



Automotive Electronics: Model-Based Development with Virtual Prototypes

July 2004

Introduction

Today's automobiles contain many complex electronic systems, each of which may incorporate a large number of electronic control units (ECUs) performing a single function, communicating through layers of networks. Even as complexity increases, design cycles are under pressure to shorten, so that manufacturers can deliver the latest in safety, fuel efficiency and convenience to consumers in a highly competitive industry. At the same time, quality and reliability remain of paramount concern. The challenges of managing these factors – complex technology, time-to-market and quality – are dictating a new approach to automotive electronics, yet one that is as old as the automotive industry itself.

Since its inception, the automotive industry has used models for the physical and mechanical aspects of vehicle design and development. Engineers build scale models of cars to predict how the real vehicles will look, feel and behave long before production begins. This tried-and-true method is equally appropriate for the design and development of automotive electronics. Modeling the silicon systems, ECUs and entire networks of ECUs in automobiles allows engineers to evaluate design alternatives, weigh tradeoffs and predict results long before the electronics are built. Modeling also simplifies troubleshooting and can reveal dangerous interactions between systems early, to prevent potentially disastrous results that may result in product recalls.

To answer this need for modeling, there is a new design paradigm for automotive electronics called model-based design, or electronic system-level (ESL) design. At the heart of ESL design methodology are virtual prototypes, which are software versions of the silicon systems, ECUs or networks of ECUs. Breakthrough technology has enabled modeling to be applied to the exploding area of automotive electronics as it has been applied in the mechanical world for years: technology with simulation speed and accuracy that allows the models to predict the behavior of the hardware long before the hardware exists. Modeling is also being used in wireless and consumer electronics, two other industries with high complexity, increasing software content and relentless shortening of time-to-market windows.

This white paper discusses the changes in automotive electronics that dictate the evolution to model-based design and virtual prototyping. Virtual prototyping is contrasted with bench testing to illustrate the specific differences. Other software-based modeling techniques are described to clarify the requirements of an effective model. Finally, virtual prototyping is explored in depth and an example of an automotive safety system designed using a model-based process is provided.

Growth of Automotive Electronics

The automotive electronics market has been growing faster than the overall electronics market and much faster than actual vehicle production. For the next several years, research predicts that automotive electronics will grow at a rate of more

Further penetration of electronic control into automotive systems, plus the growing sophistication of next generation systems will drive growth at 7.5% by 2007.

- Source: Strategy Analytics

World demand for OEM automotive electronics will grow 7.1 percent annually through 2007, driven by rising per vehicle electronic content.

- Source: The Freedonia Group

than seven percent. Over the course of this decade, the worldwide market for automotive electronics is expected to double.

Electronics systems, which account for more than 20% of a car's cost today, will increase to 30% by 2008.

- Source: Strategy Analytics

Software's share of car value will climb from 4% in 2000 to 13% by 2010—40% of the total value of automotive electronics.

- Source: Mercer Management Consulting

The percentage of an automobile's value attributable to the electronic content also is growing and is projected to reach 30 percent by 2008. In addition, a larger portion of the electronic content will be embedded software, tripling by the end of the decade. By 2010, more than 40 percent of the total value of automotive electronics will be in the software. Behind these increases is the reality that most ECUs will contain a huge amount of embedded software. Almost all the future technologies expected to appear in vehicles, such as drive-by-wire technologies and electrically operated valves, involve millions of lines of embedded software.

The Challenge of Embedded Systems

The explosive growth of electronics in the automotive industry, especially the growth of embedded system software, changes the dynamics of automotive design and presents significant challenges. Now that there are so many ECUs – up to 70 in a vehicle, connected by up to five buses – project managers and engineers are faced with new issues arising from how those ECUs behave internally and how they interact with each other. Warranty and quality issues hinge on being able to diagnose difficult hardware/ software problems within and between ECUs. Bus communication between ECUs is complex and critical.

These new challenges escalate when coupled with the demands of a highly competitive industry. Automotive OEMs and Tier 1 suppliers who can bring new functionality to market in the areas of safety, fuel efficiency and convenience will reap the benefits of market share. But this new functionality must meet stringent quality standards, at competitive costs. The cost/ quality dynamic is very difficult given the increased complexity of ECUs, the interactions between them and the software proliferation within them. A BMW executive reported at the Jan 2004 automotive electronics conference in San Diego, California, that 50-70% of the development cost of an ECU is related to software and already 40% of a vehicle's cost is determined by electronics and software.

According to an Embedded Market Forecaster's study in April 2003, 54% of embedded systems projects are late. Four of the primary reasons were inadequate specifications, specification changes, application complexity and too few developers and testers. All these factors clearly contribute to the rising cost of automotive electronics.

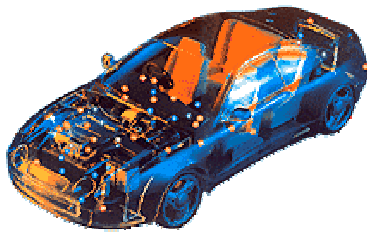
The combination of market dynamics and increasing hardware and software technology is resulting in a broken process for embedded system development in automotive electronics design. The confluence of these factors is driving the change in automotive electronics design methodology to model-based design, using virtual prototypes. The traditional hardware-based bench development methodology no longer suffices. Car manufacturers realize the difficulties of

changing to an embedded systems approach, and are showing a new level of attention to embedded software development success in other markets. For example, General Motors reportedly has two major, high-level strategic programs: fuel cells and embedded software.

The Growth of Electronic Control Units and Networks

ECUs used to be relatively simple, hardware-oriented systems. Today, they are multi-purpose, multi-chip computer systems where more functionality often is delivered in software than hardware. The most complex ECUs operate the powertrain. Simpler ones operate functions such as seat and mirror adjustments, but even these ECUs need to be networked so that the seat and mirrors can be adjusted for different drivers. The massive network requires a wiring harness that can weigh over 50 Kg; its many connectors contribute negatively to automotive reliability and hence to increased warranty costs.

The trend of increasing automotive electronic content is the direct result of many new features that will greatly increase both safety and comfort but that will require more sophisticated ECUs with a large embedded software component. The safety features include steer-by-wire, brake-by-wire and drive-by-wire (collectively known as “X-by-wire”), automatic lane-following, drowsy driver detection, intelligent cruise control and airbag systems that can adjust deployment based upon passenger weight and the specific nature of an accident. Improving fuel efficiency is an important goal: hybrid (internal combustion and battery) and fuel-cell electric drive place high demands on software.



Telematics and in-car entertainment will further increase the electronic content of cars, requiring the combination of such technologies as wireless connectivity, global positioning, digital radio and Internet access, all with hands-free voice activation whenever possible.

Various factors work against each other to create a complex optimization problem. To reduce wiring weight, ECUs should be located close to whatever they control: for example, the engine control ECU should be mounted in the engine compartment and brake controls should be close to the wheels. This tends to proliferate ECUs, with each performing a few specific functions. On the other hand, to reduce connector count, the number of ECUs should be as low as possible. But to keep reliability up, inter-ECU wiring must be kept to a minimum and redundancy for the most critical functions must be built into the system. The network connecting the ECUs needs to guarantee service for functions such as X-by-wire.

Resolving this series of contradictory requirements requires a more systemic approach to the architecture of the electronics system. Many potential trade-offs must be evaluated before the overall hardware/software system is defined and detailed development can begin. Evaluation, in turn, requires the capability to do quantitative analysis of potential architectures of both networks of ECUs and of the internals of the ECUs themselves.

The Old Way: Bench Testing

The traditional way to develop automotive embedded systems has been to build hardware boards that represent all or part of each ECU and part of its surroundings, often called plant models, and use them for bench testing. Usually several different bench setups are required for complete development since it is rarely possible to make all the desired measurements from a single setup.

It is not practical or cost-effective to wait until the chips arrive to begin software development. Bench testing allows software engineers to begin development before the actual hardware, including silicon, is available. This has helped reduce both the high cost and long manufacturing time needed to develop the semiconductors for automotive electronics, because hardware and software engineers can work interactively during the development cycle. Starting software development after hardware is essentially complete increases the likelihood of serious hardware errors that would cause a chip re-spin costing millions of dollars.

Unfortunately, the bench approach has many limitations. First, creating all the needed hardware boards is costly. Companies developing the software may be able to transfer this cost to the semiconductor vendor as part of the business arrangement, but the development cost still ends up reflected in the final product. Second, the performance requirements of the most powerful ECUs (those used for powertrain control) are so demanding that it is no longer possible to build boards that allow adequate measurements to be taken. Even when boards do exist, observability within the ECU ranges from difficult to impossible, preventing engineers from debugging problems in the hardware. Since hardware doesn't generate error messages, simple errors such as an uninitialized register can take a long time to track down. Debugging software on a hardware prototype, even with in-circuit emulators, is increasingly difficult and time-consuming. This situation will only get worse as automobiles incorporate more features requiring complex embedded systems.

The hardware approach does not scale. Even today, it often takes three to four spins of silicon to reach the desired level of performance and reliability. Development teams are large and geographically dispersed, and they need a centralized software system to debug various parts of the system under development. A particular car that lasts many years may have as many as 1000 different ECUs over time, all of which need to be available for software development in case legacy problems emerge or in case updates to more modern technology are needed as new models are introduced. It's a management nightmare to make sure that enough boards are available at the correct revision levels for each ECU, especially with dispersed programming teams.

Finally, and most importantly, this bench testing approach is based on a sequential design process where hardware is developed, plant model prototypes are built and software development begins. With software now representing so much of the content of ECUs, it's critical that engineers explore architectural options as part of an ECU design. Should an ECU use one or three processors? Should a critical control capability be implemented in software, for flexibility, or hardware, for speed? How will that impact overall ECU performance, reliability and cost? These tradeoffs must be evaluated up front, before a hardware board is built and silicon is specified.

Architectural tradeoffs can make or break a project. Carefully and quantitatively evaluating them prior to hardware and software development reduces project risk and improves development efficiency. The potential for re-work is dramatically reduced. Furthermore, once the architecture is defined, hardware and software development can proceed concurrently, allowing hardware and software engineers to work together and reducing overall project development time.

Software Approaches Offer Some Improvement

To be useful, any embedded software development system or test environment must deliver both speed and accuracy. Speed is needed to run large bodies of software, including real-time operating systems (RTOSs) and network protocol stacks. Accuracy is needed so that measurements of real-time performance or detailed hardware interaction can be used to predict correct performance in the vehicle.

Two software approaches in use today are improvements over hardware bench testing for software development, but both have problems. One, the “host-based approach,” gives speed but little accuracy; the other, the “ISS approach,” delivers accuracy but little speed.

The host-based approach is a “write-once/port-twice” methodology, in which the software is developed on a host PC within a test scaffold, often with an operating system simulator, and then ported to the ECU. This approach works well for software that has negligible real-time dependency and that does not interact intimately with the hardware, but it breaks down in an automotive environment where there are hard real-time constraints on the software. It is impossible to predict actual performance from the PC implementation. The approach fails totally for software that interacts closely with the hardware, such as device drivers or operating system modules, since the hardware is not modeled at a level detailed enough even for the code to run.

The ISS (instruction set simulator) approach uses a functional model of the processor, usually supplied by the processor vendor. Such an approach varies in the fidelity of hardware modeling accuracy; sometimes the model does not match the cycle-by-cycle behavior of the processor. Even if the model is accurate enough for a detailed analysis of real-time performance and hardware interaction, speed is a major problem. ISSs are too slow, typically 100-500 KIPS (0.1-0.5 MIPS), to run a realistically large body of software, and much too slow to be a part of every software engineer’s day-to-day edit-compile-debug cycle.

The Best of all Worlds: Virtual Prototypes

A new approach is now available that delivers speed and accuracy, giving embedded software developers the benefits of both host-based development and instruction set simulators without the drawbacks. This virtual prototype approach enables system architects, hardware engineers and software developers to build a system-level model or virtual prototype of an ECU or a subsystem of networked ECUs. The virtual prototype is a software version of the hardware design that runs the actual real-time embedded software. It is changeable and configurable to explore architectural options and measure results.

Using the virtual prototyping approach, the architecture is established by quantitatively evaluating potentially hundreds of candidate architectures. Hardware engineers then use the virtual prototype as a golden reference design – an executable specification for the ECU. Software engineers use it as a development platform, much like the plant model, but with better visibility into internals, more measurability and more controllability for debugging software than is possible with hardware. Software engineers do their software development entirely on the model until the final integration stage, when the software is moved to the real ECU hardware.

VaST's virtual prototype technology can simulate an embedded processor at 15-200 MIPS on an off-the-shelf PC – fast enough to be used in the day-to-day work of an embedded software engineer. Because the processor models can be cycle-by-cycle accurate, the engineer can test full interaction with the hardware and make detailed performance measurements of the software and the hardware/software interface.

The virtual prototype offers complete transparency to the ECU internals -- a much better debugging environment than hardware-based approaches. In addition, error checking in the models can catch many simple errors that would cause unpredictable behavior, or take a long time to discover, in hardware. As a result, companies can produce more high-quality software, with fewer resources, faster than ever before.

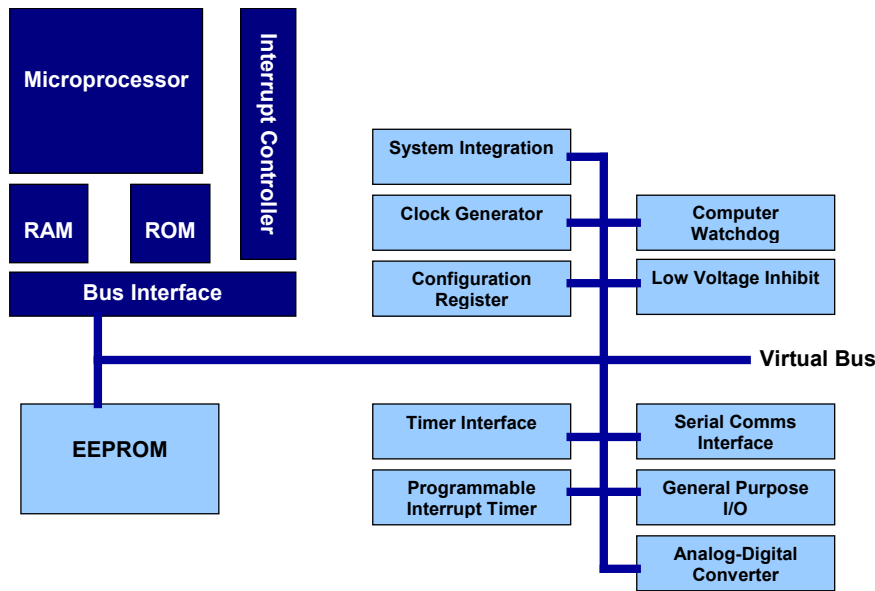
Virtual prototypes also are highly beneficial for hardware development. They enable relatively high-level, but accurate, modeling of the hardware components. As the hardware engineers develop the specific implementations of the semiconductors and other components, more detailed models can be brought into the virtual prototype environment to ensure consistency between different levels and models.

Virtual prototypes also are easy to update and distribute. With today's geographically dispersed development teams, it is much simpler to transmit software models over networks than to ship boards, with all the associated customs and duty issues. With virtual prototypes, everyone on the team can have his or her own model and it can be up to date with current system revisions. This methodology provides engineers with a dramatic productivity increase as well as quality improvements because everyone is working from the same model.

Finally, and most importantly, virtual prototypes enable hardware and software design and development to occur concurrently, rather than sequentially as with the traditional methods. Once the architecture is finalized, it is implemented in the virtual prototype, which is distributed to hardware teams as a golden reference design and to software teams as a development platform. Thus development proceeds in parallel and the teams can interact. Should design issues arise, the teams can work together to address them long before hardware is finalized. This concurrent hardware and software engineering offers significant benefits in time-to-market as well as quality.

Example: Rollover Detection ECU

This example involves an ECU that forms the heart of an automotive rollover detector, based upon an actual design done by a customer of VaST. The ECU takes as input an analog signal coming from a rotational accelerometer that is mounted transversely in the vehicle. If the vehicle starts to roll over, this rotational accelerometer signal can be integrated mathematically to calculate the vehicle’s angular velocity and its position. In the event of danger, side airbags are deployed in the vehicle to protect the occupants. The timing of airbag deployment is not straightforward. If the vehicle is rolling over quickly, then the airbags are deployed immediately; if the vehicle is rolling over slowly, then deployment is delayed so that the airbags have not deflated before they are able to cushion the occupant.



In the real world, such an ECU is complicated by many peripherals, some of which must be modeled within the virtual prototype to ensure that the full range of hardware/software interaction is tested. Clock generators, memories, timers and an interrupt controller are present, in addition to the expected interfaces to the accelerometer and airbag igniter.

For reliability reasons, the processor is monitored by a “Computer-Operating-Properly” watchdog timer that will assert the reset line if it times out, indicating that the processor is in an unexpected loop. In addition, the “Low Voltage Inhibit” peripheral monitors the supply voltage to prevent airbag deployment in the event of a dead battery or during the startup phase just after the key has been turned in the ignition.

This system has hard real-time constraints. The time from detecting a rollover to deploying airbags must be known accurately. But there are many other real-time issues, from ensuring that the watchdog timer never expires, to sequencing operations during startup so that the system comes up cleanly.

Using the virtual prototype approach, the engineers on this project were able to

Problems found with virtual prototype of rollover detection ECU:

1. Computer-Operating-Properly watchdog reset processor erroneously
2. Accelerometer device-driver read wrong register
3. Software segment not being initialized to zero caused an OS halt
4. Memory-fail register was not being reset following memory error
5. Math overflow problem caused late deployment of airbag by one second
6. Math overflow problem in timer code
7. Stack overrun
8. Bootstrap switched to PLL clock before PLL was running
9. Programmable timer initialized incorrectly
10. Asymmetrical rounding errors

integrate the RTOS and all the peripheral drivers, and then test the airbag deployment algorithms. The adjacent box lists some of the problems discovered while developing the software for this ECU. The most serious was problem #5, which resulted in airbag deployment one second late. This is clearly unacceptable since such a late deployment would provide no protection in a high-speed crash.

It is unlikely that any of these problems would have been discovered with a host-based approach using an operating system simulator, except perhaps for the stack overrun. Several of these problems involve real-time software requirements or require accurate modeling of the interaction between hardware and software. The accuracy of a virtual prototype was needed to find the full range of problems.

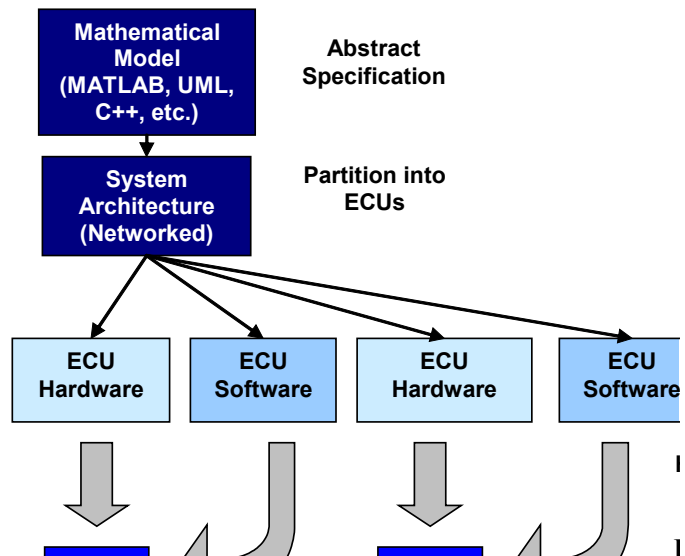
Although bench analysis likely would have been more successful than a software approach for this particular project, it would not have been as efficient or effective as using a virtual prototype. The engineers on the customer’s team estimated that they saved six months -- almost half of the development schedule -- by using the virtual prototype approach as opposed to the bench approach that they had used on previous development projects. Furthermore, they reported that several of the problems they detected would not have been found using the bench approach. For such safety-critical functions, quick, thorough and accurate testing is absolutely necessary. It is not hard to imagine the danger to occupants, expensive recall and loss of credibility that could have been suffered if the bug causing a one-second-late airbag deployment had not been discovered.

Quantitative Architectural Analysis and Optimization

The growth in embedded software content, combined with the continued demand for ECU reliability, creates a huge challenge, but also an opportunity. Forward-thinking automotive companies, seeing this as a complex embedded systems rather than just a hardware problem, are adopting model-based designs by building mathematical models of subsystems well before implementation. However, such models still require the creation of large amounts of software and an efficient network of ECUs to execute the code and interface to the mechanical systems of the vehicle.

This complex optimization exercise requires a change in methodology to a more systemic or architectural approach that enables quantitative analysis, both of potential architectures of ECU networks and of the internals of the ECUs themselves.

As previously mentioned, the virtual prototype approach provides a solution here, too. By offering both speed and accuracy,



system architects can run a full software load on the ECU or networked subsystem and make measurements of performance, network traffic, and other metrics that exactly reflect the real hardware.

Systems architects also can perform very high-level simulation using the abstract specifications expressed in mathematical or formal terms in languages like MATLAB, UML, SystemC, high-level C or C++ models. Once the basic behavior is correct, analysis can be done to allocate the functionality among a number of ECUs and investigate the performance requirements that the partition requires of the ECUs and the interconnecting network. Finally, each ECU can be architected at the level of processors, buses and peripherals, and functions can be moved between hardware and software to reach the desired performance point.

The efficient nature of the quantitative analysis allows the systems architects to experiment with different partitions among different numbers and kinds of ECUs, evaluating the trade-offs to determine the optimal implementation. It is sometimes said that engineering is primarily the study of trade-offs; nowhere is this more true than in the complex interactions between the numerous hardware and embedded software portions of an advanced automotive electronics system. The challenges of optimization simply cannot be met without the breadth and depth of the features offered by a virtual platform.

In a typical design and development project, some of the ECUs and much of the software will be inherited from earlier projects. Where this is not the case, such as in the first design of a new brake-by-wire system, the architecture of the ECU and the creation of the embedded software can be done concurrently, leading to a better implementation of both. Once the desired performance seems certain, the hardware can be designed in parallel with writing and debugging the remainder of the embedded software. Virtual platforms foster reuse of both hardware and software, making it easy to integrate existing functions so that engineers can concentrate on developing new features.

Summary

Automotive ECUs, and networks, are growing in number and complexity. The combination of complex ECUs and networks with an increase in embedded software, in particular, presents a requirement for a change in automotive electronics design methodology.

Automotive companies who want to address competitive pressures for delivering new functionality in safety, fuel efficiency and convenience are adopting an ESL-based approach to ECU design. This new method, called model-based development using virtual prototypes, enables rapid adoption of new technology while maintaining high quality and reliability standards and meeting time-to-market delivery pressures.

Virtual prototype technology offers a range of capabilities, from architectural design, analysis, and verification to a detailed embedded hardware and software development and debugging environment. In addition, the virtual platform approach offers many advantages over the outdated bench-based methodology. It enables architectural optimization as well as concurrent hardware and software development, and provides superior visibility into system internals.

Automotive electronics systems produced using virtual prototypes have higher quality and reliability, greater engineering efficiency, and can be done within a shorter overall development schedule.

The most advanced companies, including the world's #1 automotive manufacturer and the world's #1 Tier 1 automotive parts supplier, are switching to virtual prototypes as the technology of choice. As the positive experiences of the top-tier companies becomes more widely known, virtual prototypes will become the dominant development methodology for automotive electronics.