Simulator–in–the-Loop Environment for Autocode Verification

Girish Palan, Pavan K S, Rajani S R
Electronics and Safety, Delphi Technical Centre, India

ABSTRACT

Model-based development processes have increasingly been adopted for the development of automotive embedded control software to help implement the complex systems and reduce development time. Control algorithms are designed with Matlab®Simulink® /Stateflow®; IBM® Rational® Statemate or similar modeling tools and converted to code using autocoding tools. Though code is auto-generated, software verification is still required since auto-generation involves human intervention, and it is especially critical for fixed point code. Various levels of testing are adopted to verify the model and auto generated production code.

Model-in-the-Loop (MiL) testing allows testing of algorithms in the simulation environment at early stages of the development cycle. Software-in-the-Loop (SiL) testing involves execution of the production code on the same host platform within the simulation environment that is used by the modeling environment. Processor-in-the-Loop (PiL) testing provides a setup to test the production code on a target board with the target processor. In PiL, code is tested on the final target processor which helps uncover any processor specific issues at an early stage.

This paper talks about a new, alternate and hardware independent approach to PiL called “Simulator-in–the-Loop” testing in which code is tested on a simulator of the target microprocessor. In addition to the advantage of eliminating the target board, this approach also reaps all the benefits of PiL. This paper suggests the workflow starting with the model development in Simulink and autocode verification using target simulator in Target Link environment.

INTRODUCTION

Embedded Software is indisputably the core of the current electronic era. From simple handheld calculators to high-end mobile phones; from remote controlled toy cars to complex space crafts, embedded software plays the key role.

The advancement in technology has turned the concept of a global village into reality. The automotive domain, in particular, is a conglomeration of the best innovations in the field of engineering. In the present era of smart vehicles, the demand for sophisticated automotive technology is increasing exponentially. The pressure on suppliers to provide new technology on time and at an affordable price is growing, and providing the technology is necessary to survive in the market. The constraint on the development period is a concern. The tremendous increase in the complexity of embedded software to incorporate all the high-end features makes a rigorous testing process inevitable in the automotive field. With the model-based approach, development time could be reduced considerably while maintaining the standard and the quality of the product. [1] This is the driving force for the automotive industry to adopt the model-based software development.

MODEL-BASED SOFTWARE DEVELOPMENT (MBSD)

Model-Based Design approach provides an efficient approach of rapid prototyping, automatic code generation, as well as continuous testing and verification. It helps us save development time and provides ample time to detailed testing of the end product.
Figure 1: Model-Based Design Workflow [3]

The design phase includes modeling of all components that affect system behavior, algorithms and the control logic that represent the plant. The model simulation analyzes system performance and allows design modifications to the model for optimal behavior. During the implementation phase, automatic code generation tools help transform the model into hardware specific production code in a short period of time. The testing process is conducted at all levels, i.e. design phase, implementation phase and integration phase, to find any flaws during the process of development. All faults identified are fixed, and the process is repeated until the integrated software is virtually error free.

Model-based development has resulted in:

- Greater synergy between various steps of software development
- Better quality of software
- Shorter development time

The benefits of the model-based approach include inherent simulation ability and possibility to verify the design early have been discussed in several papers at various conferences [1][2].

**AUTOMATIC CODE GENERATION**

Automatic code generation has changed the way that embedded software engineer’s work today. The former method of writing thousands of lines of code manually has been eclipsed by code generation tools that allow engineers to generate code automatically from the model. The automatic code generation tool provides flexibility to generate either fixed point or floating point code from the model. This results in increased productivity, improved quality, and enables engineers to focus on systematic development of the model, thereby fostering innovation [2].
Automatic code generation tools offer advantages over handwritten code by:

- Providing flexibility to generate code in different programming languages (like C, C++ etc.) required by the project.
- Optimizing code for specific processor architectures.
- Providing the developer an option to choose between fixed point and floating point code generation accordingly to the target processor requirements.
- Generating code that is compliant with industry certified standards.
- Eliminating manual coding errors.
- Reducing software development time.

NEED FOR TESTING AUTO GENERATED CODE

Testing is the integral part of any software development life cycle (SDLC). It helps to identify and fix any issues or design flaws which might have been overlooked at the design or implementation phase. The auto-generated code, especially fixed-point code, is prone to two types of representation errors namely, overflow/underflow errors and resolution loss errors. Also, the code behavior might change from host environment to processor environment depending upon compiler or the hardware configuration of the processor. It is extremely important to make sure that the code produces identical results in both host and processor environment in Model-Based Design [6].

The autocode generated from TargetLink® Automatic production code generator tool has to be verified for adherence to its functionality. This is generally done by running the functional test vectors on the model and code and comparing the outputs. The functional test vectors can be either simulated data from Simulink or real-time logged data from vehicle. Additional test vectors can be created manually for a specific functional scenario. Overview of the model-based testing process is illustrated in figure 2.
Different testing approaches discussed in this paper help reveal many unpredictable issues and find the root cause of such issues. These can be used to test autocode in model-based design workflow. Three established methods are given below:

- Model-in-the-loop (MIL)
- Software-in-the-loop (SIL)
- Processor-in-the-loop (PIL)

**MODEL-IN-THE-LOOP (MIL)**

The model-in-the-loop (MIL) simulation mode is used for controller design and parameterizing, for behavior validity checks and test purposes. When simulating in MIL simulation mode, the particular subsystem is simulated on a block basis and therefore with the greatest possible precision. The simulation results obtained serve as reference for the production code test – SIL and PIL.

**SOFTWARE-IN-THE-LOOP (SIL)**

Software-in-the-loop (SIL) testing involves execution of the production code for the target 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 was used for MIL. A code wrapper of the generated code provides the interface between the simulation and the generated code.

**PROCESSOR-IN-THE-LOOP (PIL)**

Being one of the non-real time simulation modes, PIL is that phase of testing where the generated production code executes on the actual embedded processor connected to the host PC. PIL ensures functional equivalence of code running on the target processor relative to the model behavior captured in simulation.

![Figure 3: PIL Test Approach](image-url)
For PIL testing, the production code is cross compiled on the host PC. The resulting executable is downloaded onto the Evaluation Board (EVB) that holds the microcontroller same as that of the ECU (Electronic Control Unit). This equips the developer with the processor at a stage as early as software development. With PIL in place, the developer gets an opportunity to test the developed software in an environment similar to the target platform.

For every timestamp, the inputs are calculated and passed on to a PIL block. The PIL block serves as a conduit and passes the model inputs to the code running on the embedded microprocessor. Once the target processor receives the model inputs, it executes single timestamp and computes the output data. The outputs are then passed back to model using the PIL block. The model continues to simulate while the target processor waits for new inputs.

PIL tests for functional equivalence. PIL also lets the developer address compiler related issues with the generated production code. PIL testing identifies unpredictable flaws due to differences in results between the host pc and target. In some projects, PIL is an opportunity to compare algorithm behavior on processor boards that meet the same specifications, but come from different vendors.

A few limitations have been experienced with the PIL approach. The availability of target and cost involved in procuring targets for different platforms during the developmental stage is one of the concerns with PIL testing. To develop a test environment in PIL for specific Evaluation Board (EVB), a thorough understanding of communication protocol employed between the host PC and target along with good knowledge of the target architecture is a must. Various third party tools used for auto generation of production code often support only a limited set of target platforms. This handicaps the developer and requires rigorous PIL testing across projects to be performed.

The above mentioned limitations with PIL testing have led to the development of a new approach called “Simulator-in-the-loop” testing where the developer is no longer handicapped due to the observed constraints.

SIMULATOR-IN-THE-LOOP (SIMIL)

Although Simulator-in-the-loop (SimIL) testing is similar to PIL, the vital difference is that during SimIL, code executes on an instruction set simulator instead of actual embedded target. With SimIL, tests are executed with the embedded algorithm compiled and deployed on the target simulator. This approach requires a simulator which provides instruction set to command simulator from outside world. Trace32® from Lauterbach GmbH is one of the finest simulators available today and helps to set up the SimIL test framework. The Trace32® simulator supports large numbers of target processors. In SimIL, the in-house tool developed in the Simulink environment controls the execution of code on the target simulator. Data flow between the simulator and the test framework is illustrated in Figure 4.

![Simulink Environment (S-function to interact with Target Simulator)](Simulink Environment (S-function to interact with Target Simulator))

![Target Simulator (Embedded Object Code)](Target Simulator (Embedded Object Code))

Write Input and Calibration value

Read Code response

Figure 4: Data Flow between Simulink and Simulator

SimIL exercises the production code for the controller in non-real-time. For this approach of model-based software testing, the Simulink/Stateflow model serves as a requirement for testing purposes. The outputs from the model can be used as a basis for assessing outputs generated by the production code. The Simulink model is provided as the input to the Reactis tester (From Reactive systems Inc.) along with test vector. The Reactis Tester also offers automatic test generation from Simulink models. Each test case in a test suite consists of a sequence of inputs fed into the model as well as the responses to those inputs generated by the model. The test suite generated by the Reactis tester, which includes model response, is later used as the input to test the auto-generated production code using TargetLink® automatic production code generator.
The embedded cross compiler compiles the auto generated code and provides the embedded object code. The object code, test suite and the calibration set are fed as input to an in-house test tool which automates the execution of test data on target simulator and logs the actual output from embedded code. The tool is flexible enough to work with floating point and fixed point code. It parses through the Target Link model and calibration data dictionary to populate the software data type. This information is required to convert the real-time value to an engineering value. The tool generates the final difference report by populating model response, code response and the difference between code and model value. This information is very useful in analyzing the reason for mismatch, if any. The tool is developed and executed in Simulink environment. Input data is fed from Simulink to Simulator and output response from code is fed back to Simulink.

![Figure 5: Simulator-in-the-Loop framework](image)

The Automated Test Tool employs this workflow for every simulation step until all input data is read and output responses are logged. The advantage of this tool is that it logs the output values of both the specification model (expected output) as well as the code (actual output) for each time step as shown in the snapshot below. The tool allows automated running of many vectors in batches, and it also displays a summary report for all of the different scenarios. Subsequently, only the failed test vector which will be highlighted by the tool can be analyzed.

![Figure 6: Test Report](image)
CONCLUSION

The SimIL approach is one of the best alternatives for PIL to overcome the non-availability of the target board and test environment. It provides the same level of confidence in terms of code behavior on an actual target. It provides the developer an opportunity to test the production code on a virtual target at an early stage of development. The Trace32® simulator provides flexibility to test the production code on various target platforms used across projects; however, the developer cannot adopt this approach unless the simulator used supports an external command set to control the interaction between the simulator and the external environment.

ACKNOWLEDGMENTS

The authors would like to acknowledge and thank the modeling and autocode teams at Delphi Technical Centers in Bangalore, India, Kokomo, IN, USA and Shanghai, China for all of the contributions and guidance provided during the development of the tools and process described in the paper.

REFERENCES

5. The dSpace, www.dspace.de

CONTACT

Girish Palan
Email: Girish.M.Palan@delphi.com

Pavan Kumar S
Email: Pavan.K.S@delphi.com

Rajani S R
Email: Rajani.Sr@delphi.com