

Virtual Validation Cloud : An Electronic System Validation Platform Based on Cloud Computing Resources

Yasuhiro Ito, Yasuo Sugure and Shigeru Oho
Hitachi Ltd

ABSTRACT

This paper describes a Virtual Validation Cloud (V²Cloud), a new computation platform designed for extensive failure mode and effect analysis (FMEA) of an automotive electronic control system. Recently, automotive developers have been requested to execute enormous amounts of system-level tests in order to assure functional safety of electric control systems. Moreover, the upcoming industry standard, ISO26262, demands that system-level validation assures extensive sets of test vectors and clarifies results of the tests as evidence of functional safety. It is unrealistic to execute these system-level tests by only using conventional hardware in the loop simulation (HILS).

Virtual HILS (VHILS), a system-level validation environment for automotive systems based on co-simulation of software and mechatronics, has already been proposed. Virtualizing an automotive electronic control systems makes its testing environment more software-centric and independent of actual hardware. The VHILS can insert test vectors, suspend and resume execution of simulation it and reproduce simulation status.

A V²Cloud system has been developed that automates these VHILS functions on a scalable computing resources. The V²Cloud not only executes VHILS in parallel, but also abstracts the differences of types of simulation, structure of target systems and configuration of computers in order to hide them from developers and enable them to form an extensive set of test-vectors.

The V²Cloud system was applied to a safety validation process of an actual automotive control system. The system was prototyped on multiple types of cloud-computing resources. Moreover the results of cost statistics said that V²Cloud can reduce simulation turn-around-time enough to complete validation extensively.

CITATION: Ito, Y., Sugure, Y. and Oho, S., "Virtual Validation Cloud : An Electronic System Validation Platform Based on Cloud Computing Resources," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.* 5(1):2012, doi:10.4271/2012-01-0501.

INTRODUCTION

Recently, the main issue in developing automotive products has been how to assure their safety. There are various kinds of faults, such as incorrectness of software implementation, inherency of system-specifications, defects of mechanical components, even process variation of semiconductors. However all these faults rarely affect the behavior critically, since they are caught and tolerated by software components for fail-safes that run on electric control units(ECUs).

Generally, fail-safe functions are validated by using actual components with test-vectors considering single-point faults. However, as target systems become more and more complicated, combining of multiple components connected asynchronously causes faults that the current approach cannot catch completely. For this reason, future fail-safe processes must cope with multiple and accumulative faults.

To satisfy this requirement, developers face two problems: difficulties in observing and emulating fault phenomena and an exploding number of fault combinations. The first problem is now becoming critical in developing ECUs. Currently multiple functions of control are integrated into a single MCU in ECUs and this integration makes it difficult for developers to observe software and electrical behavior inside MCUs. For this reason, generally developers have generally used specific instruments for observing and injecting faults, but these instruments make the structures of systems different from that when they are actually used. The second problem becomes critical when the first problem is solved. Usually, there is a limited selection of where to observe and to inject faults because of the disadvantages as it is mentioned in the first problem. When this limitation is overcome, developers can consider much more complex combinations, such as how soft-error in SRAM of MCU propagates to a control of mechanic behavior, than the usual

validation process. The number of combination will explode in an exponentially, thus current actual component based validation such as HILS will become insufficient.

To tackle the first problem, a simulation-based approach was applied. The authors proposed Virtual HILS(VHILS), a whole-system level validation method based on integrated multi-physics simulation. [5] VHILS emulates the system-level behavior of automotive systems that includes multiple ECUs, networks and mechanical systems by using no actual components. Moreover, its operating speed is almost the same as that of the target automotive systems. Since every behavior in every component in VHILS is modeled by a simulator specific to it, developers can access to every behavior. By adding a way to share progress of test vectors among simulators included in VHILS, VHILS become more capable of executing complicated test vectors that include strict timing-constraints over two or more simulators.

As VHILS acquires greater ability to express test-vectors, throughput of executing simulations should become the next source of bottlenecks. To differ from the second problem, VHILS should run on massive computational resources such as grid- or cloud-computing platforms.

When VHILS runs on those resources, software is needed for controlling multiple simulations, and mapping test vectors for each computers are needed. As previous researchers have proposed augmentation of single physical-layer simulation of analog devices and parallel software validation using operating system-level simulation runs of a cluster system had been proposed [3, 4].

The proposed approach in this paper, V²Cloud System, realizes system-level FMEA that integrates multiple simulations with universal simulation controllers and an acceleration technique using VM technology. A group of software in this system is implemented as libraries that can be used via the Internet to enable developers to use multiple types of computers. V²Cloud was prototyped and ran on several cluster-computing systems, and its performance and operating cost were evaluated.

The rest of this paper introduces common features of Virtual HILS, implementation of V²Cloud System's software: Simulation Abstraction Layer, VHILS unified Controller and performance of the system using multiple types of computer resources.

VIRTUAL HARDWARE IN THE LOOP SIMULATION

VHILS had been proposed for validating system-level ECU software without using any actual components [5]. This simulation based validation system processes test-vector with affordable speed, since it is focused on observing behavior of embedded software. However, models included in VHILS are not limited to software components, but VHILS emulates all the behavior of system components such as mechanical parts, and digital or analog circuits, the networks of which affect the target software. This implementation results in

reconciling the high speed and high fidelity of multi-physics simulation.

VHILS FOR ECU LEVEL-TESTING

Primarily, the usage of VHILS has been used to replace of HILS instruments, which are now used broadly for validating software behavior of ECUs. While HILS needs actual ECU components, developers had instrumental cost problems when they wanted to reinforce their validation throughput by increasing sets of the validation components. The aim of VHILS was to overcome limitations of using actual components by replacing everything with simulations.

The previous implemented of VHILS consisted of three components: ECU, network(CAN) and machines(vehicle, ambient environments). Behaviors of the ECU and network were modeled by an instruction set simulator(ISS), and machines were modeled by a multi-physics simulator such as MATLAB/Simulink[®]. ISS was selected for emulating ECU behavior in order to use software in a binary format that is exactly the same as that integrated in actual ECU. All the simulators included in VHILS were connected by a simulation interface that optimized their communication packet size and frequency for transferring virtual CAN packet. Developers could observe how the software in ECU worked by changing condition variables processed in vehicle or ambient environment models.

By shifting the border of models run on simulators from the original border of components, VHILS gained simulation higher speed and accuracy than the previous research. Some parts in environment and vehicle models that had sensitive timing consistency were shifted to the ISS and Digital simulations which supported cycle-level accuracy. This resulted in reducing frequency for synchronizing each simulator, which effectively improves VHILS performance.

VHILS was applied to the actual validation of a vehicle dynamics controller and produced more accurate system reactions than the original product. Moreover, its simulation speed was one-third that of the original environment, although that environment used ISS that emulated the behavior of target CPU's instruction one-by-one.

COMMON PROCEDURE FOR VHILS'S OPERATION

While VHILS integrates multiple layers of simulations that are executed individually, and can collaborate across the network, VHILS must be controlled as if it were a batch system that integrates multiple executables. [Figure 1](#) shows the example of execution process of VHILS. There must be at least one script controlling VHILS procedure that configures parameters of each simulators. Parameters can be categorized into four kinds on the basis of their usage.

1. Project specific parameters : configuring target specification. (ex.) clock frequency and temperature.

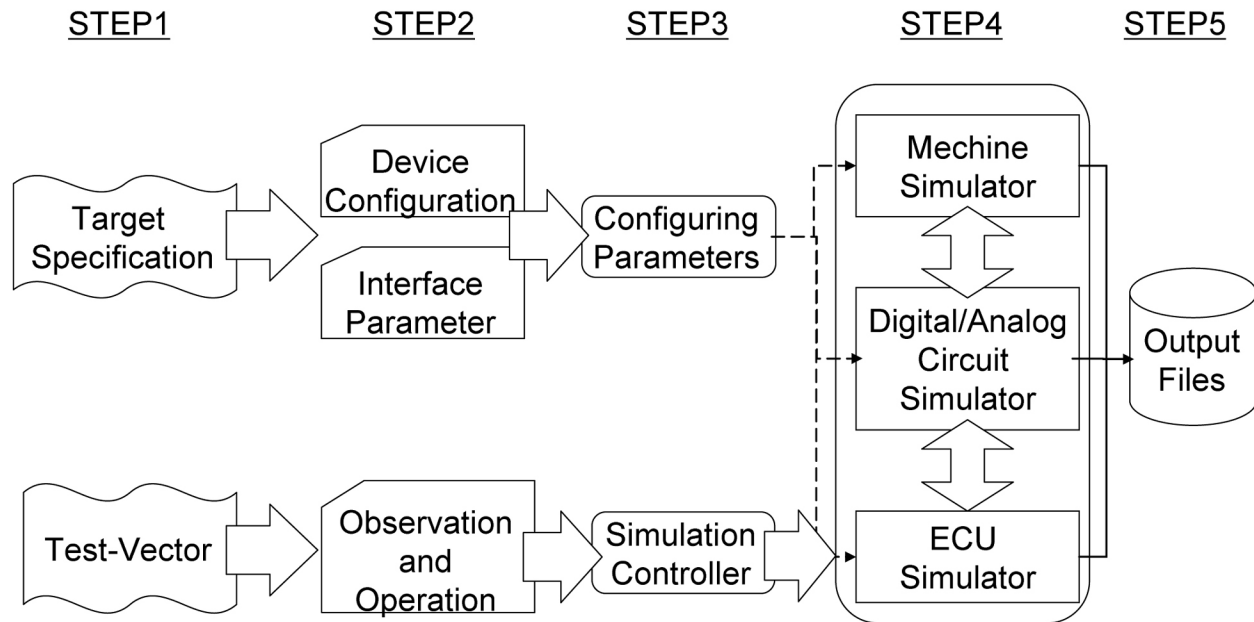


Figure 1. Procedure of launching Virtual HILS.

2. Simulation interface parameters : configuring how to connect with other simulators. (ex.) IP address, port numbers and communication frequencies.

3. Input-file parameters : configuring a link of a file that is referred to a simulation. (ex.) target binary file

4. Output-file parameters : configuring a link of a file that stores result and traced data of VHILS. (ex.) network timestamp file, instruction traces

The script sets these parameters and configures launch sequence in accordance with the connection relationship of simulations.

Additionally, VHILS can accept requests that change conditions of simulation dynamically, from outside of the simulators. Currently these configurations are set manually and individually, though this is inefficient after VHILS increases throughput of executing test-vectors. Thus, automating configuration of VHILS's parameters and connection data resources input from developers are challenges in operating VHILS.

POTENTIAL ABILITIES OF VHILS

The usage of VHILS is not limited to the replacement of HILS. While all the states and conditions are calculated on simulators, developers can access them theoretically. VHILS can access to those variables without any additional instruments, thus developers should be able to acquire dynamic information as if the target ECUs are integrated in products.

By applying this technology, VHILS can execute various validation methods, such as fault injection, and dynamic software analysis, while visualizing these events triggered by other variables across the border of components. However, one technology that integrates methods for observing and

injecting events that completely differ between each simulators is still missing from current research. This is the problem of VHILS.

METHODS FOR ENHANCING VHILS THROUGH-PUT

There are two approaches to improve throughput of VHILS: speeding up single execution of VHILS and executing multiple test vectors in parallel. Simply speeding up a single VHILS has a inevitable problem: timing constraints. Since VHILS emulates whole system-level behavior of target system by cooperating with multiple simulators, this can be treated as a parallel computing program. When there is a shared variable that is updated frequently in a parallel computing system, that variable causes an execution speed bottleneck. Simulators included in VHILS must share timing information each other in order to maintain orders of events emulated on it. Timing information is shared among the simulators and updated frequently, thus it causes the bottleneck of simulation performance. Loosing frequency of sharing timing constraints results in speeding up, however it degrades accuracy of simulation on the other hand. While tuning strategy of this performance trade off strongly depends on target structure, it is difficult to develop one common solution that is suitable for all types of VHILS.

The second approach has larger timing granularity. When VHILS has no components that have randomly calculated behavior, multiple VHILS executions with the same condition act in the same way. Only when they are run with other sets of test-vectors do, they react differently. When developers can distribute multiple test vectors for multiple VHILS environment, then validation performance is enhanced. From this theory, the approach to speed up VHILS

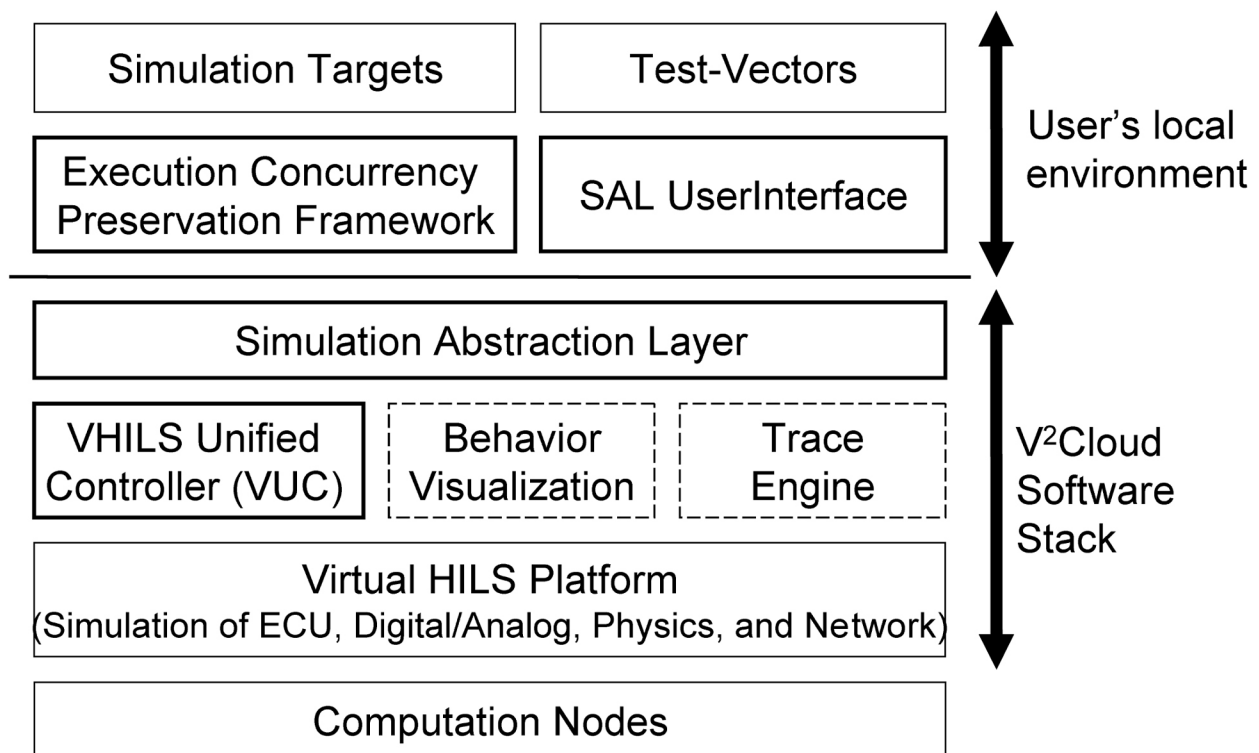


Figure 2. Software layers of V²Cloud System

execution depends on the shape of the test-vectors that developers want to validate.

VIRTUAL VALIDATION CLOUD

The Virtual Validation Cloud(V²Cloud) has been developed to solve the problems stated in the previous chapter. This system enables VHILS to execute system-level validation automatically on large scale computing resources. The system consists of controllers of VHILS, simulation abstraction layers, and a software framework for maintaining consistency of VHILS execution environment.

Figure 2 shows an overview of the layer of V²Cloud software. VHILS unified controller(VUC) executes observation and event injection from outside of simulators. VUC connects simulations to each other and treats all behavior calculated inside simulators in flat, even across borders of simulators. By using VUC users can form test-vectors including severe timing constraint that are difficult to recreate when developers use actual components. A simulation abstraction layer (SAL) formalizes the procedure for executing VHILS, and automates generation, distribution, and execution of test vectors. Execution concurrency preservation framework (ECPF) supports developers to advance their validation environment. ECPF provides developers a protocol that determines the I/O information used by VHILS such as, simulation parameters, kinds of input files, and formats used for outputting results. By using ECPF, developers can immediately start parallel VHILS

execution once they complete single VHILS execution on their own computer resources.

VHILS UNIFIED CONTROLLER

VUC consists of two parts: a central controller script(CCS) that runs independently from simulators included in VHILS, and EventProcessors that are embedded inside simulators. CCS accepts a test-vector input and controls its progress. As shown in Figure3, CCS can connect with multiple EventProcessors via inter-process communication.

Since CCS and EventProcessors are connected by using a protocol that is designed for interfacing test-operation and its results, all simulators can be accessed from CSS in completely the same way. Figure4 shows an example of a VUC protocol. The VUC protocol has a flexible range messaging format, so it is flexible to change under a number of types and arguments of test operations.

When VHILS launches, CCS picks up a test-operation from test-vector, distinguishes which EventProcessor to execute the test-operation, and sends it. The selected EventProcessor receives the test-operation and executes the ordered operation in the simulator, and sends the result and timestamp of the operation back to the CCS when it needs to. By using this procedure, VUC can trigger test-operation across border of simulators, which results in the capability to execute complicated validation.

Developers must develop their EventProcessors when they want to add simulators that they have made by or that are not supported by the system. However, while a common