Shift Gears Quickly

Processes and methods for using TargetLink in model-based development of transmission function software

The goal is to noticeably reduce the development time of prototypes and end products. To meet this, innovative methods for developing embedded software are necessary. That is why years ago, Continental AG decided to introduce model-based development and production code generation for developing transmission electronic control units (ECUs). This article shows the big picture, from working out the processes and methods, to the successful use of the code generator TargetLink in two production projects.

During the past decade, enormous demands were placed on the development of embedded software. On top of the traditional process of manual coding and software tests, more stringent requirements regarding safety standards and development methods have now entered the scene. The continually increasing complexity makes these require-ments even more demanding, especially with regard to the control of the new transmission generation with double clutch transmission (DCT) and continuously variable

The double-clutch transmission combines the energy efficiency of a manual transmission with the comfort of torque converter transmission.

transmission (CVT). In addition, automotive industry suppliers are constantly looking for new ways to shorten development times – as the market demands. As a result, the Business Unit Transmission at Continental AG performed a study a few years ago, which analyzed the application of model-based design and production code generation. The potential productivity increase resulting from this innovative, model-based method was initially assessed by using the method as an internal project. At the same time, the process, methods and tools for model-based design, including automatic code generation, were defined and adapted to the specific needs of the Business Unit Transmission.
After being used successfully in the first production project, the entire application software of a double clutch transimission was developed with this tool chain. This article describes the individual development steps and experience gained from the initial tests, the first production projects, and the double clutch transmission, which was autocoded entirely with TargetLink.

Designing the Processes and Methods

As part of an internal project, first the necessary process steps were drawn up, in order to start off the model-based design and automatic production code generation. The following requirements were placed on the model-based design to cover the complete V-cycle for the function and software development, with regard to the interaction between the OEMs, transmission and electronics suppliers (see fig. 1):

- Model the physical requirements of the ECU
- Simulate the actual ECU behavior (tasks, execution order, operating system)
- Support distributed development by setting up a multi-user environment
- Verify, validate and archive the models, code and scripts by adapting them to the traditional process

First, the model-based development method was applied to functionalities in demo vehicles. The preferred solution that arose was to use established tools based on the latest technology. Thus, MATLAB®/Simulink® were selected for the function design. The success of the project depended greatly on whether the demonstration of new functionalities in demo vehicles could be completed within just a short time frame. The simulation of concepts, and the subsequent test and verification were performed by model-in-the-loop (MIL) simulation on the PC. After successful verification, the function was ready for tests in the demo vehicle and on the test bench with the rapid control prototyping (RCP) environment. For a fast validation and fine-tuning of the functions, Real-Time Workshop® and dSPACE AutoBox/MicroAutoBox were used. Front-loaded tests made it possible to assess the customer's requirements quite early. After model-based development was used successfully in the rapid prototyping phase, the V-cycle phase was to switch from manual C coding to automated production code generation. dSPACE TargetLink was selected as the code generator for this step.

> "For the project performed with TargetLink, the time-to-market was even shortened, which our customer applauded."
>
> *Georg Grassl, Continental AG*

To prove the feasibility of automatic, optimized code generation for a 16-bit microcontroller, a position control algorithm for a transfer case transmission was used. First, the already existing function was transferred into a model-based design and then expanded with implementation information for production code generation. Then the automatically generated code was integrated and validated on the production ECU. To prove the efficiency of the generated code, its resource consumption was compared with that of manually coded software already used in series production.
Finally, on the basis of these preliminary tests, the important process elements for generating production-intent software with the new development method were identified. The following section illustrates these process elements in closer detail, including a few supportive methods:

Model-Based Function Design

By simulating Float models it is possible to quickly evaluate whether the functional requirements can be met. This step is then followed by rapid prototyping to validate the functional requirements in a demo vehicle. It is necessary to define the function architecture at an early state, together with the resulting generation of model libraries, so that partitioned development can be achieved and the configuration elements created.

In addition, the modeling guidelines for the function and software also contribute towards making the compatible, model-based design possible. Industrial standards regarding adherence to in-house guidelines are evaluated and, if necessary, project- and application-specific agreements are added.

Model-Based Software Design

Model-based design templates to generate optimized target code are introduced and must already be adhered to during the function design stage. This is in close connection to the standards established by MISRA (Motor Industry Software Reliability Association) and the tool manufacturers: for example, the MISRA AC TL guidelines. Designing software means scaling, defining block properties, adapting the configuration of the code generator and thereby supporting the automatic code generation. The so-called implementation model is made when the implementation information (memory assignment, etc.) is added.

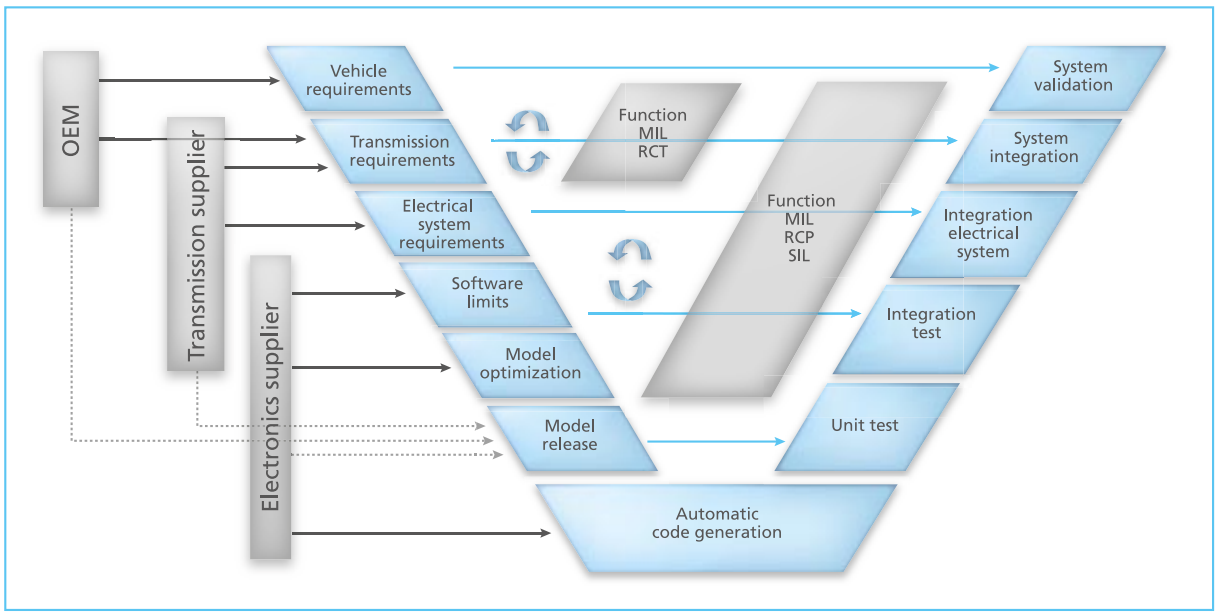


Fig. 1: The development cycle in the Business Unit Transmission.

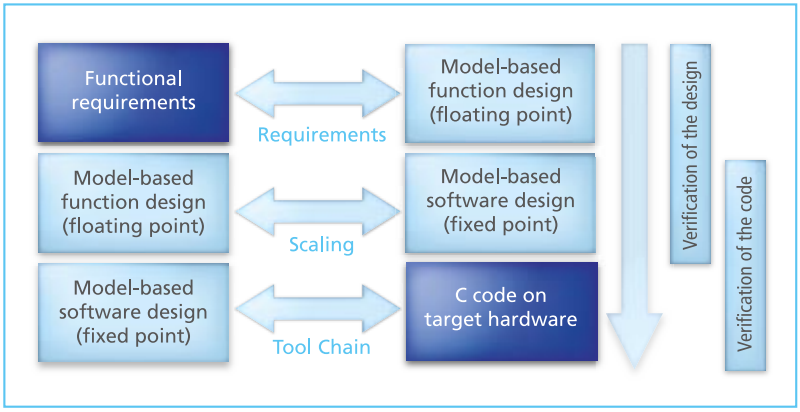


Fig. 2: Model-based testing – design and code verification.

At this the point the software architecture needs to be defined early, in congruence with the function architecture. The result is the partitioning into C modules and C functions, which enables the testability, configuration (archiving) and maintenance.

Model-Based Collaboration

Tight collaboration between all involved parties at the model level is necessary to ensure that the models are designed and exchanged consistently. Complex functionalities are partitioned into smaller units and placed in model libraries. Documentation that is based on the models replaces the traditional, hand-written software requirements specifications.

Model-Based Testing

To optimize the process, especially via automation, model-based testing and the corresponding quality measures are essential. The standard software quality measures are modified. This makes quick, micro-V-cycles possible, resulting in fast progress towards model maturity. Model-based testing offers the possibility of a step-by-step approach (see fig. 2).

> "Within the framework of an in-house study by the OEM, the result was that the handwritten code cannot measure up to the code generated by TargetLink in terms of freedom from error."
>
> *Georg Grassl, Continental AG*

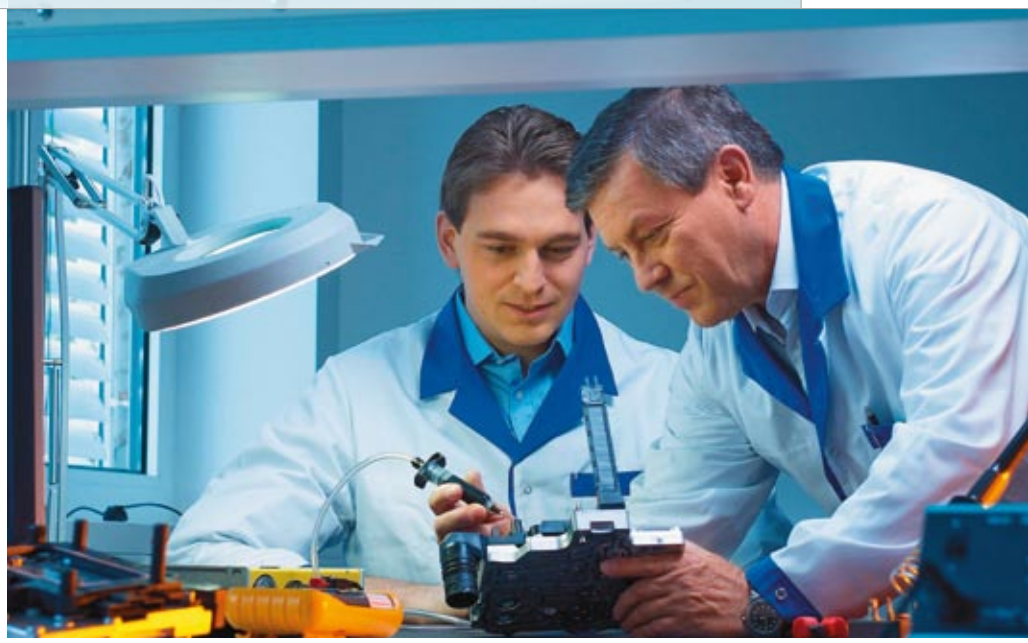
First Production Application of the New Process and Methods

Based on the results of the preliminary studies mentioned above, the first high-volume project was started with a defined, well-coordinated cooperation between the partners in all areas concerning model-based design and automatic code generation. The modeled algorithm – a subfunctionality of the application software – is a gear-shift strategy for a six-stage automatic transmission (see fig. 3) that should be used for a 16-bit platform. Due to their success in the preliminary studies, MATLAB/ Simulink and dSPACE TargetLink were used in accordance with the previously developed processes and methods. The test of the function software was performed according to conventional quality assurance procedures, which were already defined for the written C-code and adjusted to fit the conditions of automatic code generation.
To ensure compatibility with in-house code rules, a static code analysis was performed. At model level, structural coverage tests were performed and supported by tools for test vector generation. Thus, the quality goal of reaching 100% coverage of the fixed-point code (highest test coverage level: modified condition/decision coverage, MC/DC) was achieved. These module tests were performed with a standard tool that can simulate the microcontroller platform. Thus, the complete tool chain (code generator, compiler and linker) was verified successfully.

Results and Insights from the First Production Projects

The method we chose ensures that the tasks were performed at the required time, thereby supporting the stringent time frame. As a result, the time-to-market for this project was even shortened, which our customer applauded. Furthermore, the developed transmission software had to adhere to tight restrictions regarding resource consumption:

- a maximum CPU utilization of 15 %



The electronic control unit for double clutch transmission has to pass specific qualification tests in the Continental Automotive systems lab.

- a maximum ROM assignment of 100 kByte

With TargetLink, the values stayed far below these levels, whereas the size of the automatically generated function software was:

- maximum 10 % CPU assignment
- 60 kByte compiled code
- 50 modules
- 20 % ROM assignment of the entire application software

It became clear that the software freeze of the applied code generator version had to be performed, at the latest, when the software quality measures and code verification began. Any update to the code generator can lead to altered source code, which would make it necessary to repeat all software tests and quality measures. Within the framework of an in-house study by the OEM, the result during validation was that the handwritten code cannot measure up to the code generated by TargetLink in terms of freedom from error. By using model-based development and automatic code generation we were able to make an exceptionally high number of software deliveries precisely on schedule, thanks to the structured process for this new method. The method was sent to other users at different locations to be used as the standard method for transmission projects.

Development of the Function Software for Dual Clutch Transmission

In 2006, due to the positive experience when using the new methods and processes, we started the production development of hardware and software for a DCT transmission ECU (see fig. 4). The entire function software for this was created by using model-based development and automatic code generation with TargetLink. The tasks were divided as follows:

- The OEM makes the function requirements.
- Continental makes the model-based function design available, validated by using rapid control prototyping.

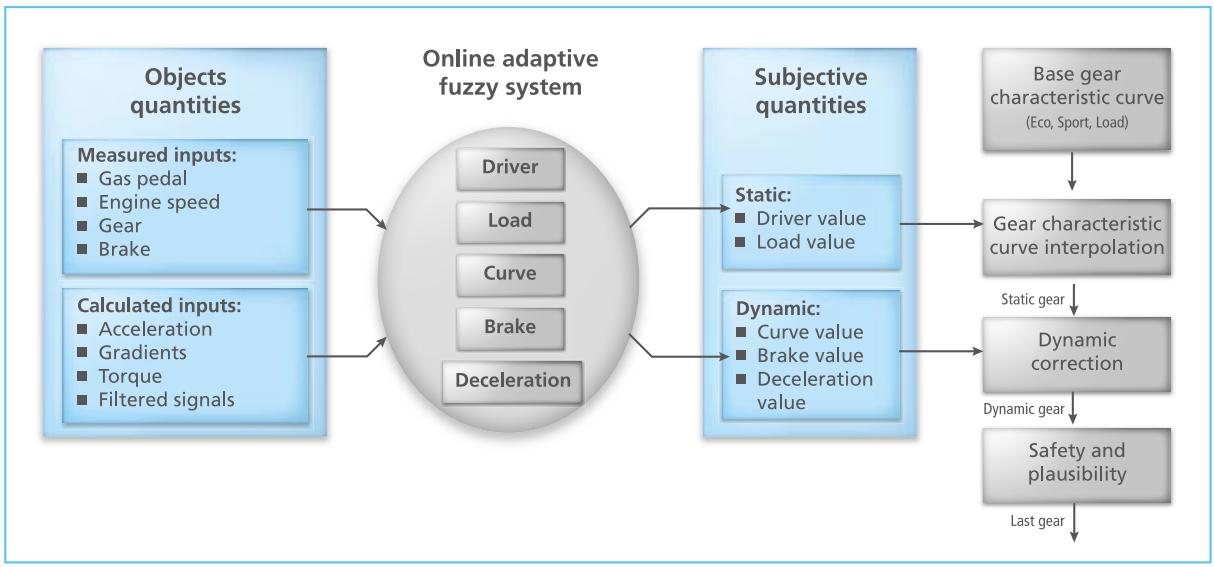


Fig. 3: Gearshift strategy – structure and signal flow.

- Continental, supplier of the transmission ECU, makes the hardware and software available (incl. the model-based software design, automatic code generation, model-based tests, and quality assurance).
- The OEM and Continental perform the system integration and validation of the function in the vehicle via hardware-in-the-loop simulation.

The handling of the process elements, the corresponding methods for model-based development, and the automatic code generation for this DCT project were similar to the handling of the first production project. During the project's run time, additional challenges arose due to the complex DCT functions:

- Function reuse
- Adapted code generation for device-specific calibration methods
- Partitioned design in a multi-user environment

The following three sections briefly describe how these challenges were met.

> "Using model-based development and automatic code generation led to an exceptionally high number of software deliveries precisely on schedule."
>
> *Georg Grassl, Continental AG*

Function Reuse

When looking at a DCT system, it becomes clear that two clutches and four gear components call for the reuse of functions (see fig. 4). The limited hardware resources make reuse an even greater necessity. Actuator functions for two clutches, two shafts and four gear actuators must use a common algorithm.
Simulink libraries are also used in this project for low-level functions (i.e. filter routines), resulting in "nested reuse". The requirements are fulfilled by TargetLink's Function Reuse feature, which made it possible to reduce the resource consumption. Since developers are aware that the functions can be reused, they can take this into account when designing the model architecture.

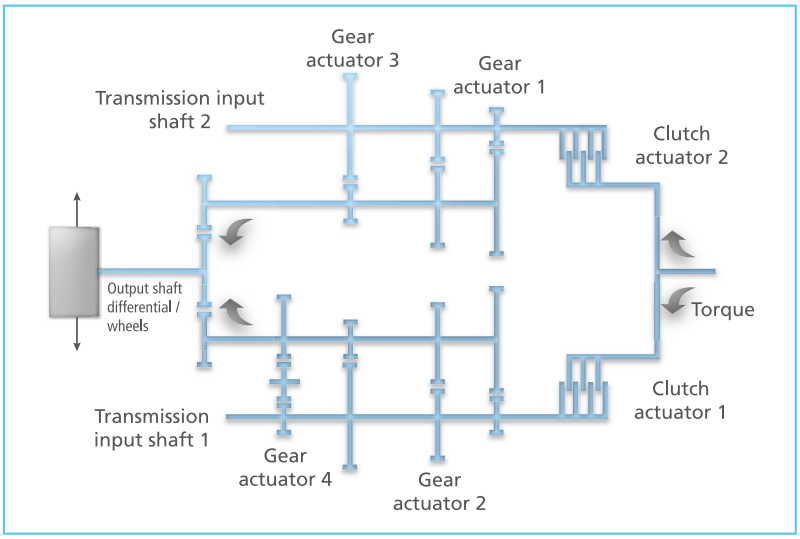


Fig. 4: DCT schematics.

Adapted Code Generation for Device-Specific Calibration Methods

In order to cut development costs in this project, a specific calibration method for electronic control units was to be implemented without calibration devices and/or memory expansion. As a result, a set of calibration data is organized as a structure, and access to this during run time is carried out by rerouting the pointer to the structures from the ROM to the RAM. To implement the necessary code pattern, TargetLink features such as variant coding and templates were used on the one hand (see fig. 5). On the other, we worked together with dSPACE to make changes in the code generator. The conversion of the entire DCT model to these specific calibration methods was completed within just one week, which proves how high the performance of the open program interfaces delivered by the code generator truly is.

Partitioned Design in a Multi-User Environment

In this project, the function design was performed by Continental as well as by the OEM involved. The high number of involved developers was much too large for the previously used single-user method, so the developed processes had to be expanded to meet multi-user capability. To be specific, large functionalities were partitioned into smaller model fragments and

Fig. 5: Definition of a pointer to data variants by using a variable template and assigning variants to parameters.

managed in Simulink model libraries. To keep the data within the environment consistent, the multi-user support offered by the include files of the dSPACE Data Dictionary was used.
The Transmission Team decided to keep the model, the data description, the test vectors (functional and structural), the test areas, and the fine-tuning guidelines as a package (see fig. 6) in the configuration management. This smoothes the way for reuse in other projects and for consistent development.

Results and Important Findings

Thanks to the previously developed processes and methods, 100 % of the DCT project's function software could be developed on the basis of models and autocoded with TargetLink. The function software that arose in this high volume project amounted to:

- 250 kByte compiled code
- 120 model libraries

Considering the challenges and corresponding solutions described above, it was important that we could place our trust in the cooperation with those who delivered the code generator. This was especially true for the specific support inquiries the project team had regarding memory assignment, data variants, and how robust the code generator could be. For model-based development, including automatic code generation, it was also helpful to have the updates and patches for the code generator synchronized with the project's software tasks.

Looking Ahead

With the know-how gathered by using the project-specific approach for a model-based design and automatic code generation, the Transmission Team is now ready to use this method for low-level software (actuator control). In addition, we are working on moving from a project-specific approach towards a platform approach, while still maintaining quick methods. This way, we can support the worldwide, in-house introduction of the development method, and still maintain consistency. ■

Georg Grassl
Business Unit Transmission
Gerd Winkler
Business Unit Engine Systems
Continental AG
Germany

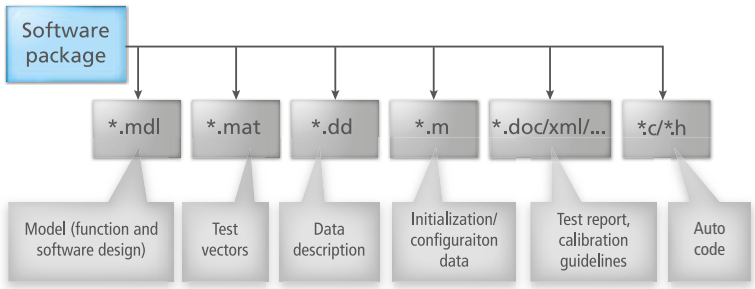


Fig. 6: Software package for multi-user environments.

Summary

The Business Unit Transmission implements a seamless method for model-based development and automatic code generation. The development phases are closely connected with one another by an executable specification (a model). Meanwhile, the entire function software for the DCT is implemented by using model-based design and automatic code generation. By working together with the OEMs, the Business Unit Transmission is able to set up an efficient process for developing function software.

The tool chain used for this method already proved its worth for high volume projects. The process is based on reliable elements from projects with conventional development methods, which were then adapted to the meet the requirements of model-based development and automatic code generation. This made it possible to introduce the new methods efficiently.

The process was done step-by-step: first, an internal project to implement the new methods; then the initial production project with a 20% share of the autocoded application software; finally, the highly complex DCT transmission with 100% automatically generated application code.

Using this new method resulted in immediate quality improvement. When comparing identically complex projects, one using traditional development methods and the other using model-based development with the resulting options (rapid prototyping, automatic production code generation, model-based testing), it becomes quite evident that model-based development is clearly superior to traditional methods.