Is Software Coded Processing an Answer to the Execution Integrity Challenge of Current and Future Automotive Software-Intensive Applications?

Majdi Ghadhab, Jörg Kaienburg, Martin Süßkraut, and Christof Fetzer

BMW AG, majdi.el.ghadhab@bmw.de; SIListra Systems GmbH, joerg.kaienburg@silistra-systems.com, martin.suesskraut@silistra-systems.com; Technische Universität Dresden, christof.fetzer@tu-dresden.de

Abstract

In upcoming automotive systems, the high integration of safety-critical software and the use of high-performance controllers with limited integrity is a demanding challenge. Innovations like driving automation require significantly higher computational power than it is available via special-purpose controllers equipped with safety features. The qualification high-performance commodity hardware for a use in safety-critical systems becomes desirable. To cope with the dilemma of low integrity of such controllers, Software Coded Processing reliably shifts the detection of execution errors into the application software allowing high diagnostic coverage of processing units' failures.

1 Introduction

The increasing extent and autonomy of software-driven E/E systems accentuates the necessity to guarantee their correct functioning. ISO 26262 defines functional safety as the "absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems" [1]. Embedded controllers represent the central part of such E/E systems. Strict safety requirements have to be fulfilled during their lifecycle.

In this paper, we first present current trends of automotive applications and their impacts on embedded computing. Next, we highlight the contradicting requirements of high performance versus high safety integrity versus cost efficiency. While reliable systems typically employ established hardware techniques to detect random faults of processing units, we focus on lower-cost and more flexible software im-

plemented fault detection mechanisms to enhance powerful general-purpose controller. We propose to apply Software Coded Processing (SCP) as it reliably addresses systematic and random, both permanent and transient, hardware failures. The last section explains the working principle of SCP, its underlying assumptions, the failure model, and the way how this technique can be deployed in a day-to-day development environment. Finally, we present experimental results with focus on utilizing Commercial-Off-The-Shelf (COTS) controllers and trade off achievable detection rates against performance implications.

2 Future Automotive Software-Intensive Systems

Future vehicles are seen as a connected and distributed network of complex software systems. Visions of a future intelligent networking of driver, vehicle, and environment, e. g. in the context of BMW ConnectedDrive [2], require new approaches for architectures of control unit platforms and their interfaces within the overall systems [3].

▶ **Centralized Domain-Controlled Architecture**

The trend of continuously integrating and networking additional ECUs is getting strongly restricted by communication, power consumption, space, and cabling. It becomes essential to shift in-car-networking complexity into software and to integrate more functions per computing unit [4]. Large Scale Software Integration (LSSI) and domain controllers are introduced in [5]. An LSSI system centralizes several high integrity vehicle software components onto a single ECU. Domain controllers are capable integration platforms and server ECUs which control several bus systems [4]. Relocating the rising software workload to domain-controllers requires more performance and accentuates safety integrity aspects.

▶ **Driving Automation**

Advanced driver assistance systems moving towards fully automated driving need data from the core vehicle network and the evolving sensor and communications technologies [6]. The central computing platform executes data intensive functions and complex algorithms for environment perception, maneuver planning, and motion control.

In order to cope with the trends of software-intensive systems, design engineers of embedded computers have to fulfill following requirements with respect to cost-efficiency:

▶ **High-Performance Computing**

Research projects for autonomous driving and robotics use modern personal computers to its full capacity to process the required software-intensive functions. This implies memory consumption in the range of Gigabytes, CPU consumption in the range of GFLOPS, and utilization of hardware acceleration [7].

▶ **Dependable Computing**

A platform shall provide computing capabilities with guaranteed timing, reliability, and integrity. Control functions have real-time constraints, must satisfy functional

safety requirements such as handling of hardware and software faults in a fail-safe and increasingly fail-operational manner.

Other design criteria, e. g. scalability, are also gaining high importance. They are not discussed in this paper as we focus on the challenge of performance and dependability.

3 The Challenge of High-Performance, Dependable, and Cost-Efficient Computing

Intended computing platforms require both high performance and high integrity. Available processors on the market have either high performance or high integrity. [8] and [9] provide examples of recent automotive high-integrity controllers. Communication with various hardware vendors proved the lack of high-integrity processing hardware providing the computing performance as required by future software-intensive applications [7].

Typically, architecture trends from desktop, laptop and server computing migrate into embedded microcontroller applications [10]. High volume standard processors have significantly better cost per performance than special-purpose ones. Table 1 lists price and performance values of popular CPUs and SoCs. Freescale's MPC5643L [8] microcontroller, designed for automotive safety-critical ECUs, is taken as a reference. Performance values are quotes from the respective technical specification documents.

Table 1. Performance per price for selected CPUs [11]

Vendor	*Model*	*CPU*	*Price/$*	*DMIPS*	*DMIPS/$*
Freescale	MPC5643L	Power PC Lockstep	15	250	16.6
Freescale	MCIMX6U5 DVM10AB	ARM Cortex-A9	26	2500	96.2
Texas Instruments	66AK2H12	ARM Cortex-A15	250	19660	78.4
Intel	Atom N270	n/a	32	3,846	120.2
Intel	i7 4770k	n/a	339	124,850	368.3
AMD	FX-8350	n/a	180	97,125	539.6

The values as of Table 1 indicate that the performance per price can be a magnitude higher for commodity hardware than for automotive-specialized ones. Other parameters like power consumption, failure modes and distribution, product availability, and operational limitations play also an important role when evaluating CPUs regarding their suitability for automotive application. Nevertheless, the per-

formance per price motivates the investigation of such processors in order to provide economically priced computing power. A significant difficulty is the circumstance that these controllers are not self-checking and have limited fault detection capabilities. The functional safety of embedded controller includes the safety integrity of the application software and the computing platform. The integrity of the computing platform is typically ensured through self-checks and hardware redundancy. We use the term "execution integrity" to refer to the detection and handling of systematic, permanent, and transient hardware failures and interference leading to safety goal violations of an executed application.

ISO 26262 lists in Annex D of Part 5 safety measures and mechanisms with high diagnostic coverages which are considered as achievable for processing units. These measures include hardware and/or software implemented fault detection.

The most straightforward way to duplicate and compare a microprocessor is the technique of a lockstep [12]. Each processor is expected to produce the same outputs given the same inputs. Unfortunately, lockstep-microprocessors double the computing cycle budget while providing just the same performance as single processing. And they are susceptible to non-determinisms. A number of mechanisms in current CPUs increase non-determinism and might disconnect lockstep CPUs of recent developments [13].

Additional to the potential limitations of hardware-implemented fault detection, relying on general-purpose processors requires safety measures without changes to the hardware architecture. Therefore, we propose to investigate software-implemented and hardware-independent fault detection for future automotive safety-critical systems. To achieve high diagnostic coverages by software-implemented fault detection, we consider software diversified redundancy, recommended by ISO 26262, and introduce an alternative approach based on Software Coded Processing.

▶ **Software Diversified Redundancy**

The design consists of two redundant and diverse software implementations in one hardware channel [1]. The redundant path is often implemented using separate algorithm designs and code to provide software diversity. The design must include methods to coordinate these two paths and to resynchronize the paths for transient errors. Due to potential common cause failures, an additional watchdog processor can be used and a detailed analysis is required to prove the diagnostic coverage.

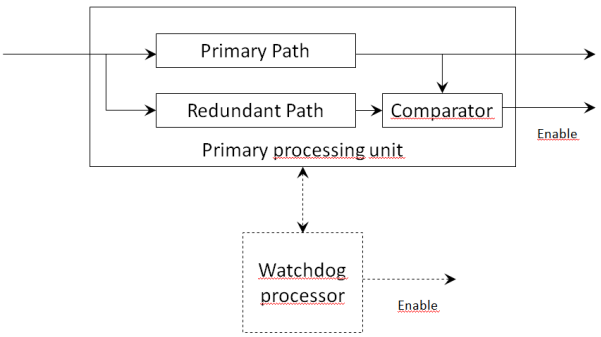

Fig. 1. Software Diversified Redundancy - architecture

▶ **Software Coded Processing**

SCP extends the functional code by the ability to detect whether it is correctly executed by its underlying hardware or not. SCP detects transient, permanent, and systematic hardware execution errors with high detection rates. Due to potential common cause failures, the checking of the output validity is performed by an external unit, e. g. a watchdog. SCP permits the efficient execution of non-critical applications and the correct execution of critical applications [14].

Table 2. Software-implemented fault detection techniques

Mechanism	*Pro*	*Con*
Software Diversified Redundancy	Allows detection of software faults	Requires diverse software implementations Requires a detailed analysis to prove independence and diagnostic coverage
Software Coded Processing	Detects systematic and random hardware failures Diagnostic coverage can be flexibly configured Detects interference failures between critical and non-critical software	Needs evaluation of performance requirements Requires detailed analysis of dependent and common cause failures

Table 2 highlights the main difference between software diversified redundancy and Software Coded Processing. Software diversified redundancy requires two (time and effort consuming) implementations, whereas Software Coded Processing requires only one implementation. Considering the achievable diagnostic coverage at lower development cost, SCP deserves further investigation to evaluate its implementation process, detection rates, and performance implications.

4 Software Coded Processing

Software Coded Processing (SCP) is a software technique. The fundamental principle of SCP is the arithmetic encoding of variables, constants, and operations. The result is an end-to-end protection which is hardware independent and continuously present.

4.1 Execution Errors and Error Model

Execution errors affect the execution integrity of embedded systems' functioning. In this paper, execution integrity is split into hardware execution errors and software

execution errors. A hardware execution error is an error that appears in the hardware, e. g. in a processor or in the memory, which potentially affects the execution of the software that is run by this hardware. This could result in a malfunction even if the software is an absolutely correct program (Fig. 2).

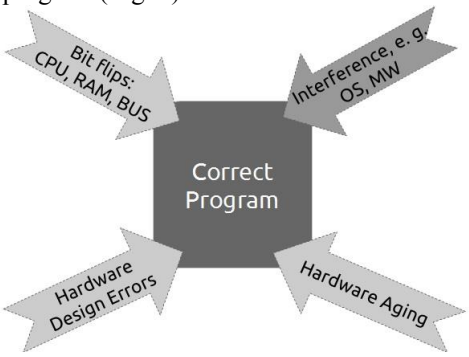

Fig. 2: Execution errors harming a correct program

Besides hardware execution errors, interference can cause as well erroneous behavior of a given system. This kind of a software execution error results from an unintended interaction between two different programs whereas one could be the operating system, firm- or middleware, or another kind of software. A prominent example of interference is faulty data in memory caused by a faulty write access to a memory region which is supposed to be used only by a given safety-critical application.

Execution errors can influence three aspects of an execution: the data flow, the control flow, and the timing. The data flow is the data stored in a system together with all calculations (arithmetic, comparison, etc.) that a safety function performs on this data. The control flow are all decisions that a safety function contains (loops, function calls, etc.). The timing is the timely execution of the safety function. Execution errors can interfere with the timing by making the execution too slow or by stopping the execution completely.

Execution errors are distinguished into transient, permanent, and systematic errors. One prominent example for transient errors are bit flips which could occur in memory, processors, and in bus signals. The characteristic property of transient errors is their randomized occurrence and that they are of very short appearance. A permanent error is an error which appears – maybe only after a distinct period of time – and remains, i. e. it becomes a permanently present error. Hardware aging is one possible reason for permanent errors because it materializes in an irreversible alteration of the electrical behavior of an electronic component. Systematic errors are permanent errors which inherently reside in a given product, e. g. caused by a design error during the development of such a product. Thus, systematic errors appear equally in all products from one development stage.

SCP is capable to detect transient, permanent, systematic errors and errors caused by interference once they propagate into the execution of the software. Errors which do not propagate will not be detected since they have no effect on the execution of the software.

4.2 Arithmetic Encoding

SCP adds information redundancy to a software program to enable it to detect execution errors. The operating principle of SCP is based on a code transformation during which variables, constants, and operations are arithmetically encoded. This code transformation can be realized either manually or via a software development tool SIListra Safety Transformer that works like a compiler and carries out the code transformation in an affordable time.

One part of the generalized procedure of developing an embedded system is writing the functional code. This functional code determines the behavior of the final system. After this functional code is written, it gets compiled and uploaded onto the system. In order to deploy SCP, one additional step is needed: The functional code has to be transformed from its original version into a new version which then contains the arithmetical encoding. As a result, a new program code is generated which still contains the original behavior and, in addition, intrinsically carries the protection via SCP. This new code, functionality-wise identical and SCP-protected, must be compiled instead of the original functional code. As a result, a different binary file is created that has to be uploaded onto the embedded system. Having done that, the systems' function remains identical compared to its origin plus it becomes *seamlessly and intrinsically protected and safe* due to the presence of SCP. Furthermore, safety is achieved *independent of the used hardware*. By deploying SCP, also COTS hardware can be used in safety-critical applications.

SCP itself is not limited to one kind of arithmetic encoding. There are different encodings which are providing different degrees of protection. The most known encoding is the AN encoding [15]. The basic principle of this encoding is based on the multiplication of any value with a constant A and, to validate the correctness of calculations or results, the check whether a result is still a multiple of this constant A. Values that are not multiples of A are considered as invalid. With SCP, all operations in a program must work with these encoded values. An execution error produces invalid values.

As an example, the numbers 2 and 3 shall be added in the original program code. The expected result is 5. If protected by AN encoding with A=7, the sum 2+3 turns into 14+21. Without an execution error, the result is 35 and a valid value because it is a multiple of 7.

4.3 Diversified Encoding

SCP can be deployed in different ways during product development. A lean approach is the Diversified Encoding based on the AN encoding [16]. Fig. 3 depicts the block diagram of Diversified Encoding.

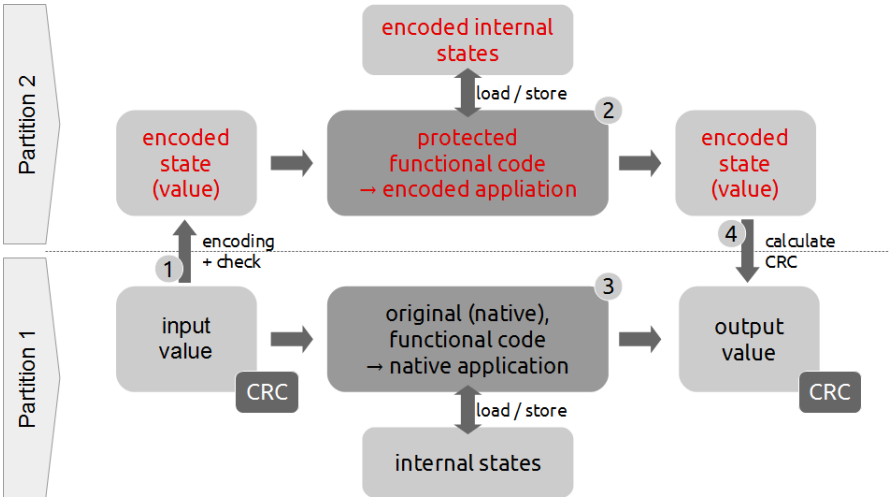

Fig. 3: Diversified Encoding - block diagram

Diversified Encoding is based on two distinct executions of the same safety function on one channel. These two executions are the native execution and its corresponding encoded execution.

▶ **Native Execution**

The native execution is the result of the original source code of the safety-critical function. This source code operates on the native input values and native states. The native execution changes only native states. The result of the native execution is the native output.

▶ **Encoded Execution**

The encoded execution is based on the encoded variant of the safety-critical function. The encoded execution operates on encoded input values and on encoded states. It produces an encoded output.

Both executions are completely distinct computations but operate on the same values. The encoded input values are the encoded variants of the native input values. The source code of the original, native code is used to generate the encoded source code thereof.

The data flow starts at step (1) with the native input values. The native input values are encoded to become the input values for step (2). In this step, the encoded function is computed. It reads the encoded input values and the encoded internal state. It performs its calculations, updates the encoded internal state, and produces the encoded output values. In step (3), the native function is executed. It reads the native input values and the native internal state. It updates the native internal state and produces the native output values. The checksum of the native output values and the checksum of the encoded output values are computed in step (4). If both checksums are identical, the calculation of the native data flow is considered as correct. The native output values are taken for the subsequent data processing.

4.4 Experimental Results

Since COTS hardware is experiencing higher attention even for safety-critical automotive applications, respective investigations were carried out. A test software was run on different COTS hardware. The symptoms of execution errors were injected into the test software during run-time, i. e. while the test software was executed on the different COTS hardware. The failure injection was done by the injection tool SIListra Safety Evaluator [17].

Fig. 4 visualizes the results from an experiment that was run on a PC with an Intel i7 processor. The test software consisted of three parts: input, an interpolation of characteristics[1], and output. The interpolation part was defined as safety-critical and, thus, was subject to the failure injection after it was protected via SCP (AN encoding with Diversified Encoding). A grand total of over 300.000.000 failure symptoms were injected and analyzed. The results were categorized into:

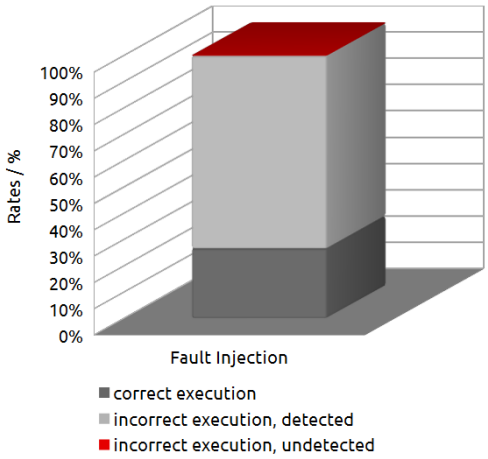

Fig. 4: Diversified Encoding - experimental results

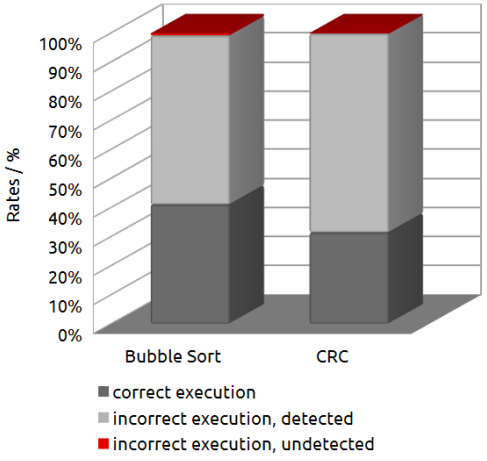

Fig. 5: New encoding - experimental results

- Correct executions: The injected failures did not falsify the execution and results of the test software.
- Incorrect executions, detected: The injected failures falsified the results and SCP detected the falsification. Cases when the test software was aborted were also counted in this category.
- Incorrect executions, undetected: The injected failures falsified the results but SCP did not detect the falsification.

[1] German: Kennfeldinterpolation

Only 0.002 % of the injected failure symptoms were not detected. In other words, 99.998 % of the injected errors were either detected or had no impact on the correctness of the executions.

Next, an equivalent experiment was carried out which made use of a new kind of encoding (Fig. 5). Diversified Encoding was not used in this experiment. Instead, only this new kind of encoding was used. As a goal, it was targeted to have less than 1 % of undetected failures while requesting a minimum performance adder to the system. The test software was replaced by Bubble Sort and CRC as test programs for memory-intensive load and CPU-intensive load. Fig. 5 shows the results of this experiment. Although not fully optimized yet, this new encoding provides as well rates of "undetected" in the range below 1 %: 0.88 % for Bubble Sort and 0.24 % for CRC (both with control flow check, CFC). The slight reduction in the ratio of undetected execution errors – in absolute figures still on ASIL D levels – results from the circumstance that this new encoding does not use the Diversified Encoding (Fig. 4 vs. Fig. 5).

Table 3. Investigated COTS hardware - performance and resources

	ARM v6k (Raspb. PI)	*ARM Cortex-A7*	*AMD E-450*	*Core i7 3720QM*
Architecture	ARM	ARM	INTEL	INTEL
Bit width	32	32	64	64
CPU clock / GHz	0.7	1.0	1.65	2.6
Memory clock / GHz	not available	not available	0.508	0.65

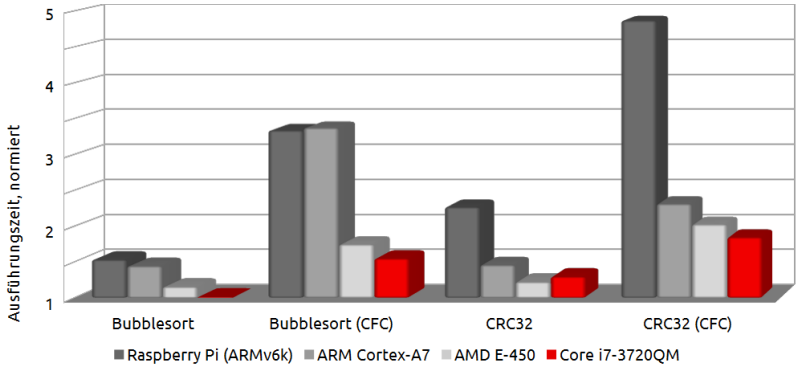

Fig. 6: New encoding - performance requirements

The evaluation of the performance requirements of this new encoding was carried out on different hardware platforms allowing to derive indications which detection rates could be achieved with which performance requirements. Because

ARM and INTEL architectures currently gain interest in the trend towards COTS hardware in safety-critical applications, two 32 bit ARM and two 64 bit INTEL architectures were selected. Table 3 tabulates the main characteristic properties of the investigated COTS hardware platforms.

Fig. 6 visualizes the performance requirements. All values were normalized to the performance required without SCP. No overall rule-of-thumb number can be derived to quantify the performance requirements. The extent of the required performance depends on different parameters which can differ from case to case. The explicit composition of the native code and the used processor architecture influence significantly the resulting performance requirements. Since 32 bit code transforms into 64 bit code via SCP, 32 bit processors require extra performance for the processing of 64 bit (encoded values). The result that the control flow check (CFC) requires performance while it provides additional protection is known.

Bottom line, SCP improves the safety of systems it has been deployed to. It detects execution errors with high detection rates allowing to be used in ASIL D applications. As a pure software technique, SCP is independent of the underlying hardware and provides a continuous protection against execution errors. Thus, SCP is an ideal technique for domain-controllers as well as COTS hardware.

On multi-core systems, SCP can be used to implement fail-operational behavior: Two channels are protected with SCP. SCP detects whether a failure occurs in the channels. When one channel fails, operation can continue with the results of the correctly executed channel.

Furthermore, SCP can replace or supplement software mechanisms for safety-critical systems such as instruction set tests, cyclic memory checks, and redundant data storage. SCP provides the same detection capabilities as these software mechanisms in addition to the other advantages already mentioned.

5 Conclusion

SCP enables commodity high-performance processors to be used within safety-critical automotive applications with respect to execution integrity. We expect a tolerable performance impact of SCP when software is accurately split into critical and non-critical and adequate hardware is used. Hence, Software Coded Processing is a suitable and already available solution to the challenge of computing performance and execution integrity for future automotive applications.

References

[1] International Organization for Standardization, “ISO 26262: Road vehicles - Functional safety.” International standard, First edition, 2011.

[2] BMW, “BMW vernetzt die Freude am Fahren.” http://www.bmw.de, 2014.

[3] Michel H-U., Kaule D., and Salfer M., “Vision einer intelligenten Vernetzung.” BMW AG in elektroniknet.de, 2012.

[4] Gut G., Allmann C., Schurius M., and Schmidt K., “Reduction of Electronic Control Units in Electric Vehicles Using Multicore Technology.” ForTISS GmbH, Munich and Audi Electronics Venture GmbH, Gaimersheim, Germany, Springer-Verlag, 2012.

[5] Reinhardt D. and Kucera M., “Domain Controlled Architecture, A New Approach for Large Scale Software Integrated Automotive Systems.” in PECCS - International Conference on Pervasive and Embedded Computing and Communication Systems, 2013.

[6] Ainhauser C., Bulwahn L., and Hildisch A., “Autonomous driving needs ROS.” BMW Car IT GmbH, ROSCon, Stuttgart, Germany, 2013.

[7] Bulwahn L., Ochs T., and Wagner D., “Research on an Open-Source Software Platform For Autonomous Driving Systems.” BMW Car IT GmbH, Munich, Germany, 2013.

[8] Baumeister M., “Addressing Safety Standard Requirements for IEC 61508 (SIL 3) and ISO 26262 (ASIL D) with the MPC5643L 32-bit Power Architecture®Microcontroller.” Freescale Semiconductor, Inc., 2010.

[9] Ben Cheikh L. and Verma A., “Safety joins performance.” Infineon Technologies AG, 2014.

[10] Circello J., “Rationale for Multicore Architectures in Auto Apps.” Freescale Technology Forum, 2011.

[11] Joachim Fritzsch, “Software-based Controller Integrity in Safety-critical Automotive Systems.” Master thesis, BMW Group and Technische Universität Dresden, 2014.

[12] Beckschulze E., Salewski F., Siegbert T., and Kowalewski S., “Fault Handling Approaches on Dual-Core Microcontrollers in Safety-Critical Automotive Applications.” Embedded Software Laboratory, RWTH Aachen University, Germany, 2008.

[13] Bernick D., Bruckert B., Del Vigna P., Garcia D., Jardine R., Klecka J., and Smullen J., “NonStop Advanced Architecture.” Hewlett Packard Company, Proceedings of the International Conference on Dependable Systems and Networks (DSN), 2005.

[14] Wappler U. and Fetzer C., “Software Encoded Processing: Building Dependable Systems with Commodity Hardware.” Technische Universität Dresden, Department of Computer Science, SAFECOMP 2007.

[15] Schiffel U., Süßkraut M., and Fetzer F., “AN-Encoding Compiler: Building Safety-Critical Systems with Commodity Hardware”, Technische Universität Dresden, Department of Computer Science, SAFECOMP 2009, Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security, Springer-Verlag, 2009.

[16] Süßkraut M., Kaienburg J., and Schmitt A., “Safe Program Execution with Diversified Encoding”, SIListra Systems GmbH, embedded world Conference 2015, Nuremberg, Germany.

[17] Süßkraut M. and Kaienburg J., “Safety-Critical Smart Systems with Software Coded Processing”, SIListra Systems GmbH, Smart Systems Integration 2015, Copenhagen, Denmark.

Full Authors’ Information

Dipl.-Ing. M. Eng. Majdi Ghadhab
BMW AG
Taunusstr. 41
80807, München
Germany
E-mail: majdi.el.ghadhab@bmw.de

Dipl.-Phys. Jörg Kaienburg
SIListra Systems GmbH
Niederwaldstr. 37
01277, Dresden
Germany
E-mail: joerg.kaienburg@silistra-systems.com

Dr.-Ing. Martin Süßkraut
SIListra Systems GmbH
Niederwaldstr. 37
01277, Dresden
Germany
E-mail: martin.suesskraut@silistra-systems.com

Prof. Dr. Christof Fetzer
Technische Universität Dresden
Nöthnitzer Straße 46
012787 Dresden
Germany
E-mail: christof.fetzer@tu-dresden.de

Keywords

Software-intensive Systems, Automotive Controller, Functional Safety, Coded Processing.