2008-01-1469

Best Practices for Verification, Validation, and Test in Model-Based Design

Brett Murphy, Amory Wakefield, and Jon Friedman
The MathWorks, Inc.

ABSTRACT

Model-Based Design is no longer limited to R&D and pilot programs; it is frequently used for production programs at automotive companies around the world. The demands of production programs drive an even greater need for tools and practices that enable automation and rigor in the area of verification, validation, and test. Without these tools and practices, achieving the quality demanded by the automotive market is not possible. This paper presents best practices in verification, validation, and test that are applicable to any program, but are critical when applying Model-Based Design in production programs.

INTRODUCTION

Moore's law holds that the number of transistors in an integrated circuit doubles every couple of years. Thanks to this trend, the amount of software embedded in automobiles, aircraft, consumer electronics, and many other types of systems continues to grow. In the past, engineers developed software using traditional development tools such as editors, compilers and debuggers. But as the size and complexity of the embedded software programs grew, these tools and the development processes they support fell short. The challenges were particularly acute in the development of safety-critical embedded systems, many of which are found in automotive and aerospace applications.

Model-Based Design helps address the challenges of embedded system development [1][2]. Using models at the core of the development process provides engineers with insight into the dynamics and algorithmic aspects of the system through simulation. In addition, the models are commonly used:

- as executable specifications
- to communicate (sub-)system requirements and interface definitions
- to provide virtual prototypes or models of the complete system
- for automatic code generation of embedded software algorithm or logic

One of the most valuable aspects of Model-Based Design is the availability of executable models to perform Verification, Validation, and Test (VV&T) throughout the development process, especially its earliest stages. A study conducted by NASA on Internal Verification and Validation found that a large number of errors discovered in the testing stage late in development processes are actually introduced at the beginning of the process as requirements errors. Moreover, fixing these requirements errors in the testing phase was more than 10 times more expensive than if they had been found earlier in the process during the design phase [3].

In our experience, organizations attempting to ensure a rigorous development process using Model-Based Design need to define and establish effective VV&T activities, especially early in the development process. This paper presents a fairly common (and perhaps common-sense) set of best practices found across a number of organizations with rigorous processes.

DEVELOP MODEL TESTS WITH THE DESIGN

Through modeling and simulation, embedded developers are able to design, visualize, and debug larger amounts of system logic than by using traditional development methods. Many engineers, however, do modeling and simulation in an ad-hoc manner. They use their judgment, and perhaps a design specification, to select the conditions used to simulate their designs. They may capture and document results, but the simulations are often difficult to replicate later. For example, if a design error is found during code testing, it is often difficult to replicate the same test on the model to determine the root cause of the error and provide a complete and verified fix.

The increasing complexity of embedded systems and the increasing need for development standards in building safety-critical systems are driving development groups to use more systematic processes [4]. This growing need for systematic techniques holds for simulation as well. To this end, development groups using simulation techniques are developing and applying

test suites to the models and then reusing these suites to test the software implementation. These tests include both the initial conditions and input sequences for the simulation, as well as the expected outputs for the test. Well defined tests also have a description that explains its purpose, the requirement it is testing, or both. The entire suite is developed as a test harness that can be executed repeatedly in simulation. This process is often referred to as model-in-the-loop (MIL) testing.

As shown in Figure 1, the best practice is to reuse the test harness developed for MIL testing against the software implementation of the model on the host development environment. This practice, known as cosimulation of the code or software-in-the-loop (SIL), provides confidence that the code matches the desired behavior developed and specified in the model. Because the test harness is the same in MIL and SIL testing, it is easier to compare the output of both tests and replicate any identified errors. Often engineers will "elaborate" the set of tests used in SIL, adding tests that cannot be performed in MIL testing. In addition to ensuring there have been no changes in behavior from stage to stage, SIL tests are used to ensure that the development step did not introduce errors.

Tools are now available to take this process one step further by testing the software after it has been compiled and downloaded to an embedded target or processor. This cosimulation step is often called processor-in-the-loop (PIL) testing. PIL tools provide a method to execute the tests - originally created on a development host computer like a PC – on the software while it is running on an embedded processor. One technique uses the host-to-target communications mechanism provided by the embedded Integrated Development Environment, which engineers use to compile and download the code on to the target. With this technique, tests or simulations running on the host development computer communicate synchronously with the code running on the embedded target, enabling engineers to run the tests against the code on the target. Again, using the same test harness makes it straightforward to compare PIL results with the original MIL results. This comparison of test results between the embedded code and the original model gives engineers confidence that the behavior of the component has not changed after compilation and download and that the code is functionally correct.

As noted previously, errors become increasingly more costly the closer to production they are found. A best practice in VV&T for Model-Based Design is to develop model tests with the design. Developing a test harness that can be used for MIL, SIL, and ultimately PIL testing helps developers find errors early in the process and ensure errors have not been introduced during implementation or integration.

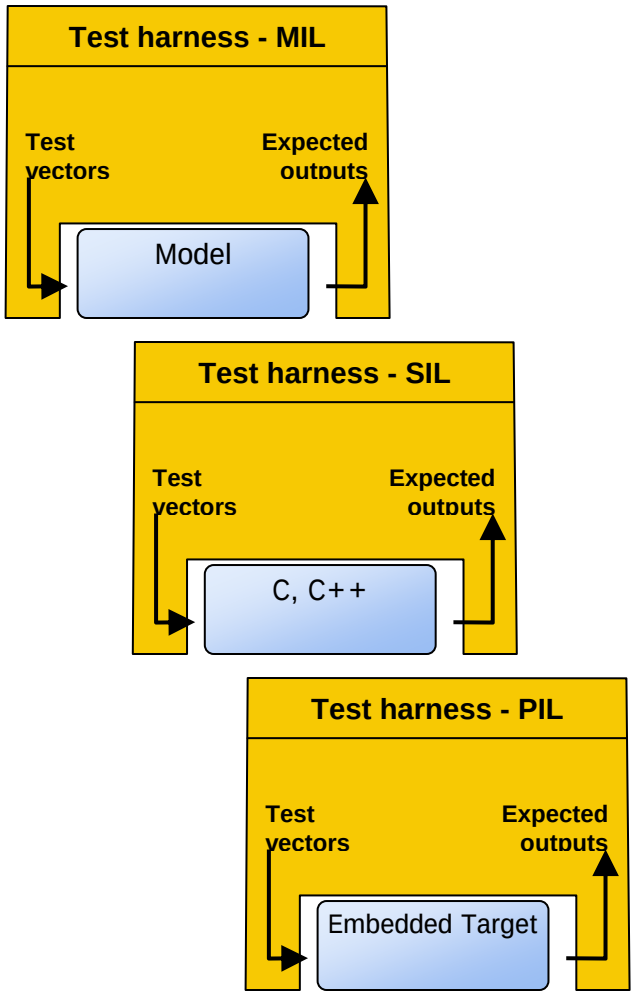

Figure 1. Develop tests to run against the model (model-in-the-loop); reuse those same tests with the code (software-in-the-loop) and embedded code (processor-in-the-loop).

TEST EXHAUSTIVELY IN SIMULATION

There is an old saying, "the only person who believes the simulation results is the engineer who developed the model." The corollary to that statement is "the only person who does not believe the test results is the test engineer." While there is no one solution that meets all testing needs, most engineers are more confident with the results from hardware testing. Fully exploring the design space variability in simulation, however, will save time later in the development cycle. In simulation, engineers develop requirements validation and initial verification-based test scenarios.

Almost every test scenario involves varying something: inputs, plant parameters, environmental factors. Time and expense often limit how much variability can be tested. By testing in a model environment, however, different test cases can be run much faster and, if the processing power is available, in parallel. Exploring the entire parameter space in simulation can narrow down the set of critical tests that must be run in real time or in the real world later. Simulation models also enable

engineers to test conditions that would be destructive or cost-prohibitive to run in the lab or on the road. For example, if a brake system controller does not work correctly the first time it is tested on a vehicle then costly prototypes could be damaged or destroyed.

RUN THE SAME TESTS IN SIMULATION AND IN THE LAB

The best results are achieved when there is a continuum of tests that run throughout – and in parallel with – the design process. When models are developed, tests that focus on the aspects of the design captured in the model should be run on the model. As the design evolves and is implemented, the corresponding tests should also evolve and be applied in parallel. Thus, at the end of the design, engineers are not merely relying on what they believe in, but also have a well established set of continuous verification results that demonstrate the efficacy of the design.

To run the same tests in simulation and in the lab, engineers need tools that facilitate hardware connectivity, measure physical quantities in a laboratory environment, and link to the modeling environment. These tools enable the reuse of test vectors run under simulation in hardware-based testing. Using identical test cases makes comparing hardware and modeling results much easier.

HARDWARE-IN-THE-LOOP TESTING

It is common for powertrain engineers to develop control and on-board diagnostics algorithms in parallel with the physical powertrain components. The software realization of those algorithms takes a long time and must be started very early in the design cycle so both the hardware and software arrive at the end of the assembly line functioning properly. To perform validation of the algorithms before production hardware is available, engineers use hardware-in-the-loop (HIL) testing. In this context, HIL is simulation used to test production ECUs and to test prototype control algorithms.

This kind of testing requires models of the physical components that have sufficient fidelity to test the various execution paths within the software. An environment for modeling and simulating physical systems is needed. With Simulink® and Simscape™ from The MathWorks, engineers can simulate the plant model and then use Real-Time Workshop® to generate code that can be run on a HIL computer.

INTEGRATION TESTING

One cornerstone of Model-Based Design is to make continuous testing and validation a part of the design process. In this process, components and subsystems are built, tested, integrated with larger systems, and tested again. This approach exposes defects in the interface between components or subsystems as early as possible.

When integrating several subsystems, it is important to have a thorough understanding of the requirements of each system and component. A best practice is to construct test scenarios for each of these requirements up front in the design and validate the subsystem models against these test scenarios. This process validates the models, the requirements and the test scenarios and reduces the number of errors introduced at this early development stage. When engineers reuse the same model tests in production testing, they reduce development effort significantly.

USE ALL AVAILABLE TECHNIQUES

Often engineering teams look to just one set of verification tests or verification criteria to determine whether a design meets the specifications. For example, some engineering organizations use a set of functional tests to verify the design. Others seek to “cover” the code with tests and use coverage metrics such as Modified Condition/Decision Coverage (MC/DC), a required metric in some safety-critical system development standards [5]. In most cases, however, there is no single test that can fully verify that the design has met the requirements and will not fail in operation. Instead, verification can be viewed as a series of filters that are applied to a design to ensure that the final production implementation meets all of the design specifications – explicit, implicit, and derived.

With the new generation of tools, a new verification technology is now available that goes beyond testing. Formal methods can prove that a design meets certain specified properties and generate examples of violations if it does not. These tools can also generate tests automatically to meet specific test objectives inserted by the developer or turned on model-wide. For example, a test engineer can instruct the tool to generate a set of test vectors that will provide complete MC/DC structural model coverage. For code verification, formal methods can be used to detect runtime errors or prove source code reliability without compiling. While currently limited to the component, sub-system, or module level due to scalability of the underlying formal analysis technology, these tools provide powerful new methods for VV&T in Model-Based Design.

DESIGN VERIFICATION USING TEST GENERATION FROM MODELS

Determining when a test suite is sufficient has often been considered more art than science because it is typically based on the judgment of a design or test engineer. MC/DC is an emerging metric that provides a more objective measure of test completeness. Coverage is a measure of how much of the logic in a model – or in source code – has been exercised during testing. MC/DC is the most stringent measure of coverage. The U.S. Federal Aviation Administration’s (FAA) DO-178B

safety-critical standard requires complete MC/DC of any software being deployed in a safety-critical system [3]. Writing a set of tests that achieve 100% MC/DC is challenging. The controller shown in Figure 2 is not very complicated. We found, however, that writing the tests that exercise every logical path in the model would take a design engineer at least as long as the original design effort. With a test generation tool this can take as little as three seconds, resulting in the tests shown in Figure 3.

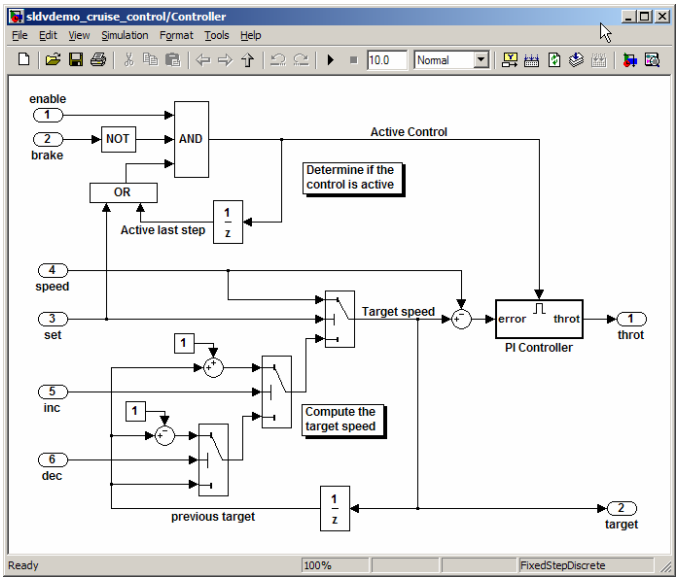

Figure 2. A cruise control model requiring a test suite that achieves 100% MC/DC coverage.

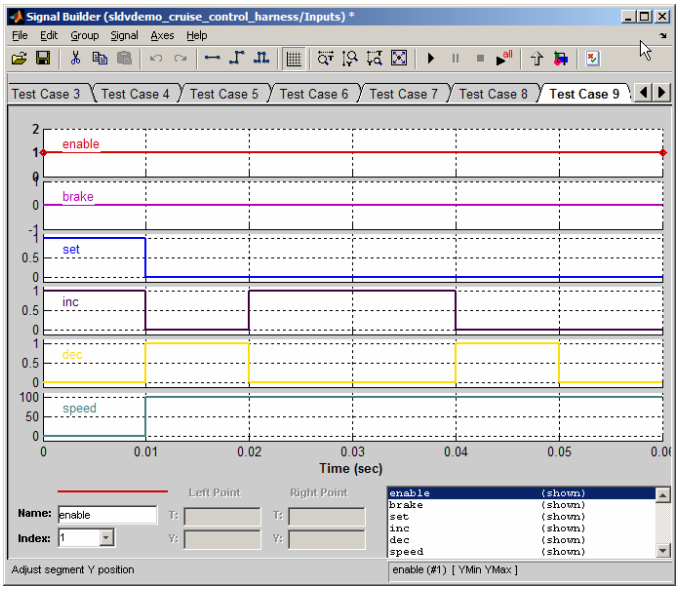

Figure 3. Automatically generated tests for 100% MC/DC structural coverage of a cruise control model.

Automatic test generation can be useful to an OEM handing off a subsystem or component model to a supplier as a software specification. The OEM can use test generation to develop tests that they can use as acceptance tests via SIL. The supplier can generate tests from a more detailed version of the specification model to ensure the code they develop matches the behavior of the model via PIL. Both sides are better able to ensure that the code is correct before it is integrated into a controller. As a result, problems are identified before integration or hardware-in-the-loop testing.

CODE VERIFICATION WITH FORMAL METHODS

Until recently, there were three options for detecting run-time errors in embedded software: code reviews, static analyzers, and trial-and-error dynamic testing. Code reviews are labor-intensive and often impractical for large, complex applications. Static analyzers identify relatively few problems and, most importantly, leave most of the source code undiagnosed. Dynamic or white-box testing requires engineers to write and execute numerous test cases. When tests fail, additional time is required to find the cause of the problem through an uncertain debugging process. Even with tests developed from a model, the tests can miss design errors that manifest themselves in the code as run-time errors. These can include:

- Overflows and underflows
- Division by zero and other arithmetic errors
- Out-of-bound array access
- Dead (or unreachable) code

Other code errors that can show up, especially in a mix with hand-generated code, include:

- Illegally de-referenced pointers
- Read-only access to non-initialized data
- Dangerous type conversions

Code verification tools based on formal methods, such as MathWorks™ PolySpace® tools, analyze code and detect these types of run-time errors. More importantly, use of these tools can prove the absence of such errors and provide strong assurance of the code's reliability. For the code shown in Figure 4, proof of code correctness or the absence of errors would be indicated by all lines colored as green. In addition, formal methods allow a tool to automatically verify important dynamic properties of programs. By verifying the dynamic properties of embedded applications, formal methods can encompass all software behaviors and all variations of input data, including how software can fail.

Used in addition to testing, as part of a development process, code verification tools provide another technique for identifying design and implementation errors that would be difficult to find and costly to correct later in the development process.

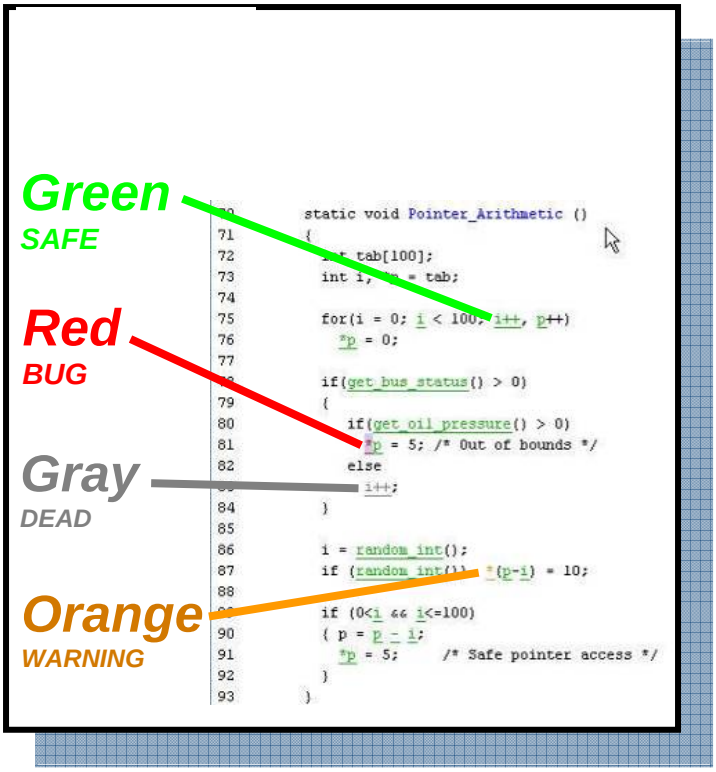

Figure 4. Code verification tools using formal methods can detect run-time errors, identify dead code, and prove code correctness.

CONCLUSION

Verification, validation, and test activities are critical to the success of any development process. With Model-Based Design, models are used to verify, validate, and test a design early and continuously through out the design process. This improves a team's ability to deploy a high-quality embedded system on time compared to traditional methods, which rely on verification, validation and testing at the end of the process. There are many ways a development organization can apply verification and validation techniques when using Model-Based Design, but our experience with several process adoptions reveals a clear set of best practices: Develop model tests with the design, Test exhaustively in simulation, Run the same tests in simulation and in the lab, and Use all available techniques. Applying these best practices helps ensure a development process that takes full advantage of Model-Based Design to provide more rigorous verification, validation, and test.

REFERENCES

1. User presentation from The MathWorks Automotive Conference, available here: www.mathworks.com/industries/auto/iac/ and user presentations from Model-Based Design conferences located: www.mathworks.de/company/events/mbd/ www.mathworks.co.uk/company/events/mbd/
2. G. Hodge, J. Ye, W. Stuart, "Multi-Target Modeling for Embedded Software Development for Automotive Applications," Paper 2004-01-0269, 2004 SAE World Congress.
3. Return on Investment for Independent Verification & Validation, NASA, 2004.
4. T. Erkkinen, "Production Code Generation for Safety-Critical Systems," Technical Paper, 2004 SAE World Congress.
5. B. Aldrich, "Using Model Coverage Analysis to Improve the Controls Development Process," AIAA 2002.

CONTACT

Brett Murphy is responsible for the technical marketing of verification, validation, and test products at The MathWorks.

Brett has extensive experience in controls analysis, real-time software development, systems engineering and product marketing in the aerospace and embedded systems industries built while working at Fujitsu in Tokyo, Japan; Space Systems/Loral in Palo Alto, CA; and Real-Time Innovations, Inc. in Santa Clara, CA. Brett holds a BS and MS in Aerospace Engineering from Stanford University.

E-mail: Brett.Murphy@mathworks.com