

ASDEN: A Comprehensive Design Framework Vision for Automotive Electronic Control Systems

Deborah Wilson, Daniel Dayton
JRS Research Laboratories Inc., Anaheim, CA 92653

R. Todd Hansell
Motorola Automotive and Industrial Electronics Group/AECS, Detroit, MI

ABSTRACT

The automotive electronics industry is experiencing an era of unprecedented growth. Driven by emissions and safety legislation, fuel economy constraints, cost constraints, and customer demand for convenience features and enhanced performance, electronic controls are steadily replacing their mechanical and hydraulic predecessors. As the sophistication of these systems grows, their complexity has grown dramatically as well, creating difficulties in the application of traditional engineering methods to modern systems. New design paradigms, such as model-based control, have begun to emerge. These factors have created a need for more sophisticated, integrated tool sets to help support the systems engineering process and manage the designs of the new systems. The Automotive Systems Design Environment (ASDEN) project has been undertaken by Motorola to address this need for a sophisticated, capable framework of interoperable tools. This project paves the way for a future where the "Virtual Automobile" becomes a reality: a car designed, simulated, and "driven" before the first physical prototype is even built.

1. The Challenges in Today's Automotive Market

Today, nearly 40 million vehicles per year worldwide are equipped with electronic engine controls. In 1998, the total market for automotive electronics just in North America exceeded \$20 billion, with a projected growth rate of 12% per year for the next decade [1]. Given this growth and Motorola's current leadership position in the sales of automotive electronic components and electronic subsystems, remaining competitive in this industry will be crucial. Some important challenges will include the following:

- Under pressure to reduce costs and vehicle development cycle time, the major automotive manufacturers have come to realize that traditional ad-hoc procedures for electronic control systems are no longer effective for today's complex production efforts.
- Whereas hardware costs over the past few decades have remained relatively stable, software development costs have increased by nearly a factor of 10.
- Unfortunately, many features that might be more appropriately implemented as custom hardware (e.g., ASICs) are implemented as software features, for no automated way exists to assess the tradeoffs between hardware and software solutions.
- Despite all its advantages, the major challenge of today's model-based controls engineering lies in the increased

difficulty and computational requirements stemming from the complexity of the control algorithms and the models used. Adding to this complexity is a trend in the industry towards networked, distributed control.

- Proper design of automotive electronic and mechanical systems depends upon the application of a rigorous engineering process (such as ISO9000, SEI's CMM, etc.). However, the difficulty in deploying and maintaining such a process lies in the management of the project's vast array of work products, quality checkpoints, and interrelated design information. Essential then is to provide the appropriate integrated tools to enable the management of this complexity.
- Process automation can and will continue to deliver numerous benefits to Motorola and alliances working within the automotive industry. The ideal process provides: (1) a framework for the capture of intellectual property; (2) lowering of design costs and design cycle time via early validation of requirements; (3) responsiveness to customer changes for a longer period of time, for functions are bound to implementations later in the design process. This ideal process is now feasible as the result of advances in the technology of design automation tools.

2. The Automotive Systems Design Environment

In order to meet today's challenges and to pursue

comprehensive process automation, Motorola and JRS Research Laboratories, a subsidiary of Motorola, began the Automotive Systems Design Environment project (ASDEN) in January of 1997. Its objectives are to provide an open tool framework to support a design automation methodology for the systems development life cycle, from requirements capture through implementation and target software generation. ASDEN implements the design tool set by:

- Specifying and implementing the most important interfaces between framework tools.
- Working with key vendors in the design automation industry.
- Developing tools where none currently exists.
- Demonstrating instantiations of the tool framework, on actual projects in powertrain control and navigation, in order to show the full concept and tool interoperability.

The rationale and details for these objectives and procedures appear in the sections that follow.

2.1 Hierarchical System Design

Automotive systems engineers must think about today’s complex systems hierarchically to consider how they need to interact in passing data. Requirements are placed on this vehicular network for function, response time, and cost. The system functionality is designed and tested before any architectural constraints are placed on the system. The environment is modeled at this high level of representation with sensor data and sinks added to verify results.

After these first few levels of hierarchy have been fleshed out, architectures of Motorola intellectual property can be pulled into the design framework from reusable parts libraries that have previously been populated off-line by hardware and software design experts. Designers need analysis information to give them insight into the areas where a design may be limited and may not meet requirements, such as via evaluation of hardware and software tradeoffs, refinement, and testing.

2.2 The ASDEN Design Methodology

The above-described hierarchical design flow defines part of the operational processes required for ASDEN. A complete methodology is required to achieve an optimized, cost-effective systems design, independent of any specific tools or functional blocks. The methodology does not dictate the use of specific tools; but it requires that certain *kinds* of tools be used. If unavailable, then the needed tools must be developed.

Figure 1 shows the automotive systems engineer building an automotive ECU. The engineer defines the system functionality as control system models and state models. Performance data enables the engineer to make HW/SW tradeoffs that put design constraints on the architecture and the resulting software code. The automatically generated software drives the calibration tool and the resulting data feeds back into

the user’s framework for further iteration and design refinement. Code generation also leads to additional code optimizations for the designed architecture. If multiple processors exist in the architecture, data is automatically routed through the system to facilitate the functional system behavior. The application software is scheduled properly with links to I/O drivers and custom hardware when needed. Simulations are shown throughout the development process.

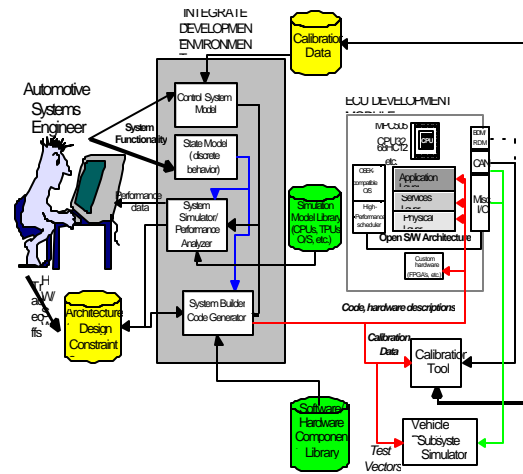


Figure 1 – The ASDEN Framework

The ASDEN project interprets this ideal development environment and converts it to a real design framework, complete with tools underlying the functional boxes in this diagram. Because ASDEN has a key objective to build an *open* framework, it is important that interfaces be specified independent of particular tools.

Figure 2 (next page) shows a functional representation of the ASDEN tool set and how functions and data can flow through the system. Along the left side of the diagram, a variety of databases are shown with data representing application functions, architectures, and characterization data necessary to estimate power, cost, and performance:

- The Reusable Software parts library consists of software primitives and complete application graphs, operating system elements, test vectors, and device drivers for various target machines.
- The Reusable Architectural parts library includes individual components and hierarchies of Motorola architectures as well as competitors’ architectures, perhaps to better enable Motorola designers to measure and understand competing configurations.
- The Characterization library includes data relating the software elements to architectural elements, where a relation is meaningful. Performance, timing, cost, and power formulas for the domain application primitives are captured in this database for each architecture.

Formulas exist for software executing on a programmable processor as well as for functions executing on a hardware accelerator.

The large design database along the right side of the diagram shows the data repository for the design process. It contains versions of system design data sets, tradeoff data sets, and validation data sets, all managed by a layer of enterprise framework overlaying the complete ASDEN tool set.

The topmost level in the diagram shows requirements specification and tracking tools as the first step in the system design process. Budgets for cost, size, functionality, and reliability are saved at this level. Requirements feed into the system application functional modeling phase where the automotive design can be represented in one or more modeling paradigms in order to do functional simulation, debugging, and even hardware-in-the-loop verification of the system functionality. Analysis data about the functional execution is saved in the data repository.

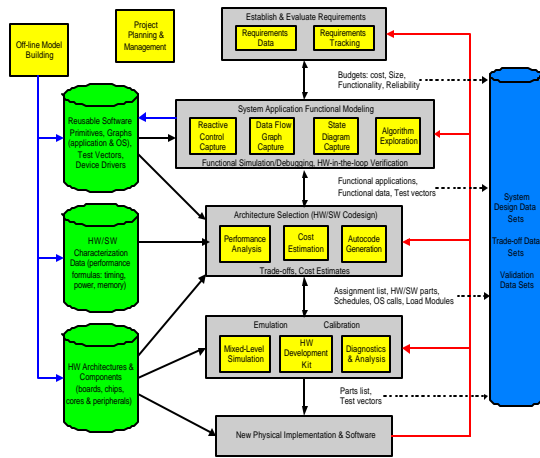


Figure 2 – ASDEN Information Flow

Once the system functionality has been designed and tested, the architectural selection phase takes place. Hardware/software co-design occurs at this level because here one can view the effects of the architecture relative to the application. Assignment is performed here with functional elements mapped onto resources that realize the functionality. Implementation of candidate solutions is derived from instantiation of a parameterized platform, consisting of subsets modeled as pre-validated reusable library elements. This reduces development time, increases the solution’s reliability, and allows reuse of Motorola IP. The candidate solutions are constructed such that performance simulation and analysis can be done to better aid the user in making design trade-offs to develop the best solution. The ASDEN framework also supports promotion of the solution to be a reusable library element.

Each candidate solution must go through a rigorous

verification process to meet all requirements and design constraints. Analysis tools provide the designer with the means for investigating the solution in as much detail as necessary to determine solution acceptance. If requirements have not been met, then design alterations are necessary, or the requirements must be modified. The analysis process also assists in finding design improvements that were not obvious during the design synthesis process.

A key component to high-level systems design is the ability to evaluate alternative solutions rapidly and cohesively. Each iteration of the design process produces a candidate solution with specific design criteria in mind. If a design is optimized for performance it may have made sacrifices for power consumption; thus, the designer made a tradeoff, choosing performance over power. Throughout the entire system design process tradeoffs are constantly made in the form of decisions to choose one technique/method/ implementation over another. At high levels of system design, these tradeoffs often have drastic effects on design realization, and more often than not the effects are unforeseen. ASDEN generates alternative solutions rapidly and performs comparative analysis across the alternative designs. Tradeoff analysis provides the means for narrowing the solution space prior to entering the lower levels of system design refinement.

It is this hardware/software co-design level where cost estimation, performance analysis, and automatic code generation take place. The results of this design stage are:

- Lists of hardware parts.
- Lists of software parts and on which hardware element they are executed.
- Operating system (OS) calls that are needed.
- Device drivers.
- Schedules of software function execution.
- The compiler to use to compile and link the code.

The next design phase shown in the diagram is emulation and calibration. There are thousands of calibration variables that must be specified for a given vehicle model, and this is the design phase where those variables are specified. Mixed-level simulation, hardware development kits, and diagnostic/analysis tools are used to help the calibration engineer perform this important design step.

The last phase of the system design process is for new physical implementation and software. This step crosses abstraction level boundaries by taking the design into a more detailed representation. Requirements and models in the form of executable specifications are now realized. Software lists become load modules for the actual target machines through invocation of compilers and linkers. If a new compiler is required, automatic retargetable compiler generation must be invoked. For new hardware, synthesis processes must be invoked.

The design process is an iteration of analysis, refinement, evaluation, implementation, verification, comparative

evaluation, and prototype generation. Multiple iterations of these processes are performed at each level of design abstraction, creating a recursive technique for systems design as the development progresses from customer requirements capture down through product release and in-field test. Design processes are iterated in all phases of systems design:

- System applications.
- Subsystem applications.
- System functional components such as high-level language procedural code streams.
- New hardware architectures at the network/functional entity level.
- New processors at the RTL level.
- Hardware/ software assignment.
- Entire systems as a whole, encompassing all co-design phases.

2.3 A Usage Scenario for the ASDEN Tools Framework

- System Engineer develops a continuous-time block diagram model of control system and then maps that model to a discrete-time implementation.
- System or Software Engineer adds functionality to the model in terms of finite state behavior diagnostics, communications, user interfaces, etc., via a state modeling notation like statecharts, finite state machines, or systems design language (SDL).
- After verification of model functionality, the model components are mapped onto hypothetical hardware/ software architecture.
- A performance analyzer returns design feasibility and cost information to the systems engineer; design adjustments are made.
- The design is finalized and a hardware-in-the-loop prototype is generated. Test vectors are generated.
- The prototype functionality is confirmed; production-intent hardware and software descriptions are generated automatically for the target system.

2.4 Models

Models are used within ASDEN for simulation, analysis, and code generation. Models span a variety of design levels and technology areas in order to cover ASDEN's large scope. In particular, models represent application functionality, operating system facilities, and architectures. Meta-models represent how architectures are instantiated from parameterized component models.

2.4.1 Functional Models

The system functional requirements are modeled in the form of *executable specifications* in order to represent the system behavior and then demonstrate it on a computer. The

form of the executable specifications promotes expression of the system functionality at the highest possible level of abstraction. Use of different *models of computation* enable the designer to capture different types of system behavior represented from many possible perspectives—control flows, data flows, modes or state transitions, reactive control, discrete and continuous functions, etc. Each view represents a paradigm or model domain that has a computational model and language for expressing behavior.

The computational model's *language* semantics are directed towards a specific way of expressing functionality. As a result, the expressiveness of the designer can be very high-level yet specific for the type of behavior being modeled. These different views and their associated behavioral semantics have been referred to as *models of computation* [2].

Each language must have a *vocabulary* in order to be useful for the system designer to express the behavior of a system. These sets of vocabulary should be predefined and contained in a library (dictionary). From a system design and reuse perspective, the library contents would be a set of pre-validated meanings within the context of each computational model. The system designer also should be allowed to add new library entries either directly or by proxy.

By allowing the system design to be specified using a combination of models of computation and their associated libraries, the system designer can express the meaning/behavior of a system straightforwardly. The resultant system specification is executable, for the semantics of each model of computation define how the system behaves. If the designer uses multiple models of computation there must exist a definition for how the computational models interoperate in order to create a complete functional specification of the system behavior. Then, the specification can be executed and the system behavior can be studied and analyzed. Some of the more common models of computation are the following (many exist):

1. Dataflow: in which computations are performed when their inputs become available and when space is available for outputs.
2. Synchronous dataflow: in which the amounts of data consumed and produced by each operation are fixed, so that the schedule can be computed a priori.
3. Dynamic dataflow: in which the amounts of data consumed or produced by each operation may be variable, thus resisting computation of a fixed schedule.
4. Finite state machine: A computational model consisting of states and transitions between states. Each state and each transition may have an associated computation.

2.4.2 Operating System (OS) Models

OS models must represent all related facilities in order to generate code, simulate the code, and analyze the results: task management, timer management, interrupt handling, and context switch timing, memory management, I/O—control of sensors and actuators, communication—passing data between processes on a single processor and between processors in a network.

2.4.3 Architectural Models

The key ASDEN architectural model is the multiprocessor network architecture that provides representations for the purposes of synthesis, assignment, simulation, analysis, and code generation. The multiprocessor network architecture model is composed of architectural parts, connections, and attributes of parts and networks. Related to architectural parts, this model represents templates, supports instantiations and their modifications, and supports the generation and modification of architectures automatically or manually, etc.

2.4.4 Platform Models

The platform concept involves three important aspects:

- Predetermined silicon design organization into which cores and modules (including memory arrays) may be placed.
- Predefined software organization, including OS, task scheduling, protocols, control, etc.
- Design methodology and tools for tradeoff selection and integration of the design.

The ASDEN view shows the system designer starting with the requirements of the system; and from those high-level requirements the ASDEN system selects a platform, cores and hardware modules, algorithms, and scheduling software. The selection of which portion of the algorithm is implemented in hardware and which portion in software is part of the ASDEN tradeoff analysis process.

ASDEN generates design systems automatically via its methodologies, open framework of tools, and libraries of platforms, hardware, and software. In this environment, very fast cycle times are possible because a new design can be quickly built from existing library elements. Populated libraries are the key to fast design times; when library design elements exist that satisfy the requirements, design of individual components need not be done during the system design process. If the library is not fully populated, more design work needs to be done.

2.5 Software Component Architecture

The objective of ASDEN's "open" platform software architecture is to provide a consistent, application-independent method for controlling the physical hardware interfaces for an electronic control unit. The actual code for the device drivers is dependent upon the hardware configuration. Some primary elements of a hardware device driver library are input drivers, output drivers, and communication drivers. Hardware device drivers transform physical signals to raw data and vice versa.

Above the device driver layer is an API consisting of application function drivers that can be called directly by the application control software without regard to the hardware implementation details of the hardware device driver level. ASDEN contains application support in the following areas:

- Virtual device managers—hardware-independent view, engineering unit interface, and error processing
- Waveform generation—spark, fuel, PWM
- Filters—low pass, high pass, bandpass
- Navigation
- Communications network services

ASDEN includes this open platform software architecture. The reusable software parts library contains the software models described here; they are components of the automatic code generation step defined in the ASDEN methodology. An off-line effort develops these reusable software library parts.

3. Conclusions

An integrated framework of sophisticated development tools is required to manage the design complexity of model-based control systems. This development tool framework must be flexible enough to accommodate differences in customers' design methodologies.

The design tool framework must be interoperable with tool frameworks from other automotive design domains.

The ability to rapidly assess hardware/software tradeoffs has become a key advantage in automotive system design.

Industry must develop systems engineering expertise in various application domains to *anticipate* customer needs for silicon and electronic control modules.

4. References

- [1] Chowanietz, Eric. 1995. *Automobile Electronics*. London: Reed-Elsevier.
- [2] JRS Research Laboratories Inc. Design Methodology for the Automotive Systems Design Environment (ASDEN), 8 August 1997.
- [3] Orfali, Robert, Harkey, Dan, and Edwards, Jeri. 1997. *Instant CORBA*. New York: John Wiley and Sons.