately, the inherent complexity and license cost of complete commercial bundles makes the adoption of these automotive standards unlikely for low-cost engines.

Moreover, the new challenge in the field is the application of functional safety concepts (becoming a requirement for automotive systems, after the ISO26262 norm publication [11]), which are soon expected to become the standard also for motor-motive systems. Functional safety includes several measures and best practices to assist the maker in providing the highest achievable safety level. These measures have an impact on the design of new products, on their bench and road tests, as well as on their assembly, service and product lifecycles.

All these changes will have a non-negligible short-term impact on costs. Knowing that the concepts of functional safety apply to all the stages of the software and hardware development,

Piaggio & C. planned for a significant investment in the development of a platform to guarantee adequate safety levels for its vehicles.

The first step has been the development and assessment of an electronic platform based on new hardware and software components developed in accordance with functional safety requirements.

This paper presents the results of this initial development effort, aimed at building low-cost two and three-wheelers software applications. The proposed solution is composed of a set of open-source target software modules, supported by a set of design tools for the simulation and the efficient code generation of control laws and state machines. The idea is to enable manufacturers to add their own functional and (system-level) controls know-how easily, adapting the platform to various kinds of applications.

Being open-source is a major advance in the development of software for small engines: it guarantees the possibility to develop applications while retaining control of the intellectual property and avoiding the disclosure of proprietary code (thanks to the GPL2+Linking Exception license [3]), but still enables manufacturers to share code not contributing to product differentiation. Moreover, it allows to build products without recurring expenses.

The resulting software platform has been ported to the Freescale MPC5643L Leopard MCU, and is composed by a Bootloader, the ERIKA Enterprise RTOS, Peripheral Drivers and High-level communication protocols aimed at diagnostic and calibration support;

Special attention has been dedicated to the testing of the RTOS and of other part of the system, including MISRAC: 2004 compliance [5], and black box regression tests including the MODISTARC and OSEK/VDX certification tests.

Finally, the platform supports the integration with model-based development tools like Scicos and E4Coder [6]: a set of tools that can be used to model and simulate hybrid control algorithms, and generate code for embedded microcontrollers, in a similar fashion as the widely used Matlab and Simulink [7].

We believe that such an open platform has the potential to create a common low-cost execution platform that will help manufacturers in creating innovative control applications.

The following sections describe the choice of the open-source licensing model, and will provide a high-level view of the proposed platform. Afterwards, the microcontroller, bootloader, RTOS, Drivers and Model based Design tools are described in more details. Finally the current Piaggio demonstrator, composed by a BLDC motor control, is described in detail, including experiments and benchmarks.

Open-source Licences and Automotive software

The choice of using open-source software for the components of the platform has been driven by an analysis of automotive market trends and the evolution of software components development.

Automotive software architectures are becoming increasingly complex. The control architecture is transitioning from the federated networked infrastructure of the 90s to the integrated architecture concept, enabled by the AUTOSAR [4] standard. Market trends for infotainment systems are pushing for subsystem integration on multicore ECUs, with a widespread adoption of embedded Linux platforms.

In the AUTOSAR view, complexity is managed by cooperation at the level of the basic software and differentiation (or competition) on the application-level functional content. Today, ECU requirements are common across several car manufacturers, with a small percentage of notable exceptions that provide for product differentiation [9].

We believe that the small engine market will experience the same kind of evolution. In this view, open source software and an open platform represents the most radical form of collaboration in the development and exchange of basic software solutions. For this to happen, a crucial choice was the selection of the most appropriate licensing terms. In particular, the licensing term must allow the static linking of proprietary code in order to preserve the confidentiality of the application part (providing the real value in the differentiation of the product), while retaining the capability of cooperation and sharing on the open-source part. For this reason, the selected license is the GPLv2 with Linking Exception [3] for the code to be embedded in the target ECUs, and the EPL license [10] for the Eclipse plugins of the development environment.

PLATFORM DESCRIPTION

The project aims at the creation of a framework that support all the development stages for the design and implementation of controls for modern hybrid combustion small engines. The developed systems should include the powertrain controller for the endothermic motor as well as for the electrical motor hosted in two and three-wheelers.

Platform requirements include the capability of providing automotive-grade performance, without jeopardizing future requirements on safety, and access to the platform at an affordable cost for the tight 2-3 wheelers budgets.

The cornerstone of the run-time execution platform architecture is the open-source ERIKA Enterprise OSEK/VDX RTOS. The development framework is based on the open source Eclipse Modeling Framework, on which the RT-Druid open plugin provides for component customization and configuration. Complementing the runtime components and the configuration tools, we provided for an open source flow for model-based

development, based on an extension of the Scicos modeling tools [26] with a state machine modeler and efficient code generators [27].

The overall view of the runtime architecture is represented in Figure 1.

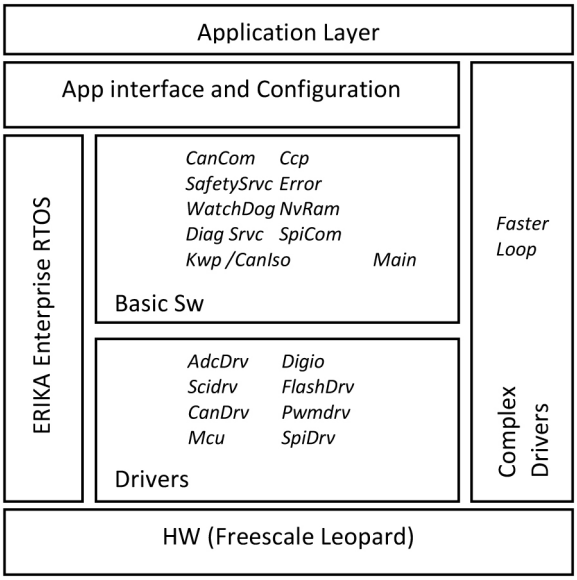

Figure 1. Components of the runtime architecture.

The main components are:

- The Application layer, which contains the functional part of the application;
- The Application Interface/Configuration Layer, which contains all the configurations and initial setup of the underlying Basic Software and Drivers;
- The Basic Software Layer, which provides application services to the main application, including the communication services (CAN, CCP [31], KWP2000 [32], ISO-TP [33]);
- The Drivers Layer, which abstracts the underlying hardware peripherals (ADC, PWM, …);
- The ERIKA Enterprise OSEK/VDX RTOS, offering operating system services;
- The Complex Drivers Layer, which contains a set of software components that for their complexity and criticalness cannot be described by a standard layer (in particular, the ADC/PWM coupling implemented with the CTU of the Leopard MCU);
- The Freescale Leopard MPC5643L MCU.

In addition to these layers, the Platform offers:

• A Bootloader, enabling CAN-based reprogramming, reconfiguration, boot, and system diagnostics.

The following sections will analyze in detail the Leopard Microcontroller, the Bootloader, the RTOS, and provide information on the Drivers and the Basic Software. The Complex drivers and Application Layer will be discussed in the context of the test application Section.

THE LEOPARD MICROCONTROLLER

The platform target for the hybrid engine project is the Freescale MPC5643L Leopard MCU. The microcontroller was selected because it may support safety-critical applications up to ASIL D level, offering an easier path toward future safety qualification in the motor-motive market.

The Leopard MCU offers two PowerPC CPUs, which can be configured to run in Lockstep (as safety measure), or in multicore mode (to boost performance), and offers fast ADC with PWM integration which are ideal for implementing electric machine control strategies.

The Flash memory of the MCU is 1Mb, organized on three address spaces:

1. Low address space (256 KB);
2. Mid address space (256 KB);
3. High address space (512 KB).

The MPC5643L flash memory is managed as follow (see Figure 2):

- Block L0 (0x0 - 0x00004000) hosts the boot key of the microcontroller. The MPC5643L has several boot sectors, which can be configured to run in VLE or BOOK E mode.
- Block L1, L2 and L3 (0x00004000 - 0x00020000) are dedicated to Non-Volatile-RAM (NVRAM) data.
- Block L4 and L5 (0x00020000 - 0x00040000) contain the boot code. The content of this flash area is explained in the section dedicated to the description of the Bootloader.
- Block M0 and M1 (0x00040000 - 0x00080000) contain the calibrations data.
- Block H0 and H1 (0x00080000 - 0x000C0000) contain the application code.

FLASH_BASE address offset	Use	Block	Size (KB)	Partition
0x0	Low Address Space BOOT KEY	L0	16	1
0x0000_4000	NVRAM	L1	48	
0x0001_0000		L2	48	
0x0001_C000		L3	16	
0x0002_0000	BOOT	L4	64	2
0x0003_0000		L5	64	
0x0004_0000	Mid Address Space CALIBRATIONS	M0	128	3
0x0006_0000		M1	128	
0x0008_0000	High Address Space APPLICAZIONE	H0	256	4
0x000C_0000		H1	256	
0x0010_0000 – 0x00EF_FFFF	Reserved			

Figure 2. Platform Flash Memory Map.

THE BOOTLOADER

The Bootloader represents the first application that starts at car key-on. It is located in Flash in blocks L4 and L5 (partition 2) for a total of 128 Kbytes.

At startup, the Bootloader verifies (using a CRC) the presence of a valid application located at a given Flash address. If the verification succeeds, the bootloader runs the application. In the unlikely event the CRC check fails, or no valid application is found in the Flash, or the bootloader is launched by the application for reprogramming, then the bootloader enters into its main control loop waiting for commands from the user by using the CAN interface. The communication between the host and the bootloader is based on a subset of commands of the KWP2000 protocol [32].

Since ISO26262 [11] clearly suggests the use of CRC mechanisms to enforce the reliability of a communication infrastructure, and therefore the safety of the entire system, the communication between the host and the bootloader (e.g. for ECU reprogramming) is based on multiple CRC checks, interleaving communication and write operations. The most relevant steps of the typical “communication and write cycle” can be summarized as follow:

1. Input data are provided to the Bootloader by high-level communication components, implementing the KWP2000 [32] protocol and ISO-TP transport layer [33]. Each message is part of a sub-block that the application intends to write into the Flash. Data sections are compressed to improve the communication performance;
2. The CRC is computed when the required number of messages is received (a compressed sub-block is characterized by a configurable number of messages). Optionally transmitted data can be compressed and/or encrypted.
3. After the first CRC check, another CRC check is performed on the uncompressed data. If this check is positive, the sub-block is ready to be written into the Flash memory;
4. Steps from 1 to 3 are performed for all the sub-blocks of the given application. Note that the size of the sub-blocks is a configurable parameter which varies from 256 to 4096 bytes;
5. When all the application sub-blocks have been received and successfully written into the Flash memory, another CRC check is performed for the entire application image in Flash. If this last check is positive, the application is valid and ready to be launched at the next power-on.

The whole process takes approximately 20/30 seconds for a typical Automotive application and it includes the application code and data, calibration data, and (if necessary) the application code and data for the second core (in multicore scenarios).

THE RTOS

Several RTOS and API have been evaluated for the Platform. The rationale of the choice is to focus on automotive standards to allow cost reductions, future extensibility, and functional safety enhancements.

Unfortunately creating a full AUTOSAR compliant stack or using a commercial offering was not a viable option due to the cost of a complete AUTOSAR solution. However, we wanted an API which could be ported to AUTOSAR with little effort, together with an affordable development cost. Among the possible options, we chose the ERIKA Enterprise Project [2], which is the first open-source RTOS that has been certified as OSEK/VDX compliant [13]. ERIKA is currently in use by several automotive and white goods companies, including Cobra Automotive and Magneti Marelli. Its openness, its industrial usage, together with the other features listed in the following sub-sections led us to select ERIKA Enterprise for our Platform.

ERIKA Enterprise Main Features

ERIKA Enterprise implements the OSEK/VDX API (all four conformance classes, BCC1, BCC2, ECC1, ECC2, plus the OSEK COM CCCA and CCCB), which is the basis for the AUTOSAR OS specification [14]. The RTOS also includes optional earliest deadline first and resource reservation scheduling implementations, developed in collaboration with The ReTiS Lab of the Scuola Superiore S. Anna in Pisa as well as other universities [15, 17, 18]. The RTOS supports fixed-priority scheduling [16], immediate priority ceiling [19] and stack sharing [20], allowing for bounded and predictable execution and blocking times with minimal RAM usage.

ERIKA supports a wide range of microcontrollers (from small 8 bit MCUs to Multicore Freescale PPC and S12, Infineon AURIX, ARM Cortex MX, and many others [21]), compilers and ORTI-enabled debuggers (like Lauterbach Trace32 [22] and iSystem winIDEA [23]). ERIKA Enterprise is currently used by a growing numbers of Universities and Research projects, including the P-SOCRATES FP7 project [12] and other ITEA / FP7 projects.

RT-Druid Configuration Tool

ERIKA Enterprise is configured using the RT-Druid Eclipse plugin. RT-Druid allows management of the OIL configuration file inside a customized editor, and then produce the ERIKA Enterprise configuration files as well as the makefiles needed to compile the final application. RT-Druid also generates the ORTI files needed to obtain kernel-awareness. Finally, RT-Druid is fully scriptable to allow a complete integration inside custom build systems.

Multi-core Support

ERIKA Enterprise supports multicore MCUs since its inception in 2002. The main idea is that tasks are statically partitioned into the various cores, each one running a copy of the RTOS. In this way tasks can execute efficiently, using interprocessor IRQs and atomic operations only when needed (for example, when task activations must be sent to the other cores). Multicore support is currently available for various MCUs, including the Freescale MPC5643L used in the Platform.

Certifications, Standards, Testing

ERIKA Enterprise has been certified OSEK/VDX for ARM Cortex M4F (TI Stellaris) in August 2012 and for Infineon Tricore 26x in February 2014 [13]. The certification of the kernel meant the execution of a large set of regression tests on a specific MCU with a specific compiler, testing the API specification as well as the expected scheduling behavior.

ERIKA Enterprise is not an official AUTOSAR OS, but it implements an API similar to the one proposed by the AUTOSAR consortium for what concerns Memory protection, Multicore, Stack Monitoring, and Schedule Tables.

The kernel has been checked for MISRA-C compliance in collaboration with the Tool & Methodologies team of Magneti Marelli Power Train. In this context, the Erika code has been verified through the use of Flexelint (ver. 9.00h) and Lin (ver. 7.10), using a specific configuration testing a subset of the MISRA C requirements.

The kernel undergoes continuous testing using a regression test suite implemented using Jenkins [24]. Each version which passes all regression tests is then uploaded directly on the project web site.

DRIVER AND BASIC SW

In addition to the RTOS, the Platform includes a set of peripheral drivers and Basic Software modules. Those modules have been implemented taking into account the need to have a simple and configurable implementation. The drivers do not formally comply with the AUTOSAR standard API, but are inspired by the AUTOSAR concepts, providing a stable API and the static configuration of data structures that provide for the specific configuration of the devices.

The RT-Druid Eclipse plugin provides for the configuration and the automatic generation of the driver configuration files. The platform device specific information is encoded in an XML file, defined according to an EMF metamodel. The XML file is imported in Eclipse/RTDruid and defines an internal model of the devices, with their data and control registers.

RTDruid parses the device description file, detects resource conflicts, and automatically generates the data structures and the definition that configure the devices for use in the specific application. The code generation is performed by the open source Acceleo tool, that provides an implementation of the OMG standard model-to-text template language.

This feature is currently available as a free plugin as part of the Evidence EFORMS [34] but not yet available as open source. However, the solution is entirely based on open standards and tools.

The following paragraph will shortly describe the features of the main components of the Device Drivers and Basic Software.

For the PWM Driver, we selected a simplified implementation to control the two PWM modules of the MPC5643L. Each PWM module has four PWM devices, and only a subset of them have been used in the test application.

The Analog to Digital Converter (ADC) is a 12 bit Successive Approximation Register (SAR) ADC with a mixed capacitive/resistive DAC. A conversion can be triggered by software or hardware (the hardware method is the one used in the test application below). Currently the system is able to start the acquisition of 8 different ADC signals when the internal counter of the PWM0 module matches its maximum value.

The Can Driver uses the Freescale FlexCan peripheral in the MPC5643L MCU. The Can Driver can be statically configured, and can raise interrupts on message receive/transmit. At higher level, Basic Software modules have been implemented to provide a minimal implementation of the CCP [31], KWP2000 [32], and ISO-TP [33] protocol.

The flash driver component (FlashDrv) provides an easy-to-use interface to access the MPC5643L flash without the need to know flash driver write/read mechanisms. The engine used by FlashDrv to perform operations in flash is based on the SSD Driver provided by Freescale. The flash driver can be copied into SRAM at the initialization phase to allow the driver to operate on the entire flash memory space, without any risk to erase/compromise itself while running.

The NvRAM module is part of the Basic Software and is responsible for managing all the variables that must be saved after the power-off (when the ECU is turned off). These variables can be the serial number, hardware identifier, odometer, etc. These variables have a shadow version in SRAM that is saved to flash when the ECU is powered-off.

MODEL BASED DESIGN AND AUTOMATIC CODE GENERATION

The development of the application functionality is supported by a set of open-source tools realizing a Model-Based Design (MBD) toolchains. Model-based development is today widely used in automotive and aerospace, allowing the decription and simulation of the physical control system, including the

(electric) engine dynamics and the controller functionality. The automotive market today largely relies on the Matlab/Simulink/Stateflow toolset from Mathworks [25]

Automatic code generation techniques are then used to generate a software implementation that is equivalent to the model, reducing the possibility of adding errors during the manual coding stage.

Our proposed MBD toolchain has been implemented by leveraging and extending the open source modeling and simulation environment ScicosLab [26]. ScicosLab is based on the Scicos simulator, with a number of available solvers and support for Modelica implicit blocks.

Scicoslab provides for the simulation of the continuous dynamics of physical systems, but lacked the capability of representing the finite state models of hybrid systems and the state-dependent behavior of automotive controllers.

In addition, the code generation pluging available with ScicosLab was limited to the generation of simulation-oriented code to be executed on the host and did not have the performance and size required by embedded production code.

This is why we extended ScicosLab with the E4Coder [27] toolbox based on the use of the open Eclipse EMF metamodeling and code generation tools and standards. E4Coder is commercial and not open source, but some of its components (the state machine modeler) are freely available with reduced functionality.

The objective of E4Coder is to provide an alternative modeling framework with a dedicated environment for modeling extended synchronous finite state machines and production-level code generation at a fraction of the cost that is currently required on the market. ScicosLab, together with E4Coder is currently being evaluated for the modeling and implementation of the controls functionality, as described in the following Sections. In detail, the E4Coder toolbox consists of the following environments.

- *SMCube*, a tool for modeling, simulation and code generation of Finite State Machines (FSM). SMCube supports hierarchical and concurrent FSMs, with support for both simulation and code generation of the designs.
- *E4CoderCG*, a tool for the C code generation of ScicosLab models. E4Coder supports the possibility to generate the task bodies implementing the software architecture of the generated code, with support for multi-rate designs as well as code generated for ERIKA Enterprise, Bare Metal, Linux, RTAI and Windows targets.
- *E4CoderGUI*, a GUI prototyper based on Qt, which allows to construct mockups of simple graphical panels, and to generate Qt code for Embedded Linux targets.

THE TARGET AND TESTBENCH APPLICATION: AN ELECTRIC MOTOR CONTROL

The platform has been evaluated by Piaggio & C. on the implementation of a control algorithm for a brushless motor. In particular, the application has been made of a set of ERIKA Enterprise concurrent tasks implementing the control algorithms. Those tasks use the device drivers and Bootloader which have been described in the previous sections, allowing a proper evaluation of their performance. This section describes a subset of the experiments on the real testbench.

The control algorithm is a standard proportional integral current regulator with Park and Clark transformation, which uses a Space Vector Modulator to transform the output of the regulator into timing pulses applied via the PWM peripheral. The control loop has been implemented inside a Category 1 ISR with a period of 50 µs. Figure 3 shows the functional structure: the ADC peripheral provides information of the currents flowing inside the motor, as well as the sensed position of the rotor. The currents then are processed using a cascade of Clark/Park transformations, the latter using an estimation of the Electric Angle.

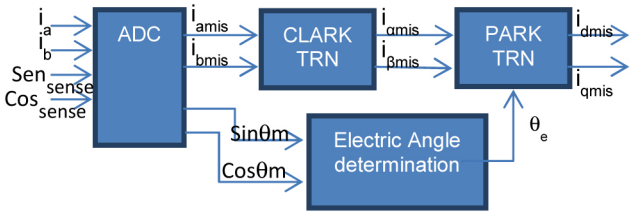

Figure 3. Test control algorithm, from the ADC to the Clark/Park Transformations.

Figure 4 shows how the results of the Park transformation is compared with the reference currents to obtain an error estimation which is then regulated by two PI algorithms. Afterwards, the Inverse Park/Clark transformations are applied, providing the input to the Space Vector Modulator, which computes the PWM outputs to be applied to the motor. More details about the implementation of this algorithm can be found in [28,29,30].

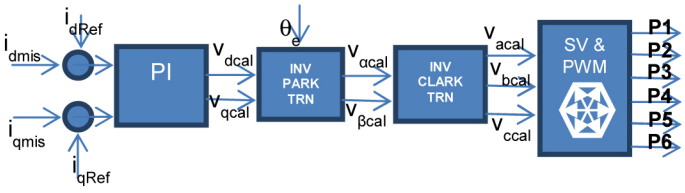

Figure 4. Test control algorithm, from the PI to the Space Vector and PWM output.

PWM/ADC Timing and Synchronization

One of the main issues that needed to be solved when implementing the 50 µs control loop is that in PWM-driven systems it is important to schedule the acquisition of the state

variables with respect to the PWM cycle. State variables are obtained through appropriate programming of the following input peripherals: ADC, position counter (such as quadrature encoder, resolver and sin-cos sensor) and the PWM duty cycle decoder (such as the input capture engine), to measure the pulse width and period.

The synchronization of the acquisition process with the PWM wave generation is performed through the cross triggering unit (CTU). The CTU is designed to spare the CPU from the task of performing the time acquisitions of the state variables during the control cycle. In this application, the control cycle is the same as the PWM cycle. The CTU uses a double-buffered register structure to guarantee the activation of a new setup only at the beginning of the next control cycle.

The critical requirement for this subsystem concerns the ADC timing (see Figure 5). In order to speed up the acquisition, the PWM A0 rising edge is used to trigger the Start-Of-Conversion (SOC). This means that whenever the PWM counter ramp reaches the maximum value, the CTU triggers an ADC acquisition without the intervention of the CPU. This CTU action starts an ADC conversion at the beginning of the 50 µs interval (SOC). Once the ADC conversion terminates at the End-Of-Conversion point (EOC at point 2), an ISR is raised. The remaining time interval (from the EOC to the end of 50 µs period), can be used for secondary activities of the CTU (it is denoted as Free Bandwidth). At the end of 50 µs interval, the CTU triggers a new acquisition.

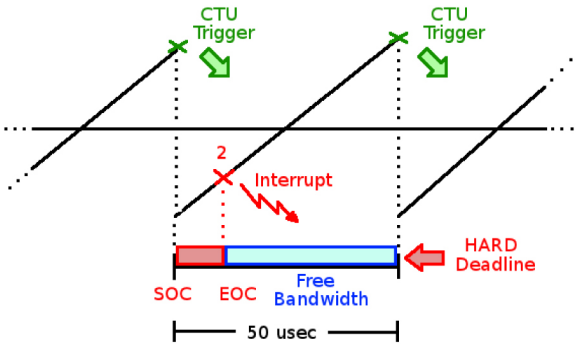

Figure 5. ADC / PWM Timing.

The end of the 50 µs window is a hard deadline for the computations, since the previous control computation must finish before the start of the next PWM cycle, where the newly computed PWM setting will be applied by the CTU using the double buffering scheme described before.

The current implementation is capable of meeting this timing constraint: measures on the current testbench show that the interval of time between the SOC and the EOC (the red interval in Figure 6) has a length of 790 PWM counter ticks (approximately 14 µs with a 120 Mhz CPU clock), and the control algorithms takes approximately 10 µs of computation. Considering that the computation is implemented into a high-priority ISR Category 1, and considering the negligible blocking time due to short interrupt disabled critical sections, the total time required by the computations is well below the 50 µs threshold.

Test Bench and Experimental Results

The final test application integrated with the open source platform presented in this paper has been tested in the Piaggio electric laboratory with the instrumentation listed in Table 1. The scheme of the bench used to test the application is shown in Figure 6. In particular, the figure shows how the Power Analyzer has been used to measure the various voltages and currents in the motor and in the power supply.

Table 1. Platform reference Footprint information.

Instrument	Maker	Model
Dynamometer Controller	MAGTROL	DSP6000
Hysteresis Dynamometer	MAGTROL	HD-805-7NA
Digital Oscilloscope	TEKTRONIX	DSO4014B
Digital Power Analyzer	INFRATEK	107A
Digital Multimeter	AGILENT	34401A
Power Supply	DELTA ELEKTRONIKA	SM 45-70 D

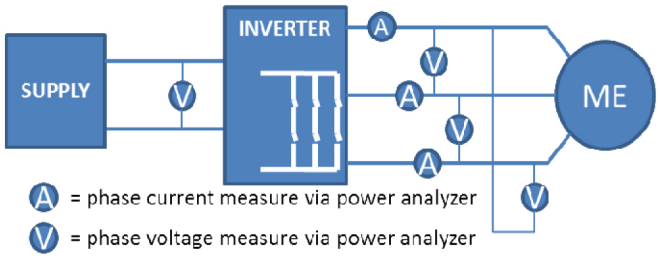

Figure 6. Functional scheme of the test bench used by Piaggio to verify the test application integrated with the platform.

Figure 7 reports the phase current in [A] acquired via oscilloscope and stored in CSV format. The time (X axis) is expressed in ms. Figure 8 is an enlarged view of Figure 7 and shows the low ripple thanks to the applied symmetric PWM.

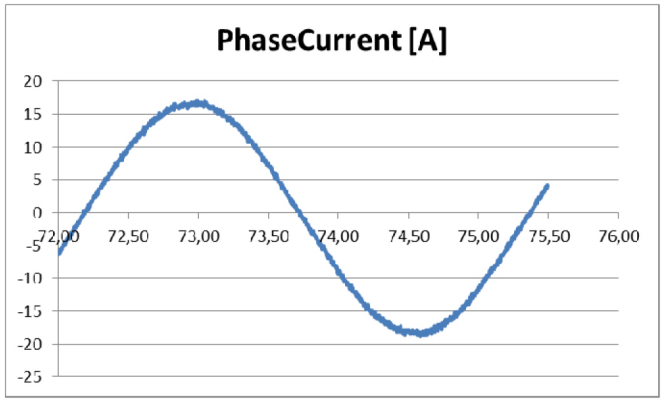

Figure 7. Current in [A] (time in ms).

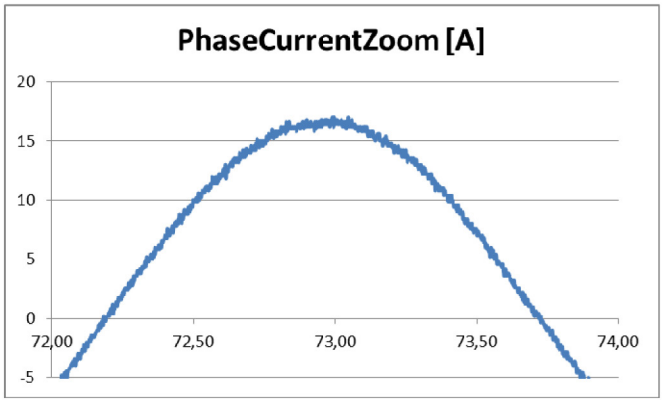

Figure 8. Zoom of Figure 8, showing low ripple thanks to the symmetric PWM.

Finally, Table 4 shows the current Idref and Iqref which are the input of the current regulator at the same speed. The expected pick phase current is calculated as follows:

$$i_{phCalc} = \sqrt{i_{dref}^2 + i_{qref}^2}$$

The pick phase currents have been measured using the Digital Power Analyzer, the Torque and the speed are measured by Dynamometer Controller (see Table 1), as listed in Table 2.

Table 2. Table of current values for idref and Iqref with measured speed and torque.

Idref [A]	Iqref [A]	IphCalc [A]	Iph [A]	Speed [rpm]	Torque [Nm]
9	17	19.72	19.03	1000	7
17	23	28.6	29.3	1002	11.8
13	20	23.85	24.22	1080	9.7
10	13	16.04	16.87	1001	7

The numbers obtained from the testbench have been evaluated as satisfactory for the application of the platform on production hardware.

FOOTPRINT AND TIMING BENCHMARKS

In conclusion, and considering the tests on the platform, this Section shortly describes the application composition in terms of OS objects, as well as the footprint and timing performance of the platform.

The reference test application consists of 6 tasks, 18 ISRs Category 2 (with RTOS support), 1 ISR Category 1 (without ISR support), 1 resource (for implementing mutual exclusion), 1 counter and 5 alarms (mainly for the implementation of periodic tasks).

The code has been compiled using the Codewarrior 10.2 compiler, with option -O2. The Freescale Leopard MCU is running at 120 MHz, with lockstep enabled (1 CPU), cache disabled, 3 Wait States on Flash access. The prefetch support on the Flash is enabled, allowing accesses in “single-cycle response” for each hit on the prefetch buffer (hits are the majority of accesses).

Table 3 shows some reference footprint information. The ERIKA Enterprise has been configured as BCC1, and its footprint includes the scheduler, ActivateTask, Resources, Alarms, IRQ Handlers, and System Timer. The complete platforms uses less than 40 Kbytes of flash, and less than 4 Kbyte of RAM (excluding the 9.5 Kbytes of the CCP DAQ list which depend on the CCP configuration for the specific application).

Table 3. Platform reference Footprint information.

Component	Flash	SRAM
ERIKA Enterprise	4354 bytes	
ERIKA Enterprise data	308 bytes	384 bytes
Device Drivers and BSW	17 Kbytes	200 bytes
Device Drivers configuration data	800 bytes	600 bytes
Communication (CCP, CAN)	15 Kbytes	12Kbytes (9.5Kbytes for the CCP DAQ Lists)

Table 4 shows some reference timings, which are in line with the data measured on other PPC architectures with ERIKA Enterprise [21]. Figure 9 shows a screenshot of the MCU trace taken with Lauterbach Trace32, showing the wakeup of a task using an ISR. The first instruction of the ISR is on the first line of the trace (function DummyFn1), whereas the start of the task is at the last line (function FuncTask1).

Table 4. Platform reference Timing information.

Measurement	Timing
Task activation inside an ISR2 from idle	5.3 µs
ISR2 Latency	0.8 µs

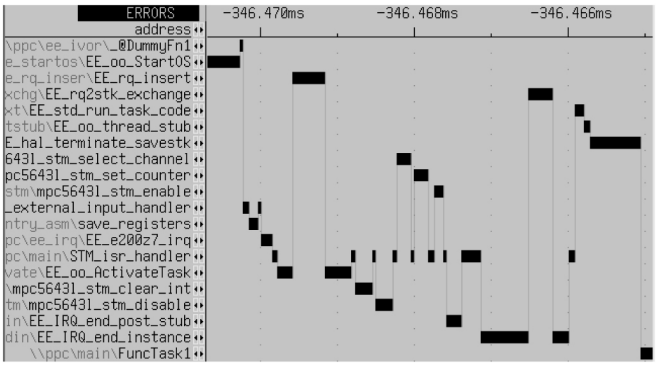

Figure 9. Timings of the task activation from an ISR2, acquired with Lauterbach Trace32.

CONCLUSIONS

This paper presented an open-source platform for motor-motive market electronic ECUs. Open-source solutions allows companies to share a common software platform while competing on the differentiation requirements. The proposed platform includes the Boot loader, the ERIKA Enterprise open-source OSEK/VDX RTOS, device drivers, communication infrastructure and automotive code generation tools. The platform is an appealing solution for 2-3 wheelers application because of its reduced cost, obtained mainly by using open-source software solutions.

The paper also showed a test control application implemented by Piaggio, with the accompanying test bench experiments. The tests show how the proposed Platform could be used in production environments.

REFERENCES

1. Freescale Semiconductor, http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC564xL
2. ERIKA Enterprise RTOS, http://erika.tuxfamily.org
3. GPL2+Linking exception, http://en.wikipedia.org/wiki/GPL_linking_exception
4. AUTOSAR Association, http://www.autosar.org/
5. MISRA C Compliance of ERIKA Enterprise, http://erika.tuxfamily.org/wiki/index.php?title=Misra_compliance
6. E4Coder Toolset, http://www.e4coder.com/
7. MathWorks, http://www.mathworks.com/
8. OSEK/VDX, http://www.osek-vdx.org
9. Wired, Oct 12th, 2012, http://www.wired.com/opinion/2012/10/automakersbecome-software-makers-the-next-battle-between-openand-closed/
10. EPL License, http://en.wikipedia.org/wiki/Eclipse_Public_License
11. ISO 26262:2011 - Parts 1 to 9, and ISO 26262-10:2012
12. P-SOCRATES FP7 Project - http://www.p-socrates.eu/
13. OSEK/VDX Certification, http://portal.osekvdx.org/index.php?Itemid=8&id=5&option=com_content&task=view, ERIKA Enterprise listed in the CB 4.5
14. AUTOSAR OS Specification. www.autosar.org/download/AUTOSAR_SWS_OS.pdf
15. Marzario Luca, Lipari Giuseppe, Balbastre Patricia, Crespo Alfons, “IRIS: a new reclaiming algorithm for server-based real-time systems”, Real-Time Application Symposium (RTAS 04), Toronto (Canada), May 2004
16. Liu, C. L. and Layland, James W., Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, J. ACM, Jan. 1973, doi:10.1145/321738.321743
17. Alessandro Biondi, Giorgio Buttazzo, and Marko Bertogna, “Schedulability Analysis of Hierarchical Real-Time Systems under Shared Resources”, Technical Report TR-13-01, RETIS Lab, Scuola Superiore Sant'Anna, July 2013.
18. Biondi Alessandro, Melani Alessandra, Bertogna Marko, Buttazzo Giorgio, “Optimal Design for Reservation Servers under Shared Resources”,In proceedings of ECRTS 2014, July
19. Rajkumar R., Sha L., Lehoczky J.P. and Ramamritham K., “An Optimal Priority Inheritance Policy For Synchronization in Real-Time Systems”, pages 249-271, In Advances in Real-Time Systems, Prentice nHall, Son S., Editor, ISBN 0-13-083348-7, 1995.
20. Baker T. P., A stack-based resource allocation policy for realtime processes, Proceedings of the Real-Time Systems Symposium, 1990. Proceedings., 11th, doi:10.1109/REAL.1990.128747
21. ERIKA Enterprise, supported architectures, http://erika.tuxfamily.org/wiki/index.php?title=Category:Supported_Devices
22. Lauterbach Gmbh, http://www.lauterbach.com
23. ISystem Gmbh, http://www.isystem.com/products/software/winidea
24. Jenkins, http://jenkins-ci.org/
25. Mathworks, http://www.mathworks.com
26. ScicosLab, http://www.scicos.org
27. E4Coder, http://www.e4coder.com
28. Mohan N. et al. “Power Electronics, converters, applications and design”, Wiley, New York, 1989
29. Freescale Motor control library User Reference Manual http://cache.freescale.com/files/motor_control/doc/MCF51_MCLIB.pdf
30. Krishnan R. “Electric Motor Drives Modeling, Analysis and Control”, Prentice Hall, February 25, 2001
31. ASAP Standard, CCP (Can Calibration Protocol) vers. 2.1, 18 feb 1999
32. KWP2000 - ISO 14230: Road Vehicles: Diagnostic Systems. Part 2: Data Link Layer
33. ISO 15765-2:2011: Road vehicles -- Diagnostic communication over Controller Area Network (Do-CAN) -- Part 2: Transport protocol and network layer services
34. Evidence EFORMS. http://www.evidence.eu.com/it/products/eforms.html

DEFINITIONS/ABBREVIATIONS

ADC - Analog to Digital Converter.

API - Application Programming Interface

ASIL - Automotive Safety Integrity Level, as specified in the ISO26262 [11] standard.

BCC1, BCC2, ECC1, ECC2, CCCA, CCCB - Conformance classes specified by the OSEK/VDX standard. Basically they are a standardized subset of the RTOS API.

BOOK E - Specification of the assembler instructions for the PowerPC architecture.

CRC - Cyclic Redundancy Check, a way to compute checksums to ensure the correct transmission of a set of data.

CSV - Comma Separated Values, a way to store tabular information inside a text file.

CTU - Cross Triggering Unit, a special configurable peripheral of the Leopard MCU.

ECU - Electronic/Engine Control Unit.

EOC - End Of Conversion.

FSM - Finite State Machine.

GPL - General Public License (an open-source license for software).

GUI - Graphical User interface.

ISR - Interrupt Service Routine.

MCU - MicroController Unit.

MBD - Model Based Design.

NVRAM - Non-Volatile RAM

OIL - Configuration language specified by the OSEK/VDX consortium for the configuration of the kernel

ORTI - Configuration file for Kernel-awareness debugging, specified by the OSEK/VDX consortium.

OSEK, OSEK/VDX - Standard API developed in the Automotive sector.

PWM - Pulse Width Modulation.

QT - Qt framework, an Object oriented C++ Library for implementing GUIs.

RTOS - Real Time Operating System.

SOC - Start Of Conversion.

VLE - Variable Length Encoding, a special encoding of assembler instructions on the PowerPC architecture.