CEAS Aeronautical Journal
https://doi.org/10.1007/s13272-022-00633-y

ORIGINAL PAPER

A semi-automated approach for requirement-based early validation of flight control platforms

Philipp Chrysalidis[1] · **Hauke Hoeber**[1] · **Frank Thielecke**[1]

Received: 24 February 2022 / Revised: 8 November 2022 / Accepted: 6 December 2022

Abstract
With the current trends in aviation like Single-Pilot-Cockpits and more autonomous functions in aircraft, flight control avionics are bound to become more complex. Future platforms will need to compensate for one or even both pilots, which will require systems that are more reliable. However, state-of-the-art development of flight control avionics does not yet support these demands efficiently. The development process involves numerous stakeholders who are communicating without streamlined interfaces. This leads to a slow and error-prone process during development. New methods are required to improve efficiency and to pave the way for future technologies. In this work, the authors introduce a semi-automatic toolchain which derives usable code for the configuration of devices from natural language requirements. The requirements are noted through modular components, stored as a model-based configuration file and are transformed into executables in the last step. This novel approach allows engineers to input their expertise when defining requirements, while removing tedious transformation tasks. Through automatic configuration testing, the validity of the approach is confirmed.

Keywords Early validation · Automatic toolchain · MBSE

1 Introduction

Avionic functions have become progressively more complex in recent years and with the planned advancements like Single-Pilot-Cockpits or completely autonomous aircraft, this trend will most likely continue. To allow for development of these futuristic systems, the development process itself also must be adapted and made future-proof, since modern Flight Control Systems (FCS) will have to substitute one or even both pilots reliably in civil aviation. While the required technology has not matured enough yet, its implementation must be made easier to allow for a feasible use once it is ready. Therefore, current development, which is done mostly manually by the various stakeholders involved in the process, must be changed to accommodate for the higher complexity, while also ensuring the same level of safety and reliability.

Consequently, a semi-automatic toolchain was developed which assists the stakeholders at the device development, verification and validation, as to reduce the workload in this specific domain. This is part of AvioNET, a broader model-based approach for connecting avionics development tools developed at the Institute of Aircraft Systems Engineering (FST) of the Hamburg University of Technology (TUHH). The holistic approach aims to solve the problem of increasing complexity with generic and formalized processes for the development of avionic systems. This is described in greater detail, after a short discussion of related work regarding toolchain development (Sect. 2) in Sect. 3. The methodology for the device development domain in the scope of this paper is presented in Sect. 4 and afterward demonstrated in a specific FCS use case in Sect. 5. The paper finishes with Sect. 6 where the conclusion and an outlook for further development are given.

Hauke Hoeber and Frank Thielecke have contributed equally to this work.

✉ Philipp Chrysalidis
philipp.chrysalidis@tuhh.de

Hauke Hoeber
hauke.hoeber@tuhh.de

Frank Thielecke
frank.thielecke@tuhh.de

[1] Institute of Aircraft Systems Engineering, Hamburg University of Technology, Nesspriel 5, 21129 Hamburg, Hamburg, Germany

Published online: 20 December 2022

2 Related work

An automated toolchain for the formal verification of avionic Simulink designs has been developed at the Masaryk University in cooperation with industry partner Honeywell [1]. The toolchain is targeted at safety-critical systems, as the manual verification for such complex systems is error-prone and time-consuming. A finished Simulink design acts as the input of the toolchain and is checked by constraints automatically derived from the system requirements. The approach is supported by proprietary Honeywell tools and provides the user with the information on the design validity. Should the design be invalid, an alternative model design is proposed.

Fortiss and various research and industry partners, including among others THALES, developed a Model-Based Systems Engineering (MBSE) approach to create the DREAMS toolchain for a mixed-critically system [2]. While not fully automated, the respective tools for the individual development steps have manual exchange interfaces. Therefore, output artifacts can be used as inputs when necessary. Additionally, the artifacts ensure the process traceability required by safety standards such as DO-178C [3]. Since the toolchain is applied to multi-core processors, it is far more detailed than any approach for a single-core platform would be, as multi-core constraints must be considered. Nevertheless, a trend for future development is emerging in research and in the aviation industry and must be taken into account.

Together with various industry partners, the University of Oxford developed and validated a requirement-based automated testing method [4]. Instead of manually producing test scripts for low-level requirements, they are automatically generated through the combination of natural language and model checking. Thereby, the overall testing effort is reduced compared to the common industry standard. Moreover, the authors prove, that automatic test case generation is indeed capable of satisfying verification objectives as required in DO-178C.

The Institute of Aircraft Systems at the University of Stuttgart is currently developing a MBSE toolchain for the development of Integrated Modular Avionics (IMA) platforms [5]. The toolchain is embedded in a toolsuite which aims to support multiple stakeholders in the IMA development process. Artifacts are generated in their respective development phase and can be distributed among tools with appropriate interfaces. Nevertheless, the stakeholders provide the majority of the system knowledge and remain in control of the process. Through the removal of tedious transformation tasks between parties, the focus of the manual labor can be focused on providing better engineering solutions. Thus creating a more suitable framework for semi-automated knowledge-based engineering.

An MBSE approach for an avionics development toolchain was already introduced in a prior work at the FST. Halle and Thielecke propose a seamless toolchain spanning the complete IMA development process, separating it into the four key segments Architecture, Configuration, Testing and Simulation [6]. By providing interfaces for all aligning segments, the manual workload will be reduced significantly, as tedious tasks (i.e., data transformation between different formats) are automated. Additionally, the development focuses on automating recurring and tedious processes inside the individual segments. Thereby, the stakeholders are not replaced, but instead have their development focus shifted toward knowledge-based problem solving. Furthermore, the artifact exchange between segments provides the opportunity of process traceability, which is a key avionic certification requirement.

The need for further automation in the development process is not unique to the aviation industry. It is also present in other sectors, such as the automotive industry. Aniculaesei et al. designed a SCADE-based toolchain with which the user is assisted in generating test cases [7]. The only remaining manual step is to derive and formalize system requirements, while various tools generate the tests automatically in a step-by-step process. Similarly to the previously presented works, the tools share common interfaces and artifacts are generated for traceability purposes.

3 Overall tool framework

As the trend toward further automation of the avionics development process is imperative, the toolchain developed at the FST is continually being improved. The authors advanced the methodology to encompass a holistic tool network, which loosely follows the V-model for development [8]. With the Avionics Next-Gen Engineering Tools (AvioNET) shown in Fig. 1 the already existing segments of

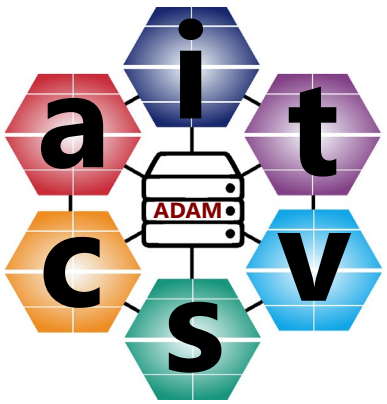

Fig. 1 AvioNET Tool network overview

Architecture (A), Configuration (C), Testing (T) and Simulation (S) in [6] are expanded by Verification and Validation (V &V), Visualization and Insights (I) and Avionics Data Management (ADAM).

All sections are highly interconnected and data is freely transferable between them. Moving from a toolchain to a tool network emphasizes the high degree of interconnectivity, which is necessary to create a seamless development process from beginning to end through a high level of formalized methods and automation. Therefore, individual tools need to have interfaces not only to tools from aligning segments, but they should also be compatible with the tools of unaligned segments. However, allowing this amount of flexibility by simply linking all the tools between each other would lead to an infeasible workload, which is why ADAM was introduced into AvioNET. As a central hub for data exchange and storage, the tools only require a single interface to ADAM, which allows for continuous data transformation through standardized interfaces. The toolchain presented in this paper (see Sect. 4) is set into this tool framework and therefore requires a generic MBSE approach to satisfy the interconnectivity requirements in its own segment and outside its own scope as well.

3.1 Validation and verification

In the context of AvioNET the toolchain is set in the Validation and Verification (V &V) segment, in which validation and verification processes are aimed to be introduced earlier in the development. This would decrease the workload and cost of design flaws, as they are detected more timely and therefore have a smaller effect on following steps. Consequently, validation and verification must be done continuously, which is made possible through the assessment of virtual products.

By establishing a virtual environment in which a virtual platform mirroring the real hardware is available, test cases can be executed independently of the real product even before it is finished. Additionally, the real hardware operating system can be emulated to provide the possibility of testing run time behavior of applications in a real-time environment. Once the real hardware is finished, it can be tested as part of a hardware-in-the-loop simulation, since the virtual and the real product are kept consistent with each other.

Through early validation, the common V-Model of development will be improved and modernized to contain a virtual branch, transferring it to a W-Model as depicted in Fig. 2. When failing validation in the virtual branch, the additional iteration cycle is far simpler as neither development nor integration on real hardware is required.

When using a W-Model, it becomes apparent why the high level of interconnectivity is required for AvioNET. As setting up a virtual environment is additional work, the invested time must be regained through a high degree of reusability of artifacts and methods, i.e., for testing.

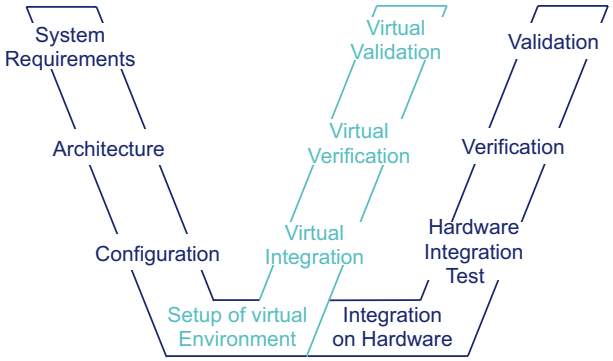

Fig. 2 W-Model for early validation

4 Easy configuration toolchain

The objective of the Easy Configuration Toolchain (EasyConfig) is to assist the configuration of avionics hardware and validating said configuration continuously, as described in the previous section. The necessary work flow is shown in Fig. 3 and closely resembles the theoretical structure established in subsect. 3.1. To initialize the toolchain the user sets requirements for the device. These are relatively low-level requirements, which are derived from the overall system requirements and are set manually.

The requirements are written in natural language and checked for relevant keywords representative of configuration elements. Thereby, the requirement documentation remains easy to understand while still providing a rule set for the transformation into formal definitions. When setting the requirements manually, it is important to formalize them accordingly, so that their attributes follow a predefined set of rules. The formalized natural language requirements can then be interpreted and transformed into formal language. Afterward, the formally defined system behavior and interface definition of the avionics platform lay the basis for the following model and test generation. The model generation is twofold. Both a virtual model and a real configuration are derived and are based on similar metamodels as to provide an easy way for integrating interfaces. The configuration for the virtual model is set to run in a virtual environment, which mirrors the real hardware environment. In keeping both the configurations and the environments consistent with each other, their information value remains the same. Thereby, the virtual tests partially replace the hardware tests as a form of early validation. Instead, the demonstration on real hardware ideally evolves to a mere formal step in the certification process, not leading to any alterations. This way

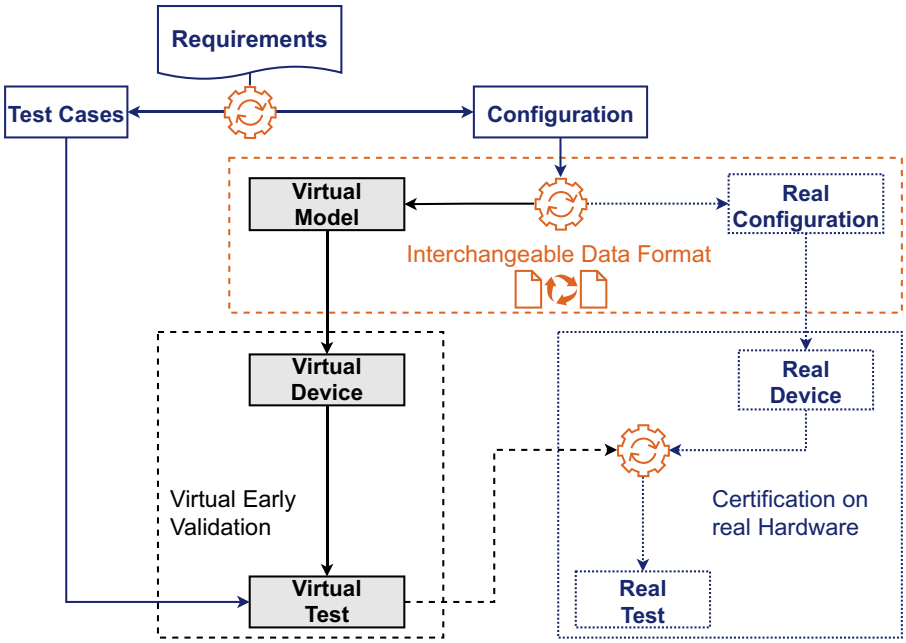

Fig. 3 Toolchain for automatic test generation

expensive iteration cycles with hardware adaption would be cut, but if retroactive changes prove to be necessary, both configurations are easily kept consistent through their interchangeable data format. Therefore, changes are transferable in both directions if adaptions are required. By following this work flow, EasyConfig is the real world application of the introduced W-Model, with a virtual and a real hardware branch. As demanded in the AvioNET concept, a high degree of interconnectivity is provided for in this toolchain and the metamodel-based configurations allow for simple data transformation and storage through easily generated generic interfaces.

The device will be tested via bare module tests, since the focus is the device itself and not the functionality of an eventual application. Therefore, the test application will be as complex as necessary but as simple as possible. This way, the application design is easily verifiable beforehand and does not require an additional major workload. Using this approach also guarantees adaptability through simplicity, as a simple application can be changed easier to fit the respective test case.

4.1 Automatic test case generation

The generation of the test cases follows a heuristic approach. Through already available information of the hardware, environment boundary conditions are derivable. These are predetermined trough keywords such as "OS-Partition" and further specified by more detailed definitions and the quantity. Device ports would be defined as an "I/O-Capability" with the respective communication protocol as the specification, i.e., "AFDX" or "CAN" with the amount of ports depending on the chosen hardware. These requirements are tested by running edge case scenarios, covering a possible range or the complete spectrum. When testing possible CAN IDs covering the whole range would produce a high amount of unnecessary data, because testing the edge cases already fulfills the goal. Using similar approaches for other requirements is key in reducing the overall test effort and speeding up automation.

The generated test cases are transformed into a MATLAB format and thereby available for toolchain processing. In the next step, user action regarding the required tests is needed. Since a metamodel approach was chosen and devices from different suppliers usually also have differing metamodels, the user must actively choose, which model elements fit the given test cases. As an example, the partition ports for an ARINC 653 [9] conform device can be examined. Each partition is required to provide either sampling or queuing ports. In the model these can be named either "Sampling Port", "Queuing Port" or "Partition Sampling Port" and so on. Each of these possible naming conventions differs and is therefore not necessarily identifiable by model interpreting algorithms. But an engineer operating the toolchain is capable of recognizing the fitting model name. Through this manual step no extensive model knowledge must be incorporated into the test generation code and therefore the approach is adaptive and applicable to all models following a matching metamodeling approach. This makes the toolchain highly generic, as it was demanded by the AvioNET requirements.

The results of the tests get recorded automatically in the virtual and hardware environment via the respectively

chosen test systems and are stored as separate artifacts and analyzed against the set requirements.

4.2 Early validation

In the following step, the virtual environment for the early validation is set up. As shown in Fig. 4, the early validation is almost completely independent of a real product and only the configuration for the virtual and the real device share a common root, as to keep the models consistent. Therefore, a virtual product must be created mirroring the capabilities of the real hardware, which in this case is a standard ARINC 653 capable device. While this model must be improved upon to exactly match the behavior of the specific device, it fulfills the task of validating the configurations as the real hardware restrictions are applied in the virtual world as well. Additionally, no real-time behavior has been implemented yet, but since the real hardware tests will be done later in the process, a preliminary validation still provides valuable information.

The early validation is done in MATLAB Simulink and requires no additional tools, once the requirements have been transformed. Therefore, the automation process is straight forward and easy to use, which makes it easy to integrate into the existing development process. Because of the chosen model-based approach, the virtual configuration is transformable into the real hardware configuration, meaning that possible changes can be included seamlessly in both environments without any additional workload.

4.3 Hardware integration and test

Following a successful early validation, the toolchain finally will be applied to real hardware and the main goal of AvioNET to reduce complexity and improve the development process is realized. Configuring a safety-critical piece of hardware such as a flight control device requires various tedious tasks from the engineer which are replaced through smart assistance by various tools and through a high degree of formalization, leaving less room for human errors. By linking various applications through strictly defined manual and automatic interfaces, the process becomes more clear and is easier to manage. The applications used in the toolchain are chosen based on their common use in the aviation industry and research. Requirements management is done in IBM Doors, since it has a user-friendly graphical user interface, while also allowing for exporting data in various machine-readable formats. However, this process step is not limited to IBM Doors and could be done in any other model-based descriptive language. While being noted in natural language, the requirement definitions must adhere to formulation rules, as to make automatic extraction possible. Therefore, the respective keywords must be used to correctly predefine

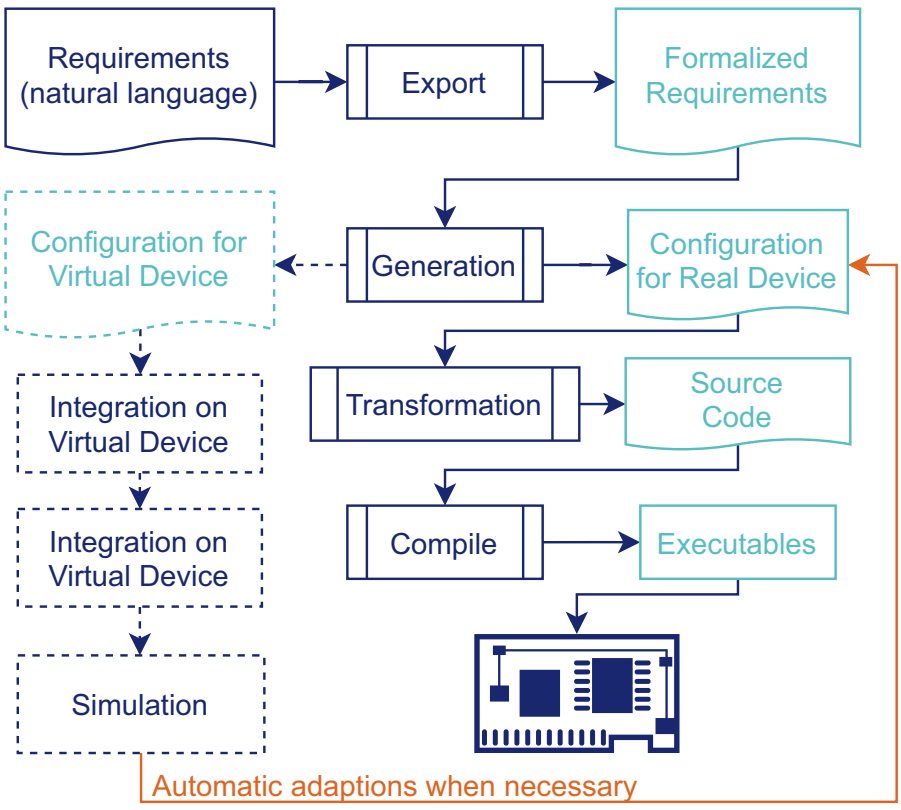

Fig. 4 Detailed work flow for EasyConfig

the device. On the basis of these formalized requirements, configurations and test descriptions are automatically generated.

Based on the formalized requirements configurations, test applications and test descriptions are generated via an automation procedure, which was developed in MATLAB. The automation environment executes the aforementioned keyword-based algorithms for each requirement. These algorithms must be defined by the test engineer once but are completely reusable as most of the necessary tests for e.g., a specific interface do not depend on the hardware. The set of automatically generated tests is not meant to be a complete set, but to replace tedious and repetitive tasks otherwise done by an engineer. The tests can be easily expanded manually, either by expanding the generation algorithm or by creating the artifacts by hand.

The configurations for the tests are generated in a systematic AvioNET specific hardware independent format, that aims to be easy to transform in a variety of hardware-specific configurations. The applications are generated as MATLAB Simulink models as Simulink is widely supported by the industry already, and it is also possible to generate code from the model that could be integrated.

The configurations and the generated test applications are used to generate a virtual representation of the device (see [10]), but also used to generate a load for the real hardware. As mentioned in subsect. 4.2 with the virtual representation of the device, it is possible to define and execute tests without the need for real hardware. The transformation between the proprietary virtual and the standardized real hardware environment is done via XML style sheet transformations (XSLT). The proprietary configuration and the test application is compiled with the HIGHTEC compiler to provide the load for the device. The load is flashed onto the device via the Universal Debug Engine (UDE) from PLS. This process is automated through the Component Object Model (COM) interface between the UDE and MATLAB. The test descriptions for both the virtual and the real device are adapted to their respective test systems, Simulink Test and the UDE respectively. Both of these test systems are highly flexible, which makes them applicable to every configuration, which is necessary to fulfill the generic requirements for EasyConfig derived from AvioNET. The UDE provides the possibility of recording all commonly used avionics communication protocols as separate channels. Using a common real-time recording device for all possible communication configurations, each specific result documentation remains comparable and consistent even between different devices and models. Additionally, this allows for using a device and model independent approach for analyzing the data, which is especially important for the real hardware, as the virtual environment is always modeled within the same framework.

4.4 Toolchain artifacts

As it was already pointed out in Sect. 2 ensuring traceability is of the utmost importance when developing safety critical avionics software. Therefore, the toolchain was developed with traceability as a necessary requirement. For every step, artifacts are created and stored independently of the previous and following steps, as to keep each artifact uncorrupted. For traceability, the corresponding requirement of the artifact is stored in the metadata of the artifact. The following artifacts are created:

- Requirements document
- Virtual device configuration
- Hardware device configuration
- Hardware device source code
- Hardware device executables
- Test specification document
- Test scripts for generated test cases
- Recorded test data

The readability of the different artifacts in their pure form varies drastically. While the requirements document is written in natural language and therefore is easy to understand for a human user, both the source code and the configuration files are generated in harder to read formal language. While the source code must be analyzed with common code reviewing techniques, the hardware configuration can be analyzed easily, as an additional Eclipse-based tool for reading and writing the document was developed for a more user-friendly experience. Thereby, traceability is possible through either common methods or specifically designed tools, which makes the toolchain applicable for a proper avionics development processes.

Additionally, to providing traceability, one of the artifacts also acts as an interface document between EasyConfig and AvioNET. As described in Sect. 3 EasyConfig is supposed to be implemented into the holistic MBSE tool network developed at the FST. To ensure this interconnectivity, the hardware configuration document is based on an Ecore-metamodel. This document design is key, as it provides the possibility for automatically transforming the data through metamodel-based interfaces between the different process segments in AvioNET.

5 Use-case for a flight control system

The toolchain was applied to an FCS platform to determine its usability in a realistic use case. The executable is configured for a platform used for hosting industrial Flight Control

Applications. Therefore, the obtained data about the validity of the toolchain is applicable to a proper industrial development process. In the scope of the process, the set requirements and the chosen application are low-level as to provide a simple-to-understand Proof of Concept (PoC).

5.1 Device requirements

The chosen device is an Aurix Tricore Board (ATB) designed by Infineon. As Commercial Off The Shelf (COTS) single-core hardware becomes less available, choosing a multi-core board is a future-proof method, as development and support will continue over the next decades [11]. Furthermore, research into multi-core systems has gained significant traction, as already discussed in Sect. 2 (see [2]). Proving the validity of a newly developed toolchain for such a device is of utmost importance, since the flexibility of providing configuration to both single- and multi-core platforms will become a necessity.

Since the objective is to develop an FCS platform, the original ARINC 653 specification for IMA systems is over defined. Instead, the platform devices will be specified by the subset ARINC 653 Part 4 [12]. This standard is specifically designed for systems with fewer capabilities and a lower complexity than usual IMA platforms. Thereby, it is possible to develop IMA-like applications and still meeting a well-defined and commonly used standard, which ensures the quality of the designed platform. Furthermore, testing conditions are already predefined in the standard and can be implemented seamlessly in EasyConfig.

For the communication between devices CAN was chosen, as the protocol is commonly used in avionics applications. An ATB provides two CAN-Interfaces without any additional hardware expansions, which satisfies the requirements for the chosen PoC. With the described constraints taken into consideration, the requirements listed in Table 1 were derived.

As already mentioned, these are simple requirements for proving the validity of EasyConfig. However, they are still sufficient for providing the necessary basis for this research. The blue marked parts for each requirement are the keywords the algorithm analyzes and on which the configuration is build upon. It becomes apparent that only small parts of the natural language sentence are actually formalized, and that each requirement is easily readable still. However, the keywords must be set beforehand and made available to the requirements engineer, so that proper formalization is possible. As demanded in AvioNET the process is overall more standardized and this allows for a higher degree of automation and therefore provides easier access to interfaces between the various process steps, while remaining intuitive for any user.

Some of these requirements are static and don't need to get reconfigured every test run, since they are the basis an ARINC 653 system is build on. Dynamic requirements include the CAN IDs, the message content, the baud rate, the message length and the periodic cycle for the partition process. Changing the parameters of these dynamic requirements produces the biggest amount of the unproductive workload, aimed to be reduced by EasyConfig.

Said dynamic parameters are depicted in Fig. 5 in square brackets in relation to each other. As previously discussed, testing the entire range of every requirement is undesirable,

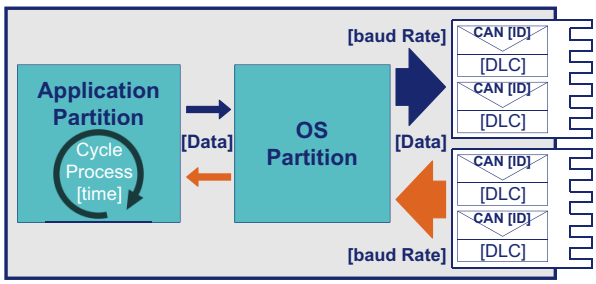

Fig. 5 Device configuration

Table 1 Requirement list

No.	Requirement
1	The device shall host 1 separate OS partition
1.1	The OS partition shall have 2 CAN sampling ports
1.1.1	The CAN ports shall have a nonextended specifiable CAN ID
1.1.2	The CAN ports shall send and receive messages
1.1.3	The CAN ports shall support specified baud rates [125;250;500;1000] kBaud
1.1.4	The output message length shall be configurable
1.1.5	All viable message lengths shall be receivable
2	The device shall host 1 application partition
2.1	The partition shall host 1 periodicprocess
2.1.1	The process cycle time shall be freely configurable
3	The partitions shall communicate via 1 intra-partition sampling port

especially considering the need for cross-testing of the parameters, as to ensure that no fault is caused by combinations which were not taken into account. Therefore, each relevant parameter is given an individual range of values to be tested, which is accounted for in all variations. The "Cycle Process Time" is divided in values equal to 2 to the power of n in milliseconds, with n being a value between 0 and 10, as to cover a wide range of possible cycle times. Since no precise information on possible cycle times is available, these values were chosen under the assumption that most application requirements are fulfilled. "Data Values" were set depending on the given data type, with the focus set on examining the resulting edge cases, which are derived from the minima and maxima. Data types included are:

- (un-)signed int8
- (un-)signed int16
- (un-)signed int32
- single

This list can be extended if needed, but was deemed as sufficient for the examined PoC.

The "CAN ID" is non-extended and therefore varies between the natural values of 0 and 2047. These are the edge cases that will be tested for the in- and outgoing messages. Therefore, each test configuration needs at least two out- and two ingoing CAN Messages. The "CAN Data Length Code (DLC)" ranges between 1 and 8. But since the range is determined indirectly through the bitwise setting of the respective messages, not all DLC values must be checked. Instead, the focus will be set on ensuring that each data field is written and read correctly. The "baud rate" will be tested for every case, since the usage domain is fairly small due to the defined requirements. Additionally, 125 kbit/s and 1000 kbit/s represent the minimum and maximum baud rate for CAN communication, so that both low and high speed buses are analyzed.

5.2 Test setup

The tests are set up as depicted in Fig. 6. On the Control-PC, the executables get generated from the source requirements and are subsequently loaded onto the UDE via the USB connection before the configuration is flashed from the UDE onto the FCS-Device via the JTAG-connection. All of these steps are monitored and controlled through the MATLAB instance running on the Control-PC, which controls the UDE via a COM interface.

As shown in Fig. 6, the two CAN-Ports from the FCS-Device are connected to each other and are used to close a CAN-Loop for testing both in- and output of the device with as little effort as possible. The CAN-Loop is forked, with the UDE as a read-only participant in the CAN-bus.

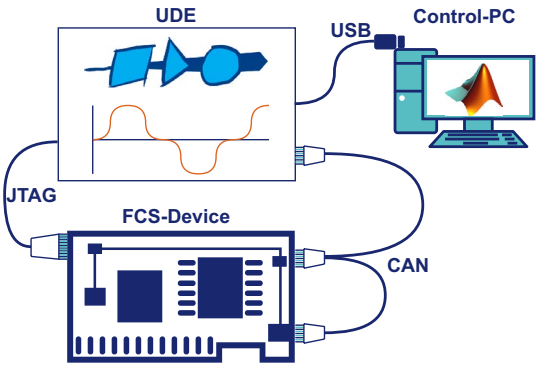

Fig. 6 Test setup

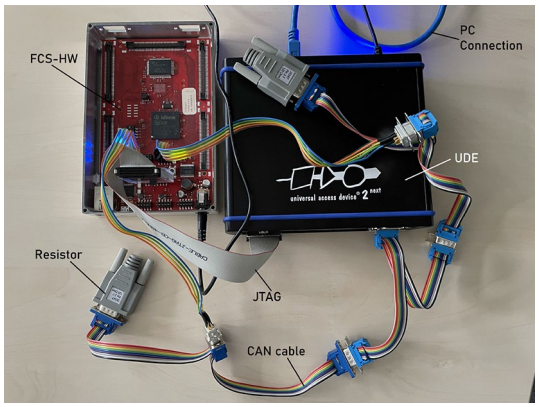

Fig. 7 Test setup with real hardware

Thereby, the UDE can record all messages sent on the bus and store them on an internal storage before transferring them to the Control-PC via the USB connection, where the validity of the test run is examined. A live validation is not necessary, as the goal is to test multiple configurations in a row without any manual interference. Should one or more configurations exhibit faulty behavior, it can be manually analyzed in a following step not covered in EasyConfig, as this task would require more extensive user input. Alternatively, since AvioNET is planned as a holistic tool network approach, eventually test analysis methods will be made available as well. The engineer would then have to use the respective interface within AvioNET to get provided with assistance to improve the debugging process. Nevertheless, the workload for finding faulty behavior or confirming the validity of the configurations is completely removed and only an eventual fix must be designed manually.

The proper hardware setup is shown in Fig. 7. The Flight Control Hardware (FCS-HW) is shown on the right, with the UDE being on the left. The CAN cables are fitted with resistors as required by the CAN protocol, and the JTAG

and PC connection from the UDE to the FCS-HW and PC respectively are shown as well.

6 Conclusion and outlook

The objective of this paper was to develop a semi-automatic toolchain for implementing a configuration derived from requirements on a Flight Control Hardware. This was successfully achieved with EasyConfig as a part of the newly developed holistic approach AvioNET at the FST, which provides a seamless model-based approach for avionics development. By deriving the requirements written in natural language automatically and transforming this data into configurations for virtual and real hardware development, the overall workload is reduced significantly, and continuous validation was enabled. Furthermore, test cases for the generated configurations were derived parallelly and applied for verification purposes. EasyConfig utilizes popular and commonly available tools in the aviation industry, which makes it easy to implement for manufacturers. Additionally, the virtual and hardware configuration artifacts are convertible throughout the complete development, which also makes early validation without the specific devices possible. Overall, it was possible to remove various tedious transformation steps and to decrease the manual workload significantly. Nevertheless, the user still has full control over the process and is able to monitor all important artifacts and intervene in the process. This shifts the focus of the stakeholders to a knowledge-based engineering approach, since time spent on trivial tasks is reduced.

The automatically generated test cases prove that the configurations are valid and fulfill the requirements set by the ARINC 653 Part 4 standard, which means that the toolchain is applicable for the analyzed IMA-like FCS systems.

While the application used in this paper was merely a PoC, future development should focus on implementing more complex applications, as to examine the scalability of the approach. Additionally, the configuration of the I/Os must be expanded further, so that the full spectrum of avionic communications is covered. Furthermore, the possibility of adapting the toolchain to provide assistance for configuring multi-core systems should be explored. Since this technology will gain importance in the future and the access to a multi-core board is already established, this step is a natural follow-up. Lastly, an intuitive user interface should be added, as to further improve the usability of the toolchain.

Moreover, it should be noted that the presented approach is not just limited to Flight Control Hardware, but applicable to other fields as well. Due to the focus on standardization and formalization of the underlying models, interfaces are easily introduced, which allows the methodology to be easily be integrated into any IMA workflow and the authors plan to adapt the toolchain as such. Future works will expand the toolchain's interfaces to allow for additional (virtual) device configurations.

Acknowledgements This work was funded by the German Federal Ministry of Economic Affairs and Energy (BMWi) within the PLATEAU project. Their support and the cooperation of the partners is greatly appreciated.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Barnat, J., Beran, J., Brim, L., Kratochvíla, T., Ročkai, P.: Tool Chain to Support Automated Formal Verification of Avionics Simulink Designs. In: Formal Methods for Industrial Critical Systems, 17th International Workshop, Paris, France (2012)
2. Barner, S., Diewald, A., Migge, J., Syed, A., Fohler, G., Faugère, M., Pérez, D.G.: DREAMS Toolchain: Model-Driven Engineering of Mixed-Criticality Systems. In: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), Austin, TX, USA (2017)
3. DO-178C - Software Considerations in Airborne Systems and Equipment Certification. Standard, Radio Technical Commission for Aeronautics (RTCA), Washington, D.C., USA (2011)
4. Sun, Y., Brain, M., Kroening, D., Hawthorn, A., Wilson, T., Schanda, F., Jiménez, F.J.G., Daniel, S., Bryan, C., Broster, I.: Functional Requirements-Based Automated Testing for Avionics. In: 2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS), Fukuoka, Japan (2017)
5. Darwesh, D.N., Annighöfer, B., Reichel, R.: Semi-automated Deployment of a High-lift System on IMA Using the Selective Middleware. In: 2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS), San Diego, CA, USA (2019)
6. Halle, M., Thielecke, F.: Tool Chain for Avionics Design, Development, Integration and Test. In: 1st Workshop on Avionics Systems and Software Engineering AvioSE'19, Stuttgart, Germany (2019)
7. Aniculaesei, A., Vorwald, A., Rausch, A.: Automated Generation of Requirements-Based Test Cases for an Automotive Function Using the SCADE Toolchain. In: The Eleventh International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2019), Venice, Italy (2019)
8. Halle, M., Thielecke, F.: Avionics Engineering Tool Network (AvioNET):Experiences With Highly Automised and Digital Processes for Avionics Platform Development. In: 2021 AIAA/

IEEE 40th Digital Avionics Systems Conference (DASC) Proceedings, San Antonio, TX, USA (2021)
9. 653P1-5 Avionics Application Software Standard Interface, Part 1, Required Services. Standard, SAE ITC, ARINC Industry Activities, Bowie, MD, USA (2019)
10. Hoeber, H., Thielecke, F.: Eine Simulationsbasierte Methode zur Fruehzeitigen Validierung Von Avionik-Plattformen. In: Deutscher Luft- und Raumfahrtkongress DLRK, Friedrichshafen, Germany (2018)
11. Kim, J.-E., Yoon, M.-K., Bradford, R., Sha, L.: Integrated Modular Avionics (IMA) Partition Scheduling with Conflict-Free I/O for Multicore Avionics Systems. In: 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden (2014)
12. 653P4 Avionics Application Software Standard Interface, Part 4, Subset Services. Standard, SAE ITC, ARINC Industry Activities, Bowie, MD, USA (2012)