Automated traceability of requirements in the design and verification process of safety-critical mixed-signal systems

Gabriel Pachiana, Maximilian Grunwald, Thomas Markwirth, Christoph Sohrmann
Fraunhofer IIS/EAS, Dresden, Germany
{gabriel.pachiana, maximilian.grunwald, thomas.markwirth, christoph.sohrmann}@eas.iis.fraunhofer.de

***Abstract* - System-level design and verification of safety-critical hardware requires a consistent methodology which complies with industrial safety-standards, for example ISO 26262 for automotive applications. For certification of safety-critical systems, the development process has to implement and enforce a strict traceability of requirements, linking the requirement specification, the design implementation, and the verification testbench and results. This is particularly challenging in the AMS domain since it spans multiple engineering and tool environments. No standardized industrial solution is available for this task. In this work we propose an industry-compatible tool flow for an automated implementation of test cases from system-level requirements including traceability of the verification flow. This automated tool flow accelerates the design and verification process of safety-critical systems while making it robust against human error. The workflow is demonstrated in the design and verification of an AMS block of an automotive SoC, a 12-bit 4-stage pipelined ADC.**

***Keywords*- traceability, functional safety, mixed-signal, AMS, verification**

I. INTRODUCTION

The amount and complexity of electronic systems in the automotive domain today is primarily driven by applications such as advanced driver-assistance systems (ADAS) and autonomous driving. In order to avert physical damage of any road user, traditionally automotive manufacturers have to comply with strict functional safety standards such as ISO 26262 [1]. Given the increasing complexity of these systems, all the participants of the supply chain must support functional safety standards. This involves overcoming several challenges not only in adopting the standards with its safety management activities [2] but also supporting the exchange of safety-related information within tools, i.e. tools interoperability.

The ISO 26262 standard specifies the functional safety lifecycle and development phases of such complex systems. In the concept phase (ISO26262:Part 3) the automotive safety integrity level (ASIL, i.e. the level of risk reduction needed to achieve a tolerable risk), safety goals and functional safety requirements needs to be defined, these can be obtained by performing safety analysis like hazard and operability study (HAZOP), failure mode and effect analysis (FMEA) and fault tree analysis (FTA). For the next phases, safety analysis has to be iteratively carried out to review, refine and extend safety goals and requirements to achieve the level of risk reduction required [3].

The relations between safety goals, requirements and architectural components (subsystems, hardware components, software components) are crucial for understanding the system and how its safety is guaranteed. Therefore, ISO 26262 explicitly states that requirements traceability between these development artifacts needs to be ensured. One of the challenges of developing a safe system compliant with such a standard is to generate and adopt a workflow that contains full requirements traceability and state information of all the components of the development process, also across different modeling domains and tool environments.

This paper proposes a workflow for system design and verification (D&V) solving the special challenges to comply with requirements traceability. It goes through requirements engineering (RE), system-level D&V, and continuous integration (CI). As an example case the proposed workflow is shown employing Siemens' RE software Polarion, SystemC/SystemC-AMS for system-level analog mixed-signal (AMS) design, UVM-SystemC for system-level verification and Jenkins as the CI server. However, the workflow can be adapted for any combination of tools fulfilling similar tasks. In our example, system test cases are described in Polarion and automatically converted to UVM-SystemC test cases that contain full traceability information as well as specific configuration and stimulus

This work received funding from the German Federal Ministry of Education and Research (BMBF) within the project AnastASICA (16ES0990) as well as the European Union via EFRE within the project FLEUR (100325667).

implementations. The execution of the test cases (regressions) can be triggered from the RE tool or by a commit hook in the CI system, results are processed and fed back automatically to the RE tool.

This paper is structured as follows. Section II provides an overview of related flows and relevant approaches. Section III presents the proposed workflow from system requirements specification to system test case runs and reporting. In Section IV the proposed workflow is demonstrated in the design and verification of an AMS block. Finally Section IV provides conclusions and future improvements.

II. Related work

Requirements traceability is a well-known problem [4] but it remains to be a challenge when developing complex systems involving different engineering disciplines. Several tools exist in the market and in academia for requirements traceability between different domains and/or to generate implementation code from a description in Unified Modeling Language (UML) [5][6][7][8]. On the one hand, requirements traceability tools require to manually set traceability comments or link manually the different artifacts to implementation code [9][10]. On the other hand, automatic code generation tools from UML are mainly focused on the generation of architectural components of the design process but do not provide a complete solution including verification code. Furthermore, even for flows that are aware of traceability [5], there is a lack of tracing from requirements to the generated code. As a workaround, unambiguous naming conventions for modeling elements are defined in a deterministic way. However, still there is no reliable connection to the requirements using an RE tool.

The use of SystemC-AMS [11] and UVM-SystemC [12] has been demonstrated as a framework for system level D&V [13][14] but it does not exists a complete solution known to the authors that provides traceability.

III. Proposed workflow

The proposed workflow involves three major domains representing different disciplines and processes, these are Requirements Engineering (RE) domain, Design & Verification (D&V) domain and Continuous Integration (CI) domain. Those three domains are represented in Figure 1.

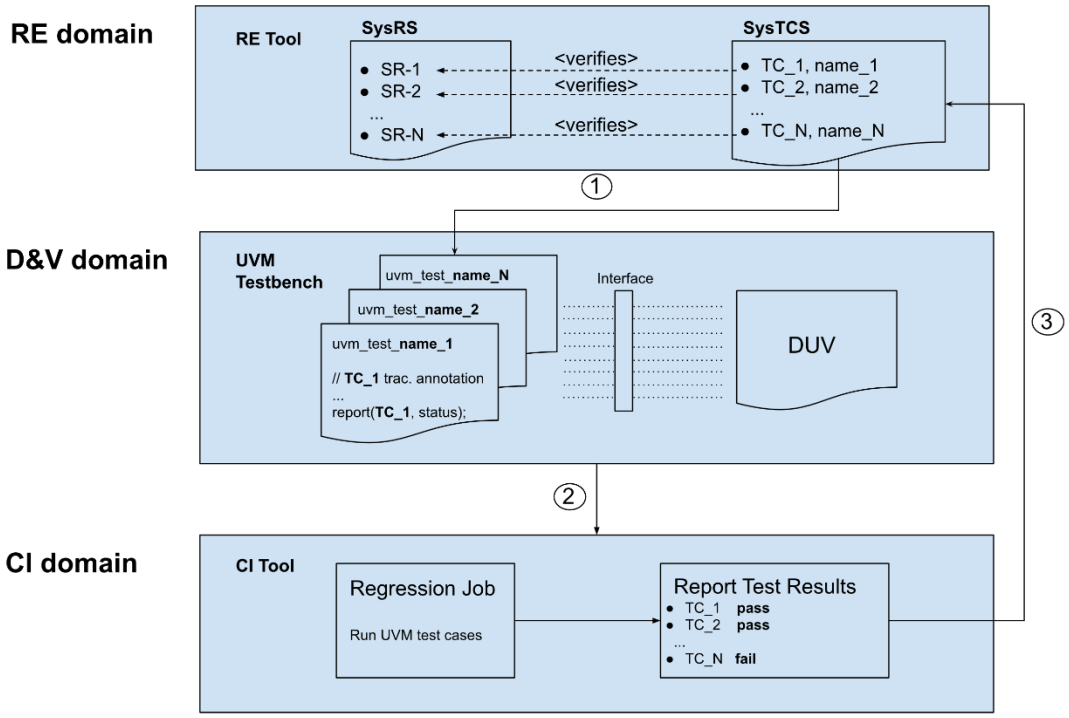


Figure 1: Proposed workflow.

A. *RE domain*

At system level, the activities in the RE domain usually start with the creation of the System Requirements Specification (SysRS) document. It is a structured collection of information that describes what the system is expected to do, its environment, performance parameters, and its expected quality and effectiveness. Even when it is written with textual description i.e. is not a formal definition of the system, writing complete, clear and unambiguous SysRS is crucial for the development process given that is the golden reference for the subsequent design and verification process. SysRS in practice is a "living document", i.e. it evolves over time. However, to keep it alive during the entire design process, the requirements must go through a voting and approval process. Any change to an approved requirement has to go through a change request process and an impact analysis which involves collecting all the work items linked to it and evaluating the effort on its change. For this reason, traceability is a fundamental property.

The next step is to define a System Test Case Specification (SysTCS) document. It collects test cases that are used for testing the full system or the collection of system requirements and specifies the details of the test approach. All the system requirements from the SysRS must be covered by test cases collected in the SysTCS, which provides an guaranty that the requirement is checked with a specific test case. Both documents are recommended to be managed using a tool that supports specific features and processes in the RE domain such as traceability (bidirectional link between SysRS requirements and SysTCS tests represented in Figure 1) and the approval process.

A system test case contains textual description and details of the test approach, e.g. test steps, description of each step and expected result, plus test records that show the pass/fail history of the test in the development lifecycle. The proposed workflow extends this definition by adding important parameters of the AMS testing approach for the automatic code generation. These contain:

- Parameter values for instantiating the design under verification (DUV) in the UVM testbench (TB) environment that will be implemented by the parametrization helper object.
- Configuration values for the input stimulus which are used to define an analog input signal e.g. sinusoidal, ramp, constant, etc.

An overview of the data structure of the system test case with the extensions for the proposed generation is presented in Figure 2.

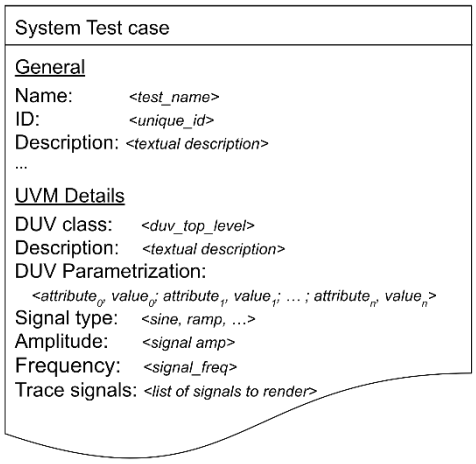


Figure 2: System Test Case data overview.

B. *Process ① : from RE domain to D&V domain*

In order to move from the RE domain to the D&V domain a process is required, that is visualized as ① in Figure 1. This process can be initiated from the RE tool and executed via scripting, which involves exporting the test cases from the SysTCS, parsing tests details, and filling custom templates of the UVM verification code (details are given in Section III.C).

An important aspect of this process is the format for exchanging requirements and functional safety information. Although there is an ongoing effort from Accellera's Functional Safety Working Group [15] as well as the existence of the requirements exchange format *ReqIf*, there is no standardization nor industry agreement on the definition of this interface. Thereby, the choice for our flow is to use SysML [16], an industry-accepted format which supports the modeling of requirements and tests with a *Requirement diagram*. However, this choice of format is not substantial for the overall flow.

Traceability information processing is also part of this process. The unique ID of each test case is exported from the RE tool to the SysML requirement description. Then the *UVM code generator* creates the test cases adding traceability annotations, this can be seen for example in Figure 1 *TC_1* test case. The generated code is automatically committed to the code repository, and a resource traceability script is triggered which parses the repository for the inserted annotations containing the URL to the referenced code line in the repository. This hyperlink is finally inserted back into the RE tool. In this way the proposed workflow provides a robust traceability implementation avoiding human errors. Manually editing the annotation pointing to a wrong or missing ID will produce an error in the parser script.

C. D&V domain

The generation of the verification code via the *UVM code generator* should be integrated into the RE tool, for example as a plugin that selects the test cases to export and launches the generator, or alternatively as a standalone application that is using an API to query the test cases and then launches the generator. The output of the script is the stable baseline of the verification TB which can be extended by the verification engineers. It includes the following main functions and classes of UVM-SystemC:

- *sc_main:* Entry point of the TB program, it calls the parametrization helper with the test name as a parameter, instantiates the design under verification, binds the interface with the design and calls the uvm_test with test_name as a parameter representing each derived system test case.
- *Parametrization helper:* Depending on the selected test case to run, it sets the parameters of the design under verification according to the system test case.
- *uvm_test:* Class for each *System Test Case* that includes the creation of a base test (test_base) that inherits from uvm_test, it provides the default test to run and is a template for the specialized tests defined in the RE tool. For each *System Test Case* a class inherits from test_base, each class contains a C++ annotation as a comment that makes reference to the system test case unique ID that will be parsed by a C++ traceability parser which will create a link between the System Test Case and its UVM-SystemC implementation. It also contains the instantiation of the sequence to be run in the sequencer and a report mechanism called in the UVM report_phase of the test to write the results in JUnit test result [17], a report format accepted by several CI and RE tools.
- *uvm_sequence:* For each System Test Case it creates the items with the specific stimulus that will be send to the driver. The stimulus contains information of an analog wave.
- *uvm_driver:* Gets the items from the sequencer and implements the analog signal to be inserted to the design under verification. This component has to convert items from a discrete event domain to a timed data flow (TDF) domain, which requires a specific data format conversion (for more details refer to [13]).

All the UVM classes are automatically generated creating the baseline testbench (Figure 3). The only manual work is inside the scoreboard, where the verification engineer has to implement the specific reference model since it is not possible to derive it from the RE tool directly.

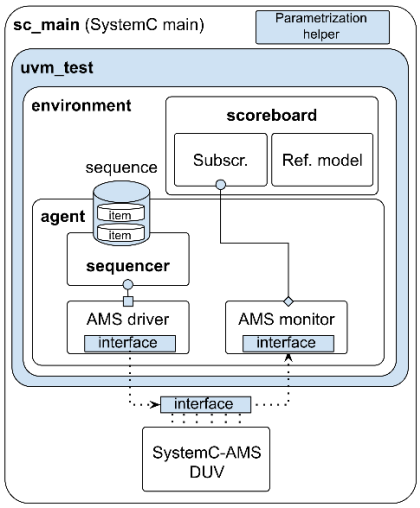


Figure 3: UVM-SystemC baseline testbench.

This stable baseline of the verification TB contains automatically generated bidirectional traces. However, in the case of design implementation code, the designer has to insert the traceability annotations given that the proposed generation flow does not store design information, e.g. the architecture / functional safety concept / functional safety architecture, in the RE tool and therefore is not able to derive the implementation code from it. Other authors have proposed generation of design code from SysML [5].

D. Process ② : from D&V domain to CI domain

The connection between both domains is made through the regression report that contains the results of each test run as *pass* or *fail*. This is also added on the implemented tests using the phasing mechanisms of UVM, specifically in the *extract_phase* the test gets the result from the scoreboard (*pass* or *fail*) and afterwards in the *report_phase* this result can be communicated using UVM_INFO macro (as proposed in [13]) or, in our proposed flow it is written into a file using the JUnit format.

JUnit is a XML format that includes a tree of test cases organized by test suite, and includes test cases with each status, and the stack trace produced by test cases which failed or errored. The schema can be found at [17] but it's not based on any standard known to the authors. In our proposed flow the regression run is structured on a single JUnit file that contains one test suite with all the UVM test cases results, each contains the unique requirement ID exported from the RE tool which is used to publish the results back into the RE tool.

The processing pipeline of building, running and publishing the regression results is triggered by the CI tool.

E. CI domain and process ③ to RE domain

Continuous Integration (CI) is a development practice widely adopted in software engineering. Developers integrate their code into a shared repository frequently - once or more per day - instead of submitting bigger changes less frequently. In this way the effort of integrating the new code into the main repository is smaller, as well as errors can be detected faster with higher probability. Each integration is then verified by an automated build and test run. Although automated testing is not strictly part of CI, it is typically implied.

In this domain, a version control system (e.g. SVN, Git) as well as a CI tool or automation server (e.g. Jenkins, GitLab) are needed. While the former keeps track of the changes in the code repository, the latter is connected to the version control system and triggers the build of the project, the execution of the test suite runs and reports the results. Some automation servers such as GitLab include the version control system by default.

The proposed flow includes reporting the results of the regression back into the RE tool (process ③ in Figure 1). This is either done via the CI tool pipeline or the RE tool can trigger the CI jobs and automatically pick up the results, as described in IV. When a test of a regression fails, it is marked as such in the RE tool and a defect/issue/bug must be created and linked to the System Test Case. This allows to have a complete and traceable status of the project inside the same RE tool, closing the loop from requirement and test case definition to test case implementation, run and reporting.

IV. CASE STUDY

The proposed workflow is demonstrated on a system-level design and verification project which consists of an model-driven development project of an automotive mixed-signal circuit, a 12-bit 4-stage pipelined analog-to-digital converter (ADC). In order to apply the flow, different tools and languages are employed, which are:

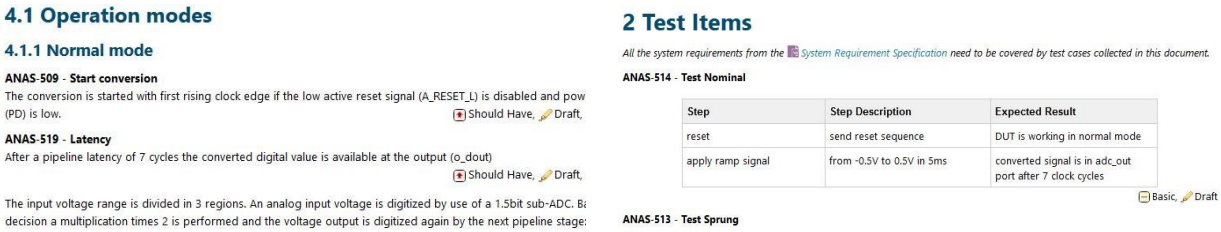
- RE tool: Siemens Polarion,
- Exchange of System Test Cases: XML and SysML,
- D&V language: SystemC/SystemC-AMS [11] and UVM-SystemC [12] (implementation of the test cases),
- IDE: COSIDE [18],
- CI tool: Jenkins,
- Version control system: Git,
- Exchange of results: JUnit.

Having defined the domains and processes in Section III, a detailed mapping of its application is given in the following subsections:

A - Implementation in RE domain and process ①

Most RE tools, such as Siemens' Polarion, provide a template for the development of a V-model project on which System Requirements Specification and System Test Case Specification documents are part of the Requirements section.

For Polarion, the information is structured as work items giving entity to the textual description, this is a data structure with a specific type and fields to fill with the proper information depending on the item type. For the System Requirements Specification the information is structured with work items mostly of type System Requirement, as an example in Figure 4 (a) part of this document is shown were a two work item of this type with unique IDs *ANAS-509* and *ANAS-519* can be seen. Work item also contains properties in which links can be added to other work items, in this case study *ANAS-509* is linked with the role "is verified by" to a System Test Case work item. In Figure 4 (b) the System Test Case Specification document is shown with the test case work item *ANAS-514* that verifies *ANAS-509*. In this way traceability from requirements to test cases is achieved using the links and its relations.



4.1 Operation modes

4.1.1 Normal mode

ANAS-509 - Start conversion

The conversion is started with first rising clock edge if the low active reset signal (A_RESET_L) is disabled and pow (PD) is low.

Should Have, Draft,

ANAS-519 - Latency

After a pipeline latency of 7 cycles the converted digital value is available at the output (o_dout)

Should Have, Draft,

The input voltage range is divided in 3 regions. An analog input voltage is digitized by use of a 1.5bit sub-ADC. Bi decision a multiplication times 2 is performed and the voltage output is digitized again by the next pipeline stage:

2 Test Items

All the system requirements from the System Requirement Specification need to be covered by test cases collected in this document.

ANAS-514 - Test Nominal

Step	Step Description	Expected Result
reset	send reset sequence	DUT is working in normal mode
apply ramp signal	from -0.5V to 0.5V in 5ms	converted signal is in adc_out port after 7 clock cycles

Basic, Draft

ANAS-513 - Test Sprung

Figure 4: (a) System Requirement Specification (left) and (b) System Test Case Specification (right)

Once the SysRS is approved and all requirements are covered by test cases specified in SysTC, the transformation process (see process (1) of Figure 1) can be launched. This process can for instance be triggered by a plug-in for the RE tool that takes as an input the collection of "System Test Case" work items and the main class of the design under verification, and generates the classes of the UVM-SystemC testbench.

The plug-in queries the test case work items and export them to XML format. In order to avoid an approach that is too strongly tied to a single RE tool, we transform the RE-tool-specific XML to SysML. Eventually the script takes the information needed to fulfil a set of UVM templates.

The importance of the transformation lies in its reusability. Given that a SysML-based format is used, another RE tool can be readily connected to the proposed workflow. This only requires the transformation from the tool-specific format to SysML. The SysML can then be used to generated the UVM-SystemC code for any tool flow.

B. Implementation in D&V domain

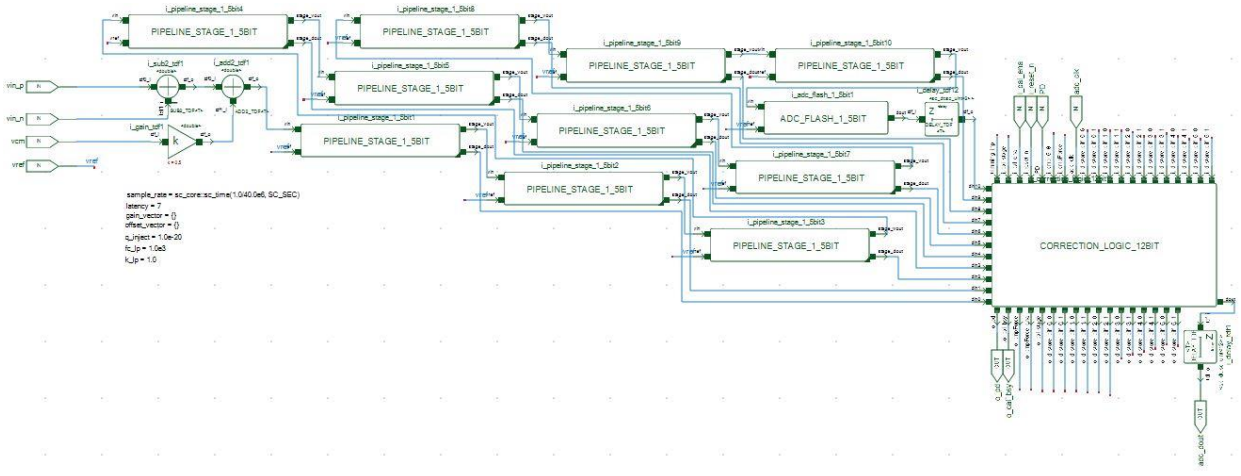


Figure 5: ADC System model in COSIDE IDE

The ADC is implemented using SystemC-AMS, an electronic system level (ESL) modeling language. It is an extension of SystemC with analog and mixed signal (AMS) modeling capabilities which makes it a good fit for the need of system level design of this project [11]. Also the electronic design automation (EDA) tool COSIDE [18] is used to ease modeling. In Figure 5, part of the architecture is described using a block diagram with custom blocks and blocks provided by the tool vendor. The IDE automatically creates SystemC-AMS code and templates, and provides simulation and debugging capabilities.

Another design task is to add comments inside the model code that will be later parsed and linked to the requirements as explained in Section III.C. On the verification side the *UVM code generator* outputs the UVM-SystemC classes presented in Section III.C. Following the example of the System Test Case *ANAS-514* the UVM test class definition is shown in Figure 6.

```
// Comment for Traceability parser
/**
* @wi.implements https://fraunhofer-eas.plmcloudsolutions.com/polarion/#/project/anastASICA/workitem?id=ANAS-514
* Test class to implement ANAS-514 - Test Nominal
*/
class test_nominal : public test_base
{
 public:

  test_nominal( uvm::uvm_component_name name = "test_nominal")
  : test_base(name) {}

  UVM_COMPONENT_UTILS(test_nominal);
```

Figure 6: Traceability annotation to be parsed

Figure 6 shows the traceability annotation to be parsed. Next, the UVM testbench details are completed by the verification engineer and the simulation can be launched. The example code in Figure 6 includes creating the sequence that implements an input stimulus of a ramp signal from -0.5V to 0.5V in 5ms described in *ANAS-514* .

C. Implementation in CI domain

The implementation in this domain is done using Jenkins, an open source automation server that provides features for building, testing and deploying software. In this case, a Jenkins job is configured to compile the design and verification code, run the test suite, collect the JUnit generated file and upload the results to the RE tool. The last step can be done by a script or via the RE tool's API. As a final step in the RE tool, the test cases are updated with a test record showing the status i.e. pass or fail. In case of a failure, a defect is created and linked to the failing System Test Case. Figure 7 (a) shows one of the System Test Cases that passed, while in Figure 7 (b) the last run of the test case failed creating automatically (by the RE tool) a defect *ANAS-515*.

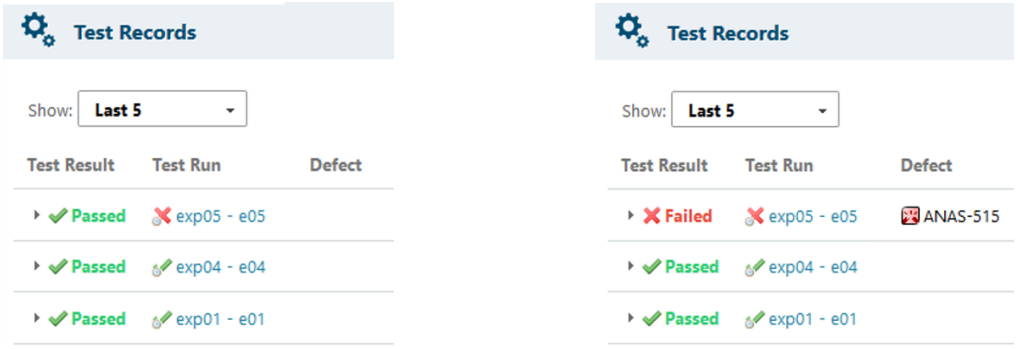


Figure 7: (a) TC pass ANAS-514 (left) and (b) TC fail ANAS-513 (right)

V. CONCLUSION AND FUTURE WORK

We have presented a system design and verification workflow for safety-critical AMS designs. Using the proposed approach we were able to demonstrate full traceability of the system requirements specification and system test cases from its textual description inside an RE tool to its implementation in UVM-SystemC. Moreover the execution of the test cases are automated using a CI tool and results are published back into the RE tool. Therefore, the complete state

of the project can be monitored and tracked by the RE tool, such as coverage as well as pass/fail status of all test cases. It is shown how the proposed automation in the creation of the UVM-SystemC testbench speeds up the verification process significantly and avoids potential human error that may be introduced by a predominantly manual work flow.

Future enhancements to this workflow may include (1) exchanging functional safety requirements with functional safety specific tools (e.g. [19]) and creating traces to functional safety analysis, (2) generating code templates from a SysML description like in [5] automating the design process and adding traceability annotations, and (3) adding/creating a consistency check in the RE tool that parses the code repository looking for traceability annotations and checking its correct correspondence to System Test Cases. Also a more generic approach for the description of the test stimuli is planned using the Portable Stimulus Standard (PSS) and a PSS tool to generate the tests.

ACKNOWLEDGMENT

The authors would like to thank to Dirk Voigt and Zeinab Labaf for their contributions in the development of plug-ins and scripting. The authors are responsible for the content of this publication.

REFERENCES

[1] ISO, “ISO 26262-1:2018 Road Vehicles – Functional Safety,” 2018, https://www.iso.org/standard/68383.html

[2] G. Xie, Y. Li, Y. Han, Y. Xie, G. Zeng and R. Li, "Recent Advances and Future Trends for Automotive Functional Safety Design Methodologies," in IEEE Transactions on Industrial Informatics, vol. 16, no. 9, pp. 5629-5642, Sept. 2020, doi: 10.1109/TII.2020.2978889.

[3] A. Nardi and A. Armato, "Functional safety methodologies for automotive applications," *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Irvine, CA, 2017, pp. 970-975, doi: 10.1109/ICCAD.2017.8203886.

[4] *O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," Proceedings of IEEE International Conference on Requirements Engineering, Colorado Springs, CO, USA, 1994, pp. 94-101, doi: 10.1109/ICRE.1994.292398.*

[5] R. Görgen, E. de Kock, “SysML based Architecture Definition and Platform Generation Flow”, 2019 Design and Verification Conference and Exhibition Europe, Available: events.dvcon.org/Europe/2019/proceedings/papers/11_1.pdf

[6] W. Mueller *et al*., "The SATURN Approach to SysML-Based HW/SW Codesign," *2010 IEEE Computer Society Annual Symposium on VLSI*, Lixouri, Kefalonia, 2010, pp. 506-511, doi: 10.1109/ISVLSI.2010.95.

[7] A. Abdulhameed, A. Hammad, H. Mountassir and B. Tatibouet, "An approach based on SysML and SystemC to simulate complex systems," 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Lisbon, 2014, pp. 555-560.

[8] IBM, “IBM Rational Rhapsody,” https://www.ibm.com/us-en/marketplace/rational-rhapsody

[9] Dassault Systèmes, “Reqtify”, https://www.3ds.com/products-services/catia/products/reqtify/

[10] Willert, “ReqXChanger”, https://www.willert.de/software-tools/seamless-integration-technologies/willert-reqxchanger/

[11] "IEEE Standard for Standard SystemC(R) Analog/Mixed-Signal Extensions Language Reference Manual," in *IEEE Std 1666.1-2016* , vol., no., pp.1-236, 6 April 2016, doi: 10.1109/IEEESTD.2016.7448795.

[12] Accellera Systems Initiative, ‘‘UVM-SystemC Library 1.0-beta3,’’ 2020. Available: https://www.accellera.org/images/downloads/drafts-review/uvm-systemc-10-beta3tar.gz

[13] M. Barnasconi et al., "UVM-SystemC-AMS Framework for System-Level Verification and Validation of Automotive Use Cases," in IEEE Design & Test, vol. 32, no. 6, pp. 76-86, Dec. 2015, doi: 10.1109/MDAT.2015.2427260.

[14] T. Machne et al., "UVM-SystemC-AMS based framework for the correct by construction design of MEMS in their real heterogeneous application context," 2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS), Marseille, 2014, pp. 862-865, doi: 10.1109/ICECS.2014.7050122.

[15] Accellera Systems Initiative, “Functional Safety Working Group”, URL: https://www.accellera.org/activities/working-groups/functional-safety

[16] Object Management Group, “System Modeling Language”, Available: https://www.omg.org/spec/SysML/1.6/PDF

[17] Windy Road Technology Pty. Limited, “JUnit-Schema”, Available: https://github.com/windyroad/JUnit-Schema

[18] COSEDA Technologies GmbH, “COSIDE® - The Design Environment for Heterogeneous Systems,” https://www.coseda-tech.com/

[19] ANSYS, “Medini Analyze,” https://www.ansys.com/de-de/products/systems/ansys-medini-analyze