Originally published at Embedded World, 17-19 February 2004, Nuremberg, Germany

Embedded Steer-by-Wire System Development

Joachim Langenwalter and Tom Erkkinen
The MathWorks

Abstract

Model-based design enables the automatic generation of final-build software from models for high-volume automotive embedded systems. A software engineering framework is needed to support this.

This paper presents a framework of processes, methods, and tools for the design of automotive embedded systems. A steer-by-wire system serves as an example.

Introduction

Production code successes have recently been reported in various industries from companies including Denso, Motorola, and Toyota [1]. This technology is taking hold as an important component within the next evolution of software development.

Although the impact on the software engineering process as a whole is understood, it has not been clearly established. This is especially apparent to participants of previous, similar evolutions including the migration from machine code, to assembly code, and then to source code.

With increased abstraction and automation came new processes, methods and tools. Waterfall processes have fallen by the wayside in favor of spiral and iterative approaches. Real-time methods have appeared, displacing static flow design. New tools have arisen such as IDEs with debuggers, optimized compilers, and automated testing tools.

However, due to difficulty of use, lack of understanding, or limited tool support, not every good idea has flourished. Evidence shows that these methods and tools aren't always practical for mainstream production usage. For example, formal methods where proofs are used to ensure software correctness are written in a language that only a few experts worldwide truly understand. Further, real-time case tools in the 1980s aided design, but did not provide an easy path to final code.

Production code generation has done well in the early adoption stage, due mainly to its practicality. However, further growth requires integrated process, methods, and tools to support it. A new process will be successful only with the methods and tools needed to support it. If one of these pieces is missing, the effort to reengineer a company's mature embedded system is no longer feasible or practical.

This paper presents such a framework focused on production code generation:

- **Process** – Model-based design
- **Methods** – Modeling, simulation, rapid prototyping, production code generation, model testing and coverage, and in-the-loop testing
- **Tools** – Development tools, V&V tools, and integral tools

Process

Model-based design supports the needs of controls/DSP systems engineers and software developers by providing a common environment for graphical specification and analysis. In this process, models are made and used to specify system data, interfaces, feedback control logic, discrete/state logic, and real-time behavior.

Model-based design is used in nearly every industry that requires embedded control systems development. It is particularly well-entrenched in development processes for embedded applications such as large-scale

automotive electronic control units. DSP and communications applications also use this approach, but emphasize modeling and prototyping rather than production code generation.

To satisfy these various applications, the model-based design process must address the needs of safety-critical systems such as steer-by-wire systems. The process must also yield a final, executable code that is extremely compact, fast, and traceable. This is due to the high-volume nature of mass-produced ECUs, which necessitate the use of low-cost, fixed-point microcontroller units and DSPs.

Model-based design fits within the context of any process framework, including those characterized in IEEE Software Engineering Standards [2].

IEEE Std. 730 applies to any general-purpose software project. A good understanding of its process framework results from a review of its stated requirements for “critical” project documentation.

IEEE Std. 730 requirements include:

- Software Requirements Specification (SRS)
- Software Design Description (SDD)
- Software Verification and Validation Plan (SVVP)
- Software Verification and Validation Report (SVVR)
- User Documentation
- Software Configuration Management Plan (SCMP)
- Other documents including Software Project Management Plan (SPMP)

A common way to view a software process is through use of the V diagram, as shown in Figure 1. The diagram corresponds to most engineering processes, however, the process is iterative with many repetitive steps throughout the development life cycle. The software process in this diagram is composed of:

- **Development** (requirements, design, coding, and integration)
- **Verification and Validation** (V&V)
- **Integral** (software configuration management, requirements traceability, and documentation)

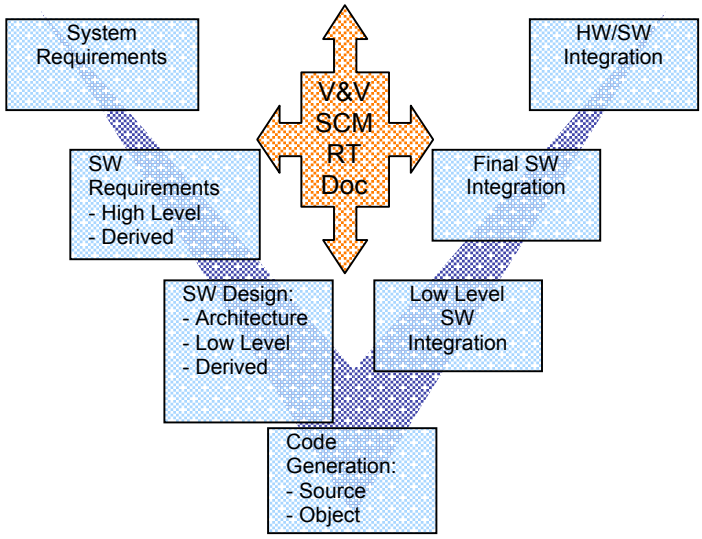

Figure 1: V diagram of software process

Model-based design places great emphasis on process iterations, early testing, and reuse throughout the development process, making it both unique and powerful. The practicality inherent to this process is demonstrated at the bottom of the V: production code generation is an automatic transition from design.

In model-based design, a block diagram or a state diagram model can serve as the system and software requirements, software design, or, with a slight change of perceptions, the source code.

Also unique to this process is the extensive V&V effort that is made prior to a final build. The benefit of early V&V is clear: fewer bugs will be found and less rework will be performed during final system integration and test. Catching a bug on the desktop is highly preferred to encountering it during a winter test drive in Finland. Organizations can

leverage this benefit to achieve faster time to market.

Methods and Tools

Model-based design methods are employed during the software engineering process.

The development methods include:

1. Behavioral Modeling
2. Detailed Software Design
3. Distributed Architecture Design
4. Production Code Generation
5. Embedded Target Integration

The V&V methods include:

a. Simulation and Analysis
b. Rapid Prototyping
c. Model Testing and Coverage
d. Code Tracing and Reviews
e. Hardware-In-the-Loop (HIL) Testing

The Integral methods include:

a. Source Control Interface
b. Requirements Management Interface
c. Report Generation

A concise description of each development method is provided below, with examples and tool support information. Note that all the tools shown herein are commercially available [1]. The following sections step through the development activity and include key V&V methods. Finally, the paper will conclude with the integral components.

1. Behavioral Modeling

Models are used for specifying requirements and design for all aspects of individual subsystems (e.g., steer-by-wire).

A typical system includes:

- Inputs (e.g., steering wheel sensors)
- Controller or DSP model
- Plant model (DC motor, rack and pinion, wheels)
- Outputs (change of direction)

In Figures 2 and 3, a system model can be created to represent the desired behavior using control system block diagrams for feedback control, state machines for discrete events and conditional logic, and DSP blocks for filters.

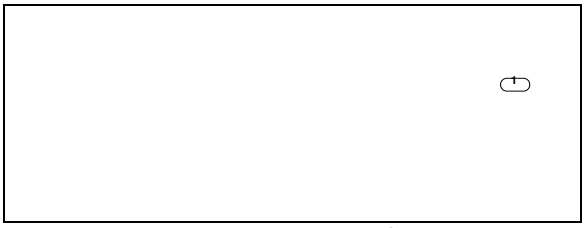

Figure 2: Feedback model of a PI controller for steer-by-wire

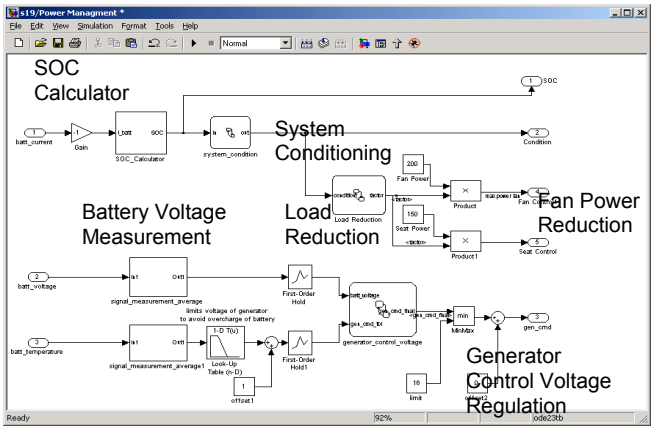

Figure 3: Power management to maintain power level for x-by-wire system

Simulation and Analysis

The model is then executed and analyzed to ensure that the requirements are satisfied, using methods such as time- or event-based simulation and frequency domain analysis. For example, a steer-by-wire system must respond to a sensor failure and “shall attenuate high-frequency response below 3 db and not lag commanded rate by more than 1.5 m/sec.”

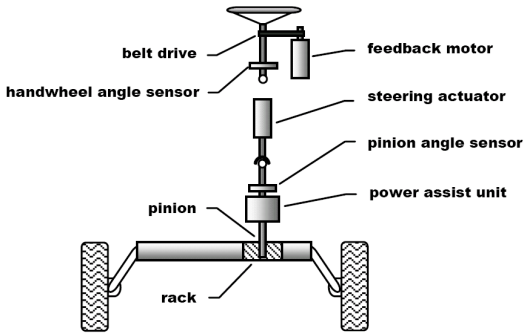

Figure 4A: Steer-by-wire system

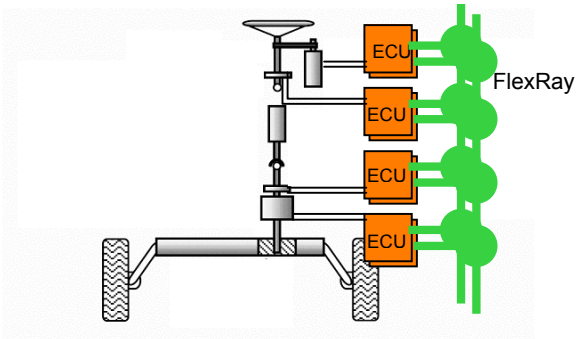

Figure 4B: Steer-by-wire system with fault-tolerant redundant bus system (FlexRay)

Modeling and simulation of the steer-by-wire system in Figures 4A and 4B determines if these requirements conflict or are valid. Simulation is a core validation activity and ensures that a system can be realized to satisfy the requirements.

Rapid Prototyping

Due to inaccurate plant models and insufficient processing power required to get a working solution on production silicon, modeling alone does not provide the total solution.

To overcome these shortcomings, rapid prototyping is highly useful because it replaces the plant model with the physical plant. In the steer-by-wire example the plant might be a car, and in that case, an actual car is used. However, because the system is not yet built, a real-time or embedded platform runs the controller software and interacts with the plant.

There are two forms of rapid prototyping: functional and on-target. Functional prototyping uses a powerful real-time computer such as a multiprocessor floating point PowerPC or DSP system. The goal is to determine if the system controls the physical car as well as it controlled the modeled car. If so, the plant model inaccuracies are shown to be insignificant, and the control strategy is validated.

On-target rapid prototyping executes the software in the same or similar production MCU or DSP, rather than a high-end PowerPC core or other dedicated high-end rapid prototyping hardware. The goal is to download the code into the actual production target for quick testing with the physical plant. If it performs well, the controller is not only deemed to be valid, it can feasibly be realized in production.

2. Detailed Software Design

The software design activity includes fixed-point data specification, real-time tasking, data typing, built-in-test, and diagnostics.

With model-based design, the same model used for algorithm specification and validation is refined and constrained by the software engineers as part of the production code generation process.

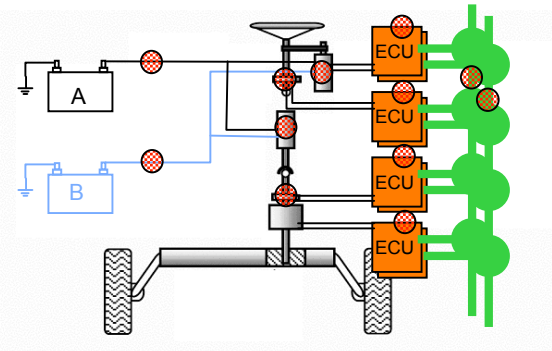

Figure 5: Possible faults on the steer-by-wire system

Model Testing

It is more beneficial to test the model on a desktop rather than deploy it on hardware for build and integration. Source code-based testing has existed for many years, but recent methods now allow for model testing and structural coverage. The usage scenario is that a developer fully stresses the controller to verify its design integrity using simulation and coverage. Another type of testing is a Failure Mode Effect Analysis, to ensure the save operation of the steer-by-wire under fault condition, as shown in Figure 5.

Examples of poor design integrity are numerical overflow and dead code. Stress testing of the model using minimum and maximum numerical values helps ensure that overflow conditions will not occur. This type of testing is easy with simulation, but dead code is not so easily detected because detection requires structural coverage. Dead code differs from deactivated code in that the latter is known to the developer and is deactivated for a reason. Actual dead code means something was missed during specification.

Model coverage assesses the cumulative results of a test suite to determine which blocks were not executed or which states were not reached. Certain types of coverages are well established in source code languages (such as C and C++) but coverages are now available for the model [3]. This effort required new theory and tools not needed (or possible) for C, since these languages do not posses constructs such as blocks or states.

Modified Condition / Decision Coverage (MC/DC) is considered by the FAA to be the most stringent coverage level necessary to satisfy safety-critical systems [4]. This coverage, among others, is now available within a model-based design framework and in many cases also required for x-by-wire designs.

Model Hierarchy/Complexity:	Test 1					
			D1	C1	MCDC	
1. req_test	35	84%	70%	50%		
2. . . . Logic	25	78%	75%	50%		
3. . . . . SF: Logic	24	78%	75%	50%		
4. . . . . SF: Altitude	11	100%	83%	67%		
5. . . . . SF: Active	4	100%	NA	NA		
6. . . . . SF: GS	13	61%	67%	33%		
7. . . . . SF: Active	6	50%	NA	NA		
8. . . . . SF: Coupled	3	33%	NA	NA		

Figure 6: Coverage for power management design in Figure 3

3. Distributed Architecture Design

Modern embedded systems contain several distributed ECUs, which communicate in real-time with each other over a fault-tolerant communication system like FlexRay. The latest DSC (Dynamic Stability Control) of BMW contains ABS as one of 15 sub-functionalities [5]. By adding blocks of network components such as hosts, tasks, and signals from DECOMSYS [6] to the individual subsystems, the embedded functions can be connected and mapped onto an architecture of ECUs. Furthermore, it facilitates the simulation of the temporal behavior of task activations of a time-triggered operating system such as OSEKtime/OS. Clusters, hosts, tasks, and connections are designed and simulated within the MATLAB / Simulink environment. Finally, the whole design integrates seamlessly with the DECOMSYS::DESIGNER product for interaction with the FlexRay xCDEF design data repository.

The distributed network design solutions from Vector (DaVinci) and Cadence (SysDesign) integrate the Real-Time Workshop / Real-Time Workshop Embedded Coder from Simulink / Stateflow generated code from subsystems and map them onto the architectures for verification.

4. Production Code Generation

Now that the model has been verified and validated, it is time to generate code. As with a compiler, this process is straightforward. Various optimization settings and user configuration options exist. The key is to keep the code efficient, accurate and integrated with legacy code or other tools. It is also important for the code to be traceable to the diagram, so that it may be reviewed and verified.

Code Tracing and Reviews

Figure 7 shows an automatically linked HTML report. When the developer selects the Sum Block in the code, it highlights the Sum Block in the diagram.

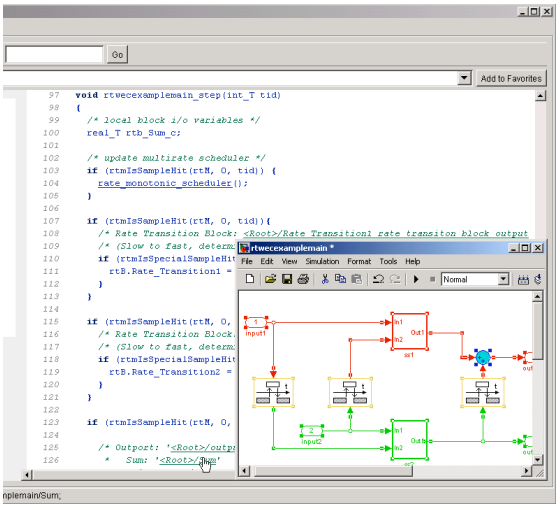

Figure 7: Code review

5. Embedded Target Integration

The application in Figure 7 uses Rate Transition blocks. However, direct links also exist to commercial RTOSs, including VxWorks and OSEK variants. As shown in Figure 8, device driver integration is also needed.

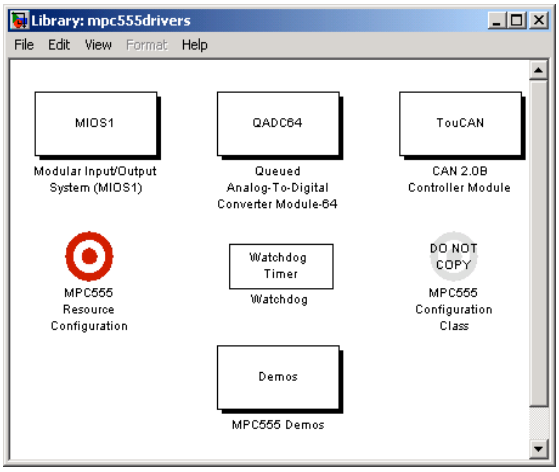

Figure 8: Device Driver block library for Motorola MPC555 used in the steer-by-wire system

Hardware-In-the-Loop Testing

Once the controller has been built, a series of open- and closed-loop tests can be performed with the real-time plant model in the loop. One example involves only the processor, and is known as “processor-in-the-loop” testing. The second example uses the actual built ECU hardware, termed “hardware-in-the-loop.” In either case, the physical controller is tested with the plant model. Through a series of tests, perhaps the same test used during requirements validation, the controller must be shown as acceptable to the customer.

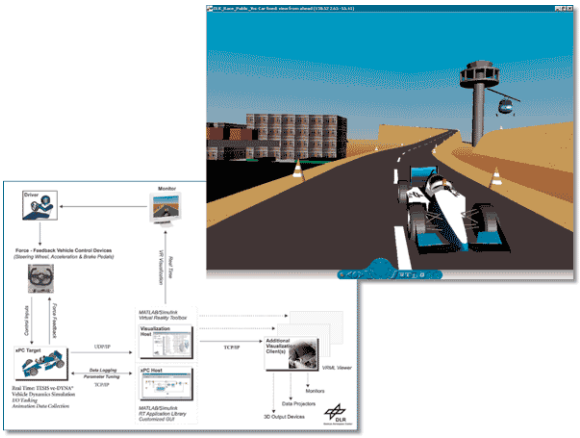

Figure 9: HIL test at DLR Germany with force feedback

Integral Components

Most software standards require traceability of requirements, perhaps originating in other requirements tools, throughout development. Also, software configuration management (SCM) is needed to store, version, and retrieve the various development artifacts. Documentation via report generators ensures management, customers and suppliers will see the model. The SCM interface is shown below in figure 10.

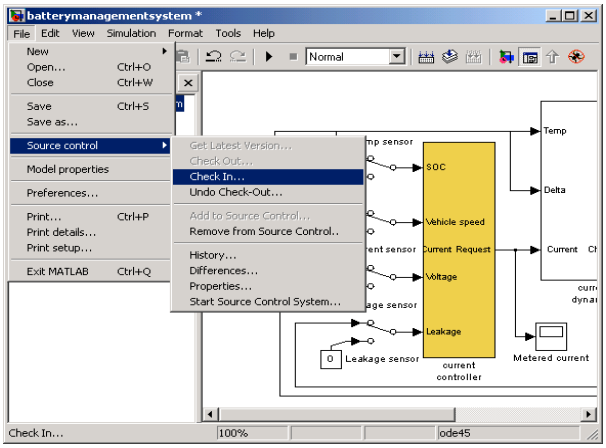

Figure 10: Source control interface

Conclusion

Major software evolutions occur when the full software engineering process activities are supported. Improving bits and pieces alone is insufficient. This paper described a full software engineering framework for model-based design and production code generation. Specific methods and tools were shown to illustrate that this is not just theory, but rather it is both practical and available.

Each topic presented easily contains enough content for a separate paper or book. Readers who wish to learn more or care to exchange ideas regarding additional methods and use cases are encouraged to contact the authors.

References

[1] www.mathworks.com
[2] www.ieee.org
[3] B. Aldrich, “Using model coverage analysis to improve the controls development process,” AIAA 2002
[4] "Software considerations in airborne systems and equipment certification," RTCA/DO-178B, RTCA Inc., Dec. 1992
[5] Dr. Michael von der Beeck, ARTIST Industrial Seminar, Paris, 23.4.2002 http://www.artist-embedded.org/PastEvents/Kickoffs/BMW.pdf
[6] www.decomsys.com
[7] Paul Yih, Jihan Ryu, J. Christian Gerdes, Modification of Vehicle Handling Characteristics via Steer-by-Wire, Dept. of Mechanical Engineering, Stanford University

Authors

[1] Joachim Langenwalter
European Automotive Marketing Manager
jlangenwalter@mathworks.com

[2] Tom Erkkinen
Embedded Applications Manager
terkkinen@mathworks.com

-- # # # --