白皮书

汽车控制系统的基于模型设计

假设您的团队正在开发汽车的高级紧急制动系统 (AEBS)。即将发生碰撞时，AEBS 会发出警报声警告驾驶员。如果驾驶员没有反应，则会施加警告制动。如果驾驶员刹车但施力不够、无法避免碰撞，则系统会计算并施加所需的额外制动力。开始设计之前，你们希望解决一些关键问题，例如：

- 如何确定制动器的尺寸？
- 如果需求发生变化该怎么办？
- 如何优化设计以确保实现所期望的性能？
- 如何全面测试设计并将风险降至最低？

如果您的团队手动编写代码并采用基于文档的需求捕获方法，则解决以上问题的唯一方法是反复试错或执行物理原型测试，无论是为工业机器人、风力发电机、生产设备、自主车辆、挖掘机还是电动伺服驱动器开发控制系统，情况都是如此。如果某项需求发生变化，您将不得不重新编码和构建整个系统，继而会导致项目延迟数日乃至数周。

如果使用 MATLAB® 和 Simulink® 进行基于模型的设计，则您无需手写代码和使用文档，而是创建系统模型。以紧急制动系统为例，模型由制动器、车辆动力学、环境和控制系统组成。您可以随时进行模型仿真，即时查看系统行为，测试多种假设分析场景，同时无需承担风险和延迟，也无需依赖昂贵的硬件。

本白皮书介绍了基于模型的设计，提供了快速入门技巧和最佳实践。书中包含真实示例，展示各行各业的团队如何采用基于模型的设计缩短开发时间、最大限度地减少组件集成问题，并且交付更优质的产品。

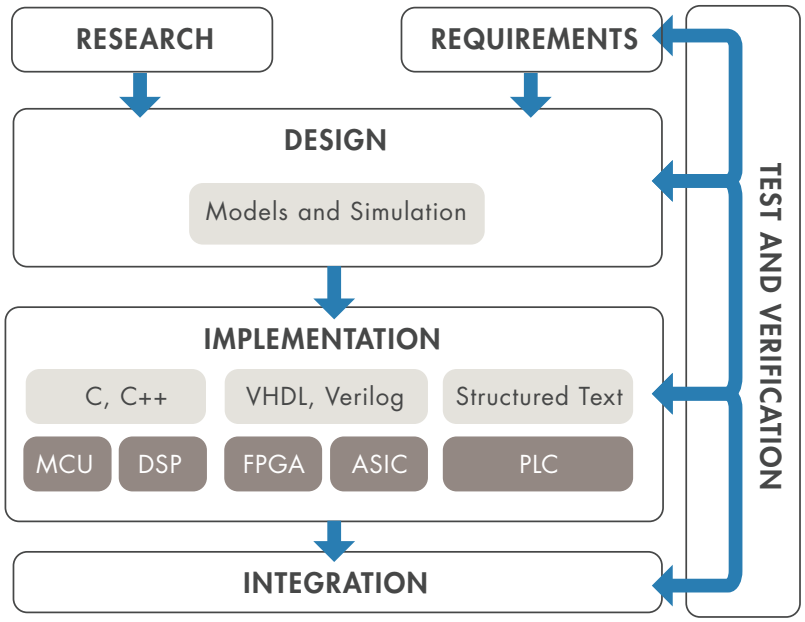
什么是基于模型的设计？

要理解基于模型设计，最好的方法是结合实例：

> 汽车工程师团队着手打造乘用车发动机控制单元 (ECU)。由于采用的是基于模型的设计，他们首先根据系统需求构建架构模型；在本例中，系统是指四缸引擎。而后，他们从中衍生出仿真/设计模型。这种高级低保真度模型包含将在 ECU 和被控对象（在本例中是指发动机和工作环境）中运行的那一部分控制软件。
>
> 团队通过在各种场景下进行高级模型仿真来执行初始系统和集成测试，验证是否正确表示系统以及系统是否对输入信号做出适当响应。
>
> 团队向模型中增加细节，对照规范不断测试和验证系统级行为。如果系统规模庞大且结构复杂，工程师可以独立开发和测试各个组件，但仍要在全系统仿真中频繁进行测试。
>
> 最终，团队针对系统及其工作环境建立了详细模型。该模型捕获了累积的系统 (IP) 知识。工程师基于控制算法模型自动生成代码，从而执行软件测试和验证。完成硬件在环测试后，团队将生成的代码下载到生产硬件，以便在实际车辆中进行测试。

该场景显示，基于模型的设计采用与传统开发工作流程相同的元素，但存在两个关键区别：

- 将工作流程中大量费时或易出错的步骤（如代码生成）自动化。
- 从需求捕获到设计、实现和测试，系统模型始终占据开发流程的核心。



基于模型设计的工作流程。

需求捕获和管理

传统工作流程基于文档捕获需求，交接可能引发错误和延迟。通常，创建设计文档或需求的工程师与设计系统的工程师并不是同一批人。两个团队对需求的认知很可能存在断层，这意味着两个团队并未建立持续清晰的沟通。

采用基于模型的设计，您可以在 Simulink 模型中编写、分析和管理需求。您可以使用自定义属性创建富文本需求，并将其链接到设计、代码和测试。同时，还可以从需求管理工具等外部源导入需求并进行同步。如果关联到设计的需求发生变化，您将会收到自动通知。因此，您可以确定受此变化直接影响的设计或测试部分，继而采取适当的措施应对变化。您可以定义、分析及指定系统和软件组件的架构与组合。

案例研究：Iveco

一辆 Iveco 重型汽车。

“我们的系统工程师直接与软件工程师合作开发 Simulink 模型。由于彼此之间对于需求不存在误解，这种方式加快了开发速度。当我们确信模型正确时，我们可以基于它生成不含实现错误的代码，从而节省更多时间。”

— Demetrio Cortese, Iveco

为了抓住拉美中型和重型车辆市场的机遇，Iveco 公司需要在大约六周时间内设计、实现、测试并交付用于带 9 速和 16 速变速箱的车辆的换挡范围抑制系统。最后期限迫在眉睫，使得软件开发计划极为紧凑，不容许出现规范或实现错误。

由于项目时间紧迫，团队计划使用一个包含 PLC 的现有硬件配置。然而，软件工程师没有为 PLC 编写结构化文本的经验。为了避免出现实现错误导致开发时间增加，Iveco 需要自动生成结构化文本。

Iveco 的传统做法是由系统工程师定义需求和规范，然后移交给软件工程师，但考虑到项目时间紧迫，这种方法并不可行。于是，系统工程师和软件工程师通力合作，在 Simulink 中开发了系统的初步模型。

软件工程师对模型进行了优化和定制，添加了约束、数据类型、内置测试和诊断。他们对模型进行仿真，以验证设计的完整性并识别溢出情况、未执行的模块和其他潜在问题。

工程师使用 PLC 和实际变速器进行实时试验台测试，快速调整模型，重新生成代码，并重新运行测试，直到管理系统满足其功能和性能需求。

设计

在传统方法中，每一种设计思路都必须在物理原型上进行编码和测试。因此，考虑到每一次测试迭代都会增加项目开发时间和成本，工程师只能探索少数设计思路和场景。

在基于模型的设计中，工程师可以尽情探索无穷无尽的各种思路。需求、系统组件、IP 和测试场景全被捕获到模型中，而且，由于可以进行模型仿真，您可以在构建昂贵硬件之前更早地开始研究设计问题和疑问。您可以快速评估多种设计思路，权衡折衷方案，了解每一项设计更改将会产生的系统影响。

案例研究：Ather Energy

Ather 450 智能电动踏板车。

“我们有很多颇具前景的创意，但身为一家小型初创企业，我们根本不具备足够的时间、资金和人力来为每个创意构建原型并逐一测试。采用基于模型的设计后，我们得以通过仿真发现并确认最佳创意，因而可以在更短的时间内交付功能更全面的踏板车。”

— Shivaram N.V., Ather Energy

在班加罗尔的 500 余万辆双轮踏板车中，绝大多数（约占该市所有车辆的 70%）以汽油为燃料提供动力，不仅噪声污染严重，二氧化碳排放更是惊人。为响应清洁替代能源倡议，初创企业 Ather Energy 打造了印度首辆智能电动踏板车。Ather 450 可以在 4 秒内从 0 加速到 40 km/h，最高速度可达 80 km/h，单次充电的最大续航里程高达 75 km。

鉴于 Ather 450 在市场上尚无同类产品，团队面临众多未知因素。构建包含踏板车及其主要部件在内的被控对象模型后，他们运行仿真来评估骑行和使用场景，包括斜坡、多人同车、极端温度、电池电量基本耗尽等情况。

他们根据仿真结果对设计进行合理的折衷；例如，结果表明增加电池容量可延长续航里程，但也将增加成本、尺寸并改变踏板车的重心。他们优化设计，直至找出达到目标加速和续航里程需求、同时又符合成本、尺寸和温度限制的电动机和电池配置。

除设计踏板车本身以外，工程师还开发了嵌入式控制算法，用于电池充电、温度管理及其他一些关键功能。在缺少详细组件数据的情况下，团队采取实证研究法进行电池电芯建模。工程师在各种温度和荷电状态水平下对电池进行测试，使用测得的输入/输出数据，根据电芯的电特性和热特性创建黑盒模型。

紧接着，团队开发出电池充电、电力控制和温度控制算法。工程师使用被控对象模型运行闭环仿真，验证其控制设计。他们基于控制器模型生成代码，将其部署到踏板车的 ARM® Cortex® 处理器及充电站的 TI C2000™ 微控制器中。

Ather 450 现已投产，在班加罗尔首次发布并设有 31 个充电站，同时钦奈也设有 7 个充电站。

代码生成

在传统工作流程中，嵌入式代码必须基于系统模型或从零开始手写。软件工程师按照控制系统工程师编写的规范编写控制算法。从编写规范、手动编写算法代码到调试手写代码，此过程中的每一步都既费时又容易出错。

采用基于模型的设计时，您无需手动编写数千行代码，而是可以直接从模型中生成代码，模型在软件工程师与控制系统工程师之间架起桥梁。生成的代码可用于快速原型设计或生产。

快速原型设计提供了在硬件上实时测试算法的一种快速经济的方法，在几分钟内即可完成原本需要几个星期的设计迭代。您可以使用原型硬件，也可以使用生产 ECU。您可以使用这些快速原型设计硬件和设计模型开展硬件在环测试及其他测试和验证活动，先确认硬件和软件设计，再投入生产。

产品级代码生成可将您的模型转换为实际代码，进而在生产嵌入式系统中实现。生成的代码可以针对特定处理器架构进行优化并与手写的既有代码集成。

案例研究：Ponsse

Ponsse 的 Scorpion 伐木机。

“我们可以基于 Simulink 模型生成正确无误的代码，这使得控制工程师能够专注于算法设计，而软件工程师则可以专注于固件层的编程。因此，我们得以加快开发、提升质量、降低成本。”

— Juha Inberg, Ponsse

Ponsse Scorpion 是一款八轮伐木机，专为崎岖的森林地形而设计。这台设备的独特框架由三个部分组成，采用旋转幅度高达 12% 的关节相连。该框架使中部的驾驶室能够保持水平，装有车轮的前部和后部则可根据地形的变化进行调整。

在以前的项目中，Ponsse 控制工程师在 MATLAB 中开发和调试算法，软件工程师将这些算法手动转换成 C 代码。但是，随着项目中控制算法的复杂度不断增加，这种方法难以为继。在 C 代码中引入人为错误的风险越来越大，算法初始设计与基于硬件的算法验证之间的间隔也十分漫长。Ponsse 希望缩短这一间隔，尽可能减少编码错误，并减少总体开发时间。

工程师开发了一个控制模型，用于处理来自加速度计和陀螺仪的输入，驱动液压阀来保持 Scorpion 中部的框架水平。另外，他们还建立了一个原型控制器，根据模型生成实时应用，然后将其部署到 Speedgoat 目标计算机硬件上。

该团队使用该实时原型在实际 Scorpion 硬件上进行测试。基于测试结果，他们对控制模型进行微调，而后重新生成更新的原型并重新测试。然后他们针对 Scorpion 的 ECU 基于模型生成 C 代码。

他们将生成的代码与 ECU 的固件及其他低级接口代码集成在一起，先在第三方仿真器上测试，之后在实际的 Scorpion 伐木机上测试。

自 Scorpion 项目成功之后，Ponsse 工程师继续使用基于模型设计为 Ponsse 产品线中的其他伐木机开发了嵌入式控制器，并在此过程中重用了 Scorpion 控制设计中的滤波器和模型组件。

测试和验证

在传统开发工作流程中，测试和验证通常安排在流程后期，因而难以识别及纠正设计和编码阶段引入的错误。

在基于模型的设计中，测试和验证贯穿整个开发周期，从建模需求和规范开始，并在设计、代码生成和集成中持续开展。您可以在模型中编写需求，并将需求追溯到设计、测试和代码。形式化方法有助于证明您的设计满足需求。您可以生成报告和工件，对照功能安全标准进行软件认证。

案例研究：Lear

Lear 硬件在环测试。

“在 BCM 项目中，我们使用 Simulink 中的可执行功能模型开展虚拟集成和测试，在实现前找出了 95% 以上的需求问题，而我们在开始使用基于模型的设计之前，这个数字仅为 30%。”

— Jason Bauman，Lear

如今，汽车 OEM 不断要求供应商在 ECU 软件中提供更多功能。汽车电子和配电系统越来越复杂，因此需求必须清晰、完整和一致。在传统的手工编码工作流程中，开发后期往往会出现需求模棱两可或相互矛盾的问题，从而导致延期或成本超支。Lear 公司的工程师使用基于模型的设计来开发、验证和实现车身控制电子系统，一举解决了以上难题。

工程师分析了客户需求，并将整个系统划分为不同的组件，如照明、电池管理以及汽车起动控制。然后，他们开发了功能行为模型和被控对象模型，用于功能和单元测试。工程师分析模型覆盖率并不断完善测试用例、设计和需求，直至达到满意的模型覆盖率水平，包括决策覆盖率和修正条件/决策覆盖率 (MC/DC)。

在验证了近 400 个单元模型后，该团队生成了 C 代码，并通过软件在环 (SIL) 测试验证了生成的代码，这些测试重用了为单元模型测试生成的测试用例。

Lear 的工程师将针对各个单元模型生成的代码集成到了二三十个功能级组件中，而这些组件又进一步集成为一个完整的系统模型。团队约见了客户，对组件和整个模型运行了仿真，以解决原始设计规范中模棱两可的部分。

团队使用 MATLAB 脚本实现自动化，将测试用例自动转换成适用于硬件在环 (HIL) 和基于车辆测试的测试向量。该团队总共生成了大约 700,000 行代码，并且在整个开发周期中多次重用测试用例，从而缩短了整体开发时间。

快速入门

尽管您的团队可能已经了解到迁移至基于模型的设计将带来的益处，但也难免担忧可能面临的种种组织、后勤及技术风险和挑战。本节解答了工程团队考虑采用基于模型的设计时提出的常见问题，并提供一系列经实践验证的、能够帮助工程团队管理过渡事宜的技巧和最佳实践。

问：引入基于模型的设计会对工程分工造成怎样的影响？
答：基于模型的设计并不会取代控制设计和软件架构方面的工程专业知识。采用基于模型的设计，控制系统工程师不仅可以提供传统书面形式的需求，还能提供模型和代码形式的可执行需求。软件工程师可以更快地完成应用程序软件编码，腾出更多时间执行架构建模，为操作系统、设备驱动程序及其他平台软件编写代码，以及执行系统集成。控制系统工程师和软件工程师自开发流程早期就能介入系统级设计。

问：现有代码会发生什么变化？
答：它可以成为设计的一部分；您的系统模型可以同时包含原生设计建模组件和既有组件。这意味着您可以逐步引入既有组件，同时继续进行系统仿真、验证和代码生成。

问：在采用基于模型的设计时，是否存在推荐做法？
答：尝试新方法和设计工具总是不免带来风险。根据一些团队的成功经验，要降低此类风险，不妨逐步引入基于模型的设计，采取有针对性的步骤帮助推进项目并避免速度减缓。不同规模的企业都选择从小团队级别开始采用基于模型的设计。他们通常从单个项目入手，快速致胜并在早期成功的基础之上继续推进和延伸。积累经验后，他们开始在部门级别推广基于模型的设计，使模型成为团队整个嵌入式系统开发项目的核心。

以下**四个最佳实践**已在多个团队的实践中证明有效：

- **使用项目的一小部分进行试验。**选择一个新的嵌入式系统领域，构建软件行为模型，然后基于模型生成代码，是一个不错的开始方式。团队成员只需投入少量精力即可掌握新工具和新方法，从而完成这项小调整。该结果能够呈现基于模型设计的一些关键优势：
    - 可以自动生成高质量代码。
    - 代码与模型行为相匹配。
    - 采用模型仿真，不仅可以更轻松地调试算法，还能获得更为深层的信息，胜过基于桌面测试 C 代码。
- **添加系统级仿真，在初始建模成果的基础之上继续推进项目。**如本文的前几节所示，您可以通过系统仿真确认需求、调查设计问题以及开展早期测试和验证。系统模型无需实现高保真度；所包含的细节只需确保接口信号采用正确的单位并连接适当的信道，同时确保能够捕获系统的动态行为即可。您可以通过仿真结果提前了解被控对象和控制器的行为模式。
- **使用模型解决特定设计问题。**即使不开发全尺寸被控对象、环境和算法模型，您的团队也可以获得有针对性的帮助。例如，假设您的团队需要为作动系统使用的螺线管选择参数。他们可以开发一个简单模型，在螺线管四周绘制概念性“控制体积”，包括驱动因素和作用因素。团队可以测试各种极端工况并提取基本参数，但不必推导方程。之后，团队可以存储此模型，以用于解决其他设计问题或融合至今后的项目。
- **从基于模型设计的核心元素入手。**基于模型设计的直接优势包括：支持创建组件和系统模型；通过仿真测试和确认设计；自动生成 C 代码以进行原型设计和测试。在此基础上，您可以进一步考虑采用高级工具和实践，引入建模指南、自动合规性检查、需求可追溯性及软件构建自动化。

案例研究：Danfoss

Danfoss VLT® AutomationDrive FC302。

“得益于基于模型的设计，尽管增加了新工程师并采用了新的设计流程，我们仍按时完成了第一个太阳能逆变器项目。至于第二个项目，我们实际将开发时间缩短了10–15%。”

— Jens Godbersen，Danfoss

为满足市场对其产品不断增长的需求，Danfoss 电力电子集团招聘了一批新工程师，并重新评估了一直以来依赖于手动编码的嵌入式软件开发流程。采用传统开发流程和手动编码，一些问题往往直到硬件原型和认证测试阶段才会被检测出来。

Danfoss 深知，他们需要一套新流程，但又担心采用基于模型的设计可能无法保证按时交付。一方面，保证整个团队跟上进度需要时间。另一方面，新产品（太阳能逆变器）的开发工作已经开始。团队必须在开发中途引入基于模型的设计，而且不能影响项目按时交付。

Danfoss 与 MathWorks 顾问密切合作，首先制定了一份计划，确保成功采用基于模型的设计。Danfoss 工程师参加了由 MathWorks 工程师主讲的现场培训课程，学习 Simulink、Stateflow®和 Embedded Coder®。

接着，该团队完成了一个试点项目，对采用手动编码的现有软件组件进行重建。对于试点项目，他们决定重点关注基于模型设计的三个核心功能：建模、仿真和代码生成。在完成试点项目后，该团队全面过渡到基于模型的设计，进行新型太阳能逆变器的开发。

在每周的电话交流上，MathWorks 顾问就起步阶段的最佳做法向他们提出建议，为早期版本的模型提供反馈，并帮助该团队运用行业最佳实践，最大限度地提高模型重用率，改进生成代码的性能。

该团队按计划完成了开发工作，由于在准备过程中进行了大量仿真，测试和认证活动进展顺利。

新工作流程的成功有目共睹，因此，企业上下越来越多的工程师开始参与到基于模型的设计中，他们构建了模型库和知识库，可在今后的项目中重复利用。

小结

从需求捕获到设计、实现和测试，系统模型始终占据开发流程的核心。这是基于模型设计的精髓。您可以使用此系统模型：

- 直接关联设计与需求
- 在共享设计环境中开展协作
- 仿真多种假设分析情景
- 优化系统级性能
- 自动生成嵌入式软件代码、报告和文档
- 更早测试，从而更早发现错误

“三年前，上汽集团并不具备太多嵌入式控制软件开发经验。我们之所以选择基于模型的设计，是因为它是一种成熟而高效的开发方法。该方法帮助我们的工程师团队成功开发高度复杂的 HCU 控制逻辑并提前完成项目。”

— 朱军，上海汽车集团股份有限公司

基于模型设计的工具

基础产品

MATLAB®

分析数据、开发算法及创建数学模型

Simulink®

嵌入式系统建模和仿真

需求捕获和管理

Simulink Requirements™

编写需求、管理需求并将需求追溯到模型、生成的代码和测试用例

System Composer™

设计和分析系统架构与软件架构

设计

Simulink Control Design™

线性化模型并设计控制系统

Stateflow®

使用状态机与流程图进行决策逻辑的建模和仿真

Simscape™

建模和仿真多域物理系统

代码生成

Simulink Coder™

从 Simulink 和 Stateflow 模型生成 C 和 C++ 代码

Embedded Coder®

生成针对嵌入式系统优化的 C 和 C++ 代码

HDL Coder™

生成用于 FPGA 和 ASIC 设计的 VHDL 和 Verilog 代码

测试和验证

Simulink Test™

开发、管理和执行基于仿真的测试

Simulink Check™

验证是否符合风格指南和建模标准

Simulink Coverage™

测量模型和生成的代码的测试覆盖率

Simulink Real-Time™

构建、运行和测试实时应用

Polyspace® 产品

证明不存在严重的运行时错误

了解更多

Mathworks.com 提供了大量资源，帮助您快速掌握基于模型的设计。
我们建议您使用以下资源开始学习：

交互式教程

MATLAB 入门之旅

Simulink 入门之旅

Stateflow 入门之旅

在线研讨会

Simulink 快速入门 (36:05)

控制系统的基于模型设计 (54:59)

促进和拓展控制系统设计 (51:03)

为太阳能逆变器建模、仿真和生成代码 (45:00)

现场或自定进度培训课程

MATLAB 基础

Simulink 系统和算法建模

使用 MATLAB 和 Simulink 进行控制系统设计

其他资源

咨询服务