A HiL Test Bench for Verification and Validation Purposes of Model-Based Developed Applications Using Simulink® and OPC DA Technology

W.Chaaban[1], M. Schwarz[1], B. Batchuluun[1], H. Sheng[1] and J. Börcsök[1]

[1]University of Kassel, Willhelmshöher Allee 73, Germany
walid.chaaban@uni-kassel.de, m.schwarz @uni-kassel.de,
batsuren@uni-kassel.de, hsheng@uni-kassel.de

Abstract

This paper deals with the development procedures of a HiL (Hardware-in-the-Loop) test bench for verification and validation purposes of embedded application software developed using MBD (Model Based Design) engineering methodology. The testing environment have been mainly developed for PLC (Programmable Logic Controller) applications using Matlab®/Simulink® as a simulation environment and the OPC DA (OLE for Process Control Data Access) specification as communication interface for data interchange with the OPC compliant (set as working condition) target hardware. The main focus of this work was to offer a tool or an instrument to end user or test engineer that helps testing auto- generated embedded software while running on target system and making decision and statements about correct functionality and performance with regard to main requirements in order to validate the intended mission of the application aimed to reach.

1. Introduction

One of the big challenges of embedded application industries nowadays is to launch superior quality products at lower cost for sale on the market as fast as possible or in a short TTM. TTM stands for Time-to-Market and represents the length of time it takes since the idea of a product has been conceived until the product has been available for sale. This period of time represents a crucial factor and plays an important role in industries due to the big competition between the market leaders and taking into account that products are outmoded quickly and have to be updated and extended as fast as possible in order to meet market needs. The more complex and various are the embedded activities intended to achieve, the more complex, time consuming and error- prone is the belonging hand-written embedded software where was the necessity to invent a new engineering methodology to overcome all facing problems and obstacles. MBD (Model Based Design) is the new trend and has been increasingly used and gaining popularity in the last years in many industrial sectors (such as aircraft, automation and automotive applications etc…), due to the big set of benefits and advantages it provides for both sides manufacturers and end consumers. Some of the benefits are listed in the following [1]:

- Reduced time to market (faster development)
- Detection and elimination of software bugs and errors in early development stages (quality improvement)
- Producing superior quality products at lower cost
- Flexibility concerning updates and last minute changes

The model-based design paradigm is significantly different from conventional design methodology. The central component or starting point of a model based developed application is a system model (simulation model), where designers or developers model the control algorithm or functional characteristics of the intended application as interconnected function blocks provided by the library of the used modelling tool, e.g. Simulink® (graphical programming language from MathWorks®). After model development, simulation in the virtual environment (MiL: Model-in-the-Loop) shows whether the model works correctly with regard to given requirment specifications. In case of requirements' fullfillment an automated code generation according to the hardware target can be started in order to implement the modelled application in an embedded code for hardware deployment, e.g. embedded C code, using commercial autocoders (automatic code generator), e.g. RTW Embedded Coder (RTW stands Real Time Workshop) from MathWorks®. This feature lying in the ability of automatically generating large amounts of code lines at one button touch and in a very small time intervall appears to be one of the main reasons for MBD's growth in popularity. This step reduces radically time taken for code implementation in case of hand-written code used in tradional design methodology and avoids manually coded errors and bugs. The next step consists of software testing, and verification which is not sufficiant for validation since the developed software will be deployed on a target hardware. This is why a Hardware-in-the-Loop (shortened HiL) simulation has to be performed in which time behaviour and hardware influences on the developed software can be tested before any integration or system test are performed. The different steps of the life cycle of the MBD engineering methodology is represented in the V- Diagramm depicted in Fig. 1 [2].

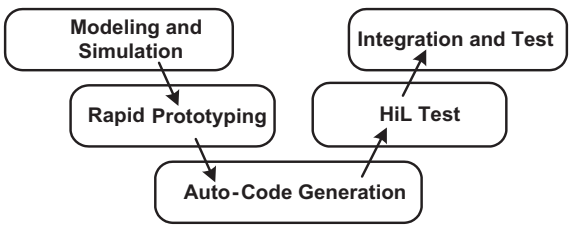


Fig. 1. MBD life cycle (V - Diagramm)

This paper is a demonstration on how a HiL test environment has been developed for validation purposes using Simulink as a development and simulation environment and OPC DA technology as a communication interface for interaction and data interchange with OPC DA compliant target hardware (mainly PLC). Development and tests have been made using HIMA PLCs product family (new generation of HIMatrix and the HIMax systems) as hardware, which can be operated and programmed with the SILworX PADT (Programming and Debugging Tool – Product of the company HIMA). The rest of the paper is structured as follows. Section 2 deals with the different testing approaches accompanied with the different development phases of the MBD life cycle. Section 3 gives a short overview to the development environment and the different software interfaces and tools used during development. Section 4 deals with the focus of this work and the building procedure of the HiL test bench. In section 5 performance analysis and evaluation of the tool are discussed by means of a simple example. Concluding remarks about the tool are explained in part 6 followed by future works in part 7.

2. Testing Approaches

For safety improvement and quality assurance embedded software manufactures have placed over the last years their major responsibility and work on software verification and testing. The main aim of testing is to get an end product that fulfils requirements specifications and satisfies a minimum level of reliability required. A further purpose is to minimize risk of failure since error free systems do not exist in reality [3] [4] and to avoid system crashes and undefined system states that may lead to hazardous situations in case a safety related mission has been dedicated to the control application.

As mentioned earlier one of the big challenges of MBD of embedded control and safety applications is to develop reliable and healthy performing software with minimal risk of failure or software that is able to lead the system into a safe or defined state in case the SIS (Safety Instrumented System) is intended to perform a safety critical function. MBD is well suited for this kind of development since each essential step of its development life cycle is armed with its own testing strategy. 3 different basic test disciplines are performed throughout the different phases of MBD life cycle. Those different testing approaches are listed below:

- *Model-in-the-Loop (MiL)*
- *Software-in-the-Loop (SiL)* and finally
- *Hardware-in-the-Loop (HiL).*

Those 3 test approaches are explained shortly in the following.

MiL Testing: This testing step is used for validation at modelling stage, in order to test the function of the designed control application. During MiL test, the behaviour of the intended function or activity which is later to implement in embedded C or C++ code is simulated as a model in a virtual environment. The model represents a rapid visualization of a control algorithm designed in a graphical development language as a set of interconnected blocks and is a kind of a simplified and an easy to process representation of the real environment, which allows validating the functionality of the intended activity in term of compliance with requirements and to find specification mistakes in early development phases [5].

SiL Testing: This is an intermediate testing approach performed between the MiL and HiL after code implementation from the designed control model. This category intends to test the behavior of the auto-generated software before integration into any target hardware or real- time hardware simulation. This testing approach is an important step but does not completely replace the HiL testing, because the generated or written code may not (for target flexibility reasons) contain yet any code describing the integrating control unit or processor and many previous experiences have shown that simulation and reality always differ [5].

HiL Testing: This testing strategy represents a real-time hardware simulation and the final product test (i.e. it requires the use of hardware targets for execution). This category of test intends to check if the software products still fulfil/meet objectives and if the main functionality keeps remaining while performing on the specific target hardware. One more aim of the HiL Test is to test the interaction between the auto-generated software and the target hardware in order to evaluate product's performance (check hardware influences on software), which represents the main objective of model-based development.

One additional important task during the testing phase is the choose of the test cases simulated during test. Those test cases have to be chosen in a comprehensively manner and required a deep understanding of the tested application. If the test cases on which the simulation is based do not adequately address or take into account the potential error conditions of the design, security and safety leak may stay undiscovered and the developer may well end up with a false sense of security. Care should therefore be taken to ensure that all of the problem space is adequately captured and understood during the modeling process in order to reach the highest or optimal level of test coverage with maximum accuracy [6].

3. Development Environment

This section deals with the environment, in which the HiL test tool has been built and the different software tools and interfaces, that have been used for development. During the development phase of the HiL test environment implemented in this work the following 2 MathWorks® products have been mainly used: Simulink® as a graphical modelling environment using function blocks from own library and the OPC Toolbox™ for the communication management with the target OPC compliant PLC hardware. Additional products like the *SILworX*® Programming and Debugging Tool (PADT) and *X-OPC DA* Server from the HIMA [7] have been used. More information about these products is given in the following subsections.

3.1. Simulink® and OPC Toolbox™

Matlab®/Simulink® from MathWorks represents one of the most widely used development tool at research institutes and universities in model-based design, simulation and auto-code generation of control algorithms due to the big set of various toolboxes and facilities it provides to the developers.

Simulink®: is a commercial modelling tool which popularity has been increasing in the last few years among embedded software developer using MBD methodology. Simulink® represents an environment for design and multidomain simulation of Model-Based Design for dynamic and embedded systems. It provides an interactive graphical environment (graphical block diagramming tool) and a customizable set of block libraries for design, simulation, implementation and testing of control algorithm and many other applications areas [8].

OPC Toolbox™: provides in the context of this work a connection to OPC DA (OLE for Process Control Data Access) servers and acts therefore as an OPC DA client. OPC DA is a Microsoft COM/DCOM (Distributed Component Object Model) based communication standard allowing users to perform read and write access (enterprise management level) to live data and processes running on any target device that conforms to the OPC DA specification, e.g. Distributed Control Systems (DCSs) and PLCs [9].

3.2. SILworX® – PADT and X- OPC DA Server

SILworX® from the HIMA (a company which is concerned with safety- related solutions) is a new commercial, easy-to-use, fully integrated configuration, Programming and Debugging Tool (PADT). SILworX® as engineering tool is IEC 61131-3 (software part) compliant, supports all functions and variable types for safety- related programming according to the prementioned standard and includes a graphical user interface for flexible drag and drop programming of PLC applications (mainly the new HIMax and the new HIMatrix generations) using Function Block Diagrams (FBD) [10].

X-OPC is also a HIMA product and represents an OPC DA specification server which performs write and read access on the address space of the HIMA OPC compliant new generation of PLCs (HIMax and new HIMatrix generations). The configuration and parameter setting of this server including defining accessible items is performed within the context of SILworX®.

4. HiL Test Bench- Idea and Building Procedures

4.1. Main Idea behind this Work

In the scope of a project that intends to develop a tool that translates Simulink® models in PLC projects as C- Interface Funktion (CIF Blocks) including steps needed for verification and validation purposes and the appropriate testing disciplines, the necessity of an instrument that performs HiL simulation for applications developed for PLC targets has arisen. More concretely, the project intends to automatically generate customized C- code from Models developed in Simulink® for the HIMA PLC series (HIMax, HIMatrix) using RTW Embedded Coder, in order to enable the development of control algorithms (e.g. a PID controller), since realizing this kind of complex tasks with the standardized IEC 61131-3 Function Block Diagram (FBD) PLC programming language is limited, due to the poor set of blocks provided by the library defined in this standard.

The tool described in the context of this work is concerned with performing real-time Hardware-in-the-Loop simulation and defines one of the last steps of the MBD life cycle. This step consists of testing and validating the functionality of the auto-generated code during real-time deployment on a HIMA PLC system.

Since Simulink® has been used as engineering tool in creating the converting instrument and the starting point being a Simulink® model, therefore came the idea to create the HiL testing environment using the same tool. One more convincing point in favour of using Simulink® was the high degree of automation one can reach through its usage. A further challenging task consisted of how to establish a connection with the hardware component since HiL testing is mainly based on interaction with hardware. On that basis the idea with the OPC communication interface was born since almost all HIMA PLCs products are OPC compliant and Simulink® supports this communication standard through its promising OPC Toolbox™.

4.2. HiL Test Bench – Building Procedure

A. Simulink® Side

As mentioned previously the HiL test bench is a Simulink® model, which will be generated automatically at the request of the user in one button touch from the main designed Simulink® application intended to be tested.

The auto created HiL test model consists mainly of a so called **Level-2 Matlab S-Function** block, which becomes the Name **HIL Test** and will be attached to an m- function that manages that connects to the preconfigured OPC server and manages data interchange. This function block performs during setup or initialization the following steps

- Determining the number of inputs in the main application's model, in order to add the same number of input pins to the **Level-2 Matlab S-Function**

- Determining the number of outputs in the main application's model, in order to add the same number of output pins to the **Level-2 Matlab S-Function**

After performing the above mentioned steps, so called **Repeating sequence stair** blocks with number of inputs are going to be added into the HiL Test model. The name and data type of each block is going to be changed according to the name and data type of the input it has been added for and it will be afterwards connected to the appropriate input pin of the previously initialized **Level-2 Matlab S-Function**. It is also important to notice that this function block is executed once every sample time. A general arrangement drawing that describes the overall building process of the HiL test environment is depicted in Fig. 2.

A **Repeating Sequence Stair** block outputs or repeats, depending on the total set simulation time of the model, a stair sequence that the user specifies in the **Vector of output values** parameter. In this tool the **Vector of output values** are going to be fed automatically test vectors from a test matrix that has been generated previously by the user and saved into the working directory.

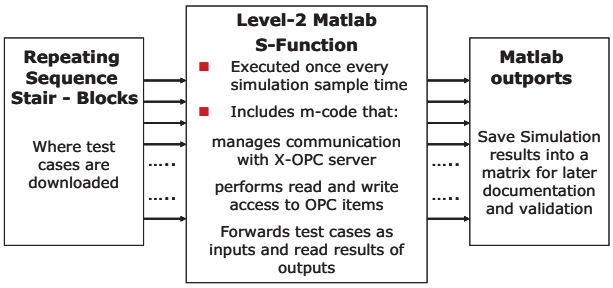


Fig. 2. Overall building process of the HiL test environment

The sample time of the **Repeating sequence stair** is set to the same sample time of the model in such a way that each sample time new test iteration is fed into the test model and forwarded to the X-OPC server, which is assumed to be configured before.

After adding and connecting the **Repeating sequence stair** blocks with the appropriate input pins of the **Level Matlab S-Function**, outport blocks are going to be added to the test model and their names and data types will be changed according to the outputs in the main developed model. Finally each added outport block is going to be connected automatically with the appropriate output pins of the **Level-2 Matlab S-Function** block. It is important to remind once again that all previously mentioned steps are really performed automatically. The user only has to start the HiL test in order to run all these steps. An example of an automatically generated HiL test environment from a Simulink® Model consisting of 4 inputs and 2 outputs is shown in Fig. 3.

B. SILworX® Side

After a specific application has been implemented in embedded C- code using the automatic code generator a function block (so called CIF- C Interface Function block type) has to be created manually in the SILworX environment. The input and output pins of this block will be imported from a file, which is supposed to be generated automatically during code model conversion. Furthermore this block will be attached to the auto-generated code also through its import option.

It should be also noticed that during code generation a file with global variables will be generated. The global variables become the same names as the inport and outport blocks of the main model and two additional control global variables are declared: **EnableIn** and **EnableOut**. These global variables are used for data interchange with the OPC DA server.

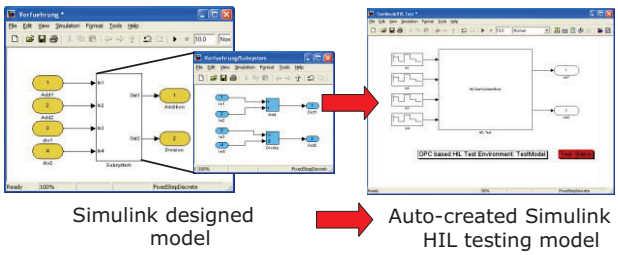


Fig. 3. Auto generation of Simulink® HiL test bench model from original designed Simulink® model

The **EnableOut** variable is not that important, since the **EnableIn** variable is controlled from the software that has been attached to the **Level-2 Matlab S-Function** block and is used as a control signal to activate and disable the CIF block. The **EnableIn** and **EnableOut** will be connected to the **EN** input pin and **ENO** output pin, which appear automatically when the user chooses the option **Show EN/ENO** from the context menu of the CIF function block in the program. The HiL test environment in SILworX® for the upper model is depicted in Fig. 4.

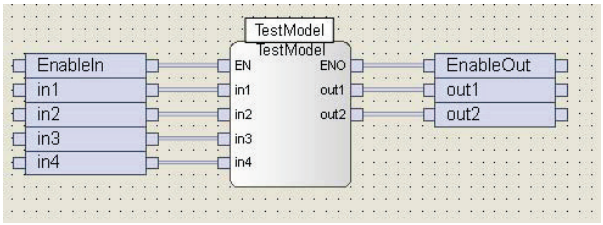


Fig. 4. HiL test model representation on SILworX® side

C. How does it function

After having finished configuring an X-OPC DA interface for the built SILworX® test program, code will be generated on SILworX® for both the program and the created OPC interface. The code generated from the program is uploaded on the PLC and the one generated from the configured OPC interface is read in the X-OPC server. Afterwards both applications are started. When those steps are achieved, hardware simulation can be started from the HiL test model generated in Simulink®. To get a better overview how the data transfer of the whole arrangement works, take a look at Fig. 5.

During simulation or runtime the **Level-2 Matlab S-Function** implemented in the Simulink® test model executes exactly once every tact or sample time. Within one execution the following steps are sequentially performed:

- The values of the outputs (SILworX® model) or the values of the OPC readable items determined by the server through read access will be given to the approriate outputs in the Simulink® model and saved into a matrix for later documentation and validation purposes.

- After finishing the read process or determining the values of outputs a FALSE value is going to be written into the **EnableIn** control signal, in order to deactivate the code execution and to begin a new write process into the SILworX® model inputs. Each new tact, the inputs are updated and a new test value lies at each input, which is going to be written to the appropriate item on the OPC server and forwarded to the hardware, i.e. a new test iteration is sent to the hardware.

- After writing all model inputs values into the appropriate OPC writable items, the **EnableIn** control signal is immediately set to TRUE per software, in such a way that the CIF – funktion block is enabled and code is executed once with the forwarded test values.

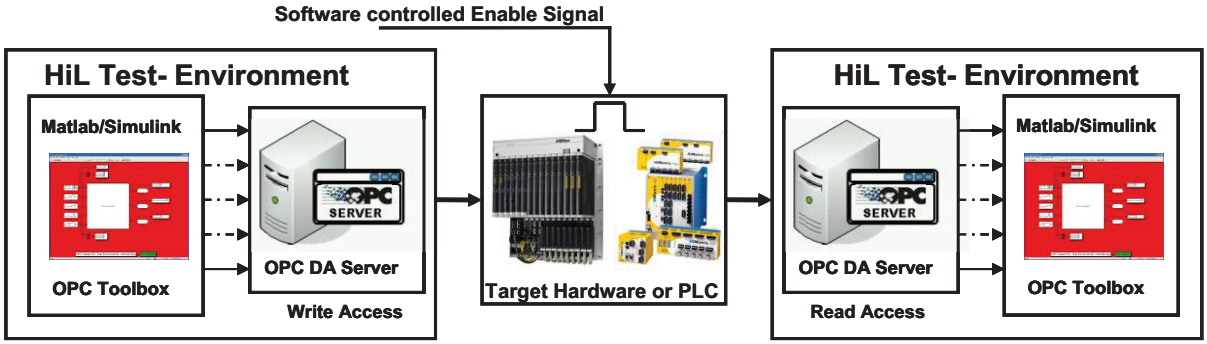


Fig. 5. Data interchange between the Simulink HiL test envionment and target hardware

As mentioned before every new sample time begins with a read process followed by a write access to the OPC writable items. Since at the beginning of the simulation no data are needed to be read at the outputs because no test values have been written to the inputs yet, solely a write access will be performed in the first tact and the appropriate execution results will be determined or read during the following sample time.

This is the reason why the total simulation time has been extended for one additional sample time, in which the results of the last write process will be read. This makes the total simulation time dependent from the number of iterations (one sample time per iteration) intended to test plus one sample time for the last read access.

The test results or the values of the outputs obtained for each test iteration will be saved each sample time into a matrix that will be used at the end of the test to compare results with the results of the previously supposed successfully performed MiL test and in the test documentation as described in the next session.

5. HIL Test Tool - Performance Evaluation

In this section, a performance evaluation of the tool is made by means of a simple example consisting of an AND- and XOR-block (see Fig. 3). The Simulink® model consists of 4 inputs with Boolean data types and 2 Boolean outputs and has a sample time of 0.2. That means in this case a total of total 2^4 =16 different test iterations are built and saved into a test matrix. It is also important to notice at this point that a special tool for automatic test case generation has been developed as a separated instrument, where different methods are followed.

After building the HiL test environments on both sides Simulink® and SILworX® and configuring the required OPC interface using the HIMA X-OPC server the simulation can be started.

If the user wants to show or follow the events taking place on the target hardware during runtime, the user has to switch to the online simulation mode in SILworX®. Some screenshots have been taken during the online Simulation of the model and are shown in Fig.6. The red colored lines represent the Boolean state TRUE while the blue ones represent the Boolean state FALSE.

As mentioned previously the inputs become each sample time new test values, which will be written to writable OPC items and forwarded to the appropriate global input variable on SILworX® side through accessing the address space of the PLC.

Every time the HiL Simulation is achieved, an html test report will be automatically generated and saved in the current Matlab® working directory. This report includes the results of the previously achieved MiL test and the results of the HiL Test that will be compared. It is also important to notice that both tests have been performed with the same test cases and if both test results match or the difference between the MiL and the HiL values for the same test vector lie within a predefined tolerance limit, the HiL test is then deemed to be achieved successfully and the converted code seems to be working as expected during real time HiL simulation.

In the documentation all passed comparisons, i.e. identical MiL and HiL test results for the same test vector are marked with ✅ symbols while failed tests are marked with ❌ symbols. Fig. 7 shows a part of the auto generated html test report for the treated example.

The test report shows that all tests are passed and no failed tests have been recognized and hence the test has been successfully accomplished. Furthermore the test was well performing and has been achieved in a very short time despite the small delays, which have been inserted between the individual tests in order to allow the user to see and follow the events taking place on hardware side when switching into online mode, because the process is really running very fast.

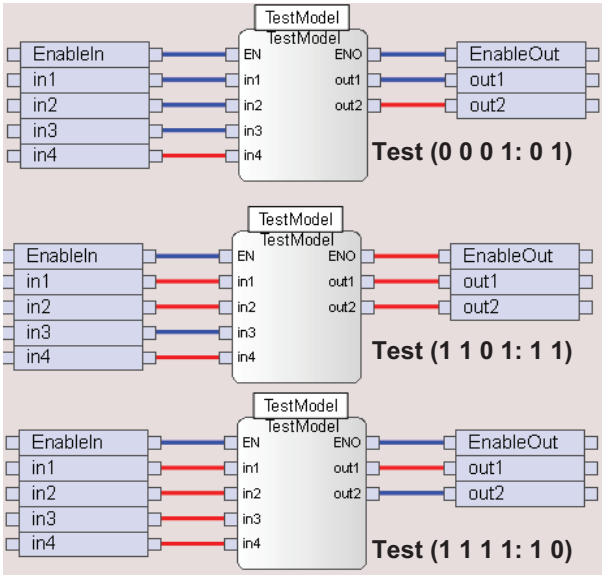
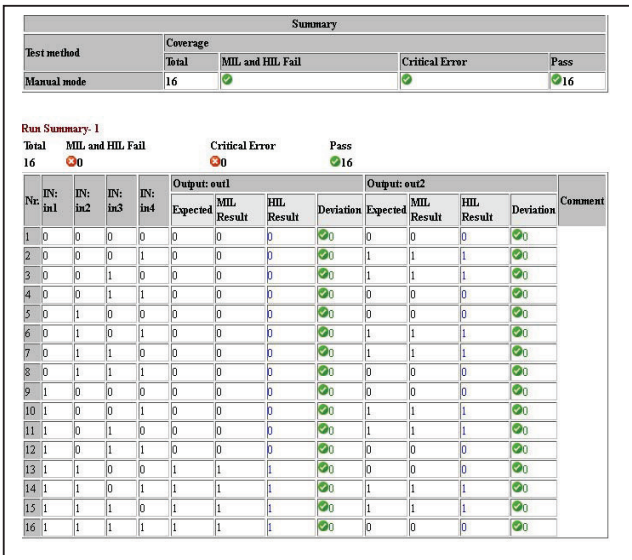


Fig. 6. Some snapshots of the online simulation mode while HiL test running



Summary

Test method	Coverage			
	Total	MIL and HIL Fail	Critical Error	Pass
Manual mode	16	✓	✓	✓16

Run Summary: 1

Total	MIL and HIL Fail	Critical Error	Pass
16	✗0	✗0	✓16

Nr.	IN: in1	IN: in2	IN: in3	IN: in4	Output: out1				Output: out2				Comment
					Expected	MIL Result	HIL Result	Deviation	Expected	MIL Result	HIL Result	Deviation	
1	0	0	0	0	0	0	0	✓0	0	0	0	✓0	
2	0	0	0	1	0	0	0	✓0	1	1	1	✓0	
3	0	0	1	0	0	0	0	✓0	1	1	1	✓0	
4	0	0	1	1	0	0	0	✓0	0	0	0	✓0	
5	0	1	0	0	0	0	0	✓0	0	0	0	✓0	
6	0	1	0	1	0	0	0	✓0	1	1	1	✓0	
7	0	1	1	0	0	0	0	✓0	1	1	1	✓0	
8	0	1	1	1	0	0	0	✓0	0	0	0	✓0	
9	1	0	0	0	0	0	0	✓0	0	0	0	✓0	
10	1	0	0	1	0	0	0	✓0	1	1	1	✓0	
11	1	0	1	0	0	0	0	✓0	1	1	1	✓0	
12	1	0	1	1	0	0	0	✓0	0	0	0	✓0	
13	1	1	0	0	1	1	1	✓0	0	0	0	✓0	
14	1	1	0	1	1	1	1	✓0	1	1	1	✓0	
15	1	1	1	0	1	1	1	✓0	1	1	1	✓0	
16	1	1	1	1	1	1	1	✓0	0	0	0	✓0	

Fig. 7. Automatically generated html evaluation report containing both MiL and HiL results

For more performance, behaviour and strain test, complex models with digital filters have been tried using the tool. Some hundreds and sometimes even thousands of test cases have been generated for this purpose. The tool was well performing and showed a stable behaviour but as expected the test was taking longer due to the numerous numbers of tests generated.

6. Conclusion

The HiL test bench discussed in this work represents a very effective and time saving environment for testing and simulating the functionality and behaviour of auto-generated software in runtime mode, which has to be performed as one of the last steps within the life cycle of a MBD project in order to reach a validation in a very short time and therefore to obtain a product in an acceptable TTM interval.

The initial purpose of the tool was to offer to development engineers in model based software development field an easy to use, automated HiL test instrument for validation intentions and particularly for devices and target hardware that conform to the OPC DA communication standard (DCS, PLC) using the Matlab®/Simulink® engineering tool as development environment. The tool delivered a very good performance even with a huge number of test cases and was performing very fast and in a stable manner.

7. Future Works

In the near future we will set the major focus of work on enhancing and increasing the degree of automation of the tool as much as possible especially on the side of SILworX® because in the recent version the biggest part of configurations and actions on SILworX® side are still performed manually contrary to the Simulink® side where user only need to touch one button, so that almost all activities and building procedures are achieved automatically and a high degree of automation is reached.

8. References

[1] Achraf Saad, Keshac Dahal, Muhammad Sarfraz, Raijkumar Roy, "Soft Computing in industrial Applications", Recent Trends, 2007, Springer Verlag.

[2] http://www.srmtech.com/model-based-development/model-based-development.htm

[3] B. Hailpern, P. Santhanam, "Software debugging, testing, and verification", IBM SYSTEMS JOURNAL, VOL 41, NO 1, 2002.

[4] J. Börcsök, "Electronic safety concepts, models and calculations", Hüthig Verlag Heidelberg.

[5] Günter Hommel and Sheng Huanye(Eds.), "Embedded Systems Modeling, Technology, and Applications", Proceedings of the 7th International Workshop held at Technische Universität Berlin, June 26/27, 2006, Springer Verlag.

[6] Devesh Bhatt, Brendan Hall, Samar Dajani-Brown, Steve Hickman, Michael Paulitsch, "Model-Based Development and the Implications to Design Assurance and Certification", Honeywell International, Minneapolis, Minnesota, IEEE 2005

[7] http://www.hima.com

[8] http://www.mathworks.com/products/simulink/?BB=1

[9] http://www.mathworks.com/products/opc/

[10] http://www.hima.com/Products/SILworX_default.php