WHITE PAPER

Model-Based Design for Aerospace Control Systems

MathWorks®

Imagine that your team is developing an antiskid brake control system for an aircraft. The system incorporates a combination of physics (e.g., lift, drag), control logic, and external conditions (e.g., temperature, precipitation). Before you begin the design, you want to address some key questions—for example:

- How do we size the brake system?
- 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—in the case of the brake control system, a model incorporating lift, drag, engine thrust, and the trailing link main landing gear. You can simulate the model at any point to get an instant view of system behavior and to test out multiple what-if scenarios 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?

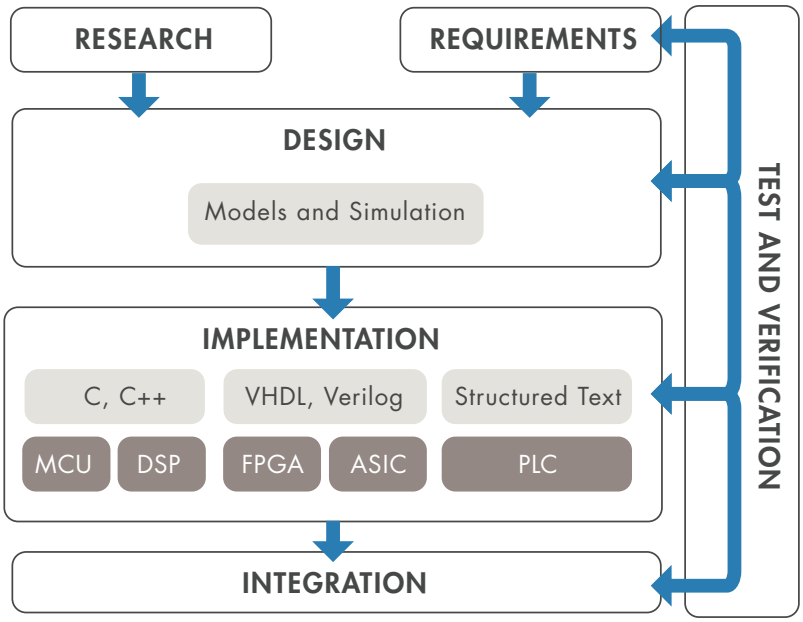
The best way to understand Model-Based Design is to see it in action:

> A team of automotive engineers sets out to build an engine control unit (ECU) for a passenger vehicle. Because they are using Model-Based Design, they begin by building an architecture model from the system requirements; in this case, the system is a four-cylinder engine. 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 ECU, and the plant—in this case, the engine 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 vehicle.

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.

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.

Case Study: *Embraer*

The Embraer Legacy 500.

> "Our group produced twice as many requirements, and had 50 times fewer issues per requirement, than was typical when we used a traditional methodology. The longest requirements-related delay using Model-Based Design was one day, whereas the shortest delay using a document basis was two weeks."
>
> *— Julio Graves, Embraer*

The Embraer Legacy 500 is the first midsized business jet with intelligent flight controls and full fly-by-wire technology. This technology, which replaces mechanical controls in the flight control system (FCS), enables more control surfaces to be actuated simultaneously, leading to smoother flights, reduced pilot workloads, and increased safety. Embraer worked with customers to develop high-level requirements for the Legacy 500. A principal challenge was translating the high-level requirements into well-written low-level requirements for the supplier who would develop the FCS software.

Early designs were developed without the intensive use of modeling and simulation. As a result, requirements sometimes had to be rewritten after they were delivered to the supplier, wasting time and increasing costs.

Embraer engineers decided to use Model-Based Design to define the low-level requirements for the Legacy 500 FCS. They created a detailed model of the FCS, as well as models of the aircraft dynamics and pilot inputs. The complete model comprises more than a million blocks and dozens of components, many with more than 700 inputs and 500 outputs.

They created functional test cases, which they ran on the model to verify that the high-level requirements were being met and to validate the low-level requirements. Embraer then delivered the written requirements to the supplier, who performed their own validation before implementing the system according to DO-178 Level A and other aviation standards.

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 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: *Intel*

The eight-rotor Intel® Falcon™ 8+.

> "Simulink makes it easy to design control algorithms and design them error-free. That means fewer design iterations and less testing time, because the algorithm is verified early on and we don't have to find every bug in manual flight tests."
>
> *— Jan Vervoorst, Intel*

To accelerate the development and testing of algorithms for UAV guidance, navigation, and control (GNC), Intel developed a simulation environment that accurately models environmental effects as well as the flight dynamics of UAVs with vastly different physical properties.

The team wanted a dynamic simulation environment that was modular and that they could customize for each existing UAV as well as for future UAVs that had entirely different geometries and structural designs.

Intel engineers developed a generic six-degrees-of-freedom (6DOF) model that calculates the state variables for a UAV given all forces and moments acting upon it. They developed a second model that computes those forces and moments based on RPM commands to the motors, 6DOF dynamics, and a set of customizable system parameters.

They combined the 6DOF model with the customized force-moment model to create a dynamic system model of each UAV, adding a state estimation block to incorporate simulated sensor data and noise.

The team logged and analyzed the flight test data, using the results to debug the control algorithms and refine their UAV models.

Code Generation

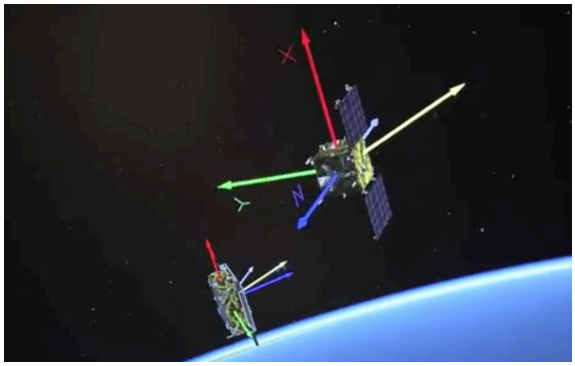
In a traditional workflow, embedded code must be handwritten from system models or from scratch. Software engineers write control algorithms based on specifications written by control systems engineers. Each step in this process—writing the specification, manually coding the algorithms, and debugging 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: *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 guidance, navigation, and control (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.

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 standards.

Case Study: ***Korea Aerospace Research***

The Korea Aerospace Research Institute manned helicopter.

"DO-178C procedures are automatically reflected in our Simulink development and verification environment. As a result, even developers who have no experience with DO-178C are able to meet the guidelines."

— Youngshin Kang, Korea Aerospace Research Institute

The Korea Aerospace Research Institute (KARI) adopted Model-Based Design to develop and test DO-178C-compliant software for an automatic flight control system (AFCS) used in manned helicopters.

On similar projects in the past, the engineers who developed the control laws would pass their designs to operational flight program software developers for hand-coding and manual testing. Subtle misunderstandings between the control law designer and the software developer led to errors in the software, some of which went undetected until flight tests. KARI wanted to eliminate these kinds of errors by generating software directly from their design.

The team developed a control law model that included submodels for basic stabilization control, external loop control, and mode authorization condition switching. They then combined the control law model with a nonlinear motion model of the aircraft and ran closed-loop simulations that included fault conditions for both actuators and sensors.

The KARI engineers followed an automated verification and validation process to perform model coverage analysis, detect design errors and dead logic, and check the model's compliance with safety standards and high-integrity modeling guidelines, including DO-178C guidelines.

The used Simulink Test™ to manage and run requirements-based tests and create test harnesses that enabled the team to achieve 100% test coverage for each submodel in the control law model. Finally, the team generated code from the validated control law model and performed unit testing on the generated code.

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: *Danfoss*

The Danfoss VLT® AutomationDrive FC302.

> "We completed our first solar inverter project with Model-Based Design on schedule, despite ramping up new engineers and adopting a new design process. For our second project, we actually reduced development time by 10–15%."
>
> — *Jens Godbersen, Danfoss*

To help meet increased demand for its products, the Danfoss power electronics group hired new engineers and re-evaluated its embedded software development processes, which up to then had relied on manual coding. With a traditional development process and manual coding, errors remained undetected until hardware prototype and certification testing.

While Danfoss knew they needed a new process, they were worried that adopting Model-Based Design would jeopardize deadlines. Bringing the team up to speed would take time. In addition, work on a new product, a solar inverter, had already begun. Model-Based Design would have to be introduced during development, and without affecting project deadlines.

Working with MathWorks consultants, Danfoss first developed a plan to ensure successful adoption of Model-Based Design. Danfoss engineers attended on-site training courses on Simulink, Stateflow®, and Embedded Coder® led by MathWorks engineers.

The team then completed a pilot project in which they rebuilt an existing software component that had been coded by hand. For the pilot, they decided to focus on three core capabilities of Model-Based Design: modeling, simulation, and code generation. After completing the pilot project, the team fully transitioned to Model-Based Design for development of the new solar inverter.

In weekly phone calls, MathWorks consultants advised them on the best way to get started, provided feedback on early versions of the models, and helped the team apply industry best practices to maximize model reuse and improve generated code performance.

The team completed development on schedule, and the test and certification campaign progressed smoothly due to the extensive simulations the team had performed in preparation.

Now that they have demonstrated the success of the new workflow, more engineers are involved in Model-Based Design across the organization, and they have built a library of models and a knowledge base that can be reused on future projects.

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
- Optimize system-level performance
- Automatically generate embedded software code, reports, and documentation
- Detect errors earlier by testing earlier

> "Three years ago, SAIC Motor did not have rich experience developing embedded control software. We chose Model-Based Design because it is a proven and efficient development method. This approach enabled our team of engineers to develop highly complex HCU control logic and complete the project ahead of schedule."
>
> *— Jun Zhu, SAIC Motor Corporation*

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

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:

Interactive Tutorials

MATLAB Onramp

Simulink Onramp

Stateflow 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)

Modeling, Simulating, and Generating Code for a Solar Inverter (45:00)

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