2008-01-2709

Verification, Validation, and Test with Model-Based Design

Tom Erkkinen
The MathWorks, Inc.

Mirko Conrad
The MathWorks, Inc.

ABSTRACT

Model-Based Design with automatic code generation has long been employed for rapid prototyping and is increasing being used for mass production deployment. With the focus on production usage, comes the need to implement a comprehensive V&V strategy involving models and resulting code.

A main principal of Model-Based Design is that generated code should behave like the simulation model. It should also be possible to verify that the model or design was fully implemented in the code. As a result, the transformation of models into generated code must be done in a way that facilitates traceability between the model and code. Also automated tests should be performed to determine that the code executes properly in its final software and hardware environments.

For example in a typical commercial vehicle application, the control algorithm and plant model are simulated together in a system simulation environment. Once the system model satisfies the requirements, the control model is checked to ensure that it has been fully exercised or covered. Once checked, code is then generated for production applications. The code is analyzed, tested, and compared to the original model results. Common model and code verification activities include software-in-the-loop (SIL), processor-in-the-loop (PIL), and hardware-in-the-loop (HIL) testing. In addition to functional results, it is important especially for high-integrity systems, that the model and code have been checked and assessed to known standards.

This paper describes recent advances in verification, validation, and test technologies involving Model-Based Design with production code generation.

INTRODUCTION TO MODEL-BASED DESIGN

A model represents a dynamic system whose response at any time is a mathematical function based on its inputs, current state, and current time. Historically, system engineers have used block diagrams as shown in Figure 1 to create models and design algorithms within numerous engineering areas such as Feedback Control and Signal Processing. In recent years, graphical modeling environments consisting of block diagrams and state machines have been used to analyze, simulate, prototype, specify, and deploy software algorithms within a variety of embedded systems and applications. Model-Based Design refers to the use of models and modeling environments as the basis for embedded system development.

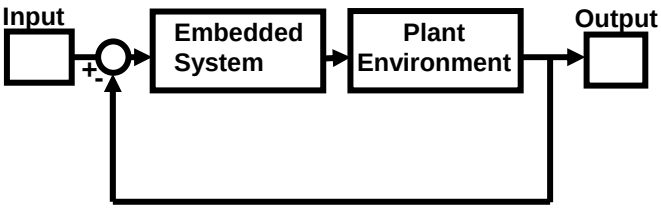

Figure 1: Feedback controller model.

Systems developed using Model-Based Design include:
- Commercial vehicle electronics
- Power plant regulators
- Digital motor controllers

Model-Based Design is used throughout the system development life cycle and provides design flows that include continuous verification and validation of requirements, designs, and implementations. This approach is important for formal software processes such as IEC 61508 [1] and for other projects seeking error prevention and early error detection.

The main development activities that occur during Model-Based Design include:
- Modeling and simulation
- Rapid prototyping
- Production code generation and integration

Verification and validation (V&V) occurs continuously during Model-Based Design and includes many

activities. This article focus on two key activities: model checking and processor-in-the-loop testing.

MODEL CHECKING

An important group of model V&V activities comprise different static analyses and checking tools. Model advisors check a model for conditions and configuration settings that can result in inaccurate or slow simulation, problematic maintenance, or generation of inefficient production code.

Reports are generated that list identified suboptimal conditions and settings. Advice is provided suggesting better modeling approaches and settings. There are several types of checks.

Basic Model Checks: Automated model advice is provided using basic checks, requirements consistency checks, and industry model standard checks. Basic checks range from support for updating the model to be compatible with the current product release version, to identifying unconnected lines and ports, to checking the root model interfaces.

To invoke the advisor, select the checks and then run them as shown in the left- and right-hand sides of Figure 2, respectively.

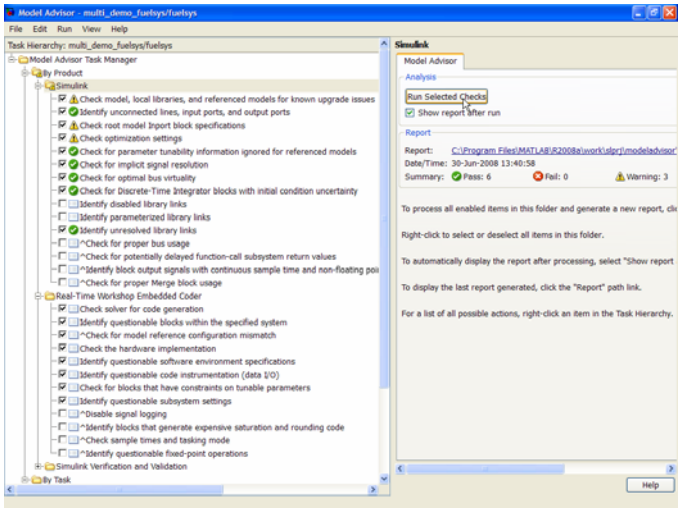

Figure 2: Model advisor basic checks - invocation

After performing the checks, results are displayed as shown in Figure 3. Hyperlinks are provided in the report, automating navigating to the dialog or menu where the setting can be adjusted based on the reported advice.

The basic model checks should be performed and reported deviations considered before other quality assurance measures such as peer reviews or industry model standard checks are done.

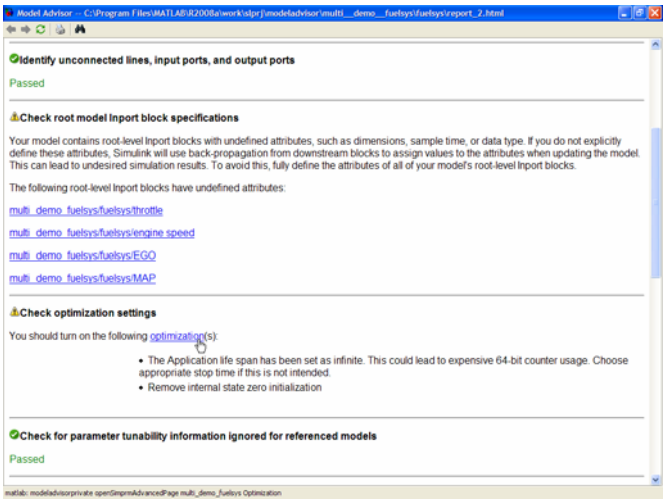

Figure 3: Model advisor basic checks - results

Requirements Consistency Checks: If the model is linked with requirements in third-party requirement management tools or databases, these checks identify inconsistent, missing, or changed requirements. Requirements consistency checks can also identify and repair requirements with missing documents and inconsistent requirements descriptions. See Figure 4.

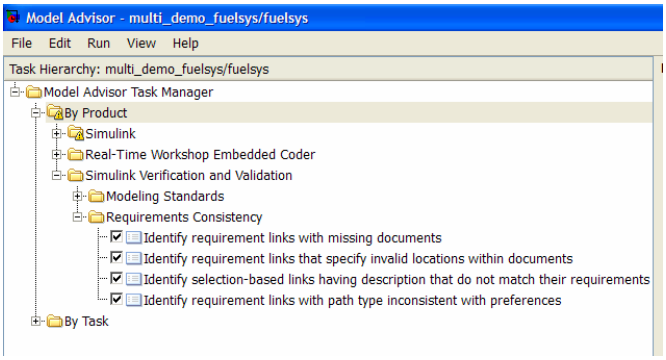

Figure 4: Requirements consistency checks

Modeling Standards Checks: Many projects use in-house or industry specific software and modeling development standards.

Model checks have already been developed for some industry standards including:

- DO-178B
- MAAB
- IEC 61508

DO-178B is an aerospace standard that will not be discussed here.

MathWorks Automotive Advisory Board (MAAB) checks facilitate designing and troubleshooting models for automotive applications. A new version of MAAB was made available in 2007, MAAB v2.0.

MAAB checks include:

- Prohibited blocks inside controllers
- Port and signal name mismatches
- Unconnected signals

IEC 61508 is a generic, application-independent standard for electrical / electronic / programmable electronic safety-related systems (E/E/PES) that is supposed to ease the development of sector-specific norms for E/E/PES. It is applied transitionally in the development of E/E/PES in those areas for which a domain-specific norm does not yet exist. IEC 61508-3 is concerned with the requirements for software development.

IEC 61508 can be considered as a prescriptive standard, which provides detailed lists of techniques and measures with recommendations.

IEC 61508 model checks analyze the model and report on items such as model usage, model metrics, and configuration management information as shown in Figure 5.

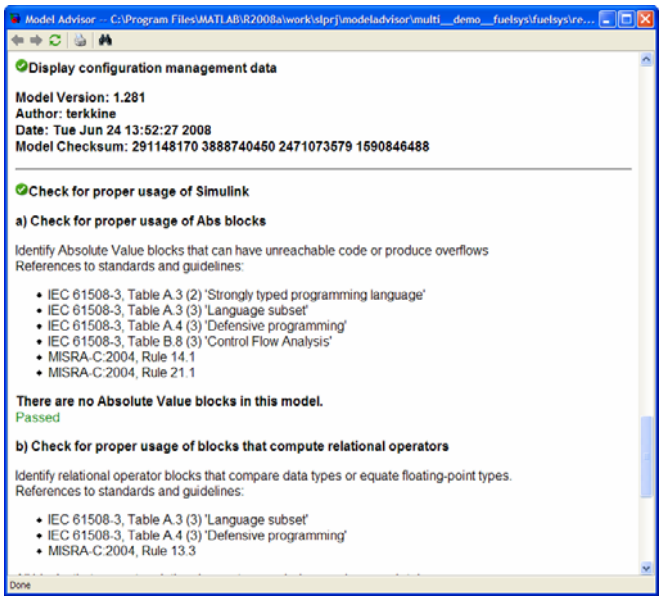

Figure 5: IEC 61508 checks - report

Model Complexity Measurement allows one to measure the complexity of the entire model as well as the individual subsystems. Cyclomatic model complexity is a measure of the structural complexity of a model. It is calculated with the IEC 61508 checks and approximates the McCabe complexity measure for code generated from the model. Model complexity measurement helps to achieve a modular approach on the model level and especially to maintain an appropriate module size limit.

Finally, if the built-in checks are not sufficient, a model advisor API is available that facilitates the development of custom rule checks by engineers using Model-Based Design.

PROCESSOR-IN-THE-LOOP TESTING

Simulation of models is an early verification and validation (V&V) technique. Testing models via simulation is a more rigorous approach than the ad-hoc simulation runs used in early algorithm development. Model testing requires a systematic approach to test case creation and execution. Special blocks, such as signal builders and assertions, facilitate this type of rigorous test procedure. New capabilities for V&V on models now exist such as structural coverage analysis and test case generation. In-the-loop testing techniques allow one to reuse the model test cases and test environment for execution with the production application during various stages of integration.

Software-in-the-loop (SIL) testing involves executing the production code for the controller within the modeling environment for non-real-time execution with the plant model and interaction with the user. The code executes on the same host platform that is used by the modeling environment. A code wrapper of the generated code provides the interface between the simulation and the generated code. See Figure 6.

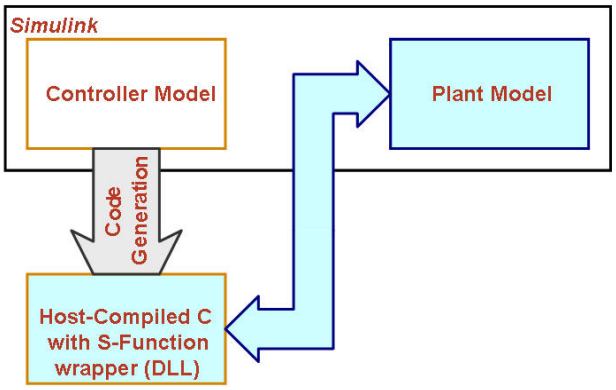

Figure 6: Software-in-the-loop testing.

For hardware-in-the-loop (HIL) testing, the code is generated just for the plant model. It runs on a highly deterministic, real-time computer. Sophisticated signal conditioning and power electronics are needed to properly stimulate the ECU inputs (sensors) and receive the ECU outputs (actuator commands). Whereas rapid prototyping is often a development or design activity, HIL serves as a final lab test phase before final system integration and field tests commence. See Figure 7.

Note that an on-target rapid prototyping and production code example using Model-Based Design based on case study by John Deere was presented at the SAE Commercial Vehicle conference in 2007 [2].

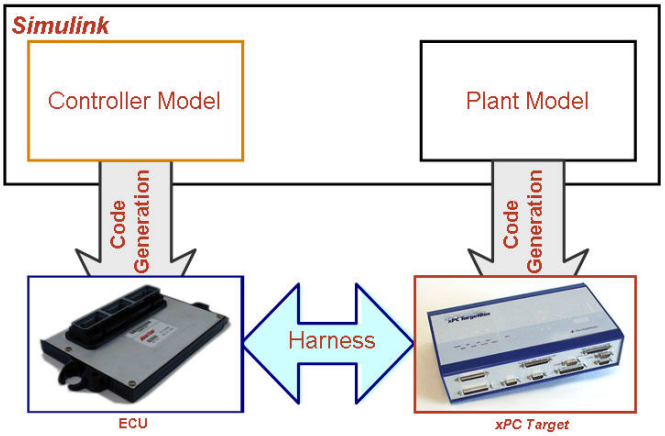

Figure 7: Hardware-in-the-loop testing.

Processor-in-the-loop (PIL) testing occurs after SIL but before HIL testing. As with SIL, PIL exercises the production code for the controller in non-real-time. However the code executes on the actual embedded processor or an instruction set simulator, using the embedded cross-compiler. Thus, it verifies the embedded object code functional behavior.

With PIL, the model does a single calculation iteration. The inputs are calculated and passed to a PIL block. The PIL block serves as a conduit and passes the model inputs to the code running on the embedded microprocessor, or emulated processor if an instruction set simulator is used. Once the target processor receives the model inputs, it executes a single time step and computes the output data. The outputs are then passed back to model using the PIL block. The model then continues to simulate while the target processor waits for new inputs.

See Figure 8 for a PIL example using an Instruction Set Simulator (ISS).

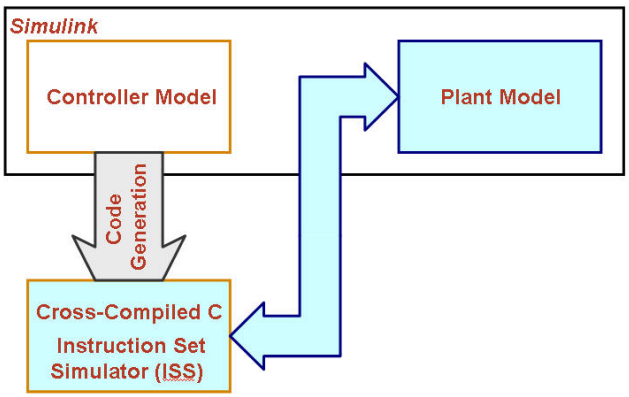

Figure 8: Processor-in-the-loop testing on the host

PIL testing can occur simultaneously with model testing or done separately. The tests can be interactive or done in batch mode via scripts. Batch processing is most convenient for repetitive, production tasks such as regressing testing.

Plots comparing the model's functional (expected) results to the PIL (actual) results can be developed and analyzed. Integer or fixed-point results should be bit-wise accurate. Floating point results will need to be assessed based on an acceptable margin of error, or epsilon. Differences in floating point results often occur between host and target platform due to factors such as variations in floating point math library implementation.

One may execute PIL tests on multiple target platforms in order to assess an algorithm's robustness to variations in floating point implementations using various hardware, compiler, and even compiler version combinations.

PIL is also useful for testing behavior that cannot be tested in a modeling environment. One example is the use of target optimized code. Some processors have special built-in instructions that are not ANSI or ISO-C compliant. These instructions may use special hardware on the processor that processes certain routines extremely fast, such as FFTs or IIRs used in signal processing applications. Some processors have built-in overflow protection for fixed-point calculations.

When target optimized code is used within Model-Based Design, one cannot execute or test the final code within the host-based model simulation environment. With PIL, however, the optimized code can easily be tested using the modeling environment as the test harness.

Figure 9 shows a PIL test occurring simultaneous with a model test and an output plot comparing the two results.

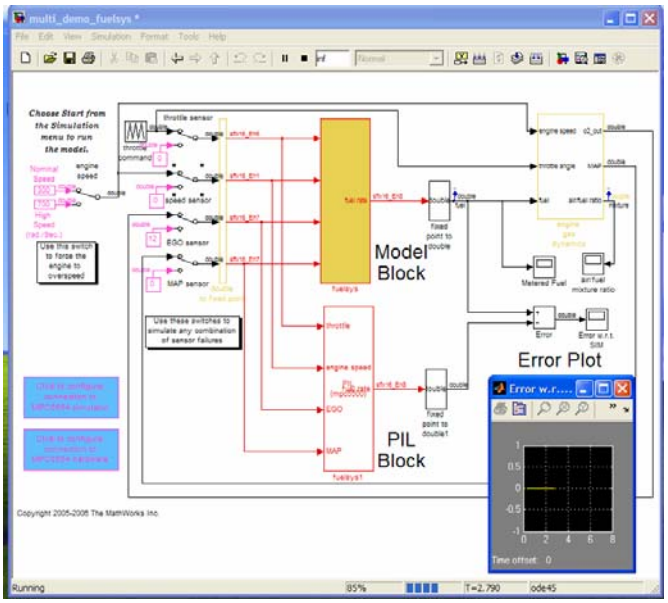

Figure 9: Processor-in-the-loop testing example

CONCLUSION

Automatic code generation with Model-Based Design is an important technology that offers embedded system developers a number of advanced options for designing and deploying production software. Model-Based Design also provides a rich verification and validation environment for embedded systems. Recent techniques were described herein that support model checking and processor-in-the-loop tests.

REFERENCES

1. IEC 61508-3:1998. International Standard IEC 61508 Functional safety of electrical/electronic/ programmable electronic safety-related systems – Part 3: Software requirements. 1st edition, 1998

2. Tom Erkkinen, The MathWorks, Scott Breiner, John Deere, *Automatic Code Generation – Technology Adoption Lessons Learned from Commercial Vehicle Case Studies,* SAE CV 2007, 08AE-22, www.mathworks.com/mason/tag/proxy.html?dataid=9939&fileid=44540