2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Efficient Fault Injection Methods for Safety Software Testing Based on Virtual Prototypes: Application to Powertrain ECU

Ons Mbarek, Robert Bosch GmbH, Stuttgart, Germany (Ons.Mbarek@de.bosch.com)

Dineshkumar Selvaraj, Infineon Technologies AG, Bangalore, India
(Dineshkumar.Selvaraj@infinieon.com)

Romero Chica Jose Miguel, Rajagopal Shenoy, Holger Riethmueller
Robert Bosch GmbH, Stuttgart, Germany
(JoseMiguel.RomeroChica@de.bosch.com, Rajagopal.Shenoy@de.bosch.com,
Holger.Riethmueller@de.bosch.com)

***Abstract*— Releasing cars without software and hardware defects requires close collaboration between semiconductor vendors, OEMs and tier-one OEM suppliers to provide safety mechanisms and related safe implementation approaches. Virtual prototypes (VP) and fault-injection (FI) are among diverse existing methods recommended by the ISO26262 standard to verify safety requirements. This paper presents a VP-based fault-injection methodology flow in order to validate the safe execution of system automotive software. Using examples of safety monitoring software that implements the level-3 E-Gas concept and runs on a powertrain electronic control unit (ECU) model, we explain our methodology and highlight limitations and future working axes for semiconductor and tier-one OEM VP products.**

Keywords—virtual prototyping; fault injection; automotive safety; E-Gas; monitoring concept; electronic control unit

I. Introduction

To comply with the ISO26262 automotive safety standard [1], the hardware and software components as well as their development process should be ISO26262-conform. In an automotive system, each electronic control unit (ECU) is composed of a set of hardware elements (e.g. microcontroller (MCU), Application Specific Integrated Circuit (ASIC), sensor) on which a complex software stack runs. Due to the high complexity of such systems and the high dependency between the different software modules of one ECU stack, system software testing should start at early development stages. It should be performed according to a well-defined methodology that fits the incremental process of integration and configuration of software packages. Such high complexity leads to a complex system safety concept which requires the development of the software that activates and configures the safety hardware mechanisms and software error reactions. Using ECU virtual prototype (VP) along with fault injection (FI) mechanisms helps accelerating the development and test of system safety-relevant software before the availability of the real device. In this paper, a well-structured methodology for the construction of VP-based fault injection (FI) test cases is presented. Mapping rules that ease the conversion of existing safety-related test software are defined and should be applied along with the proposed methodology. Our approach is explained by using a VP for the Bosch MDG1 powertrain ECU [2] based on AURIX^{TM} 2G MCU, as well as concrete examples of the level 3 (L3) E-Gas monitoring concept [3].

The paper is organized as follows: section II reviews the state of the art of FI methods and outlines our contributions. Section III introduces the e-Gas monitoring concept and the challenges of developing functional safety compliant ECU. In Section IV, we introduce the VP modeling approach of AURIX^{TM} 2G-based ECU. In Section V, we present our methodology flow using examples of our VP use case. Results from our experiments are described in section VI. Finally, section VII concludes the paper and outlines future work areas.

II. Related Work

Since the first release of the ISO26262 standard [1], a large number of FI techniques for software safety testing has been published. Many of them have targeted simulation-based FI at software level. They are mainly based on offline instrumentation of software with additional fault detectors and injectors across the different software layers [4] [5] [6]. Since modification of software is required, the main drawback of those methods reside in the non-proven degree of intrusiveness and interference with the original software under test. In addition to that, most of these methods rely on additional FI software components at the highest application level. Therefore, their use is limited during the incremental development and integration process of the ECU software, especially in a pre-silicon phase. At the hardware level, a wide spectrum of methods have coped with simulation-based FI methods, ranging from Gate Level (GL) [7] to Register Transfer Level (RTL) [8,9] and Transaction Level of Modeling (TLM) [10,11,12,13,14,15]. Among the proposed TLM techniques, non-intrusive models for error injection and detection are added to the system hardware model [16,17]. These techniques ensure indeed a high safety confidence level of a VP, but only when applied throughout the design and deployment flow of a VP. This paper shapes this need based on explicit examples of a real industrial use case. These techniques target as well FI methods for the verification of the hardware VP. Contrary to that, this work targets testing software features on a VP. In [18], authors have presented four VP-based FI methods and demonstrated their application based on examples of Infineon's AURIX^{TM} MCU. In general, the techniques in [18] lack the definition of a well-defined FI methodology. Therefore, our work complements [18] through the definition of such a methodology.

III. BACKGROUND

A. *The E-Gas Monitoring Concept*

The Electronic Gasoline (E-Gas) Monitoring Concept [3] is a safety concept standardized by the AKEGAS working group and applicable for gasoline and diesel ECU. It describes the monitoring system structure to ensure safety in automotive ECU. Since 2013, the E-Gas concept complies with the ISO26262 standard and any powertrain ECU can be based on it to claim ISO26262 compliance. As depicts Figure 1, the E-Gas concept consists of three levels (L1, L2 and L3) and relies on two independent hardware components: Function Controller (FC) and Monitoring Module (MM). The hardware-relevant monitoring within the FC (L3) incorporates self-tests between FC and MM. The MM consists in a separate silicon chip (e.g. ASIC or small-scale MCU). It incorporates an external watchdog for the FC, permanently uses L3 query-response communication with the FC, and controls separate disabling paths. *L1* is a Quality Management software (QM) including the primary functions of engine control. *L2* monitors the performance regulating functions of L1. The same controller (FC) typically runs L1 and L2. *L3* straddles the FC and monitoring module (MC). Thus, it is a combination of software monitoring tasks and hardware monitoring structures.

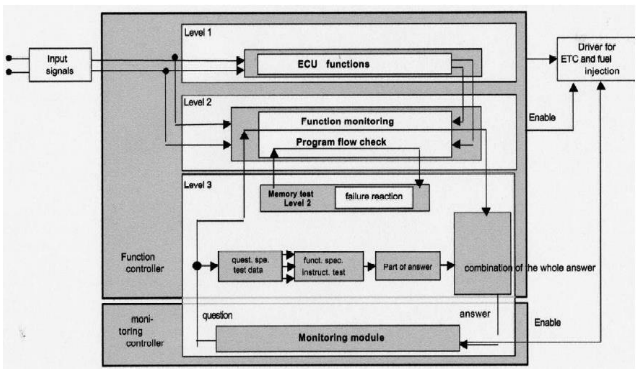


Figure 1. Block Diagram of the 3-level E-gas Monitoring Concept

B. *Challenges of Functional Safety Compliant ECU Platform*

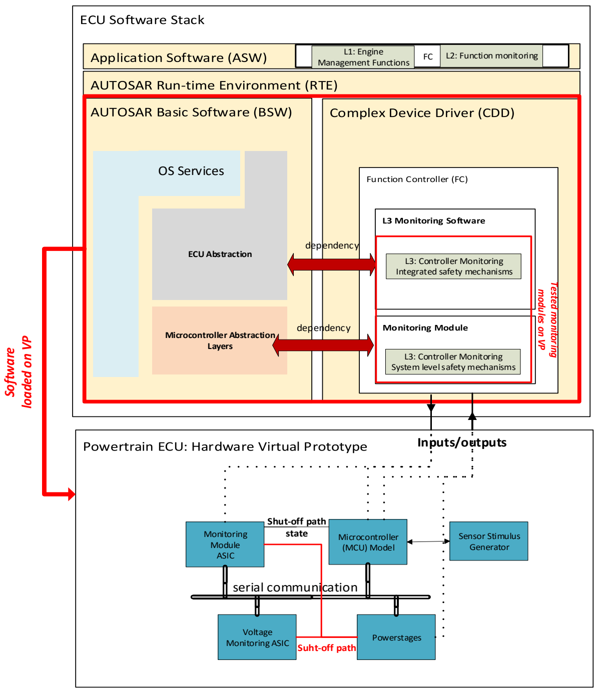
Figure 2 depicts hardware and software composition of ECU. The hardware consists typically of a main controller (e.g. MCU) connected to specialized control units (e.g. ASIC or MCU) for sensor measurements processing and actuations control. The AUTOSAR-based software [19] includes an application layer (ASW), a Run-Time Environment (RTE) and the Basic Software (BSW). The Complex Device Driver (CDD) cross layer provides means to implement project specific functionalities with direct access to RTE and ECU hardware. Drivers of complex sensors and ASIC are usually implemented in the CDD while abiding to AUTOSAR port and interface specifications. AUTOSAR-based software is developed according to the safety mechanisms specified in AUTOSAR, commercially available and ISO certified (e.g. CUBAS from Bosch [20], MICROSAR Safe from Vector [21] etc.). This reduces enormously its safety evaluation effort. Contrary to that, safety evaluation of CDD software and its interference with AUTOSAR layers presents a challenge to tier-one suppliers. Therefore, not only the definition of hardware safety mechanisms, but also the software that configures and monitors those hardware mechanisms and reacts to detected errors, should be developed according to a standard compliant with ISO26262 (e.g. E-Gas monitoring). Typically, the L3 monitoring software is hardware-dependent and usually developed as part of the CDD layer, while the L2 and L1 software are part of the QM application layer (ASW).

IV. A VIRTUAL PROTOTYPE FOR POWERTRAIN MDG1 ECU

Executing automotive safety software on a VP requires the modeling of specific features. In the following, the modeling approach of our VP use case is described from functional and safety perspectives.

A. *Overview on the VP Use Case Structure*

Figure 2 illustrates the basic structure of our VP use case. It is a TLM model of a powertrain Bosch MDG1 ECU [2] based on the AURUX^{TM} 2G MCU of Infineon Technologies Inc.. Components of this VP have different suppliers: the MCU model is provided by Infineon Technologies Inc., while the set of ASIC models for power stages and stimuli generators are developed by Robert Bosch Inc. according to software test driven requirements. The assembly of the complete VP is done according to a project specific layout. The AURIX^{TM} 2G MCU model is based on 32-bit Tricore^{TM} Fast-Timed Model (FTM) [23] from Synopsys$^{®}$ as well as a mixture of Loosely Timed (LT) and Approximately Timed (AT) memory and peripheral IP models from Synopsys Inc. (e.g. memories), Infineon Technologies Inc. and Robert Bosch Inc. A good tradeoff between timing and functionality accuracy and simulation speed was achieved in the MCU model. The validation of the processor model reaches more than 90% timing accuracy compared to RTL. The critical IP models are co-simulated with their correspondent RTL unit test environment, and calibrated against system RTL test benches. In order to run unmodified automotive software, the AURIX^{TM} VP was extended by a set of ASIC LT models for external monitoring, low-side and high-side power stages, ignition/injection and power supply.


ECU Software Stack
Application Software (ASW)
L1: Engine Management Functions
FC
L2: Function monitoring
AUTOSAR Run-time Environment (RTE)
AUTOSAR Basic Software (BSW)
Complex Device Driver (CDD)
OS Services
ECU Abstraction
Microcontroller Abstraction Layers
Function Controller (FC)
L3 Monitoring Software
L3: Controller Monitoring Integrated safety mechanisms
Monitoring Module
L3: Controller Monitoring System level safety mechanisms
dependency
dependency
Tested monitoring modules on VP
Software loaded on VP
Inputs/outputs
Powertrain ECU: Hardware Virtual Prototype
Monitoring Module ASIC
Shut-off path state
Microcontroller (MCU) Model
Sensor Stimulus Generator
serial communication
Voltage Monitoring ASIC
Suht-off path
Powerstages
Figure 2. Hardware and software components of a powertrain ECU

B. *Safety Mechanisms of the VP Use Case*

The MDG1 ECU family [2] is a scalable multi-core processor system that was designed to ease the development of a safe system conform to ISO26262 up to the Automotive Safety Integrity Level (ASIL)-D. Based on the L3 monitoring concept, its safety concept includes hardware-integrated safety mechanisms, which enable checks either by software, or by error-detection hardware, or by a combination of both. In the AURIX^{TM} 2G MCU, effort of developing software function tests (e.g. instruction-set and memory tests) is saved since dedicated hardware logic blocks (e.g. locked step cores, Built-In Self Test (BIST), Error Code detection and Correction (ECC)) are directly embedded in hardware. Nevertheless, configuration of hardware integrated safety mechanisms and error software reaction still needs to be protected and checked by the software during runtime. At the ECU level, additional hardware safety-mechanisms (e.g. watchdog timers, shut-off decoupling) should also be configured and controlled by safety software in order to ensure a certain product ASIL level. Safety software testing on the ECU VP requires therefore the availability in the MCU model of safety hardware-related features, which are relevant for safety software validation. For instance, the Safety Management Unit (SMU) is a central module of the AURIX^{TM} 2G MCU responsible for capturing all the hardware safety errors during software execution and needs to be modeled in the VP. Depending on software configuration, SMU can either notify the software or the hardware in case of error, or react directly by resetting some modules or the complete system.

V. Mapping Rules And Fault Injection Methodology Flow

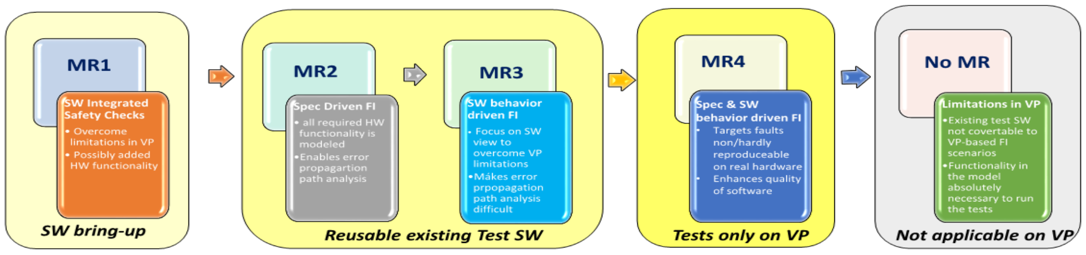

MR1
SW Integrated Safety Checks
• Overcome limitations in VP
• Possibly added HW functionality
SW bring-up
MR2
Spec Driven FI
• all required HW functionality is modeled
• Enables error propagartion path analysis
MR3
SW behavior driven FI
• Focus on SW view to overcome VP limitations
• Mákes error prpopagation path analysis difficult
Reusable existing Test SW
MR4
Spec & SW behavior driven FI
• Targets faults non/hardly reproduceable on real hardware
• Enhances quality of software
Tests only on VP
No MR
Limitations in VP
• Existing test SW not covertable to VP-based FI scenarios
• Functionality in the model absolutely necessary to run the tests
Not applicable on VP

Figure 3: Classification of mapping rules and related analysis flow

In order to test safety software on new hardware devices, software programmers tend to maximize the reuse of software tests used for old device variants. These tests consist in the original software augmented at specific locations with additional code lines, which either inject faults, or check error effects and reactions. In this section, we present our methodology flow to perform VP-based safety software testing.

A. *Mapping Rules for Efficient Safety Software Testing Based on VP*

As depicts Figure 3, we distinguished four different mapping rules to be applied for the efficient construction of safety test scenarios suitable to run a VP. They are described below using L3 monitoring software examples.

a) Mapping Rule 1 (MR1)

MR1 is applicable during the bring-up on a VP of a safety software. Most of the periodic checks performed by such software are hardware-dependent and might be constrained by a missing functionality in the VP model. Typical constraints are the stub models in the VP, where internal behavior and/or communication side effects are not modeled. MR1 states that a software modification will not be necessary in the presence of workarounds. Workarounds consist in converters of original software checks to alternative ones with the same validation purpose. These intercept at runtime the real software flow and trigger differently the monitoring checks. Figure 4 illustrates the real hardware design controlled by the ADC monitoring software. This software is responsible for the periodic control of the switch '**sw0**' by setting a test function register of the ADC module to pull '**sw0**' down and observing software reaction to a null voltage. If such register is missing in the VP, an alternative would consist in setting the analog input of the ADC (AN_x) to a null value just before writing to the test function register.

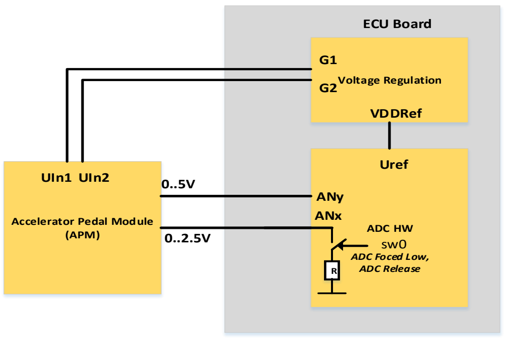


Figure 4. ADC monitoring hardware design

b) Mapping Rule 2 (MR2)

Software tests are usually implemented according to a detailed specification of safety requirements. Therefore, they implement all intermediate steps until error creation, detection and effect checks. A direct one to one mapping from the software test code to a script-based FI scenario is recommended in case no constraints are observed to

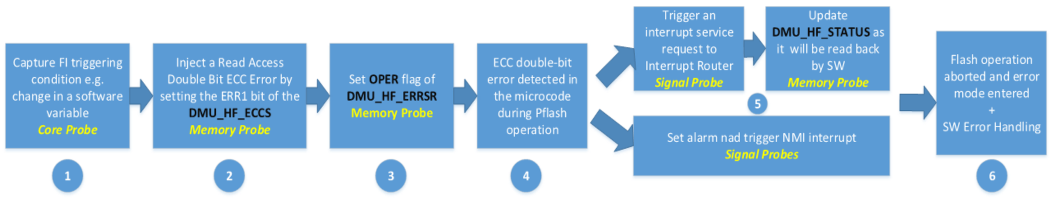

Capture FI triggering condition e.g. change in a software variable
Core Probe
1
Inject a Read Access Double Bit ECC Error by setting the ERR1 bit of the DMU_HF_ECCS
Memory Probe
2
Set OPER flag of DMU_HF_ERRSR
Memory Probe
3
ECC double-bit error detected in the microcode during Pflash operation
4
Trigger an interrupt service request to Interrupt Router
Signal Probe
Update DMU_HF_STATUS as it will be read back by SW
Memory Probe
5
Set alarm nad trigger NMI interrupt
Signal Probes
Flash operation aborted and error mode entered
+
SW Error Handling
6

Figure 5. PMU Safety Test Case Flow

apply the complete scenario flow. If it is not the case, MR3 should be considered. Figure 5 illustrates the test software flow to inject double bit ECC error during the program flash (PFlash) operation. Depending on the L3 monitoring software configuration, a reaction to this error could be handled, either by the cores as a response to received interrupt from PFlash, or by the SMU, which raises an alarm and results in a non-maskable interrupt (NMI). Figure 6 illustrates the hardware layout involving the mentioned modules. Considering the configured reaction, a direct one-to-one mapping of this scenario to a VP-based one could only be achieved in case of existing write callbacks to set the *DMU_HF_ECCS* register in the PFlash module. In general, status registers are read only e.g. *DMU_HF_ECCS*, and access to some registers is restricted and only allowed under specific rules e.g. safety protected registers. To ease VP-based FI, reading and writing in registers that might be relevant to capture a safety violation condition are still highly recommended. According to our experience, some IP model suppliers follow this recommendation while others offer implementations upon customer request due to the error-prone nature of such approach.

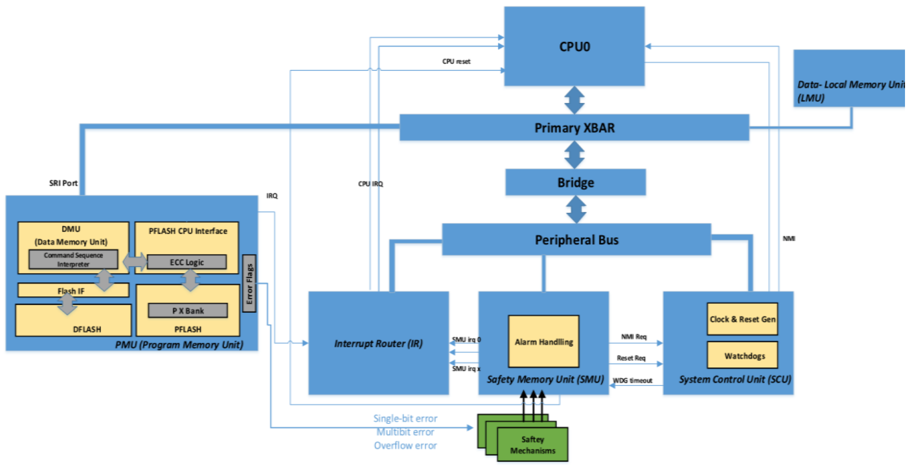


Figure 6. Interconnection between SMU, IR, CPU and PMU in the AURIX^{TM} 2G Architecture

c) *Mapping Rule 3 (MR3)*

MR3 is applicable in case of constraints to apply MR2. It consists in defining alternatives to trigger the error effect differently. This could be done by inspecting the error occurrence flow and defining a potential entry to inject a fault and get the same effect. There is often no need to perform all intermediate behavior of a fault, but more importantly consider the fault part that is directly obvious to the software and on which it reacts. The most important drawback of MR3 is the limited possibility of debug and analysis of error path propagation. A common use case for MR3 is the use of hardware error injection and detection logic by the software test. For instance, in the AURIX^{TM} 2G VP model, it was not possible to run a software test case that injects a fault into the LMU safety control register to generate an error in ECC. This was due to non-modeled error injection registers in the VP. An alternative way was to set the memory control register that corresponds to the hardware detection logic of such internal ECC errors. This was as well not possible since hardware provided error detection logic offered by the device were not present in the VP. The only way to inject this fault was by triggering its direct effect on software consisting in triggering the corresponding alarm input signal of the SMU (see Figure 6).

d) *Mapping Rule 4 (MR4)*

MR4 is used for the creation of new fault scenarios, typically faults which are hard to apply on real hardware e.g. short circuit, open load, over-and under voltage. The shut-off path test in the E-Gas L3 concept is a good candidate for MR4. Figure 7 illustrates this concept in case of MDG1. MDG1 implements a decoupling mechanism between power stages, the external watchdog and the MCU controlled by a

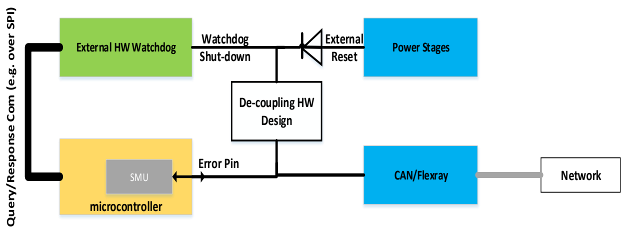


Figure 7. Shut-off Path and Query-Response Monitoring Concept

dedicated error-pin set by the SMU in case of sever faults. It is up to the system programmer to configure which events lead to an error pin event and how to recover to a safe state. Those hardware safety features can force direct error reactions e.g. CAN communication can be stopped and power stages can be switched off. An easy way to test those error reactions and dependency of monitoring software on other BSW packages, is to pull low the error pin during a CAN communication in one scenario and to pull low the power stage error-pin in another one.

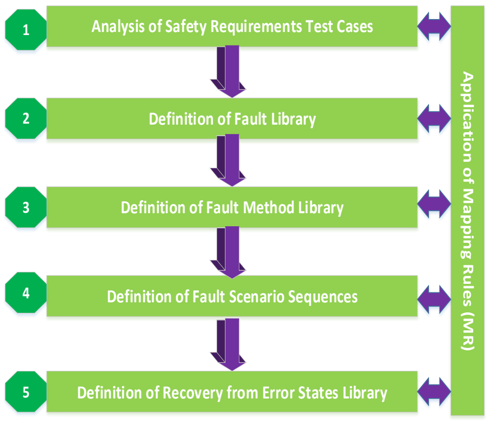

1
Analysis of Safety Requirements Test Cases
2
Definition of Fault Library
3
Definition of Fault Method Library
4
Definition of Fault Scenario Sequences
5
Definition of Recovery from Error States Library
Application of Mapping Rules (MR)

Figure 8. FI Application Methodology Flow

e) No Mapping Rule (no MR)

Due to specific limitations in the ECU VP, neither software tests can be converted to VP-based analog ones, nor new pure VP related test scenarios, can sometimes be imagined. The typical limitation is the non-modeled functionality inside the VP model. For instance, although creating over and under voltage situations is recommended to be applied on a VP (MR4), having in the AURIX^{TM} 2G VP only a stub model of the Power Management System (PSM) module, makes the creation of such fault situations impossible.

B. *The Overall Methodology for VP-based FI Tests Construction*

In order to construct Fault models and libraries based on existing software test cases, establishing and applying a well-structured VP-based FI methodology is required. Figure 8 illustrates the overall flow of our proposed methodology. It is composed of five sequential phases and a parallel phase. The parallel phase consists in applying the previously explained mapping rules as guidelines to construct a VP-based FI script across the sequential phases. In the following, each sequential phase is explained based on the example code on Figure 9, which implements a FI test scenario for the PMU based on the flow in Figure 5.

a) Step 1: Analysis of Safety Requirements Test cases

The first step consists in analyzing the software test case in order to identify necessary means (e.g. registers, signals, software global variable, etc.) which are responsible of injecting faults in the existing implementation. This is needed to understand the desired software behavior that should be triggered to cause the safety error reaction. Mapping the existing test scenario on the VP requires the application of the four mapping rules. The choice of the suitable mapping rule relies on an examination of the offered mechanisms and functionality in the VP and comparing them to software response/error detection and reaction requirements as explained.

b) Step 2: Definition of Fault Library

Based on the chosen mapping rule, a library specifying the FI inspectors should be defined. FI inspectors are responsible either to inject faults into the target module or to capture events that determine FI timing points. They are usually instances of monitor modules attached to interfaces of hardware models. These are for instance the so-called '*probes*' in the Synopsys® Virtualizer^{TM} IDE and could be created using a pre-defined VP tool python-based API. For instance, in order to inject the OPER flag error in the PMU, a probe on the target socket of the system bus (SRI) is needed (see Figure 9). Capturing the simulation timing point when to alter the OPER flag requires the creation of a probe on the CPU ("*coreprobe*"). It monitors any change in the '*tewstSMU_triggerECC*' software variable (indicating hence the start of the desired monitoring task). This is an example of a dynamic event that triggers a FI scenario, but could not be predicted offline.

c) Step 3: Definition of Fault Method Library

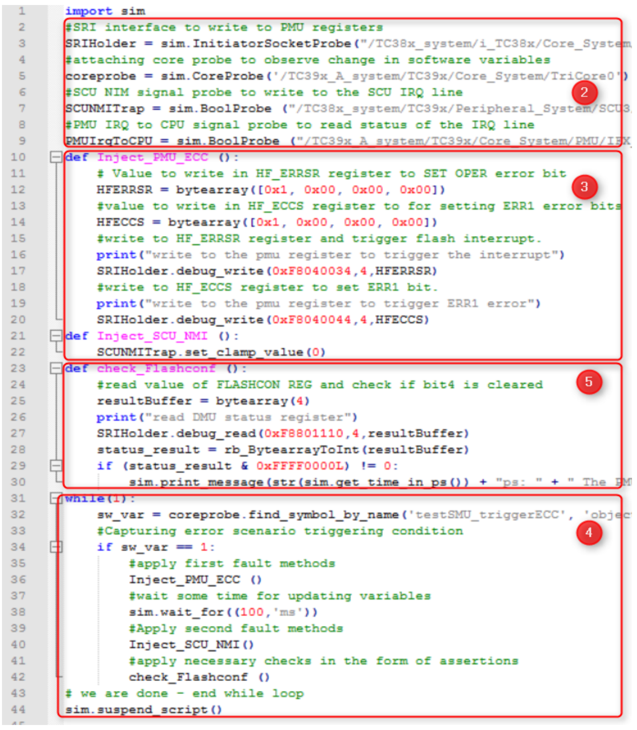
This step consists in defining the necessary set of callbacks that should be called whenever an event is captured by one of the probes defined in step 2. These callbacks are responsible to inject the faults by altering register or port values. They should be separated as atomic operations while considering their timing processing latency and side effect. As depicts the example on Figure 9, the non-intrusive TLM *debug_write* and *debug_read* interfaces, which bypass bus operations to directly get and set values in models, should be used to alter register values. Constraints to update register contents (e.g. read only or safety protected registers) should be considered.

d) Step 4: Definition of Fault Scenario Sequences

Given the probes defined in step 2 and the callbacks in step 3, step 4 specifies the test scenario flow (e.g. see flow on Figure 5) starting from a triggering condition of the FI, until calling pre-defined callback functions in the right sequential order, ending with applying checks (step 5) to validate specific conditions of error detection or reaction.

e) Step 5: Definition of Recovery from Error States Library

Step 5 focuses on the implementation of a library of checks applied for validating the desired safety reaction to the injected faults. The callback functions forming this library could then be called by the FI scenario defined in step 4. For instance, the periodic monitoring task of the L3 query-response communication between the external watchdog and the MCU (see Figure 7) is done by injecting faults in the responses to queries received by the external watchdog over the serial peripheral bus. Error software reaction is then checked inside the test software by reading specific status registers of the MCU. Analog checks could be implemented in the VP-based FI script, eventually to check the fault propagation in the dependent layers (see function *check_Flashconf* on Figure 9).



```
import sim
#SRI interface to write to PMU registers
SRIHolder = sim.InitiatorSocketProbe("/TC38x_system/i_TC38x/Core_System
#attaching core probe to observe change in software variables
coreprobe = sim.CoreProbe('/TC39x_A_system/TC39x/Core_System/TriCore0')
#SCU NIM signal probe to write to the SCU IRQ line
SCUNMITrap = sim.BoolProbe ("/TC38x_system/TC39x/Peripheral_System/SCU3
#PMU IRQ to CPU signal probe to read status of the IRQ line
PMUIrqToCPU = sim.BoolProbe ("/TC39x_A_system/TC39x/Core_System/PMU/IRX
def Inject_PMU_ECC ():
    # Value to write in HF_ERRSR register to SET OPER error bit
    HFERRSR = bytearray([0x1, 0x00, 0x00, 0x00])
    #value to write in HF_ECCS register to for setting ERR1 error bits
    HFECCS = bytearray([0x1, 0x00, 0x00, 0x00])
    #write to HF_ERRSR register and trigger flash interrupt.
    print("write to the pmu register to trigger the interrupt")
    SRIHolder.debug_write(0xF8040034,4,HFERRSR)
    #write to HF_ECCS register to set ERR1 bit.
    print("write to the pmu register to trigger ERR1 error")
    SRIHolder.debug_write(0xF8040044,4,HFECCS)
def Inject_SCU_NMI ():
    SCUNMITrap.set_clamp_value(0)
def check_Flashconf ():
    #read value of FLASHCON REG and check if bit4 is cleared
    resultBuffer = bytearray(4)
    print("read DMU status register")
    SRIHolder.debug_read(0xF8801110,4,resultBuffer)
    status_result = rb_BytearrayToInt(resultBuffer)
    if (status_result & 0xFFFF0000L) != 0:
        sim.print_message(str(sim.get_time_in_ps()) + "ps: " + " The PM
while(1):
    sw_var = coreprobe.find_symbol_by_name('testSMU_triggerECC', 'objec
    #Capturing error scenario triggering condition
    if sw_var == 1:
        #apply first fault methods
        Inject_PMU_ECC ()
        #wait some time for updating variables
        sim.wait_for((100,'ms'))
        #Apply second fault methods
        Inject_SCU_NMI()
        #apply necessary checks in the form of assertions
        check_Flashconf ()
# we are done - end while loop
sim.suspend_script()
```

Figure 9. Application of the methodology on an example

VI. Experimental Results

The methodology presented above and FI error scenarios deduced from original software test cases based on the four mapping rules, have been applied on AURIX^{TM} 2G-based powertrain ECU VP. The main resultant benefit was the enabling of system programmers to start checking issues related to safety monitoring and system reaction to defects six months before availability of real hardware, while even being not affected by delays in shipping the hardware prototypes. Up to 40% of critical L3 software reactions were tested on virtual prototype. Applying the remaining 60% test cases was mainly constrained by missing functional behavior in some of the MCU sub-models as well as to missing HW safety mechanisms in the ARURIX^{TM} 2G model (e.g. locked step cores, ECC logic, hardware error injection mechanisms). The flexibility offered by the virtual prototype model enabled the software testers to test up to 6 critical and important safety software error scenarios, which are hard to realize on the board level (e.g. shut-off path, Generic Timer Module (GTM) testing), and that by applying the MR4. Once a real hardware ECU became available, a quick software bring-up free of traps and uncomprehensive shut-down and resets was ensured, allowing hence a high quality, on-time and safe software release to OEMs.

VII. Conclusion and Future works

We presented a real industrial automotive use case for testing safety-monitoring software based on VP. We explained our FI methodology based on examples of AURIX^{TM} 2G MCU and MDG1 ECU safety concepts. Applied in a pre-silicon phase, we reported its benefits in increasing the released software quality and gaining TTM via a fast software bring-up on a real ECU. We recognized although the necessity to perform safety architecture exploration for the next generations of ECU, based on a well-established virtual prototyping flow. This will increase test bandwidth of hardware-dependent safety software testing. We highlighted as well the need for a serious consideration by VP suppliers of the ISO26262 qualification of their tools and models. Due to VP qualification lack and workaround-based conversion of some test scenarios, our approach is only an efficiency measure and safety software tests have to run in the end on the real silicon. Future work will focus on automating the application of mapping rules and on developing generic VP-based fault libraries for the post-silicon VP use e.g. continuous regression testing of software variants.

REFERENCES

[1] ISO 26262 standard for functional safety of road vehicles, 15 Nov.2011, pp. 1-10.

[2] JJ. Rueger, A. Wernet, HF. Kececi, T. Thiel, "MDG1: The New, Scalable, and Powerful ECU Platform from Bosch", Proceedings of the FISITA 2012 World Automotive Congress. Lecture Notes in Electrical Engineering, vol 194. Springer, Berlin, Heidelberg.

[3] E. W. Group, "Standardized E-Gas Monitoring concept for gasoline and diesel engine control units", Version 6.0, June 2015. https://www.iav.com/sites/default/files/attachments/seite/ak-egas-v6-0-en-150922_1.pdf

[4] A. Salkham, A. Pecchia, N. Silva. "Design of a CDD-Based Fault Injection Framework for AUTOSAR Systems", 32nd International Conference on Computer Safety, Reliability and Security, Toulouse, France, 2013.

[5] T. Piper, S. Winter, N. Suri and T. E. Fuhrman, "On the Effective Use of Fault Injection for the Assessment of AUTOSAR Safety Mechanisms," 11th European Dependable Computing Conference (EDCC), Paris, 2015, pp. 85-96.

[6] T. Piper, S. Winter, P. Manns and N. Suri, "Instrumenting AUTOSAR for dependability assessment: A guidance framework", IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Boston, MA, 2012, pp. 1-12.

[7] V. Herdt, H. M. Le, D. Grobe, and R. Drechsler, "On the application of formal fault localization to automated RTL-to-TLM fault correspondence analysis for fast and accurate VP-based error effect simulation - a case study", Forum on Specification and Design Languages (FDL), Bremen, 2016, pp. 1-8.

[8] N. Song, J. Qin, X. Pan, and Y. Deng, "Fault injection methodology and tools", International Conference on Electronics and Optoelectronics, Dalian, 2011, pp. V1-47-V1-50.

[9] D. Kammler, J. Guan, G. Ascheid, R. Leupers and H. Meyr, "A Fast and Flexible Platform for Fault Injection and Evaluation in Verilog-Based Simulations", Third IEEE International Conference on Secure Software Integration and Reliability Improvement, Shanghai, 2009, pp. 309-314.

[10] B-A. Tabacaru, M. Chaari, W. Ecker, T. Kruse. K. Liu, N. Hatami, C. Novello, H. Post and A. Schwerin, "Fault-injection Techniques for TLM-Based Virtual Prototypes", DVCon India, 2015.

[11] A. da Silva, and S. Sanchez, "LEON3 ViP: A Virtual Platform with Fault Injection Capabilities", 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, Lille, 2010, pp. 813-816.

[12] S. Reiter, A. Viehl, O. Bringmann and W. Rosenstiel, "Fault injection ecosystem for assisted safety validation of automotive systems," IEEE International High Level Design Validation and Test Workshop (HLDVT), Santa Cruz, CA, 2016, pp. 62-69.

[13] B.-A. Tabacaru, M. Chaari, W. Ecker, and T. Kruse, "Runtime Fault-Injection Tool for Executable SystemC Models," DVCon India, Bangalore, Sept 2014.

[14] B. A. Tabacaru, M. Chaari, W. Ecker, T. Kruse and C. Novello, "Speeding up safety verification by fault abstraction and simulation to transaction level," IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Tallinn, 2016, pp. 1-6.

[15] D. Mueller-Gritschneder, M. Greim and U. Schlichtmann, "Safety evaluation based on virtual prototypes: Fault injection with multi-level processor models," International Symposium on Integrated Circuits (ISIC), Singapore, 2016, pp. 1-2.

[16] J. H. Oetjens and al., "Safety evaluation of automotive electronics using Virtual Prototypes: State of the art and research challenges," 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, 2014, pp. 1-6.

[17] R. Weissnegger, M. Pistauer, M. Schachner, C. Kreiner, K. Römer, and C. Steger, "SAVESOC - Safety aware virtual prototype generation and evaluation of a system on chip", SpringSim, 2017.

[18] D. Selvaraj, K. Rachakonda, R. Babu and S. Puttappa, "Enabling Verification of Automotive Safety Application using Fault Injection VP", DVCon India, 2016.

[19] AUTomotive Open System Architecture (AUTOSAR) Development Partnership Website. http://www.autosar.org

[20] Bosch CUBAS: https://www.etas.com/data/press_room/AUTOSAR_Product_Portfolio_press_en.pdf_, in press.

[21] Vector Microsar Safe, Vector Webinar, https://vector.com/portal/medien/cmc/events/Webinars/2013/Vector_Webinar_MICROSAR_Safe_20131118_EN.pdf

[22] Technical Safety Concept Status Report AUTOSAR CP Release 4.3.1, 2017. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_SafetyConceptStatusReport.pdf

[23] "Synopsys' New Model for Infineon's Next Generation TriCore Architecture Accelerates Early Automotive Software Development and Test", in Press, July, 2016.