15th IMEKO TC10 Workshop on Technical Diagnostics
Technical Diagnostics in Cyber-Physical Era
Budapest, Hungary, June 6-7, 2017

HIL test based non-intrusive diagnostics of cyber-physical systems

Balázs Scherer

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
scherer@mit.bme.hu

***Abstract* – Cyber-physical systems have extensive contact with the physical world. Usually during the development of these systems, the testing phase cannot be done efficiently or safely in the complete real environment, and therefore HIL (Hardware In the Loop) simulators are used. During HIL testing, diagnostic protocols are used very often to gather detailed information about the DUT's (Device Under Test) internal state. Diagnostic protocols are very useful during testing, but they cause a significant load to the DUT. This paper introduces a novel approach to replace traditional diagnostic protocols with a non-intrusive solution. The presented method is based on the debug capabilities of modern ARM Cortex M core microcontroller, and uses a CMSIS-DAP (Cortex Microcontroller Software Interface Standard Debug - Access Port) based interface.**

I. INTRODUCTION

Most of the cyber-physical systems use one or more microcontrollers to perform the computation and control tasks. Developing software for these microcontrollers requires much effort and extensive testing. One of the typical tests of the cyber-physical systems is the HIL (Hardware in the Loop) test, where the behavior of the integrated software and hardware of the DUT (Device Under Test) can be investigated in a simulated and stable environment. These test are originated and widespread in the automotive and transportation industry, where the DUTs are mostly safety critical, and performing the first tests in the real environment would be too costly and dangerous. Although HIL tests have been designed primarily for safety-critical devices, but using them for general purpose cyber-physical systems have many advantages and getting more and more widespread [1]. HIL tests can reduce the time spend to real environment testing, because most of the failures can be detected earlier in the simulated environment. This can significantly reduce the overall testing cost, because real environment setups like prepared track with staff for autonomous robots, or real plant setups for industrial controllers are very costly. HIL tests also have the advantage of repeatability, controllability and stability, which is usually not given in the real environment.

A typical HIL test setup (shown on *Fig. 1.*) consists of an environment simulator, which is usually executed in a real-time hardware, and a test control station, which is usually a general purpose PC. The environment simulator runs the model of the DUT's physical surroundings, and interacts with the DUT's hardware I/O-s. While the test control station enables the tester to configure and execute the HIL test and evaluate its results.

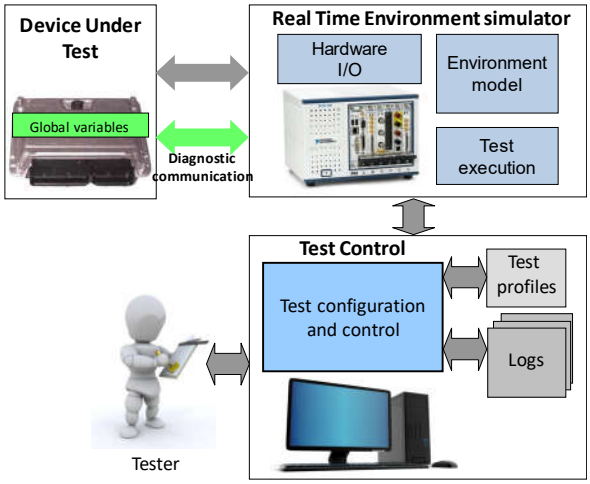

Fig. 1. Typical HIL test environment

Many HIL test setup gather additional information from the DUT beside measuring and stimulating its direct interface to the physical environment. This additional information is useful to monitor the internal behavior of the DUT's software. For example, monitoring the internal states, main input and output parameters of software modules are very useful during the testing to identify the source of problems. It is also very useful for example to calculate and modify calibration parameters for analog measurements and controls during testing, which also needs an interface to the core software parameters. The parameters above can be monitored or modified by using

diagnostic protocols.

Among others, the automotive industry has many standardized protocols with extensive tool support.

The most widespread automotive diagnostic protocols are the CCP (CAN Calibration Protocol), its extended version the XCP (Universal Measurement and Calibration Protocol Family), KWP2000 (Keyword Protocol 2000) and UDS (Unified Diagnostic Services).

These protocols have very similar main features for performing the functionality described above:

- Reading and writing RAM memory locations by address
- Programming the DUT's Flash memory

Some protocols complement these features with user authentication, periodic stimulus and measurement possibilities, and special features, like calibration page writing or diagnostic trouble code reading and clearing.

The protocols above are not restricted to the automotive industry. They can be used in any industrial segment. XCP [2] for example is well suitable for testing cyber physical system. It supports many types of communication interfaces: Serial, USB, Ethernet, CAN etc. XCP is an open, free standard and there are downloadable sample implementations of it.

Beside the useful functionality and features provided by XCP or another diagnostic protocols they also have disadvantages, which limit the scope of their use.

II. LIMITATIONS OF TRADITIONAL DIAGNOSTIC PROTOCOLS

XCP and similar diagnostic protocols are very widespread and suitable for testing cyber-physical systems, but they have many drawbacks:

- Diagnostic protocols cause significant load to the processor: for example, handling TCP/IP or USB communication requires significant processor time, which can highly influence the behavior of the investigated system.
- Diagnostic protocols require program and data memory. The simplest diagnostic server configurations start at the 10Kbyte range of program memory, but more complex ones can easily reach the 100Kbyte domain, which make this technique unusable for systems with very limited Flash and RAM.
- Diagnostic protocols often use the same communication channel as the normal function, which gives a hard limitation to the diagnostic data rate.
- Load (processor time, communication bandwidth) caused by the diagnostic can modify the behavior of the DUT.
- Diagnostic communication is hard or impossible to use for monitoring low power modes of cyber-physical systems.

The drawbacks above show that diagnostic protocols are not usable for much type of cyber-physical systems, especially for highly data rated, or low resourced, power critical ones. Therefore, a solution is needed to provide the functionality of diagnostic protocols (which is essential part of modern testing) without their limitations. The solution is to use non-intrusive diagnostic by using the features included in modern microcontrollers.

III. NON-INTRUSIVE DIAGNOSTIC

Modern 32 bit microcontrollers include very capable enhanced debug features, for example the ARM Cortex core devices [3] include the ARM CoreSight on-chip trace and debug solution, where many others implement the IEEE-ISTO 5001-2003 (Nexus) standard [4].

These debug features among others provide interface to the internal memory of the microcontrollers without stopping, or modifying the behavior of the core. Reading from and writing to the DUT's memory and reprograming it are the most important features of diagnostic protocols. Therefore, modern debug ports can be used to replace many of the functions of the diagnostic protocols, and provide a non-intrusive solution to the limitations described in Chapter II. To achieve this goal HIL tests needs to be extended to use modern debug ports for DUT diagnostic.

IV. INTERFACING DEBUG PORT FROM HIL TEST ENVIRONMENT

Traditionally at the embedded software development process the IDEs (Integrated Development Environment) provided by the microcontroller manufacturers include the debug features to help software development and debugging. However, these IDEs can be used at the very early phase of the testing process, but they are inappropriate to be used in the HIL test phase due to the following reasons:

- Software development IDEs usually cost significant amount of license fee.
- IDEs do not have real-time capabilities
- Testing is often done by a separate group or company, and the source code for them might be unavailable.
- It is very hard or even impossible to synchronize and share data between the HIL environment and the Software Development IDE.

Automotive tool manufacturer companies are aware to the above problems and try to provide solutions. Vector's VX1000 family is one of the very few tools capable to solve this topic [5]. The VX1000 family functions as an Ethernet based XCP server, and provides JTAG port or

special, target dependent port interface for typical automotive target processors.

The VX1000 series has excellent functions, but it is a very expensive tool. Depending on the configuration, its price range in the $2000 - $10000, without software tools. Other drawback of the VX1000 series, that it is designed for special line of automotive processors. Therefore, it can't be used with general-purpose microcontrollers typically applied in most of the cyber-physical systems.

V. SUGGESTED SOLLUTION

The proposition of this paper is to follow these trends, but in an environment that is not restricted to the automotive industry. Therefore, a complete novel toolset is under design and development by us, to support non-intrusive diagnostic of cyber-physical systems. This toolset is primary designed for ARM Cortex M core microcontrollers, which are the most well known platforms of cyber-physical systems [6], but it will be extensible to other microcontroller families too.

During the design of the toolset four main possible HIL – Debug port interface implementation options are identified (shown on *Fig. 2.*). To understand these possibilities an overview of microcontroller debugging is needed (GNU tools based development environment is used as example): In a normal debugging there are four components: debugger, debug server, debugger hardware and target. The debugger (for exampleGDB) is connected to the graphical IDE (Eclipse CDT, DDD etc.), it communicates through a special protocol called RSP to the Debug server, and debug server executes the debug command through the debugger hardware.

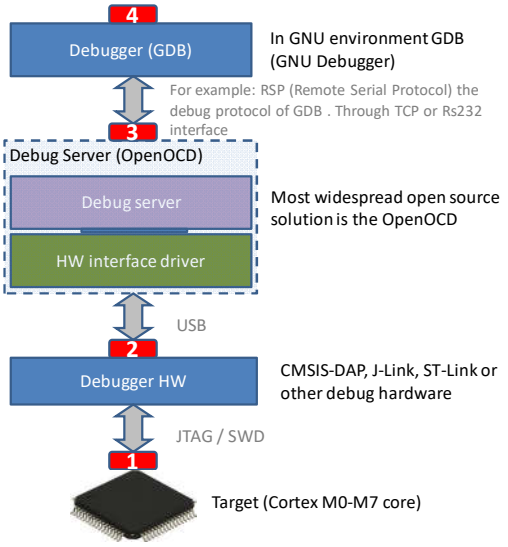

Fig. 2. Possible interfacing points of the debug port communication

The identified four integration ways based on *Fig. 2.* are the following:

1. It is possible to directly perform the SWD or JTAG communication using a HIL test integrated hardware. The straightforward solution to this implementation way is to use an FPGA based hardware, which is very costly. For example, the price of an FPGA based National Instruments PXI card starts at the $2000 range.

2. Another promising way is to use an existing debugging hardware and integrate it into a HIL environment. Traditionally the main problem with this way is that each debugger hardware manufacturer uses its own closed communication protocol through the USB connection, like SEGGER's J-Links, ST's ST-LINK/*V2* etc.. Fortunately, ARM gives a solution to this problem. The name of this solution is the CMSIS-DAP (Cortex Microcontroller Software Interface Standard - Debug Access Port).

3. It is also possible to emulate an external debugger, and send RSP commands to a Debugger Server from the HIL environments. The drawback of this solution, that a debug server implementation like Open-OCD is needed. Therefore, this solution is not suitable for real-time HIL systems (making a non real-time program like Open-OCD real-time is nearly impossible).

4. It is also possible to interface to the debugger application like GDB from the HIL system by the same way as the development environments does it. This solution also non fit for real-time systems, and needs huge third party software support.

The advantages and disadvantages of the above options are summarized in *Table. 1.*

	1, Direct JTAG/SWD interface	2, Debugger HW driver IF	3, RSP protocol interface	4, Debugger software interface
Complexity	high	medium	low	low
Speed	high	medium - high	low	low
Real-Time HIL simulation	possible	possible, Real-Time USB support needed	no RT support for Debug server	no RT version of GDB
Universality	medium: µc arcitecture depended	medium: µc arcitecture depended	medium: any target with debug s. support	medium: any target with GDBsupport
HW Cost	medium: FPGA based hardware	low: commercial debugger	low: commercial debugger	low: commercial debugger

Table. 1. Comparing interfacing options

Table 1 shows that there is no a clearly best way, but option 1 and 2 is the most promising. Therefore, the toolset is intended to support both option 1 and 2.

This paper describes the implementation of option 2 the CMSIS-DAP debugger hardware based solution in detail. This option was chosen as first to implement because it is relatively easy, and cheap, but it can provide a medium-high data rate and has real-time capabilities. Therefore, it could be a very good solution to testing general purpose cyber-physical systems. This CMSIS-DAP debugger hardware based diagnostic in HIL test is also a new idea

first described in this paper.

VI. HIL TEST INTEGRATION OF CMSIS-DAP INTERFACE

To understand the CMSIS-DAP based diagnostic a short overview of the internal debug architecture of an ARM Cortex core microcontroller (*Fig. 3.*) is needed.

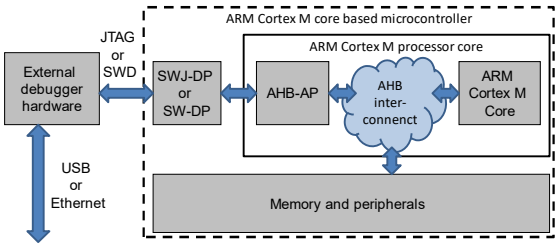

Fig. 3. ARM Cortex M debug connection

External debug hardware can be connected to an ARM Cortex the microcontroller either using 5-wires JTAG or 2-wires SWD (Serial Wire Debug) Debug Port (DP) [7]. The enabled functionality is independent from the Debug Port. Through the Debug Port, the debugger is able to access the AHB-AP (Advanced High-performance Bus Access Port). The AHB-AP enables the debugger to transfer data through the AHB bus interconnect of the microcontroller, which means that the debugger is able to read or modify any memory location, including peripheral registers in the target systems, without causing load to the microcontroller (that was the main condition for non-intrusive diagnostic).

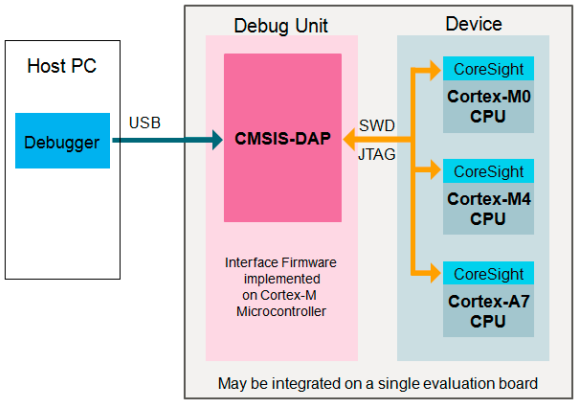

Fig. 4. The CMSIS-DAP interface [8]

Traditionally the external debug hardware is manufacturer depended as described in Chapter V. Fortunately, ARM has introduced the CMSIS (Cortex Microcontroller Software Interface Standard) with the Cortex core [8]. The goal of this standard is to give a common support to all ARM Cortex based microcontroller regardless its manufacturer. As a part of this standard, CMSIS-DAP is intended to give an open specification for the debugger hardware with sample implementations *(Fig.4.)*. Price of such commercial off-the-shelf hardware is about $20.

CMSIS-DAP also specifies an open USB-HID (USB Human Interface Device) based communication protocol to interact with the debugger hardware. The main benefit of using the USB-HID class is that there is no need for a custom driver. The operating system will automatically identify the device like it identifies a keyboard or a mouse. The CMSIS standard also provides a *RDDI-DAP* Access *DLL*, with a sample implementation to enable Windows based debugger hosts an easy interfacing.

A. Architecture of the CMSIS-DAP HIL test interface implementation

The VX1000 tool of Vector Informatik (see Chapter IV) simulates an XCP server to cover the debug communication. This approach is good, because XCP is a widespread protocol, and there are many tool supports for it including HIL test integrations.

This paper suggests using this approach, but the XCP server functionality will not be integrated into an embedded device, like the VX1000. The reason of this decision was to make our toolset solution as flexible as possible, and enable the use of commercial off-the-shelf CMSIS-DAP debuggers *(Fig.5.)*.

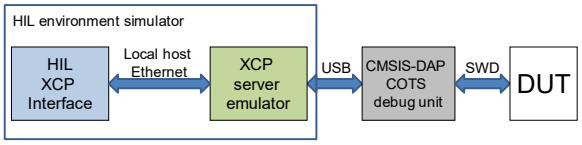

Fig. 5. Architecture of the suggested solution

B. NI-VeriStand

The NI VeriStand [9], is one of the most widespread HIL development environments and therefore it was selected as a evaluation platform for our toolset.

A Typical NI VeriStand system consists of a host computer, where the HIL test can be configured: selecting the model which describes the environment, connecting the input/output channels of the model to hardware I/O channels or communication signals, specifying the user interface and creating automated stimulus profiles.

After the configuration of the HIL test, typically the model and stimulus profile execution, and hardware handling functions are deployed to a RT target (PXI Real-Time controller, Real-Time PC, or Compact RIO), while the user interface remains in the host PC *(Fig. 6.)*.

A special TCP/IP communication maintains the connection between the Host PC (user interface), and the RT Target (VeriStand engine) through the VeriStand Gateway.

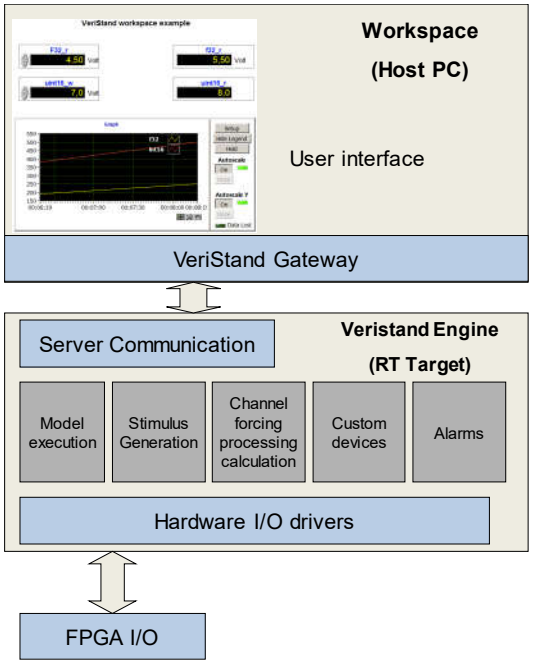

Fig. 6. NI VeriStand architecture

C. Integrating the CMSIS-DAP interface into NI-VeriStand

It is easy to integrate XCP based diagnostic into NI-VeriStand, because there is an add-on the NI ECU Measurement and Calibration Toolkit for this [10]. Using this add-on, only the *XCP Server emulator* from *Fig.5.* needs to be implemented.

This block is a gateway between XCP and the CMSIS-DAP debugger hardware. To integrate it into NI-VeriStand a so called Custom Device driver is needed.

A NI VeriStand Custom Device functions as an interface to a special hardware. A Custom Device can have any number of input and output channels, and its functionality can be executed in every VeriStand engine cycle or can be implemented as a parallel execution task [11]. In this integration the parallel execution task version is need, because the XCP server functionality is too complicated to be implemented in in-cycle mode.

To interface the debug hardware the RDDI-DAP Access DLL calls can be used with the *Call Library Function Node* VI. This is the easiest way if real-time execution is not needed, because the DLL provides high level functionality. If real-time execution is needed, then the NI-VISA blocks based raw USB communication blocks can be used. From the performance point of view there is no difference between the two solutions.

A skeleton of CMSIS-DAP interface integration into NI VeriStand has been implemented. The architecture of this skeleton is shown on figure *Fig. 7.* This implementation currently is a minimal version, which is suitable to test the concept. Therefore only the most basic XCP commands, like direct memory reads and writes are implemented, and the RDDI-DAP Access DLL interfacing method is used for the debug hardware communication in a non-real-time environment (The NI-VISA blocks based raw USB communication is also tried out, but it is more complicated to implement).

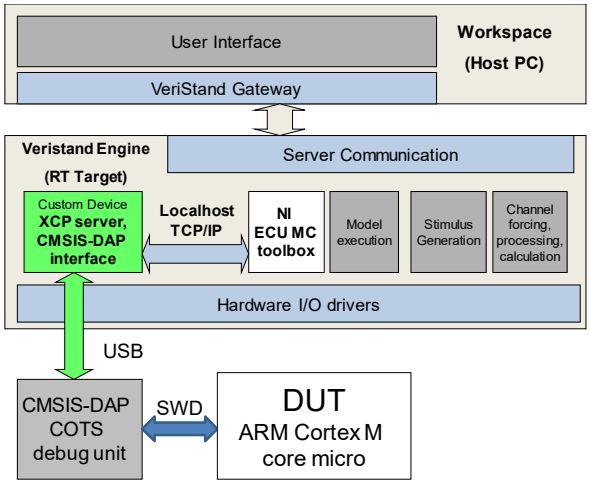

Fig. 7. NI VeriStand architecture

VII. COMPARING THE PERFORMANCE

The performance of this implementation depends on the capabilities of the CMSIS-DAP debug hardware mainly. Current COTS debug hardware solutions and open source software and hardware versions support a relatively low bandwidth. This means a maximum of 1MHz SWD clock frequency, and a Full speed 12 Mbps USB communication.

Determining the number of clock phases required reading, or writing one 32 bits variable of the DUT through the SWD debug port is not easy: the number of clock phases is depend on whether the debugger wants to read or write a continuous RAM area, or just a single 32 bits block. To make the calculation even harder the AHB-AP access port use an aligned data transfer, therefore variable addresses crossing a 32 bits word boundary cannot be handled with one read or write cycle.

To enable simple calculations, 138 clock cycles can be considered as 32 bits variable accesses. This means 138 µs pro 32 bits variable access using a 1MHz SWD clock.

HIL tests usually use a 1ms, or 10ms cycle times depending on the DUTs requirements. This means 7 pieces of 32 bits variable accessing for 1ms cycle time and 72 pieces for 10ms cycle time.

As a comparison 1Mbits/sec CAN bus based XCP diagnostic could provide a lower data rate with direct reads or writes, and about the same data rate if the system

uses the event oriented DAQ messages features of XCP. Native USB or Ethernet based XCP servers could provide a significantly higher data bandwidth, but they cause huge overhead, and cannot be used in low resourced cyber-physical systems.

The data rate of the CMSIS-DAP port based solution can be increased relatively easily. The SWD clock frequency is not limited in 1MHz. The DUT's hardware gives limitation to the maximum SWD clock rate, but typically this limitation is in the N*10MHz range. If the SWD clock rate is increased, probably the Full speed 12 Mbps USB port became a limitation. Therefore, the NXP LPC11U35 low end 32 bits microcontroller used in most of the COTS debug hardwares should be replaced by a more capable microcontroller, for example one from the LPC18xx series. This microcontrollers are more costly than the LPC11U35, but their price is still lower then $10, which would not increase the overal cost of the debug hardware significanlty. This replacement means hardware redesign and software porting, but both design process is relativly easy. Probably COTS debug hardware with these features will appear on the market soon.

VIII. CONCLUSIONS

This paper introduced a novel approach, and the first part of a toolset for non-intrusive diagnostics of cyber-physical systems. Our goal is to give a toolset supporting multiple non-intrusive diagnostic interface options for testing cyber-physical systems. This toolset is under development, and in this paper a novel way, the CMSIS-DAP based tool is presented in detail. The CMSIS-DAP debug port support is integrated into the NI VeriStand environment as a custom device, and this custom device provides an XCP server interface to enable the use of COTS diagnostic software modules. The sample implementation is currently a skeleton enabling only few functionalities of the XCP protocol, but it is suitable to evaluate the concept.

First measurements have shown that the concept is good, and this method enables the diagnostic of low cost, resource frugal cyber-physical systems with a relatively good data bandwidth.

Future work includes the improvement of the toolset and functionality provided by the CMSIS-DAP interface implementation. For example, the XCP server can be extended. Beside the implementation of more data acquisition and stimulation XCP commands, the program downloading to the DUT feature also could be implemented.

The program downloading is one of the hardest features to implement, because it highly depends on the target microcontroller's Flash memory handling. Usually XCP sample codes needs user contributions to perform this operation. The CMSIS-DAP based method could simplify this process. The CMSIS-DAP interface is used by the MBED community, and they complemented the CMSIS-DAP functionality, with a mass storage device based programming feature and a virtual serial port support. This new platform is called the mbed Hardware Development Kit (mbed HDK [12]), and there are open source implementations for that. Using this support the program downloading could be done easily for devices supported by the MBED community. The virtual serial port feature of mbed HDK complementation also can be used to provide more feature, for example DUT – HIL system synchronization, but this is not the scope of this paper.

REFERENCES

[1] A. Biagini, R. Conti, E. Galardi, L. Pugi, E. Quartieri, A. Rindi, S. Rossin "Development of RT models for Model Based Control-Diagnostic and Virtual HazOp Analysis". 12th IMEKO TC10 Workshop on Technical Diagnostics. June 6-7, 2013, Florence, Italy.

[2] ASAM, "ASAM MCD-1 XCP (Universal Measurement and Calibration Protocol) v.1.3.0", 2015

[3] J. Yiu, "The Definitive Guide to the ARM Cortex-M3, Second Edition", Elsevier Inc. ISBN 978-1-85617-963-8, 2010.

[4] IEEE-ISTO 5001™ "The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface Version 2.0" 23 December 2003.

[5] Vector Informatik GmbH, „VX1000 Measurement & Calibration Hardware," [Online]. Available: https://vector.com/vi_vx1000_en.html. [Access date 27.02.2017].

[6] UBM "Electronics Embedded Markets Study 2015" EETimes 2015.

[7] ARM Technical Reference Manual *"ARM® Debug Interface v5 Architecture Specification"* ARM IHI 0031A, ARM Limited 2006.

[8] ARM Technical Documentation: *"CMSIS Version 4.5.0 Cortex Microcontroller Software Interface Standard"* ARM Limited 2017.

[9] National Instruments, "NI VeriStand Fundamentals Course Manual" Part Number 325785A-01, 2011.

[10] National Instruments, "NI ECU Measurement and Calibration Toolkit". [Online] Available: http://sine.ni.com/nips/cds/view/p/lang/hu/nid/210569 [Access date: 27.02.2017]

[11] National Instruments, "NI VeriStand Custom Device Developer's Guide (Beta)", NI White paper.

[12] ARMmbed, "The mbed HDK (Hardware Development Kit)" [Online] Available: https://developer.mbed.org/handbook/mbed-HDK#access-the-mbed-hdk-repository [Access date: 27.02.2017]