7th LuK Symposium

11./12. April 2002

Publisher: LuK GmbH & Co.
Industriestrasse 3 • D -77815 Bühl/Baden
Telephon +49 (0) 7223 / 941 - 0 • Fax +49 (0) 7223 / 2 69 50
Internet: www.LuK.de

Editorial: Ralf Stopp, Christa Siefert

Layout: Vera Westermann
Layout support: Heike Pinther

Print: Konkordia GmbH, Bühl
Das Medienunternehmen

Printed in Germany

Foreword

Innovations are shaping our future. Experts predict that there will be more changes in the fields of transmission, electronics and safety of vehicles over the next 15 years than there have been throughout the past 50 years. This drive for innovation is continually providing manufacturers and suppliers with new challenges and is set to significantly alter our world of mobility.

LuK is embracing these challenges. With a wealth of vision and engineering performance, our engineers are once again proving their innovative power.

This volume comprises papers from the 7th LuK Symposium and illustrates our view of technical developments.

We look forward to some interesting discussions with you.

Bühl, in April 2002

Helmut Beier

President
of the LuK Group

Content

Code Generation for Manufacturing

The Electrical Central Release Bearing Software

Reinhard Ludes, AFT Werdohl
Thomas Pfund

Introduction

Software plays a very significant role in automotive technology these days (figure 1).

Particularly in the vehicle itself, the number and importance of mechatronic systems, which possess a high degree of integration of mechanical and electronic software-based functions, have increased markedly. The software here performs automatic control functions or automates behaviour previously performed by the driver.

The systems' functionality is growing increasingly more complex, and this complexity of system behaviour, as a result, is continuously shifting to the software. While more and more platforms are being used in the area of mechanical and electronic hardware, even across brands, the tasks of individualising the system and the vehicle's characteristics are increasingly being given to the software.

This trend is supported by the apparent ease with which the software can be modified. It is also encouraged by the emergence of high-performance processor systems, which are not only becoming more and more affordable, but are offering ever more functions and increased storage space. This is an irresistible temptation for ambitious development teams.

But what is often overlooked is the fact that more and more effort must be invested in software specification, implementation and testing and the validation of the entire mechatronic system, both on the test-bed and in the vehicle.

The reason is that the different methods of generating software have not kept pace with the demands made on software systems. They have also not kept pace with the complexities created or caused daily by large development teams.

Of course there is and has been progress in programming languages: a number of testing methods have been established to optimise the generation process and language extensions like object-oriented programming (OOP), which are also slowly becoming established in the area of real time programming. The fact remains, however, while it is making significant contributions to control and automation, software itself is generally still written by hand. Software development is thus still a classical manufacturing process.

The present project takes an alternative route in developing the software for an electrical central release bearing. Using TDC, the abbreviation for Total Development Chain, by **AFT (Atlas Fahrzeugtechnik)**, the required control software is generated automatically. A full description follows.

software in automotive technology	
control& automation of functions, tasks and procedures	
in the vehicle:	
• **automatic control functions**	**injection, ignition, engine management, ABS, ASR, ESP, etc.**
• **automation**	**clutch (TCI), transmission (ASG)**
in development& production:	
• **methods& tools**	**CAD, CAE, CAM**
• **automation**	**robotics**

Fig. 1: *Software in Automotive Technology*

The Standard Development Process for Software

The standard process for developing software is outlined in the section to follow. Naturally, there are many variations, but we will mention only the major and generally typical steps here (figure 2).

First, the control strategy is formulated based on an idea. This can be accomplished by writing a performance specification for the control strategy. An advanced procedure includes the specification and all types of detailing of the controller design using a simulation program, where its structure and configuration is continuously improved based on models. The advantage of this method concerning written specifications and drafts, is that it is easy for professionals specialising in different areas, such as systems analysis, control and simulation to discuss the control unit model.

Once the control or automation function – at least in the area of simulation – has reached a sufficient level, in the classical method, it is now the software expert – whose qualifications come from the area of real-time software and microcontroller design – who is ultimately responsible for the implementation.

The basic controller concept is the beginning of a software expert's work. They must also first analyse the task given to them and create an appropriate design concept for the real-time system before they can even begin to think about implementation, i.e., translating the functions into concrete code. The reason for this break, or gap in the process, as can also be seen in the diagram, is the existence of different points of view, which are expressed in different model approaches.

In controller design, the main consideration is the object to be controlled, such as, in the current case, the control motor as the actuator and the clutch as the controlled system.

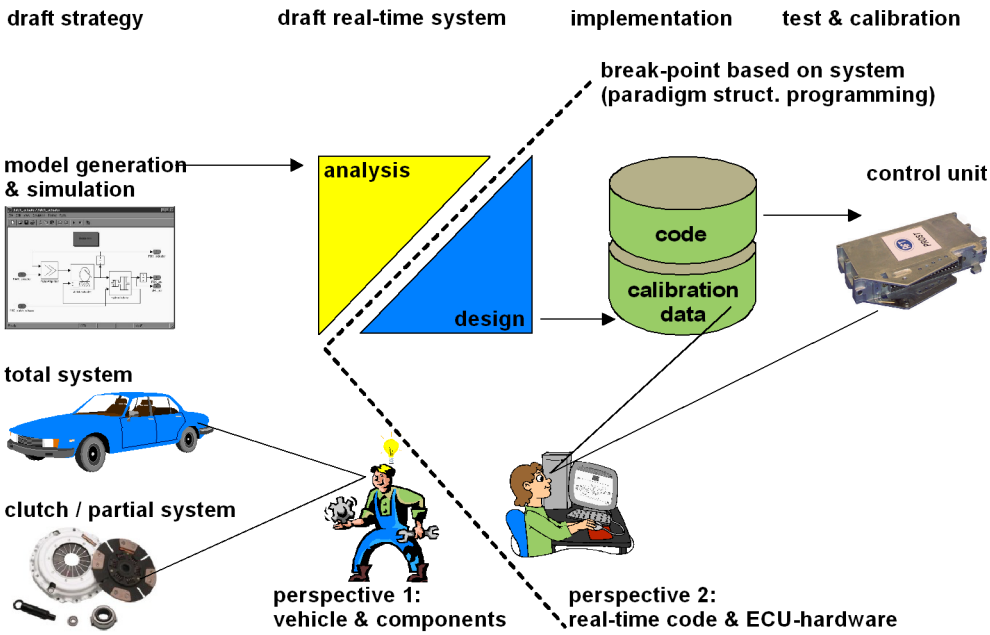

Fig. 2: *Standard Development Process for Software*

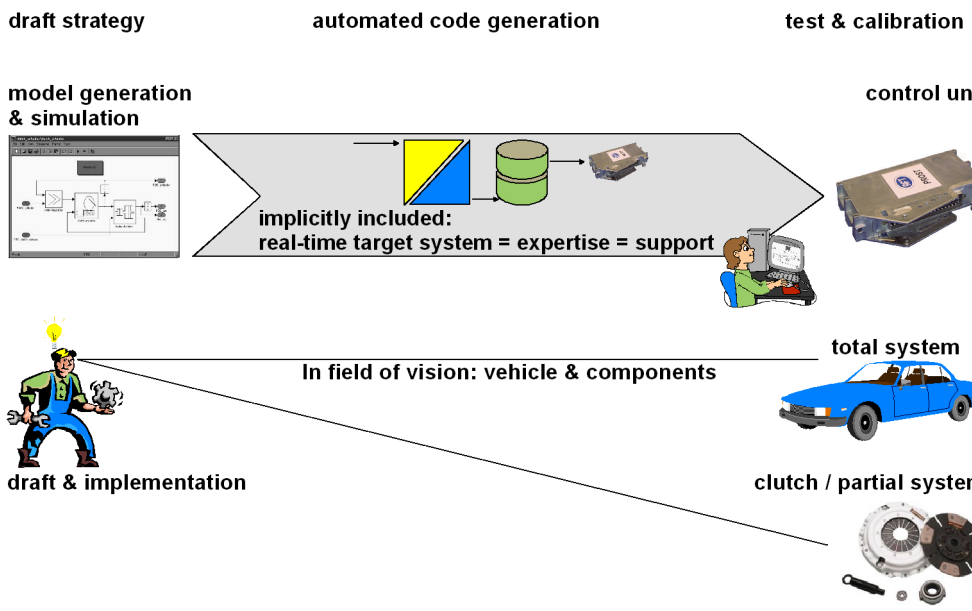

Fig. 3: *AFT TDC in the Development Process*

All models are oriented in this way. In real-time software design, as in any other software design, the architecture of the computer or microcontroller system is the main issue. The models used as the basis are thus primarily oriented toward the standard architecture of computer cores or processors (Von-Neumann machines), and all standard procedural programming languages are designed on this basis.

The severity of this break is different, and depends on the languages and methods used. The break is, however, part of the basic nature of the task. Only object-oriented programming shows a way to eliminate this.

There are several approaches for working around this break. Above all, the effects depend on the development process; how well the communications between the two basic developer types work.

At LuK, for example, we have been successful in providing both qualifications at the same time in the team or with individual developers. In addition, code segments are transferred from the offline simulation to the source code level in the control software.

Code Generation in the Development Process

In this project, we have taken the road of automatic code generation. The code generator used here takes the controller's model, or simulation-based specification, and generates close-to-production real-time code for the microcontroller in the automotive control unit. This is therefore a seamless path; from the specification of the general, to the detailed strategy, to the real-time code in the control unit (figure 3).

This approach is in line with LuK's philosophy of using only one expert or group of experts to design and implement the entire system. It is also the system engineer who designs the control unit in the classical method. The knowledge of how to generate the optimum code is implicitly present in the code generator. The functions relating to the vehicle or one of its components are thus kept in mind from the beginning on. The processor, the programming language and the operating system that are used in the concrete application remain in the background.

The break in the specification or the entire development chain is overcome.

Using TDC in Electric Central Release Bearing Engineering: The Closed Tool Chain

The use of the AFT TDC means more than just using a code generator. TDC reflects a heterogeneous, but closed chain of tools, which supports the entire development process. Heterogeneous, because different tools are used together – in fact the exact tools that are best suited for the specific project. Closed, because the results of the test are fed back in to improve the original model in a gradual evolution (figure 4).

In this project, we used MATLAB®-Simulink for simulation, TargetLink for code generation, AFT **PROST** as a suitable prototype control unit, and **MARC I** for recalibration in the test phase. It is also worth noting that this data, which is used for calibration, i.e., fine-tuning, is already specified in the simulation software. For the calibration system, description files are also generated (according to ASAP 2), which are automatically read into the calibration system. The test engineer therefore does not need to perform any additional configuration to his calibration system.

The data obtained during the calibration can also be worked into the simulation model for fine-tuning the model. Furthermore, the measured data obtained in the test can be fed back into the simulation from the calibration system or from additional installed measuring systems. The dataloggers ***RAM*boX™** and **TORnadO®** are available at LuK for this purpose and are currently in use.

Finally, only the code generator and a series of expansions to the calibration system had to be installed in addition to be able to form the development chain as described. All additional tools were on hand and known to the developers.

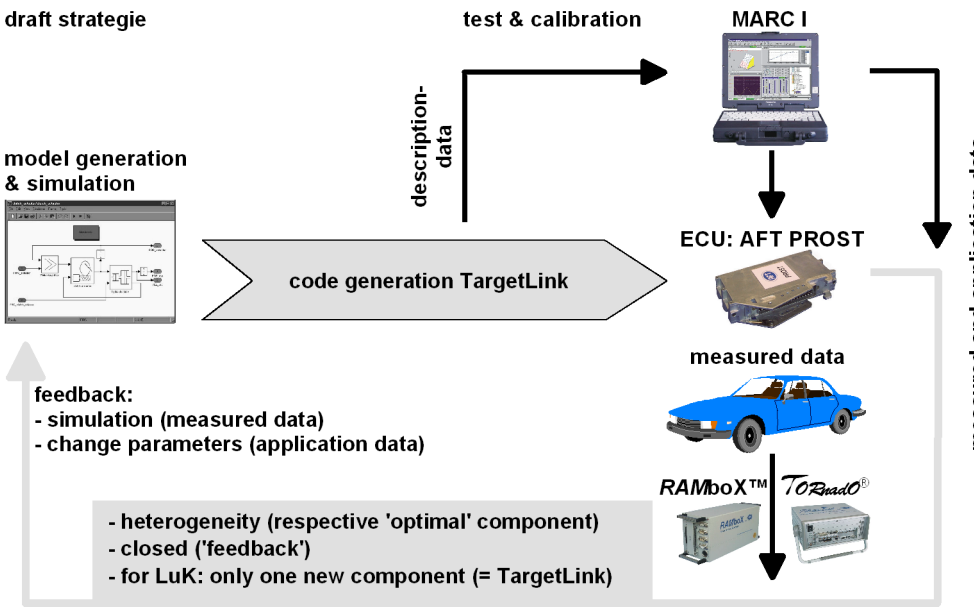

Fig. 4: *Characteristics of AFT TDC: Example – LuK Electric Central Release Bearing*

The Electric Central Release Bearing Project as Reflected in the V Model

As indicated in the previous illustration, TDC reflects a development method that goes beyond pure code generation. It is a simulation-based development method with executable specifications. The entire chain is based on the V model. Figure 5 shows the implementation of the V model with all components that are used at AFT to develop software for control devices.

The following are characteristic of the V model as the basis of the development strategy:

- Each design step on the left has a corresponding test and validation step shown opposite on the right.
- Transfers and tests are required between the steps on the left side to ensure that the specifications given in the previous step are also fulfilled.
- This also means that the later an error is detected, the more steps must be rerun during the redesign, i.e., the more difficult and expensive correction becomes. Basic specifications that are not or are improperly met, which can then be discovered only in the final step, are thus especially critical.

The left side shows the two significant design steps:

- Specifications phase – the system specifications are made to provide clear definitions, in some cases based on the simulation models and automatic condition generators.
- Model generation – based on the system specification and any existing partial models, a structure that can be simulated is generated, embedded in a more or less exact plant model.

Both steps use MATLAB® SimuLink, first for the rough system structures and specifications and in the second step, the entire model of the control unit and controlled system is defined in detail.

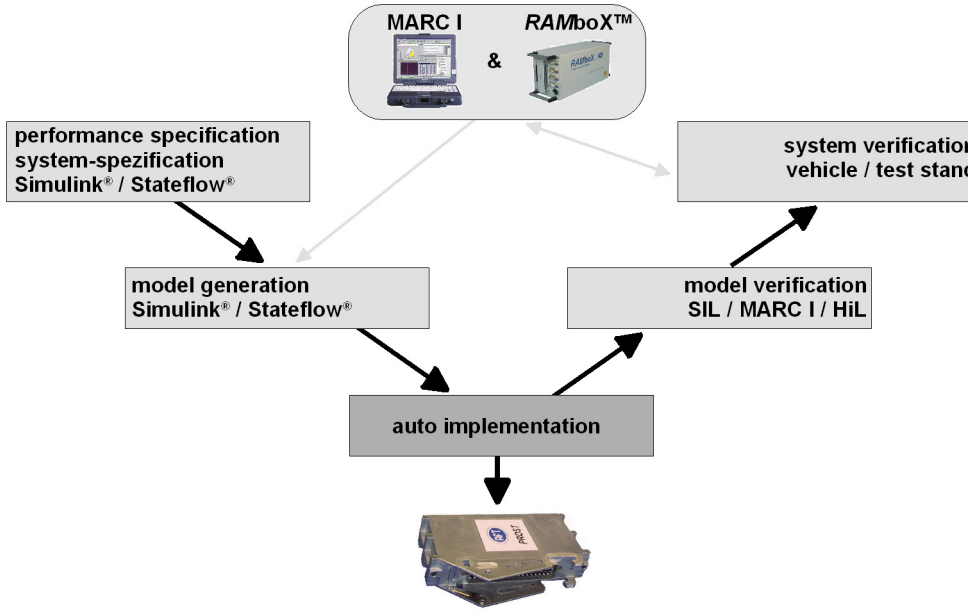

Fig. 5: *Electric Central Release Bearing Project Reflected in the V model*

The process ends in the peak of the V model with auto-code generation on the specific target system. Thus the original control unit specification is not only executable in a simulated environment, but also on a control unit as the target system. The entire chain is continuous.

On the right side, across from the two design steps are the corresponding test and validation steps:

- Model-verification, i.e., validation of the basic control function, in the SIL simulation (SIL = software in the loop) or if applicable already in a HIL system (HIL = hardware in the loop).
- Total system verification on the test stand or test section.

The controller is first tested in an SIL simulation. In the SIL simulation, all interfaces to the specified software are depicted as models, so that the simulation can run without any additional hardware expense. The system supports this process in that the I/O interfaces to the controller – operating system are available as TargetLink – blocks and that a simulation is possible both with floating decimal and fixed decimal arithmetic.

A HIL system is used for model verification when individual components of the controlled system cannot, or can only with great effort, be represented in the simulation environment. However, at the same time a test in the entire system, e.g. due to high application expense, is not or not yet economically feasible. It must also be decided on a case-by-case basis which parts in the controlled system must be physically present and which can be represented as a software model.

No decision must be made for the control unit, whether it will be 'operated' for real or in the real-time simulated controlled system. It is worth noting that in this simulation-based development process, the system model required for the HIL simulator is largely already available from the system specification step. In the project described here, the main partial systems were installed on a test stand (central release bearing against non-rotating clutch, PROST control unit and higher-level TCI control unit).

The system verification is based on the overall system design. If the behaviour of the clutch is determined in the vehicle, the validation must also be completed in the vehicle or in a vehicle-adequate system environment. In this case too, the final verification was performed in the vehicle.

Thus, all stations of the V model were run in this project.

Additionally, the measured data that was obtained in the control design stage and in the final test was fed back.

Results, Part I: The Controller Structure

The specification of the controller consisted at first consisted of the initial rough blocks, which were then filled out in refinements and parameter configurations. The blocks can be output graphically directly with their links (signal paths) and thus can support an optimal specification and documentation:

- the target and actual value calculation for the controller
- the condition controller as the core
- the pre-control to raise the adjustment dynamics
- the logic, which differentiates different basic conditions (e.g. normal, 'limp-home' and test operation)
- the follow-up functions, which encompass different engagement and disengagement procedures
- the block 'set outputs' for checking limit values and adjusting to the hardware
- the block 'measured values on CAN' as a control function during the test phase

A graphic display will be created automatically. The integration at this point, however, is such that it is far too extensive and detailed. Each block can be further broken down to

show more detail. The controller as an entire system is once more a block in the entire offline simulation model, in which the remaining components of the controlled system are represented: end stage, electric motor, the belt gears and the clutch itself.

Since the controller's specifications finally end in a controller model that can run both 'against' the model of the controlled system in the simulation and after code generation in the control unit 'against' the real controlled system, it is an executable specification.

When the controller is in a real control unit after code generation, the generated software accesses the same input and output interfaces as it already showed in the model. The interface to the system software then makes up the AFT controller interface (ACI).

Results, Part II: Savings Potential

Three man-months were originally planned to generate the controller for the electric central release bearing: to design the simulation, implement the code and for initial testing. The project was completed two weeks after the project's start using the new method. The new method also required a total of three man-months.

No control group was defined, which solved the same problem using conventional methods. There is, however, sufficient experience with the definition and execution of such projects, that we can assume that approximately the same amount of work was required.

Some of the effort certainly went into breaking in the new tools. The first decision was made after two weeks to change the method which cost a few extra hours, since the simulation models had to be readjusted to the real-time requirements.

It is estimated that the development time for trained employees can be reduced about 50% to 1.5 man-months. This results first from an analysis of the time records and secondly from the back-calculation: According to experience, 1.5 man-months are required to write the coding for the controller, including testing and configuration of the calibration system, an expense that is eliminated with the new method.

The possible ratio effect is thus a good 50% and results from eliminating the generation of the controller code and the time used for fine-tuning in the test phase, which is required to configure the calibration system.

It should be mentioned that validation steps required for production release (functional continuous operation, code inspection, etc.) were not included in the time estimate. If these were taken into account, the ratio effect would be reduced.

It is very important to establish at this point that this is not a purely functional representation on a high-performance automotive test system.

The AFT **PROST** prototype control unit includes a standard microcontroller with fixed-point arithmetic, suitable for production. Likewise, the generated code is transferable to a conventional production unit without any great effort.

This design, because it is closer to production than other systems that are based on high-performance floating-point processors and only allow a functional depiction in a prototype vehicle, is thus far superior.

Answers to Frequently Asked Questions

What is the Source of the Effectiveness of the Code Generator? What is it Based on?

The effectiveness is based on the execution time of the generated code. It is between 0.9 and 1.2 times that of hand-written code, according to the manufacturer's specifications. This is an excellent amount, since previous code-generators generated code with 2.5 to 3 times the runtime. Even in the worst case scenario, 20% additional runtime is absolutely

acceptable. It requires approximately the same memory space as hand-written code.

This information was not tested in the course of this project; the same tasks would have had to be performed by experienced programmers at the same time. However, the runtimes and file sizes are within the expected ranges.

There are essentially three reasons for this run-time efficiency:

First there is the use of special codes of the target processor that are not available to C programmers. Second, the code generator implicitly has much special knowledge of the efficient programming of the target processor and the operating system environment used, which a code-writer would first have to painstakingly establish. Furthermore, with proper handling of the tools, the variables are scaled to the value range used. This allows the use of favourable multiplication and division operations, which would otherwise require a great deal of time. The scaling can be done automatically or by indicating the value range.

Is the Generated Code Readable?

The code is readable. Variable names are likewise taken over in a changed, but readable form. A prefix or suffix is simply added to connect the data structure with the higher-level system structure.

Certainly this question is related to the fact that there is a certain level of mistrust on the part of developers with regard to generated code. Even code generated by high-level language compilers is not generally subject to inspection by the developers.

How Can I Work with Projects that Go Back to Existing Code?

AFT TDC has particular advantages over other development tools in this area: There are generally three sources (figure 6):

1. Code generation from the SimuLink models as previously described.
2. Adding hand-written code in addition.
3. The option to integrate code from other projects.

In all cases, if the conventions are followed, even the parameter-setting for the calibration system, i.e. the description data of the control unit, can be generated.

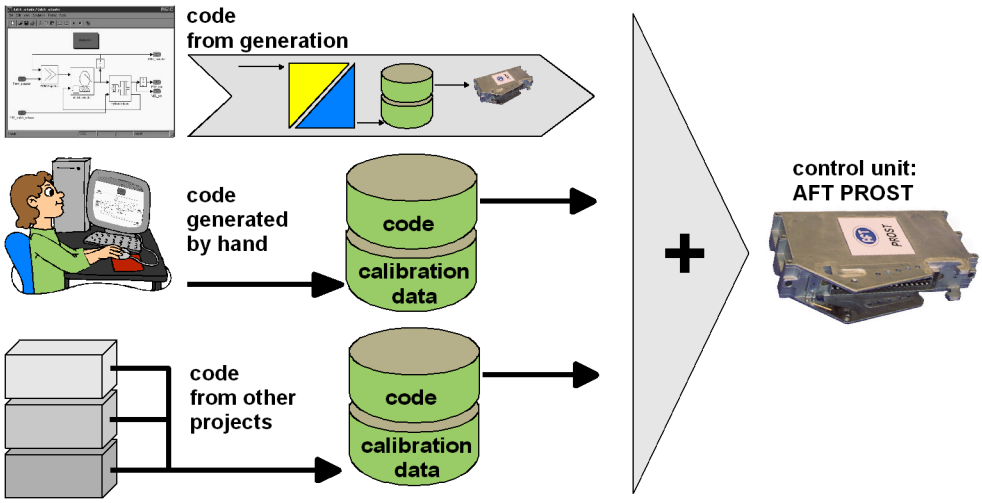

Fig. 6: *An Advantage of AFT TDC: Code Mixing*

To What Extent is the Generated Code Suitable for Production?

As mentioned previously, the code is nearly production-ready, since it is generated for conventional microprocessors as the target system. The function unit used – the automotive prototype control unit AFT **PROST –** is likewise close to production-ready. The results are thus transferable to the subsequent series development 1:1.

Summary and Outlook

Figure 7 summarised once more the general advantages of a closed development chain and the specific advantages of AFT TDC. These advantages originate from the typical characteristics, also shown.

The development of the central release bearing software was a pilot project, which the performance capabilities of the AFT TDC for the design and generation of the control unit production code covered. At LuK Bühl, the current thinking is, after this pre-development project, to also run a pilot project in production development.

Other AFT projects have been completed successfully in the past. These include projects for DaimlerChrysler research, who – after appropriate training of their employees – have since used the same tools and methods for their own development projects.

AFT carries out other projects as engineering service, using AFT TDC as the method and tool to complete projects quickly and cost-efficiently. These also include the development of control unit software for LuK-FH in Bad Homburg.

summary	
general features of TDC:	**general advantages of TDC:**
• **General simulation-based development process from initial design to implementation, including all test phases (SIL, HIL, application and testing)** • **High-level design (model-based graphics)** • **Executable specification** • **Implicit documentation** • **Expertise real-time system, partially in code generator**	• **shorter development times** • **reduction of possible error sources** • **cost-reductions** • **shifting the required expertise to the system engineer** • **tested intermediate results in each development step**
basic characteristics of AFT TDC:	**specific advantages of AFT TDC:**
• **Heterogeneous system** • **Combination of the best components that have proven themselves as standard: MATLAB®/SimuLink®, code generator by dSPACE, MARC I, *RAM*boX™, TORnadO®**	• **adaptable to changing standards** • **can be integrated into existing projects: combination or mixture of hand-coded and automatically generated code** • **adaptable to individual requirements by combining for the customer** • **production-ready code**

Fig. 7: *Summary*