A Concurrent Real-Time White Paper

SIMulation Workbench® and HIL testing at Ford Motor Company

Charlie Johnston
Concurrent Real-Time

Tim Cardanha
Ford Motor Company

Background

In the August 2003 issue of SAE Automotive Engineering International magazine, Ford and Concurrent first published "*Virtual Powertrain, Real Results*", describing the architecture of the Ford ECU HIL developed in partnership with Concurrent. This article laid out the now generally accepted primary reasons for automotive HIL testing – ECUs are often available before vehicles, bench testing is faster/easier/more repeatable than in-vehicle testing, HIL testing can be automated, to name just a few.

That architecture was built around Concurrent's quad-CPU computer running Concurrent's PowerMAX OS. It was generic in some ways (in that it was reconfigurable for different tests and ECUs) and inflexible in others (hand-coded C and Fortran modules, custom coded I/O, and an ad-hoc test manager). While modular, the architecture was somewhat inflexible in terms of growth and the addition of new capabilities. Adding new simulation components, like MATLAB/Simulink derived components, was difficult and time consuming, or simply impossible.

Introduction

Technologies have evolved greatly over the intervening decade. On the software front, there has been a move away from hand-coded models to high-level modeling tools like Simulink, SIMPACK, Dymola, etc. These may be plant models, virtual control models, driver behavior models, models of the environment, etc. They are typically much higher fidelity models than the hand-coded variety, but at the expense of higher complexity "under the hood"; that is, when the model is converted into a form usable by a real-time HIL simulator.

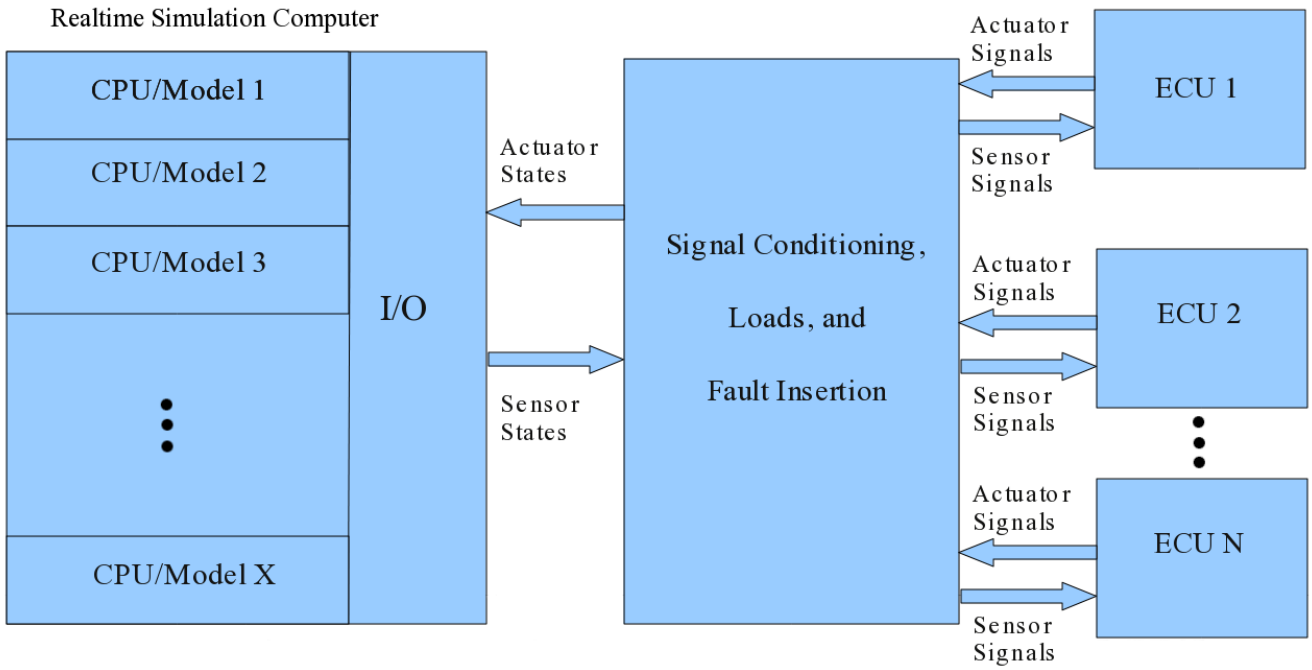

Fig. 1 – HIL Simulation Architecture

Simulations may need to support multiple models, each with sub-components running at multiple rates (Fig. 1). All of this being said, there must still be the capability to incorporate legacy software that has not yet, or will never be, migrated to the newer tools. All of this complexity must be managed.

Ford has advanced along with technological changes and now embraces a more model-based approach to testing. Formerly hand-coded plant models are now Simulink-based. Multiple organizations are sharing models and new test methodologies are being defined in a way to increase efficiency and reduce redundancy across the company. The groups that are customers of the HIL labs are also embracing these technologies, and increasingly want to supply additional modeling components as higher-level models as well.

SIMulation Workbench Overview

Concurrent's SIMulation Workbench (SimWB) is a simulation configuration and management tool. Running under Concurrent's RedHawk Linux real-time operating system on Concurrent computers, SimWB pulls together all of the software components into a test framework that is easily configured and managed by the user. Support is provided for models developed under Simulink, Simscape, SIMPACK, GT Power, VI Grade, CarSim, veDYNA, Dymola, OpenMP, MapleSim, AmeSIM, and any vendor that supports the FMI/FMU standard. Legacy software can be imported with very few changes. The user can configure their simulations to use models from any combination of these sources. Many of these packages result in tasks (running programs) with multiple threads of execution. SimWB lets the user independently target models and their threads to individual hardware CPUs for increased parallelism, resulting in better overall performance.

SimWB Workflow

The SimWB configuration and monitoring tools can be run on either Windows or Linux computers. In its typical usage, a Simulink model is developed and tested within the standard MATLAB/Simulink environment on Windows PCs. There are no special blocks that must be added to the model - not even device I/O blocks. When the user decides the model is complete, he will then use the SimWB MATLAB toolkit to import the model into SimWB on the real-time simulation computer. The PC can be anywhere – it only needs a network connection to the real-time computer. The toolkit guides the user through the two simple steps needed to bring their model into SimWB.

The first step is RTDB (real-time data base) generation. The RTDB essentially defines what things (variables, parameters, signals, etc) in the model will be visible to the outside world. All data in the simulation flows through the RTDB. It is via the RTDB that all I/O, monitoring and control functions are done – all *outside* of the executing model. We will return to the role of the RTDB later.

The final step is to build the model within SimWB. When the source code is generated the appropriate references to the RTDB are inserted into the code. This code is then packaged up and sent to the SimWB and compiled on the real-time computer. At this point the model is now available as a component of a SimWB simulation (also known as a “test”).

A test definition is a user-defined set of models (Simulink, Dymola, legacy user code, etc.) that have been brought into SimWB and that we want to run together, along with their associated RTDB. To run our HIL test, I/O will be needed. I/O definitions are actually part of the RTDB. I/O “mapping” between channels and RTDB variables are all done via board-specific GUIs (Fig. 2). You can see the board-level device options selected on the upper left panel. The lower left panel shows all the channels on this device. To the right, this panel shows all the RTDB items that this channel can be mapped to. The user simply checks the box next to the RTDB item name to map it.

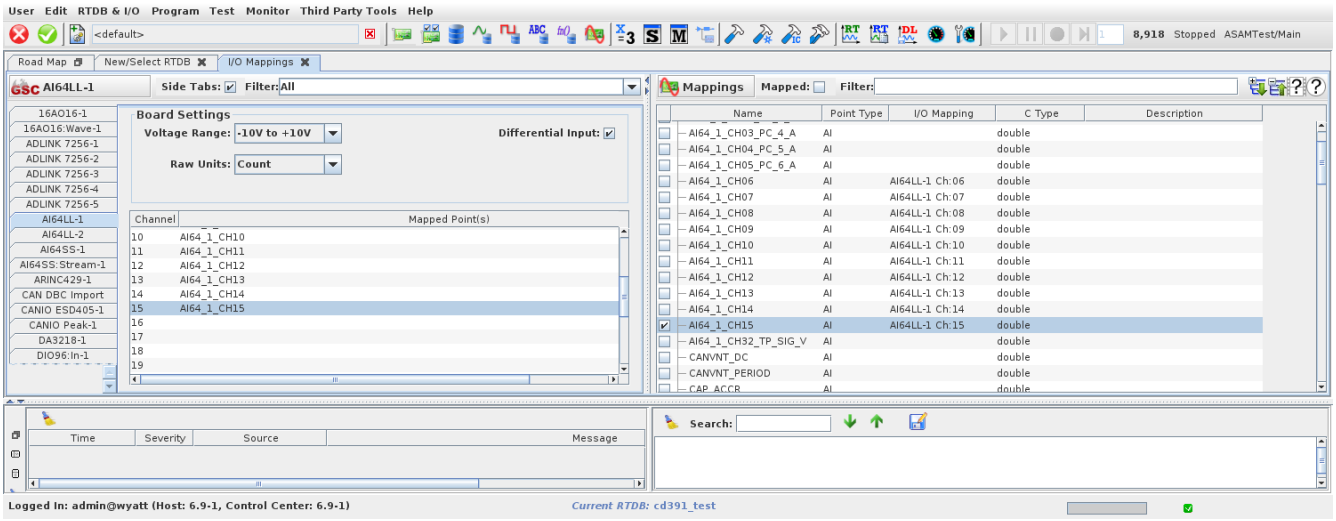

Fig. 2 - I/O Mapping of GSC AI64LL Channels (right) to RTDB Items (left)

Once an association (or “mapping”) is made between an RTDB item and an I/O channel then whenever SimWB runs the test it will automatically start the software needed to perform that I/O operation. These I/O operations are completely outside of the execution of the running model(s) – inputs are always written into the RTDB before the model(s) run, and outputs are always gathered after the model(s) have completed a time step. The running model has no notion of I/O – its inputs are simply there in the RTDB when it needs them. Because of this independence, I/O devices can be replaced, channels remapped, etc., without ever having to touch a test model.

Using SimWB at Ford Motor Company

In a recent SimWB-based HIL ECU setup, the Ford plant model, developed in Simulink, is some 7200+ blocks in size. The SimWB real-time computer system is a 16-core server running Concurrent's RedHawk OS. All the I/O resides in a separate 20-slot PCI chassis directly connect to the server. All I/O boards are COTS (commercial off-the-shelf) except two (Concurrent developed analog output and PWM input cards which are much faster than their COTS alternatives).

An additional component required for the HIL is the inclusion of the Ford developed signal conditioning and fault insertion (SCFI) subsystem and its software. This software – legacy C code running in the previous generation HIL system, needed to be incorporated into SimWB. The software was designed to monitor requests via shared memory, and to control the SCFI subsystem via a serial interface. SimWB API calls were added to the software to allow control by the SimWB scheduler. The shared memory monitoring and status reporting sections of the code were modified to read and write RTDB locations using other SimWB API calls. The configuration files used by the previous generation system could be used as-is on the new system. It took about a day to modify the software. The software was brought into SimWB as a "user model", and is now available to be included in any test definition.

In order to achieve platform-independent test specification, Ford developed a PC-based test editor and Python back-end that incorporates the ASAM HIL interfacing protocol. The ASAM HIL spec defines a "model access port" and a collection of API's that can be used to both provide plant model stimulus and define data capture tasks. SimWB provides plant model access to ASAM HIL test tools via the ASAM package available in the SimWB Python Toolkit (PYToolkit), available for Linux and Windows. The Ford editor lets the user define test sequences that can be run separately or in groups. The test sequence may involve one or more stimuli (signal generators), one-shot variable changes, timing events, logical and data-driven events, and definitions of capture tasks. Capture tasks can be time based or event driven based on plant variable values. Data from captures are automatically stored for subsequent analysis. Since all test definitions are built on top of the ASAM standard, these tests can be executed on any real-time target that supports the standard. This allows test management and target technologies to evolve independently without the need to "rework" the interface between them (because it is controlled by the ASAM standard).

At times the Ford engineers need to interactively "drive" the simulated vehicle. Using SimWB user models made adding this interaction easy and modular. Sound and USB pedals were added as optional test components via user models that can be pulled into any test definition. The sound model uses plant model RPM and crank to modify and playback a sound file. The output is fed directly into an RTDB location that is mapped to an analog output channel driving a speaker. The pedal model takes USB throttle, brake and clutch inputs from a gaming controller and feeds those values into the appropriate RTDB items for input to the plant model. This is in addition to the standard SimWB RTViewer GUI which provides the user complete visibility into the RTDB. Variables can be modified, displayed, or plotted with just a few mouse clicks. Signal generators are easily defined and can be attached to any variable and controlled via run/stop/pause buttons.

The interactive environment also takes advantage of the rich set of display and control widgets available from the SimWB human/machine interface (HMI) or GUI builder. These widgets include things like buttons and sliders, gauges, LED's and active SVG elements. The display widgets can be connected to any item in the RTDB. They are Java based, so the same HMI will work on either Linux or Windows. As an example, Fig. 3 is an HMI that Ford test engineers use to drive the simulation and monitor a few key model outputs.

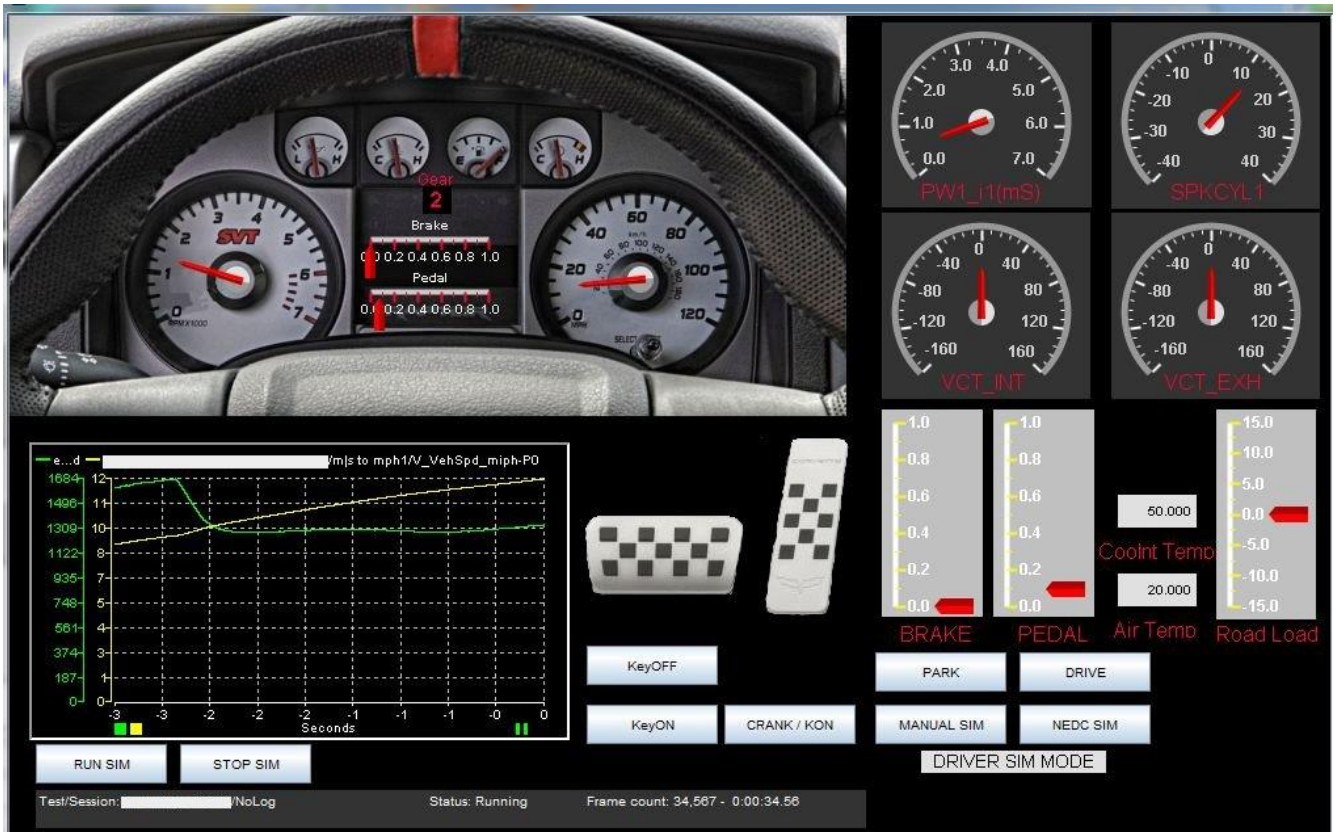

Fig. 3 – Ford HMI

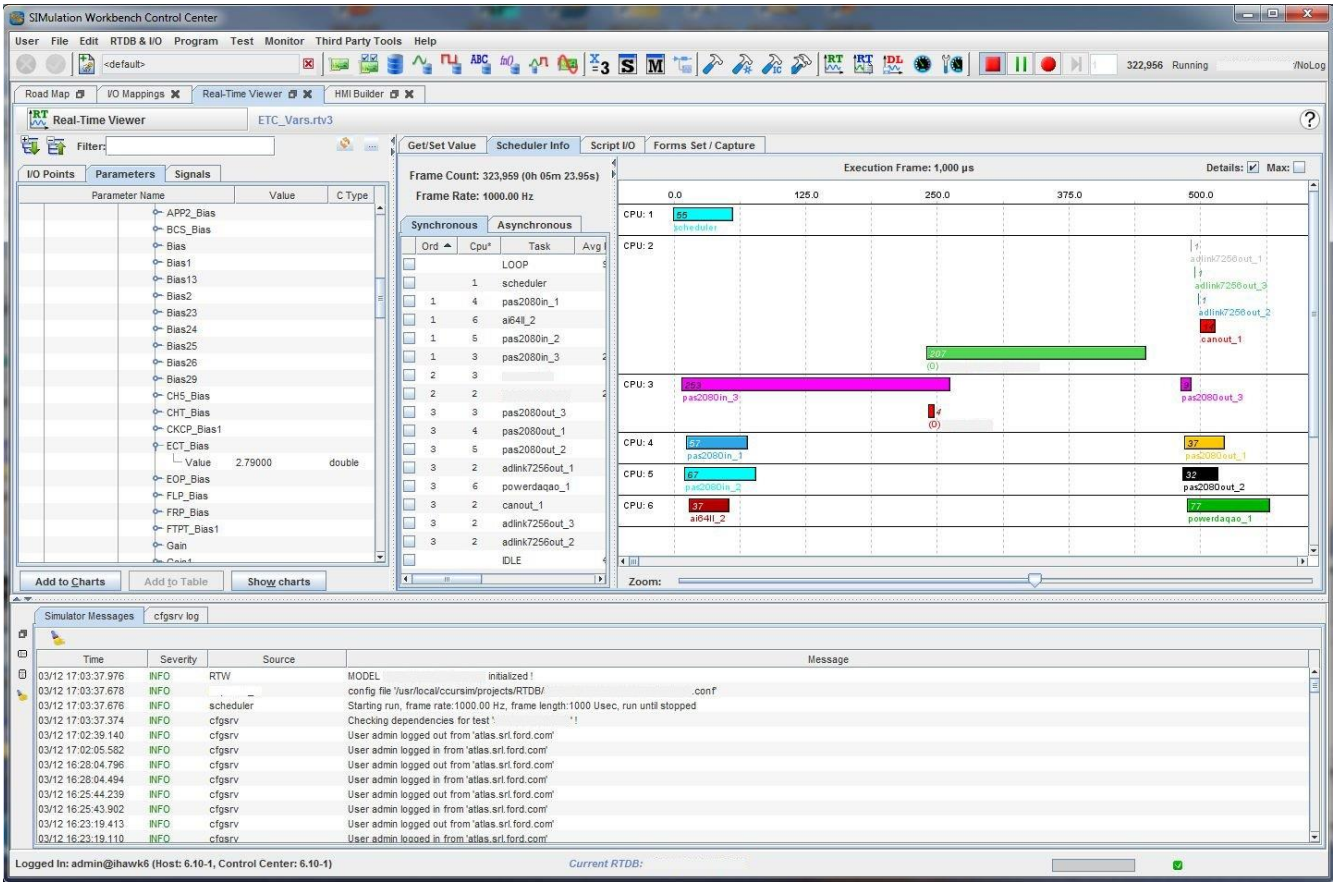

Fig. 4 – RTViewer
Left pane - RTDB item selection for “peeking” and “poking”
Center/Right pane – scheduling data/graph during the session
Lower pane – message window

In Fig. 4 you see this test running. The I/O tasks can be clearly seen graphically in the right-hand pane. On CPU 1 is the SimWB scheduler. The input I/O tasks are to the left, the output tasks to the right. The plant model and an associated task are in the middle (both starting just before the 250 µsec mark, their names have been redacted). This clearly shows that the model cycle, including all I/O, completes in approximately 560 µsec (of a 1000 µsec frame). This is a real-time display that updates as the test is run. An interesting note here is that the pas2080in_3 task seems to be rather long. This is a candidate for optimization by moving some of the inputs from that board to one or both of the pas2080in_1 and _2 boards in an effort to balance the load and give more time to the plant model if needed. Because of the independence of I/O from the model in SimWB these changes can be made without model modification or rebuild.

Conclusion

In order to support the higher complexity of today's advanced modeling technologies, Ford needed a tool to provide a framework for real-time simulation that was easy to use, flexible and open. The tool needed to integrate with Ford's vision of a target agnostic simulation computer, and still be open enough to integrate legacy application software. SimWB provided that framework. Models from today's most advanced packages are easily integrated alongside legacy code, moving virtually untouched from the desktop to the simulation target without need of modification. And by designing to the ASAM HIL specification, the Ford test editor can communicate with and control SimWB as a fully supported ASAM HIL model access port via the SimWB PYToolkit. This approach is enjoying successful use today, and should take Ford ECU HIL testing well into the future.

Glossary

COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit (on the SimWB host)
ECU	Engine Control Unit
GUI	Graphical User Interface
HIL	Hardware in-the-loop
HMI	Human/Machine Interface
OS	Operating System
RTDB	SimWB Real-time Database
SCFI	Signal Conditioning and Fault Insertion
SimWB	SIMulation Workbench
SVG	Scalable Vector Graphics