White Paper

Systems Engineering for Smart Products

Today's products are increasingly more intelligent and interconnected — thus the widely adopted "smart" prefix — smart cars, smart buildings, smart appliances. Software-controlled mechatronic systems are used routinely to deliver advanced features, improve safety and reduce power consumption. Consequently, product development processes intended for hardware-dominated systems are no longer adequate. Special attention must be given to the electronics and embedded software that control these systems — in addition to all the interacting hardware components. Managing product complexity, reducing software development costs and optimizing system performance at all design phases are mounting challenges that many companies are addressing. This paper examines three best practices being implemented to overcome these challenges: model-based systems engineering, virtual system prototyping and model-based software development.

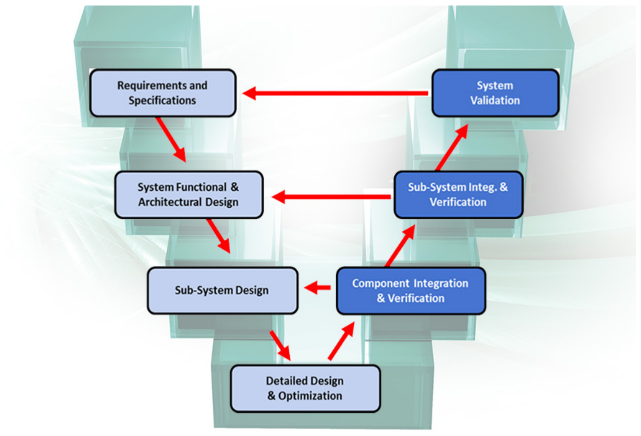
Requirements and Specifications
System Validation
System Functional & Architectural Design
Sub-System Integ. & Verification
Sub-System Design
Component Integration & Verification
Detailed Design & Optimization

Figure 1. The design V is a standard process framework for the systems engineering lifecycle. The left side of the V represents concept development and decomposition of requirements into functions and physical entities that can be architected, designed and developed. The right side of the V represents integration and verification of these entities and ultimate validation of the final system.

Product Complexity Drives New Development Process

In nearly every industry, consumers and businesses benefit from the evolution of smart products. These are highly engineered, multi-functional products that interact with people and their environments in new ways to ensure safety, improve efficiency and/or reduce energy consumption. Take the automotive industry as an example, in which safety systems automatically take corrective action to avoid collisions: Google is sponsoring R&D of a completely autonomous vehicle. In the aerospace community, intelligent avionics are being developed that fly a damaged plane and land it safely. Meanwhile, investments in renewable energy result in smart turbines that continuously change shape and pitch depending on weather, current energy demands, and the performance of nearby turbines.

Under the hood of every smart product is a complex composition of mechanical assemblies, micro-electronics and embedded software, each with hundreds of design parameters and interfaces that need to be engineered, verified and validated. Add in the compounding effects of product variances, and the scope of engineering and system design can become overwhelming.

With the evolution of smarter, more-complex products, many organizations struggle with applying their standard development processes. Traditionally, this process is led by the dominant engineering discipline (often mechanical or electronics), then other disciplines (typically embedded software) are incorporated and synchronized late in the cycle. Today's best-in-class companies have established the necessary processes and adopted tools to shift from a disciplined-based development process to a systems engineering process, like that depicted by the design V (Figure 1).

While the design V serves as a guideline for systems engineering thinking, process details and tool selection are paramount for reducing development costs and time to market. In particular, systems engineering processes, like the design V, traditionally have been document-centric. That is, the system design and associated decisions are captured in a series of static documents. Research shows that deploying systems engineering in the context of dynamic models is much more effective for communicating designs across the enterprise and then maintaining them throughout the product development lifecycle.

Moreover, the system development lifecycle is rarely linear, as the design V implies. There are iterative cycles to verify and improve product design at every phase. Early deployment of virtual system prototyping and optimization can compress these iterative cycles while improving overall product performance.

Finally, the exponential growth in embedded software requires engineering leaders to manage the costs associated with developing and testing embedded systems. More lines of embedded code deployed across multiple product variants also means an increasing number of software bugs and risk of product defects. In response, software teams are moving away from manual coding and instead are deploying model-based software development methods with automated code generators, all designed to accelerate the production and certification of embedded systems.

Managing product complexity, reducing software development costs, and optimizing overall system performance at all design phases are reshaping how companies develop products. Best practices in model-based systems engineering, model-based software development and virtual system prototyping are integral to overcome these challenges and deliver safer, higher-quality smart products to the market faster.

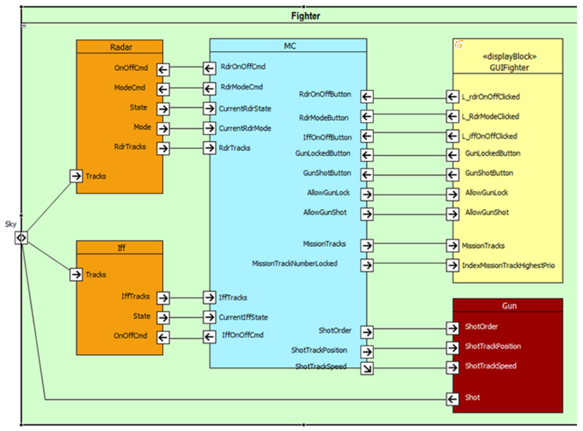

Figure 2. A model in MBSE unambiguously delineates the structure and behavior of the system with block diagrams. System requirements, functions and component architecture are interconnected in an interactive view, which can hide or expose details as necessary.
Image from ANSYS® SCADE System™.

Model-Based Systems Engineering

In traditional systems engineering approaches, documents were the authoritative source of system design information. R&D teams produced and managed a series of documents, such as requirements documents, architecture design documents, detailed design documents and engineering change documents. One shortcoming is that these documents do not capture meaningful relationships between all the elements in the design. If an engineer changed an aspect of one subsystem, the impact of that change on other subsystems had to be discovered and then the documents manually updated. This process can be expensive and burdensome, and the inevitable result is that design contradictions are discovered late in the development cycle — when they are most likely to significantly increase development costs and delay launch schedules.

To better manage the complexities of today's product architectures and truly understand dependencies across subsystems, traditional systems engineering practices have evolved to model-based systems engineering (MBSE). The fundamental difference between the two is that the authoritative system definition no longer resides in a set of static text-based design documents, but rather in a living model, one that provides a thorough understanding of dependencies and interfaces between various subsystems (Figure 2).

In addition to representing large amounts of design information in more sophisticated, interrelated ways, models can be easily shared and communicated across teams, are more amenable to change management, and support automated and comprehensive traceability from stakeholder requirements through implementation. A change to a requirement or architecture specifications would propagate through the system description, enabling impact and trade-off studies.

MBSE inherently requires assembling a wide range of component and subsystem designs, often from globally distributed teams and supply chains. Open standards for model exchange are critical. In recent years, SysML has gained considerable traction as an open standard for MBSE. It is a derivative of UML (Unified Modeling Language), but it was designed particularly for systems engineers and systems engineering best practices.

Model-Based Software Development

Embedded software is on the rise as a main source of product failure. Some industry leaders claim every 1,000 lines of embedded software contain eight bugs. This means that an S-class automobile with 20 million lines of the code could contain 160,000 errors! To manage this quality risk, as well as meet tighter standards for embedded software certification, software engineers need to leverage model-based software simulation tools and qualified automatic code generators.

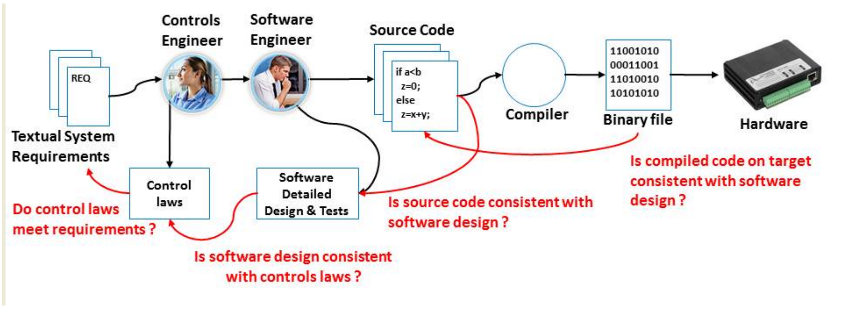
Controls Engineer
Software Engineer
Source Code
REQ
if a<b
z=0;
else
z=x+y;
11001010
00011001
11010010
10101010
Textual System Requirements
Compiler
Binary file
Hardware
Control laws
Software Detailed Design & Tests
Is compiled code on target consistent with software design ?
Do control laws meet requirements ?
Is source code consistent with software design ?
Is software design consistent with controls laws ?

Figure 3. A typical workflow for embedded software system development. Unconnected specifications for controls and software engineers, in conjunction with hand-written source code, inevitably results in software defects.

Figure 3 depicts a typical embedded software development workflow. A controls engineer interprets a set of system requirements and develops the controller logic. The controls engineer then communicates the control logic to an embedded software developer, who then authors a software specification document and writes the software by hand. The software is compiled and flashed onto hardware, which is then tested in a lab. The entire development process is akin to the telephone game children play: As the requirements and design are retold and re-interpreted by the various contributors, there are multiple opportunities for miscommunication and failure.

Compare this to a process based on model-based software development and automatic code generation, shown in Figure 4. Controls engineers and software engineers collaborate on a working model of the control system. The model itself represents the common design specification. From this model, the actual embedded software is automatically generated with qualified code generator.

An important attribute of the model-based workflow is that the control system model must be completely deterministic: The model is represented exactly by the generated code with no ambiguity. Therefore, the same behavior is observed in the simulation and on the target embedded software platform. For safety-critical systems and other applications, the code generator must be qualified in accordance with industry standards, particularly for safety-critical systems. A certified code generator reduces testing requirements and costs associated with certifying embedded systems for ISO 26262, DO 178C and other standards.

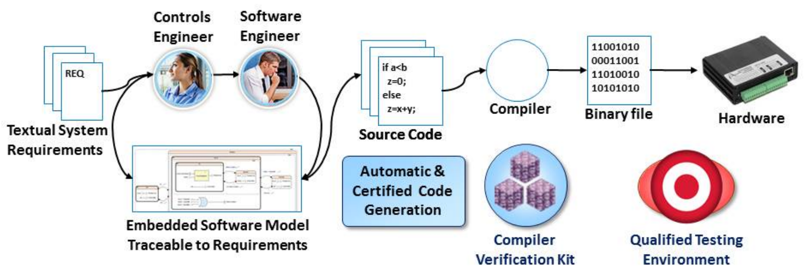
Controls Engineer
Software Engineer
REQ
Textual System Requirements
Embedded Software Model Traceable to Requirements
if a<b
z=0;
else
z=x+y;
Source Code
Automatic & Certified Code Generation
Compiler
Compiler Verification Kit
11001010
00011001
11010010
10101010
Binary file
Hardware
Qualified Testing Environment

Figure 4. Model-based software development and certified code generation. A common model serves as the single specification. Automatic code generation eliminates errors between the model and the final production code.

Virtual System Prototyping and Optimization

With today’s complex product architectures, each component not only has to operate perfectly by itself but in tandem with many other components in the system. In many industries, virtual system prototyping and optimization throughout the system V framework has become a strategic product development initiative to ensure that a product meets power, reliability and safety requirements.

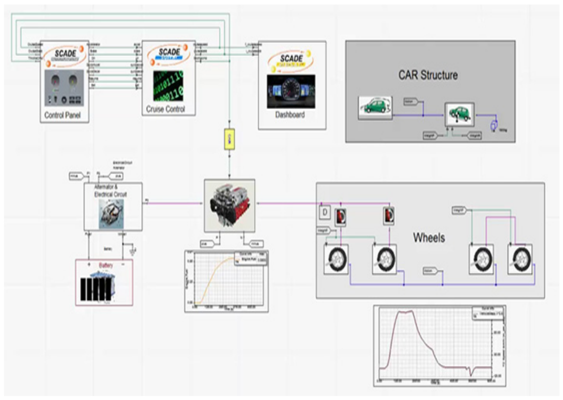

Figure 5. A systems virtual prototype of a hybrid electric vehicle, which combines detailed mechanical and electric drive train components with controls and display software
Image from ANSYS Simplorer®.

Model-based systems engineering provides a framework for virtual system prototyping and optimization because, as soon as the system is described in a model, an engineering team can simulate and systematically optimize it. It is not necessary for engineering to wait until system topology or detailed CAD is available to perform simulation and apply optimization methods. It can start as soon as the functions and architecture of the system are understood.

A systems engineering simulation platform that supports the full array of physics and various levels of fidelity is a critical component for virtual prototyping and optimization. For example, at the requirement and functional decomposition phase, systems engineers simulate how a hierarchy of system functions work together to transform inputs into outputs, react to events or respond to excitations. Downstream product architectures use behavior (0-D) modeling to verify the logic and allocation of functions to physical components, taking into account physical conservation laws. Finally, at the bottom of the system V, detailed 3-D physics simulation and control system modeling are used to design and optimize individual components.

Throughout this process, the ability to readily transcend different levels of fidelity is important. Results from 3-D modeling and simulation should be added back into the behavioral model to virtually verify the detailed designs meet system requirements. To make this computationally cost-effective, reduced-order modeling (ROM) and 0-D to 3-D cosimulation are brought to bear. ROM refers to a set of methods that reduce computationally intensive 3-D models into smaller forms that approximate the full 3-D model for a given set of operational conditions. 0-D to 3-D cosimulation is the coordination of a partitioned system model between a 0-D system solver and full 3-D physics solvers.

A key insight from organizations that deploy MBSE and system virtual prototyping is that the process can produce a proliferation of disparate models. This is due in part to unmanaged workflows as well as fragmentation in the tool chain and the use of proprietary modeling languages and file formats. What is needed is a systems engineering platform based on open standards that allows engineering teams to build upon the system model throughout the development cycle and virtually verify the system at each phase. Open standards for describing physical systems — such as VHDL-AMS and Modelica — as well as an emerging standard for cosimulation called the functional mockup interface (FMI) are key enablers for such platforms. Simulation tools that fully support FMI can export a dynamic model of a subsystem that can then be imported into another tool and connected to other subsystem models for dynamic simulation. It is easy to envision how these open standards enable collaboration and model exchange between an OEM responsible for the complete system and the supply chain delivering individual subsystems.

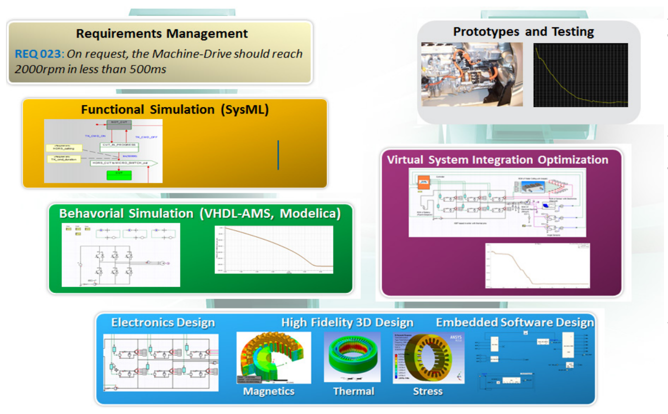
Requirements Management
REQ 023: On request, the Machine-Drive should reach 2000rpm in less than 500ms
Prototypes and Testing
Functional Simulation (SysML)
Virtual System Integration Optimization
Behavorial Simulation (VHDL-AMS, Modelica)
Electronics Design
High Fidelity 3D Design
Embedded Software Design
Magnetics
Thermal
Stress

Figure 6. Model-based development process throughout system V

Summary

Today's products are increasingly more intelligent and comprise a complex composition of hardware, electronics and software. Designing and engineering smart products requires systems engineering methods based firmly in model-based methods and complete system prototyping. Moreover, with the growth and increasing costs associated with embedded systems, a model-based software development process based on certified and automatic code generation can substantially reduce development time and risks associated with software defects.

When your organization is ready to implement a model-based development process, make sure the systems engineering simulation tools you choose can meet these critical requirements along the systems V (Figure 6):

- Interactively design a system architecture using model-based system engineering best practices and the open system modeling language SysML.
- Perform conceptual and 0-D behavioral modeling to optimize and validate the system architecture. Leverage standard languages like VHDL-AMS and Modelica to enable model exchange with the supply chain.
- Design, optimize and verify individual hardware and electronics subsystems using best-in-class 3-D multiphysics simulation.
- Design and automatically generate embedded control systems (or human–machine interfaces) using model-based software development methods that are deterministic and meet industry and safety certification standards.
- Perform virtual systems optimization by combining software, electronics and hardware designs into a complete system virtual prototype with reduced-order modeling and/or cosimulation.

ANSYS, Inc.
Southpointe
275 Technology Drive
Canonsburg, PA 15317
U.S.A.
724.746.3304
ansysinfo@ansys.com