Online Self Tests for Microcontrollers in Safety Related Systems

T. Tamandl, P. Preininger

***Abstract*—Nowadays it has become common practice to use microcontrollers in safety related systems. Such systems are deployed to achieve functional safety. The international standard IEC 61508 provides requirements to avoid or reduce the risk caused by these systems. The project SafetyLon aims at enhancing the Local Operating Network (LON) to meet the requirements of IEC 61508. For achieving a defined level of integrity either highly reliable hardware with a low residual error probability has to be used or comprehensive online self tests have to be performed. The chosen standard microcontrollers must perform online self tests in order to detect as much hardware faults of the CPU internals as possible. These tests cover the testing of the volatile and non volatile memory and the CPU. All online self tests have to be tailored to the chosen microcontroller and have to meet the requirements given by IEC 61508.**

I. Introduction

Former safety related systems were set up using conventional discrete electromechanical components. Such systems guarantee that failures in components do not result in an unsafe system behavior that leads to the risk of injuring people or damaging environment. In the early 1980s the use of microcontrollers became popular. More and more conventional wired logic was replaced. That resulted in the problem, that no proven failure models were available for microcontrollers. In 1984 Holscher and Rader described requirements for microcontrollers in safety related systems [1]. On the basis of their work several different standards such as DIN V VDE 0801 and EN 954 were developed which finally led to the introduction of IEC 61508.

Nowadays microcontrollers are accepted in safety related systems. Nevertheless microcontrollers in such systems have to fulfill special requirements according to IEC 61508 [2]. It must be avoided that failures of the microcontroller internals result in an unsafe state of the equipment under control (EUC). Undetected failures bear the risk of injuring people or damaging environment. To overcome this risk the microcontrollers used in safety related systems have to perform self tests. Doing so allows the detection of failures of the microcontroller and in case the EUC can be shut down in a safe way.

Manuscript received 2007-01-25

Thomas Tamandl is with Vienna University of Technology, Institute of Computer Technology, Gußhausstraße 27-29, 1040 Vienna, Austria (e-mail: tamandl@ict.tuwien.ac.at).

Peter Preininger is with Vienna University of Technology, Institute of Computer Technology, Gußhausstraße 27-29, 1040 Vienna, Austria (e-mail: preininger@ict.tuwien.ac.at)

Performing comprehensive self tests consume a lot of computational power. The effort depends on the complexity of the microcontroller, e.g. its instruction set, the size of the volatile and non volatile memory. Thus online self tests have to be adjusted to fit these conditions. For example, big memory areas cannot be tested as a whole because of the long execution time of the tests [3]. Long testing times contradict the requirement of a short reaction time. Consequently proven testing strategies must be redesigned in order to be applicable for the selected microcontroller.

Due to the trend to achieve the goal of area wide automation of safety critical applications, the integration of safety in field bus systems is necessary. The EC founded project SafetyLon is concerned with the development of a SIL 3 compliant field bus system based on EN 14908 [4]. For handling the communication and fulfilling all requirements of IEC 61508 the use of powerful microcontrollers is necessary. Within the project ARM7 microcontrollers are used due to their good cost/performance ratio. This document describes how to apply online self test to the volatile, the non volatile memory and the CPU (central processing unit) according to IEC 61508 on ARM7 microcontrollers.

The rest of the document is structured as follow. First an introduction of IEC 61508 is given. Afterwards the project SafetyLon is introduced and on this basis the technical application and implementation of the microcontroller online self tests according to IEC 61508 is described.

II. IEC 61508

The standard [2] defines safety in general as “The absence of unacceptable risk of physical injury or damage to health of people […]”. Within the document safety refers to functional safety that is a part of the overall safety. It does not refer to electrical safety. The international standard IEC 61508 provides a framework how to develop safety related devices. This framework is built around a safety life cycle that guides developers through design, implementation, maintenance and decommissioning of a safe system. Every stage specifies requirements necessary for avoiding systematic failures and handling stochastic failures. So the risk for an injury of humans or damage of environment can be minimized. The aim of safety related devices is to lower the inherited risk of the EUC below the maximum tolerable risk.

By performing a hazard and risk analysis of the EUC the associated hazards and risks are identified. For achieving

safety the whole system must be considered. Hazards can cause unsafe states which can result in dangerous situations. The hazard analysis also reveals the origin of the identified hazards. Finally the risk assessment defines the performance requirements for the safety functions.

To overcome the identified hazards safety functions are executed by the safety related systems. The behavior of these functions is based on the requirements derived from the hazard and risk analysis.

The performance of the safety functions is categorized into four different safety integrity levels. These levels define the likelihood of successfully performing the safety functions. According to IEC 61508 safety integrity level 1 (SIL 1) is the lowest and safety integrity level 4 (SIL 4) is the highest one. These levels correspond to a specific error probability per hour, refer to Table I. IEC 61508 categorizes devices in two groups, low demand and high demand or continuous mode, depending on the frequency of safety operations. SafetyLon devices are designed to meet the requirements for high demand or continuous mode.

TABLE I
SAFETY INTEGRITY LEVEL ACCORDING TO IEC 61508

Safety integrity level (SIL)	High demand or continuous mode (Error probability per hour)
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

The highest possible safety integrity level is limited by the hardware fault tolerance and the achievable safe failure fraction (1), refer to [2] for details. The safe failure fraction (SFF) defines the percentage of failures that do not result in a dangerous situation.

$$SFF = \frac{\lambda_{SD} + \lambda_{SU} + \lambda_{DD}}{\lambda_{SD} + \lambda_{SU} + \lambda_{DD} + \lambda_{DU}} \quad (1)$$

λ_{SD} defines the safe detected, λ_{SU} the safe undetected, λ_{DD} the dangerous detected and λ_{DU} the dangerous undetected failures. The higher the hardware fault tolerance, refer to [2] for details, the lower is the required safe failure fraction in order to achieve a high safety integrity level. For an example see section III.

Due to online self tests dangerous failures can be detected. So the number of λ_{DD} can be increased and on the other hand the number of λ_{DU} decreases. This enhances the safe failure fraction proportional to the detection rate of dangerous failures. The detection rate of dangerous failures is described with help of the diagnostic coverage (2). λ_D is the sum of all dangerous failures ($\lambda_{DD} + \lambda_{DU}$).

$$DC = \frac{\lambda_{DD}}{\lambda_D} \quad (2)$$

Depending on the required level of the required safe failure fraction appropriate self tests must be selected. According to IEC 61508 the self tests are classified into 3 categories depending on the quality: low, medium and high. That corresponds to a diagnostic coverage (DC) of 60%, 90% and 99%.

The information provided is the base for selecting appropriate self tests. More information concerning IEC 61508 are beyond the scope of the document. The rest of the document focuses on the online self tests themselves.

III. CASE STUDY

The project SafetyLon aims at integration safety mechanisms into the Local Operating Network (LON). LON technology was developed in the early 1990s by the American company Echelon. It is one of the most common network solutions used for building automation. Due to its open interoperable standard it is used in a lot of buildings of public interest such as airports, hospitals and office buildings.

With the progress in the area of automation and networking, the integration of safety aspects into field bus systems takes place. Field bus systems like CANopen-safety or PROFISafe are already designed to meet the requirements for SIL 3. For further information on CANopen safety and PROFISafe refer to [5], [6] and [7]. The project SafetyLon, a collective research project of the sixth framework program funded by the EC, aims at making LON technology safe.

As described in section II SIL 3 can be achieved either by reaching a high safe failure fraction or by designing a system architecture with a specified hardware fault tolerance. Within the project a 1oo2 (1 out of 2) system architecture with a hardware fault tolerance of one is chosen. Hence the required safe failure fraction for a SIL 3 compliant system amounts to $90\% \leq 99\%$ [2]. Equation (1) and (2) show the relation between the safe failure fraction and the required diagnostic coverage and vice versa. According to [2] for complex devices it can be assumed that 50% of all component faults cause a dangerous state. The evaluation of the required SFF for the SafetyLon project results in a DC of at least 80% (medium to high DC).

Appropriate self tests for testing the microprocessor internals such as the volatile or non volatile memory and the central processing unit (CPU) that offer the required diagnostic coverage are presented in section IV.

The major concern for selecting self tests is that the runtimes of the online self tests must meet strict timing requirements in order not to compromise the operation of

the safety fuctions. Regardless of the performed self test, SIL 3 must always be guaranteed.

IV. Implementation of Online Self Tests

This chapter describes how to implement self tests according to IEC 61508 for the selected ARM7 microcontroller. The controller runs at 48 MHz core clock frequency, the equipped FLASH memory is working at 33 MHz. At a maximum 256 Kibyte of FLASH memory and 64 Kibyte of RAM must be tested. Referring to section II all tests are designed in order reach a SFF of 90% ≤ 99%.

A. Implementation of the RAM test

Testing the volatile data storage follows the walking pattern method presented in [1], [2] and [3]. According to IEC 61508 using this method results in a medium diagnostic coverage. The RAM test is the test which is most critical of all tests. First of all the effort of the test is very high ($2n^2+5n$, where n is the number of tested bits). Secondly no interrupt is allowed, because an interrupt service routine (ISR) could change a memory cell which is currently under test. These ISRs usually are very time critical and because of that the RAM test is only granted a specific execution time. For the SafetyLon project this time is 200 µs. During this time all interrupts are disabled.

First a feasibility study is implemented in order to test if the implementation is able to fulfill the timing requirements. This feasibility study is implemented rather straight forward. The test algorithm uses one register for the bit to write (this bit is shifted through the register during the test) and one register for reading back the memory cells after altering one cell. One register is used for the written cell for comparison and another register is used for the comparison of the checked memory cells. The test is implemented in a loop. Fig. 2 shows the Nassi-Shneiderman diagram of the first implementation of the algorithm. If one of the grey marked comparisons fails the test is considered as failed.

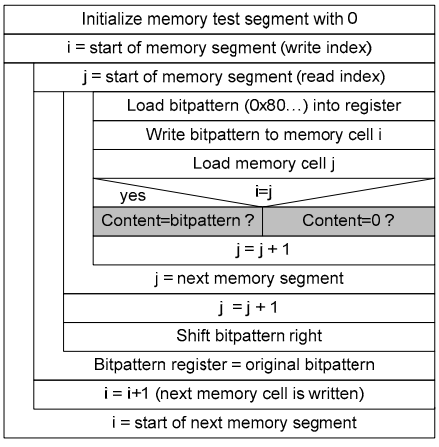

Fig. 2 Nassi-Shneiderman of RAM test (walking 1)

The shown algorithm is executed from the FLASH. Due to the reduced access speed of the FLASH the memory test is slowed down. In fact the running '1' using this straight forward method needs an execution time which is longer than 2 ms for a test size of 32 byte.

An optimization of the implementation is indispensable in order to reduce the execution time of the test. Either the test size of this part has to be reduced or the test has to be executed more efficiently. The first method can be done by splitting the memory test into smaller tests parts. Fig. 1 shows the split RAM test for a test size of 8 bit. First the '1' runs through the first 4 memory cells, in the second run the '1' runs through the upper 4 bit. Run 3 and 4 are used for the walking '0'. The RAM test is destructive. For the data contents of the test area a backup area is reserved within the RAM. Before the test is executed, the test segment is copied to the backup segment. After a successful test the segment is written back.

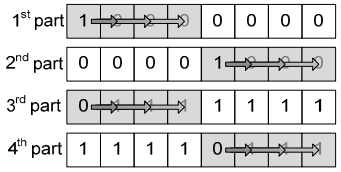

Fig. 1 RAM test splitting

The second method is the optimization of the program code itself. This method is chosen for the SafetyLon project. For the optimization an exact knowledge of the microcontroller itself and its assembler language is necessary. The implementation presented above uses standard instructions for memory manipulation – the single load and store commands. The trick for the speed up is using the ARM7 multiple load and store commands. For additional information on multiple load and store command refer to [8].

Using the block transfer commands makes it possible to read the complete memory segment under test (32 byte) into the registers with a single command. Fig. 4 shows the Nassi-Shneiderman diagram of the optimized RAM test – only the walking 1 is displayed here. The inner loop is responsible for shifting the '1' through the appropriate memory cells (the memory cell given with the index i). The outer loop is responsible for selecting the memory cell the '1' is shifted through. If one of the grey marked comparisons fails, a failure is detected and the test is considered as failed. This is implemented by the use of the conditional execution of instructions.

Using the algorithm displayed in Fig. 4 results in additional need of program storage. Due to the use of the block read command which is only able to load the memory contents into the registers in a specific order, the register which is compared to the bitpattern has to be changed for every memory cell written. So in fact, for the 32 byte test

segment the code size according to Fig. 2 is multiplied by eight. A comparison of the original and the optimized algorithm shows that the optimized algorithm is executed within a time of approximately 200 µs.

B. *Testing the program memory – Flash test*

The test of the program memory (no matter if the program is stored in the RAM or in the ROM) requires other mechanisms than the RAM test. In opposite to the RAM test, where hardware integrity is cheched, the test of

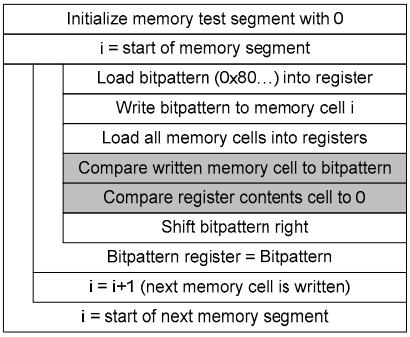

Fig. 4 Nassi-Shneiderman of the optimized memory test

the program memory verifies the data integrity. Cyclic redundancy check calculations are used for this test. The frame protection of the SafetyLon frames already uses 8 and 16 bit cyclic redundancy checks (CRCs). In order to minimize the programming effort, for the ROM test a 16 bit CRC is chosen. The selection of the CRC polynomial is crucial for the quality of the test. [9] offers a wide selection of different 8 and 16 bit polynomials. A very efficient 16

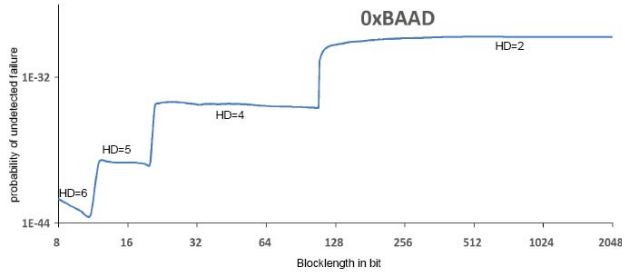

Fig. 3 error probability of undetected failures for a bit error probability of 10^{-6}.

bit CRC polynomial proposed by [9] is 0xBAAD. The polynomial is presented as an irreducible factor – for the final presentation and calculation the polynomial has to be multiplied by (x+1), resulting in 0x1755B. The performance of this CRC polynomial is shown in Fig. 3. For the figure the bit error probability is assumed with 10^{-9}. The figure presents the probability of undetected failures at different block lengths. As displayed in Fig. 3 at 256 byte the probability of undetected double failures is $2.1*10^{-21}$. According to [9] the polynomial 0xBAAD has the best performance of currently used 16 bit CRC polynomials. Therefore it is chosen for the ROM test.

In order to take the CRC limitations into account the program storage is split into logical regions. As displayed in Fig. 5, for each logical code segment a CRC is generated and stored at the end of the program storage. During the online test the CRC is recalculated and compared to the stored value.

A bitwise algorithm is used in order to save code space, even though this means a longer execution time. The implemented CRC algorithm supports intermediate CRC results. With the intermediate results the CRC calculation can be split into several calculations.

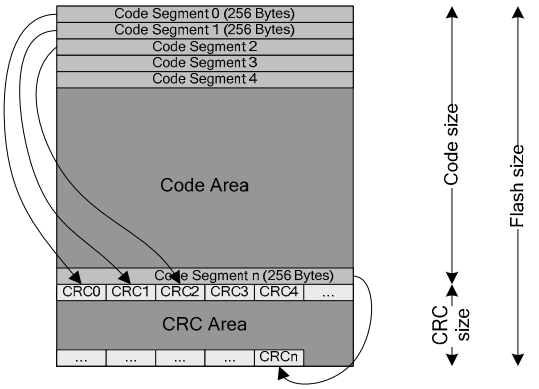

Fig. 5 ROM segmentation

Fig. 6 shows the split CRC calculation. The same CRC module can be used for the calculation. The first call is initialized with the initial CRC value (this value is specified as 0xFFFF for the SafetyLon project). Every other calculation is given the intermediate CRC result from the calculation before. Only the last calculation has to be done differently, because a number of '0's has to be augmented to the input data.

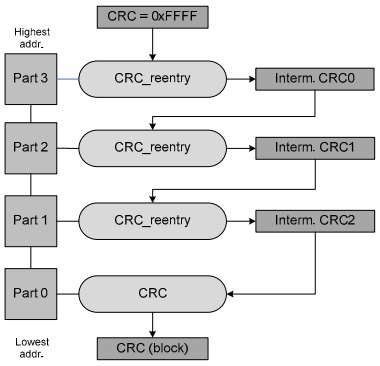

Fig. 6 intermediate CRC calculation

C. *Implementation of the CPU tests*

The CPU test consists of the test of the registers used, the stack test, the flag test and the test of the CPU's arithmetic logic unit. This section shows the implementation of the CPU tests for the SafetyLon project.

1) Register Test

The register test causes the greatest timing effort of the CPU tests. The galloping pattern (Galpat) test method is chosen for the SafetyLon project. The Galpat test is similar to the walking pattern method. The only difference is that after every verification of a memory bit with the Galpat test, the originally written bit also has to be verified. This leads to a test effort of $4n^2+n$, where n is the number of tested bits.

The ARM7 core used has got a total of 16 registers including additional banked registers. Basically all registers used have to be tested. Due to the register layout of the microcontroller, the registers R13 to R15 are reserved for special purposes (stack pointer, link register, program counter – also refer to [8]). Therefore only the registers R0 to R12 can be tested following the test method, the registers R13 to R15 cannot be tested. In order to reduce the effort of testing only the supervisor mode of the ARM7 core is used. Hence only the register bank for the supervisor mode must be tested. Like the RAM test also the register test has to be split into several parts in order to fulfill the strict timing requirements.

The main problem with the implementation of the register test is that the registers, which have to be tested also have to contain the comparison values. In order to save execution time, the main memory is not used. Therefore a lot of data swapping between the registers is performed. Like in the RAM test, program code is duplicated in order to save execution time. The implementation of the Galpat test for the registers is given in detail in [10].

2) Flag test

Testing the flags is done by setting and resetting the CPU flags. Additionally the conditional execution of instructions has to be verified. The following code sample shows the verification of the set ability of the carry and the zero flag.

```
// check combined C & Z
ldr  R0,=HIGHEST_NEG   // 0x80000000
movs R0, R0, lsl #1    // set the z and c-flag
bne  CPU_TEST_ERROR    // Z clear?
bcc  CPU_TEST_ERROR    // C clear?
bmi  CPU_TEST_ERROR    // N set?
bvs  CPU_TEST_ERROR    // V set?
```

D. Stack test

Testing the stack pointer without manipulation of the program flow is not possible. The stack area therefore is only verified by the program flow (e.g. return addresses are stored on the stack – failures here would cause an incorrect program behavior).

A rather common stack failure is that the stack pointer runs out of its limits. In order to deal with this failure, the layout of the stack is modified as shown in Fig. 7. The complete stack is initialized with a specific value. This is done during the processor startup before the stack pointers are set. After the stack initialization the stack pointer is incremented two times (the specific value is pushed two times to the stack).

During the test the stack pointer is verified to be within the valid stack area. Furthermore the forbidden stack areas are checked for the special value. If one of these memory cells contains a different value, the stack pointer ran out of bounds and the stack test is finished unsuccessfully.

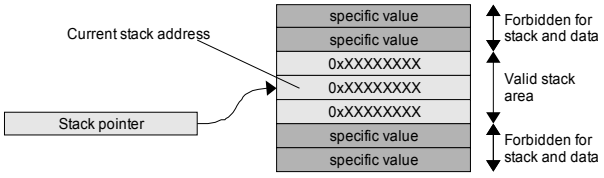

Fig. 7 Stack layout

E. Test of the arithmetic logic unit

The test of the ALU tries to cover all physical paths within the CPU. Obviously the problem is, that the entire CPU layout is unknown and also too complex for a comprehensive analysis. In order to address this problem, the microcontroller's instruction set is divided into command classes. The selection of the different command classes is done by using the processor's opcode. This results in 16 command classes, as the ARM7 core has got a 4 bit opcode. Command classes are for example the AND and the MOV command. For every command class the input values are chosen in a way that every bit of the output performs a '1-0' and a '0-1' transition.

In the following a code sample is displayed (in order to save space, only the least significant byte is shown in the sample):

```
//ADDTEST
add   R2,R11,R1    //R2=0xFF+0x01=0x00
teq   R2,R0        //R2=0x00 ???
addeq R2,R5,R10    //R2=0x55+0xAA=0xFF
teqeq R2,R11       //R2=0xFF ???
addeq R2,R11,R11   //R2=0xFF+0xFF=0xFE
teqeq R2,R12       //R2=0xFE
addeq R2,R5,R5     //R2=0x55+0x55=0xAA
teqeq R2,R10       //R2=0xAA ???
bne CPU_TEST_ERROR
```

V. Conclusion

The goal of the project SafetyLon is to develop safe EN 14908 technology that fulfills the requirements of a safety related SIL 3 system. It proves that there is no need for using very expensive microcontrollers with extremely low failure rates. Modern cheap standard controllers which have adequate computational power are sufficient for performing modern control applications as well as the online self tests.

As described in section III the project SafetyLon uses a

special 1oo2 system architecture in order to lower the required safe failure fraction to 90% ≤ 99%. Nevertheless comprehensive testing of the microcontrollers used is mandatory. The tests comprise all microcontroller internals such as volatile and non volatile memory and the CPU. Performing online self tests reduces the number of undetected failures which results in a higher diagnostic coverage. This enhances the safe failure fraction.

The implemented tests are able to fulfill all the requirements specified within the SafetyLon project. They can be easily adjusted to other timing prerequisites if necessary. In this case the splitting of the tests is the correct approach. The tests can be split into more pieces if necessary. Still, even if the execution time of the split testjob is much lower than before, the overall test time stays the same (the overhead for task switching is ignored in this case). This creates an upper boundary for the test because the testcycles have to be finished within a specified time which depends on the failure probabilities of the hardware used. The SafetyLon online tests are designed in order to have enough reserve for future requirements. The implemented tests provide a medium to high diagnostic coverage and can easily be applied to any ARM7 microcontroller available.

TABLE II
EXECUTION TIME FOR ONLINE SELF TESTS

Tested component (single job / total size)	Execution time	Time for completing test
Online RAM test (32 byte → 64 Kibyte)	232 µs	0.95s
FLASH test (256 bit → 256 Kibyte)	190 µs	1.55 s
ALU	22µs	22 µs
Register test (13 registers)	132 µs	3.43 ms
Complete CPU test	164 µs	4.26 ms

Table II provides an overview on the runtimes of the self tests. The table lists the execution times of the split testjobs for the online self tests of RAM, ROM and CPU (ALU and register test together) in the second column. If applicable the size of the single testjob and of the complete component are presented. The third column shows the overall execution time of the online self tests of the given components. For example the split RAM test has an execution time of 232 µs and has to be executed approximately 4,000 times. This results in 0.95 s for the test of the complete RAM. The ALU test is always executed together with the register test. Both tests are executed 26 times (13 registers, one run for the running '1' and one for the running '0'), resulting in a total time of approximately 4.26 ms.

REFERENCES

[1] H. Hölscher, J. Rader, "Microcomputers in Safety Technique, An Aid to orientation for Developer and Manufacturer," TÜV Rheinland, 1986.

[2] "IEC 61508 – Functional safety of electrical/electronic/programmable electronic safety-related systems," 1999.

[3] P. Wratil, M. Kieviet, "Sicherheitstechnik für Komponenten und Systeme," Hüthig Verlag, Heidelberg, 2006.

[4] "EN 14908 – Open data communication in building automation, controls and building management – control network protocol," 2006.

[5] "CANopen Framework for Safety-Related Communication," CiA work Draft 304, CAN in Automation e.V., 2000.

[6] "PROFISafe, Profile for Failsafe Technology," PROFIBUS-Nutzorganisation Karlsruhe, v1.0, 1999.

[7] D. Reinert, M. Schaefer (Publisher), "Sichere Bussysteme in der Automation," Hüthig Verlag, Heidelberg, 2001.

[8] "ARM7TDMI (Thumb) Datasheet," January 1999.

[9] P. Koopman, T. Chakravarty, "Cyclic redundancy code (crc) polynomial selection for embedded networks", The International Conference on Dependable Systems and Networks, DSN-2004, 2004.

[10] P. Preininger, "Hardware Selftests For Safety Critical Fieldbus Nodes," M.S.thesis, Institute of Computer Technology, Vienna University of Technology, Austria, 2006.

Thomas Tamandl (M'07) was born in Vienna, Austria and received a master's degree in electrical engineering from the Vienna University of Technology in Austria in 2005. His major field of study is safety in filedbus systems.

He is working intensively in the field of safety in fieldbus systems as project assistant at the Institute of Technology, Vienna University of Technology.

DI Tamandl is expert in CEN/TC247/WG4.

Peter Preininger was born in Vienna, Austria and received a master's degree in electrical engineering from the Vienna University of Technology in Austria in 2006. His major field of study is safety in filedbus systems.

He is working intensively in the field of safety in fieldbus systems as project assistant at the Institute of Technology, Vienna University of Technology.