WHITE PAPER

Developing IEC 62304–Compliant Embedded Software for Medical Devices

MathWorks®

Whether you are an embedded software developer or a medical device engineer, this white paper will introduce you to the best practices for designing software for medical devices, from requirements management through design, implementation, and integration to verification and testing. You will learn how to produce higher-quality software while reducing design errors and development time.

Imagine that your team is developing controls for the actuation part of a surgical robotic system. The system is driven by electric motors with controls for haptic feedback. Before you begin the design, you want to resolve some key questions—for example:

Corindus CorPath GRX Surgical Robotic System.

- How do we size the motors for optimal power and precision?
- Can we incorporate late-stage requirement changes without project delay?
- How can we test the design at the system level before integration?
- Does our design process comply with medical standards such as IEC 62304?

If your team is using handwritten code and document-based requirements, the only way to answer these questions is through trial and error or testing on a physical prototype. This is the case whether the application is a surgical instrument, a ventilator, an infusion pump, or a dialysis machine. If a single requirement changes, the entire system might need to be recoded and tested, delaying the project by weeks or even months.

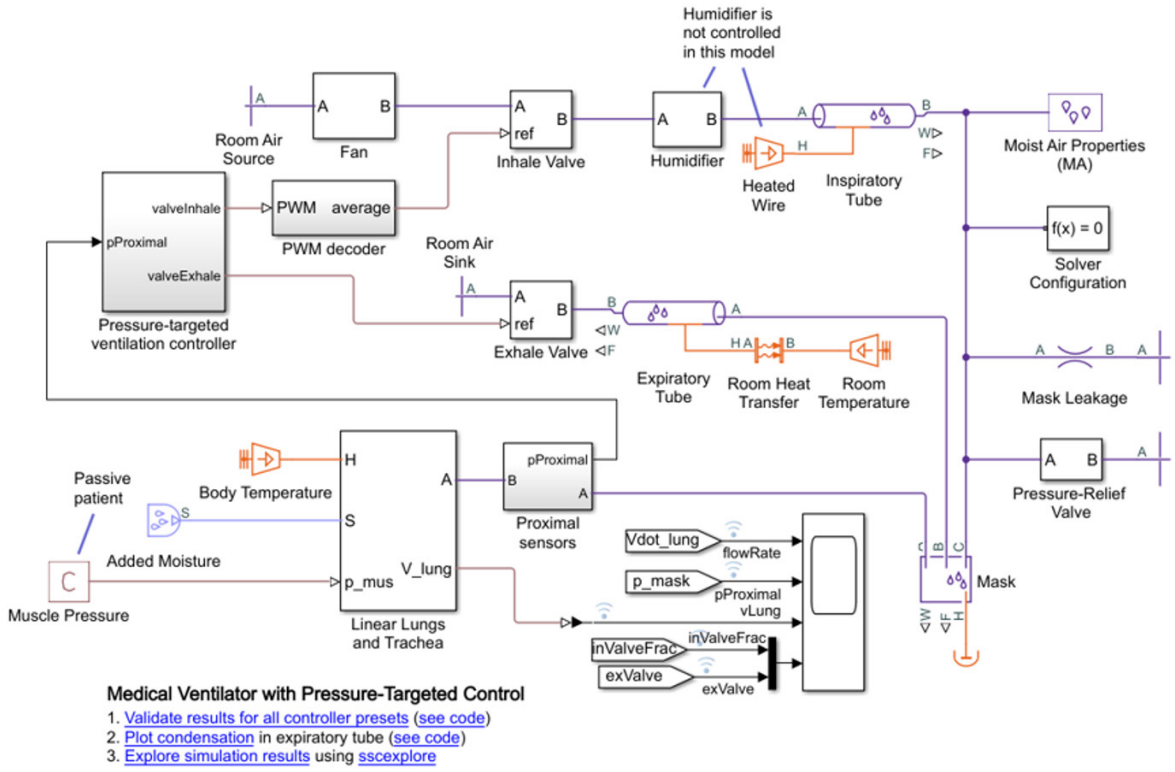
Instead of handwritten code and documents, your team can create a system model using Model-Based Design with MATLAB® and Simulink®. In the case of the surgical robotic system, the model would comprise the robotic arms, motors, and control unit. You can simulate the model at any point to get an instant view of system behavior and to test out multiple what-if scenarios without risk, delay, or reliance on costly hardware. The design can then be integrated into the final hardware—all within an IEC 62304-compliant development process.

> "Model-Based Design enabled our small product development team to develop and demonstrate telerobotic capabilities in just four months, reducing costs and development time."
>
> — *Per Bergman, Corindus*

This white paper introduces Model-Based Design for medical device development and provides tips and best practices for getting started. Using real-world examples, it shows how teams that have adopted Model-Based Design to reduce development time, minimize integration issues, and deliver safety-critical products while complying with medical device regulations and standards.

I. What Is Model-Based Design?

The best way to understand Model-Based Design is to see it in action, using the design of a mechanical ventilator as an example.



Mechanical ventilator model in Simulink and Simscape.

A team of medical device engineers sets out to build a control unit for the valves that regulate the flow, volume delivery, and inspiratory and expiratory functions on a mechanical ventilator. Because the engineers are using Model-Based Design, they begin by building an architecture model from the system requirements.

The team then develops a model of the ventilator that includes tubes, valves, and a humidifier. This high-level, low-fidelity model also includes the control algorithms that will be running in the control unit, as well as a model of the plant—in this case, the patient attached to the ventilator.

The team performs initial system and integration tests by simulating this high-level model under various hospital and emergency scenarios to verify that the system is functioning correctly and that it responds to different situations without risk to the patient.

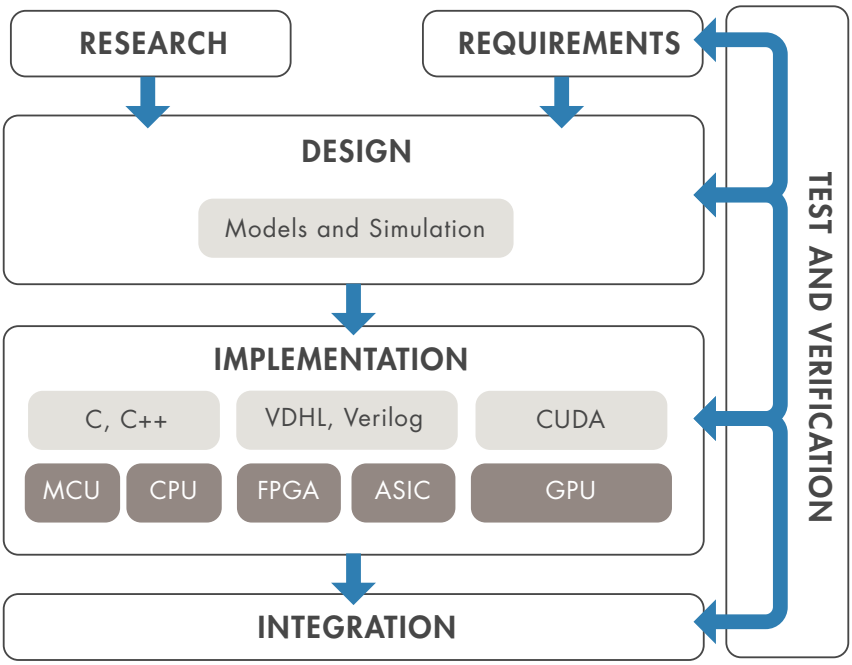
To test the responsiveness of the ventilator, the team adds more detail to the simulation in the form of an active patient. They continue testing and verifying the system-level behavior against the specifications. If the system is large and complex, the engineers can develop and test individual components independently but still test them frequently in a full system simulation.

Ultimately, they build a refined model of the system and the conditions in which it operates. This model captures knowledge about the system (the IP) accumulated from all the R&D efforts. The engineers then generate code from the model of the control algorithms for software testing and verification. They also generate all the documentation of unit and system-level tests needed for the regulatory approval process.

Following hardware-in-the-loop tests with rapid prototyping, they implement the generated code on production hardware to validate the operation of the ventilator.

You'll see from this scenario that Model-Based Design uses the same elements as traditional development workflows, but with two key differences:

- A system model is at the center of the development process, from requirements specification through design, implementation, and testing.
- Many of the time-consuming and error-prone steps in the workflow—for example, writing code, manual testing, and documentation—are automated.



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 the ones who design the system. Requirements may be "thrown over a wall," meaning there is 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. Furthermore, requirements can be traced all the way to generated code using traceability matrices. As a result, you can directly assess how a change in requirements affects the model and the code and take appropriate action.

Case Study: Corindus

Corindus bedside unit with extended-reach arm and touchscreen.

"Model-Based Design is essential to our ability to innovate because it lets us develop new capabilities and deploy them quickly. We can rapidly build a prototype, show that it meets requirements, and then perfect it as we move into the product development phase."

— Doug Teany, Chief Operating Officer, Corindus

Corindus has developed and deployed a robotic platform that enables physicians to perform percutaneous coronary intervention (PCI) or neurovascular intervention (NVI) on patients located hundreds to thousands of miles away.

The platform is an extension of the company's CorPath® GRX System, which allows physicians to operate on patients from a radiation-shielded workstation in the catheterization lab. Corindus used Model-Based Design with MATLAB and Simulink to create the system and to add support for real-time transmission of video and control data.

For the original system, Corindus wanted to accelerate development by validating their robotic control design through simulation before committing to hardware. They also wanted to verify the design via real-time simulation and testing and implement it on an embedded microcontroller.

To add remote capabilities, the Corindus team needed to send fluoroscopy and hemodynamics video data from the patient's location to the physician in real time and send joystick and other control data back. To incorporate telerobotic capabilities, they built a communication link that sends video data and control commands between the remote and local sites via two Speedgoat target computers running Simulink Real-Time™.

The first in-human, long-distance telerobotic-assisted PCIs were performed by Dr. Tejas Patel, who completed five successful procedures over two days on patients located 32 km away at the Apex Heart Institute in Ahmedabad, India.

Design

In a traditional approach, every design idea must be tested on a physical prototype. As a result, only a limited number of design ideas and scenarios can be explored because each test 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 design tradeoffs, and see how each design change affects the system.

Case Study: Weinmann

The MEDUMAT Transport ventilator.
Image © Weinmann Medical Technology.

"Model-Based Design with MATLAB and Simulink enabled us to handle the increased complexity and was instrumental in our achieving compliance certification. Working with models instead of handwritten code makes the embedded software easier to maintain and reuse and helps us explain the technology to a certification authority."

— Dr. Florian Dietz, Weinmann

The MEDUMAT Transport ventilator moves a mixture of oxygen and air into and out of the lungs of patients who require breathing support. It is designed for use in emergency care and during transport for intrahospital or interhospital transfers.

MEDUMAT Transport has a variety of sensors to measure pressure, flow, temperature, and molar mass (used to measure oxygen concentration). These sensors, combined with advanced pneumatics and electromagnetic valves, make MEDUMAT Transport the most advanced—and the most complex—ventilator that Weinmann has ever developed. To find the optimal algorithms for this system, the engineers needed to evaluate numerous design alternatives.

Weinmann engineers recognized that their traditional process, in which embedded software was handwritten, was not feasible for this project.

To overcome the challenge, they developed a plant model, which included hardware components as well as a mechanical model of human lungs. The team also modeled the controller and its state machines, including one state machine that tracks standby, startup, shutdown, and other operating modes and a second that manages the entire ventilation process. The system-level controller model served as the top level in a hierarchy of subsystems supporting the fundamental requirement for a modular software design and architecture.

After running closed-loop simulations of the controller and plant, the team generated production code for the control system and the sensor signal processing subsystem. They deployed the code to Infineon® and Texas Instruments™ MCUs, respectively, and performed unit tests on each subsystem within the model.

Code Generation

In a traditional workflow, embedded code must be written manually 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, 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. Handwritten coding errors are eliminated, and the model acts as a bridge between the control systems engineers and the software engineers. The generated code can be used for prototyping or production. It can be optimized for specific processor architectures and integrated with handwritten legacy code.

Case Study: World of Medicine

A 50L insufflator from WOM.

"Simulink enabled us to produce a stable control system in a short time. We modeled the entire system, including a state machine and cascaded PI controls. We refined this model to improve robustness and response times, then verified it with rapid control prototyping and generated embedded code."

— René Pätznick, WOM

Laparoscopy and other minimally invasive procedures must be performed within tightly confined spaces in the abdomen. To increase freedom of movement for surgical instruments, insufflators are used to expand the body cavity by blowing CO_2 gas into it. WOM, a market leader in insufflator and pump technologies for laparoscopy and hysteroscopy, uses Model-Based Design to accelerate the development of high-quality insufflator control software.

On similar projects in the past, WOM engineers used a traditional development workflow that involved hand-writing code. This approach made it difficult to identify and correct design and coding errors until late in the process, delaying software delivery.

For the new insufflator, WOM engineers moved to Model-Based Design. They used measured input-output data to create a nonlinear mathematical model of the abdominal cavity and incorporated this model into a plant model that included pressure sensors, actuators, and other hardware components.

Next, they developed a control model with two cascaded proportional integral (PI) controllers, one for flow and one for pressure. The team verified control functionality by running closed-loop simulations of the control model with the plant model.

To verify the real-time performance of the design, they generated C code from their control model and deployed it to real-time hardware connected to sensors and actuators in a prototype insufflator. After refining the design based on customer input, the team generated production code for the target Arm® Cortex-M® processor.

Following comprehensive integration tests and system-level tests, WOM received approval from the FDA and European regulatory authorities for the new insufflator, which is now in production and in clinical use.

Test and Verification

In a traditional development workflow, test and verification typically do not begin until the application is complete, making it difficult to identify and correct errors introduced during the design and coding phases.

In Model-Based Design, test and verification happen throughout the development cycle, from the moment you start modeling requirements and specifications to the moment when the completed design is ready for integration. Although you are testing more often and more thoroughly, you are also saving time, because you can prove that your design meets requirements: with the requirements captured in your model, you can verify and trace them to the design, tests, and code. You can automatically generate tests, create test reports, and check compliance with coding standards and guidelines using static analysis and formal methods.

Case Study: ITK Engineering

Dental drills featuring ITK Engineering's sensorless brushless motor control.

"Our plant model accurately reflected motor behavior, which enabled us to verify our controller and the hardware early in development. We quickly identified the root cause of an error on the first hardware prototype."

— *Alexander Reiss, ITK Engineering*

Sensorless brushless DC (BLDC) motors operate with less abrasion than brushed motors and are more reliable, quieter, and easier to maintain and sterilize. Compared with BLDC motors with sensors, sensorless BLDC motors are less expensive and more compact. However, the complex algorithms needed for sensorless control require much more engineering effort to develop.

ITK engineers needed to design and optimize a rotor position estimator, as well as a sophisticated cascade control for the dental drill motor that would comply with the IEC 62304 standard for medical device software.

When the project began, a prototype motor was unavailable. To meet their client's project deadline, ITK had to develop the controller software in parallel with the motor hardware.

ITK engineers designed, optimized, implemented, and tested the sensorless BLDC motor controller with Model-Based Design. Working from data sheets for existing motors and information provided by their client, the engineers modeled the BLDC motor, including its electrical and mechanical components, in Simulink.

After converting their floating-point controller design to fixed point, they reran simulations to verify the fixed-point model. The team also developed MATLAB scripts that performed batch unit testing of individual model components.

The controller and sensorless BLDC motor are currently in series production in dental drills.

II. Getting Started

While you and your team might see the benefits of moving to Model-Based Design, you 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 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 hand-writing 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 when we move to Model-Based Design?

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 carries an element of risk. Successful teams have mitigated this risk by introducing Model-Based Design gradually, taking focused steps that can 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.

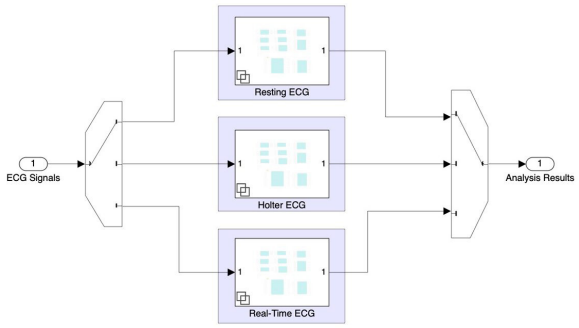
Q. Is Model-Based Design compliant with IEC 62304 software development process?

A. You can develop IEC 62304-compliant embedded software for medical devices with Model-Based Design. Model-Based Design incorporates verification and validation into the workflow, which ensures that the software is comprehensively tested and verified before integrating it into a medical device. In addition, parts of the documentation required by IEC 62304 are automatically generated for regulatory compliance. Most Simulink tools used in Model-Based Design have been TÜV SÜV certified for IEC 62304 compliant development workflow.

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 created without manual coding.
  - 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 dynamically 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: Khawaja Medical Technology



ECG signal analysis algorithms modeled in Simulink.

"We've seen a significant return on the investment we made in Model-Based Design, including higher quality, reduced development times, and faster ISO and IEC certification."

— *Dr. Antoun Khawaja, Khawaja Medical Technology*

Electrocardiogram (ECG) data analysis is essential for the recognition and treatment of cardiac diseases. It is applied in a variety of diagnostic settings, including preclinical, clinical, ambulatory, and in-home settings, as well as in clinical trials of new drugs.

As part of its cardiac drug development and approval process, a pharmaceutical company must investigate the new drug's effect on the heart. This involves analyzing ECG signals to identify abnormalities and ensure cardiac drug safety.

Engineers at Khawaja Medical Technology have developed novel and advanced algorithms that fully automate ECG signal analysis. The algorithms enable real-time monitoring and analysis of ECG signals from a subject who is resting, exercising, or wearing a Holter monitor. The engineering team used Model-Based Design with MATLAB and Simulink to develop and deploy algorithms for automated ECG analysis.

Using Simulink Check™, the team checked their models for compliance with modeling guidelines and standards, including IEC 62304. They authored and executed simulation-based tests with Simulink Test™, traced the tests to requirements, and measured test coverage with Simulink Coverage™.

They also developed a set of MATLAB classes, which they used to create reusable System objects™ for the signal processing and classification layer. These System objects perform a variety of tasks, such as detecting peaks in ECG signals, measuring signal characteristics, classifying arrhythmias, and diagnosing ventricular hypertrophy, myocardial infarction, and other heart conditions.

Leveraging these tools, Khawaja Medical Technology was able to reduce development time by 40%, accelerate compliance with ISO 13485 and IEC 62304, and build a prototype in months instead of years.

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

GPU Coder

Generate CUDA code for NVIDIA GPUs

Test and Verification

Simulink Test

Develop, manage, and execute simulation-based tests

Simulink Check

Measure design quality, track verification activities, and verify compliance with standards

Simulink Coverage

Measure test coverage in models and generated code

Polyspace® Products

Prove the absence of critical run-time errors in C/C++ code

Simulink Design Verifier

Identify design errors, prove requirements compliance, and generate tests

IEC Certification Kit

Qualify code generation and verification tools for IEC 62304 certification

Learn More

These resources will help your team ramp up quickly with Model-Based Design.

Interactive Tutorials

MATLAB Onramp

Simulink Onramp

Stateflow Onramp

Simscape Onramp

Videos

What Is Simulink? (2:15)

Model-Based Design with MATLAB and Simulink (2:08)

Getting Started with Simulink for Controls (11:30)

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

MATLAB and Simulink for Medical Devices