2008-01-2663

Applying Model-Based Design to Commercial Vehicle Electronics Systems

Tom Egel, Michael Burke, Michael Carone, Wensi Jin
The MathWorks, Inc.

ABSTRACT

Commercial vehicle manufacturers face unique challenges for the development of vehicle electronics systems. For one, customers typically have unique requirements coupled with an expectation of high reliability. Vehicle electronics is often the enabler for customized features. Ensuring that the vehicle will perform as demanded and promised adds a degree of burden on the vehicle manufacturers. Furthermore, the verification and testing of a large number of unique electronic system configurations is very expensive and time-consuming. This paper will explore how Model-Based Design can be used to meet these challenges and provide a high degree of confidence for both the manufacturer and the customer that requirements have been met. It will discuss factors to consider to support configurability, approaches for defining a system architecture that facilitates reuse, and capabilities for modeling state-based systems.

VEHICLE ELECTRONICS ENGINEERING CHALLENGES

While the powertrain provides the propulsion necessary to perform the required work, vehicle electronics, which include such functions as lighting, operator comfort and assistance, display, power management, and security and safety systems, provide important differentiation in terms of operator comfort, convenience and productivity. As a result, the embedded electronic content in commercial vehicles continues to grow, which leads to the complexity that comes with managing multiple electronic control modules (ECUs). Additionally, many commercial vehicles are developed for a wide range of applications requiring significant customization. It is not atypical for every vehicle that rolls off the assembly line to be unique. Developing, testing and maintaining a large number of custom vehicle configurations pose a significant challenge in terms of both cost and time-to-market. The challenge manifests itself in the need to leverage the benefits of vehicle electronics while effectively managing the increased system complexity and customizations required to service the diverse needs of the customers.

WHAT IS MODEL-BASED DESIGN

The traditional development process for vehicle electronics is characterized by

- Use of paper specification generated from requirements gathered from various sources
- Building physical prototypes for design.
- Writing code manually to implement control algorithms and complex logic.
- Dependence on test vehicles for verification and validation.

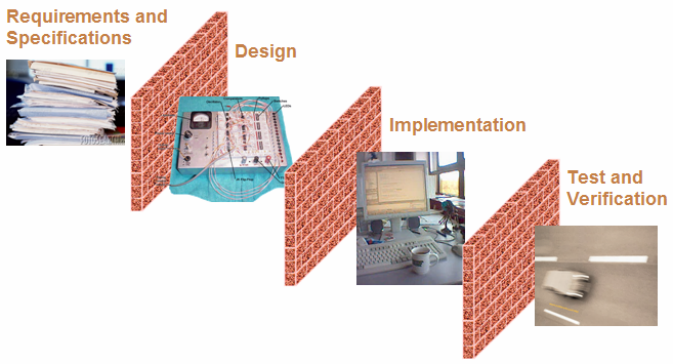

Figure 1: Traditional Vehicle Electronics Development Process, where design, implementation, and test are done by individual team each suffering from barrier of communication to its supplier and customer.

A significant problem with this process is that errors in specification are often not discovered until final validation, which leads to specification change, re-design, re-implementation, and re-validation. Consequently, cost of fixing errors is high [1]. Because of the many vehicle electronics configurations in commercial vehicles, complete test coverage of the design is practically impossible using test vehicles. As a result, errors may be “leaked” into products, leading to field repairs.

Model-Based Design emerged from the need to reduce development cost and improve product quality at the same time, while the complexity of vehicle electronics multiples. In Model-Based Design, a model of the

system being developed serves as the common thread throughout the development process, from requirements capture to final validation. As an executable specification, this system model is refined throughout the development process. Simulation is used at each process step to verify whether the design meets the requirements. Code generation is used for implementing control algorithm and complex logic, thus eliminating errors introduced by writing source code by hand.

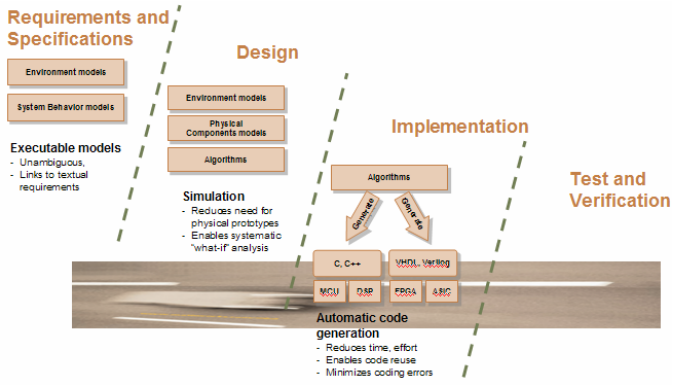

Figure 2: Development Process with Model-Based Design

Since its introduction, Model-Based Design has become widely adopted in the development of electronic control systems, powertrain and vehicle alike for commercial vehicle applications. Examples of applying Model-Based Design to commercial vehicles have been extensively highlighted in past SAE technical papers or presented at conferences in the automotive industry [2] [3] [4].

The remaining sections of this paper will provide a set of guidelines for applying Model-Based Design to vehicle electronics applications, with emphasis on system architecture and modeling state-based systems. Typical scenarios for implementation and verification are also described to provide a complete description of Model-Based Design.

SYSTEM ARCHITECTURE

MOTIVATION

The system model is key to Model-Based Design. Given the requirement for customization in commercial vehicle electronics applications, it is critical to establish the system architecture in the modeling environment such that it provides a flexible framework to facilitate re-configuration and reuse. As a result, the system level models can be quickly customized to support a range of applications.

The design requirements for a system model include

1. A clear representation of control flow between components
2. A clear representation of signal (information) flow between components
3. A clear system hierarchy
4. Uniform / locked down interface between components
5. A simple infrastructure for component integration

Using an architecture with Model Based Design that satisfies these design requirements facilitates the integration of components (requirements 4 and 5 above), the debugging of the system (requirements 1 and 2), the readability and understandability of the system (requirements 1, 2, and 3), as well as the modularity of the system (requirements 2, 4, and 5).

An example of top level architecture is represented in Figure 3. It is composed of hardware input and output components (A/B), the scheduler (C), fault detection (D) and the algorithmic (E) component. Additionally in this case a closed loop test component (F) is included in the system architecture; the test system is not required and is fully decoupled.

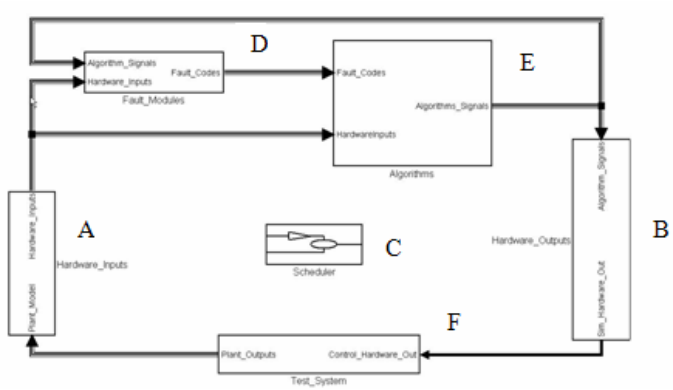

Figure 3: Top level system architecture

Additionally, system designs should take into consideration product line architecture considerations to improve the reusability of the component. In this case a "base" interface is established for all versions of the control algorithm. More advanced versions of the algorithm may use additional interface components, however they should be able to function using the basic interface information.

ARCHITECTURE FRAMEWORK

With Model-Based Design, as in standard architectural design environments, the system designer performs the task of partitioning complex systems into usable components. The partitioning task defines the components data interface (e.g. function prototype, local / global data), the function interface (e.g. the call method, call rate, scope / access to the functions) and the scheduling interface (e.g. execution order and interrupt information)

Figure 3 showed an example of the top level model; in this case information was passed between components in a bus, e.g. a grouped collection of information. Let's look at an example of the hierarchy of in the algorithm component. There are three layers of architecture for the algorithm component, Output data packaging (Figure 4), Signal routing (Figure 5), and the functional component (Figure 6). The goal of these three layers is to

Explicitly show the signal flow between components
Minimize the visual clutter in the diagram (improve readability)
Simplify the addition of components to the system

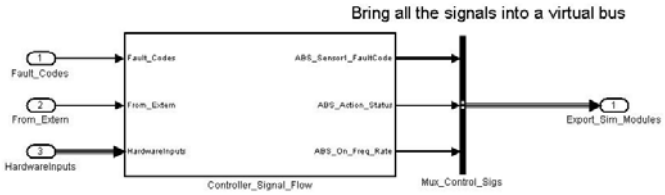

Figure 4: Output data packaging

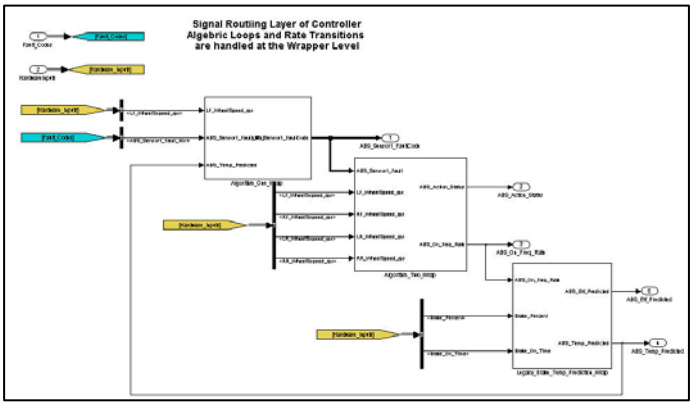

Figure 5: Signal Routing

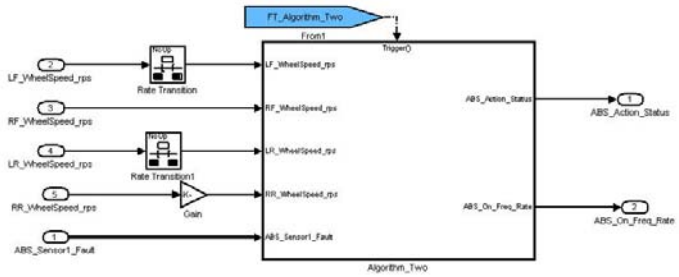

Figure 6: Functional component

Engineers working on functional component start their work with a shell model that has the fully defined I/O, internal data and triggering mechanism. The definition includes signal name, data type, storage class, and ownership; this is known as a "locked down interface". Starting from this point their work can be done independent of the system level designers.

SCHEDULING THE SYSTEM LEVEL MODEL

An architecture in Model Based Design can be created to replicate all standard scheduling methodologies. The most common approaches are the centralized and distributed (Parent / Child) scheduling methodologies. In the centralized approach the total execution context is controlled from a single scheduling component; this enables complete context control of the full system; however for large models this approach can be come difficult to manage. By contrast the distributed Parent / Child scheduling approach has a single parent scheduler that enables multiple child sub-schedulers which enable the final execution context. This approach reduces the flexibility of the scheduler however it is more modular and easier to maintain. From component developers perspective the two approaches are identical.

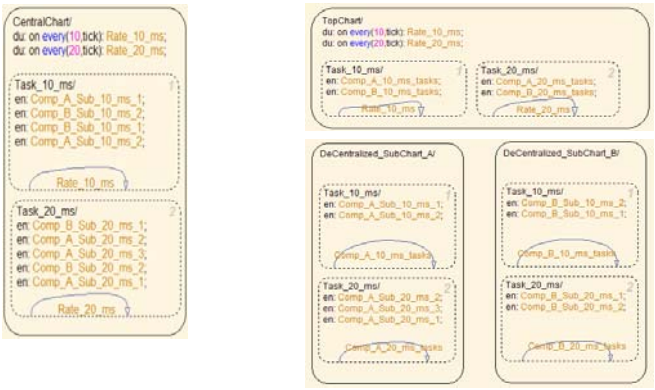

Figure 7: Central and Distributed Scheduling Architectures

APPROACHES TO MODEL INTEGRATION

With Model-Based Design, there are three primary integration scenarios used for a system level model

1. Integration of models into models
2. Integration of C code into models
3. Integration of automatically generated C from models into existing C code.

In all three cases the well defined system architecture simplifies this process. In the first case, integration of models into models looking specifically at Simulink from The MathWorks there are three integration methods supported; libraries, model reference and s-functions (compiled C code). The following table provides a summary of the integration / simulation trade off's for the three methods.

Integrating C code into models or models into C code is a relatively simple task when the architecture has used a locked down interface; it is a N step process. Verification that function interfaces match, collection of

the required support files, writing “wrapper” functions (if required), compiling the complete source code.

	Libraries	Model Reference	S-Function
Storage as a separate file	Y	Y	Y
Supports multiple instance in parent model	Y	Y	N
Inherits model configuration options	Y	N	N
Independent simulation	N	Y	N
Independent code generation	N	Y	NA
Obfuscate model for distribution	N	N	Y
Logging signals during simulation (debugging)	Y	Y	N
Simulates as compiled model	N	Y	Y
Multi-rate / multi-task simulation & code generation	Y	Y	Y
Number of input function called triggers	Multiple	1	0
Can include components of type…	Lib: Y MR: Y S-F: Y	Lib: Y MR: Y S-F: N	Lib: Y MR: N S-F: N

Table 1: Integration Scenarios

MODELING STATE-BASED SYSTEMS

Many vehicle electronics systems involve complex logic and are inherently state-based. For example, the transmission can be in one of several states or gears. Indicator lights can be either on or off. A trash compactor might be active, inactive, or malfunctioning. One way to model the states for all of these subsystems and how they interact with each other is to use a state machine modeling environment. Stateflow, shown in Figure 8, extends Model-Based Design with an environment for developing state machines and flow charts.

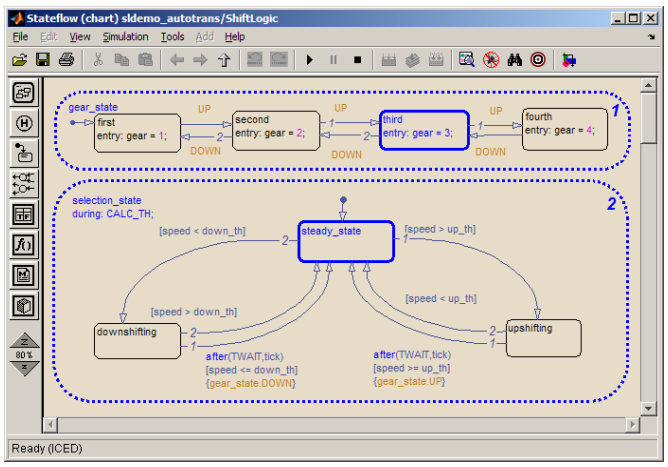

Figure 8: State-Based System Implemented in Stateflow

The tool provides a graphical design environment that allows the user to get a high-level view of the state of various subsystems at any given time via the animation that takes place during simulation. In Figure 8, for example, the active states are gear_state.third and selection_state.steady_state, as shown by the highlighted area.

The most common applications for Stateflow are in the areas of mode logic, scheduling, and fault management. As mentioned in the previous paragraph, subsystems can be in one of several modes, each mode represented by a state. In the system shown in Figure 8, the gear_state can either be in the “first”, “second”, “third”, or “fourth” state.

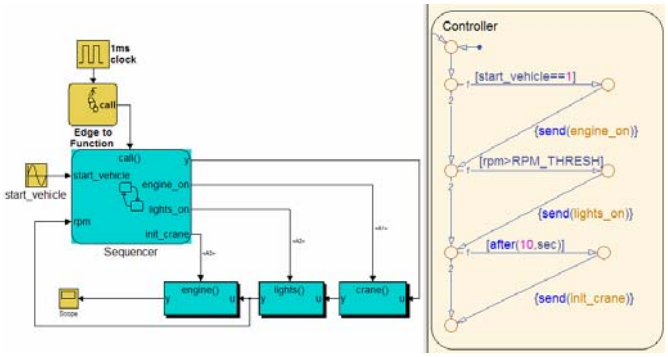

Figure 9: Simulink model (left) and Stateflow chart (right) illustrating scheduling logic

Using Stateflow, designers can schedule precisely when certain tasks are performed and when specific subsystems are enabled or disabled. An example of a scheduling subsystem is depicted in Figure 8. The Stateflow chart “Sequencer” controls when the engine, lights, and crane Simulink subsystems are activated by broadcasting events out to Simulink (e.g., send(engine_on)). Note that the timing can be determined dynamically (e.g., when the engine rpm is greater than a specific threshold, turn the lights on) or statically (e.g., after 10 seconds, initialize the crane).

Fault management is used to maintain the system if failures occur. For example, if the drill for a mining vehicle malfunctions, an indicator light should turn on and the drill should automatically shut down to minimize damage to the equipment.

IMPLEMENTATION AND VERIFICATION

With Model-Based Design, control algorithms and complex logic are captured in model as an executable specification, which is elaborated during design. As a result, the code used to implement the ECU is basically a byproduct when it is generated from the model using production code generation technologies [4] [5]. The automated transformation from the executable specification to code increases productivity both in terms

of reducing errors and the ability to quickly investigate multiple algorithm alternatives. The impact for the commercial vehicle electronics is significant. Because there are relatively few units of each configuration built and many vehicle electronic configurations, design engineers cannot afford to generate a custom software program for each machine. Automatic code generation allows engineers to manage these customizations in the model by using parameter sets to control what part of the software that is active in a specific configuration. Production code generation technologies and their application have been extensive discussed in previous SAE Technical Papers [2] [4].

A main benefit of Model-Based Design is that it allows verification and validation to start early in the development process. Because various engineering teams use models to unambiguously communicate the technical design information, at any time in the process, the design can be verified against the requirements and adjustments can be made to accommodate cost or performance issues [6]. For example, in a crash avoidance system, tests for the response time created during specification development can be re-used during the implementation stage for verification to ensure that the specific implementation selected for the vehicle electronics configuration best meets the requirements. Typical verification and validation techniques in Model-Based Design include

Requirements/model traceability
Modeling standards checking
Model functional testing and automated test execution
Automated test case generation and model structural testing
Test coverage collection
Model/code and requirements/code traceability
Automated report generation

FUTURE TRENDS

AUTOSAR, or Automotive Open System Architecture, is an industry group of over 100 vehicle manufacturers, component suppliers and tool providers working together to develop a standard architecture for automotive embedded software. Although passenger car applications remain the focus for AUTOSAR, commercial vehicle manufacturers have expressed interest in leveraging AUTOSAR to help standardize their vehicle electronics system architecture. Working with vehicle manufacturers and other tool providers, The MathWorks have demonstrated that Model-Based Design can be effectively applied for developing AUTOSAR compliant software [7]. The development approach integrated into Simulink and Real-Time Workshop Embedded Coder [8] allows design engineers to use one model for both AUTOSAR and non-AUTOSAR projects, which maximizes the reuse of existing models. a benefit particularly important to commercial vehicle manufacturers.

CONCLUSION

Model-Based Design provides a range of benefits to commercial vehicle electronics development, including modeling of control algorithms and complex logic, including state-based systems, implementation with automatic code generation, and early verification. A key consideration for such applications is a flexible system architecture to support customization and reuse. This paper shows how system architecture can be effectively established with Model-Based Design.

REFERENCES

1. Dabney, J.B., “Return on Investment of Independent Verification and Validation Study Preliminary Phase 2B Report.” Fairmont, W.V.: NASA IV&V Facility, 2003.
2. Jeffrey M. Thate, Larry E. Kendrick, Siva Nadarajah - “Caterpillar Automatic Code Generation”, SAE Paper 2004-01-0894, SAE World Congress, 2004.
3. Mark Pyclik, “The Role of Real-Time Workshop Embedded Coder in Supporting the Vision of Cummins for Model-Based Development”, MAC 2007.
4. Automatic Code Generation - Technology Adoption Lessons Learned from Commercial Vehicle Case Studies”, SAE Paper 2007-01-4249, SAE Commercial Vehicle Engineering Conference, October, 2007.
5. Tom Erkkinen, “Fixed-Point ECU Development with Model-Based Design”, SAE Paper 2008-01-0744, SAE World Congress, 2008.
6. Brett Murphy, Amory Wakefield, Jon Friedman, “Best Practices for Verification, Validation, and Test in Model-Based Design”, SAE Paper 2008-01-1469, SAE World Congress, 2008.
7. Andreas Köhler, Volkswagen AG, Tillman Reck, Carmeq GmbH, “AUTOSAR-Compliant Functional Modeling with MATLAB, Simulink, Stateflow, and Real-Time Workshop Embedded Coder of a Serial Comfort Body Controller”, MAC 2007.
8. Guido Sandmann, Richard Thompson, “Development of AUTOSAR Software Components within Model-Based Design”, SAE Paper 2008-01-0383, SAE World Congress, 2008.
9. Gavin Walker, Jonathan Friedman, Rob Aberg, “Configuration Management of the Model-Based Design Process”, SAE Paper 2007-01-1775, SAE World Congress, 2007.

CONTACT

Tom.Egel@mathworkscom

Michael.Burke@mathworks.com

Michael.Carone@mathworks.com

Wensi.Jin@mathworks.com