ACTA IMEKO
ISSN: 2221-870X
March 2018, Volume 7, Number 1, 27-35

Hardware-in-the-loop test based non-intrusive diagnostics of cyber-physical systems using microcontroller debug ports

Balázs Scherer

Department of Measurement and Information Systems, Budapest University of Technology and Economics, H-1117 Magyar tudósok krt. 2 IE444, Budapest Hungary

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. This paper also introduces a solution to integrate this non-intrusive measurement method to NI LabVIEW based test environments and NI VeriStand based HIL simulations.

Section: RESEARCH PAPER

Keywords: HIL testing; non-intrusive measurement; ARM Cortex; CMSIS-DAP; LabVIEW

Citation: Balázs Scherer, Hardware-in-the-loop test based non-intrusive diagnostics of cyber-physical systems using microcontroller debug ports, Acta IMEKO, vol. 7, no. 1, article 6, March 2018, identifier: IMEKO-ACTA-07 (2018)-01-06

Section Editor: Lorenzo Ciani, University of Florence, Italy

Received September 29, 2017; **In final form** November 29, 2017; **Published** March 2018

Corresponding author: Balázs Scherer, e-mail: scherer@mit.bme.hu

1. 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 behaviour 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. HIL tests are used for the verification and validation of systems like Transmission Control Unit, Electronic Power Steering systems and Break systems. Safety standards like ISO26262 explicitly names HIL as a suitable test environment for software unit tests and integration tests for such devices.

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 spent to real environment testing, because most of the failures can be detected earlier in the simulated environment. This can reduce the overall testing cost significantly, 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 Figure 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.

Many HIL test setups 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

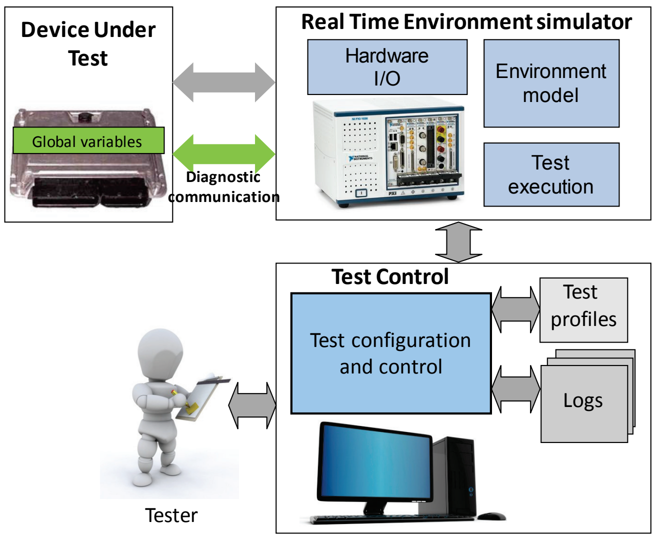

Figure 1. Typical HIL test environment.

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 support tools.

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.

This paper introduces these disadvantages and proposes a novel approach to replace traditional diagnostic servers with a non-intrusive solution based on the debug capabilities of modern ARM Cortex M core microcontrollers. This novel approach removes the overhead and load of the diagnostic communication from the DUT, while providing similar functionality. Therefore, it can be used as a solution for most of the applications including resource limited or very real-time systems.

Beside the description of the proposed novel approach, the paper also introduces a new LabVIEW based free to download basic toolset to enable the application of it.

2. 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 protocol servers cause significant load to the DUT's processor: for example, handling TCP/IP or USB communication requires significant processor time in simple embedded systems, which can highly influence the behaviour of the investigated system.
- Diagnostic protocol servers require program and data memory of the DUT. The simplest diagnostic server configurations start at the 10 kB range of program memory, but ones, that are more complex can easily reach the 100 kB 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 behaviour of the DUT.
- Diagnostic communication is hard or impossible to use for monitoring low power modes of cyber-physical systems.
- Monitoring after reset: many times an errors cause a reset or reboot to the system. For these situations it is very useful to get information about the memory content during this reset procedure.

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

3. NON-INTRUSIVE DIAGNOSTIC IN EMBEDDED SYSTEM TESTING

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 behaviour 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 2. To achieve this goal HIL tests needs to be extended to use modern debug ports for DUT diagnostic.

4. INTERFACING DEBUG PORT FROM HIL TEST ENVIRONMENT

At the embedded software development process, the IDEs (Integrated Development Environment) are provided by the microcontroller manufacturers traditionally. These IDEs 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, where the developers debug their code, but they are inappropriate to be used in later verification activities like HIL tests, due to the following reasons:

- Testing is often done by a separate group or company to provide independence of verification required by many guidelines. The source code for them might be unavailable.
- Software development IDEs usually has a significant amount of license fee.
- IDEs do not have real-time capabilities required by many integrated hardware-software tests
- It is very hard, or even impossible to synchronize and share data between the test environment like HIL environment and a 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 issue [5]. The VX1000 family functions as an Ethernet based XCP diagnostic server, and provides JTAG (Joint Test Action Group) debug port or special, target dependent debug 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 the software tools like Vector CANoe, which is also very costly. Other drawback of the VX1000 series that it is designed for special line of automotive processors. Therefore, it cannot be used with general-purpose microcontrollers typically applied in most of the cyber-physical systems (many automotive microcontroller has special features not required by normal applications, and available only in large quantities like 10 000 pieces or more).

5. SUGGESTED SOLUTION

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.

5.1. Comparing interfacing options

During the design of the toolset four main possible HIL – Debug port interface implementation options are identified (shown on Figure 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 example GDB the GNU Debugger) is connected to the graphical IDE (Eclipse CDT, DDD etc.), it communicates through a special protocol called RSP (Remote Serial Protocol) to the Debug server, and debug server executes the debug command through the debugger hardware.

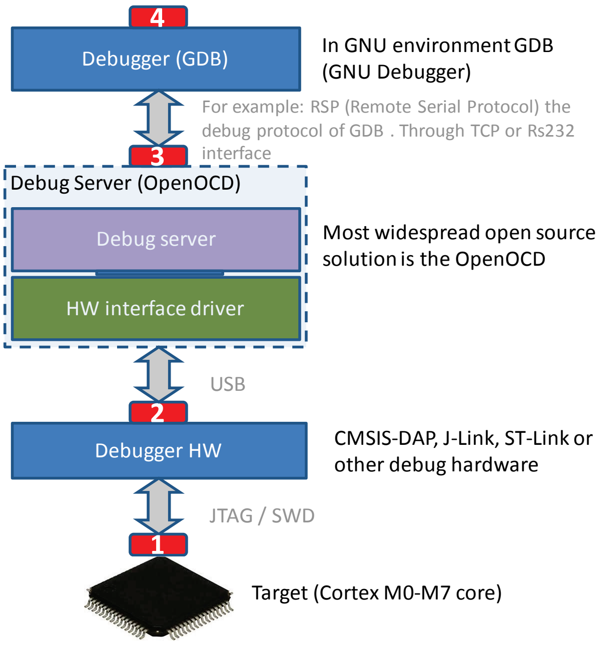
4
Debugger (GDB)
In GNU environment GDB (GNU Debugger)
For example: RSP (Remote Serial Protocol) the debug protocol of GDB . Through TCP or Rs232 interface
3
Debug Server (OpenOCD)
Debug server
HW interface driver
Most widespread open source solution is the OpenOCD
USB
2
Debugger HW
CMSIS-DAP, J-Link, ST-Link or other debug hardware
JTAG / SWD
1
Target (Cortex M0-M7 core)

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

The identified four integration ways based on Figure 2 are the following:

1. It is possible to directly perform the SWD (Serial Wire Debug) or JTAG debug communication using a HIL test integrated hardware. The straightforward solution to this implementation way is to use an FPGA based hardware integrated to the HIL environment. Such FPGA based hardware is very costly. For example, the price of an FPGA based National Instruments PXI card starts at the $ 2000 range. This solution is costly but has the advantages that it provides a direct fast interface with low response time. The FPGA hardware used for this purpose could be used to other interfacing too in the test environment.

2. Another promising way is to use an existing USB based debugging hardware and integrate it into a HIL environment. Traditionally the main problem with this option is that each debugger hardware manufacturer uses its own closed communication protocol through the USB connection, like SEGGER's J-Links or STMicroelectronics's ST-LINK/V2 etc. In case of ARM Cortex core, ARM gives a solution to this problem and make the debug interface tool vendor independent. The name of this solution is the CMSIS-DAP (Cortex Microcontroller Software Interface Standard - Debug Access Port) that will be introduced later.

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). Other drawback of this solution is the more and more interfaces and layers used during the debug process makes the response time higher.

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.

Table 1 shows that there is no a clearly best way. Option 1 provides the best functionality, but it is expensive and requires significantly more development time than the others. Option 3 and 4 are the easiest to implement, but have many drawbacks and these Options are not usable for testing many types of DUTs. Option 2 provides the best compromise in term of cost and functionality.

Based on the above, our 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 for implementation, 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.

5.2. Suggested architecture of the CMSIS-DAP HIL test interface implementation

The VX1000 tool of Vector Informatik (see Chapter 4) 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 as flexible as possible. Integrating the XCP server functionality to an embedded device would mean that anyone, who would like to use the toolset should have to buy or manufacture this special hardware.

We decided to make a pure software implementation of the XCP server (box XCP server emulator on Figure 3). This makes the tool flexible, because only one hardware component a commercial off-the-shelf CMSIS-DAP debugger is needed.

The XCP server emulator will be written in National Instruments LabVIEW [7]. LabVIEW was selected because it is one of the most widespread programing environment used for testing, and NI also has extensive hardware and software support for HIL testing. The OOP (Object Oriented Programing) feature of LabVIEW is used during the implementation. The LabVIEW OOP programing makes the implementation of XCP server emulator extensible. It separates the debug port interfaces, and the XCP transport layers from the XCP server functionality, therefore make it possible to use the implementation with different XCP transport layers or debug port interfaces.

Table 1. Comparing interfacing options.

	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

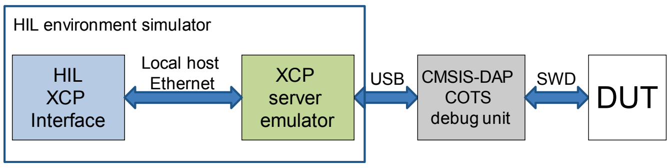

Figure 3. Architecture of the suggested solution.

The simplified class diagram of the XCP server emulator is shown on Figure 4. The XCP Server class contains one XCP Transport Layer, one ECU Interface and a XCP DAQ processor class.

6. IMPLEMENTATION OF THE ECU INTERFACE

The goal of the ECU interface is to provide the non-intrusive connection to the target. The various ECU interface implementations are based on the ECU Interface abstract parent class. Currently only two ECU Interfaces are implemented, an ECU Simulator interface for toolset testing purposes, and a CMSIS-DAP interface based one to demonstrate the concept.

6.1. The CMSIS-DAP interface

To understand the CMSIS-DAP based ECU Interface a short overview of the internal debug architecture of an ARM Cortex core microcontroller (Figure 5) is needed.

External debug hardware can be connected to an ARM Cortex microcontroller either using a 5-wire JTAG or a 2-wire SWD (Serial Wire Debug) Debug Port (DP) [8]. 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

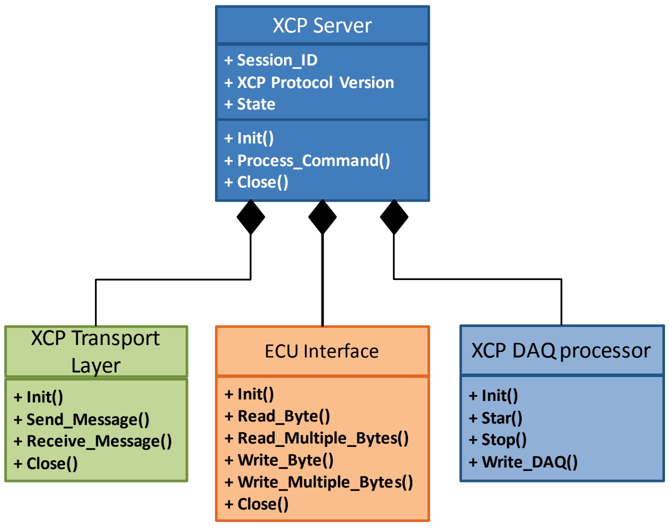

Figure 4. Simplified class diagram of XCP server emulator.

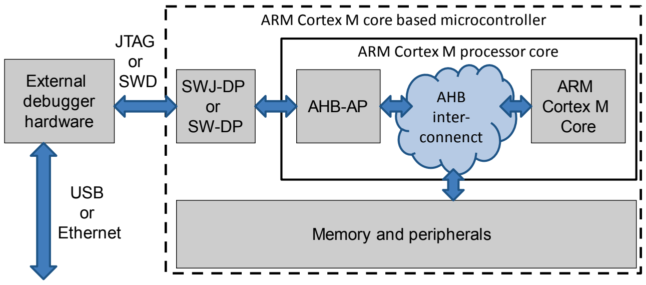

Figure 5. Debug port connection of an ARM Cortex M core.

condition for non-intrusive diagnostic).

Traditionally the external debug hardware is manufacturer depended as described in Chapter 5. Fortunately, ARM has introduced the CMSIS (Cortex Microcontroller Software Interface Standard) with the Cortex core [9]. 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 (Figure 6). 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.

6.2. Implementation of the CMSIS-DAP based ECU Interface

The CMSIS DAP ECU Interface implements the Init(), Read_Byte(), Read_Multiple_Bytes(), Write_Byte(), Write_Multiple_Bytes(), Close() methods of the ECU Interface abstract class.

To perform this implementation, the connection to the debug hardware is needed. This requires an USB-HID class communication under LabVIEW. LabVIEW do not support custom USB-HID devices, therefore a Raw VISA USB implementation is needed [10]. This means, as a first step a special driver needs to be created by using NI Driver Wizard, then a RAW interrupt endpoint communication needs to be performed.

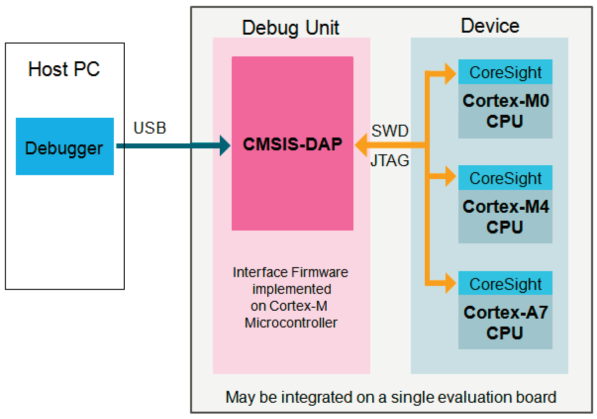

Figure 6. The CMSIS-DAP interface [9].

The commands available for interacting to the debug hardware is described in [9]. To initialize the debug hardware, the debug connection mode (JTAG or SWD) should be selected and the debug clock rate need to be determined.

Writing and reading memory values are done by sending transfer commands through the AHB-AP (Debug port connection of an ARM Cortex M core Figure 5). The AHB bus is a 32-bit aligned bus therefore this read and write operations can be done in 32-bit aligned quantities. Aligned means in this case that, read and write addresses divisible with 4 (aligned to 32-bit) can only be used. This means an extra care is required for variables not keeping these boundaries. The debug port allows the transfer of multiple words, that can increase the effective data rate of the communication.

7. IMPLEMENTATION OF XCP TRANSPORT LAYER, XCP SERVER AND XCP DAQ PROCESSOR

7.1. XCP Transport Layer

The goal of the XCP transport layer is to transport the XCP messages through various network communication interfaces. XCP supports: Serial, USB, Ethernet-TCP, Ethernet-UDP, CAN and FlexRay interfaces.

Currently one UDP (User Datagram Protocol) implementation is created of the XCP Transport Layer Abstract Class. This implementation uses the built in UDP communication VI-s of LabVIEW. The transport layer functionality is simple: receives frames from the UDP port, handle the transport layer specific headers and provides data for decoding and processing in the Process_Command() method of XCP server. The response messages of XCP Server and DAQ messages of XCP DAQ Processor is also transmitted through, the XCP Transport Layer class. In this case the UDP specific headers are added to the data and after that the message is sent to the XCP master application by using NI UDP VI-s.

7.2. XCP Server

The XCP Server class is the main component of the toolset. It processes the incoming messages of XCP master and interacts with the DUT by using the ECU Interface class.

Current implementation of the XCP Server is a basic implementation, which only implements the most vital mandatory commands of XCP. These commands include the connection and disconnections to an XCP master, directly reading data from the DUT's memory, directly writing data to the memory. It also implements very basic commands for handling the XCP DAQ processor, and commands giving some state and configuration information about the XCP Server itself.

However, this is a basic implementation, but these mandatory functions are enough to demonstrate the capabilities of the concept. The XCP Server class is easily extendable with new commands in the future.

7.3. XCP DAQ Processor

The purpose of the DAQ processor it to provide an automated data acquisition. To perform this the DAQ processor maintains ODT-s (Object Descriptor Table). These ODT-s are used to make connection between the memory of the DUT and the XCP DAQ messages, each ODT describes a set of variables to be transmitted as part of a DAQ message.

Current implementation contains a very basic version of the DAQ processor, which is extensible in the future.

8. INTEGRATING THE XCP SERVER EMULATOR INTO A HIL TEST ENVIRONMENT

The XCP server emulator is written in LabVIEW; therefore, it is a straightforward decision to use NI VeriStand as HIL environment form National Instruments.

8.1. NI VeriStand

The NI VeriStand [11] is one of the most widespread HIL development environments used by many automotive and non-automotive companies for HIL testing.

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, the 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 (shown on Figure 7).

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

8.2. Integration into NI VeriStand

It is relatively easy to integrate XCP based diagnostic into NI-VeriStand, because there is an add-on the NI ECU Measurement and Calibration Toolkit for this [12]. Using this add-on, only the XCP Server emulator from Figure 3. needs to be integrated manually.

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 executed task [13]. In this integration the parallel executed task version is need, because the XCP server functionality is too complicated to be implemented in in-cycle mode. For example, the DAQ Processor functions could not be executed without running it in a parallel task.

The XCP Server emulator is integrated into NI VeriStand. The architecture of this integration is shown on Figure 8. This implementation currently a minimal version, where the configuration of the XCP Server is limited, but it is suitable to demonstrate the concept.

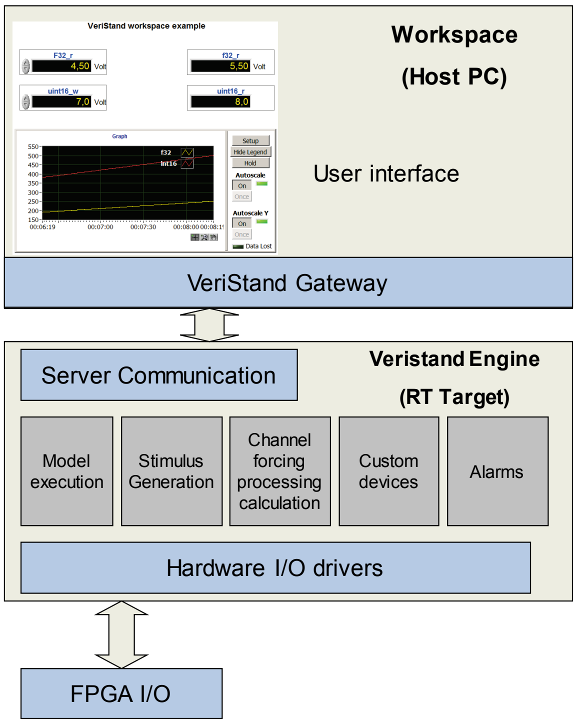

Figure 7. The architecture of NI VeriStand.

9. EXPERIMENTAL MEASUREMENTS

This chapter presents an experimental setup and measurements used for testing our toolset.

9.1. Experimental Setup

The experimental setup is shown on Figure 9. It consists of a Test Station running LabVIEW 2016, NI ECU Measurement and Calibration Toolkit, and our new toolkit. This test station is connected to a Device Under Test embedded system, the mbed LPC1768 [14] (it costs about $50), which contains an on-board CMSIS-DAP debugger and the LPC1768 Cortex M3 core based microcontroller.

9.2. Code Running on the LPC1768 microcontroller

A very basic code was used to demonstrate the concept. This code is presented on Figure 10. The

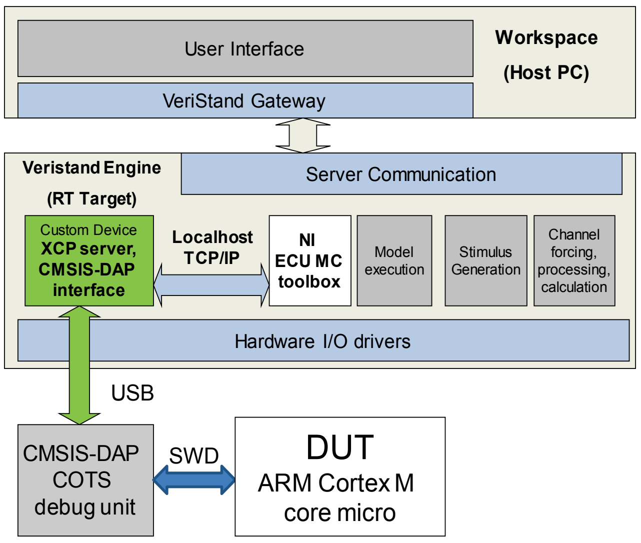

Figure 8. Integrating the XCP Server functionality into NI Veristand.

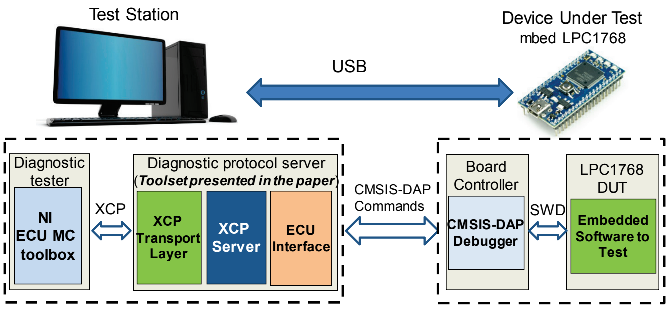

Figure 9. Experimental Setup for testing the toolset.

```
volatile uint16_t a,b,x,y  = 0;
volatile uint32_t counter = 0;
void my_function_to_test(void)
{
            y = a*x+b;
            counter++;
}
```

Figure 10. Part of the code running on the LPC1768 microcontroller.

my_function_to_test() function is called periodically and the diagnostic will be able to read or modify the variables without modifying the execution of the program.

9.3. Configuring the Diagnostic Protocol Server

The use of our toolset can be very easy. In most of the cases only the XCP Transport Layer and the ECU Interface type should be specified. The demo code for the presented example is shown on Figure 11. The XCP Server initialisation requires the type of the XCP Transport Layer (in this case UDP), and the type of the ECU Interface (in this case CMSIS-DAP), then the XCP command processor should be executed in a while loop.

9.4. Testing and measuring

The test software is written using the VI-s of the NI ECU Measurement and Calibration Toolkit. The front panel of the example measurement window is shown on Figure 12. For the measurement, the XCP Transport protocol need to be determined here again, and there also a need for a database that contains the variables intended to measure or modify. This database is called the ASAM MCD2 database or A2L file, that contains the memory addresses and scaling of the parameters downloaded to the Device Under Test embedded system.

The response times were measured during this experimental test. The response time measurements include the writing of every parameter: a, b, x and reading back the measurement

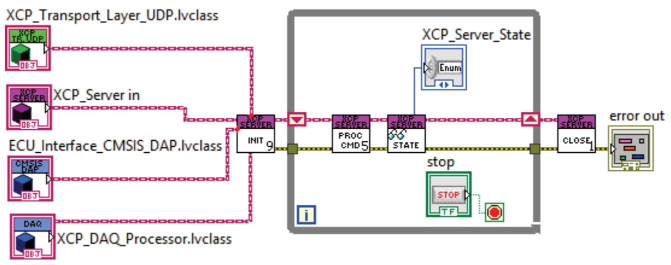
XCP_Transport_Layer_UDP.lvclass
XCP_Server in
ECU_Interface_CMSIS_DAP.lvclass
XCP_DAQ_Processor.lvclass
XCP_Server_State
INIT
PROC CMD
STATE
stop
STOP
CLOSE
error out

Figure 11. XCP Server with CMSIS-DAP communication implemented with the new toolset.

variables: counter and y. The results are shown on Figure 13. This example is using a complex read, write scenario, similar to the one used in normal test cases.

A complex analysis of performance comparing to traditional approach is presented in Chapter 10.

10. COMPARING THE PERFORMANCE OF THE CMSIS-DAP BASED SOLLUTION TO THE DIAGNOSTIC PROTOCOL BASED SOLLUTIONS

10.1. Performance of a diagnostic protocol based implementation

The performance of an XCP based diagnostic protocol based solution is highly depending on the used communication layer.

A 1 Mb/s CAN bus can be considered as a very typical communication layer for a calculation example. This 1 Mb/s CAN bus data rate would mean about a maximum of 450 kb/s diagnostic data rate in case of DAQ message based acquisition. However, in reality this data rate in unachievable, the CAN bus cannot be loaded to 100 %, and in many situations the diagnostic protocol shares the communication bus with the application. Therefore, usually a busload higher than 10 % - 20 % is not allowed for the diagnostic communication that results a 40-100 kb/s diagnostic data rate. This data rate is enough to read the values of about 30 – 100 variables in every 10 ms.

If direct memory reads or writes, are used instead of DAQ based synchronised measurements, then the above data rate can drop to even 1/10 due to the command – response type of operation.

Native USB or Ethernet based XCP servers could provide a significantly higher data bandwidth, but they cause huge overhead to the device under test, and therefore cannot be used in low resourced cyber-physical systems.

10.2. Performance of the CMSIS-DAP interface based implementation

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

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

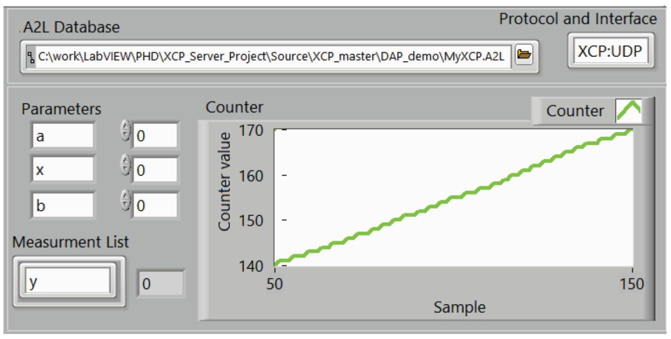
A2L Database
C:\work\LabVIEW\PHD\XCP_Server_Project\Source\XCP_master\DAP_demo\MyXCP.A2L
Protocol and Interface
XCP:UDP
Parameters
a 0
x 0
b 0
Measurment List
y 0

Figure 12. Measurement and test panel using NI ECU MC toolset.

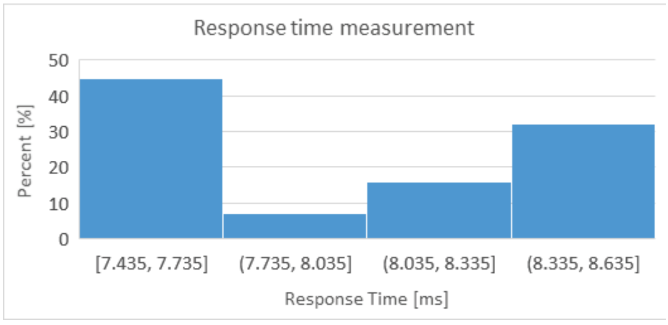

Figure 13. Measurement results of the experimental setup.

cycle.

To enable simple calculations, 138 clock cycles can be considered as 32-bit variable accesses. HIL tests usually use a 1 ms, or 10 ms cycle times. This means 7 pieces of 32-bit variable accessing for 1 ms cycle time, and 72 pieces for 10 ms cycle time. Measurements based on an LPC1768 mbed hardware [14] show slower real data rates, as shown in Figure 14. These measurements show that the data rate is mainly depend on the SDW clock rate. Well organized memory architecture allowing continuous data block reads can also increase the performance significantly. However, these data rates are smaller than the calculated maximum value, but are still in the range that a 1 Mb/s CAN diagnostic can provide.

Fortunately, the data rate of the CMSIS-DAP port based solution can be increased relatively easily. The SWD clock frequency is not limited in 1 MHz. The DUT's hardware gives limitation to the maximum SWD clock rate, but typically this limitation is in the N*10 MHz 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-bit microcontroller used in most of the COTS debug hardware should be replaced by a more capable microcontroller, for example one from the LPC18xx series. These microcontrollers are not much costlier than the LPC11U35 (the cost difference is not higher than $ 10). This replacement means hardware redesign and software porting, but both design processes are relatively easy. Probably COTS debug hardware with these features will appear on the market soon.

11. CONCLUSIONS

This paper has introduced a novel approach, and the first part of a toolset for non-intrusive diagnostics of cyber-physical systems. Our goal is to create a toolset, which supports 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 solution 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 basic version 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.

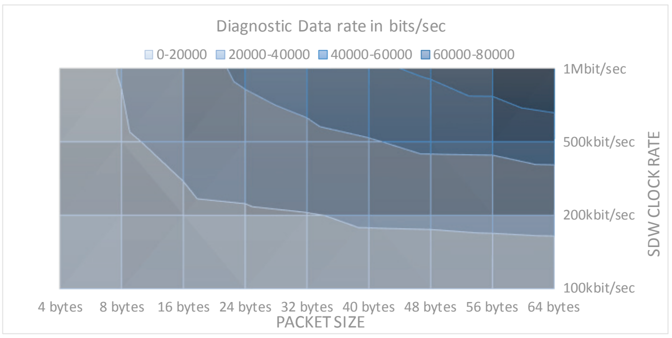

Figure 14. Diagnostic data rate in function of SDW clock rate and packet size.

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 download to the DUT could also be implemented.

The program download is one of the hardest features to implement, because it highly depends on the target microcontroller's Flash memory handling. Usually XCP sample implementations need user contribution 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 [15]), 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 out of the scope of this paper.

The presented toolset is downloadable at [16]. It is tested with the mbed LPC1768 hardware with on board debug support, but any low cost (around $ 20) CMSIS-DAP debugger can also be used with the tool.

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 20.11.2017].

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

[7] National Instruments LabVIEW https://www.ni.com/getting-started/labview-basics/ [Access date 20.11.2017].

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

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

[10] National Instruments, "USB Instrument Control Tutorial", 2014 http://www.ni.com/tutorial/4478/en/ [Access date 20.11.2017]

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

[12] National Instruments, "NI ECU Measurement and Calibration Toolkit". [Online] Available: http://sine.ni.com/nips/cds/view/p/lang/hu/nid/210569 [Access date: 20.11.2017]

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

[14] The mbed LPC1768 development board:. https://os.mbed.com/platforms/mbed-LPC1768/ [Access date: 20.11.2017].

[15] ARMmbed, "The mbed HDK (Hardware Development Kit)" [Online] Available: https://developer.mbed.org/handbook/mbed-HDK#access-the-mbed-hdk-repository [Access date: 20.11.2017].

[16] Balázs Scherer, "Homepage of the CMSIS-DAP based XCP Server Project": http://mit.bme.hu/~scherer/XCP_Server_CMSIS_DAP/introduction.htm [Access date: 20.11.2017].