Hardware/Software Design Considerations for Automotive Embedded Systems

Falk Salewski, *Member, IEEE*, and Stefan Kowalewski

Abstract—**An increasing number of safety-critical functions is taken over by embedded systems in today's automobiles. While standard microcontrollers are the dominant hardware platform in these systems, the decreasing costs of new devices as field programmable gate arrays (FPGAs) make it interesting to consider them for automotive applications. In this paper, a comparison of microcontrollers and FPGAs with respect to safety and reliability properties is presented. For this comparison, hardware fault handling was considered as well as software fault handling. Own empirical evaluations in the area of software fault handling identified advantages of FPGAs with respect to the encapsulation of real-time functions. On the other hand, several dependent failures were detected in versions developed independently on microcontrollers and FPGAs.**

Index Terms—**Embedded systems, empirical evaluations, fault handling, safety-critical systems.**

I. Introduction

HIGH numbers of embedded systems are present in today's automobiles. Upcoming trends, as the growing amount of safety-critical functions and the centralization of several (safety-critical and noncritical) functions on single electronic control units (ECU) complicate the design of these systems. Furthermore, additional requirements given by the safety standard IEC61508 [14] and the emerging automotive safety standard ISO26262 (working draft) have to be considered. Beside safety requirements, further aspects have to be considered in the automotive domain. These aspects include real-time and reliability requirements, the need to design low cost products as well as the need to produce a high variety of different functionalities.

Up to now, these requirements have been met mostly by standard microcontrollers (MCUs) or, in specific applications with high volumes, by application specific circuits (ASICs). However, the decreasing costs of alternative hardware devices, as field programmable gate arrays (FPGAs), different variants of dual-core microcontrollers and MCUs with specific reliability/safety functions implemented in hardware (memory protection, error correction codes (ECCs) on memories, hardware schedulers, etc.) make it reasonable to investigate their potentials for automotive applications. Because of their fundamental difference in design and architecture, a comparison of CPU based systems (as MCUs) and programmable logic devices (as FPGAs) seems particularly interesting.

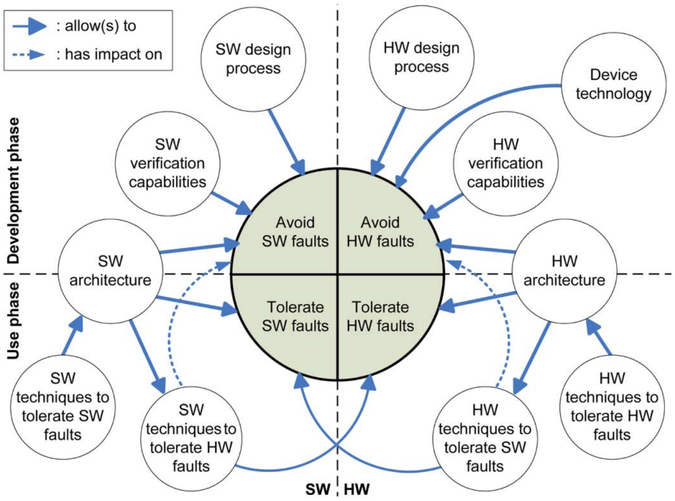


Fig. 1. Fault handling in embedded systems: hardware (HW) and software (SW) aspects.

For a comparison of the suitability of these two hardware platforms with respect to safety and reliability requirements, the structured approach for comparison of fault handling properties proposed in [26] is applied. The main idea of this approach is to consider all possible impacts on *fault avoidance*[1] (*fault prevention* and *fault removal* during the *development phase*) and fault tolerance (*error detection* and *handling of errors and faults* during the *use-phase*) as depicted in Fig. 1. It has to be noted that all physical parts of an embedded system are referred to as hardware, while the languages, which determine the behavior of these systems, are referred to as software (e.g., assembly, C, VHDL).

For some of the aspects in Fig. 1, well-founded results are present. These results are achieved on basis of physical experiments and reliability reports, which give suitable information, at least with respect to hardware issues. Therefore, differences of hardware fault handling in MCUs and FPGAs can be discussed in Section II-A on basis of these results.

Other approaches are based on best practice approaches and are often subject to controversial discussions. This is typically the case if *human factors* contribute to the fault handling. These human factors can have major impacts on the quality of intellectual products, as for example SW. Experiments, which in-

Manuscript received October 15, 2007; revised March 7, 2008 and June 23, 2008. Current version published September 10, 2008. Paper no. TII-07-10-0156.R2.

The authors are with the Embedded Software Laboratory, RWTH Aachen University, 52074 Aachen, Germany (e-mail: salewski@informatik.rwth-aachen.de; kowalewski@informatik.rwth-aachen.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2008.2002919

[1] The terms *fault, error, failure, reliability, safety, fault prevention, fault tolerance, fault removal, fault avoidance, development phase, use phase, random testing, deterministic testing, content failure, silent failure, timing failure, common mode failures,* and *fault activation reproducibility* used in this paper are compliant with those proposed in [2].

vestigate aspects influenced by human factors have to mask out the differences of the individual developers by using a statistically significant number of developers which complicates the conduction of these experiments.

Despite their importance, relatively few empirical studies have been published that investigate issues relating to the dependability of software [22]. Reasons can be seen in the specific needs for this investigations mentioned above. In addition, these studies published (e.g., [3], [10], [15], [19], [22]) deal with faults in software only and do not take HW/SW dependencies into consideration which are an important factor in embedded systems. Therefore, an approach considering HW and SW aspects will be presented in this paper.

The remaining paper is organized as follows: After a brief comparison of the handling of faults in both devices in Section II, an empirical evaluation considering the handling of SW faults is presented in Section III. This section is subdivided in the design of the experiment, the evaluation, the test of hypotheses with the corresponding results, and a discussion of threats on validity. Finally, a conclusion is given in Section IV.

II. Handling of Faults in MCUs and FPGAs

A. Handling of HW Faults

In case of the first impacts on the handling of hardware faults (right part of Fig. 1), namely the design process, the device technology and HW verification capabilities, no major differences had been identified between those two hardware platforms [26]. The design process is not device dependent and HW verification capabilities do not differ substantially. Finally, different device technologies (memory type, structure size, etc.) are available for MCUs and FPGAs as well.

Otherwise, the hardware architecture has an impact on the avoidance and on the tolerance of hardware faults. With respect to fault avoidance, certain aspects as signal integrity, power supply decoupling, clock decoupling, protection of I/O circuits and component derating have to be considered. The architecture can also reduce the single event upset (SEU) sensitivity of a device. Although MCUs and FPGAs are affected differently by transient hardware faults caused by SEU, these faults can change the behavior of a device fundamentally in both cases [26]. Beside transient faults, permanent hardware faults have to be kept in mind, which are less likely but also harder to mitigate (e.g., by partial reconfiguration in an FPGA). Another important type of hardware faults are *intermittent faults* caused by variations in the manufacturing process. This type of fault can have transient behavior (driven by temperature, etc.) and its importance is increasing with shrinking structure sizes [7]. Therefore, fault handling techniques to mitigate hardware faults are needed on both devices, if they are applied in safety-critical applications.

In MCUs, transient faults can be introduced in the program memory, the data memory, as well as in general and special purpose registers. Mitigation of faults in memories can be achieved by error correction codes implemented in hardware. The faults in MCU registers, as stack and program counters, are more challenging but can be tolerated by approaches based on specific hardware as presented in [4] (specific fail-safe MCU), [21] (use super-scalar processors and duplicate instructions), and [5] (heterogeneous dual core device).

In FPGAs, faults of transient nature can affect the configuration memory, flip-flops, and, if present, the data memory. As with MCUs, error correction codes allow to detect and correct faults in the configuration and data memories. These approaches, as for example [1] (using an inbuilt constantly running CRC in combination with reconfiguration) and [32] (read back and check of configuration in combination with reconfiguration), need additional hardware. Another option is to use triple modular redundancy (TMR) on a single FPGA, which is considered as a SW measure (see below). However, in this approach, *common mode failures* for all three modules have to be considered (e.g., faults in common clocks and power lines).

Finally, SW measures to handle HW faults are available on both devices (an overview is given in [12]). This is resulting in an area overhead in FPGAs (e.g., TMR, partial TMR [23], coded state machines) and a runtime and memory overhead in MCUs (e.g., recovery blocks, ECC on memories, SEU tolerant registers [20], SW encoded processing [30]). Moreover, further references targeting software based fault tolerance on CPU based systems can be found in [11]. Depending on the application, these SW measures can be applied instead or in combination with the HW based techniques mentioned above.

B. Handling of SW Faults

As in the case of HW faults, several impacts on the avoidance and tolerance of SW faults are present (left part of Fig. 1), which partly depend on the hardware platform used.

Software fault tolerance measures in MCUs can be based on HW. Examples are memory protection units (achieve data integrity), HW schedulers (achieve guaranteed processing time for individual tasks), and different types of watchdogs. In FPGAs, measures to assure data integrity can be applied, too (corresponding function blocks can be synthesized as part of the FPGA design), while measures to assure processing time are typically not needed (parallel structure of FPGA). The positive effect of these measures is obvious from the safety point of view, but other needs, like low product costs, often restrict the use of these measures in today's automobiles. However, this might change in the future according to the decreasing costs of microelectronics.

Furthermore, different types of SW based measures of SW fault tolerance are known, reaching from defensive programming techniques to N-version programming (NVP). While the success of some techniques is intuitive, empirical investigations are needed to determine the potentials of other techniques. A good example is the well known experiment, in which Knight and Leveson showed the limitations of the original approach of NVP [15]. Based on these results, further improvements of this approach had been proposed. One of these suggested improvements is the idea of "forced diversity" introduced in [16]. The evaluation of an application of forced diversity will be presented later in this paper.

Moreover, MCUs and FPGAs differ in their SW design process (including languages, levels of abstraction, tools and the process itself) and their SW architectures (major difference: sequential versus parallel execution). Finally, verification capabilities for the SW developed (review, testing, formal verification) might differ. All these aspects have an impact on the avoidance of software faults (see Fig. 1) and should be considered in safety-critical applications for this reason.

III. Empirical Evaluation

Hardware fault handling considerations, as presented in Section II-A, are an important factor in many automotive applications. They are considered during hardware platform selection for critical applications, while aspects of SW fault handling are typically not considered at this stage. To investigate the aspects of SW fault handling, several experiments have been conducted at our institute. A first experiment investigated the failure independence between software developed on MCUs and complex programmable logic devices (CPLDs) [28], while a second experiment replicated this investigation with MCUs and FPGAs [25] and additionally investigated potentials of encapsulation and verification by review in these devices. A third experiment compared the potentials of fault avoidance by review and testing with the approach of fault tolerance by N-version programming [24]. While the results of all three experiments are considered, the second one will be described in the following in more detail.

A. Design of Experiment

The investigations in this experiment were driven by three main hypotheses:

A first interesting aspect was to find out, if software developed by different independent teams on different hardware platforms differs in its failure behavior. If this approach would lead to diverse failures, it would be very useful for N-version techniques. This aspect was expressed by the following hypothesis.

- H_1: The differences between failures, which appear in software written for MCUs, and failures, which appear in software written for FPGAs, is higher than the difference between failures, which appear in software versions written for the same type of hardware platform.

Secondly, differences between the hardware platforms with respect to encapsulation of subtasks were investigated. This investigation was driven by the following hypothesis.

- H_2: According to their parallel structure, FPGAs allow a better encapsulation of real-time functions.

For the third aspect, the potentials of review to uncover SW faults, half of the versions implemented were reviewed by three student assistants after the end of the main experiment described above. These investigations were expressed by the following hypotheses.

- H_3a: In case of the FPGA versions, reviews reveal more timing failures than in case of the MCU versions.
- H_3b: In case of the FPGA versions, the reviewability[2] is rated better (on average) than in case of the MCU versions.

For this experiment, a small automotive application, namely a four channel speed measurement (concurrent measurement and processing of four independent speed signals with a given timing and accuracy) in combination with a CAN bus communication (transmission of the speed values measured) was chosen (see Fig. 2). Moreover, several *additional tasks* (e.g., presenting of status information on LEDs or in the CAN message, calculation and transmission of peak values, generation and transmission of test messages, each upon request by three user buttons) had to be added in a second step.

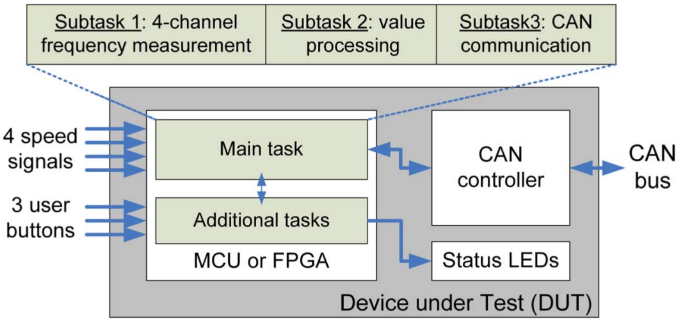


Fig. 2. Automotive application used for the experiment (white boxes = hardware components, light gray boxes = software modules).

With the knowledge of the application, the second hypothesis can be concretized by using the following subhypotheses.

- H_2a: In case of FPGAs, changes in the speed signals fed into the DUT lead to fewer changes in the timing of the CAN communication than in case of MCUs.
- H_2b: In case of FPGAs, the integration of the additional tasks lead to fewer changes in the timing of the CAN communication than in case of MCUs.
- H_2c: In case of FPGAs, the integration of the additional tasks lead to a smaller increase of content failures (speed values in CAN message sent) than in case of MCUs.
- H_2d: In more FPGA than MCU versions, equal test data for all four inputs always lead to four equal results (independence of measurement channels).

The experiment took place in a lab course with 24 computer science students forming 12 groups of development teams. We established two treatment groups and the teams of both groups received a common specification (described briefly above). Teams in the first treatment groups had to implement this specification on a MCU target hardware platform while teams of the second groups had to implement the same specification on an FPGA target. After the implementation was finished, all teams had to pass an acceptance test, which was applied to guarantee a certain minimum level of quality of all versions. Further details of this test can be found in the following Section III-B. If a team did not pass this acceptance test, it had the chance to correct the version and try another acceptance test. Only those versions that passed this acceptance test were considered for the later evaluation. After this first half of the experiment, the hardware platforms were exchanged. Those teams which had programmed the MCU target before programmed the FPGA now and vice versa. At the end, the same acceptance test had to be passed and again only those versions that passed this test were considered for evaluation.

The design of this experiment has the advantage that the initial knowledge and skills of the teams can be masked out (every team has to program both targets in random order). Another advantage was the educational aspect. By implementing a common specification on both platforms, the differences between the two hardware devices and their corresponding design methodologies became clearer to the students [27].

Furthermore, it has to be noted that the acceptance test mentioned above actually consisted of two steps. In a first step, the minimum quality of the *main task* was checked. As soon as the teams had passed this first step, they stored the version in

[2] *Reviewability* of a certain software represents the ease to determine if this software implements the specification correctly or not.

a given directory and proceeded with the *additional tasks*. After these tasks had been added to their design, we conducted the same acceptance test as before in combination with a few additional tests for the additional tasks. This approach allowed a later comparison between versions consisting of the main task only and versions consisting of the main task and the additional tasks (named *final versions* in later sections) as well.

Moreover, a questionnaire was used at the beginning of the lab course to obtain information about the individual skills of the experiment participants. Parts of the results were used in Section III-D for validity evaluation.

B. Evaluation

In earlier experiments, we recognized the need for an acceptance test. In a first experiment (see [28]), 20 input combinations generated randomly were used for the acceptance test, but several versions were lacking certain fundamental functionalities after they passed this test. Thus, the acceptance test for this experiment was generated manually with certain functionalities in mind (representing the minimum quality). This acceptance test was passed by 11 MCU and 11 FPGA versions (two teams passed with only one version).

In the following evaluation test, black box testing was applied, as all experiment versions had to be treated identically. The test cases were based on *random testing* and *deterministic testing* with an overall number of more than 50 000 input combinations. In the latter case, the test cases were determined by a specific driving scenario (a tool was developed to generate test data based on a given velocity profile) and by manual selection of specific input constellations which were considered as especially critical. In the case of random testing, the test cases were determined randomly (even probability distribution) by another tool developed for this purpose. Since the maximum allowed response time to certain input combinations was up to 2 s, the test runs for the 22 versions created needed more time than might have been expected by the number of test input combinations. The variable measured by the evaluation test was the number and the type of failures found in each version. The following types of failures were considered: 1) the time needed after a change of input values to send a correct CAN message was longer than specified *(timing failure)*, 2) no CAN messages were received for a certain test input *(silent failure)*, and 3) the message contained wrong values *(content failure)*.

For both, the acceptance and the evaluation test, dedicated test hardware was needed. In case of the acceptance test, a semi automated test approach was chosen: The generation of the test cases was implemented in an FPGA (defined test vectors could be selected via an array of switches) and the result was check for correctness manually by comparing the result displayed on a CAN bus monitor with a given table of correct results. In order to allow a sufficient number of test vectors, an automated test environment, which we developed for this purpose, was applied (see Fig. 3). Specific properties of real-time systems had to be considered, as for example the need for a defined reset of the device under test before each test run to improve the reproducibility of the test results *(fault activation reproducibility)*. In [18], test coverage metrics have been applied to characterize the different experiment outcomes. However, no correlation between test coverage and identified faults could be shown in this work and no common test metric could be found for MCUs and FPGAs. Moreover, real-time properties have an impact on usual test coverage metrics [8] and therefore require sophisticated coverage models. Since the aim of the given experiment is not to show that the different versions implement the specification correctly, but to identify differences in failure behavior of the various versions, no coverage criteria were applied in this evaluation.

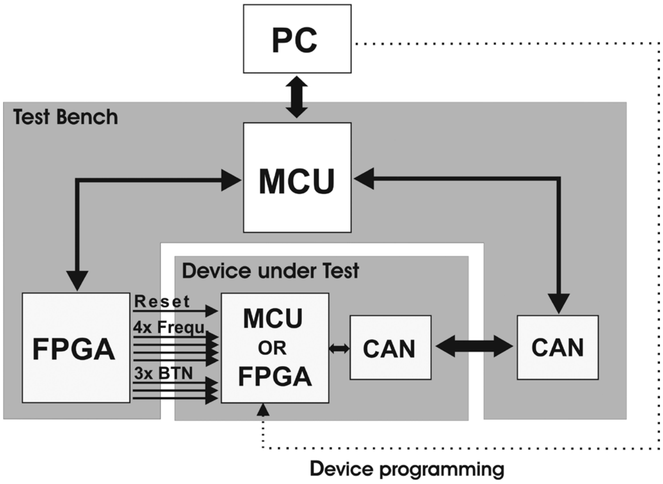


Fig. 3. Automatic test environment.

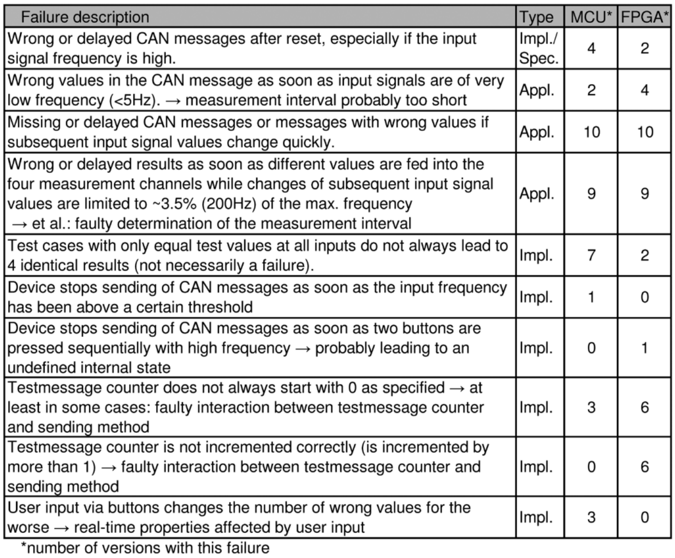
The evaluation of the potentials of review was achieved by three reviewers reviewing six MCU and six FPGA versions each. All three reviewers were familiar with both hardware platforms and were guided by a given review instruction and a given review report form. Part of the review report form was a grading of the reviewability of the different versions.

C. Test of Hypotheses

1) Test of Hypothesis H_1: First, the failure behavior of the different versions created on different hardware platforms was compared in order to test hypothesis H_1. As depicted in Fig. 4, we observed several dependent failures (failures which appeared on the same input in different versions). Although different hardware platforms with different programming languages and development methodologies had been applied, certain *specification related faults*[3] (e.g., ambiguous or incomplete statements) and *application related faults* (e.g., a difficulty in the application was not seen) lead to these common failures in several versions developed independently. Moreover, *implementation related faults* leading to dependent failures could be identified in MCU and FPGA versions. In one case, a test counter had to be implemented starting with the value 0×00. However, in 5 versions, this counter started with the value 0×01 for different reasons. One reason, present in several teams, was a wrong placement of the increment command while another team initialized the counter with 0 × 01 instead of 0 × 00.

As can be seen in Fig. 4, dependent failures on one MCU and one FPGA version were present, but occurred less often than de-

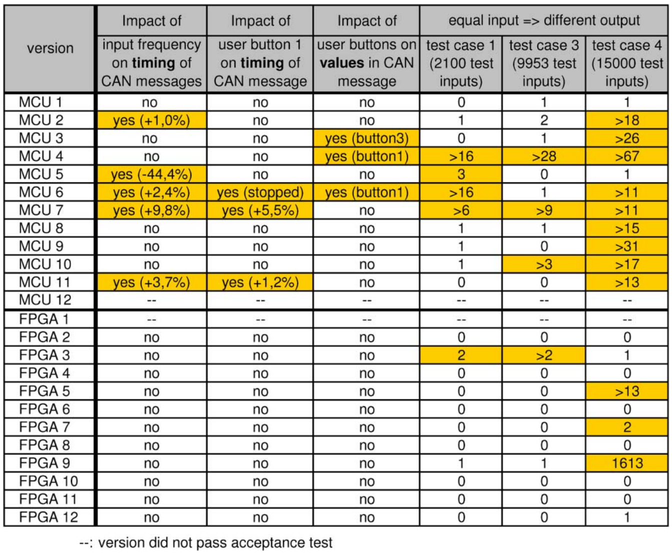
[3] The terms *specification related faults* and *implementation related faults* are defined in accordance with [18]. However, the additional term *application related fault* is used for all faults which originate from certain difficulties in the application itself (as proposed in [24]).



Failure description	Type	MCU*	FPGA*
Wrong or delayed CAN messages after reset, especially if the input signal frequency is high.	Impl./ Spec.	4	2
Wrong values in the CAN message as soon as input signals are of very low frequency (<5Hz). → measurement interval probably too short	Appl.	2	4
Missing or delayed CAN messages or messages with wrong values if subsequent input signal values change quickly.	Appl.	10	10
Wrong or delayed results as soon as different values are fed into the four measurement channels while changes of subsequent input signal values are limited to ~3.5% (200Hz) of the max. frequency → et al.: faulty determination of the measurement interval	Appl.	9	9
Test cases with only equal test values at all inputs do not always lead to 4 identical results (not necessarily a failure).	Impl.	7	2
Device stops sending of CAN messages as soon as the input frequency has been above a certain threshold	Impl.	1	0
Device stops sending of CAN messages as soon as two buttons are pressed sequentially with high frequency → probably leading to an undefined internal state	Impl.	0	1
Testmessage counter does not always start with 0 as specified → at least in some cases: faulty interaction between testmessage counter and sending method	Impl.	3	6
Testmessage counter is not incremented correctly (is incremented by more than 1) → faulty interaction between testmessage counter and sending method	Impl.	0	6
User input via buttons changes the number of wrong values for the worse → real-time properties affected by user input	Impl.	3	0

*number of versions with this failure

Fig. 4. Distribution of identified failures on different versions (*Type* indicates if the corresponding fault is considered as specification, application, or implementation related).



version	Impact of	Impact of	Impact of	equal input => different output		
	input frequency on **timing** of CAN messages	user button 1 on **timing** of CAN message	user buttons on **values** in CAN message	test case 1 (2100 test inputs)	test case 3 (9953 test inputs)	test case 4 (15000 test inputs)
MCU 1	no	no	no	0	1	1
MCU 2	yes (+1,0%)	no	no	1	2	>18
MCU 3	no	no	yes (button3)	0	1	>26
MCU 4	no	no	yes (button1)	>16	>28	>67
MCU 5	yes (-44,4%)	no	no	3	0	1
MCU 6	yes (+2,4%)	yes (stopped)	yes (button1)	>16	1	>11
MCU 7	yes (+9,8%)	yes (+5,5%)	no	>6	>9	>11
MCU 8	no	no	no	1	1	>15
MCU 9	no	no	no	1	0	>31
MCU 10	no	no	no	1	>3	>17
MCU 11	yes (+3,7%)	yes (+1,2%)	no	0	0	>13
MCU 12	--	--	--	--	--	--
FPGA 1	--	--	--	--	--	--
FPGA 2	no	no	no	0	0	0
FPGA 3	no	no	no	2	>2	1
FPGA 4	no	no	no	0	0	0
FPGA 5	no	no	no	0	0	>13
FPGA 6	no	no	no	0	0	0
FPGA 7	no	no	no	0	0	2
FPGA 8	no	no	no	0	0	0
FPGA 9	no	no	no	1	1	1613
FPGA 10	no	no	no	0	0	0
FPGA 11	no	no	no	0	0	0
FPGA 12	no	no	no	0	0	1

--: version did not pass acceptance test

Fig. 5. Experiment results regarding encapsulation.

pendent failures on two identical hardware platforms. For this reason, the hypothesis H_1 can be accepted. Nevertheless, it has to be noted that especially in case of application related faults high dependencies were present, also between versions implemented on different hardware platforms. These results support the findings of our previous experiment investigating dependent software failures on diverse hardware platforms presented in [28]. While the focus of the investigations presented in this work was an identification of the problems leading to the remaining dependencies between the versions, we showed the failure dependency of the versions as well as a benefit from using diverse hardware platforms in the previous experiment by using a resampling approach (bootstrap method, see [28] for details of this experiment).

These results have an impact on the idea of N-version programming, especially on the approach of forced diversity. The idea of this approach is that different design methodologies lead to differences in the design and thus to differences in failure behavior [16]. However, the dependencies identified above indicate limited potentials of this approach, especially in case of application and specification related faults, but also in case of certain implementation related faults (see also [24]).

2) Test of Hypothesis H_2: While several failures were similar or even identical, a few differences in failure behavior could be identified which were implementation related. Furthermore, differences might be present which did not lead to failures in this application. Therefore, the following subhypotheses were tested to investigate differences with respect to encapsulation.

In order to investigate hypothesis H_2a, the values of the input frequencies fed into the device under test (and thus the execution of subtasks one and two, see Fig. 2) have been varied and changes in the CAN communication (subtask 3) were recorded. Two different frequencies (about 6% and 100% of the maximum input value specified) were used as input and the time between two consecutive messages on the CAN bus was recorded in both cases. As a result, the CAN communication was slowed down with increasing input frequency in case of 4 of the 11 MCU versions (see 2nd column of Fig. 5, raise stated in brackets, the decrease in version MCU5 was intended). Therefore, an impact of the speed measurement functionality on the timing of the CAN communication cannot be neglected for MCU versions. Otherwise, this effect was not present in any FPGA version. Accordingly, the hypothesis H_2a could be accepted.

For the test of hypothesis H_2b, the timing of the CAN communication of the *main versions* was compared with the timing of those versions including the additional tasks (*final versions*). In case of six MCU versions, the inclusion of the additional tasks led to changes in the timing of the CAN communication (increase of time between two consecutive CAN-messages from 0,5 ms to 20 ms) while the CAN communication was not affected by the additional tasks in any FPGA version. As depicted in the third column of Fig. 5, this difference was even increased by pressing user button 1 (changes behavior from displaying the CAN status to displaying the input frequencies on LEDs) in three MCU versions. According to these results, the hypothesis H_2b was accepted.

As in case of the testing of the previous hypothesis, versions implementing the main task only (main versions) were compared with versions including the additional tasks for the test of hypothesis H_2c. However, instead of impacts on the timing of the CAN messages, *content failures* in these messages were considered in this case. It had been expected that the final versions lead to more failures according to their increased complexity. However, four of the final MCU versions (No. 2,3,8,10) showed fewer failures than the corresponding main versions while only two demonstrated more failures (No. 5,7). Constant improvement of the code which implemented the main function, during implementation of the additional tasks can be seen as a reason. However, it has to be mentioned that one of the final MCU versions (MCU6) stopped all CAN activity if user button 1 was pressed and frequencies higher than ∼33% of the maximum frequency specified were fed into the DUT (see third column of Fig. 5), while the main version was working well for specified inputs. Regarding the FPGA versions, only two of the final versions showed fewer failures (No. 3,6), while four revealed more failures (No. 2,7,8,9) than the respective main versions. The failures can be grouped in failures that represent faults in the measurement itself and those which represent faults in the sending of the CAN messages. In terms of the results presented above, no FPGA version revealed any faults in the measurement with respect to button inputs while three MCU versions had major failures according to button inputs (MCU6: stopped as described above, MCU3 and MCU4: content failures in CAN messages, see also 4th column of Fig. 5). Concerning the CAN functionality, two MCU and four FPGA teams revealed failures as soon as more than one button was pressed. Even though some additional tasks revealed potentials for encapsulation in FPGAs (no faults in measurement according to any input via the three buttons, no influence of input data on intervals of CAN messages) other additional tasks led to typically higher numbers of failures in the final FPGA versions while final MCU versions tended to reveal less failures than their respective main versions. For this reason the hypothesis H_2c was not accepted.

In order to clarify hypothesis H_2d, identical data was fed into all four measurement channels of the DUT and the results of all four channels were compared. If all measurement channels were correct and independent, the results had to be identical. This has been checked with three test cases and differences in the failure behavior have been recorded. For all three test cases, at least three times more MCU versions led to nonidentical results than FPGA versions (see last three columns of Fig. 5). Thus, the hypothesis H_2d was accepted.

Moreover, to determine the significance of the successful hypothesis tests, the probability that the observed results could have occurred randomly was determined by using the resampling method.[4] The achieved values are $\alpha = 0.051$ in case of H_2a, $\alpha = 0.055$ in case of H_2b, and $\alpha = 0.001$ in case of H_2d. While the first two values are slightly above the accepted threshold of 0.05 (5%), we still consider the results as useful, especially as no faults occurred in any FPGA version.

3) Test of Hypothesis H_3: Finally, reviews were conducted with an overall number of three reviewers to test the hypotheses H_3a and H_3b. To evaluate the success of the reviews, the review results were compared with results gained by testing. This was done for the following three scenarios:

1) car is breaking with all four wheels blocking;
2) car is breaking with single wheels blocking;
3) wheelspin of single wheels.

As a result, a slightly higher compliance of review and test result can be found with MCUs (compliance in 32 cases) than with FPGAs (compliance in 29 cases). For this reason, hypothesis H_3a cannot be accepted. However, it has to be noted that certain aspects have been shown exactly for FPGA versions only, as the determination of the exact CAN sending interval (only intervals were given for MCU versions).

In order to test the hypothesis H_3b, the grading of the reviewability given by all reviewers for each version was evaluated and presented graphically in Fig. 6 (grades: $1 =$ very good, $\ldots, 5 =$ not possible). It has to be noted, that the grading was normalized by dividing the values by the average grading of the corresponding reviewer. As can be seen in this figure, the average grading of the reviewability for FPGA versions is better than the grading for MCU versions in case of all three reviewers. The significance of the difference between the review results has been evaluated again with the resampling method. The corresponding value of $\alpha = 0.176$ states only limited significance of the observed result. However, it is expected, that the significance could be increased by using higher numbers of reviewers for the investigation (a simulation based on the measured values showed that a significant result might be shown by quadrupling the number of the considered review results). However, hypothesis H_3b could not be accepted on basis of the results observed in this experiment. Moreover, it can be seen in Fig. 6 that comparatively good as well as comparatively bad grading is present for MCU and FPGA versions and that a team creating an MCU version with good reviewability not necessarily created an FPGA version with good reviewability and vice versa.

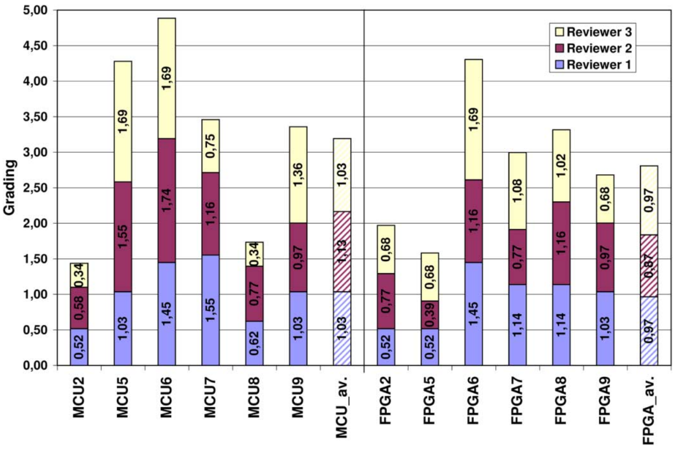


Fig. 6. Independent grading of reviewability (normalized) by three reviewers.

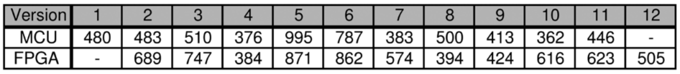
In accordance with the results of our review with only one reviewer presented in [25], no clear advantage of FPGAs with respect to reviewability could be identified. However, we revealed differences in the problems during review. The problems in the MCUs resulted from interrupts and difficulties in determining the execution times of subtasks (e.g., the determination of the CAN transmission interval). The problems in the FPGA probably resulted from difficulties in the understanding of the behavior of the interconnected parallel processes.

D. Threats to Validity

Next, threats to internal and external validity are discussed. In this context, *internal validity* represents the correctness of the experiment itself while *external validity* represents the portability of the results to other applications [31].

1) Threats to Internal Validity: According to a questionnaire conducted at the beginning of the lab course, students were more experienced in C/MCUs than in VHDL/FPGAs. This different previous knowledge was tried to adjust by a two day introductory course (described in [27]) prior to the experiment. The different skills in the programming languages might have influenced the structure and complexity of the resulting codes and

[4] A random sample was drawn with replacement for 100 000 times; then the probability of obtaining the observed results, which is represented by α, was determined on basis of the simulated data.



Version	1	2	3	4	5	6	7	8	9	10	11	12
MCU	480	483	510	376	995	787	383	500	413	362	446	-
FPGA	-	689	747	384	871	862	574	394	424	616	623	505

Fig. 7. Lines of code (final versions), excluding comments and blank lines.

thus might have influenced the results, especially in case of hypothesis H_2.

Moreover, one compilation of VHDL code took up to 2 min while C code was compiled in a few seconds. This aspect might have influenced the development. Nevertheless, no threat on internal validity is seen in this aspect since it is a given difference between FPGA and MCU programming.

While we increased the number of reviewers to three for the results presented in this paper, a higher number would still be beneficial to support our results. However, we do not expect fundamental changes in the results.

The selection of the scenarios for the evaluation of the review might have influenced the test of hypothesis H_3a. Since the same scenarios were applied for both platforms, we expect no major threat to validity of this selection.

2) Threats to External Validity: The results could depend on the type and the complexity of the task used. Therefore, a minimum complexity of the experiment task is required to allow representative results, and hard real-time requirements (typical for the automotive domain) should be present. The task chosen included several real-time requirements (four channel frequency measurement, communication, etc.). While the complexity with respect to lines of code (see Fig. 7) was slightly less than in a comparable experiment [18], sufficient challenges were present in this application according to several concurrent real-time tasks. Hence, the given requirements are in our opinion fulfilled by our experiment task, but additional experiments are desirable to minimize remaining dependencies between the task and the results.

Another important threat to external validity is the quality of the participants regarding experience and development knowledge. Although this problem is applicable on this experiment, the aim here was to show a general difference of an effect and not to give an absolute assessment of the overall number of failures for both systems. In this context, using students to test the initial hypothesis is a viable approach referring to [29]. In addition, in [6], it is stated that no difference of programming expertise between professional and nonprofessional developers could be found, while [13] states that at least last-year software engineering students and professionals have a comparable assessment ability.

The failures found by review during the test of hypothesis H_2a might have been influenced by the review technique used. In contrast to formalized reviews, as for example *code inspection* described in [9], only three reviewers worked independently in this experiment and the guidelines for the reviewers were limited to the review report form. While the internal validity is not affected by this aspect (both platforms were reviewed by the same technique), the external validity is limited to the review technique used in this experiment.

In case of the MCUs, most teams chose an interrupt based approach. However, the task might allow a cyclic scanning approach (static scheduling) which could have a positive impact on the aspect of encapsulation in MCUs. But this approach is suitable for periodic input signals only or requires very short cycles resulting in a need for higher CPU frequencies. For this reason, we expect no threat on external validity in this aspect, but nevertheless, an investigation with this alternative approach would be desirable.

IV. Conclusion

The importance of HW and SW fault handling is increasing in automotive applications which makes it reasonable to consider new hardware platforms. Thus, fault handling measures for FPGAs have been compared with those for MCUs.

For a comparison of the handling of HW faults, a survey on current fault handling approaches has been presented briefly in Section II-A. As a result, handling of random HW faults is needed in both devices if they are applied in safety-critical applications. However, fault handling can be applied on MCUs as well as on FPGAs. This is typically resulting in memory and processing time overheads in MCUs and area overheads in FPGAs. Moreover, trends to integrate fault handling modules in MCUs might be important for future designs.

Regarding the handling of SW faults, not many well founded results are present. For this reason own experiments have been conducted and the most recent one has been described in Section III. On basis of this experiment, three main hypotheses were tested showing the following results.

H_1: NVP based on diverse HW platforms (MCU and FPGA) revealed only small benefit over NVP based on homogeneous platforms. The reason are mostly faults originating from the specification and the application itself and thus present on both HW platforms. These results correspond to our previous experiment [28] and the problem statement given in [17].

H_2: Three of the four subhypotheses with respect to a better encapsulation in case of FPGAs could be confirmed (H_2a, H_2b, H_2d) while a fourth subhypothesis could not be confirmed (H_2c). As a general result, encapsulation works better on FPGAs, if the separated functions have only limited functional interactions (e.g., frequency measurement/value processing and CAN bus communication: only the values measured and processed by the measurement/processing module are given to the CAN communication module at defined points in time). On the other hand, stronger interaction between functions (as in case of some additional tasks) tended to result in an undesired coupling between the modules in several FPGA versions in our experiment.

H_3: The reviewability of the FPGA versions was rated better (on average) than those of the MCU versions by each of the three reviewers, but the significance of this result is limited. Moreover, a higher compliance of review and test results was revealed with MCU versions. Therefore, no fundamental difference regarding reviewability could be shown in this case.

Finally, threats to validity have been discussed and no major threats on internal and external validity were identified. Therefore, the results presented can support the systematic comparison of MCUs and FPGAs with respect to fault handling, which is considered as particularly important in automotive applications. However, further investigations, especially in the area of SW fault handling are desirable to achieve more comprehensive results for hardware platform selection.

References

[1] ALTERA, Error Detection & Recovery using CRC in Altera FPGA Devices, Application Note 357, Apr. 2006. [Online]. Available: http://www.altera.com/literature/an/an357.pdf(10.03.2007).

[2] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Depend. Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan.–Mar. 2004.

[3] V. R. Basili and B. T. Perricone, "Software errors and complexity: An empirical investigation," *Commun. ACM*, vol. 27, pp. 42–52, 1984.

[4] M. Baumeister, P. Fuhrmann, and R. Mariani, "A single channel, fail-safe microcontroller to simplify SIL3 safety architectures in automotive applications," in *Proc. VDI Fachtagung Elektronik im Kraftfahrzeug*, Baden-Baden, Germany, 2007.

[5] S. Brewerton, "Dual core processor solutions for IEC61508 SIL3 vehicle safety systems," in *Proc. Embedded World Conf.*, 2007.

[6] J.-M. Burkhardt, F. Deétienne, and S. Wiedenbeck, "Object-oriented program comprehension: Effect of expertise, task and phase," *Empir. Softw. Eng.*, vol. 7, pp. 115–156, 2002.

[7] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Jul.–Aug. 2003.

[8] A. En-Nouaary, F. Khendek, and R. Dssouli, "Fault coverage in testing real-time systems," in *Int. Conf. Real-Time Computing Systems and Applications (RTCSA'99)*, 1999, pp. 150–157.

[9] M. Fagan, Design and Code Inspections to Reduce Errors in Program Development, IBM, 1976, Tech. Rep.

[10] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 797–814, Aug. 2000.

[11] P. Gawkowski and J. Sosnowski, "Assessing software implemented fault detection and fault tolerance mechanisms," in *Proc. 12th Asian Test Symp. (ATS'03)*, 2003, pp. 462–467.

[12] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante*, Software-Implemented Hardware Fault Tolerance*. New York: Springer, 2006.

[13] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjects—A comparative study of students and professionals in lead-time impact assessment," *Empir. Softw. Eng.*, vol. 5, pp. 201–214, 2000.

[14] IEC61508, Functional Safety for Electrical/Electronic/Programmable Electronic Safety-Related Systems, International Electrotechnical Commission, 1998.

[15] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Trans. Softw. Eng.*, vol. 12, no. 1, pp. 96–109, Jan. 1986.

[16] B. Littlewood and D. R. Miller, "Conceptual modeling of coincident failures in multiversion software," *IEEE Trans. Softw. Eng.*, vol. 15, no. 12, pp. 1596–1614, Dec. 1989.

[17] B. Littlewood, P. Popov, and L. Strigini, "A note on modelling functional diversity," *Reliab. Eng. Syst. Safe.*, pp. 93–95, 1999.

[18] M. Lyu, J. Horgan, and S. London, "A coverage analysis tool for the effectiveness of software testing," *IEEE Trans. Reliab.*, vol. 43, no. 4, pp. 527–535, Dec. 1994.

[19] K.-H. Moller and D. J. Paulish, "An empirical investigation of software fault distribution," in *Proc. IEEE 1st Int. Software Metrics Symp.*, May 1993, pp. 82–90.

[20] B. Nicolescu, R. Velazco, M. Sonza-Reorda, M. Rebaudengo, and M. Violante, "A software fault tolerance method for safety-critical systems: Effectiveness and drawbacks," in *Proc. IEEE 15th Symp. Integrated Circuits and Systems Design*, 2002, pp. 101–106.

[21] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalarprocessors," *IEEE Trans. Reliab.*, vol. 51, no. 1, pp. 63–75, Mar. 2002.

[22] T. J. Ostrand and E. J. Weyuker, "The distribution of faults in a large industrial software system," in *Proc. Int. Symp. Software Testing and Analysis (ISSTA)*, 2002, pp. 55–64.

[23] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *Proc. Military and Aerospace Programmable Logic Devices (MAPLD) Int. Conf.*, 2005.

[24] F. Salewski and S. Kowalewski, "Achieving highly reliable embedded software: An empirical evaluation of different approaches," in *Proc. Int. Conf. Computer Safety, Reliability and Security (SAFECOMP'07)*, Sep. 2007, vol. 4680, LNCS, pp. 270–275, Springer.

[25] F. Salewski and S. Kowalewski, "The effect of hardware platform selection on safety-critical software in embedded systems: Empirical evaluations," in *Proc. IEEE Symp. Industrial Embedded Systems (SIES'07)*, Jul. 2007, pp. 78–85.

[26] F. Salewski and A. Taylor, "Fault handling in FPGAs and microcontrollers in safety-critical embedded applications: A comparative survey," in *Proc. IEEE 10th Euromicro Conf. Digital System Design (DSD'07)*, H. Kubatova, Ed., Aug. 2007, pp. 124–131.

[27] F. Salewski, D. Wilking, and S. Kowalewski, "Diverse hardware platforms in embedded systems lab courses: A way to teach the differences," *SIGBED Rev., ACM,* Special Issue: The First Workshop on Embedded System Education (WESE), vol. 2, pp. 70–74, Oct. 2005.

[28] F. Salewski, D. Wilking, and S. Kowalewski, "The effect of diverse hardware platforms on N-version programming in embedded systems—An empirical evaluation," in *Proc. 3rd. Workshop Dependable Embedded Systems (WDES'06)*, Nov. 2006, vol. TR 105/2006, pp. 61–66, Vienna Univ. Technol., Vienna, Austria.

[29] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanović, and M. Vokáĉ, "Challenges and recommendations when increasing the realism of controlled software engineering experiments," in *ESERNET 2001–2003*, 2003, vol. 2765, LNCS, pp. 24–38.

[30] U. Wappler and C. Fetzer, "Software encoded processing: Building dependable systems with commodity hardware," in *Proc. Int. Conf. Computer Safety, Reliability and Security (SAFECOMP'07)*, Sep. 2007, vol. 4680/2007, LNCS, pp. 356–369, Springer.

[31] C. Wohlin, P. Runeson, M. Höst, M. Ohlson, B. Regnell, and A. Wesslén*, Experimentation in Software Engineering*. Norwell, MA: Kluwer, 2000.

[32] XILINX, Correcting Single-Event Upsets Through Virtex Partial Configuration, Application Note 216, Jun. 2000. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp216.pdf(05.03.2007).

Falk Salewski (M'08) received the Dipl.-Ing. degree in electrical engineering from the University of Siegen, Siegen, Germany. Currently, he is pursuing the Dr.-Ing. degree at RWTH Aachen University, Aachen, Germany.

He is a research assistant at the Embedded Software Laboratory at RWTH Aachen University. His main research interests are the different impacts of microcontrollers and FPGAs on the fault handling in safety-critical embedded applications.

Stefan Kowalewski received the Dipl.-Ing. degree in electrical engineering from the University of Karlsruhe, Karlsruhe, Germany, and the Dr.-Ing. and Habilitation degrees from the University of Dortmund, Dortmund, Germany.

He is a Professor and head of the Embedded Software Laboratory at RWTH Aachen University, Aachen, Germany. His main research interests are safety-critical embedded systems, model-based design of embedded software, discrete and hybrid systems, and formal verification.