09CV-0113

EHOOKS – Prototyping is Rapid Again

Vivek Jaikamal
ETAS Inc.

Nigel Tracey
ETAS Ltd.

ABSTRACT

Automotive controls engineers have traditionally used bypass rapid prototyping techniques to quickly try their new ideas before integrating them into the final embedded control unit (ECU) software. In order to make this possible, switches (or "hooks") are required to be inserted in the ECU source code. The development disruption and costs associated with this can be extremely high. ETAS has developed a unique patent pending technology which solves this problem in a reliable way. This technology, to be productized under the name "EHOOKS", allows the controls engineer to quickly add the necessary hooks in the base ECU software without any modifications to the source code. EHOOKS technology, therefore, enables "rapid" software prototyping and testing.

INTRODUCTION

Rapid controls prototyping is an integral part of the mainstream algorithm development process at most major car manufacturers (OEMs) and ECU suppliers (Tier 1s). For the end user (i.e., the algorithm developer) it means major gains in productivity. However, in order for it to be really effective, organizations have to create formal methods and allocate resources to maintain the several moving parts involved. In a lot of cases, companies find it extremely expensive to maintain and support the tools and processes required. As a consequence, algorithm engineers often find themselves unable to leverage the power of rapid prototyping and companies do not realize the return on the heavy investments made in tools and resources.

In the following sections, we will first describe the technical challenges facing conventional bypass rapid prototyping and then present how the ETAS EHOOKS technology addresses those challenges. We will also explain in detail how the EHOOKS technology works and what makes it stand out as compared to other solutions on the market today. Finally, we will propose how EHOOKS may be used in other applications to help reduce development time and increase productivity.

CONVENTIONAL RAPID PROTOTYPING

DEFINITION - Let's first define bypass rapid prototyping in order to set the framework for the rest of this paper. Figure 1 shows a high level concept of bypass rapid prototyping. Here, an ECU native function (Throttle) is bypassed (or over-written) by calculations in an external model (e.g. in Simulink). As shown in this simple diagram, the external model first reads the inputs to the Throttle function, performs its own calculations and then overwrites the output of the Throttle function before it is routed to the ECU output ports. Conventional rapid prototyping requires several pieces to work in harmony. For example, the following components are needed (see Figure 2):

1. **ECU:** The ECU hardware needs to be instrumented with the correct data interface (e.g. ETK from ETAS, or a CAN controller), and the base software needs to be configured to support those interfaces.

2. **Prototyping Hooks:** The ECU function or variable that will be bypassed needs to have a "hook" implemented that allows the software to switch between its internal value and that calculated by the bypass function. It also allows the user to control which value is used.

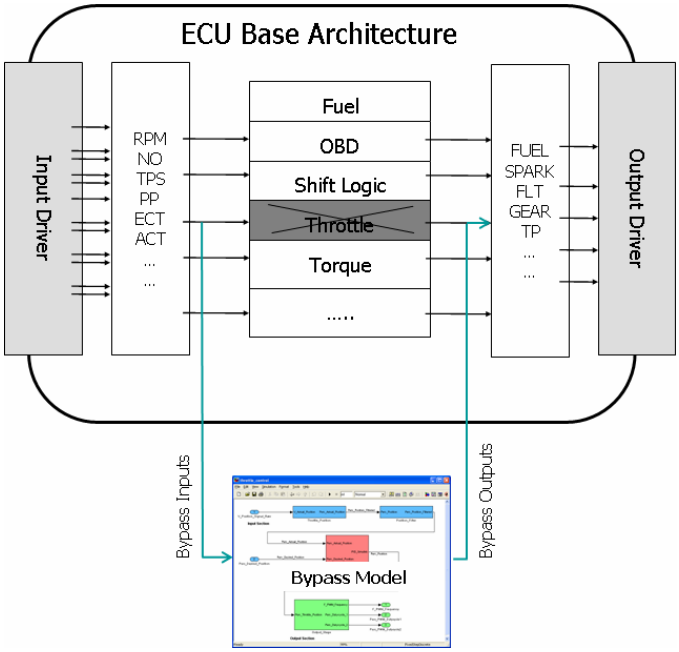

Figure 1: Schematic of Bypass Rapid Prototyping

3. **External Processor:** Usually the bypass function is executed on an external processor that is more powerful than the ECU's microprocessor.
4. **The data interface:** The data interface between the ECU and the external processor is crucial for the real-time performance of the bypass system.
5. **Software Services:** Support services in the ECU software are needed to manage the efficient transport of data to/from the external processor. These services also monitor the health of the data link, and abort bypass if the link is broken.
6. **Trigger Mechanism:** In order for the bypass function to run synchronously with the timer and event based tasks of the real-time operating system (RTOS) implemented in the ECU, a trigger mechanism needs to be in place. This allows the ECU microprocessor to be the master scheduler, and the external processor to be the slave.
7. **Configuration Tool:** The PC application that allows the new control algorithms to be mapped to the bypass hooks in the ECU base software, compile and execute the algorithms in the external processor, and manage/control the experiment in real-time.

Typically, an ECU tool vendor provides some of the pieces above, while the Tier 1 supplier or OEM provides the rest. There is an initial integration phase where the software services, hooks, interfaces and tools are tested to conform to the targeted development process. After this is done, the tools are made available for the algorithm engineer to use.

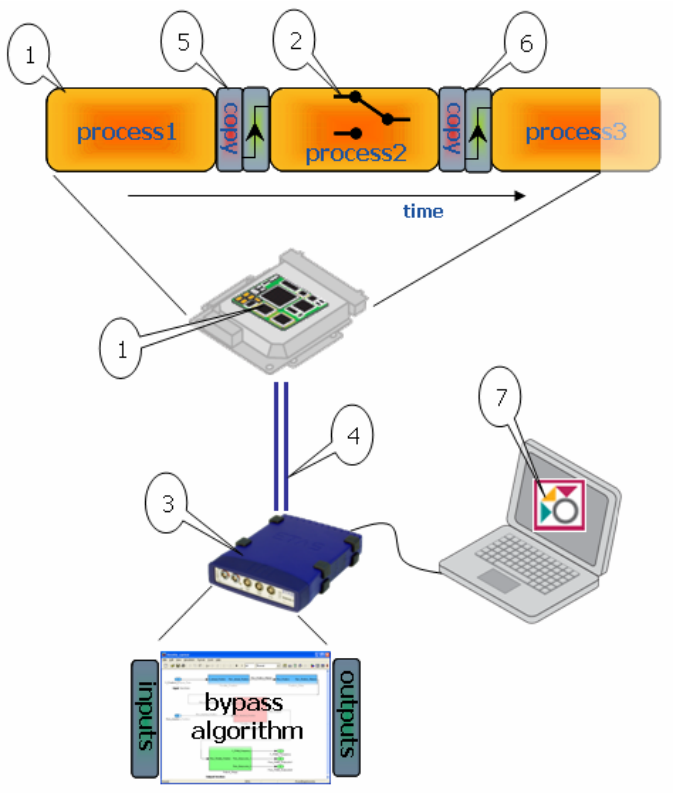

Figure 2: Typical System Components

EXAMPLES OF HOOKS – The usefulness and application of hooks (or switches) is determined by their location relative to an ECU function of interest. For example, for rapid prototyping, a hook is typically placed after an internal ECU function calculation is complete (Figure 3). Thus the ECU calculated value is overwritten by a different value. Bypass rapid prototyping relies on hooks of this type to allow testing of new algorithms.

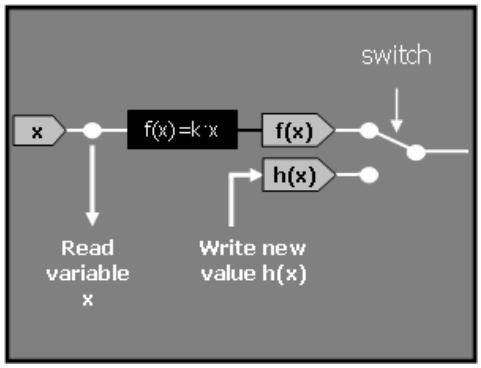

Figure 3: A Rapid Prototyping Hook

If the hook is placed at the input of the ECU function, however, it represents a different application (Figure 4). Test and validation engineers may use hooks of this type to test the robustness of the ECU function in the presence of input variability. Design engineers, on the

other hand, find it useful to test new sensor designs with existing ECU software.

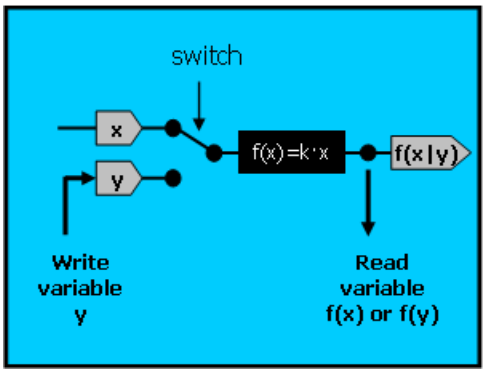

Figure 4: A Test and Validation Hook

Hooks may also be placed to re-write any ECU variable located in RAM with a calibration parameter that may be adjusted during run-time (Figure 5). Such hooks are useful when you need to fix a certain ECU value in order to focus on behavior observed downstream or isolate problems one by one. Calibration and diagnostic engineers find such hooks to be especially useful.

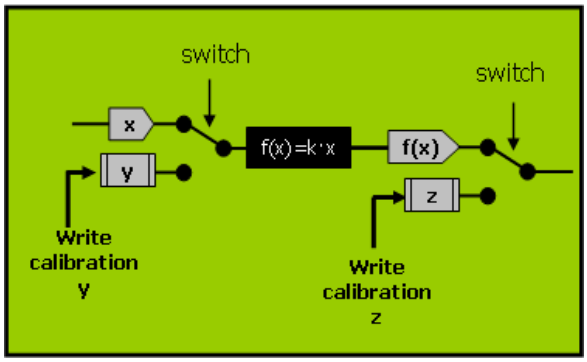

Figure 5: Calibration and Diagnostics Hooks

EHOOKS TECHNOLOGY

In the conventional framework described above, new control algorithms invariably require new software hooks to be placed in the base software. This task is arduous, expensive and causes significant development disruption – taking the rapid out of rapid prototyping! EHOOKS enables the end user to place all necessary hooks without the hassle of making changes to the base software. EHOOKS does this in the following way.

PREPARATION PHASE – An essential element of EHOOKS technology is the Preparation Tool. This software tool allows the initial one-time preparation of the ECU base software (e.g., by the Tier 1 supplier). It also captures and stores the following information inside an encrypted block in the ASAM-MCD-2MC (or A2L) file:

- Memory information – e.g. ROM and RAM for the addition of user functions and bypass hooks.
- Information regarding the ECU software architecture (e.g., the structure of global and local variables, update rate of key variables etc.)
- ECU variables and functions that may be bypassed by the end user. Software suppliers can use this feature to protect certain areas of their software (e.g. key input/output and diagnostic functions) from being overwritten unintentionally by the end user.
- ECU variables and functions that can be utilized by the bypass functions (e.g., math libraries, internal sensor values etc.)
- ECU RTOS tasks and function containers that may be used to execute bypass models.

In addition, this phase disables the ECU checksums (which might otherwise cause the ECU to shut down) and adds certain EHOOKS support functions and administrative calibrations/variables to the base software. Through this process, EHOOKS ensures that the subsequent hook installation phase results in a robust and reliable ECU image.

HOOK INSTALLATION PHASE – The second tool (called the EHOOKS Hook Insertion Tool) exploits the information stored in the A2L file by the Preparation Tool and uses it to configure and install the bypass hooks into the base HEX file. It does this in several steps:

- First, the user selects (from the A2L file) which variables in the base software will be bypassed, and with which type of hooks (e.g. constant, calibration, external, internal). EHOOKS also allows a copy of the original ECU value to be created for comparison purposes. Example dialogs are shown in Figures 6 and 7. Both replacement and offset modes of bypass are supported (Figure 8).

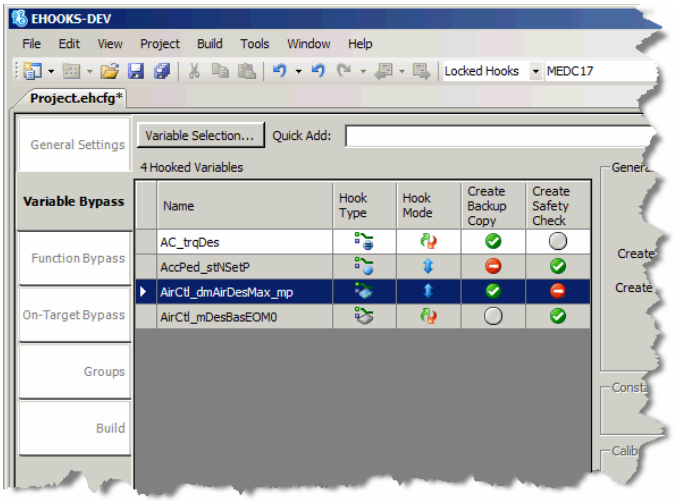

Figure 6: Selection dialog for Installing Hooks

- The Installation Tool then locates the write instructions to these variables in the base HEX file. As part of EHOOKS technology, ETAS has

developed an instruction set simulator (ISS) that allows these write instructions to be correctly located even in the presence of complex address calculations (Figure 9).

- Once the writes are located, the Hook Insertion Tool patches those locations with jump instructions to reserved memory areas where special hook-code is inserted (Figure 9).
- The hook control parameters are also added to the calibration memory of the ECU (e.g. the Enable parameter for each hook in Figure 8).
- User provided code for bypass functions is compiled by EHOOKS, which then generates the necessary link script to ensure the code (and its data) are located within the ECU memory sections reserved for use by EHOOKS as described in the encrypted section of the A2L file.
- During each build EHOOKS checks to ensure the available ECU resources reserved for EHOOKS use are sufficient for the specific configuration. If not, the build is halted with an error.
- The A2L file is updated with all the necessary hook control parameters

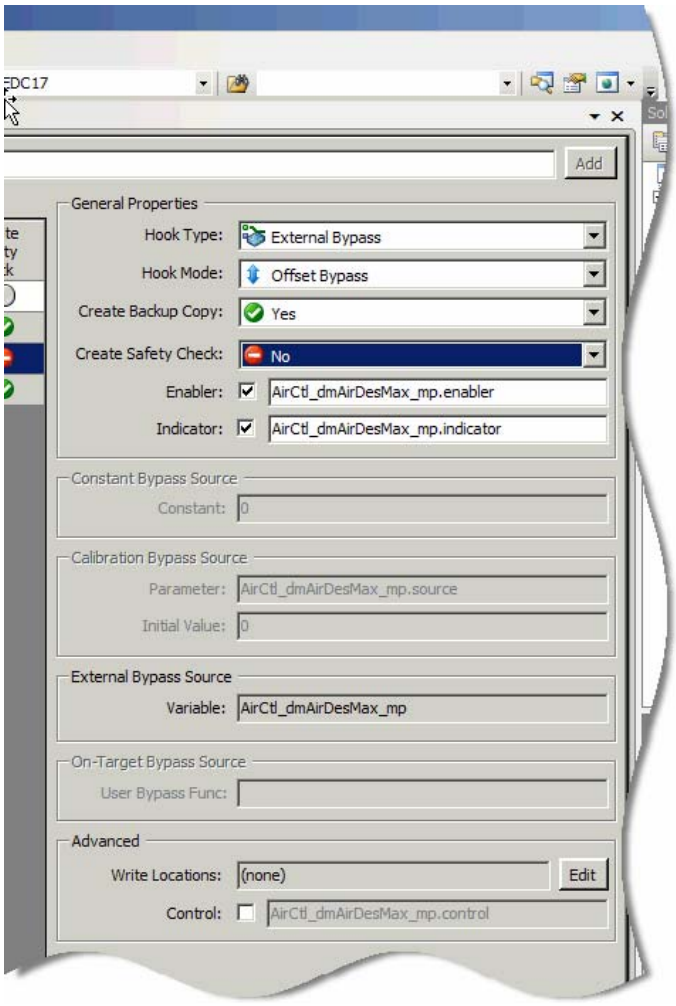

Figure 7: Selection of Hook Properties

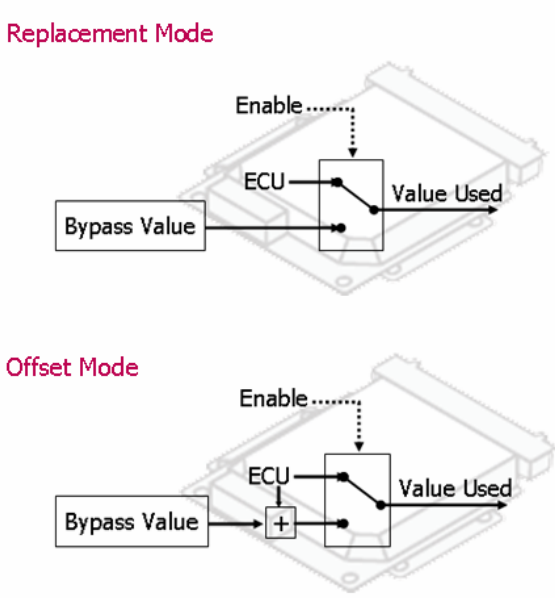

Figure 8: Bypass Modes supported by EHOOKS

The user can now rewrite the ECU memory with the new HEX file using standard flashing and/or debugging tools or calibration tools that support flashing (e.g. ETAS INCA software). The new A2L file may be used by standard calibration tools for accessing and controlling the operation of the hooks during run-time.

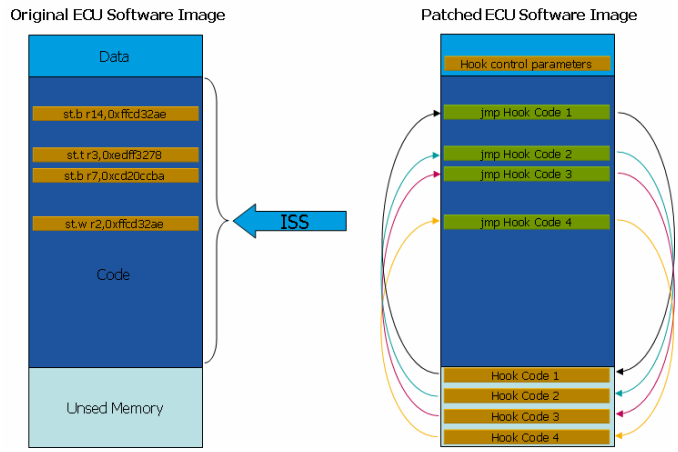

Figure 9: The original and patched ECU images

TYPES OF HOOKS – With EHOOKS, several types of hooks may be inserted in the original ECU software image, enabling several different use cases.

1. Constant Value Hooks (Figure 10) are useful when you need to fix an ECU RAM variable to a constant value (e.g. to simulate a faulty sensor). Once fixed, the value cannot be changed during run-time. It has to be changed by the EHOOKS Installation Tool and a new software image has to be flashed to the ECU.

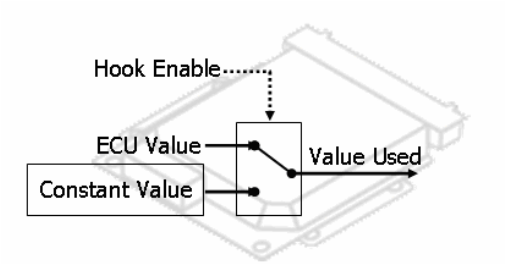

Figure 10: A Constant Value Hook

2. Calibration Hooks (Figure 11) are useful when you need to control the value of an ECU RAM variable during run-time using a standard calibration tool (e.g., INCA from ETAS). The calibration parameter is automatically added to the A2L file by the EHOOKS Installation Tool.

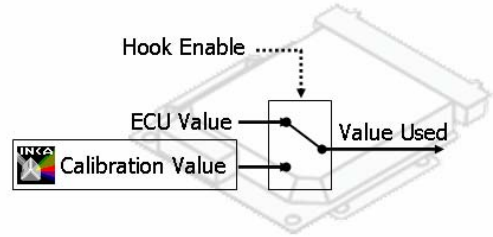

Figure 11: A Calibration Value Hook

3. Internal Bypass Hooks (Figure 12) allow an ECU RAM variable to be overwritten by an algorithm running directly on the ECU. This methodology, commonly known as on-target prototyping, is very useful when the ECU has spare resources (ROM, RAM, task-time) available to run new algorithms.

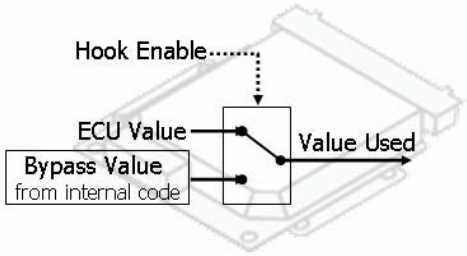

Figure 12: Internal Bypass Hooks

4. External Bypass Hooks (Figure 13) allow traditional bypass rapid prototyping using an external processor. Bypass algorithms may be developed in model-based tools such as ASCET from ETAS or Simulink from The Mathworks, or directly in C-code. EHOOKS only configures the hooks in the ECU image to accept values written by an external target, independent of which external target or modeling language is actually used.

EHOOKS, therefore, provides a completely scalable solution for a wide variety of applications – starting from simple calibration use cases (that require minimum additional ECU resources) to on-target and external rapid prototyping.

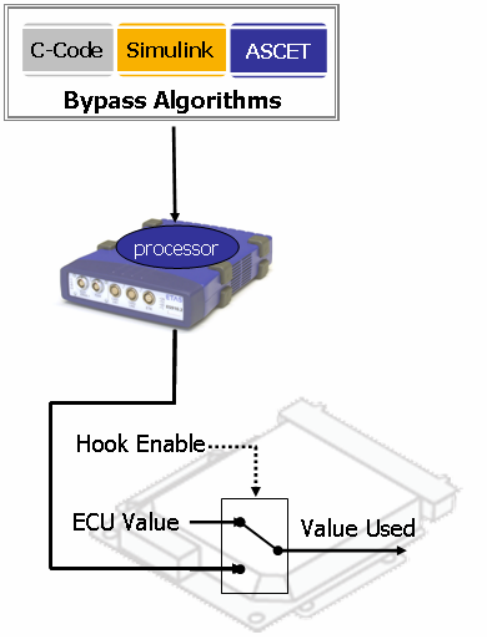

Figure 13: External Bypass Hooks

The technology leverages proprietary ECU internal information encrypted and stored in the A2L file by the Tier 1 supplier. This allows EHOOKS to not only reliably detect all write instances of a RAM variable, but also institute run-time safety detection mechanisms that alert the user to potential data inconsistencies during run-time. In addition, the hook code inserted by EHOOKS includes the code from the original ECU software such that, if the hook is disabled, the original ECU software functionality is restored.

ADVANCED APPLICATIONS

In addition to the use cases described above, EHOOKS technology may be applied to solve a myriad of advanced problems related to software development and validation. Some of these are described below.

1. Disabling an ECU process: The order of execution of processes inside an ECU task is pre-defined and normally cannot be altered by the user.

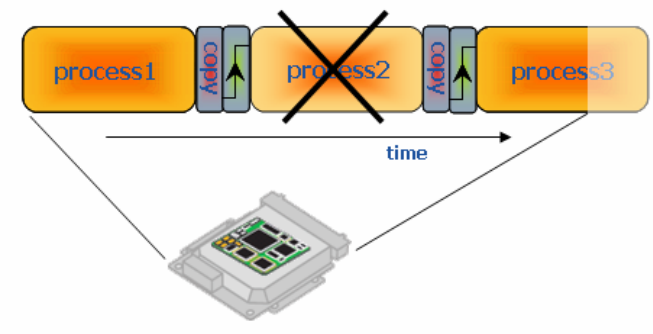

Figure 14: Disabling an ECU process

The encrypted A2L block created by the EHOOKS Preparation Tool allows the Tier 1 supplier to securely provide the process list to the EHOOKS Installation Tool, which then allows the user to select and control the execution of specific processes (Figure 14). This feature is useful, for example, when:

- An entire ECU function has to be bypassed. By disabling the whole process, significant CPU resources can be freed-up for a new algorithm, or
- An errant ECU process is preventing downstream processes from functioning properly, or is paralyzing the ECU.

2. Scheduling of on-target algorithms: EHOOKS provides the flexibility to leverage existing ECU resources in the best way possible. One way to do this is to allow user code for on-target algorithms to run within a dedicated process created solely for that purpose. These processes are known as "bypass containers" (Figure 15). The actual software hook may reside in a different (e.g. faster) process, but due to bandwidth limitations it may be not be possible to execute the new algorithm in the same task. In many cases, it is still be acceptable to run the new algorithm in a slower task in order to check its validity. The EHOOKS Hook Insertion Tool (optionally) allows the user to pick the bypass container from an available list of processes, and assign the algorithm to it.

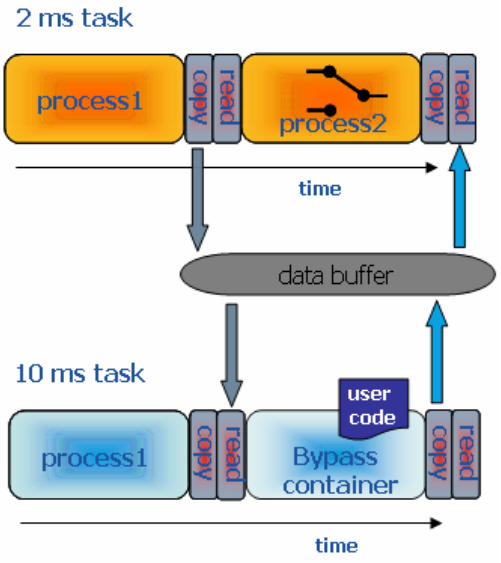

Figure 15: Bypass Container Process

3. Function in the Loop (FiL): In a traditional hardware-in-the-loop (HiL) setup (Figure 16), the ECU is connected via an electrical harness to a HiL system that runs a vehicle system model in real-time and generates the necessary sensor signals (e.g., crank, cam, wheel speeds). The sensor signals are identical to those available on a real vehicle and are passed to the ECU via the electrical harness. In return, the actuator signals computed by the ECU software are sent back over the harness to the HiL system, where electrical or simulated loads are used to emulate the conditions found on the vehicle. This process requires extensive signal conditioning on the HiL system in order to convert physical values (e.g. temp in deg F, speed in m/s) to electrical values (e.g. voltage) and vice-versa.

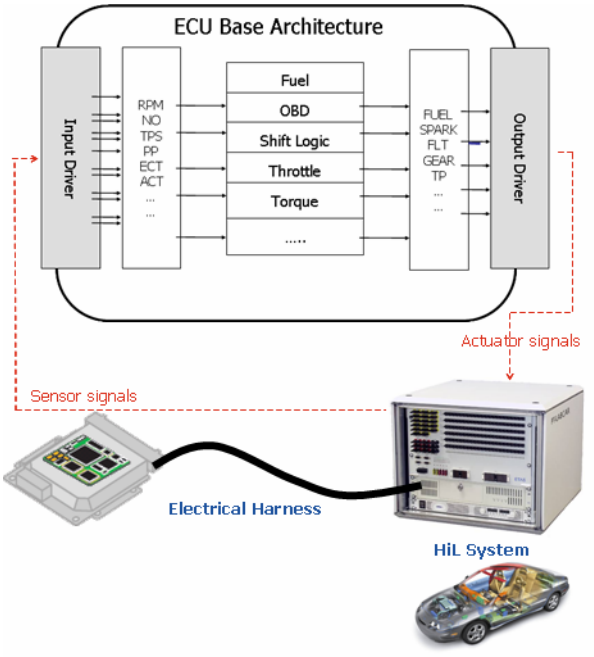

Figure 16: A conventional HiL system setup

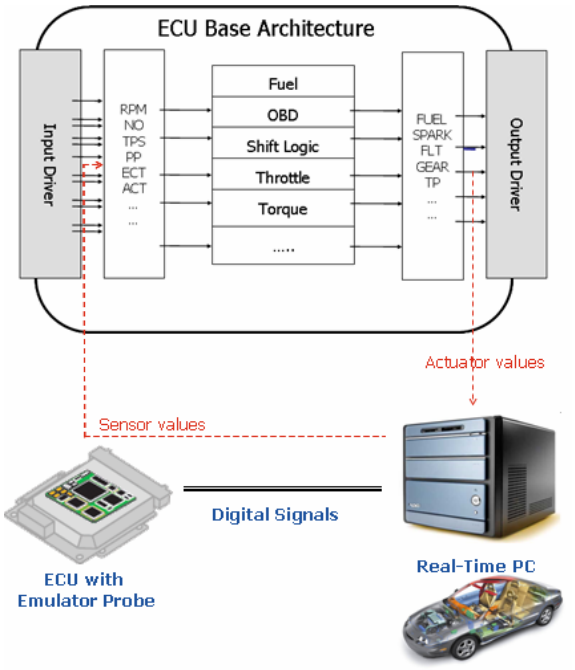

Figure 17: A Function-in-the-Loop System

Although HiL systems come close to simulating the environment of an ECU, there are significant costs involved in building the signal conditioning hardware necessary for proper operation. In the recent past, the FiL methodology has emerged as a way to overcome these challenges of HiL and reap some of the same benefits. With FiL (Figure 17), the expensive signal conditioning and load hardware is avoided. The plant model running on a real-time PC (e.g. the ETAS LABCAR-RTPC) generates sensor signals that are copied directly to the ECU memory. Similarly, the actuator commands are read directly from the ECU memory and connected to back to the plant model. This direct exchange of software signals is made possible via a memory emulator probe (e.g. the ETAS ETK device) mounted on the ECU. Since the electrical interfaces of the ECU are idle, it is necessary to disable diagnostic functions in the ECU software for FiL to work.

EHOOKS technology is used to install the necessary software hooks for FiL – e.g., to overwrite an ECU memory location that is normally used to store a sensor value calculated by the ECU.

CASE STUDIES

At the time of writing, ETAS is in the final stages of releasing the EHOOKS product to the market. However, the technology has already been proven and refined through select lead customer projects over the last eighteen months. Some key examples of customer projects are presented below.

- In Asia, a major OEM is using EHOOKS for accelerating the development of new ECU functions via traditional rapid prototyping. They are now able to avoid the cost (typically €10k - €30k) of each new software change request for new hooks charged by their Tier 1 supplier. They are also able to reduce the number of ECU software releases required before start of production by about 50%.
- In Europe, a major OEM is reporting that by using EHOOKS they are able to create a new software version in about 5 minutes, when it previously took them anywhere from one to eight weeks.
- In Europe, another major OEM's Tier1 supplier is unable to manually add hooks to the software because they don't have access to certain sections of the code. In this situation, the only option for the OEM is to use EHOOKS to add the necessary bypass hooks.
- A FiL system developed by ETAS jointly with a leading European Tier 1 supplier is in use for software development, system integration and testing.

CONCLUSION

In this paper, we have presented a new patent-pending technology from ETAS that inserts software hooks and code into an existing ECU software image without the need for the source code. This technology allows the software vendor (Tier 1) to securely share proprietary ECU architecture and implementation details (necessary to make EHOOKS flexible and robust), without exposing their core intellectual property to the OEMs. Using this technology, controls engineers can once again "rapidly" develop and test new strategies either directly on the ECU (if there is enough memory and processor bandwidth available) or on an external rapid prototyping system (such as the ETAS ES910) without incurring the cost and delays associated with traditional software hooks.

ACKNOWLEDGMENTS

We wish to acknowledge the tireless efforts and innovative spirit of the entire EHOOKS development and product management team at ETAS. In addition, thanks go to several lead customers who provided invaluable input and feedback to help ETAS refine the technology.

REFERENCES

1. Gebhard, M.; Lauff, U.; Schnellbacher, K.: Operation am offenen Herzen – Entwicklung und Test von Steuergerätefunktionen mit der Bypass-Methode, Elektronik Automotive 2008, Issue 6, Pg. 34-39.
2. Dubitzky, W.; Eismann, W; Schinagl, J.: Einsatzmöglichkeiten der Bypass-Methode für Entwicklung und Test von Steuergerätefunktionen, Electronik Automotive 2008, Issue 8, Pg. 52-56.
3. Triess, B.; Müller, C; Lauff, U.; Mößner C.: Entwicklung und Applikation von Motor- und Getriebesteuerungen mit der ETK-Steuergeräteschnittstelle. ATZ Automobiltechnische Zeitschrift 109 (2007), Issue 1, Pg. 32-39.
4. INCA, ASCET, LABCAR-RTPC, ETK and other ETAS products: www.etas.com
5. ASAM-MCD-2MC, www.asam.net

CONTACT

Vivek Jaikamal is a Marketing Manager for software engineering tools at ETAS, Inc. in Ann Arbor, MI. He can be reached via email at vivek.jaikamal@etas.com

Nigel J. Tracey is a Product Manager for ETAS embedded software tools based out of York, UK. He may be reached via email at nigel.tracey@etas.com