WHITE PAPER

Model-Based Design for Space Control Systems

MathWorks®

Imagine that your team is developing the power system of a satellite. The system incorporates a combination of physical elements (e.g., battery, solar panels), control logic, and external conditions (e.g., temperature, radiation). Before you begin the design, you want to address some key questions—for example:

- How do we size the batteries?
- What if the requirements change?
- How can we optimize the design to ensure the desired performance?
- How can we test the design thoroughly while minimizing risk?

Whether you're developing controls for a flight system, an industrial robot, a wind turbine, a production machine, an autonomous vehicle, an excavator, or an electric servo drive, if your team is manually writing code and using document-based requirements capture, the only way to answer these questions will be through trial and error or testing on a physical prototype. And if a single requirement changes, the entire system will have to be recoded and rebuilt, delaying the project by days, or even weeks.

Using Model-Based Design with MATLAB® and Simulink®, instead of handwritten code and documents, you create a system model—a model incorporating the physical model, the control algorithms, and the environment. You can simulate the model at any point to get an instant view of system behavior and to test out multiple what-if scenarios and tradeoff analyses without risk, without delay, and without reliance on costly hardware.

This white paper introduces Model-Based Design and provides tips and best practices for getting started. Using real-world examples, it shows how teams across industries have adopted Model-Based Design to reduce development time, minimize component integration issues, and deliver higher-quality products.

What Is Model-Based Design?

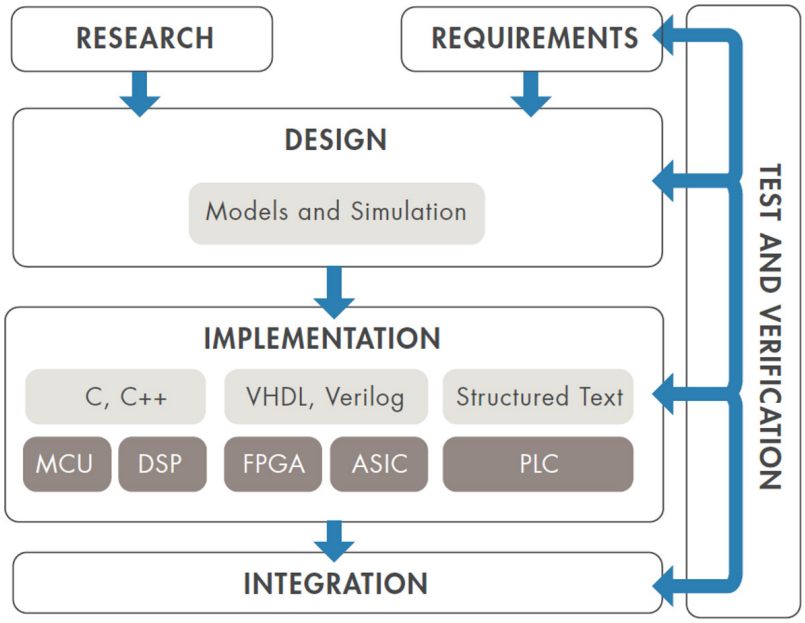
The best way to understand Model-Based Design is to see it in action:

> A team of aerospace engineers sets out to design the guidance, navigation, and control (GNC) system for a satellite. Because they are using Model-Based Design, they begin by building an architecture model from the system requirements; in this case, it's the satellite model itself. A simulation/design model is then derived. This high-level, low-fidelity model includes portions of the controls software that will be running in the satellite, plus the plant and the operating environment.
>
> The team performs initial system and integration tests by simulating this high-level model under various scenarios to verify that the system is represented correctly and that it properly responds to input signals.
>
> They add detail to the model, continuously testing and verifying the system-level behavior against specifications. If the system is large and complex, the engineers can develop and test individual components independently but still test them frequently in a full system simulation.

Ultimately, they build a detailed model of the system and the environment in which it operates. This model captures the accumulated knowledge about the system (the IP). The engineers generate code automatically from the model of the control algorithms for software testing and verification. Following hardware-in-the-loop tests, they download the generated code onto production hardware for testing in an actual final system.

As this scenario shows, Model-Based Design uses the same elements as traditional development workflows, but with two key differences:

- Many of the time-consuming or error-prone steps in the workflow—for example, code generation—are automated.
- A system model is at the heart of development, from requirements capture through design, implementation, and testing.



Workflow for Model-Based Design.

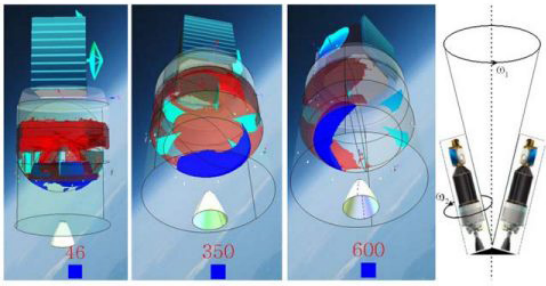
Systems Engineering, Requirements Capture and Management

In a traditional workflow, where requirements are captured in documents, handoff can lead to errors and delay. Often, the engineers creating the design documents or requirements are different from those who design the system. Requirements may be "thrown over a wall," meaning there's no clear or consistent communication between the two teams.

In Model-Based Design, you author, analyze, and manage requirements within your Simulink model. You can create rich text requirements with custom attributes and link them to designs, code, and tests. Requirements can also be imported and synchronized from external sources such as requirements management tools. When a requirement linked to the design changes, you receive automatic notification. As a result, you can identify the part of the design or test directly affected by the change and take appropriate action to address it. You can define, analyze, and specify architectures and compositions for systems and software components.

In addition, systems engineers can use MATLAB and Simulink to perform dynamic analysis. They use executable models of multidomain spacecraft and ground systems for requirements validation and verification, providing insights into system-level behavior and performance that cannot be obtained by static analysis alone. This approach also enables them to trace requirements from high-level specifications, monitor the detailed implementation of the requirements in the design, and track the requirements in the automatically generated source code. They can map the requirements to test cases and automatically measure requirements coverage as the test cases are executed.

Case Study: *ESA & Airbus Defense and Space*



Propellant motion in spinning upper stages at 46, 350, and 600 seconds. Distribution after 350 seconds becomes uneven.

"Model-Based Design enabled us to create a framework for designing flight controllers with state-of-the art robust control design algorithms, creating multidomain physical models, tuning the design through optimization, and generating code for PIL testing on target hardware—all in the same environment."

— Hans Strauch, Airbus D&S

When a European Space Agency (ESA) launcher, such as the Ariane 5 or Vega, delivers its satellite payload into orbit, the attitude control system (ACS) takes control, orients the payload, and commands the separation from the upper stage of the launcher. In addition to orienting the satellite, the ACS must identify and manage problems associated with the separation process, the sloshing of propellant, and a wide range of potential hardware faults.

ESA and Airbus wanted to simulate separation failures with a physical model in order to test the controller's ability to detect failures and undertake corrective action. They also needed to simulate sloshing propellant, leaks in pipelines, stuck valves, and a range of other faults. In addition, they wanted to run optimizations to identify the worst-case performance of the system should faults occur.

Engineers sought to test their control algorithms on flight computer hardware as early as possible in development. As control algorithms grow in complexity, they push the limits of processor performance and other computing resources. The engineers needed to verify algorithm performance and resource utilization on a representative flight computer while the controller was being designed, when it would be easiest to correct problems.

ESA and Airbus engineers used Model-Based Design with MATLAB, Simulink, and a combination of code generation, physical modelling, control logic design, and verification and validation tools to create the Upper Stage Attitude Control and Design Framework (USACDF), which enables closed-loop simulation and verification of control algorithms with physical models and is used to build demonstrators for complex orbital servicing mission operations concepts.

Design

In a traditional approach, every design idea must be coded and tested on a physical prototype. As a result, only a limited number of design ideas and scenarios can be explored because each test iteration adds to the project development time and cost. In the space domain, design choices are especially critical to ensure the physical system used in the mission is right.

In Model-Based Design, the number of ideas that can be explored is virtually limitless. Requirements, system components, IP, and test scenarios are all captured in your model, and because the model can be simulated, you can investigate design problems and questions long before building expensive hardware. You can quickly evaluate multiple design ideas, explore tradeoffs, and see how each design change affects the system.

Case Study: *Tessella*

Artist's rendition of the Solar Orbiter.

"We saw the benefits of Model-Based Design on several previous projects. On this project, MATLAB and Simulink enabled us to create a detailed specification that minimized deviation between the prototype algorithms we developed, tuned, and tested in Simulink and the final software implementation."

— Andrew Pollard, Tessella

The Solar Orbiter mission of the European Space Agency's Cosmic Vision program is set to answer fundamental questions about the workings of the solar system and the origins of the universe. The attitude and orbit control subsystem (AOCS) will be responsible for keeping the spacecraft and its solar shield oriented toward the sun and for maintaining a precise attitude to maximize the accuracy of the instruments.

The AOCS must continuously adjust the Solar Orbiter spacecraft's attitude so that the solar shield provides maximum protection as the spacecraft passes close to the sun. For safety reasons, the AOCS cannot allow the spacecraft to depoint more than 6.5 degrees from the Sun at any time, even after a failure. During scientific observations, pointing stability must be within a few tenths of an arcsecond.

In addition to meeting these requirements, the AOCS had to account for disturbance torques from solar radiation pressure, gravity gradient, and aerodynamic forces.

The spacecraft's physical structure compounded the AOCS design challenge. The solar shield contributed to an unusual mass distribution that made stability a challenge. In addition, multiple flexible appendages—including solar arrays—made the entire structure susceptible to resonance.

Tessella engineers used Model-Based Design to design, model, simulate, and perform preliminary tuning of the algorithms, and prove their suitability for formal coding and verification. Working in Simulink, the team modeled the spacecraft's actuation systems, including its four reaction wheels and chemical propulsion thrusters. To provide fine-grained control of the thrusters, the team developed and modeled an actuator commanding algorithm using pulse-width modulation.

Power System

Power systems engineers use MATLAB and Simulink to run simulations for mission power profile analysis, predict the system impacts of battery aging, and perform detailed design of electrical components such as DC-DC converters.

They can rapidly model electrical components and systems, such as solar arrays and voltage regulators, using provided blocks, or they can create custom blocks where the design calls for it. Engineers can then simulate the model to solve the underlying complex systems of equations without writing low-level code, and immediately visualize the results. They can also include thermal and attitude effects in their models to perform multidomain simulation within one environment.

Case Study: *Lockheed Martin*

NASA's Orion spacecraft.

"With Simscape Electrical we created an integrated power system model that connects electrical and thermal domains, so we get the whole picture during our mission-level simulations. If we need to model the motors that turn the solar arrays, we have the capability to integrate those mechanical components, too."

— Hector Hernandez, Lockheed Martin

Lockheed Martin engineers developed a power system model for NASA's Constellation program using Microsoft® Excel® and Visual Basic® for Applications (VBA). As engineers tried to examine more and more test cases, however, the model slowed and occasionally crashed. Because it was a single-node power model, the team could not use it to evaluate voltage drops throughout the system or to assess power quality requirements.

Lockheed Martin engineers used Simulink and Simscape Electrical to model and simulate the power system for the Orion spacecraft. To calculate an accurate current-voltage (IV) curve for the solar array, the model takes into account the number of cells per wing, pointing angles of the solar array, shadowing, and thermal effects such as solar, planetary infrared and albedo. Performance degradation factors such as radiation, contamination, and collisions with micrometeoroids and orbital debris were also considered.

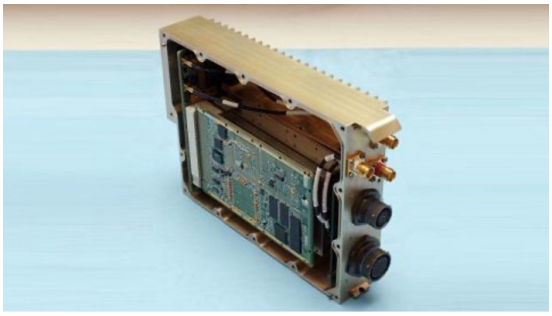
The model of the lithium-ion battery accounts for battery hysteresis effects, temperature, and battery state of charge. The team also created a custom power load block in order to simulate the varying loads generated by the spacecraft's propulsion, guidance, and other systems as they switch on and off throughout a mission. Using the integrated power system model, the team ran simulations for a variety of mission profiles. They monitored mission and load changes to assess power impacts during development, simulated fault scenarios, including battery failures, stuck switches, and solar array failures in the ascent and orbit phases. The results were used to guide the development of mission opportunities and protocols for failure cases.

Communications System

Communications systems engineers use MATLAB and Simulink as a common design environment to develop, analyze, and implement spacecraft communications systems. Engineers can use MATLAB and Simulink to prototype signal chain elements—including RF, antenna, and digital elements. They can then combine the work of multiple teams as a system-level executable model.

Engineers can quickly explore imperfections at the system level and examine what-if scenarios difficult to produce in the lab. As the design matures, engineers can automatically generate C code for embedded processors or HDL code for FPGAs.

Case Study: *BAE Systems*



Custom board used in the traditional design workflow.

"It took 645 hours for an engineer with years of VHDL coding experience to hand code a fully functional SDR waveform using our traditional design flow. A second engineer with limited experience completed the same project using Simulink and Xilinx System Generator in fewer than 46 hours."

— Dr. David Haessig, BAE Systems

Long at the forefront of SDR technology, BAE Systems has traditionally used a design flow that relied on hand-coding FPGAs in VHDL®. Recently, however, BAE Systems saw an opportunity to evaluate this approach against Model-Based Design using MathWorks and Xilinx® tools. Running two SDR waveform development efforts in parallel, they found that Simulink® and Xilinx System Generator dramatically reduced development time.

BAE Systems was tasked with developing a military standard (MIL-STD-188-165A) satellite communications waveform for implementation in a command, control, communications, computers, intelligence, surveillance, and reconnaissance (C4ISR) radio. At the same time, BAE Systems sought to evaluate a new design flow for reducing development time.

Working with Xilinx, BAE Systems applied Model-Based Design using Simulink and Xilinx System Generator to design and deploy an MIL-STD-188 SDR waveform 10 times faster than with their hand-coding approach. Based on the success of this project's initial effort, BAE Systems has begun a joint effort with MathWorks, Virginia Tech, Xilinx, and Zeligsoft to improve waveform portability. This group is developing an interface that enables code generated by Simulink Coder™ or Xilinx System Generator to be directly incorporated into Software Communications Architecture (SCA) radios.

Guidance, Navigation, and Control

Using MATLAB and Simulink, control engineers can test their control algorithms with plant models before implementation, so they can achieve complex designs without using expensive prototypes. They can design for multiple physical configurations, such as the common bus architecture of a satellite design. In a single environment, engineers work on:

- Building and sharing GNC models
- Integrating and simulating system-level effects of controls and mechanical design changes
- Reusing automatically generated flight code and test cases
- Integrating new designs with legacy designs and tools

Case Study: *Lockheed Martin Space Systems*

The IRIS observatory.

"A team of about four engineers designed, integrated, and tested the GN&C system in just 23 months. We were more efficient because we used the same tools for both analysis and code development, and generated 20,000 lines of defect-free code. For us, that makes a compelling case for Model-Based Design."

— *Vincentz Knagenhjelm, GN&C engineer, Lockheed Martin Space Systems*

The Interface Region Imaging Spectrograph (IRIS) observatory is currently in Earth orbit, where it is capturing ultraviolet spectra and high-resolution images of the sun. These images will help scientists better understand the flow of energy and plasma in the lowest levels of the solar atmosphere.

On similar projects in the past, Lockheed Martin engineers produced extensive algorithm design documents, some more than 1000 pages long. Programmers wrote the code by hand based on their interpretation of these documents. The entire process was slow, and defects were sometimes introduced during the hand coding. With just 23 months scheduled for software design, integration, and testing, the team needed to accelerate the software delivery process significantly.

Lockheed Martin engineers accelerated the development of the IRIS GNC flight software by using Model-Based Design. Working in MATLAB and Simulink, the engineers developed a basic model of the control system to analyze pointing performance, or how accurately the spacecraft could be reoriented.

They verified the initial GN&C design by running closed-loop simulations with the plant model and performing model coverage analysis on the simulations using Simulink Coverage™. They used Embedded Coder to generate C code for each component, adding a small amount of hand-generated "glue" code for a Moog Broad Reach Engineering radiation-hardened microprocessor and its executive software. Using a custom MATLAB user interface, the team exercised a variety of Simulink test cases for each GN&C flight software unit.

Case Study: *The Apollo 11 Moon Landing: Spacecraft Design Then and Now*

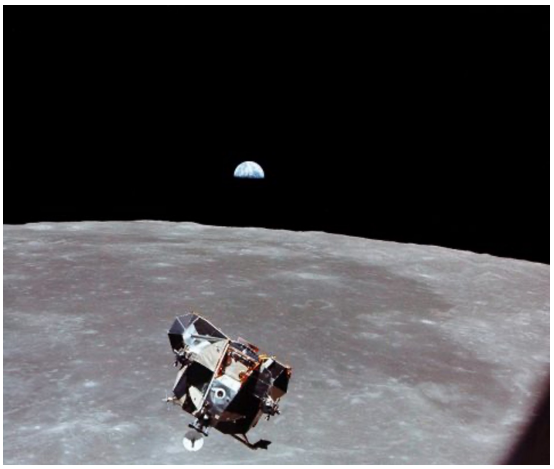


Image courtesy NASA.

Apollo 11, carrying astronauts Neil Armstrong and Buzz Aldrin, landed on the moon over 50 years ago. To commemorate that momentous event, and to celebrate current programs working on the next moon landings, we revisit Richard J. Gran's firsthand account of designing the Lunar Module digital autopilot, published in 1999. In that article, Richard described the approach he used in the 1960s and compared it with the way Model-Based Design with MATLAB and Simulink could be used to design GN&C systems in 1999. In the 20 years since Richard wrote his article, Model-Based Design has evolved significantly. To highlight this evolution, we describe how MATLAB and Simulink can be applied to GNC system design today.

Read the article to learn more, and see the Simulink system model Richard Gran developed when revisiting the LM digital autopilot.

Data and Image Processing and Computer Vision

Vision-based sensing systems are enabling increased levels of autonomy and precision in navigation for space missions. With high-precision relative and optical navigation, vision-based sensing systems are playing a critical role in:

- Rendezvous and proximity operations (RPO)
- Entry, descent, and landing (EDL)
- Robotic exploration of the solar system

Today's increasingly ambitious mission requirements, along with an energized and innovative private sector, are motivating a surge of research in vision-based sensing and perception techniques that use artificial intelligence with machine learning.

Read the white paper to learn more about:

- Vision-based sensing as an enabler of spacecraft autonomy
- How machine learning trends in the space segment are poised to affect spacecraft artificial intelligence (AI)
- How you can use MATLAB and Simulink routines to focus on the higher-level design

Case Study: *Cornell University*

An acoustic analysis device used by the Bioacoustics Research Program to collect data from large baleen whales and other marine mammals. Photo courtesy Dimitri Ponirakis.

"High-performance computing with MATLAB enables us to process previously unanalyzed big data. We translate what we learn into an understanding of how human activities affect the health of ecosystems to inform responsible decisions about what humans do in the ocean and on land."

— Dr. Christopher Clark, Cornell University

For more than 30 years, scientists have studied local animal populations by recording animal sounds in oceans, jungles, forests, and other natural environments. They use the results to assess the effect of man-made noise on natural environments, monitor endangered animal populations, and investigate animal communication. Passive acoustic monitoring systems record sounds continuously, generating terabytes of data. Scientists are often unable to process even 1% of this data because they lack the necessary advanced algorithms and processing capacity.

The variability of animal sounds across individuals within a species is a further complication. Noisy data and variability increase the number of false positives and negatives, reducing the detection algorithms' accuracy. Processing the hundreds of terabytes of data that BRP is gathering presents another challenge. A typical project involves processing years of raw acoustic data—up to 10TB—recorded on multiple channels. Each channel may capture hundreds of millions of events—sounds that stand out when the data is viewed as a spectrogram. Algorithms tested on small, high-quality samples are often considerably less accurate when applied to larger, noisier data sets.

BRP data scientists used MATLAB to develop high-performance computing (HPC) software for automatically processing acoustic data. They begin a detection-classification project by collecting audio clips of the animal they wish to detect, clips of background noise in the animal's environment, and MAT-files of archived acoustic data. Working in MATLAB, they develop new or refine existing algorithms that detect audio sequences in the archived data similar to those in the clip catalog. The BRP team developed a MATLAB interface that enables researchers to specify the algorithms, data sets, and number of processors. In addition to detection and classification algorithms, BRP uses MATLAB for noise analysis and acoustic modeling, in which the time and frequency dispersion effects of marine or terrestrial environments are captured and simulated.

Code Generation

In a traditional workflow, embedded code must be handwritten from system models or from scratch. Software engineers write control algorithms based on specifications written by control systems engineers. Each step in this process—writing the specification, manually coding the algorithms, and debugging the handwritten code—can be both time-consuming and error-prone.

With Model-Based Design, instead of writing thousands of lines of code by hand, you generate code directly from your model, and the model acts as a bridge between the software engineers and the control systems engineers. The generated code can be used for rapid prototyping or production.

Rapid prototyping provides a fast and inexpensive way to test algorithms on hardware in real time and perform design iterations in minutes rather than weeks. You can use prototype hardware or your production ECU. With the same rapid prototyping hardware and design models, you can conduct hardware-in-the-loop testing and other test and verification activities to validate hardware and software designs before production.

Production code generation converts your model into the actual code that will be implemented on the production embedded system. The generated code can be optimized for specific processor architectures and integrated with handwritten legacy code.

Case Study: *Swedish Space Corporation*

Artist rendition of SMART-1 traveling to the Moon.

"We successfully developed the SMART-1 AOCS in a very short time frame and with a very low budget. MathWorks tools for simulation and flight-code generation played a key role in this success and will serve as the foundation for future satellite programs, such as Prisma."

— Per Bodin, Swedish Space Corporation

Swedish Space Corporation (SSC) developed the attitude and orbit control system (AOCS) of the Small Missions for Advanced Research and Technology (SMART-1) using automatically generated flight code. The AOCS orients the spacecraft for thrust vectoring, scientific instrument pointing, and ensuring that the solar arrays are illuminated by the sun. It also controls the electric propulsion thrust vector alignment within the spacecraft body during lunar transfer and descent phases while providing an advanced failure detection, isolation, and recovery (FDIR) system.

SSC needed to develop an AOCS within a low-cost mission profile, strict software development standards, and a short software development cycle of fewer than two years. Moreover, because the AOCS needed to perform in a harsh space environment of intense radiation, minimal gravity, and other effects not testable in a lab or on earth, SSC required rigorous proof-chain test capabilities to ensure the system performed correctly during flight.

SSC implemented a new development process based on MathWorks tools for Model-Based Design to model, simulate, automatically generate code, and to test the onboard AOCS software. Engineers developed accurate simulation models to predict system behavior and to create exhaustive system and software test cases, which met the ESA PSS-05 software development standard. SSC also performed software system testing on a hard real-time simulation environment and analyzed the results with MATLAB. They verified the AOCS at the system level using the integrated spacecraft. These tests included open- and closed-loop tests at the European Space Research and Technology Centre (ESTEC) in the Netherlands.

Test and Verification

In a traditional development workflow, test and verification typically occur late in the process, making it difficult to identify and correct errors introduced during the design and coding phases.

In Model-Based Design, test and verification occur throughout the development cycle, starting with modeling requirements and specifications and continuing through design, code generation, and integration. You can author requirements in your model and trace them to the design, tests, and code. Formal methods help prove that your design meets requirements. You can produce reports and artifacts and certify your software to functional safety standardssuch as ***NPR 7150.2*** (NASA Software Engineering Requirements) and ***ECSS-E-40*** (European Cooperation for Space Standardization, Space Engineering Software).

Case Study: *OHB*

Simulation of GNC strategies for advanced autonomous formation flying.

> "Those models evolved into full flight models, which we verified in closed-loop simulations with a Simulink plant model. From there, generated flight code was just a click away."
>
> *— Ron Noteborn, OHB*

Planned space missions often depend on autonomous formation flying, in which one spacecraft approaches or flies alongside another. The Prisma project, led by OHB AG (OHB) in collaboration with the French and German space agencies and the Technical University of Denmark, tests and validates GNC strategies for advanced autonomous formation flying.

OHB engineers used Model-Based Design to develop GNC algorithms, run system-level real-time closed-loop simulations, and generate flight code for Prisma's two satellites, Mango and Tango.

OHB partitioned the GNC design into formation flying, rendezvous, and proximity operations. They tested and analyzed algorithm ideas in MATLAB before modeling them to verify the algorithms in closed-loop simulation.

Engineers generated code from their GNC models and plant model. They deployed the plant code to Simulink Real-Time™ and compiled the GNC code for the onboard target LEON2 processor. OHB then ran hardware-in-the-loop (HIL) tests of the combined Simulink Real-Time system and LEON2 controller to verify the real-time operation of the algorithms.

Getting Started

While your team might see the benefits of moving to Model-Based Design, they might also be concerned about the risks and challenges—organizational, logistical, and technical—that could be involved. This section addresses questions frequently asked by engineering teams considering adopting Model-Based Design and provides tips and best practices that have helped many of these teams manage the transition.

Q. How are engineering roles affected by the introduction of Model-Based Design?

A. Model-Based Design does not replace engineering expertise in control design and software architecture. With Model-Based Design, control engineers' roles expand from providing paper requirements to providing executable requirements in the form of models and code. Software engineers spend less time coding application software and more time on modeling architecture; coding OS, device driver, and other platform software; and performing system integration. Both control and software engineers influence the system-level design from the earliest stages of the development process.

Q. What happens to our existing code?

A. It can become part of the design; your system model can contain both intrinsically modeled and legacy components. This means that you can phase in legacy components while continuing to perform system simulation, verification, and code generation.

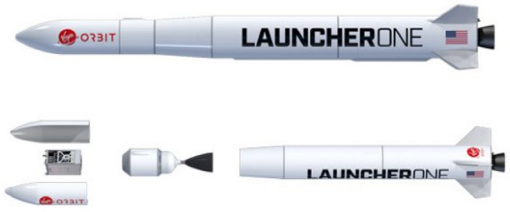
Q. Is there a recommended way to adopt Model-Based Design?

A. Trying new approaches and design tools always carries an element of risk. Successful teams have mitigated this risk by introducing Model-Based Design gradually, taking focused steps that help a project along without slowing it down. Organizations of all sizes begin their initial adoption of Model-Based Design at the small group level. They usually start with a single project that will provide a quick win and build on that early success. After gaining experience, they roll out Model-Based Design at the department level so that models become central to all the group's embedded systems development.

These four best practices have worked well for many teams:

- **Experiment with a small piece of the project.** A good way to start is to select a new area of the embedded system, build a model of the software behavior, and generate code from the model. A team member can make this small change with a minimal investment in learning new tools and techniques. You can use the results to demonstrate some key benefits of Model-Based Design:
  - High-quality code can be generated automatically.
  - The code matches the behavior of the model.
  - By simulating a model, you can work out the bugs in the algorithms much more simply. and with greater insights than by testing C code on the desktop.
- **Build on your initial modeling success by adding system-level simulation.** As previous sections of this paper have shown, you can use system simulation to validate requirements, investigate design questions, and conduct early test and verification. The system model does not need to be high-fidelity; it just needs to have enough detail to ensure that interfacing signals have the right units and are connected to the right channels, and that the dynamic behavior of the system is captured. The simulation results give you an early view of how the plant and controller will behave.
- **Use models to solve specific design problems.** Your team can gain targeted benefits even without developing full-scale models of the plant, environment, and algorithm. For example, suppose your team needs to select parameters for a solenoid used for actuation. They can develop a simple model that draws a conceptual "control volume" around the solenoid, including what drives it and what it acts upon. The team can test various extreme operating conditions and derive the basic parameters without having to derive the equations. This model can then be stored for use on a different design problem or in a future project.
- **Begin with the core elements of Model-Based Design.** The immediate benefits of Model-Based Design include the ability to create component and system models, use simulations to test and validate designs, and generate C code automatically for prototyping and testing. Later, you can consider advanced tools and practices and introduce modeling guidelines, automated compliance checking, requirements traceability, and software build automation.

Case Study: *Virgin Orbit*



Virgin Orbit's LauncherOne vehicle assembled (top), with exploded view showing the fairing, payload, and first and second stages (bottom).

"MATLAB and Simulink saved us about 90% on costs compared with the alternative we considered while giving us the coding flexibility to develop our own modules and fully understand the assumptions being made, which is essential when reporting results to other teams."

— Patrick Harvey, Virgin Orbit

LauncherOne is Virgin Orbit's two-stage launch vehicle for delivering small satellites into low earth orbit. To reduce costs and increase launch location flexibility, LauncherOne is designed to be air-dropped from a 747-400 carrier aircraft in flight. Each mission will entail several crucial separation events, including the separation of LauncherOne from its carrier aircraft, the first stage from the second, the fairing from the second, and the satellite payload from the second.

When the structural design for LauncherOne was still in development, the team had to account for a number of unknowns in the analysis of separation events, including the mass properties of each component as well as the forces and timing characteristics of the pneumatic and spring pushers used to initiate separations. The team needed to run thousands of Monte Carlo simulations while varying the values of these uncertain parameters to determine whether a specific parameter combination would cause a collision.

Virgin Orbit engineers modeled and simulated LauncherOne stage and payload separation events with Simulink and Simscape Multibody, using Parallel Computing Toolbox™ to run simulations in parallel on multicore processors. Working in Simulink with Simscape Multibody, the team constructed a preliminary model consisting of basic 3D shapes, including spheres, cones, and cylinders. The team is currently working on simulating the air-drop separation event, which will incorporate a model of aerodynamic forces and effects. The team is also refining the model based on results from floor tests of flight hardware in preparation for the spacecraft's maiden launch.

Summary

A system model is at the center of development, from requirements capture to design, implementation, and testing. That's the essence of Model-Based Design. With this system model you can:

- Link designs directly to requirements
- Collaborate in a shared design environment
- Simulate multiple what-if scenarios
- Perform wise design choices based on simulation
- Optimize system-level performance
- Automatically generate embedded software code, reports, and documentation
- Detect errors earlier by testing earlier

Tools for Model-Based Design

Foundation Products

MATLAB

Analyze data, develop algorithms, and create mathematical models

Simulink

Model and simulate embedded systems

Requirements Capture and Management

Simulink Requirements

Author, manage, and trace requirements to models, generated code, and test cases

System Composer

Design and analyze system and software architectures

Design

Simulink Control Design

Linearize models and design control systems

Stateflow

Model and simulate decision logic using state machines and flow charts

Simscape

Model and simulate multidomain physical systems

Satellite Communications Toolbox

Simulate, analyze, and test satellite communications systems and links

Code Generation

Simulink Coder

Generate C and C++ code from Simulink and Stateflow models

Embedded Coder

Generate C and C++ code optimized for embedded systems

HDL Coder

Generate VHDL and Verilog code for FPGA and ASIC designs

Test and Verification

Simulink Test

Develop, manage, and execute simulation-based tests

Simulink Check

Verify compliance with style guidelines and modeling standards

Simulink Coverage

Measure test coverage in models and generated code

Simulink Real-Time

Build, run, and test real-time applications

Polyspace Products

Prove the absence of critical run-time errors

Learn More

mathworks.com has a range of resources to help you ramp up quickly with Model-Based Design. We recommend that you begin with these:

Overview

MATLAB and Simulink for Space Systems

Interactive Tutorials

MATLAB Onramp

Signal Processing Onramp

Image Processing Onramp

Machine Learning Onramp

Deep Learning Onramp

Simulink Onramp

Stateflow Onramp

Simscape Onramp

Control Design Onramp

Reinforcement Learning Onramp

Webinars

Simulink for New Users (36:05)

Model-Based Design of Control Systems (54:59)

Accelerating the Pace and Scope of Control System Design (51:03)

Impact of Digital Transformation and AI in Space Systems Engineering (1:23:50)

Onsite or Self-Paced Training Courses

MATLAB Fundamentals

Simulink for System and Algorithm Modeling

Control System Design with MATLAB and Simulink

Additional Resources

Consulting Services