Accelerating Sensor Development with Rapid Prototyping and Model-Based Design

By Martin Hein, Hella Fahrzeugkomponenten GmbH

Before automotive OEMs purchase a new sensor system, they want to see a real-time prototype of the sensor working in a vehicle environment so as to evaluate its performance and potentially modify its specifications. For example, we often need to demonstrate an early version of our sensors, which usually incorporate algorithms and logic implemented on an FPGA or microprocessor. To meet this requirement, we use a custom rapid control prototyping platform and Model-Based Design to build real-time prototypes of new designs early in the development process. The real-time prototype (known at Hella as an A-sample) serves both as a proof of concept and as a living specification throughout development.

Instead of waiting up to two years for an ASIC implementation, we can produce an A-sample that incorporates about 80% of the final product's functionality in a few months. The A-sample enables us to work with the customer early in development to refine the sensor's functionality and evaluate the code size, partitioning of modules, and hardware requirements. Testing groups use the A-sample to set up the test environment and test suites so that testing can start as soon as a production sample, implemented as an ASIC or on a microprocessor, is ready.

Creating a Flexible Prototyping Environment

We built the Hella Fahrzeugkomponenten GmbH Rapid Control Prototyping (HFK RCP) unit because commercially available alternatives lack the flexibility we need. Most off-the-shelf prototyping systems only support ECU software development, but a sensor design may also include VHDL® code and discrete electronic components. A second limitation of a commercial system is the fixed set of interfaces it provides. At Hella we must support a wide array of communications protocols and interface hardware, including SPI, I^{2}C, LIN, XCP, CAN, and SENT.

With Model-Based Design and our custom prototyping environment we can add new interfaces, protocols, and capabilities as needed. We can target microprocessors and FPGAs as we develop specifications and use the prototyping environment to extend or enhance algorithms we have already implemented on a production processor.

From Requirements to Design

Our development process, which follows the V-model, consists of five main steps: requirements analysis, algorithm design, production code generation, code verification, and testing. In the requirements analysis phase, we work with our customer to define system requirements in IBM® Rational® DOORS®. We then create an initial model of the design in Simulink® (Figure 1).

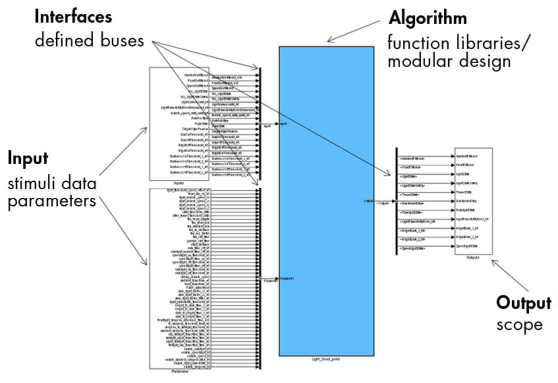

Figure 1. Simulink system model.

We use Simulink Verification and Validation™ to map requirements in DOORS to elements of the model, enabling bidirectional traceability of requirements.

When creating our model, we use Model Advisor to ensure that we are adhering to MathWorks Automotive Advisory Board (MAAB) algorithm modeling guidelines. We also include Model Advisor checks based on the guidelines developed internally at Hella.

For early functional verification of the initial floating-point design, we run simulations in Simulink, stimulating the model with test data gathered from a similar sensor or generated with a Simulink block. After these model-in-the-loop tests, we examine the model coverage reports created with Simulink Verification and Validation to identify untested elements in our model, updating the tests as needed to increase coverage.

In preparation for testing on the rapid prototyping platform, we model the communication interface that will enable the sensor algorithms to run in a vehicle. Working with MathWorks consultants, we have developed a local interconnect networking (LIN) blockset for Simulink, which has enabled us to expand the capabilities of our prototyping system to support LIN.

From Model to Prototype

After an internal design review of the model, we move the design into the HFK RCP unit (Figure 2). HFK RCP supports a broad range of design configurations with a standardized set of components that include TI's C2000™ microprocessor, a Xilinx® FPGA, and connectors for automotive buses and sensors, as well as an area for discrete electronic components.

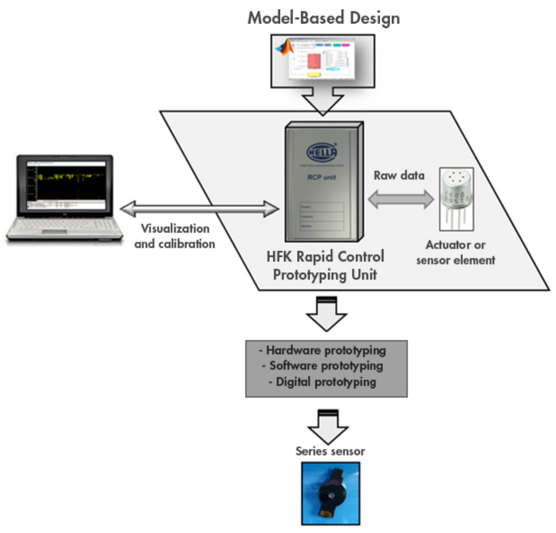

Figure 2. The Hella workflow using the HFK RCP unit and Model-Based Design.

For designs that target a microprocessor, we use Embedded Coder™ to generate code from our Simulink model and deploy it to the TI C2000 processor on the HFK RCP unit. If all or part of the design requires an FPGA, we use HDL Coder™ to generate VHDL code from the model for deployment on the Xilinx FPGA.

From Prototype to Production

Once we have verified the proof-of-concept A-sample using the HFK RCP unit, we prepare the design for implementation on the series processor. We use the Fixed-Point Advisor in Fixed-Point Designer™ to convert our floating-point model to an initial fixed-point design. Fixed-Point Advisor enables us to optimize for code size, memory footprint, and fixed-point scaling. We then recheck compliance with modeling guidelines and rerun our simulations, using Simulink Verification and Validation to generate model coverage and cyclomatic complexity metrics information. We compare the results from our floating-point and fixed-point models to ensure that no errors were introduced during the conversion.

We use software-in-the-loop (SIL) testing to verify the implementation of our algorithms in C and processor-in-the-loop (PIL) testing to verify the algorithm on real-time hardware. In this way we ensure that the model, which we have already verified, is implemented without introducing errors.

For SIL testing, we use Simulink Coder™ to generate C code from the sensor algorithm components in our Simulink model. We then replace these components in the model with an S-function containing the generated C code, and rerun the simulations. Once more we compare the results with earlier test results, this time to verify the software implementation.

We recently worked with MathWorks consultants who developed an Embedded Coder target for the 78K family of microcontrollers to enable PIL testing on our Renesas® 78K microcontroller. We can now use Embedded Coder to generate code, which we deploy to the device for PIL and in-vehicle testing.

The path from prototype to production varies depending on how the prototype was implemented. If we used HDL Coder to generate VHDL for an FPGA, we hand off our design and the generated VHDL to an external partner, who produces an ASIC based on the prototype. Because we have thoroughly verified the model and its HDL implementation, our ASICs require far fewer iterations, which in addition to reducing costs, virtually eliminates project delays.

If, however, we targeted a microcontroller for the prototype, we continue production within Hella. We use Embedded Coder to generate ANSI C code from the fixed-point Simulink model and target the production microcontroller. The generated code undergoes a rigorous test process within Hella that includes integration testing and static analysis with Polyspace Client™ for C/C++ and Polyspace Server™ for C/C++. The final integration of the ANSI C code into the target is then done using IBM Rational Rhapsody® (C/C++).

By tuning code generation settings and following established modeling guidelines, we can generate production code that is more compact than equivalent handwritten code. By reusing our prototype model for production code generation, we have reduced development time by about 60%. Further, with Model-Based Design and the HFK RCP unit we can run tests early in development, which enables us to validate requirements and verify design decisions months earlier than was previously possible.

Products Used

- MATLAB
- Simulink
- Embedded Coder
- Fixed-Point Designer
- HDL Coder
- Polyspace Client for C/C++
- Polyspace Server for C/C++
- Simulink Coder
- Simulink Verification and Validation

Learn More

- Video: MATLAB and Simulink for Simulation, Modeling, Control Design, Rapid Prototyping, and HIL (3:38)
- Improve Design Efficiency Using Modeling Standards Checking

See more articles and subscribe at mathworks.com/newsletters.

Published 2013
92092v00