Testing of Electronic Control Unit Communication Interface

Petr Šimek, Kamil Kostruk

***Abstract*—**This paper deals with the problem of testing the Electronic Control Unit (ECU) for the specified function validation. Modern ECUs have many functions which need to be tested. This process requires tracking between the test and the specification. The technique discussed in this paper explores the system for automating this process. The paper focuses on the introduction to the problem in general, then it describes the proposed test system concept and its principle. It looks at how the process of the ECU interface specification file for automated interface testing and test tracking works. In the end, the future possible development of the project is discussed.

***Keywords*—**Electronic control unit testing, embedded system, test generate, test automation, process automation, CAN bus, Ethernet.

I. Introduction

THIS paper is about the use of the structured file with the ECU interface description. This file is used for automating interface tests, or making it control the unit simulation for full virtual testing. This virtual tool collection can be used for the test case development before the real ECU exists. Automatically generated test skeletons help to cover the ECU interfaces by tests. The use of this technique is based on standard software unit test tools used for the classic programming language. With this tool, it is possible to automate and control their behavior. The results are used to generate test reports.

A. Motivation

The reason for the verification of this system was the need to create a large number of tests and controls in the development of ECUs. With a large number of tests, it is very difficult to keep the actual state of all test specifications and their implementations in sync. From this situation came the idea of using a file to describe the interface of the control unit, to help with the creation of test specifications and their implementation. To implement this idea, it was necessary to create a unified interface for access to communication with the control unit. Another important part is the supporting tools for generating source codes and processing information from database systems. This entire system is intended to ensure more effective testing and a substantial reduction in repetitive activities, which are a frequent source of human error. The main expectation from the use of this system is to reduce the cost of test development and increase the quality of both the tests and the tested product. Communication interfaces for signal transmission over the Controller Area Network (CAN) bus and Scalable service-Oriented MiddlewarE over IP (SOME/IP) interface over the Ethernet bus were used for the system verification.

Petr Šimek and Kamil Kostruk are with the Electronics and Information Technology Department, University of West Bohemia, Pilsen, Czechia (e-mail: pesimek@fel.zcu.cz, kosturik@fel.zcu.cz).

The design of the test environment was based on the principles below. An example of an electronic control unit that can be tested by this system can be seen in [3]. The principles for testing real controllers described in Section VI *F* were obtained from [4]. Automotive software testing requirements were obtained from [5]. The principles for creating a runtime environment and simulation in Section VI *B* were obtained from [6].

II. CAN Bus

CAN bus is a shared communication with a non-deterministic approach to the communication bus. The main advantage is the collision-free access control, which significantly reduces communication latency. The most widespread type of communication is the transmission of signals used for real-time control. The transmitted signals occupy any position and size in the data area of the transmission frame. This information is also contained in a file describing the interface of the control unit. By processing this information, the tests are able to transmit and receive these signals. The data area of a communication frame consists of three objects nested inside each other. Theoretical knowledge was acquired from [1].

A. Frame: It is object describing the parameters of a CAN bus communication frame, such as an identifier or the length of a data area.
B. Packet Data Unit: It is object describing the composition of the data area of the communication frame, defines the size of the data segment and its location.
C. Signal: It is an element transmitting the required information. An element is described using a set of information such as size, position, offset, ratio, minimum, and maximum value.

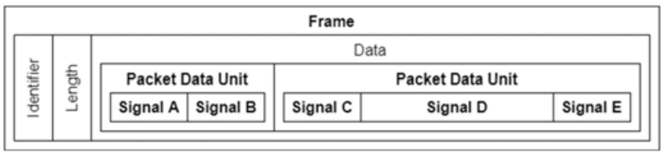

Fig. 1 CAN communication frame

III. Ethernet

It is a common communication line known from the field of computer technology, which began to assert itself in other branches of industry. The main advantage is a standardized interface to which various types of circuits can be connected to access the communication medium. SOME/IP communication using the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) transport layer was selected to verify this system. Example of practical use [2].

A. Data Packet

Data packet is a set of data encapsulated in several communication layers, similar to a signal interface. The data contain important information for the identification of the transmitted service, methods and parameter data.

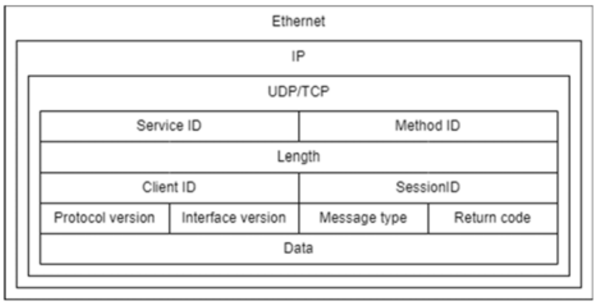

Fig. 2 Ethernet communication frame

B. Communication Diagram

Service-oriented communication uses various types of communication protocols, see Fig. 3. The basic communication is client/server, sender/receiver and event-driven multicasting.

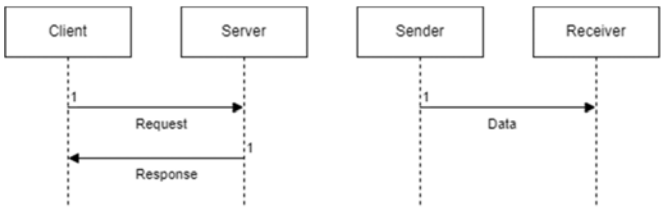

Fig. 3 Communication diagrams

IV. Test Set Description

A. ECU Interface Description

For interface description the AUTomotive Open System ARchitecture (Autosar) Extensible Markup Language (XML) format is used. For processing this file there are many different tools and a software library.

B. Signals

It is the basic quantity processed by the ECU. The description file contains all important information, such as the communication medium, coding and the position in the data section.

C. Functions

Complex communication uses the function interface. This is interpreted as a standard programming function call. There are differences in function processing. This function is the call from the client but the execution is processed by the server. The function can be collected into object call services. The basic information is the communication interface, input and output parameters, coding and the position in the data section.

V. Data Processing

The XML file structure can be processed by using the description file XML Schema Definition (XSD). From the XSD file the conversion object is created, which is used for transforming the XML file into the memory object. The data describe the ECU interface. Based on this interface description, the test skeleton and simulation will be generated.

A. Control Unit Simulation

The file is created as a description of tested electronics control units networks. The file describes input and output signals, and the provided and consumed function. When generating the source code, a decision is made on how to create skeletons, simulation, all events and functions. Simulation events are called if the unit receives the message with signal or function data. When the defined function is called, the communication message is sent to the corresponding node. For the signal, the variable buffer is generated with the read and write function. Signals' timings are read form the description file, too.

B. Tests

During the generation of the test skeleton, it is necessary to swap all the communication direction for functions and signals. The main difference from generating a simulation is the creation of test skeletons for the simulation interface. For all signals and functions a set of random and limit value tests is generated. For example, signal "A" is specified as an 8bit unsigned value, with the range from 10 to 100. The first test is possible to do as a random value in a valid range. The second test is a minimum value. The third test is a maximum value and the last test is an invalid value. These tests verify the behavior of the ECU. For each function and signal, a separate file with a basic test skeleton is made. This structure is suitable for better clarity. Every test skeleton is marked as inconclusive. This marking ensures that one does not forget to test the complete ECU interface.

C. Architecture

For simulation and the test correcting function, it is necessary to create a runtime module and communication interface. The modules allow access to a virtual or real communication interface. The software component for the runtime environment and communication interface is designed symmetrically. It means that it is possible to send and receive data from any interface. This symmetry allows us to use the same module for tests and simulation.

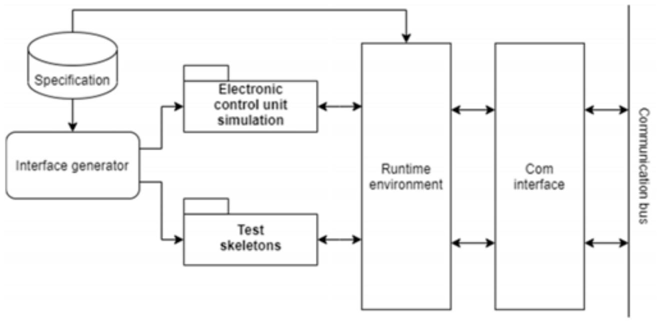

Fig. 4 Architecture of the tested system

D.Runtime Environment

The runtime environment is used to process the signal and function arguments. The main purpose of this module is to make signal transformation and an assembly data packet. Data processing is driven by a specification file. In the initialization phase, the specification file is loaded and objects representing signals and functions are created. The read object is used for processing and constructing the communication data packet. All information requested for constructing and processing the data packet, signal transformation and function or signal identification, is also loaded from the specification file. A fixed user interface is defined to control the runtime. This function unifies access to the internal object by the text name. For example, the function for setting the signal value contains the argument represented by the signal text name and signal value like the float variable. This function allows us to set any signal from the specification file, and enables to unify generated interface for testing and simulation. A similar principle is applied to functions defined in the specification file.

E. Communication Interface

The communication interface is used for the communication direction set and communication buses access. This module handles the setting for the system, virtual or physical communication buses. All configurations are read from the specification file. The input for this module is the data packet prepared in runtime environment. These data are sent to the corresponding place by the runtime setting. Selection between the virtual and physical communication interface is made by the information read from the environment configuration file which is used for the configuration of the whole test environment.

F. System Environment

The system environment is the collection of tools. The purpose of these tools is running and controlling the whole test system. One part of this tools set is the source code generator. This generator creates a set of test skeletons for all the ECU input interface, and receives events for the output interface. After implementing tests and simulation, it is possible to verify that the tests generate correct communication sequences. Both modules enable to work separately. It is possible to simulate the ECU specified in the description file or run all test cases separately with a real ECU. This behavior is controlled by the main configuration file. The main purpose of this system is to create the full virtual environment, for testing before the real ECU exists.

G.Runtime Tools

Simulation and tests are the standard program function. It is possible to compile them as the standard software library. These functions can be initialized and called from a unit test framework. The unit test framework is the standard test environment which enables to automate, control and generate the test report. The main goal of this setup is the high level of automation. Another important thing is the possibility of using the source control versioning and project management tools.

H.Automation

To accelerate the development, it is necessary to make all steps automatically. This reduces a large count of human-caused errors. This system is a collection of a source code, which is necessary to be versioned by any tools. The version control system must by connected to the project management tool. The project management imposes requirements on functionality of ECU. All requirements must by linked to the test source code stored in the versioned control system. Every test change triggers a complete system compilation and run of all the tests. After the test run, the test report is generated and stored in the database. This automation helps us to discover errors in ECU software implementation very fast.

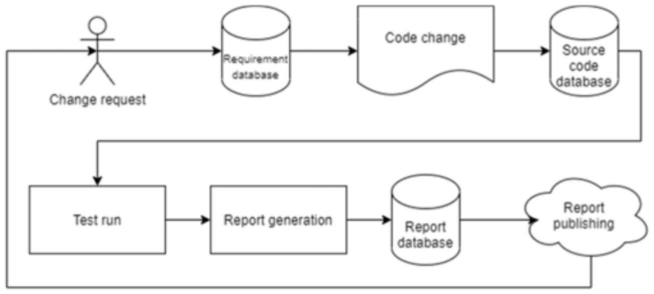

Fig. 5 Automation steps

VI. Used Technologies

To validate the system, the C# programming language with the ".NET6" runtime environment was used. The MS test tool was used to run the tests. The access to the communication buses is provided by a communication interface from Vector Informatics. The Windows operating system takes care of the network access. Despite the higher demands of this environment on computing power, communication latency of about +- 1 ms was achieved. This performance was achieved experimentally for a single message. Real control units do not use such short repetition periods in practice.

VII. Measurement

For the signal interface on the CAN bus the test measurement was made, to verify the performance of the system, see Figs. 6 and 7. With a transmission period of 1 ms for 10 s, a maximum of 5 messages skips its period. These values should be below the resolution of the internal control mechanisms of the controller. The next step should be to verify the performance for more different messages, carrying a larger amount of signal. The goal of this measurement should be to determine whether the system is able to process such a large amount of information with sufficiently low latency.

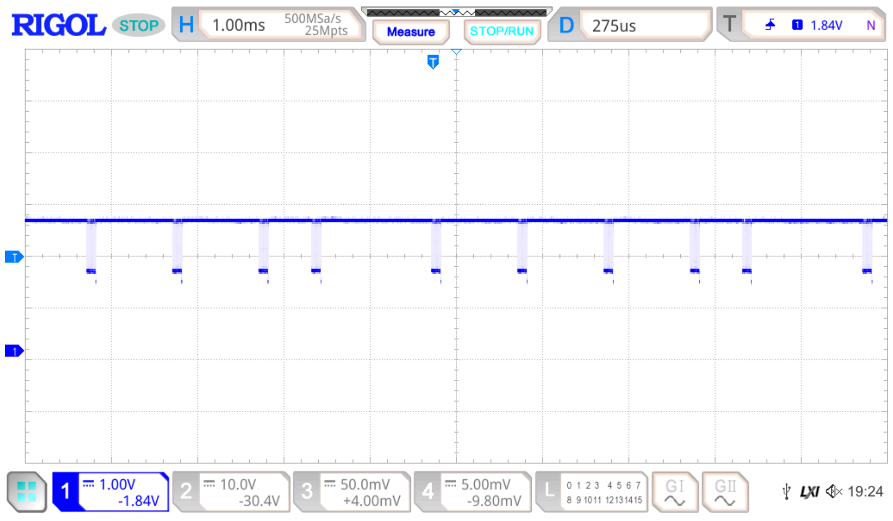

Fig. 6 Log period message

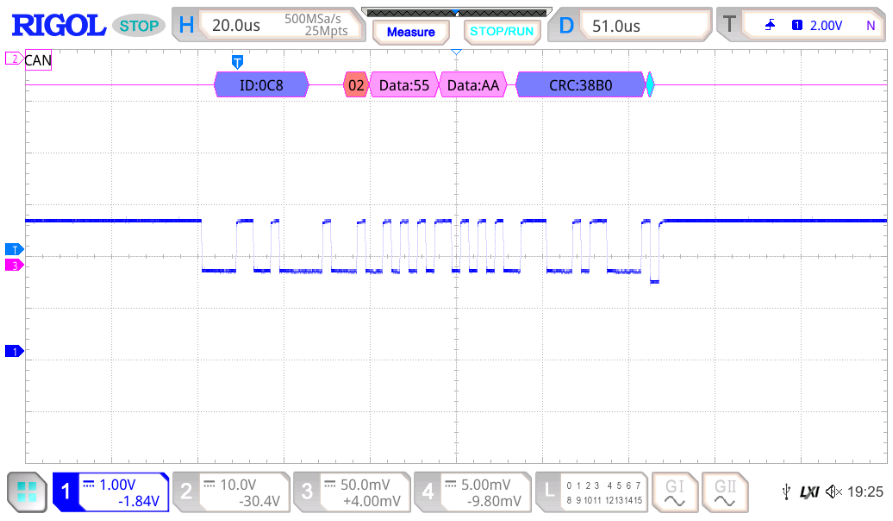

Fig. 7 Test data

VIII. Future Developments

A large number of tests examine the behavior and responses to system error states, and these are recorded into the memory. For this reason, Unified Diagnostic Services (UDS) appears to be an additional component suitable for expansion to provide access to diagnostic information. When developing the control units, it is often required to gain access to inside information that is not part of the standard interface. These can be internal state or calibration values. For this purpose, the Universal Measurement and Calibration Protocol (XCP) is used, which can make part or all of the internal memory of the unit available to the developer. This component could follow the development of the diagnostic module.

IX. Conclusion

The developed system enables to use parallel work on the ECU and tests. It ensures better control over the complete testing process and decreases the time for error discovery. The test skeleton generator significantly speeds up test development, and improves functionality. In the test development it is not necessary to focus on adding a new scenario, and it is possible to be fully focused on the test scenario design from the test requirement. It is possible to use simulation instead of a real ECU. Before the real unit exists, it is possible to use it for the complete system behavior simulation. After releasing the ECU, it is possible to use the designed test for continual error detection from the requested functionality.

Acknowledgment

This article was created with the support of an internal project SGS-2021-005: Research and development electronic

and communication systems in scientific and engineering applications.

References

[1] Wolfhard Lawrenz. (2013). CAN System Engineering, ISBN: 978-1-4471-5613-0

[2] T. Steinbach, K. Müller, F. Korf and R. Röllig, "Demo: Real-time Ethernet in-car backbones: First insights into an automotive prototype," *2014 IEEE Vehicular Networking Conference (VNC)*, Paderborn, Germany, 2014, pp. 133-134, doi: 10.1109/VNC.2014.7013331.

[3] H. Moon, G. Kim, Y. Kim, S. Shin, K. Kim and S. Im, "Automation Test Method for Automotive Embedded Software Based on AUTOSAR," *2009 Fourth International Conference on Software Engineering Advances*, Porto, Portugal, 2009, pp. 158-162, doi: 10.1109/ICSEA.2009.32.

[4] A. Varshney, S. Joshi and K. Namrata, "Automated Testing of Faults of an Automotive System," 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Bombay, India, 2019, pp. 1-5, doi: 10.1109/I2CT45611.2019.9033751.

[5] K. N. Hodel, J. Reinaldo Da Silva, L. R. Yoshioka, J. F. Justo and M. M. D. Santos, "FAT-AES: Systematic Methodology of Functional Testing for Automotive Embedded Software," in IEEE Access, vol. 10, pp. 74259-74279, 2022, doi: 10.1109/ACCESS.2021.3128431.

[6] D. Brkić, A. Kostić, M. Herceg and M. Popović, "Test environment code and test-case generators," *2022 IEEE Zooming Innovation in Consumer Technologies Conference (ZINC)*, Novi Sad, Serbia, 2022, pp. 159-164, doi: 10.1109/ZINC55034.2022.9840658.