American Journal of Mechanical Engineering, 2013, Vol. 1, No. 7, 276-281
Available online at http://pubs.sciepub.com/ajme/1/7/25
© Science and Education Publishing
DOI:10.12691/ajme-1-7-25

Model Based Design and HIL Simulations

Tatiana Kelemenová[1], Michal Kelemen[2,*], Ľubica Miková[2], Vladislav Maxim[3], Erik Prada[2], Tomáš Lipták[2], František Menda[2]

[1]Department Biomedical Engineering and Measurement, Technical University of Košice, Faculty of Mechanical Engineering, Košice, Slovak Republic
[2]Department of Applied Mechanics and Mechatronics, Technical University of Košice, Faculty of Mechanical Engineering, Košice, Slovak Republic
[3]Department of Automation, Control and Human Machine Interaction, Technical University of Košice, Faculty of Mechanical Engineering, Košice, Slovak Republic
*Corresponding author: michal.kelemen@tuke.sk

Received October 02, 2013; Revised October 11, 2013; Accepted November 18, 2013

Abstract Model-Based Design is a process that enables faster, more cost-effective development of dynamic systems, including control systems, signal processing, and communications systems. It enables to reduce time to market and these products have higher safety and reliability. Many products are mechatronic and this design approach is dedicated right for this type of products.

Keywords: *model, design, simulation, hardware-in the-loop simulation*

Cite This Article: Tatiana Kelemenová, Michal Kelemen, Ľubica Miková, Erik Prada, Tomáš Lipták, František Menda and Vladislav Maxim, "Model Based Design and HIL Simulations." *American Journal of Mechanical Engineering* 1, no. 7 (2013): 276-281. doi: 10.12691/ajme-1-7-25.

1. Introduction

Many industries are under pressure to reduce their development times when they produce unique and innovative products. Working efficiently is indispensable to success in a globalized market, especially for high-tech industries such as automotives, aerospace and communications, where electronic controls are a vital part of each new product. Model-based control design is the time-saving, cost-effective approach, because control engineers work with just a single model of a function or complete system in an integrated software environment. This model-based development process results in an optimized, validated system, and there is no risk that individual components do not fit together optimally [1,2].

In Model-Based Design, a system model is at the center of the development process, from requirements development, through design, implementation, and testing. Model-Based Design is transforming the way engineers and scientists work by moving design tasks from the lab and field to the desktop [3].

Model-Based Design is a math-based visual method for designing complex control systems and is being used successfully in many motion control, industrial, aerospace, and automotive applications. It provides an efficient methodology that includes four key elements in the development process: modelling a plant (from first principles or system identification), synthesizing and analyzing a controller for the plant, simulating the plant and controller together, and programming/deploying the controller. Model-Based Design integrates all these multiple phases and provides a common framework for communication throughout the entire design process [4].

The Model-Based Design paradigm is significantly different from traditional design methodology. Rather than using complex structures and extensive software code, designers can formulate advanced functional characteristics by using continuous-time and discrete-time computational building blocks. These models and associated simulation support tools can provide rapid prototyping, virtual functional verification, and software testing and hardware/software validation. Model-Based Design is a process that enables faster, more cost-effective development of dynamic systems, including control systems, signal processing, and communications systems. In Model-Based Design, a system model is at the center of the development process, from requirements development, through design, implementation, and testing. The control algorithm model is an executable specification that is continually refined and elaborated throughout the development process [4].

Model-Based Design is a process that enables faster, more cost-effective development of dynamic systems, including control systems, signal processing, and communications systems. In Model-Based Design, a system model is at the center of the development process, from requirements development, through design, implementation, and testing. The model is an executable specification that you continually refine throughout the development process. After model development, simulation shows whether the model works correctly.

When software and hardware implementation requirements are included, such as fixed-point and timing behaviour, you can automatically generate code for embedded deployment and create test benches for system verification, saving time and avoiding the introduction of manually coded errors [1-8].

Model-Based Design (as shown in Figure 1) allows you to improve efficiency by:

- using a common design environment across project teams,
- linking designs directly to requirements,
- integrating testing with design to continuously identify and correct errors,
- refining algorithms through multi-domain simulation,
- automatically generating embedded software code,
- developing and reusing test suites,
- automatically generating documentation,
- Reusing designs to deploy systems across multiple processors and hardware targets.

Model-Based Design helps engineers achieve certification to safety standards by supporting requirements traceability, verification, and documentation. These capabilities span multiple design stages. For example, requirements linked to model are inserted as comments in generated code. Qualification kits, available for several verification tools, can reduce the amount of manual review needed.

It is also increasingly common for organizations to adopt Model-Based Design on large programs spanning multiple organizations. This allows system-level performance to be assessed and integration issues to be uncovered much earlier in the design process.

When detailed models from multiple organizations are combined, resulting models can contain hundreds of thousands of blocks. Modelling tools have evolved to meet these challenges with improved support for large-scale modelling, including support for composite models from other model files and support for signal buses.

When organizations adopt Model-Based Design, they improve product quality and reduce development time by 50% or more. It also causes that product will be cheaper and more competitive on market [1-8].

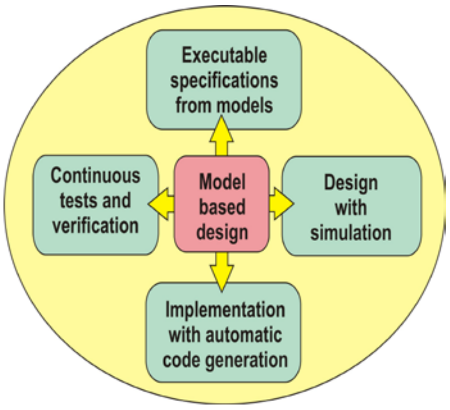
Executable specifications from models
Continuous tests and verification
Model based design
Design with simulation
Implementation with automatic code generation

Figure 1. Model based design philosophy [8]

2. Model-Based Design Workflow

In a traditional workflow, requirements analysis and high-level design tradeoffs are often done on paper or with a basic tool such as Excel or using expensive prototype hardware. Algorithm implementation in C or Hardware Description Language (HDL) is done manually, as is the testing and verification. This process often requires several iterations between the algorithm design team and the C and HDL teams due to unclear requirements and design documentation or inherent design difficulties.

With Model-Based Design, system engineers use models to derive low-level requirements and then use the models to interface with customers and suppliers. They model the behaviour of digital, analogue, and RF components of the system and perform design tradeoffs in simulation, which can be difficult or impossible using paper-based design reviews. Algorithm developers can then reuse and elaborate the same models to build and test more detailed designs. Further in the development process, these models can become the design artefacts from which hardware engineers automatically generate HDL code. Then the system level models and tests can be reused as a test bench to validate the performance of the HDL implementation and the final hardware against the model level results. With Model-Based Design, models are re-used and elaborated at every development phase, reducing the amount of translation inefficiencies and errors in the process. With the model at the centre of the design process, design iterations are faster, design artefacts are automatically generated, and engineering teams have a common platform to share design [9].

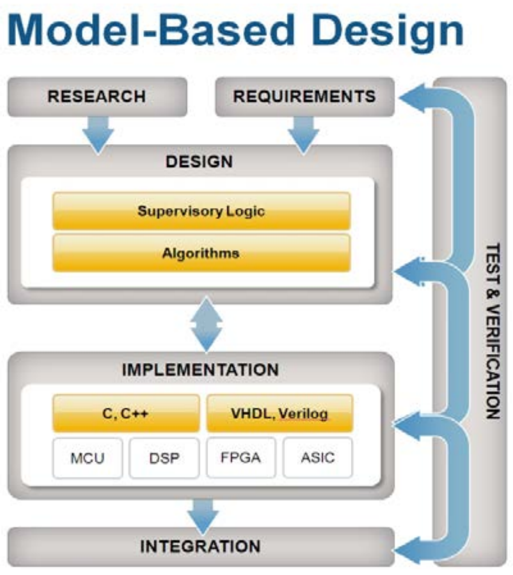

Figure 2. Model-Based Design provides a multi-domain tool environment and integrates all phases of development [9]

In generally, there are six steps to modeling any system:

- Defining the System
- Identifying System Components
- Modeling the System with Equations
- Building the model
- Running the Simulation
- Validating the Simulation Results

We perform the first three steps of this process outside of the software environment before we begin building our model. Mainly, these first three steps are important and many people make mistakes in these steps. Finally, when system is modeled through the equations, next building of the model is more or less routine operation. It is important to say that every model is not perfect and every time we have to neglect any points and simplify system description. Overall process requires the experiences. When we will try to make absolutely prefect model, very complicated model and slowly simulation will be as the result of them [5].

Defining of the system - the first step in modeling a dynamic system is to fully define the system. If we are modeling a large system that can be broken into parts, we should model each subcomponent on its own. Then, after building each component, we can integrate them into a complete model of the system.

Identifying System Components - the second step in the modeling process is to identify the system components. Three types of components define a system: parameters (system values that remain constant unless you change them), states (variables in the system that change over time), and signals (input and output values that change dynamically during a simulation).

Modeling the System with Equations - the third step in modeling a system is to formulate the mathematical equations that describe the system. For each subsystem, use the list of system components that we identified to describe the system mathematically. Model may include: algebraic equations, logical equations, differential equations, for continuous systems and difference equations, for discrete systems etc.

Building the Simulink Block Diagram - after we have defined the mathematical equations that describe each subsystem, we can begin building a block diagram of our model for example in MATLAB/Simulink. Build the block diagram for each of our subcomponents separately. After we have modeled each subcomponent, we can then integrate them into a complete model of the system.

Running the Simulation - after we build the Simulink block diagram, we can simulate the model and analyze the results. Simulink allows us to interactively define system inputs, simulate the model, and observe changes in behavior. This allows us to quickly evaluate your model.

Validating the Simulation Results - finally, we must validate that our model accurately represents the physical characteristics of the dynamic system. We can use the linearization and trimming tools available from the MATLAB command line, plus the many tools in MATLAB and its application toolboxes to analyze and validate our model [5,6,7].

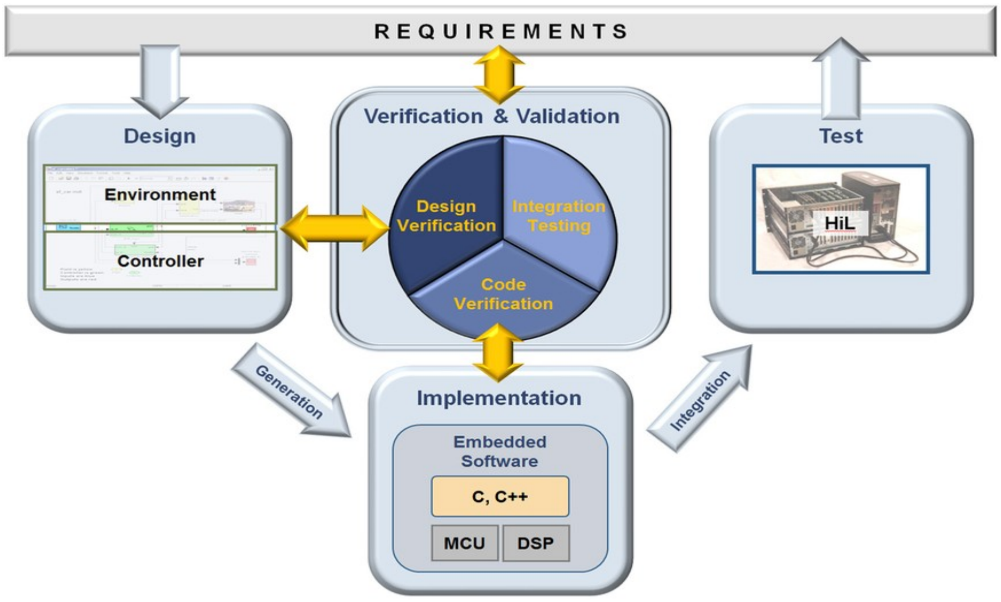

Figure 3. Early verification as part of Model-Based Design streamlines embedded control design with modeling, simulation, and automatic code generation [7]

Model-Based Design with MATLAB and Simulink is an efficient and cost-effective way to develop complex embedded systems in aerospace, automotive, communications, and other industries. It enables system- and component-level design and simulation, automatic code generation, and continuous test and verification. [5,6].

Plant models provide another perspective on the system. Modeling the non-software parts of the system gives engineers another view into system behavior. Engineers can often learn more about system dynamics through simulation than from the real system because simulation provides details on force, torque, current, and other values that are difficult or impossible to measure on the actual hardware [7].

Creating plant models requires engineering effort, but this effort is often overestimated, while the value provided by plant modeling is underestimated. When developing plant models, it is a best practice to start at a high level of abstraction and add details as needed. Choosing a level of abstraction that is just detailed enough to produce the needed results saves modeling effort as well as simulation time (see Figure 2) [5,7].

System behavior is defined not only by the embedded control software, but also by the electronic and mechanical components, including the connected sensors and actuators. Early simulations in which the architecture is executed provide more insight when they are performed in a closed loop with plant or environment models. System-level optimization requires multi-domain simulations. It is impossible to optimize today's sophisticated systems by tuning one parameter at a time. To deliver maximum energy efficiency and highest performance at minimal material cost, engineers must optimize the system as a whole, and not just the embedded software [5,6,7].

3. Model Based Simulation Modes

There are different simulation modes Figure 5 for the model (model-in-the-loop (MIL) simulation), the host implementation (software-in-the-loop (SIL) simulation) and the target implementation (processor-in-the-loop (PIL) simulation) [2].

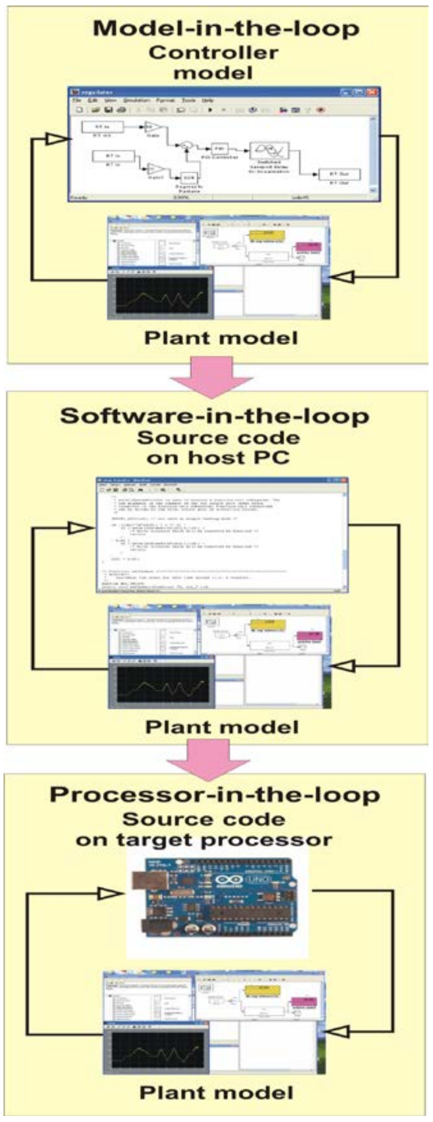

Figure 4. Simulation modes [2]

These methods Figure 5 avoid work-intensive iterations in later development phases, and save time and money by:

• Verifying at an early stage, by means of model simulation, that the model and requirement are correct

• Verifying that the code and the mode are consistent, and that the code correctly represents the model's functionality, by simulating the generated code on the host PC

• Verifying seamless traceability for documenting the software development

• Allowing resource requirements to be estimated at an early stage by simulating the code on the appropriate evaluation hardware [2].

4. Hardware-in-the-loop Simulation (HIL)

Not only is the number of electronic control units (ECUs) in modern vehicles constantly increasing, the software of the ECUs is also becoming more complex. Both make testing a central task within the development of automotive electronics. Testing ECUs in real vehicles is time-consuming and costly, and comes very late in the automotive development process. It is therefore increasingly being replaced by laboratory tests using hardware-in-the-loop (HIL) simulation Figure 5 [10,11].

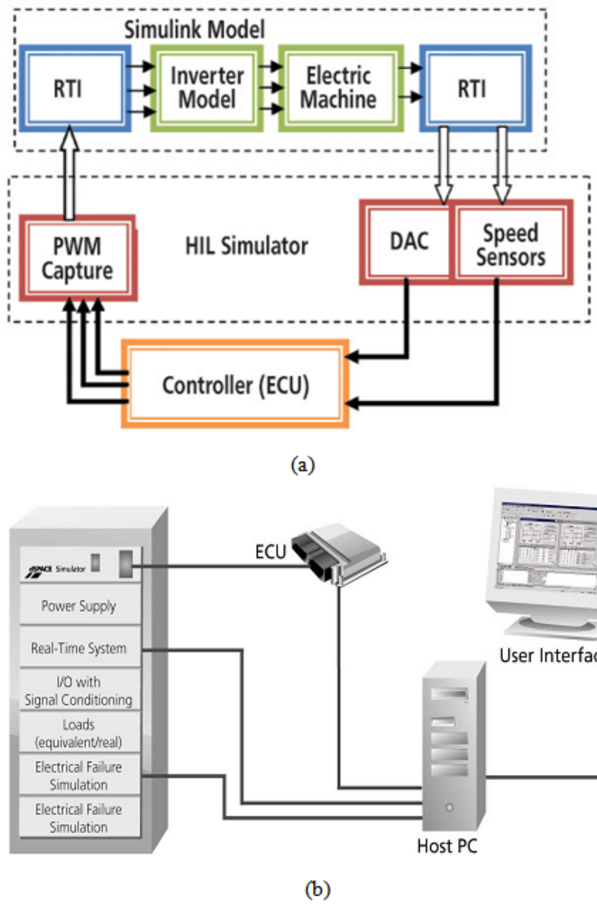

Figure 5. (a) The Components Required for High-Speed Simulation of the Electric Machine (b) Typical architecture for Hardware-in-the-loop HIL simulations [10,11]

Time to market is speeding up, especially in automotive electronics. 90% of automotive innovations are currently connected with new electronics. Test drives can scarcely cope with the volume of systematic testing needed, especially just before start of production. The growing number of recall campaigns is a clear indication of this. It is little wonder that testing and error finding have become key tasks in the development process.

ECU testing typically is done using hardware-in-the-loop simulation. The ECU (prototype) is connected to a real-time simulation system simulating the plant (engine, vehicle dynamics, transmission, etc.) or even the whole vehicle.

As HIL has become a standard method for testing ECUs and control strategies during the whole development cycle (i.e., not only after availability of the final ECUs), different needs of different users have to be addressed by the various test systems [10,11].

The dSPACE software components are standardized and can be integrated in any dSPACE simulator. The tight integration of dSPACE software and the modelling tool MATLAB®/Simulink® from The MathWorks provides a powerful development environment.

dSPACE Simulator's graphical user interface provides a convenient and flexible environment. Simulated driving cycles, data acquisition, instrumentation, monitoring, test automation and all other tasks are executed graphically within dSPACE Simulator. The hardware requirements, however, vary immensely depending on the HIL application Figure 6. For example, function tests typically are executed with simulators that have a fixed (super)set of I/O, and adaptations to the ECU are most often made in the cable harness. In contrast, acceptance tests call for flexible and combinable simulator setups [10,11].

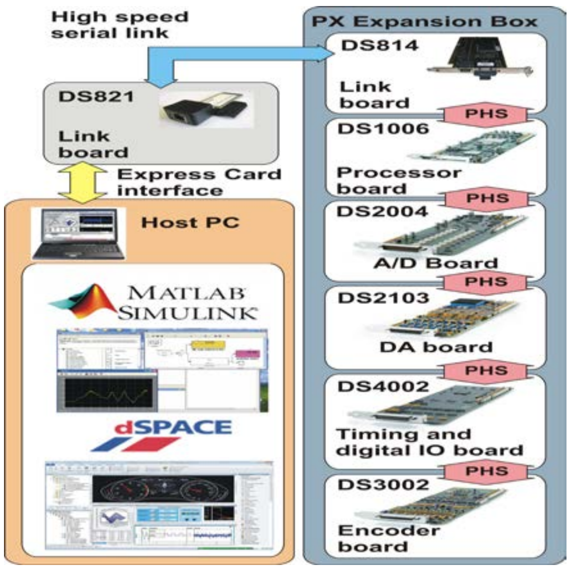

Figure 6. dSpace system for HIL simulation [5]

An appropriate test strategy is the key to getting maximum benefit from an HIL simulator. While the first tests during function development are typically performed initially on a manual basis, the function developer soon goes over to automated tests. Typically, very detailed tests are created at this stage. A thorough knowledge of the implemented structure of the software is required.

These so-called white-box tests are based on a thorough knowledge of the internals of an ECU. They use not only input and output variables, but also internal variables (model variables and internal ECU variables). At this stage, measuring internal ECU variables is indispensable, as described in diagnostic interface.

White-box tests typically are applied during function development. They have proven successful in error finding, for example, when problems occur during release and acceptance tests.

During classical HIL simulation at the end of the development process, black-box tests are typically performed.

Black-box tests concentrate on the specification of the functionality of the ECU under test, so usually only its outer interface (inputs and outputs, no internal values) is accessed. The test description and implementation can already be done according to the specification of the function under test.

A test pool of all types of tests allows recursive testing for the different ECU versions, including the final test for ECU release. This test pool also allows white-box tests to be rerun if a problem occurs during integration tests. Problems can be narrowed down to their source with the help of existing tests.

For each of the above-mentioned areas, the HIL test specifications are developed at the same time as the performance specifications [10].

Conclusion

With Model-Based Design, engineers use models as a golden reference that links every part of the development process-requirements, design, implementation, and testing. The time invested in creating models upfront saves more time at every subsequent stage of the development process through reuse or elaboration of design elements or by identifying errors when it is less time and cost consuming to fix them. The models are used as an executable specification in the requirement phase, a design platform in the algorithm design phase, a code generation source in the hardware implementation phase, and a test bench in the testing phase. By reusing the models, system engineers are free to focus on design innovation to achieve higher system performance [9].

Model-based design comprises four elements: modelling of desired behaviour or reference designs; design exploration and refinement through simulation; implementation with code generation; and continuous test and verification throughout the development process. These elements address the design and verification issues inherent in today's electronic systems by letting engineers progress systematically from specification to implementation to verification, leveraging executable system models that unambiguously specify functional and physical requirements.

In traditional design flows, there can be no systematic testing of the entire system until it has been fully implemented. As a result, bugs remain hidden until late in the system development, when fixing them is significantly more costly and disruptive.

In contrast, model-based design enables system test and debug from the earliest stage of development, when most flaws are introduced. Models can be validated early on through simulation and verified continuously as the component models are refined with additional implementation detail. C, HDL and Spice implementations, as well as ESL models, can be incorporated to support existing workflows, design reuse and final integration testing. Components from different design teams can be integrated as they become available, ensuring that any changes do not degrade system performance and that bugs can be quickly isolated to the offending component [10-16].

Acknowledgement

The authors would like to thank to Slovak Grant Agency – project APVV-0091-11, VEGA 1/1205/12. This paper is the result of the project implementation: Research of modules for intelligent robotic systems (ITMS: 26220220141) supported by the Research & Development Operational Programme funded by the ERDF.

References

[1] dSpace GmbH, "*Model Based Control Design*", available online at http://www.dspace.com/en/pub/home/products/systems/controldesign.cfm.

[2] dSpace GmbH, "*Model-Based Development of Safety-Critical Software: Safe and Efficient Translation of "Sicherheitskritische Software entwickeln*" Published at: MEDengineering, 06/2012. Available online at http://www.dspace.com.

[3] MathWorks, Inc., "*What is Model-Based Design*?" available online at http://www.mathworks.com/model-based-design/.

[4] A. Rousseau, "*Model-Based Design Approach*", Argonne National Laboratory, available online at http://www.autonomie.net/projects/model_based_design_10.html.

[5] The MathWorks, Inc., “*Model-Based Design*”. Documentation Center. Cited 07-11-2013. Available online. http://www.mathworks.com/help/simulink/gs/model-based-design.html.

[6] R.Otterbach, “*Automotive Solutions, Systems and Applications*”, dSPACE GmbH. 2013. Available online. http://www.dspace.com.

[7] G. Sandmann, J. Schlosser, “*Maximizing the benefits of Model-Based Design through early verification*”. Embedded Computing Design. An OpenSystems Media publication. Available online. Page last refreshed: Mon, 08 Jul 2013 05:04:12. http://embedded-computing.com/articles/maximizing-benefits-model-based-design-early-verification/#.

[8] MathWorks, Inc., “*Evolution of model-based design in aerospace*”, IML Group PLC. Available online. http://www.epdtonthenet.net/article/41911/Evolution-of-model-based-design-in-aerospace.aspx.

[9] J. Lin, “*Developing Next Generation Signal Processing and Communications Systems: Engineering Tools and Design Flow Advancements*”, Electronics News, January 2013, Available online. http://www.electronicsnews.com.au/technical-articles/developing-next-generation-signal-processing-and-c.

[10] S. Köhl, D. Jegminat, “*How to Do Hardware-in-the-Loop Simulation Right*”. dSPACE GmbH. SAE International. 2005. 2005 SAE World Congress Detroit, Michigan, April 11-14, 2005. SAE Technical paper series. Reprinted From: Controller System Software Testing and Validation (SP-1928).

[11] A. Dhaliwal, S. Nagaraj, S. Jogi, “*Hardware-in-the-Loop Testing for Hybrid Vehicles*”. dSPACE GmbH, Evaluation Engineering, November 2009, NP Communications, LLC. dSPACE, 50131 Pontiac Trail, Wixom, MI 48393. Also available online. Cited 07-11-2013.

[12] K. Karnofsky, “*Putting the system in electronic system design*”. EETimes Newsletter 2/4/2008, UBM Tech. Also available online. Cited 07-11-2013. http://www.eetimes.com/document.asp?doc_id=1271606.

[13] F. Trebuňa, F. Šimčák, “*Handbook of experimental mechanics*”. 1st edition. Košice: TU of Kosice, Fac. Of Mech. Eng. 2007. 1526 pages. 2007.

[14] A. Vitko, L. Jurišica, M. Kľúčik, R. Murár, F. Duchon, “Embedding Intelligence Into a Mobile Robot”, *AT&P Journal Plus*. No. 1: Mobile robotic systems (2008), s. 42-44. 2008.

[15] L. Hargaš, M. Hrianka, D. Koniar and P. Izák, P. “Quality Assessment SMT Technology by Virtual Instrumentation”, *Applied Electronics* 2007, Pilsen, 5. - 6. 9. 2007.

[16] P. Kuryło, P., M. Nagórny, Some Problems of Automation and Robotization of Welding Process in the Large Size Constructions. Pomiary, Automatyka, Robotyka, Vol. 16, No. 2, pp. 132-136. 2012.