

# Design Talk - Prototyping and Design

*There are always things to consider when prototyping a design. Here are a few essays on the subject to aid you in your efforts.*



## **Award-Winning Model Based Design at Challenge X Hybrid Vehicle Competition By Craig Pavlich, Ohio State Center for Automotive Research**

In large scale, or highly integrated systems design, getting a handle on the complex interactions between the disparate system components can be difficult. Furthermore, large scale systems are often expensive, allowing for limited prototyping, and the various design layers (electrical, mechanical, controls, etc.) are often handled by separate expert sub-teams. One method that many design teams involved in complex designs choose is to use model-based design as part of the process. In simple terms, model-based design places equal importance on the development of a highly accurate mathematical system model along with the development of the physical product. The team at Ohio State University used model-based design as a strategic differentiator in the Challenge X Hybrid Vehicle Competition. By basing the development process in modeling and simulation, the Ohio State University team developed a fully functional, alternative-fuel vehicle using only a single prototype and a very small team of relatively inexperienced students.

### **Model-Based Design: Advantages**

Before examining how the Ohio State University team used model-based design, we should look at the advantages that model-based design holds for these types of complex systems. In any product development cycle, typically some decisions are made about the initial design specifications and requirements. In terms of our

## Design Talk - Prototyping and Design

Published on Electronic Component News (<http://www.ecnmag.com>)

vehicle project team, these requirements included: vehicle performance, fuel economy, emissions reduction, cargo capacity, drive train smoothness (driveability), and passenger amenities.

We started the competition from the outset with a plan to develop our vehicle using model-based design on software tools provided by The MathWorks. Using this approach, as a first order of business our team needed to formally quantify the design inputs. The selection of quantitative design inputs -- such as expected vehicle duty cycles, minimum fuel efficiency, or emissions limits -- aided the team in identifying some basic design parameters, but more importantly, the process allowed our inexperienced team to highlight the unknowns in our design's intent. In essence, by gaining a solid handle on all of our input unknowns, our team was able to conduct information gathering missions and critical experiments very early in the design process, ensuring that our design decisions going forward were based on solid science.

With the design intent squared away, the competition between vehicle concepts could be undertaken. Model-based design offered the architecture team several key advantages:

- A quantitative and equitable platform for the evaluation of competing designs.
- The ability to apply well understood component models developed on prior vehicle project teams, such as engine or electric machine experimental data, directly to their conceptual designs. This application of real-world data can improve confidence in initial results greatly.
- A reduction in wasted effort on design-stage calculations. The design-stage model, as utilized to make critical early design decisions, will continue to be refined alongside the resulting prototype, to act as both a "software prototype" and project documentation for future use.



While much of our early design process was occupied with the careful selection of a high-level vehicle drive train architecture, encompassing not only performance metrics, but with considerations for technical feasibility and component availability as well, our modeling process did not stop once our development plan was completed. Similar to many large scale projects, it was timing critical that both component work and systems integration

proceeded continuously, in spite of the fact that a complete physical prototype would not be available for nearly two years from the project's inception. First, the competition's format dictated that we would not receive our platform vehicle chassis (Chevy Equinox) until the end of year one (of four years), and that much time would be required to assemble the core components once the vehicle did arrive. Second, we knew that any detail work that could proceed independently would provide a jump on the tight competition. This challenge was met by parceling out component modeling and design tasks to various individuals and sub-teams over a number of years. Individual student sub-teams had the ability to study and test subsystems, developing detailed models of each component's behavior, and where applicable, the subsystem's controls system. The application of advanced computing made integrating the subsystem into the complete system model simple and seamless, offering our team both optimal allocation of human resources as well as the development of a highly-accurate model of one of the overall hybrid electric vehicle.

As the physical project vehicle began to come together late in year two, the real advantages of model-based design began to become more and more apparent, particularly in the realm of the vehicles various component and supervisory controls systems. With the need to hurriedly assemble a running vehicle for the spring competition deadline, we knew that there would be scant time for testing and refinement of our vehicle's controls system in operation. Here our model's validity would truly be tested, as our initial controls algorithms were developed solely in software, utilizing our developed overall vehicle model and our independently tested components. Our model-based approach allowed us to employ tools from dSpace, Phytex, Mototron and The MathWorks for actual controls implementation. Utilizing the capabilities of Simulink's Real-Time Workshop software from The MathWorks made control transfer from modeling to application simplistic, as control code developed in simulation could be ported almost directly onto the vehicle controllers with little or no adaptation. In our team's case, employing an army of software developers to translate high-level algorithms into machine code was simply not an option. With modern auto-code generation and available third party development and interface tools, such tedium is no longer required to produce reliable code. As a testament to the model based system's capability allow rapid development of both the control algorithm and the physical vehicle, we were able to deliver a running vehicle in year two, having had the whole thing assembled and functioning for less than two weeks at delivery.

### **Vehicle Architecture**

Our team settled on a split-parallel hybrid architecture, in which the biodiesel-fueled engine and belted starter-alternator (BSA) drives the front wheels, and a large electric machine drives the rear wheels independently. This arrangement allows for a multitude of driving modes by combining the various power sources and actuators at various times. For example, to improve stop light or traffic-jam fuel economy, the vehicle can operate in an electric-only mode, employing the rear electric motor alone at low speed. Conversely, on the highway, perhaps running the engine alone would be most efficient. In various mid-range speed conditions, some combination of several actuators may be employed. At wide-open throttle, all of the power may

be applied. In all driving conditions, these various modes are applied automatically, with the driver's only input being a traditional accelerator pedal.

Clearly, a complex control algorithm is required to get the best out of all of these systems over the wide variety of situations the car might encounter. Though in year two a running prototype was presented, and many of the physical components were in place, most of these systems were not well refined and were far from optimized. Over the following year and a half, the model-based approach began to show its true potential. With the myriad components and embedded controllability, the number of variables to be tested and tweaked was astounding. Clearly, testing every incremental mechanical and control improvement would be impossible with only a single prototype and our small team. Instead of test-driving endlessly, our team was able to simulate literally millions of possible control permutations utilizing our model and narrow our focus to testing only a few iterations of high potential. Test data collection could occur over a relatively few measurement and test-drive sessions, preventing the controls team from tripping over the mechanical development and vice versa. In cyberspace, the Ohio-State ChallengeX Equinox has covered millions of miles, burning thousands of gallons of diesel in pursuit of computer-controlled perfection, while our exhausted team had only to cover a few thousand miles over the course of the year.

Finally, Ohio State's success with the ChallengeX project is not terminal with the competition's close. Beginning in Autumn 2008, Ohio State will be participating in the Department of Energy's follow-on competition EcoCAR. The modeling and experimentation expertise, controls strategies, and even some of the individual components and component models validated over the course of the entire ChallengeX project may be applied directly to the EcoCAR program. Acting as a powerful experience database, our Model Based Design ensures that Ohio State's students are free to push the envelope of vehicle design going forward, offering Ohio State a competitive edge in what is sure to be a tight contest.

### **Competition Background**

Challenge X was a four-year competition among seventeen North American universities with the goal of re-engineering a 2005 Chevrolet Equinox for improved fuel economy and reduced emissions while maintaining performance, utility, safety, and consumer acceptability. Headline-sponsored by the Department of Energy and General Motors Corporation (GM), university teams achieve the goals of the competition by developing custom advanced hybrid powertrains and innovative emissions control techniques on an existing Chevy Equinox platform chassis. As winners of the Mathwork's Model-Based Design Award in 2008, the team presents their experience with model-based design in the development of their hybrid electric vehicle.



### Rapid Prototyping and Design

By Paul Nickelsberg, Orchid Technologies Engineering and Consulting  
[www.orchid-tech.com](http://www.orchid-tech.com) [1]

Rapid design and prototyping of board level electronic products requires high-attention to detail and collaboration between the circuit designers, circuit board fabricators, circuit board assemblers and testing people. Successful rapid prototyping and design begins with a vision of the entire process. The project leader must understand both the technical challenges and the inter-personnel team needs. The project leader must work with an eye toward the future - always thinking how will my actions now affect the on-time and accurate project outcome. These skills extend beyond mere technical competence. Successful rapid prototype design is 50% technical skill and 50% project management.



When working on design projects with rapid prototyping schedules, we believe it is best to work out the details of the circuit design while starting component procurement and circuit board layout activities all at once. The goal is to move through the design process as quickly as possible so as to get to the hard parts all the more rapidly. While designing, the goal is to uncover technical difficulties as quickly in the process as possible so as to address these difficulties in a pro-active manner.

While performing simultaneous detailed circuit design and component procurement, we may find that a certain part is not available within our time frame. Finding that

out early gives us the option of replacing that part with another more available device. When in rapid-mode schedule is king and component availability is a critical concern.



### **Bug Prevention is Better Than**

#### **Debugging**

**By Fergus Bolger, PRQA**

[www.programmingresearch.com](http://www.programmingresearch.com) [2]

Einstein once claimed that 'intellectuals solve problems, but geniuses prevent them'. Adopting this approach would turn current software development practice on its head and place greater emphasis on eliminating the source of coding problems at the developer/coding phase rather than aiming to catch bugs in the integration phase. This approach will resonate with any organization that finds considerable portions of their software engineering resources trapped in debug cycles, and is one which is driving the development of Coding Standard Enforcement (CSE) tools.

CSE tools are designed to prevent and detect, software bugs. A typical example is a company which had spent hundreds of engineering hours trying to locate the cause of a memory crash before asking Programming Research (PRQA) to carry out an on-site code CSE audit. Within half an hour, the audit had found that the root cause of the problem was inconsistent memory allocation methods being used on the same global object accessed by different system modules. The bug, and the hours of debugging, could have been prevented by following a set of coding rules which would constrain, but not ban, the use of dynamic memory allocation.

In another typical scenario, a recent switch to 64-bit hardware seemed to be the

cause of erratic system behaviour, whereas PRQA's analysis tools revealed that the true cause of the problem was unwanted conversions between pointer and integral types. This category of software fault could have been prevented by establishing a set of coding rules to identify and focus on such conversion issues. There are, of course, already ISO standards in place which define C and C++ languages, however, only about 70% of C language is strictly defined, which means there are still elements of unreliability or uncertain behaviour within the code. Using the right software tools to identify and prevent problems early in the development cycle is vital, and explains why companies developing critical software systems invest huge amounts of money to safeguard their software development.

Awareness is growing that resources devoted to late-cycle fix-and-retest approaches to achieving software correctness could be more effectively allocated to prevention. CSE tools provide users with the ability to adopt an existing coding standard or develop their own custom standard, and subsequently automate enforcement. For some customers, CSE is necessary in order to comply with safety or quality standards, whilst others use it to avoid known problematic areas of the chosen programming language, handle the wide array of portability issues within the C or C++ languages across compiler dialects or hardware platforms, or simply to achieve best-practice in software development.

Contractual requirements may also stipulate the use of CSE where, for example, an OEM is outsourcing its software development and requires the supplier to meet certain quality levels of software development. Here, CSE acts as a gatekeeper, checking all incoming source code against the specified standard. Code reviews, unit and system testing, including coverage analysis, functional verification steps, and feature upgrade cycles can all be considerably shortened by using CSE. Bug-ridden software does not magically heal itself in the next design iteration; it simply transfers its problems elsewhere. The goal, therefore, should be to empower software engineers to prevent mistakes, or to identify and correct them before they reach the test environment.

The follow-on benefits from a prevention-oriented approach to code construction can be as high as 50%. CSE protects software developers from using parts of the language that are known to give rise to problems. It can prevent straight-forward coding mistakes, misuse of language and unexpected behaviour which can result when language definitions are not sufficiently strict. PRQA's tools analyse source code on a file-by-file basis and also across a complete project to identify potentially dangerous usage of language. The tools also identify language usage which is not compliant with the relevant ISO standard.

In the QAC and QAC++ tool suites, comprehensive libraries of selectable warning messages are used to highlight source code which is non-portable, difficult to maintain, overly complex, or written in any way that is likely to cause problems. A message browser displays diagnostics on source code and associated header files for the complete project. These are categorised and grouped across all source files and can also be viewed as an annotated source code listing, with the option to view in HTML format with links to additional information and advice.

## Design Talk - Prototyping and Design

Published on Electronic Component News (<http://www.ecnmag.com>)

---

As embedded software systems become increasingly complex, companies are beginning to consider C++, rather than C, as their embedded language of choice and dealing with technologies that allow them to abstract away from the C and assembler-level code that they might have been working with for 10 years or more. This will necessarily mean that even greater levels of protection and care with software implementation will be required. These changes mean that the industry has reached a turning-point in its approach to assuring software quality. The past three years have seen a huge rise in the acceptance of the need for CSE and companies are taking more proactive control of the whole issue of software quality. Debugging is increasingly being seen as a waste of time and effort, whilst coding standard enforcement, or pre-bugging, is a significantly more efficient use of resources.

By adopting 'best practice' coding standards, organisations not only improve their time to market but also reduce rework and re-testing. Fewer bugs in released software helps to support their reputations, whilst improving customer satisfaction and limiting the risk of litigation in case of system failure. It appears therefore that companies do not need to use their intellect to solve problem bugs, but simply have the genius to prevent them by using Coding Standard Enforcement (CSE) tools.

### **Source URL (retrieved on 11/25/2014 - 11:09pm):**

<http://www.ecnmag.com/product-releases/2009/05/design-talk-prototyping-and-design>

### **Links:**

[1] <http://www.orchid-tech.com/>

[2] <http://www.programmingresearch.com/>