Standards Compliant HIL Bench Development for Dynamic Skip Fire Feature Validation

2015-01-0171
Published 04/14/2015

Paul Liu, Abhijit Bansal, and James C. McKeever
Tula Technology

CITATION: Liu, P., Bansal, A., and McKeever, J., "Standards Compliant HIL Bench Development for Dynamic Skip Fire Feature Validation," SAE Technical Paper 2015-01-0171, 2015, doi:10.4271/2015-01-0171.

Abstract

Automated software testing for both hardware and software components is one of the ways industry is gaining efficiency in testing. A standard based approach can help in reducing the dependency on one particular tool chain, reduce re-training of engineers, reducing development time and increase collaboration between supplier and OEM's.

Tula's Dynamic Skip Fire (**DSF**) technology achieves fuel efficiency by activating only the required cylinders required to achieve desired torque. Validation of the DSF algorithms requires reading of the crank, cam, spark, fuel injector, and intake and exhaust actuator positions on an individual cylinder firing opportunity. Decisions made on a cylinder by cylinder basis can be validated.

The testing architecture at its core is based on the ASAM Hardware in the loop (HIL) API standard. Following the HIL-API standard gives the flexibility of choosing the best in class measurement hardware and test case management tools.

Since the test environment is used by multiple engineers from multiple locations, the entire testing environment can be controlled remotely from a powerful web client - referred to as WebCarLab. The web client allows the user to remotely queue different automated tests and also to queue software builds. This allows for up to 100% test bench utilization.

This paper discusses the main advantages of following an industry standard while designing the test architecture, which allows the different components of the test architecture environment to be integrated.

Introduction

In the DSF system, as described in Wilcutts et al (2012) and Serrano, et al (2013), cylinders of an internal combustion engine are selectively fired, in such a way to provide driver requested torque at optimum fuel efficiency while minimizing noise and vibration issues.

Any HIL environment used to test an engine control system which uses selective firing must be able to verify each individual firing and skipping event, even under dynamic engine operation scenarios.

The traditional way of solving this optimization problem is to develop a basic calibration set and turn over this dataset to experienced calibrators who will fine-tune the system to match the needs of the automotive application in a production environment. This paper describes an ECU testing environment that uses the open HIL ASAM standard to interconnect best-in-class components in order to assemble the optimum test system which decreases cycle times to verify correct operation of DSF algorithms quickly and accurately.

Safety Critical Requirements

In order to be ISO26262 compliant, the product development process integrates the traditional V product development cycle with heavy emphasis on automation. For instance, Model in the loop (MIL) testing of simulation models is automated using MathWorks tools such as Design Verifier and Matlab scripts. Software in the loop (SIL) testing of auto-generated compiled code is done on dedicated continuous integration servers running Jenkins and Python prior to testing on the real target. Once MIL and SIL verification has been completed, HIL testing begins.

Since nearly all of the tests are automated, test failures are logged automatically in an issue tracking system. Each bug can then be tracked to final closure resulting in full visibility of issue resolution.

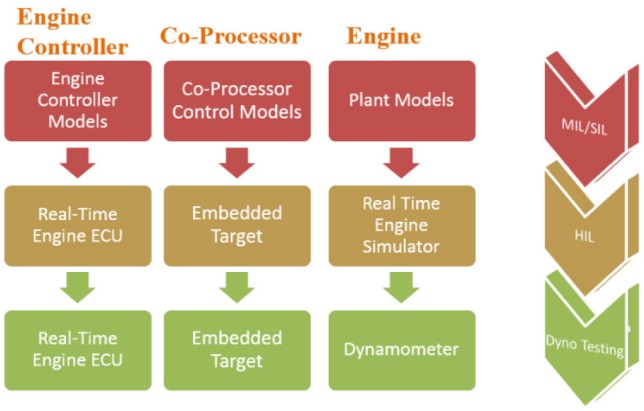

Figure 1. Test Architecture

This system can be further expanded to the engine dynamometers. The dynamometers use dSPACE and INCA tools which are supported by the ASAM HIL API. Automation of dynamometer environment configuration, test input profiles, calibration, logging, data archiving, issue tracking and data post processing is possible.

Automated HIL Testing

Writing effective and efficient tests is a key objective when automating HIL testing. Automated software testing for both hardware and the software is one of the ways industry is trying to gain efficiency in testing. Automated testing has the following benefits over conventional manual testing:

1. Standard and reproducible tests: Standardizing the test procedure takes out the human element from the tests.
2. Time effective: Tests take only a fraction of the manual test time and saves man-hours as the user is not required to manually run the tests.
3. Regression testing: Automated tests can be run overnight and over the weekend without the presence of the user.
4. Test management: Automated tests can be easily synced with the test requirements and are easily tracked.

Hardware Setup

The company relies on the synchronization of an NI Veristand test environment with a dSPACE MicroAutobox (figure 2). The NI Veristand simulates the engine whereas the dSPACE MABX acts as the engine ECU. The dSPACE MABX is supplemented by the dSPACE RapidPro hardware. Tula also employs a co-processor that communicates to the simulated ECU through either a UDP or SPI bus protocol. Since there is a co-processor as part of the solution, the traditional approach of developing a customized test environment would be expensive and take time to implement. The ideal solution to the building of the test setup would be to get the best of automotive HIL solution combined with the best in silicon validation measurement and testing equipment. After extensive research, Tula identified and built its first HIL bench using National Instruments and dSPACE products because it offered the flexibility and the expandability needed for developing and validating the company's products and services at a cost effective price. One of the key reasons for the decision was the development of the HIL-API for dSPACE and National Instruments products which facilitated interoperability while maintaining flexibility at a reasonable cost.

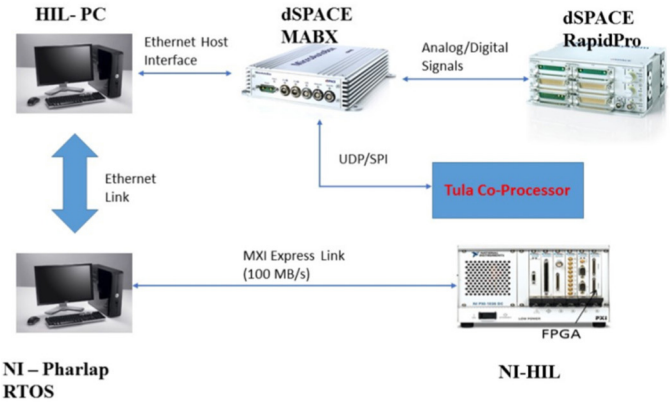

Figure 2. Test HIL Hardware

Automated Test Architecture

As shown in figure 2 the automated test setup employs hardware from different vendors in the current HIL setup. Each vendor has its own proprietary platform access API. Trying to integrate different proprietary methodologies presents a range of problems. The focus of the test engineer must be more on validating the system under test and not on struggling with HIL vendor hardware and software integration. Thus there is a need for a common platform access API that meets the following demands:

1. Synchronization between the different hardware from different vendors.
2. Standardized functions for platform access.
3. Fast execution of commands.
4. Stimulus testing
5. Triggered data capturing to synchronize and efficient recording.

After careful research and investigation the company selected the ASAM HIL - API.

HIL - API: Open Connectivity through Standard Compliance

HIL-API is a standard proposed by the automotive industry. ASAM HIL-API defines a standard interface for accessing Hardware-In-the-Loop systems. The standard has been created by German OEM's, TIER-1 suppliers and the tool vendors within the ASAM committee. The aim of the standard is to standardize access to different simulation systems. This increases test reuse, helping to protect investments and reduce development costs and time as it decouples the hardware from the software.

HIL-API acts as an intermediary layer that helps the flow of data to and from the hardware. Any software/hardware that is compatible with this standard can be used with any other hardware/software that supports HIL-API. Figure 3 shows how, with the help of HIL-API, the hardware can be decoupled from proprietary software.

Because these standards help multiple test systems to interoperate, the need to write tests using proprietary protocols is reduced to a minimum. Using a single interface also reduces the complexity associated with maintaining and tracking various tests scripts and associated test hardware configurations.

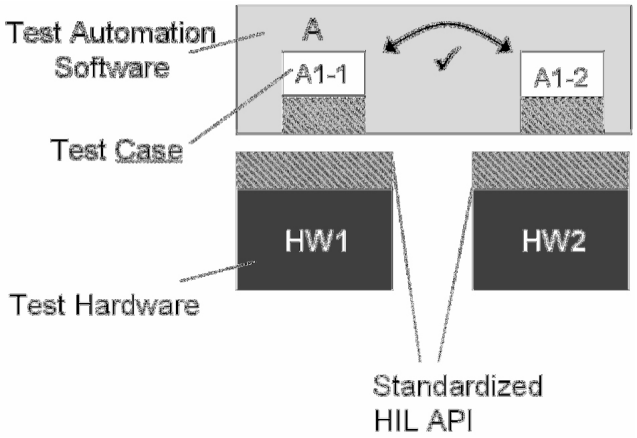

Figure 3. HIL- API

Why Chose HIL-API?

HIL-API meets the testing requirements while achieving a lot of flexibility

1. Response time: The average response time for read and write is less than 1ms, this helps in executing tests at near model sample rate.
2. Stimulus testing: It is reliable, accurate and completely scriptable. A particular example of doing this is during calibration and verification of the engine speed at which the co-processor turns DSF on. One can write an automated test that creates a new engine speed stimulus in a loop and verifies that the co-processor is running DSF.
3. Data Capture: The data capturing is accurate with no missing data points even at rates higher than 2 samples per millisecond. This is particularly important when some of the trigger pulses for the actuators are just one sample wide. Missing that one data point can be very costly.
4. Triggered Capturing: HIL-API also supports triggered capturing. The trigger can be a very complex condition, for example one can sequence conditions, so the second condition is only checked after the first condition becomes true. *Syntax: Check this &> Then check this*.

   Triggers are very important as they help synchronize the capturing of data for both NI and dSPACE hardware. The setup employs a digital signal, which acts as a trigger for both dSPACE and NI hardware as shown in figure 4. With this use of a hardware trigger, data capturing was synchronized to within 1 sample point.

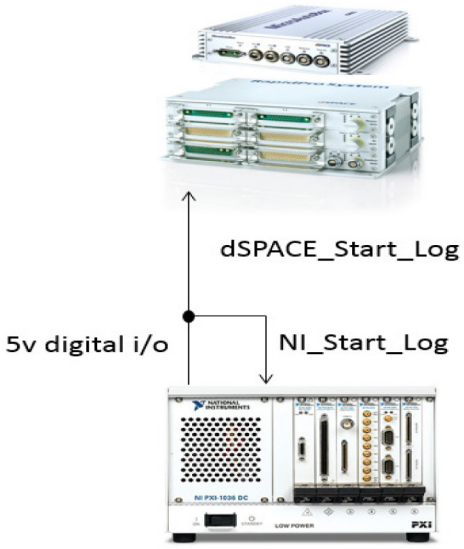

Figure 4. Hardware trigger for syncing

5. ASAM Standard: HIL-API is an ASAM standard which is defined in both Python and C#. HIL API standardizes the platform access. This results in high portability of tests. One of the tests involves verification that a fuel injection pulse is always followed by an ignition event within certain degrees of engine rotation. This is a complex test that involves a complex algorithm (Figure 5). There are many such tests which are even more critical. With HIL API, the company knows that the time invested in writing these complex tests is not wasted as the tests can be easily ported to different hardware with minimal effort. This will also help the company to share its tests with its customers regardless of the hardware they are using as long as they use hardware that follows the HIL-API standard.

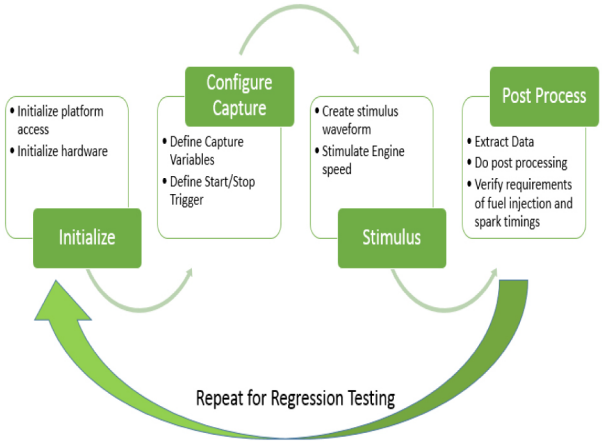

Figure 5. Engine Actuator Test

6. Object Oriented: The design of the HIL API standard is highly object oriented, thus it is easy to use and completely scriptable. It is also customizable and robust on different platforms. Being platform independent and object oriented, the HILAPI has a drawback that the objects need to be properly initialized and handled. This can sometimes result in some extra steps. For example, in order to do a simple write to a variable the steps given in figure 6 need to be followed. However, with the use of custom libraries, a simple method can be created as shown in the figure 7. Now just by specifying the hardware being

used we can use the same block for any hardware. There are many more places that this concept of encapsulation is used, for example, to configure capture or stimulus etc. This helps the company to create a set of custom libraries/wrapper to simplify and speed up development.

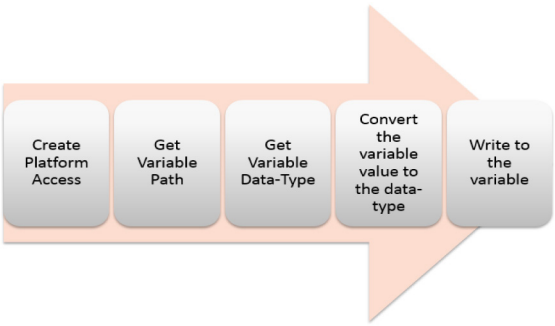

Figure 6. How to write using HIL-API

The test engineer's investment in writing scripts that use HILAPI is protected as scripts can be ported with minimal effort to various HIL environments that avail of the HIL API. With these points in mind, the test team incorporated the HILAPI into its test environment.

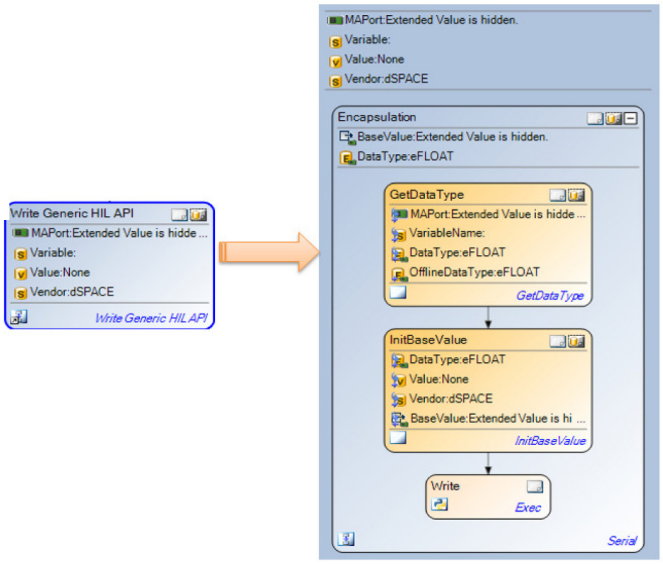

Figure 7. Automation Desk HIL-API Write library block

HIL-API Test Setup

The basic software test architecture involves the use of HIL API with dSPACE AutomationDesk

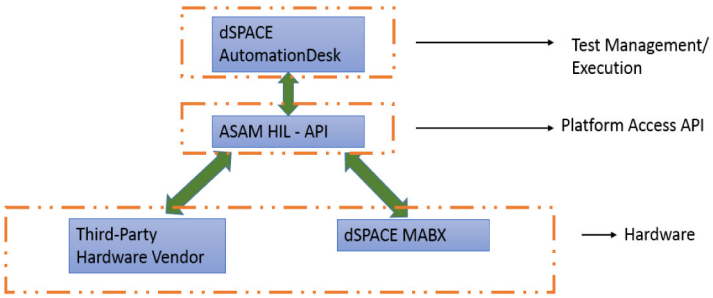

Figure 8. Test set-up

The test setup uses a combination of HIL-API and dSPACE AutomationDesk for the system testing. AutomationDesk is a test creation and test execution software. The reasons for the selection are:

1. Support for HIL API, thus supporting integration of hardware from different vendors, such as NI and dSPACE hardware support HIL API.
2. Supports creation of custom libraries or blocks,
3. Easy and effective test creations,
4. It supports a combination of script and graphic based test creation.
5. Control over the report generation
6. API access to the software to enable integration with WebCarLab.
7. Stable and Reliable testing.

WebCarLab

To make automated testing available to a wider set of engineers and provide remote access to running the tests, the company has developed and uses a proprietary software called WebCarLab.

In embedded controls development access to a hardware in the loop (HIL) testing environment can be a resource bottle neck. These HIL systems are in demand from both testing and development departments. There is therefore a need to bring HIL bench utilization to 100%.

It is common for testing to be automated via test scripts. A test harness is often created around a unit under test and test scripts automatically ran to exercise inputs and check for expected system functionality.

The automation of test scripts alone does not go far enough. Ideally the entire software build process, testing and results archiving should be automated. Retrieval of test results and data/log files is important for future reference and may be used for auditing purposes.

Version control of software and test scripts is a challenge. It is important for the test engineer to be aware of both the test script and software versions. This ensures bugs are easily traceable and reproducible at a later date

Often in software development, multiple platforms are supported by a common code mainline. A build can be configured manually or by use of a script to support multiple platforms or customers. Such a build configuration process can be cumbersome and prone to human error.

Engineers often work in multiple site locations ensuring close interaction with customers, suppliers and other process partners. In person access to a HIL environment is not always possible at all locations. Commonly “Live” remote access to a HIL environment can be achieved by using a VNC client. This allows the test engineer access for limited manual and automated testing from a remote location. Even with a live VNC client, there is a risk that the remote test engineer could interfere with tests already in progress on the HIL environment. It is important that a schedule is communicated and time for each engineer allocated. Such a methodology is difficult to control and to administer.

These problems were common daily obstacles encountered by the development and test teams. A fully custom in house solution has been specified, designed and created called WebCarLab which address all of the above points. WebCarLab is written in Python Django and runs on in house Linux Servers. It provides the user with 24hr access to all HIL bench systems, version controlled test scripts and source code.

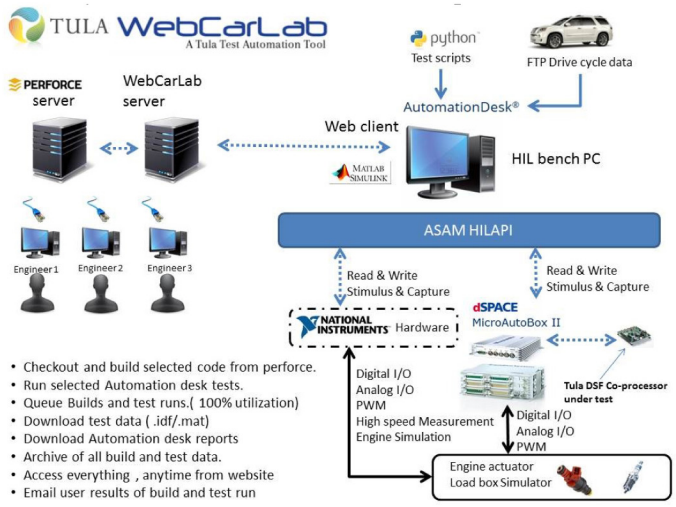

Figure 9. The automated test environment

Test scripts and software builds are checked into the version control tool (Perforce) by the test engineer or developer. WebCarLab can check out these files from the Perforce server automatically by using a Perforce Python API.

Each HIL environment has a unique hardware topology file which is needed during the build process. The Hardware topology file makes the connection between the Simulink software under test and the dSPACE rapid-pro hardware. During a manual build process the user selects the HWT file in Simulink before performing the build. The HWT file is normally stored manually on the HIL environment PC.

In order to automate the build process a method was needed to configure the Simulink build with the correct hardware topology file via the WebCarLab interface.

The solution comprised of a Matlab m-script which replaces the current Simulink topology settings with a preconfigured topology file block based on the HIL platform selected by the user.

For example, if HIL1 is selected by the user then WebCarLab will check out the Simulink code from Perforce and then replace all topology file blocks in the Simulink code with the pre-configured topology block stored on the server. The complete process is fully automated and invisible to the end user.

The company's engine control software supports multiple vehicles, customer and test environment platforms. In order to perform the variant builds reliably, the user can use WebCarLab to configure the build for the particular variant required. The user selection triggers WebCarLab to run test scripts from Perforce which configure the build for the platform/Customer in question. The platform build executable and source files are archived in the database for future reference. Pre-build configuration test scripts can be modified and added to WebCarLab as needed to support future projects.

WebCarLab is accessed by the user via a web browser. Engineers can now log into a website on or off location and schedule tests on the Perforce software version of their choosing. Deploying such a tool means engineers now do not need to worry about the availability of the HIL bench environment. Tests and software builds are queued and executed in order on a first come first serve basis. Users can choose to run their test and build job on the HIL environment of their choice. The list of queued jobs is visible for each environment via the web browser user interface.

Engineers can schedule a build and test job and walk away effectively performing “lights out” testing of their code.

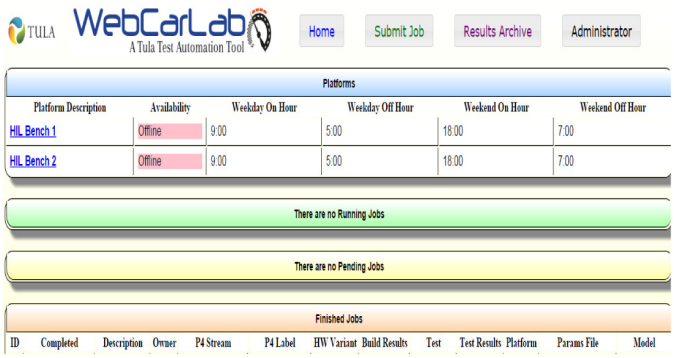

Figure 10. The Tula WebCarLab web browser interface

The test logs, builds and other files which are created by the job are stored on the WebCarLab Linux server and are easily available for download at a later date.

WebCarLab uses a Linux server to store the files and the web interface can download the files selected by the user via an SQL database. All of this is back end process is invisible the end user.

The WebCarLab browser has a built in VNC client to allow the user to view “Live” HIL bench behavior or to perform manual tests remotely. Automated Test scripts can be selected by the user from a pool of released tests stored on the Perforce server. Having a pool of released tests available to all engineers via a web interface allows developers to perform more regression and means less support is needed from test engineers during integration/debug of issues. This leaves test engineers more time to write new tests and focus on tasks such as code coverage.

The quality metrics monitored to determine success of the WebCarLab project were: Bench utilization, test team support needed during development/integration and finally number of bugs found during integration before initial release. All quality metrics showed a marked improvement after the implementation of the WebCarLab automation tool

Summary/Conclusions

By utilizing an industry standard an automated hardware in the loop testing environment was created enabling Tula's Dynamic Skip Fire (DSF) to be fully tested. Best in class hardware and software from different vendors were integrated into one HIL platform using the HILAPI. A test harness was created around the system under test

using both dSPACE and NI vendor equipment. The HILAPI allowed the test engineers easy access to both the NI and dSPACE environments via test scripts.

Automation of the entire build, test and data archiving process was achieved using an in house web HIL Interface.

Test bench utilization was increased from 40% to 90% with engineers at various locations now fully availing of the HIL remote access features.

The number of software bugs found pre-release doubled with more developer testing allowing resolution of issues before software release candidates were created.

Further gains are possible by transferring the automation techniques discussed to the dyno test environment.

References

1. Wilcutts, M., Switkes, J., Shost, M., and Tripathi, A., "Design and Benefits of Dynamic Skip Fire Strategies for Cylinder Deactivated Engines," *SAE Int. J. Engines* 6(1):278-288, 2013, doi:10.4271/2013-01-0359.
2. Serrano, J., Routledge, G., Lo, N., Shost, M. et al., "Methods of Evaluating and Mitigating NVH when Operating an Engine in Dynamic Skip Fire," *SAE Int. J. Engines* 7(3):1489-1501, 2014, doi:10.4271/2014-01-1675.
3. Bansal A., Muli M., Patil K. "Taming Complexity While Gaining Efficiency: Requirements for the Next Generation of Test Automation Tools", IEEE 2013
4. Krisp, H., Lamberg, K., and Leinfellner, R., "Automated Real-Time Testing of Electronic Control Units," SAE Technical Paper 2007-01-0504, 2007, doi:10.4271/2007-01-0504.

Definitions/Abbreviations

NI - National Instruments

DSF - Dynamic Skip Fire

The Engineering Meetings Board has approved this paper for publication. It has successfully completed SAE's peer review process under the supervision of the session organizer. The process requires a minimum of three (3) reviews by industry experts.

ISSN 0148-7191

http://papers.sae.org/2015-01-0171