About This Issue
In this issue, Asaad Makki and Dave Beard at Ford Motor Company discuss the importance of CAE analysis to reduce development costs and assure robustness in electrical system designs. As Ford looks ahead to evolving technologies that bring software validation into the virtual world, Synopsys technologist, Victor Reyes, details how such a virtual environment can enable virtual Hardware-in-the-Loop (vHIL) SW testing—using fault-injection scenarios—before ECU hardware is available. In the Q&A section, our team of experts address questions on the topics of model characterization and simulation of power semiconductor switches (MOSFETs, IGBTs and diodes), virtual prototyping for SW validation, as well as FPGA applications for automotive.

We hope you enjoy this issue!

With sincere regards,
The Synopsys Automotive Team

In This Issue
Electrical System Design, Verification and Analysis using CAE

Adopting CAE analysis tools has helped Ford to cut the number of physical prototypes it uses, reduce costs, and improve quality. Asaad Makki, Global Electrical CAE Supervisor, and Dave Beard, Senior CAE Engineer, explain their approach.

The challenges that ever-increasing electrical/electronic complexity brings to the automotive industry means that we, in the industry, increasingly rely on computer-aided engineering (CAE) and virtual design verification tools. Development cycles are getting shorter, and, to meet customer demand, we have to be able to deliver robust designs as quickly as possible. Prior to the availability of advanced EE simulation tools, we depended on spreadsheet analysis to do simple calculations to help us study the interaction between components—for example, a control module and a switch. This approach was limited to studying specific parameters for components—such as ensuring the switch received sufficient current to make a good contact. While this form of analysis was limited, we learned a lot about the importance of data accuracy and of fully capturing and understanding the data parameters, so that we could enable accurate results, and create models that mirrored the real world.

Mirroring the Real World
Creating an analysis that is thorough, accurate and mirrors the real world is a top priority. Since the tools we use now are far more sophisticated than our early spreadsheet models, we spend a lot of resources ensuring that the models we use are accurate, whether they are sourced in-house or from suppliers.
The alternative to CAE analysis—the use of physical prototypes—is becoming prohibitively expensive. In addition to the cost of building the vehicle itself, program teams must pay to have access to the vehicle for each day of physical testing. The extensive tests that we run using CAE analysis tools would take many days of testing using physical prototypes and can’t even begin to approach the number of “what-if” scenarios that CAE analysis can cover.

Improving Robustness

As well as helping to reduce development costs, CAE analysis makes a significant contribution to the robustness of our designs. We run hundreds or thousands of analyses with variations of component tolerances and look at the effects of temperature changes and aging. We can then use data from the analysis tools to isolate the component tolerances that matter, and tighten ones of interest, or make other changes that improve the robustness of the overall subsystem.

Sometimes, the quality issues we uncover have implications for our suppliers. A circuit design may pass the verification tests at the component level, but may need improvement when used in the subsystem. We work closely with engineers and suppliers, providing feedback which results in more robust designs.

Developing CAE Plans

A typical vehicle CAE plan may have over 500 EE analyses. We use Monte Carlo analysis extensively to study variations in parameters across hundreds of scenarios. We also use our CAE environment to look at DC and transient analysis, as well as sensitivity. Pareto analysis (Figure 1) helps us with our detective work; we typically use it to understand what the significant contributors are to signal variation over many Monte Carlo runs.

An example of one of the types of analysis that we focus on establishes the robustness of shared signals (Figure 2) among different modules in the vehicle. One component may be the source of a signal that is monitored by several other components. The analysis of shared signals is critical to ensure that designs from several different component suppliers are able to perform all their functions with complete accuracy and robustness under all expected operating conditions. CAE analysis allows us to perform hundreds of different analyses to get statistically valid results. This is cost and timing prohibitive when trying to use a physical prototype in the lab.

Modeling Electro-Mechanical Systems

Electromechanical systems have been and will be increasing their footprint in vehicle system and subsystem designs. To better understand the mechanical effects within electro-mechanical systems, we need to be able to analyze physical components alongside the electrical ones.
For example, we’ll model the mechanical properties of a motor. On a more complex level, we have developed an electro-mechanical model for power window subsystem. The mechanical teams (who may have little experience with the electrical domain) can use the model to help them choose what size motor they will need, or the different torque/power losses in the system they will use, based on the characteristic interaction of their mechanical and electrical components.

**Prototyping Software**

The cost of developing software for automotive applications is growing enormously, because of the growth of software content in cars. Software verification is an issue that affects quality and the performance of safety-critical systems. Now that we have a methodology in place to reduce our use of physical prototypes for the electrical and electro-mechanical subsystems, our next area of focus is the software domain. If we can have our developers use virtual models to do functional testing, we can start to reduce our dependency on the use of breadboards even further, as well as benefiting from faster system verification, better debug and improved quality in the software domain. Once we have a virtual software solution, we will be closer to being able to perform complete system validation using a combination of virtual prototyping and CAE analysis.
As we evolve, and the industry evolves, we look forward to virtual prototyping, with links to Saber, evolving to bring that missing piece—the software piece—into play.

**Solution summary**

Our EECAE analysis environment integrates well into our design environment. It enables us to:
- analyze electrical and mechanical components together,
- use advanced statistical analysis to mirror real-life situations,
- choose from thousands of pre-defined library components,
- easily connect components from different domains so that we can analyze systems at different levels of abstraction, and
- use the multi-domain simulator interface to look at mechanical, thermal and electrical issues to see how changes in one domain affects the others.

**Project Profile**

**CAE analysis environment:**
Synopsys Saber

**Algorithm and software modeling:**
MathWorks Simulink®

**Harness design integration:**
Saber Frameway

**About Ford’s EECAE Team**

Ford’s EECAE team supports all vehicle program teams with electrical computer-aided engineering analysis and design verification. It also provides vehicle program teams with support services such as current-based limit testing and with EE design alternative investigation.

---

**About the Authors**

Asaad Makki earned a Ph.D. degree in Electrical and Computer Engineering from Wayne State University, in 1993. He is currently a member of the engineering management team at Ford Motor Company, where he is leading the global electrical CAE group. He held a Technical Expert position at Ford Motor Company between years 2000 and 2002. Dr. Makki held several positions at the TRW Automotive Electronics Division ranging from a Systems Engineering manager, to a Staff Engineer, to a Senior Engineer for over eight years. Dr. Makki started his career as a software engineer at the Computer Methods Corporation in Michigan, where he worked on software algorithms design to develop new body electronics features.

Dave Beard began his EE career as a Technician in the US Navy, working on satellite communication and cryptographic equipment. After receiving his BSEE from Lawrence Technological University, he performed robustness circuit analysis on small jet engine fuel systems with Williams International. Dave moved to Ford, and has held positions involved with manufacturing quality, value engineering, systems engineering, design and release, and, for the past 12 years, EECAE, performing system and component analysis, design verification, and CAE procedure authoring. Dave's current position is EECAE Senior Engineer, based in Dearborn, Michigan.
The automotive industry faces a significant challenge: to release cars with zero software defects. The number of recalls and redesigns due to software problems illustrates the magnitude of the challenge—they have grown exponentially over the last decade. There are major economic implications for manufacturers since the cost of recalling a vehicle can be huge, especially when the issue affects the integrity of the brand.

Manufacturers already spend a lot of money on software testing. In fact, testing accounts for about 75% of the cost of software development. And that spend is set to grow as the number of tests that manufacturers have to run continues to increase.

But simply increasing the number of tests is not always the best way to reduce defects. Improving tests, so that they exercise corner cases that are not triggered by normal operation, improves quality.

Standards, like ISO 26262z, address the planning and development of safety-critical systems and place further demands on software testing. ISO 26262 provides an automotive-specific, risk-based approach based on Automotive Safety Integrity Levels (ASIL). It specifies the requirements and recommended methods for validation of the safety levels including fault-injection testing.

Fault injection helps to determine whether the response of a system matches its specification in the presence of faults. It helps system engineers to understand the effects of faults on the target system behavior. It also helps to assess the efficiency of fault-tolerance mechanisms, and enables the design team to reduce the presence of faults during the design and implementation phases.

Fault Categories
We can categorize faults as either software or hardware faults. Within the hardware category, faults are either:

- permanent (triggered by component damage),
- transient (triggered by environmental conditions, also known as soft errors), or
- intermittent (caused by unstable hardware).

Fault injection can improve test coverage of safety mechanisms (at the system level) by covering corner cases that are difficult to trigger during normal operation. It is also recommended whenever a hardware safety mechanism is defined, to analyze its response to faults, and where arbitrary faults corrupting software or hardware components must be injected to test safety mechanisms.

Virtual prototypes provide complete frameworks to create advanced fault-injection scenarios, which are non-intrusive, run faster and offer more control and visibility than traditional methods. Victor Reyes, Synopsys, explains their relevance to new safety-critical standards like ISO 26262.
Comparing Fault-Injection Techniques

Table 1 compares four traditional fault-injection techniques by considering some of the most important factors. These include:
- the type of faults that can be triggered (fault injection points),
- the ability to model permanent faults,
- intrusiveness, or how the injection of the fault changes the original execution flow of the system,
- observability, which defines how well the set of reactions (or events) triggered by the fault can be seen and recorded,
- controllability, which defines the precision in terms of exact time and location where the fault can be injected,
- repeatability, or the ability to repeat the test in a deterministic fashion, and
- speed, which will define to some extent the complexity and duration of the test scenarios.

Typically, the test team will use ‘hardware-based with contact’ fault injection to inject errors on the IO boundary of the Electronic Control Unit, and ‘hardware-based without contact’ to trigger soft errors—for example, to simulate memory corruption due to radiation or electromagnetic interference.

The ‘with contact’ techniques can model permanent faults, and they are not intrusive (although there is a risk of damage if misused). The ‘without contact’ techniques are more focused on transient errors and are also non-intrusive. However, there is little control on when and where the fault will be injected.

Software-based fault-injection techniques can only inject errors on those locations accessible by the software, that is, memory and registers for memory-mapped peripherals, which means they are only able to model transient faults. The biggest problem with software-based fault-injection techniques is that they are intrusive: because the engineer has to modify the software binary code to inject the errors, there is a risk that the test unit may behave differently from the production software running in the field.

The techniques described above use real hardware, which limits the observability. They are controllable and repeatable, but because they use real hardware, the tests are never completely deterministic. All three techniques run fast enough (real time) to handle complex software stacks.

Simulation-based fault injection (performed at the gate or RTL level) has the advantage of having full access to all hardware elements on the system. Without being intrusive, it has full observability, controllability and determinism. However, the simulations are extremely slow, which makes them unusable on more complex fault scenarios where the design team must take the software into account.

Virtual Prototyping

A virtual prototype is a software model that emulates the hardware. Design teams can use a virtual prototype to model the digital aspects of a microcontroller unit, an electronic control unit or even a complete ECU network, and run the simulation on a desktop PC. Virtual prototypes run the same binary software as the real hardware. Because the virtual prototype is a soft model, software teams can get access to it months before the actual hardware device is available.
Virtual prototypes give development teams more than just software models for simulating the hardware; they are environments that allow them to debug and analyze hardware/software interactions. They also offer full visibility of the internal and external registers and signals, provide full control over program execution, and are completely non-intrusive.

Engineers can use virtual prototypes to freeze the full system execution at any point in time (even with multi-core hardware) and read and modify internal values. They can also use advanced analysis features to correlate software (at the application level) with hardware events, measure code coverage, apply fault injection, and use scripts to automate the simulations.

Virtual prototypes integrate seamlessly into existing software tool chains and connect to external third-party tools for hardware-in-the-loop and rest-of-bus complete simulations.

Teams can easily deploy and scale simulation models. Virtual prototypes are easy to share, archive and deploy across a worldwide organization.

**Fault Injection Using Virtual Prototyping Technology**

Virtual prototypes enable users to access all the internal hardware elements of a design—memory content, registers, signals—as well as specific, fault-tolerant mechanisms, like error correction codes (ECC) on memories, assuming, of course, that those features have been modeled. Users can create virtual prototype models without much effort by mirroring the functionality of the block at a more abstract level.

Virtual prototypes can model both transient and permanent faults. Users can inject faults through mechanisms on the simulation framework, without having to modify the embedded software models. They can also visualize and trace all hardware and software events that have been modeled on the systems. Visualization tools present both hardware and software execution and events on the same windows using the same

---

**Figure 1: Fault-injection environment**

---

**Fault-injection VP framework**

- **Controller**
- **Script**
- **Workload library**
- **Fault library**
- **Workload generation**
- **Fault injection API**
- **Built-in monitors analysis views**
- **Simulation control and inspection interface**
- **Virtual hardware model**
  - Memory
  - DMA
  - Timer
  - Core #1
  - Core #2
  - INTC
  - RTC
  - SPI
  - ADC
  - I/O
  - CAN
  - ECC error
- **Host PC**

**Fault injection API**

- Function call
- Register R/W
- Pin change
- Time out

- **Trigger**
  - Once/forever

- **Element**
  - Register(s)
  - Pin(s)
  - Memory content

- **Value**

**External tools**

- **Saber**
- **Simulink**
A basic fault-injection environment (Figure 1) includes the following elements:

- a target system—the virtual hardware model,
- a fault injector—injects faults from a library,
- the workload generator—creates stimuli according to the test scenarios,
- a monitor—feeds information back from the target system and the data collector and analyzer, and
- a controller—to orchestrate everything.

The fault injector and library are based on a simple fault-injection API to model fault-injection scenarios. The API has two basic commands: trigger and inject. The trigger command invokes an injector routine when a trigger event happens. Users can concatenate triggers to enable other triggers dynamically, depending on the system status. The inject command sets the specified element to a certain value. Supported elements are IO pins, registers, internal signals and memory locations. The value can be set just once (transient) or can be forced permanently. Users can add model-dependent commands for specific purposes (for instance, to flag an ECC error on a memory after a read/write access).

Fault-Injection Scenario: Data Abort Due to ECC Error

This soft-error scenario uses fault-injection to corrupt data on the SRAM memory during normal software execution. ECC functionality on the SRAM model triggers the data abort exception. The process is:

- The MCU is running an AUTOSAR software application
- Triggered by the internal interrupt line, the processor core goes into a standard, interrupt-service routine
- The first, next-access to the SRAM memory will trigger an ECC error back

Engineers can describe this scenario using the scripting facilities in the framework and the fault-injection API using just three commands and fewer than 10 lines of code.

When the core enters the software-exception routine, an error routine shuts down the OS. We can trace the fault injection and its effects using the built-in monitoring and analysis capabilities.

Summary

Fault injection, as recommended by the ISO 26262 standard, is the best method available to test the fault tolerance of hardware blocks and to ensure the effectiveness of diagnostic software. Virtual prototypes provide a complete framework to create advanced fault-injection scenarios. They offer more visibility and fault-injection points than hardware-based fault injection, can model both permanent and soft errors, and, unlike software-based fault injection, virtual prototype frameworks are completely non-intrusive. Virtual prototypes run orders of magnitude faster than RTL/gate level simulators.

Fault-injection frameworks that use virtual prototypes enable users to put errors under version control and automate fault-injection regression testing every time the software changes. Virtual prototype simulations have the potential to be used as evidence for certification and compliance with standards like ISO 26262.

About the Author

Victor Reyes is currently a Technical Marketing Manager in the System Level Solutions group at Synopsys. His responsibilities are in the area of Virtual Prototype technology and tools with special focus on Automotive. Victor Reyes received his MsC and PhD in Electronics and Telecommunication from University of Las Palmas, Spain, in 2002 and 2008 respectively. Before joining Synopsys, he held positions at CoWare, NXP Semiconductors and Philips Research.
Q&A
Ask our panel of experts

Prasanta Kumar Panda
Saber Applications Engineer

Marc Serughetti
Director, Virtual Prototyping Solutions

Jeff Garrison
Director, FPGA Synthesis Products

Q
The increasing complexity of the power electronics system—coupled with reduced budget and time for prototyping or physical validation—demands virtual validation. Because semiconductor switches are at the heart of a power electronics system, the characterization of these devices (MOSFETs, IGBTs, and diodes) plays a major role in virtual validation accuracy. What are the best modeling and simulation techniques available to characterize these semiconductor switches?

A
To create accurate models for semiconductor switches (which are constantly evolving!), one has to capture various kinds of parameters such as capacitance effects at switching, delays, rise time, fall time and so on. It becomes extremely difficult to capture the constraints of interdependencies or 2nd and 3rd order effects. Also, circuit designers may need to create specific switching models per the end-user’s requirement or according to the specific manufacturer’s data sheets. In such circumstances, circuit designers may not find the ready models inside many of the simulation tools and the only option is to create the models themselves. This can be a very tedious job with many modeling and simulation approaches.

One approach is to create a signal-flow model. This approach is difficult and breaks down when faced with the non-linear behavior of these types of devices. Another approach is to use a macromodel approach. Macromodels often simulate slower than equation based models and have convergence issues when a circuit operates at high frequency.

Saber users have found great value in its rich library of generic and characterized power electronic switch models. Saber models...
How can a Virtual Hardware-in-the-Loop environment help me in my development process?

Looking at the current approaches for development and testing of automotive systems, we first have Model-in-the-Loop (MIL) that comprises both a functional model of the control function and the plant model in a closed loop. This approach is mostly used by control algorithm designers. Software-in-the-Loop (SIL) is then used. It assembles the plant model with the C code of the function simulated in a host PC. This approach is limited due to the difference in the target hardware (a PC in this case). To overcome some these limitations before integration, Processor-in-the-Loop (PIL) can be used. The code is now executed on a development board for the target MCU. This approach has the challenges of requiring a complex hardware-based set up limiting its deployment. Finally, Hardware-in-the-Loop (HIL) is used once the ECU is ready. There is a significant time gap between the use of SIL/PIL and HIL. However, the main challenge is the effort gap to get to the fully integrated SW stack with the target ECU. Virtual Hardware-in-the-Loop are based on device physics equations, which overcome many of the limitations of the other approaches. Saber also offers an intuitive characterization process to capture the data sheet parameters and the relationship of the different characteristic curves between various parameters. These characterization tools are available for MOSFETS, IGBTs, diodes and several other devices. Imagine how much easier model creation could be if you could scan a graph directly from the data sheet into the modeling tool and use a built-in optimizer to create models! These tools provide a feature called “Scanned Data Utility” which allows the users to readily import device terminal characteristic graphs for modeling. The built-in optimizer then helps to match the model characteristics with experimental data. Saber provides flexibility to create a wide range of switch-level devices from logic-driven behavioral switches, on down to equation-driven device physics models. Users can start with logic-driven devices that do not require complex gate drive schemes, then equation based devices can be inserted as needed. Thus, users can create the required semiconductor switch models on the fly.

Please contact Synopsys for more information about Saber’s capabilities for modeling the latest power semiconductor switches.
Loop (vHIL) bridges this gap. It provides a complete simulation-based environment running on a PC and aggregating a simulated model of the ECU HW and a plant model (plus other environment parts). The same binary as the production binary can be used. It enables developers to start creating and running tests for the complete system earlier. Due to the controllability and visibility characteristic of simulation, more complex tests triggering difficult corner cases testing and fault-injection testing are now possible in a simpler way. Finally, since vHIL can be easily duplicated and automated to cover 1000s of system configurations, a regression environment can be deployed to accelerate and bridge the gap.

Synopsys virtual prototyping technologies serve the concept of vHIL. Contact us today to learn more.

Q
Why use FPGAs in automobiles, and for what functions are they used?

A
To date, FPGAs have been primarily used in automobiles for high-end entertainment and navigation systems where the volume of units and cost of such infotainment systems match requirements. However, as the amount of electronics in automobiles continues its rapid rise, FPGAs are also being used increasingly for safety-oriented functions, such as obstacle detection and collision avoidance. FPGAs are a great choice for these types of applications as the necessary signal processing algorithms can be implemented directly in the FPGA hardware which delivers the needed performance. While somewhat annoying, it’s probably less of a concern if your kids can’t watch a DVD in the back seat than if a collision avoidance system fails. For this reason, we are seeing the need for high-reliability design techniques for safety-critical automotive electronics. Automation for fault-tolerant state machines and even Single Event Upset (SEU) mitigation are becoming a concern in automotive, and, as a result, Synopsys has begun to address these needs in its Synplify family of FPGA synthesis products.

Have Questions?
Have a burning question you want answered? Submit questions to our panel of experts. Please send your email to atb@synopsys.com.
Recent Presentations

Addressing Automotive System Electrification Challenges with Physical Modeling & Simulation
www.synopsys.com/automotive-presentations

Synopsys FPGA and Simulink-Based Design Solutions for Automotive
www.synopsys.com/automotive-presentations

Upcoming Events

SAE Convergence
Detroit, Michigan USA
October 16 & 17
Visit the Synopsys booth!

Synopsys Automotive Seminar
Novi, Michigan USA
November 8

Additional Resources

SAE article: Virtualized Fault Injection Methods in the Context of the ISO 26262 Standard
http://papers.sae.org/2012-01-0001

Saber website:
www.synopsys.com/saber

SaberRD Student/Demo Edition FREE software download:
www.synopsys.com/saber-sw-demo

SaberRD Datasheet:
www.synopsys.com/saber-ds

Virtual Prototyping website:
www.synopsys.com/virtualprototyping

Virtualizer Datasheet:
www.synopsys.com/virtualizer-ds

FPGA Design website:
www.synopsys.com/fpga

Synplify Premier FPGA Brochure:
www.synopsys.com/synplifypremiervfpga

Automotive Solution website:
www.synopsys.com/automotive

Automotive Solution Datasheet:
www.synopsys.com/automotive-ds

Feedback and Submissions

We welcome your comments and suggestions. Also, tell us if you are interested in contributing to a future article. Please send your email to atb@synopsys.com.