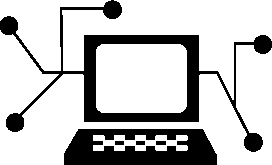
**Simulink Coder™**

**用户指南**

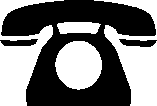
**如何联系 MathWorks**

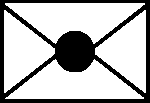
 最新动态： [**www.mathworks.com**](https://www.mathworks.com/)

销售和服务： [**www.mathworks.com/sales\_and\_services**](https://www.mathworks.com/sales_and_services)

用户社区： [**www.mathworks.com/matlabcentral**](https://www.mathworks.com/matlabcentral)

技术支持： [**www.mathworks.com/support/contact\_us**](https://www.mathworks.com/support/contact_us)

 电话： 010-59827000

迈斯沃克软件 (北京) 有限公司

北京市朝阳区望京东园四区 6 号楼

北望金辉大厦 16 层 1604

Simulink® Coder™ 用户指南

© COPYRIGHT 2011–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

**商标**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [**www.mathworks.com/trademarks**](https://www.mathworks.com/trademarks)for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

**专利**

MathWorks products are protected by one or more U.S. patents. Please see [**www.mathworks.com/patents**](https://www.mathworks.com/patents)

for more information.

|  |  |  |
| --- | --- | --- |
| **修订历史记录** |  | |
| 2011 年 4 月 | 仅限在线版本 | 版本 8.0（版本 2011a）中的新增内容 |
| 2011 年 9 月 | 仅限在线版本 | 版本 8.1（版本 2011b）中的修订内容 |
| 2012 年 3 月 | 仅限在线版本 | 版本 8.2（版本 2012a）中的修订内容 |
| 2012 年 9 月 | 仅限在线版本 | 版本 8.3（版本 2012b）中的修订内容 |
| 2013 年 3 月 | 仅限在线版本 | 版本 8.4（版本 2013a）中的修订内容 |
| 2013 年 9 月 | 仅限在线版本 | 版本 8.5（版本 2013b）中的修订内容 |
| 2014 年 3 月 | 仅限在线版本 | 版本 8.6（版本 2014a）中的修订内容 |
| 2014 年 10 月 | 仅限在线版本 | 版本 8.7（版本 2014b）中的修订内容 |
| 2015 年 3 月 | 仅限在线版本 | 版本 8.8（版本 2015a）中的修订内容 |
| 2015 年 9 月 | 仅限在线版本 | 版本 8.9（版本 2015b）中的修订内容 |
| 2015 年 10 月 | 仅限在线版本 | 版本 8.8.1（版本 2015aSP1）中的再发布内容 |
| 2016 年 3 月 | 仅限在线版本 | 版本 8.10（版本 2016a）中的修订内容 |
| 2016 年 9 月 | 仅限在线版本 | 版本 8.11（版本 2016b）中的修订内容 |
| 2017 年 3 月 | 仅限在线版本 | 8.12 版（版本 2017a）中的修订内容 |
| 2017 年 9 月 | 仅限在线版本 | 版本 8.13（版本 2017b）中的修订内容 |
| 2018 年 3 月 | 仅限在线版本 | 版本 8.14（版本 2018a）中的修订内容 |
| 2018 年 9 月 | 仅限在线版本 | 9.0 版（版本 2018b）中的修订内容 |
| 2019 年 3 月 | 仅限在线版本 | 版本 9.1（版本 2019a）中的修订内容 |
| 2019 年 9 月 | 仅限在线版本 | 版本 9.2（版本 2019b）中的修订内容 |
| 2020 年 3 月 | 仅限在线版本 | 版本 9.3（版本 2020a）中的修订内容 |
| 2020 年 9 月 | 仅限在线版本 | 版本 9.4（版本 2020b）中的修订内容 |
| 2021 年 3 月 | 仅限在线版本 | 版本 9.5（版本 2021a）中的修订内容 |
| 2021 年 9 月 | 仅限在线版本 | 版本 9.6（版本 2021b）中的修订内容 |
| 2022 年 3 月 | 仅限在线版本 | 版本 9.7（版本 2022a）中的修订内容 |
| 2022 年 9 月 | 仅限在线版本 | 版本 9.8（版本 2022b）中的修订内容 |

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at [www.mathworks.com/support/](https://www.mathworks.com/support/bugreports/) [bugreports/](https://www.mathworks.com/support/bugreports/). In the search bar, type the phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers. To save a search, click Save Search.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

### [模型架构和设计](#_bookmark0)

[**Simulink Coder 的建模环境**](#_bookmark1)

[**1**](#_bookmark1)

### [在 Simulink Coder 中进行建模](#_bookmark2)

[**2**](#_bookmark2)

[模块建模规范](#_bookmark3) [2-2](#_bookmark3)

[子系统的建模规范](#_bookmark4) [2-3](#_bookmark4)

[图建模规范](#_bookmark5) [2-4](#_bookmark5)

[MATLAB 函数的建模规范](#_bookmark6) [2-5](#_bookmark6)

[模型配置的建模规范](#_bookmark7) [2-6](#_bookmark7)

[Simulink Coder 中的计时器](#_bookmark8)

[3](#_bookmark8)

[**Simulink Coder 中基于时间的调度**](#_bookmark9)

[4](#_bookmark9)

[基于时间的调度和代码生成](#_bookmark10) [4-2](#_bookmark10)

[采样时间注意事项](#_bookmark11) [4-2](#_bookmark11)

[任务模式](#_bookmark12) [4-2](#_bookmark12)

[模型执行和速率转换](#_bookmark13) [4-3](#_bookmark13)

[在 Simulink 模型仿真期间执行](#_bookmark14) [**4-3**](#_bookmark14)

[模型实时执行](#_bookmark15) [4-4](#_bookmark15)

[单任务与多任务操作](#_bookmark16) [4-4](#_bookmark16)

[处理速率转换](#_bookmark17) [4-6](#_bookmark17)

[速率转换](#_bookmark18) [4-6](#_bookmark18)

[数据传输问题](#_bookmark19) [4-7](#_bookmark19)

[Rate Transition 模块选项](#_bookmark21) [**4-7**](#_bookmark21)

[自动速率转换](#_bookmark22) [4-9](#_bookmark22)

[可视化插入的 Rate Transition 模块](#_bookmark23) [**4-10**](#_bookmark23)

[周期性采样率转换](#_bookmark24) [4-11](#_bookmark24)

[使用 volatile 关键字保护数据完整性](#_bookmark26) [**4-14**](#_bookmark26)

[从算法代码和数据中分离 Rate Transition 模块代码和数据](#_bookmark27) [**4-15**](#_bookmark27)

[Simulink Coder 中基于事件的调度](#_bookmark28)

[5](#_bookmark28)

[异步事件](#_bookmark29) [5-2](#_bookmark29)

[异步支持](#_bookmark30) [5-2](#_bookmark30)

[用于调用示例实时操作系统的模块库](#_bookmark31) [**5-2**](#_bookmark31)

[示例和其他信息](#_bookmark32) [5-3](#_bookmark32)

[生成并同步 RTOS 任务的执行](#_bookmark33) [5-4](#_bookmark33)

[Simulink Coder 中的子系统](#_bookmark34)

[6](#_bookmark34)

[**Simulink Coder 中的引用模型**](#_bookmark35)

[7](#_bookmark35)

[Simulink Coder 中的合并模型](#_bookmark36)

[8](#_bookmark36)

[生成代码中的函数重用](#_bookmark37) [8-2](#_bookmark37)

[Simulink Coder 的代码重用](#_bookmark38)

[9](#_bookmark38)

[从顶层模型生成可重入代码](#_bookmark39) [9-2](#_bookmark39)

[生成可重入的多实例代码](#_bookmark40) [**9-2**](#_bookmark40)

[在实例之间共享数据](#_bookmark41) [9-5](#_bookmark41)

[10](#_bookmark42)

[配置运行时环境选项](#_bookmark43) [10-2](#_bookmark43)

[配置生产和测试硬件](#_bookmark44) [10-2](#_bookmark44)

[生产硬件注意事项](#_bookmark53) [10-7](#_bookmark53)

[测试硬件注意事项](#_bookmark54) [10-7](#_bookmark54)

[影响普通模式仿真的生产硬件设置示例](#_bookmark55) [**10-7**](#_bookmark55)

[注册新硬件设备](#_bookmark56) [10-10](#_bookmark56)

[为新设备指定硬件实现](#_bookmark57) [10-10](#_bookmark57)

[指定在各 MATLAB 会话之间持久保留的硬件实现](#_bookmark58) [**10-10**](#_bookmark58)

[通过修改现有实现来创建硬件实现](#_bookmark59) [10-11](#_bookmark59)

[通过重用现有实现来创建硬件实现](#_bookmark60) [10-11](#_bookmark60)

[验证硬件设备数据](#_bookmark61) [10-12](#_bookmark61)

[导出硬件设备数据](#_bookmark62) [10-12](#_bookmark62)

[为目标对象创建替代标识符](#_bookmark63) [10-13](#_bookmark63)

[升级硬件设备的数据定义](#_bookmark64) [10-13](#_bookmark64)

[Simulink Coder 中的模型保护](#_bookmark65)

[11](#_bookmark65)

[保护模型以隐藏内容](#_bookmark66) [11-2](#_bookmark66)

[准备父模型](#_bookmark67) [11-2](#_bookmark67)

[保护引用的模型](#_bookmark68) [11-2](#_bookmark68)

[受保护模型的要求和限制](#_bookmark69) [11-6](#_bookmark69)

[代码生成要求和限制](#_bookmark70) [11-6](#_bookmark70)

[嵌套受保护模型的要求和限制](#_bookmark71) [11-7](#_bookmark71)

[Simulink Coder 中的组件初始化、重置和终止](#_bookmark72)

[12](#_bookmark72)

[**Simulink Coder 中的 Stateflow 模块**](#_bookmark73)

[13](#_bookmark73)

[Simulink Coder 的模块编写和代码生成](#_bookmark74)

[14](#_bookmark74)

[S-Function 和代码生成](#_bookmark75) [14-2](#_bookmark75)

[S-Function 的类型](#_bookmark76) [14-2](#_bookmark76)

[实现非内联和内联 S-Function 需要的文件](#_bookmark77) [**14-3**](#_bookmark77)

[编写支持代码生成的 S-Function 的规范](#_bookmark78) [**14-4**](#_bookmark78)

[代码继承工具和代码生成](#_bookmark80) [14-5](#_bookmark80)

[生成内联 S-Function 文件以进行代码生成](#_bookmark81) [**14-5**](#_bookmark81)

[对已有函数应用代码样式设置](#_bookmark82) [14-6](#_bookmark82)

[解决对不同位置的文件的依存关系](#_bookmark83) [**14-6**](#_bookmark83)

[部署 S-Function 以进行仿真和代码生成](#_bookmark84) [**14-7**](#_bookmark84)

[集成外部 C++ 对象](#_bookmark85) [14-7](#_bookmark85)

[集成以定点数据类型的形参方式传递输入和输出实参的外部 C 函数](#_bookmark86) [14-9](#_bookmark86)

[集成实现 N 维表查找的外部 C 函数](#_bookmark87) [14-12](#_bookmark87)

[集成外部 C++ 对象方法](#_bookmark88) [14-15](#_bookmark88)

[编写非内联 S-Function](#_bookmark89) [14-18](#_bookmark89)

[编写非内联 S-Function 的规范](#_bookmark90) [**14-18**](#_bookmark90)

[非内联 S-Function 的参数类型限制](#_bookmark91) [**14-19**](#_bookmark91)

[编写包装器 S-Function 和 TLC 文件](#_bookmark92) [14-20](#_bookmark92)

[MEX S-Function 包装器](#_bookmark93) [14-20](#_bookmark93)

[TLC S-Function 包装器](#_bookmark94) [14-23](#_bookmark94)

[非内联 S-Function 的代码开销](#_bookmark95) [**14-23**](#_bookmark95)

[内联包装器 S-Function](#_bookmark96) [14-24](#_bookmark96)

[内联代码](#_bookmark97) [14-25](#_bookmark97)

[编写完全内联的 S-Function](#_bookmark98) [14-27](#_bookmark98)

[多端口 S-Function](#_bookmark99) [14-27](#_bookmark99)

[编写内联 S-Function 的规范](#_bookmark100) [**14-27**](#_bookmark100)

[基于消息的通信](#_bookmark101)

[15](#_bookmark101)

[生成用于在 Simulink 组件之间进行数据通信的 C 消息](#_bookmark102) [15-2](#_bookmark102)

[如何在 Simulink 中为基于消息的通信准备模型](#_bookmark103) [**15-2**](#_bookmark103)

[模型组件如何传递消息，以及代码如何实现该行为](#_bookmark104) [**15-2**](#_bookmark104)

[C 代码生成示例](#_bookmark105) [15-2](#_bookmark105)

[注意事项和限制](#_bookmark106) [15-6](#_bookmark106)

[数据、函数和文件定义](#_bookmark107)

[Simulink Coder 中的数据表示](#_bookmark108)

[16](#_bookmark108)

[生成的代码如何存储内部信号、状态和参数数据](#_bookmark109) [16-2](#_bookmark109)

[生成的代码中的内部数据](#_bookmark110) [16-2](#_bookmark110)

[生成的代码中的局部变量](#_bookmark111) [16-17](#_bookmark111)

[生成的代码中工作区变量的外观](#_bookmark113) [16-17](#_bookmark113)

[将内部数据提升到接口](#_bookmark114) [16-18](#_bookmark114)

[控制内部数据的默认表示形式 (Embedded Coder)](#_bookmark115) [**16-19**](#_bookmark115)

[在生成的代码中使用枚举数据](#_bookmark116) [16-21](#_bookmark116)

[枚举数据类型](#_bookmark117) [16-21](#_bookmark117)

[为枚举指定整数数据类型](#_bookmark118) [16-21](#_bookmark118)

[自定义枚举数据类型](#_bookmark119) [16-22](#_bookmark119)

[在生成的代码中控制枚举类型实现](#_bookmark120) [16-25](#_bookmark120)

[枚举的类型转换](#_bookmark121) [16-26](#_bookmark121)

[有关枚举类型的限制](#_bookmark122) [16-27](#_bookmark122)

[在生成的代码中创建可调标定参数](#_bookmark123) [16-28](#_bookmark123)

[将模块参数表示为可调全局变量](#_bookmark124) [16-28](#_bookmark124)

[当模块参数引用 MATLAB 数值变量时应用存储类](#_bookmark125) [**16-30**](#_bookmark125)

[创建表示标定参数的存储类 (Embedded Coder)](#_bookmark126) [**16-31**](#_bookmark126)

[从系统常量或其他宏初始化参数值 (Embedded Coder)](#_bookmark127) [**16-33**](#_bookmark127)

[存储位置对参数对象的代码生成的影响](#_bookmark128) [16-34](#_bookmark128)

[配置信号数据的可访问性](#_bookmark129) [16-35](#_bookmark129)

[用于调整参数的编程接口](#_bookmark130) [16-35](#_bookmark130)

[设置可调参数的最小值和最大值](#_bookmark131) [16-35](#_bookmark131)

[其他建模目的注意事项](#_bookmark132) [16-35](#_bookmark132)

[在仿真和代码执行期间在参数值集之间切换](#_bookmark133) [16-37](#_bookmark133)

[在生成的代码中配置数据和函数](#_bookmark134)

[17](#_bookmark134)

[模型接口元素的 C 代码生成配置](#_bookmark135) [17-2](#_bookmark135)

[迁移](#_bookmark136) [17-2](#_bookmark136)

[代码定义](#_bookmark137) [17-2](#_bookmark137)

[数据默认配置](#_bookmark138) [17-3](#_bookmark138)

[单个数据元素的配置](#_bookmark140) [17-5](#_bookmark140)

[配置数据](#_bookmark141) [17-5](#_bookmark141)

[为 C 代码生成配置参数](#_bookmark142) [17-7](#_bookmark142)

[选择参数的自定义选项](#_bookmark143) [17-7](#_bookmark143)

[为参数配置默认代码生成设置](#_bookmark144) [17-8](#_bookmark144)

[为单个参数配置代码生成](#_bookmark145) [17-10](#_bookmark145)

[以编程方式配置参数的代码生成设置](#_bookmark146) [17-11](#_bookmark146)

[为模型参数选择存储类和存储类属性](#_bookmark147) [17-13](#_bookmark147)

[为模型入口函数配置生成的 C 函数接口](#_bookmark148) [17-15](#_bookmark148)

[什么是入口函数？](#_bookmark149) [17-15](#_bookmark149)

[生成的入口函数的类型](#_bookmark150) [17-15](#_bookmark150)

[配置入口函数是否可重用](#_bookmark151) [17-17](#_bookmark151)

[如何与生成的入口函数对接](#_bookmark152) [17-18](#_bookmark152)

[C 函数接口自定义限制](#_bookmark153) [17-18](#_bookmark153)

[18](#_bookmark154)

[数组布局](#_bookmark155)

[19](#_bookmark155)

[**在生成的代码中配置数据**](#_bookmark134)

[20](#_bookmark134)

[代码生成](#_bookmark156)

[Simulink Coder 的配置](#_bookmark157)

[21](#_bookmark157)

[以编程方式配置模型的代码生成参数](#_bookmark158) [21-2](#_bookmark158)

[修改参数以支持“执行效率”](#_bookmark159) [21-2](#_bookmark159)

[系统目标文件配置](#_bookmark160)

[22](#_bookmark160)

[配置系统目标文件](#_bookmark161) [22-2](#_bookmark161)

[选择支持代码生成的求解器](#_bookmark162) [22-2](#_bookmark162)

[从 STF 浏览器中选择系统目标文件](#_bookmark163) [**22-2**](#_bookmark163)

[以编程方式选择系统目标文件](#_bookmark164) [22-3](#_bookmark164)

[开发自定义系统目标文件](#_bookmark165) [22-4](#_bookmark165)

[比较各产品的系统目标文件支持](#_bookmark166) [22-5](#_bookmark166)

[比较产品系统目标文件](#_bookmark167) [22-6](#_bookmark167)

[比较代码样式和 STF 支持](#_bookmark168) [22-8](#_bookmark168)

[按产品比较生成的代码的功能](#_bookmark169) [22-8](#_bookmark169)

[按 STF 比较生成的代码的功能](#_bookmark170) [22-10](#_bookmark170)

|  |  |  |
| --- | --- | --- |
| [**23**](#_bookmark171) |  |  |
|  | [**国际化和代码生成**](#_bookmark172). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**23-2**](#_bookmark172) |
|  | [区域设置](#_bookmark173) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**23-2**](#_bookmark173) |
|  | [准备生成支持混合语言和区域设置的代码](#_bookmark174) . . . . . . . . . . . . . . . . . . . . . | [**23-2**](#_bookmark174) |
|  | [字符集限制](#_bookmark175) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**23-2**](#_bookmark175) |
|  | [XML 转义序列替换](#_bookmark176) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**23-2**](#_bookmark176) |
|  | [生成和检查混用多种语言和区域设置的代码](#_bookmark177) . . . . . . . . . . . . . . . . . . . . | [**23-2**](#_bookmark177) |

[**Simulink Coder 基础代码生成**](#_bookmark178)

[**24**](#_bookmark178)

[**使用 Simulink Coder 生成代码**](#_bookmark179) [**24-2**](#_bookmark179)

[**飞机位置雷达模型**](#_bookmark180) [**24-8**](#_bookmark180)

[**Simulink Coder 中的源代码生成**](#_bookmark181)

[**25**](#_bookmark181)

[**配置模型、生成代码和仿真**](#_bookmark182) [**25-2**](#_bookmark182)

[关于本示例](#_bookmark183) [**25-2**](#_bookmark183)

[模型的功能设计](#_bookmark184) [**25-2**](#_bookmark184)

[查看顶层模型](#_bookmark185) [**25-2**](#_bookmark185)

[查看子系统](#_bookmark186) [**25-3**](#_bookmark186)

[仿真测试环境](#_bookmark187) [**25-4**](#_bookmark187)

[运行仿真测试](#_bookmark188) [**25-8**](#_bookmark188)

[关键点](#_bookmark189) [**25-9**](#_bookmark189)

[了解更多信息](#_bookmark190) [**25-10**](#_bookmark190)

[**管理编译过程文件夹**](#_bookmark191) [**25-11**](#_bookmark191)

[文件生成控制参数](#_bookmark192) [**25-11**](#_bookmark192)

[编译过程文件夹](#_bookmark193) [**25-12**](#_bookmark193)

[**管理编译过程文件**](#_bookmark194) [**25-15**](#_bookmark194)

[model.bat](#_bookmark195) [**25-18**](#_bookmark195)

[model.h](#_bookmark196) [**25-19**](#_bookmark196)

[rtwtypes.h](#_bookmark197) [**25-19**](#_bookmark197)

[**使用跨版本工作流集成生成的代码**](#_bookmark198) [**25-21**](#_bookmark198)

[**在工程中生成代码并进行模型仿真**](#_bookmark199) [**25-29**](#_bookmark199)

[**使用工程以编程方式生成代码和仿真模型**](#_bookmark200) [**25-32**](#_bookmark200)

[**26**](#_bookmark201)

[**Simulink Coder 中的代码替换**](#_bookmark202)

[**27**](#_bookmark202)

[**什么是代码替换？**](#_bookmark203) [**27-2**](#_bookmark203)

[代码替换库](#_bookmark204) [**27-2**](#_bookmark204)

[代码替换术语](#_bookmark205) [**27-3**](#_bookmark205)

[代码替换限制](#_bookmark206) [**27-4**](#_bookmark206)

[**替换从 Simulink 模型生成的代码**](#_bookmark207) [**27-5**](#_bookmark207)

[**部署**](#_bookmark208)

[**Simulink Coder 中的桌面**](#_bookmark209)

[**28**](#_bookmark209)

[**将 S-Function 目标用于模型或子系统**](#_bookmark210) [**28-2**](#_bookmark210)

[S-Function 部署所需的文件](#_bookmark211) [**28-2**](#_bookmark211)

[生成的 S-Function 中的采样时间传播](#_bookmark212) [**28-2**](#_bookmark212)

[具有生成的 S-Function 的顶层模型的求解器类型](#_bookmark213) [**28-3**](#_bookmark213)

[生成的 S-Function 中的可调参数](#_bookmark214) [**28-3**](#_bookmark214)

[宏参数](#_bookmark215) [**28-4**](#_bookmark215)

[校验和与 S-Function 目标](#_bookmark216) [**28-5**](#_bookmark216)

[生成的 S-Function 的兼容性](#_bookmark217) [**28-5**](#_bookmark217)

[有关 S-Function 目标的限制](#_bookmark218) [**28-5**](#_bookmark218)

[**Simulink Coder 中的实时系统**](#_bookmark236)

[**29**](#_bookmark236)

[**Simulink Coder 中的外部代码集成**](#_bookmark237)

[**30**](#_bookmark237)

[**调用可重用的外部算法代码进行仿真和代码生成**](#_bookmark238) [**30-2**](#_bookmark238)

[工作流](#_bookmark239) [**30-2**](#_bookmark239)

[选择集成方法](#_bookmark240) [**30-3**](#_bookmark240)

[将外部代码插入 Stateflow 图中](#_bookmark241) [**30-7**](#_bookmark241)

[工作流](#_bookmark245) [**30-9**](#_bookmark245)

[为集成的代码编译过程配置参数](#_bookmark246) [**30-9**](#_bookmark246)

[在编译文件夹中保留外部代码文件](#_bookmark247) [**30-10**](#_bookmark247)

[编译对 S-Function 的支持](#_bookmark248) [**30-11**](#_bookmark248)

[**生成要导出到外部代码库的组件源代码**](#_bookmark250) [**30-15**](#_bookmark250)

[建模选项](#_bookmark251) [**30-15**](#_bookmark251)

[要求](#_bookmark252) [**30-15**](#_bookmark252)

[导出函数子系统的限制](#_bookmark253) [**30-16**](#_bookmark253)

[工作流](#_bookmark254) [**30-17**](#_bookmark254)

[选择集成方法](#_bookmark255) [**30-17**](#_bookmark255)

[生成导出函数模型的 C 函数代码](#_bookmark256) [**30-18**](#_bookmark256)

[为导出函数模型生成 C++ 函数和类代码](#_bookmark257) [**30-21**](#_bookmark257)

[为导出函数子系统生成代码](#_bookmark258) [**30-25**](#_bookmark258)

[**通过使用共享库对接在开发计算机上运行的仿真器**](#_bookmark262) [**30-27**](#_bookmark262)

[**Simulink Coder 中的程序编译、交互和调试**](#_bookmark263)

[**31**](#_bookmark263)

[**编译从 Simulink 模型生成的代码的方法**](#_bookmark264) [**31-2**](#_bookmark264)

[**配置工具链 (ToolchainInfo) 或模板联编文件编译过程**](#_bookmark265) [**31-3**](#_bookmark265)

[工具链方法](#_bookmark266) [**31-3**](#_bookmark266)

[升级模型以使用工具链方法](#_bookmark267) [**31-5**](#_bookmark267)

[模板联编文件方法](#_bookmark268) [**31-7**](#_bookmark268)

[指定 TLC 以用于代码生成](#_bookmark270) [**31-11**](#_bookmark270)

[在生成代码的重新编译中使用校验和](#_bookmark271) [**31-11**](#_bookmark271)

[**转移或共享生成的代码**](#_bookmark272) [**31-13**](#_bookmark272)

[打包并转移生成的代码](#_bookmark273) [**31-13**](#_bookmark273)

[从命令行打包并转移生成的代码](#_bookmark274) [**31-14**](#_bookmark274)

[限制](#_bookmark280) [**31-16**](#_bookmark280)

[**使用 Microsoft Visual Studio 编译和调试生成的 C 代码**](#_bookmark281) [**31-17**](#_bookmark281)

[**Simulink Coder 中的主机/目标通信**](#_bookmark282)

[**32**](#_bookmark282)

[**用于参数调节、信号监控和代码执行探查的外部模式仿真**](#_bookmark283) [**32-2**](#_bookmark283)

[外部模式的通信机制](#_bookmark284) [**32-3**](#_bookmark284)

[Simulink 功能对外部模式的支持](#_bookmark285) [**32-3**](#_bookmark285)

[通过使用“在自定义硬件上运行”App 启用外部模式仿真](#_bookmark286) [**32-4**](#_bookmark286)

[外部模式控制面板](#_bookmark287) [**32-5**](#_bookmark287)

[外部模式仿真的安全性](#_bookmark288) [**32-5**](#_bookmark288)

[**使用 TCP/IP 或串行通信的外部模式仿真**](#_bookmark289) [**32-6**](#_bookmark289)

[创建和配置模型](#_bookmark290) [**32-6**](#_bookmark290)

[运行目标应用程序](#_bookmark292) [**32-9**](#_bookmark292)

[调整参数](#_bookmark293) [**32-9**](#_bookmark293)

[停止目标应用程序](#_bookmark294) [**32-11**](#_bookmark294)

[控制目标中通信缓冲区的内存分配](#_bookmark295) [**32-11**](#_bookmark295)

[TCP/IP 或串行外部模式控制面板](#_bookmark296) [**32-11**](#_bookmark296)

[配置目标应用程序信号数据的主机监控](#_bookmark300) [**32-14**](#_bookmark300)

[配置目标应用程序信号数据的主机存档](#_bookmark309) [**32-19**](#_bookmark309)

[TCP/IP 或串行外部模式仿真的图形控制项汇总](#_bookmark310) [**32-21**](#_bookmark310)

[与外部模式兼容的模块和子系统](#_bookmark311) [**32-22**](#_bookmark311)

[用于下载可调参数的外部模式机制](#_bookmark315) [**32-24**](#_bookmark315)

[选择客户端和服务器的通信协议](#_bookmark318) [**32-26**](#_bookmark318)

[以编程方式使用外部模式](#_bookmark327) [**32-31**](#_bookmark327)

[在外部模式下以动画方式显示 Stateflow 图](#_bookmark328) [**32-34**](#_bookmark328)

[TCP/IP 和串行外部模式的限制](#_bookmark331) [**32-36**](#_bookmark331)

[**为 TCP/IP 或串行外部模式通信创建传输层**](#_bookmark332) [**32-39**](#_bookmark332)

[外部模式的设计](#_bookmark333) [**32-39**](#_bookmark333)

[外部模式通信概述](#_bookmark334) [**32-41**](#_bookmark334)

[外部模式源文件](#_bookmark335) [**32-42**](#_bookmark335)

[实现自定义传输层](#_bookmark339) [**32-44**](#_bookmark339)

[**Simulink Coder 中的记录功能**](#_bookmark346)

[**33**](#_bookmark346)

[**使用 Simulink Coder 中的 C API 进行数据交换**](#_bookmark347)

[**34**](#_bookmark347)

[**使用 C API 在生成的代码和外部代码之间交换数据**](#_bookmark348) [**34-2**](#_bookmark348)

[生成的 C API 文件](#_bookmark349) [**34-2**](#_bookmark349)

[生成 C API 文件](#_bookmark350) [**34-4**](#_bookmark350)

[C API 文件的说明](#_bookmark353) [**34-5**](#_bookmark353)

[生成用于与目标系统交换数据的 C API 数据定义文件](#_bookmark363) [**34-15**](#_bookmark363)

[C API 的限制](#_bookmark364) [**34-16**](#_bookmark364)

[**Simulink Coder 中的 ASAP2 数据测量和标定**](#_bookmark365)

[**35**](#_bookmark365)

[**导出 ASAP2 文件用于数据测量和标定**](#_bookmark366) [**35-2**](#_bookmark366)

[基础知识](#_bookmark367) [**35-2**](#_bookmark367)

[支持 ASAP2 的系统目标文件](#_bookmark368) [**35-2**](#_bookmark368)

[定义 ASAP2 信息](#_bookmark369) [**35-2**](#_bookmark369)

[生成 ASAP2 文件](#_bookmark374) [**35-6**](#_bookmark374)

[ASAP2 文件的结构](#_bookmark379) [**35-7**](#_bookmark379)

|  |  |  |
| --- | --- | --- |
| [**36**](#_bookmark380) |  |  |
|  | [**生成 ASAP2 和 CDF 标定文件**](#_bookmark381). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**36-2**](#_bookmark381) |
|  | [打开生成标定文件工具](#_bookmark382) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**36-2**](#_bookmark382) |
|  | [ASAP2 文件生成](#_bookmark383) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**36-3**](#_bookmark383) |
|  | [CDF 文件生成](#_bookmark384) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | [**36-4**](#_bookmark384) |
|  | [为数据测量和标定创建基于主机的 ASAM-ASAP2 数据定义文件](#_bookmark385) . . . . | [**36-4**](#_bookmark385) |

### [对 Simulink Coder 生成的代码进行直接内存访问](#_bookmark386)

[**37**](#_bookmark386)

### [自定义生成的 ASAP2 文件](#_bookmark387)

[**38**](#_bookmark387)

### [为 ASAP2 文件生成配置模型数据元素](#_bookmark388)

[**39**](#_bookmark388)

### [性能](#_bookmark389)

[**在 Simulink Coder 中优化生成的代码**](#_bookmark390)

[**40**](#_bookmark390)

[**生成代码的执行探查**](#_bookmark391) [**40-2**](#_bookmark391)

[**控制 MATLAB Function 模块中可变大小数组的内存分配**](#_bookmark392) [**40-4**](#_bookmark392)

[为可变大小数组提供上界](#_bookmark393) [**40-4**](#_bookmark393)

[为 MATLAB Function 模块禁用动态内存分配](#_bookmark394) [**40-4**](#_bookmark394)

[修改动态内存分配阈值](#_bookmark395) [**40-4**](#_bookmark395)

[**优化时间计数器的内存使用量**](#_bookmark396) [**40-6**](#_bookmark396)

[**41**](#_bookmark397)

[**验证**](#_bookmark398)

[**Simulink Coder 中的仿真和代码比较**](#_bookmark399)

[**42**](#_bookmark399)

[**模型和生成的代码之间的数值一致性**](#_bookmark400)

[**43**](#_bookmark400)

[**自定义**](#_bookmark401)

[**Simulink Coder 中的编译过程集成**](#_bookmark402)

[**44**](#_bookmark402)

[**使用 STF\_make\_rtw\_hook 文件自定义编译过程**](#_bookmark403) [**44-2**](#_bookmark403)

[STF\_make\_rtw\_hook 文件](#_bookmark404) [**44-2**](#_bookmark404)

[STF\_make\_rtw\_hook 文件的使用约定](#_bookmark405) [**44-2**](#_bookmark405)

[STF\_make\_rtw\_hook.m 函数原型和参数](#_bookmark406) [**44-2**](#_bookmark406)

[应用 STF\_make\_rtw\_hook.m](#_bookmark407) [**44-4**](#_bookmark407)

[使用 STF\_make\_rtw\_hook.m 控制代码重新生成](#_bookmark408) [**44-4**](#_bookmark408)

[将 STF\_make\_rtw\_hook.m 用于您的编译过程](#_bookmark409) [**44-5**](#_bookmark409)

[钩子方法 after\_tlc](#_bookmark410) [**44-5**](#_bookmark410)

[**Simulink Coder 中的运行时数据接口扩展**](#_bookmark411)

[**45**](#_bookmark411)

[**Simulink Coder 中的自定义目标开发**](#_bookmark412)

[**46**](#_bookmark412)

[**自定义目标**](#_bookmark413) [**46-2**](#_bookmark413)

[使用系统目标文件控制代码生成](#_bookmark415) [**46-4**](#_bookmark415)

[系统目标文件命名和位置约定](#_bookmark416) [**46-4**](#_bookmark416)

[系统目标文件的结构](#_bookmark417) [**46-4**](#_bookmark417)

[定义和显示自定义目标选项](#_bookmark426) [**46-9**](#_bookmark426)

[自定义系统目标文件的提示和技巧](#_bookmark432) [**46-14**](#_bookmark432)

[创建自定义目标配置](#_bookmark439) [**46-17**](#_bookmark439)

[**设备驱动程序**](#_bookmark448) [**46-26**](#_bookmark448)

**模型架构和设计**

**Simulink Coder 的建模环境**

**在 Simulink Coder 中进行建模**

* [“模块建模规范” （第 2-2 页）](#_bookmark3)
* [“子系统的建模规范” （第 2-3 页）](#_bookmark4)
* [“图建模规范” （第 2-4 页）](#_bookmark5)
* [“MATLAB 函数的建模规范” （第 2-5 页）](#_bookmark6)
* [“模型配置的建模规范” （第 2-6 页）](#_bookmark7)

## 模块建模规范

代码生成建模规范包括推荐的模型设置、模块用法和模块参数。当您开发模型以进行代码生成时，请遵循这些规范。

|  |  |
| --- | --- |
| 代码生成建模规范 | “cgsl\_0101: Zero-based indexing”  “cgsl\_0102: Evenly spaced breakpoints in lookup tables” “cgsl\_0103: Precalculated signals and parameters”  “cgsl\_0201: Redundant Unit Delay and Memory blocks” |

### 另请参阅

[“模型配置的建模规范” （第 2-6 页）](#_bookmark7)

## 子系统的建模规范

当您为子系统开发模型和生成代码时，请使用建模规范建议。

|  |  |
| --- | --- |
| 代码生成建模规范 | “cgsl\_0204: Vector and bus signals crossing into atomic subsystems or Model blocks” |
| 高完整性系统建模规范 | hisl\_0007：For Iterator 或 While Iterator 子系统的用法  “hisl\_0010: Usage of If blocks and If Action Subsystem blocks” “hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks” “hisl\_0023: Verification of variant blocks” |
| MathWorks 咨询委员会 (MAB) 控制算法规范 | db\_0042: Usage of Inport and Outport blocks  db\_0081: Unconnected signals and blocks db\_0143: Usable block types in model hierarchy db\_0144: Use of subsystems  db\_0146: Block layout in conditional subsystems jc\_0201: Usable characters for subsystem names jc\_0231: Usable characters for block names jc\_0281: Trigger signal names  jm\_0002: Block resizing  na\_0008: Display of labels on signals  na\_0009: Entry versus propagation of signal labels |

有关详细信息，请参阅“Model Advisor Checks for High Integrity System Modeling Guidelines” (Simulink Check)和“Using Model Advisor Checks for JMAAB Modeling Guidelines” (Simulink Check)。

## 图建模规范

当您为图开发模型和生成代码时，请使用建模规范建议。

|  |  |
| --- | --- |
| 高完整性系统建模规范 | “hisf\_0001: State Machine Type”  “hisf\_0002: User-specified state/transition execution order” “hisf\_0011: Stateflow debugging settings”  “hisf\_0004: Protect against recursive function calls to improve code compliance”  “hisf\_0013: Usage of transition paths (crossing parallel state boundaries)” “hisf\_0014: Usage of transition paths (passing through states)”  “hisf\_0015: Strong data typing (casting variables and parameters in expressions)” |
| MathWorks 咨询委员会 (MAB) 建模规范 | db\_0127: Limitation on MATLAB commands in Stateflow blocks  jc\_0451: Use of unary minus on unsigned integers  jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow  jc\_0501: Format of entries in a State block jc\_0511: Return values from a graphical function jc\_0531: Default transition  jm\_0011: Pointers in Stateflow  na\_0001: Standard usage of Stateflow operators |

有关详细信息，请参阅“Model Advisor Checks for High Integrity System Modeling Guidelines” (Simulink Check)和“Using Model Advisor Checks for JMAAB Modeling Guidelines” (Simulink Check)。

## MATLAB 函数的建模规范

当您为 MATLAB® 函数开发模型和生成代码时，请使用建模规范建议。

|  |  |
| --- | --- |
| 高完整性系统建模规范 | “himl\_0001: Usage of standardized MATLAB function headers”  “himl\_0002: Strong data typing at MATLAB function boundaries” “himl\_0003: Complexity of user-defined MATLAB Functions” |

有关详细信息，请参阅“Model Advisor Checks for High Integrity System Modeling Guidelines” (Simulink Check)和“Using Model Advisor Checks for JMAAB Modeling Guidelines” (Simulink Check)。

## 模型配置的建模规范

当您开发模型和生成代码时，请使用建模规范配置建议。

|  |  |
| --- | --- |
| 代码生成建模规范 | “cgsl\_0301: Prioritization of code generation objectives for code efficiency” |
| 高完整性系统建模规范 | “hisl\_0043: Configuration Parameters > Diagnostics > Solver”  “hisl\_0044: Configuration Parameters > Diagnostics > Sample Time” “hisl\_0301: Configuration Parameters > Diagnostics > Compatibility”  “hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters”  “hisl\_0303: Configuration Parameters > Diagnostics > Data Validity > Merge blocks”  “hisl\_0304: Configuration Parameters > Diagnostics > Data Validity > Model initialization”  “hisl\_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging”  “hisl\_0306: Configuration Parameters > Diagnostics > Connectivity > Signals”  “hisl\_0307: Configuration Parameters > Diagnostics > Connectivity > Buses”  “hisl\_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls”  “hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion” “hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing” “hisl\_0311: Configuration Parameters > Diagnostics > Stateflow” |

### 另请参阅

[“模块建模规范” （第 2-2 页）](#_bookmark3)

# Simulink Coder 中的计时器

# Simulink Coder 中基于时间的调度

## 基于时间的调度和代码生成

[“采样时间注意事项” （第 4-2 页）](#_bookmark11)

[“任务模式” （第 4-2 页）](#_bookmark12) [“模型执行和速率转换” （第 4-3 页）](#_bookmark13)

[“在 Simulink 模型仿真期间执行” （第 4-3 页）](#_bookmark14) [“模型实时执行” （第 4-4 页）](#_bookmark15) [“单任务与多任务操作” （第 4-4 页）](#_bookmark16)

**本节内容**

### 采样时间注意事项

Simulink 模型按一种或多种采样时间运行。Simulink 产品在构建多速率系统（即具有多种采样时间的系统）方面提供了很高的灵活性。但是，这种灵活性也会使您构造的模型存在以下限制：代码生成器无法为其生成可在多任务环境中执行的实时代码。要使多速率模型按照预期（即给出正确答案）实时运行，有时您必须修改模型或指示 Simulink 引擎修改模型。通常，这些修改需要将 Rate Transition 模块放在具有不相等采样时间的模块之间。下列各节讨论在多任务环境中使用多速率模型必须解决的问题。有关采样时间

（包括速率转换）的全面讨论，请参阅 “什么是采样时间？”、“子系统中的采样时间”、“系统中的采样时间”、“解决速率转移问题” 及相关主题。

### 任务模式

定步长模型有两种执行模式：单任务和多任务。这些模式仅适用于定步长求解器。要选择执行模式，请选中模型配置参数**将每个离散速率视为单独任务**。选中此参数时，将对多速率模型应用多任务执行模式。清除此参数时，将应用单任务执行模式。

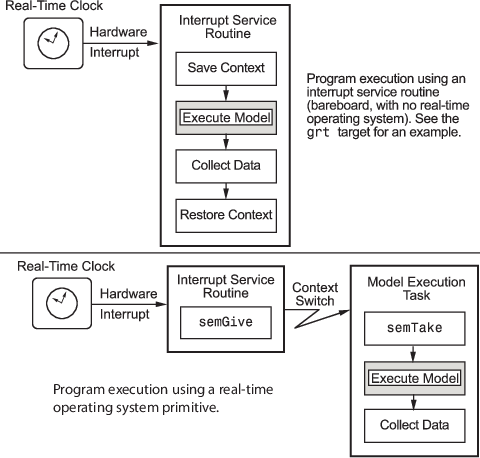
**注意** 使用多任务的多速率模型不能引用使用单任务的多速率模型。

实时系统中的模型执行可以在实时操作系统的帮助下完成，也可以在裸机目标硬件上完成（其中，模型在中断服务例程 (ISR) 上下文中运行）。

系统（如 The Open Group UNIX® 或 Microsoft® Windows® 系统）是多任务型系统并不意味着您的程序可以实时执行。这是因为程序可能不会在需要时抢占其他进程。

在给定时间只能存在一个进程的操作系统（如 PC-DOS）中，中断服务例程 (ISR) 必须执行以下步骤：保存处理器上下文、执行模型代码、收集数据以及还原处理器上下文。

其他操作系统（如符合 POSIX 的操作系统）可提供自动上下文切换和任务调度。这可以简化 ISR 执行的操作。在本例中，ISR 只需启用通常被阻止的模型执行任务即可。下图说明了这种差异。



### 模型执行和速率转换

要生成按预期实时执行的代码，您（或 Simulink 引擎）可能需要识别和处理模型中的采样率转换。在多任务模式下，如果模型包含无效的速率转换，则 Simulink 引擎默认会在仿真期间标记错误。您可以使用模型配置参数**多任务数据传输**来更改此行为。在单任务模式下，参数**单任务数据传输**可用于相同目的。

为避免增加速率转换错误，请在任务之间插入 Rate Transition 模块。您可以通过插入隐藏的 Rate Transition 模块来让 Simulink [引擎自动处理速率转换。有关此选项的说明，请参阅“自动速率转换”](#_bookmark22)

[（第 4-9](#_bookmark22) 页）。

为理解这些问题，首先要考虑 Simulink 仿真与实时程序的差异。

### 在 Simulink 模型仿真期间执行

在 Simulink 引擎对模型进行仿真之前，它会根据各模块的拓扑依存关系对模块进行排序。这包括将虚拟子系统扩展到它们所包含的各个模块中，以及将整个模型扁平化为单个列表。完成此步骤后，每个模块将按顺序执行。

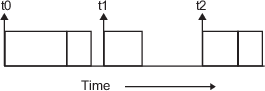
此过程的关键是模块的排序。其输出直接取决于其输入的模块（即具有直接馈通的模块）只有在为其提供输入的驱动模块执行后才会执行。

有些模块基于在前一时间步中获取的值或从指定为模块参数的初始条件中获取的值设置其输出。此类模块的输出由内存中存储的值确定，该值可独立于其输入而更新。在仿真期间，计算是在前进到与时间对应的变量之前执行的。这会导致计算在瞬间发生（即没有计算延迟）。

### 模型实时执行

实时程序与 Simulink 仿真的不同之处在于实时程序必须以实时方式同步执行模型代码。每次计算都会导致一定的计算延迟。这意味着采样间隔不能缩短或延长（因为它们可能在 Simulink 仿真中），从而导致执行效率降低。

以下面的计时图为例。



您会发现采样间隔 **t1** 中的处理效率低下。您不能压缩该间隔来提高执行速度，因为根据定义，采样时间是以实时方式计时的。

您可以使用多任务模式来规避这种潜在的低效率。多任务模式为任务指定不同的优先级，以分别执行具有不同采样率的模型代码部分。

有关多任务模式工作原理的说明，请参阅“Multitasking and Pseudomultitasking Modes”。继续往下阅读之前，请务必理解该节内容。

### 单任务与多任务操作

单任务程序需要较长的采样间隔，因为每个时钟周期内都必须执行所有计算。这可能导致可用 CPU 时间的使用率低下，如上图所示。

如果模型较大且在每个速率下执行多个模块，则多任务模式可以提高程序的效率。

但是，如果您的模型主要采用某一种速率，只有少数模块以较慢的速率执行，则多任务实际上可能会降低性能。在这样的模型中，任务切换所产生的开销可能大于执行较慢模块所需的时间。在这种情况下，以用得最多的速率执行所有模块更为高效。

如果您的模型可从多任务执行中受益，则您可能需要通过添加 Rate Transition 模块来修改模型（或者指示 Simulink 引擎修改模型），以生成预期的结果。

有关这两种执行模式和示例的详细信息，请参阅“Modeling for Single-Tasking Execution”和 “Modeling for Multitasking Execution”。

### 另请参阅详细信息

* “什么是采样时间？”
* “子系统中的采样时间”
* “系统中的采样时间”
* “Configure Time-Based Scheduling”
* “子系统中的采样时间”
* “系统中的采样时间”
* “解决速率转移问题”
* [“处理速率转换” （第 4-6 页）](#_bookmark17)
* “Modeling for Single-Tasking Execution”
* “Modeling for Multitasking Execution”
* “Time-Based Scheduling Example Models”

## 处理速率转换

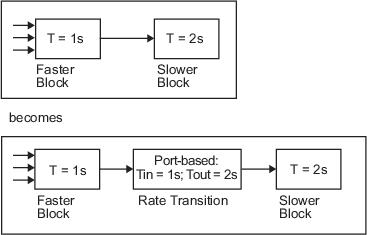
### 速率转换

一个模型中可以存在两种周期性采样率转换：

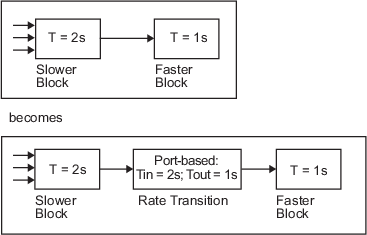
* 较快模块驱动较慢模块
* 较慢模块驱动较快模块

下列各节仅涉及具有零偏移的周期性采样时间的模型。其他考虑因素适用于涉及异步任务的多速率模型。有关如何为异步多任务生成代码的详细信息，请参阅“异步支持” [（第 5-2](#_bookmark30) 页）。

在多任务和伪多任务系统中，不同的采样率可能导致模块以错误的顺序执行。为防止计算的数据中出现错误，您必须控制在发生这些转换时的模型执行。连接较快和较慢模块时，您或 Simulink 引擎必须在它们之间添加 Rate Transition 模块。“快到慢”转换如下图所示。



“慢到快”转换如下图所示。



**注意** 尽管 Rate Transition 模块提供的功能是 Unit Delay 模块（用于慢到快转换）和 Zero-Order Hold模块（用于快到慢转换）的超集，但您还是应该使用 Rate Transition 模块而不是这两个模块。

### 数据传输问题

Rate Transition 模块处理与以不同速率运行的各模块之间的数据传输相关联的数据完整性和确定性问题。

* 数据完整性：当模块的输入在该模块执行期间发生更改时，就会出现数据完整性问题。数据完整性问题可能由抢占引起。

以下列情形为例：

* + 一个较快模块向一个较慢模块提供输入。
  + 较慢模块从较快模块读取输入值 V1，并使用该值开始计算。
  + 然而，较快模块的另一次执行（该执行用来计算新输出值 V2）抢占了上述计算。
  + 这就会出现数据完整性问题：当较慢模块恢复执行时，它将继续其计算，但现在使用的是“新”输入值 V2。

这样的数据传输被称为不受保护的传输。“实时模式下较快到较慢的转换” [（第 4-11](#_bookmark25) 页）显示了一个不受保护的数据传输。

在受保护的数据传输中，会保存较快模块的输出 V1，直到较慢模块完成执行。

* 确定性与非确定性数据传输：在确定性数据传输中，数据传输的时间是完全可预测的，因为这是由模块的采样率决定的。

非确定性数据传输的时间取决于数据的可用性、模块的采样率以及接收模块相对于驱动模块开始执行的时间。

您可以使用 Rate Transition 模块来保护应用程序中的数据传输并使其具有确定性。在大多数应用中，这些特性都被认为是令人满意的。但是，Rate Transition 模块支持还支持一些灵活的选项，您可以为了降低延迟而适当牺牲数据完整性和确定性。下一节概述了这些选项。

### 数据传输假设

在处理任务之间的数据传输时，代码生成器将作以下假设：

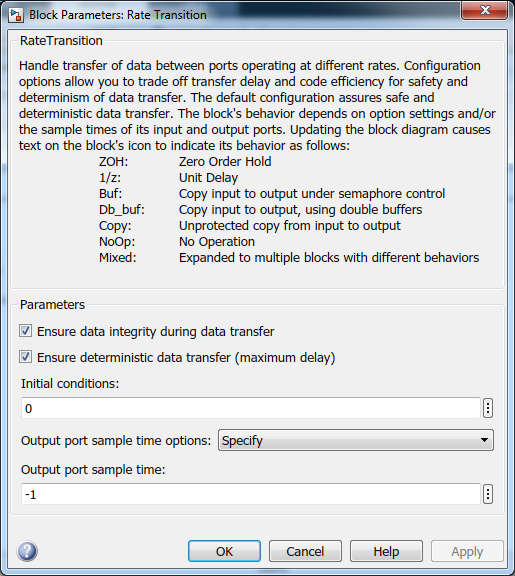
* 在单个读取任务和单个写入任务之间发生数据转移。
* 以字节为单位的变量的读取或写入是原子级别的。
* 当两个任务通过数据转移进行交互时，只有其中一个任务可以抢占另一个任务。
* 对于周期性任务，速率较快的任务比速率较慢的任务具有更高优先级；速率较快的任务会抢占速率较慢的任务。
* 所有任务都在单个处理器上运行。不允许分时。
* 进程不会崩溃或重新启动（特别是在任务之间传输数据时）。

### Rate Transition 模块选项

Rate Transition 模块有几个参数与它在代码生成中用于实时执行时相关，如下所述。有关完整的模块描述，请参阅 Rate Transition。

Rate Transition 模块处理周期性转换（快速到慢速和慢速到快速）和异步转换。在具有不同采样率的两个模块之间插入 Rate Transition 模块时，该模块会针对转换类型自动配置其输入和输出采样率；您不需要指定转换是慢速到快速还是快速到慢速（对于异步任务，则是低到高还是高到低优先级）。

在配置 Rate Transition 模块时，您必须做出的关键决策是选择要在两种速率之间使用哪种数据传输机制。进行选择时，您需要综合考虑安全性、内存使用量和性能。如下图中的 Rate Transition 模块参数对话框所示，数据传输机制由两个选项控制。



* **确保数据传输的数据完整性**：选中此参数时，数据在不同速率之间传输时仍能保持自身的完整性（数据传输受保护）。清除此参数时，数据可能无法保持其完整性（数据传输不受保护）。默认情况下，参数**确保数据传输的数据完整性**处于选中状态。
* **确保确定性数据传输(最大延迟)**：对于零偏移或快速率是慢速率倍数的周期性任务，支持此参数。如果选中此参数，Rate Transition 模块的行为类似于 Zero-Order Hold 模块（适用于快速到慢速转换）或 Unit Delay 模块（适用于慢速到快速转换）。Rate Transition 模块以完全可预测的方式控制数据传输的时间。如果清除此参数，则数据传输是非确定性的。默认情况下，对于偏移为零的周期性速率之间的转换，**确保确定性数据传输(最大延迟)** 处于选中状态；而对于异步转换，则不能选择此选项。

Rate Transition 模块提供了三种关于数据传输的操作模式。按照安全级别顺序：

* **受保护/确定性(默认)**：这是最安全的模式。此模式的缺点是：在慢速到快速的周期性速率转换情况下，它将为系统带来确定性延迟。对于这种情况，Rate Transition 模块带来的延迟是较慢任务的一个采样周期。对于快速到慢速的周期性速率转换情况，Rate Transition 模块不会带来额外的延迟。
* **受保护/非确定性**：在此模式下，对于慢速到快速的周期性速率转换，数据完整性由在不同速率之间传输的双缓冲数据机制来保护。对于快速到慢速的周期性速率转换，使用信号量标志。Rate Transition模块的下游模块使用来自 Rate Transition 模块的驱动模块的最新可用数据。最大延迟小于或等于较快任务的一个采样周期。

此模式的缺点是它具有非确定性时间。此模式的优点是低延迟。

* **未受保护/非确定性**：对于任务关键型应用程序，不推荐使用此模式。此模式的延迟与受保护/非确定性模式相同，但内存要求降低，因为不需要双缓冲和信号量。也就是说，在此模式下，Rate Transition模块仅传递信号；它的作用只是通知您存在速率转换（并且可能导致生成的代码计算出不正确的答

案）。但是，选择此模式产生的代码量最少。

**注意** 在不受保护的模式（**确保数据传输的数据完整性**处于选中状态）下，Rate Transition 模块的作用只是允许模型中存在速率转换。

#### Rate Transition 模块和连续时间

Rate Transition 模块的输出端口的采样时间在子时间步内只能是离散或固定采样时间。这意味着当 Rate Transition 模块从其目标模块继承连续采样时间时，它会将继承的采样时间视为在子时间步中是固定的。因此，Rate Transition 模块的输出函数仅在主时间步运行。如果目标模块采样时间是连续的，则 Rate Transition 模块的输出采样时间是基本速率采样时间（如果求解器采用定步长）或零阶保持连续采样时间

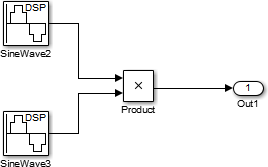
（如果求解器采用变步长）。

### 自动速率转换

在图更新过程中，Simulink 引擎可检测到多任务模型中不匹配的速率转换，并自动插入 Rate Transition模块来处理此问题。要启用此功能，请选中模型配置参数**自动处理数据传输的速率转换**。默认情况下，此参数处于清除状态。选中此参数时：

* Simulink 将处理周期性采样时间和异步任务之间的速率转移。
* Simulink 将在模块图中插入隐藏的 Rate Transition 模块。
* 代码生成器为自动插入的 Rate Transition 模块生成代码。此代码与针对手动插入的 Rate Transition模块生成的代码相同。
* 对于周期性任务和异步任务，自动插入的 Rate Transition 模块在受保护模式下运行。您无法更改此行为。对于周期性任务，自动插入的 Rate Transition 模块按照模型配置参数**确定性数据传输**指定的确定性级别运行。默认设置为“尽可能”，此设置将根据一个整数倍数来确定相关周期性采样时间之间的数据传输。有关详细信息，请参阅 Deterministic data transfer。要使用其他模式，您必须手动插入 Rate Transition 模块并设置其模式。

例如，在此模型中，SineWave2 的采样时间为 2，SineWave3 的采样时间为 3。

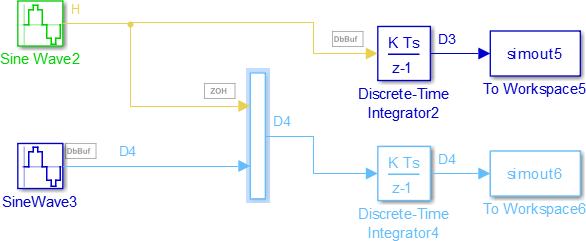


如果您选择模型配置参数**自动处理数据传输的速率转换**，Simulink 将在每个 Sine Wave 模块和 Product模块之间插入一个 Rate Transition 模块。插入的模块具有参数值以协调 Sine Wave 模块采样时间。

如果模型中的输入端口和输出端口数据采样率不构成倍数关系，则 Simulink 会插入一个 Rate Transition模块，其采样率是两个速率的最大公约数 (GCD)。如果该模型中没有其他模块包含此新速率，则在仿真期间会发生错误。在这种情况下，必须手动插入 Rate Transition 模块。

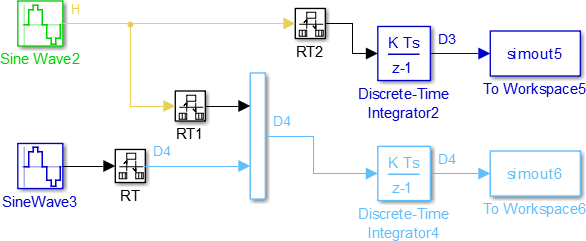
### 可视化插入的 Rate Transition 模块

如果您选择了模型配置参数**自动处理数据传输的速率转换**，Simulink 将在转移速率不匹配的路径中插入 Rate Transition 模块。这些模块默认情况下处于隐藏状态。要可视化插入的模块，请更新图。模型中将出现标记，指出 Simulink 在编译阶段插入 Rate Transition 模块的位置。例如，在此模型中，编译模型时在两个 Sine Wave 模块与 Multiplexer 和 Integrator 之间插入了三个 Rate Transition 模块。ZOH 和 DbBuf 标记指示了这些模块。



您可以显示或隐藏标记。打开**调试**选项卡。在**叠加信息/采样时间**库的**诊断**部分中，选中或清除**自动速率转换**。

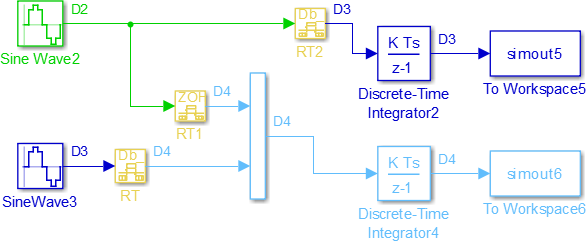
要配置隐藏的 Rate Transition 模块以使这些模块可见，请右键点击标记，然后点击**插入 Rate Transition模块**。



当您显示隐藏的 Rate Transition 模块之后：

* 您可以看到插入的 Rate Transition 模块的类型以及该模块在模型中的位置。
* 您可以设置模块参数**初始条件**。
* 您可以更改有关速率转换的模块参数设置。

通过更新图来验证您对模型所做的更改。



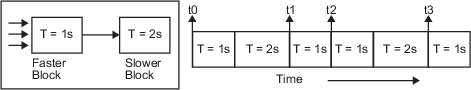
Displaying inserted Rate Transition blocks is not compatible with export-function models.有关 Rate Transition 模块类型的详细信息，请参阅 Rate Transition。

### 周期性采样率转换

下列各节描述周期性采样率转换需要使用 Rate Transition 模块的情况。这些部分中的讨论和时序图基于以下假设：Rate Transition 模块在其默认（受保护/确定性）模式下使用。模型配置参数**确保数据传输的数据完整性**和**确保确定性数据传输(最大延迟)** 处于选中状态。这些设置用于自动插入的 Rate Transition 模 块。

#### Simulink 模型中较快到较慢的转换

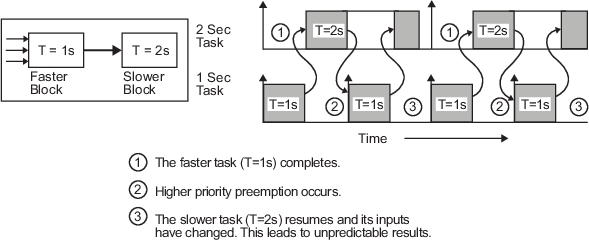
在较快模块驱动具有直接馈通的较慢模块的模型中，首先计算较快模块的输出。在较慢模块不执行的仿真间隔中，仿真进度较快，因为要执行的模块较少。下图说明了这种情况。



Simulink 仿真不实时执行，这意味着它不受实时约束的限制。仿真会等待或前进到任何完成仿真流所需的任务。采样时间步之间的实际时间间隔可能有所不同。

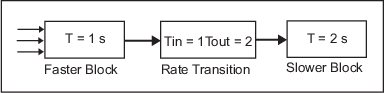
#### 实时模式下较快到较慢的转换

在较快模块驱动较慢模块的模型中，由于较慢模块的执行时间可能会是执行多个较快模块所需的时间，您必须对此予以补偿。这意味着较快模块的输出可能会在较慢模块完成计算其输出之前发生变化。下图显示了出现此问题的情况（T = 采样时间）。请注意，较低优先级的任务在完成之前被较高优先级的任务抢 占。



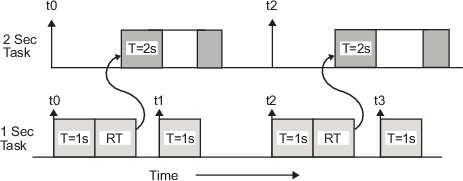
在上图中，较快模块在较慢模块完成执行之前又执行了一次。这可能会导致不可预测的结果，因为慢任务的输入数据发生了更改。在这种情况下，数据可能无法保持其完整性。

为避免这种情况，Simulink 引擎必须保持 1 秒（较快）模块的输出，直到 2 秒（较慢）模块完成执行。为此，需要在 1 秒模块和 2 秒模块之间插入一个 Rate Transition 模块。较慢模块的输入在模块执行过程中不会发生变化，从而保持了数据的完整性。



假设 Rate Transition 模块在其默认（受保护/确定性）模式下使用。

Rate Transition 模块以较慢模块的采样率执行，但以较快模块的优先级执行。

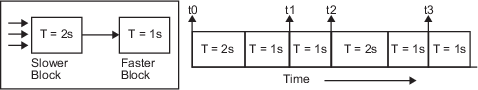


如果您添加 Rate Transition 模块，该模块将在 2 秒模块之前执行（其优先级较高），其输出值在 2 秒模块执行（以较慢采样率执行）时将保持不变。

#### Simulink 模型中较慢到较快的转换

在较慢模块驱动较快模块的模型中，Simulink 引擎会再次首先计算驱动模块的输出。在只有较快模块执行的采样间隔期间，仿真进度较快。

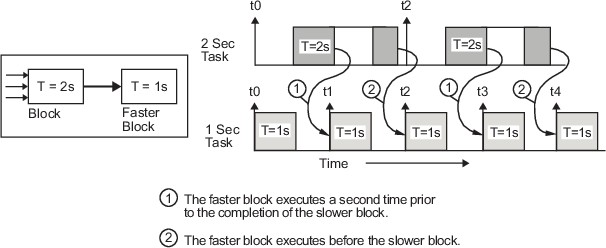
下图显示了执行顺序。



从上图可以看出，Simulink 引擎可以有效地对具有多个采样率的模型进行仿真。但是，Simulink 仿真不能实时运行。

#### 以实时方式进行较慢到较快的转换

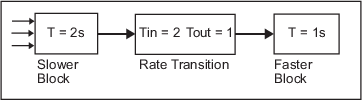
在较慢模块驱动较快模块的模型中，生成的代码为较快模块指定的优先级高于较慢模块。这意味着较快模块在较慢模块之前执行，此时需要特别注意避免不正确的结果。



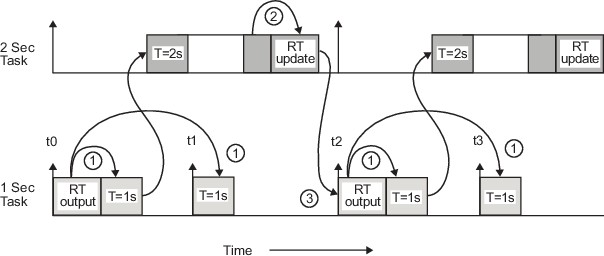
此时序图说明了两个问题：

* 较慢模块的执行时间跨越较快模块的多个执行时间间隔。在这种情况下，较快任务在较慢任务完成执行之前又执行了一次。这意味着较快任务的输入在某段时间可能包含不正确的值。
* 较快模块在较慢模块之前执行（与 Simulink 仿真操作方向相反）。在这种情况下，1 秒模块会先执行；但是较快任务的输入尚未计算出来。这可能导致不可预测的结果。

为消除这些问题，必须在较慢模块和较快模块之间插入一个 Rate Transition 模块。



假设 Rate Transition 模块在其默认（受保护/确定性）模式下使用。下图显示添加 Rate Transition 模块后产生的时序。



此图中关于转换的三个关键点（请参见带圆圈的数字）：

1. Rate Transition 模块输出在 1 秒任务中运行，但以慢速（2 秒）运行。Rate Transition 模块的输出为 1 秒任务模块馈送数据。
2. Rate Transition 更新使用 2 秒任务的输出来更新其内部状态。
3. 1 秒任务中的 Rate Transition 输出使用已在 2 秒任务中更新过的 Rate Transition 的状态。

第一个问题得到缓解，因为 Rate Transition 模块以较慢速率和较慢模块的优先级进行更新。在较慢模块完成执行后，将读取 Rate Transition 模块的输入（它是较慢模块的输出）。

第二个问题得到缓解，因为 Rate Transition 模块以较慢速率执行，并且其输出在它所驱动的较快模块进行计算期间不会发生变化。Rate Transition 模块的输出部分以较慢模块的采样率执行，但以较快模块的优先级执行。由于 Rate Transition 模块驱动较快模块并且具有相同的优先级，因此它在较快模块之前执行。

**注意** Rate Transition 模块的使用改变了模型。与不使用 Rate Transition 模块的输出相比，较慢模块的输出现在延迟了一个时间步。

### 使用 volatile 关键字保护数据完整性

当您选择模型配置参数**确保数据传输的数据完整性**时，为 Rate Transition 模块生成的代码将定义全局缓冲区和信号量，并使用它们来保护传输数据的完整性。

特别是对于多任务应用程序，数据传输中涉及的任务（速率）可能会在编译器无法预料的时间写入传输的数据、缓冲区和信号量。为了防止编译器以损害传输数据完整性的方式优化程序集代码，代码生成器将关键字 **volatile** 应用于缓冲区和信号量。代码生成器不会将 **volatile** 应用于表示传输数据的全局变量，因为 **volatile** 缓冲区和信号量通常会提供足够的保护。

使用 Embedded Coder®，您可以通过将内置自定义存储类 **volatile** 应用于 Rate Transition 模块的输 入，将 **Volatile** 显式应用于传输的数据。例如，您可以使用此方法来帮助保护外部代码与生成的代码共享的数据的完整性。

或者，为保护外部代码与生成的代码共享的数据，您可以编写自己的 C 函数，以受保护的方式读写数据。然后，您可以将自定义存储类 **GetSet** 应用于模型中的数据，这会使生成的代码调用您的函数而不是直接访问数据。

有关应用 **volatile** 的详细信息，请参阅“Protect Global Data with const and volatile Type Qualifiers” (Embedded Coder)。有关 **GetSet** 的详细信息，请参阅“Access Data Through Functions with Storage Class GetSet” (Embedded Coder)。

### 从算法代码和数据中分离 Rate Transition 模块代码和数据

您可以指定代码生成器是将它为 Rate Transition 模块生成的代码和数据内联到模型代码中，还是将代码和数据放入模型代码调用的单独的函数中。您可以通过选择 **Rate Transition 模块代码**参数对此进行控制。通过将 Rate Transition 模块代码及数据与算法代码及数据分离，您能够对 Rate Transition 模块代码和算法代码独立地进行分析、优化和测试。默认情况下，Rate Transition 模块代码设置为内联到算法代码和数据中。您可以将代码和数据分开，以便生成的代码包含 **model\_step** 函数调用的单独 **get** 和 **set** 函数以及专用的状态数据结构体。生成的代码还包含 **model\_initialize** 函数调用的单独 **start** 和 **initialize** 函

数。

#### 示例模型

打开示例模型 **rtwdemo\_ratetrans**。这种多速率、多任务模型包含几个在不同模式下运行的 Rate Transition 模块。

**open\_system('rtwdemo\_ratetrans'); set\_param('rtwdemo\_ratetrans','SystemTargetFile','ert.tlc'); set\_param('rtwdemo\_ratetrans','GenerateComments', 'Off');**

#### 将 Rate Transition 模块的代码分离出来

在“配置参数”对话框中，**Rate Transition 模块代码**参数设置为 **Function**。为模型生成代码。代码位于文件 **rtwdemo\_ratetrans.c** 和 **rtwdemo\_ratetrans.h** 中。

**currentDir = pwd; [~,cgDir] = rtwdemodir();**

**slbuild('rtwdemo\_ratetrans');**

**### Starting build procedure for: rtwdemo\_ratetrans**

**### Successful completion of build procedure for: rtwdemo\_ratetrans Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_ratetrans Code generated and compiled. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 15.321s**

**hfile=fullfile(cgDir, 'rtwdemo\_ratetrans\_ert\_rtw','rtwdemo\_ratetrans.h'); rtwdemodbtype(hfile,'typedef struct {','} DW;', 1, 1);**

**typedef struct {**

**real\_T OutportBufferForOut3[20]; real\_T Integrator1\_DSTATE[20]; real\_T Integrator2\_DSTATE[20]; real\_T Integrator3\_DSTATE[20]; real\_T Integrator1\_PREV\_U[20]; real\_T Integrator2\_PREV\_U[20];**

**real\_T Integrator3\_PREV\_U[20]; uint32\_T Algorithm\_PREV\_T; struct {**

**uint\_T Algorithm\_RESET\_ELAPS\_T:1;**

**} bitsForTID1;**

**uint8\_T Integrator1\_SYSTEM\_ENABLE; uint8\_T Integrator2\_SYSTEM\_ENABLE; uint8\_T Integrator3\_SYSTEM\_ENABLE;**

**} DW;**

对于 Rate Transition 模块，状态数据不在全局状态结构体 **DW\_rtwdemo\_ratetrans\_T** 中。这些数据在文件 **rtwdemo\_ratetrans\_rtb.h** 内专门的结构体中。

以下代码包含在文件 **rtwdemo\_ratetrans.c** 中。

**cfile=fullfile(cgDir, 'rtwdemo\_ratetrans\_ert\_rtw','rtwdemo\_ratetrans.c');**

**rtwdemodbtype(cfile,'void rtwdemo\_ratetrans\_step0','void rtwdemo\_ratetrans\_terminate(void)', 1, 0);**

**void rtwdemo\_ratetrans\_step0(void)**

**{**

**(rtM->Timing.RateInteraction.TID0\_1)++;**

**if ((rtM->Timing.RateInteraction.TID0\_1) > 1) { rtM->Timing.RateInteraction.TID0\_1 = 0;**

**}**

**rtwdemo\_rate\_DetAndIntegS2F\_get(rtY.Out1); rtwdemo\_ratetr\_IntegOnlyS2F\_get(rtY.Out2);**

**memcpy(&rtY.Out3[0], &rtDW.OutportBufferForOut3[0], 20U \* sizeof(real\_T)); rtwdemo\_rate\_DetAndIntegF2S\_set(rtU.In1); rtwdemo\_ratetr\_IntegOnlyF2S\_set(rtU.In2);**

**}**

**void rtwdemo\_ratetrans\_step1(void)**

**{**

**real\_T rtb\_DetAndIntegF2S[20]; real\_T rtb\_IntegOnlyF2S[20]; real\_T tmp;**

**int32\_T i;**

**uint32\_T Algorithm\_ELAPS\_T; rtwdemo\_rate\_DetAndIntegF2S\_get(rtb\_DetAndIntegF2S); rtwdemo\_ratetr\_IntegOnlyF2S\_get(rtb\_IntegOnlyF2S);**

**if (rtDW.bitsForTID1.Algorithm\_RESET\_ELAPS\_T) { Algorithm\_ELAPS\_T = 0U;**

**} else {**

**Algorithm\_ELAPS\_T = rtM->Timing.clockTick1 - rtDW.Algorithm\_PREV\_T;**

**}**

**rtDW.Algorithm\_PREV\_T = rtM->Timing.clockTick1; rtDW.bitsForTID1.Algorithm\_RESET\_ELAPS\_T = false; tmp = 0.001 \* (real\_T)Algorithm\_ELAPS\_T;**

**for (i = 0; i < 20; i++) {**

**if (rtDW.Integrator1\_SYSTEM\_ENABLE == 0) { rtDW.Integrator1\_DSTATE[i] += tmp \* rtDW.Integrator1\_PREV\_U[i];**

**}**

**if (rtDW.Integrator2\_SYSTEM\_ENABLE == 0) { rtDW.Integrator2\_DSTATE[i] += tmp \* rtDW.Integrator2\_PREV\_U[i];**

**}**

**if (rtDW.Integrator3\_SYSTEM\_ENABLE == 0) { rtDW.Integrator3\_DSTATE[i] += tmp \* rtDW.Integrator3\_PREV\_U[i];**

**}**

**rtDW.OutportBufferForOut3[i] = rtDW.Integrator3\_DSTATE[i]; rtDW.Integrator1\_PREV\_U[i] = rtb\_DetAndIntegF2S[i]; rtDW.Integrator2\_PREV\_U[i] = rtb\_IntegOnlyF2S[i]; rtDW.Integrator3\_PREV\_U[i] = rtU.In3[i];**

**}**

**rtDW.Integrator1\_SYSTEM\_ENABLE = 0U; rtDW.Integrator2\_SYSTEM\_ENABLE = 0U; rtDW.Integrator3\_SYSTEM\_ENABLE = 0U; rtwdemo\_rate\_DetAndIntegS2F\_set(rtDW.Integrator1\_DSTATE); rtwdemo\_ratetr\_IntegOnlyS2F\_set(rtDW.Integrator2\_DSTATE); rtM->Timing.clockTick1++;**

**}**

**void rtwdemo\_ratetrans\_initialize(void)**

**{**

**rtDW.bitsForTID1.Algorithm\_RESET\_ELAPS\_T = true; rtDW.Integrator1\_SYSTEM\_ENABLE = 1U; rtDW.Integrator2\_SYSTEM\_ENABLE = 1U; rtDW.Integrator3\_SYSTEM\_ENABLE = 1U;**

**}**

**rtwdemo\_ratetrans\_step0** 和 **rtwdemo\_ratetrans\_step1** 函数包含对 **get** 和 **set** 函数的调用。这些函数包含 Rate Transition 模块代码。这些函数定义位于文件 **rtwdemo\_ratetrans\_rtb.c** 中。

#### 为 Rate Transition 模块生成内联代码

在“配置参数”对话框中，将 **Rate Transition 模块代码**参数设置为 **Inline**。为模型生成代码。代码位于文件 **rtwdemo\_ratetrans.c** 和 **rtwdemo\_ratetrans.h** 中。

**set\_param('rtwdemo\_ratetrans','RateTransitionBlockCode','Inline'); slbuild('rtwdemo\_ratetrans')**

**### Starting build procedure for: rtwdemo\_ratetrans**

**### Successful completion of build procedure for: rtwdemo\_ratetrans Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**==================================================================================**

**rtwdemo\_ratetrans Code generated and compiled. Generated code was out of date.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 11.781s**

以下代码现在位于文件 **rtwdemo\_ratetrans.h** 中。

**hfile=fullfile(cgDir, 'rtwdemo\_ratetrans\_ert\_rtw','rtwdemo\_ratetrans.h'); rtwdemodbtype(hfile, 'typedef struct {', '} DW;', 1, 1);**

**typedef struct {**

**real\_T Integrator1\_DSTATE[20]; real\_T Integrator2\_DSTATE[20]; real\_T Integrator3\_DSTATE[20]; real\_T DetAndIntegS2F\_Buffer0[20];**

**volatile real\_T IntegOnlyS2F\_Buffer[40]; real\_T DetAndIntegF2S\_Buffer[20]; volatile real\_T IntegOnlyF2S\_Buffer0[20]; real\_T Integrator1\_PREV\_U[20];**

**real\_T Integrator2\_PREV\_U[20]; real\_T Integrator3\_PREV\_U[20]; uint32\_T Algorithm\_PREV\_T; struct {**

**uint\_T Algorithm\_RESET\_ELAPS\_T:1;**

**} bitsForTID1;**

**volatile int8\_T IntegOnlyS2F\_ActiveBufIdx; volatile int8\_T IntegOnlyF2S\_semaphoreTaken; uint8\_T Integrator1\_SYSTEM\_ENABLE; uint8\_T Integrator2\_SYSTEM\_ENABLE; uint8\_T Integrator3\_SYSTEM\_ENABLE;**

**} DW;**

对于 Rate Transition 模块，状态数据不在全局状态结构体 **DW\_rtwdemo\_ratetrans\_T** 中。这些数据在文件 **rtwdemo\_ratetrans\_rtb.h** 内专门的结构体中。

以下代码现在位于文件 **rtwdemo\_ratetrans\_rtb.c** 中。

**cfile=fullfile(cgDir, 'rtwdemo\_ratetrans\_ert\_rtw','rtwdemo\_ratetrans.c');**

**rtwdemodbtype(cfile,'void rtwdemo\_ratetrans\_step0','void rtwdemo\_ratetrans\_terminate(void)', 1, 0);**

**void rtwdemo\_ratetrans\_step0(void)**

**{**

**int32\_T i; int32\_T i\_0;**

**(rtM->Timing.RateInteraction.TID0\_1)++;**

**if ((rtM->Timing.RateInteraction.TID0\_1) > 1) { rtM->Timing.RateInteraction.TID0\_1 = 0;**

**}**

**if (rtM->Timing.RateInteraction.TID0\_1 == 1) {**

**memcpy(&rtY.Out1[0], &rtDW.DetAndIntegS2F\_Buffer0[0], 20U \* sizeof(real\_T));**

**}**

**i = rtDW.IntegOnlyS2F\_ActiveBufIdx \* 20; for (i\_0 = 0; i\_0 < 20; i\_0++) {**

**rtY.Out2[i\_0] = rtDW.IntegOnlyS2F\_Buffer[i\_0 + i];**

**}**

**if (rtM->Timing.RateInteraction.TID0\_1 == 1) { memcpy(&rtDW.DetAndIntegF2S\_Buffer[0], &rtU.In1[0], 20U \* sizeof(real\_T));**

**}**

**if (rtDW.IntegOnlyF2S\_semaphoreTaken == 0) { for (i = 0; i < 20; i++) { rtDW.IntegOnlyF2S\_Buffer0[i] = rtU.In2[i];**

**}**

**}**

**}**

**void rtwdemo\_ratetrans\_step1(void)**

**{**

**real\_T rtb\_IntegOnlyF2S[20]; real\_T tmp;**

**int32\_T i;**

**uint32\_T Algorithm\_ELAPS\_T; rtDW.IntegOnlyF2S\_semaphoreTaken = 1; for (i = 0; i < 20; i++) {**

**rtb\_IntegOnlyF2S[i] = rtDW.IntegOnlyF2S\_Buffer0[i];**

**}**

**rtDW.IntegOnlyF2S\_semaphoreTaken = 0;**

**if (rtDW.bitsForTID1.Algorithm\_RESET\_ELAPS\_T) { Algorithm\_ELAPS\_T = 0U;**

**} else {**

**Algorithm\_ELAPS\_T = rtM->Timing.clockTick1 - rtDW.Algorithm\_PREV\_T;**

**}**

**rtDW.Algorithm\_PREV\_T = rtM->Timing.clockTick1; rtDW.bitsForTID1.Algorithm\_RESET\_ELAPS\_T = false; tmp = 0.001 \* (real\_T)Algorithm\_ELAPS\_T;**

**for (i = 0; i < 20; i++) {**

**if (rtDW.Integrator1\_SYSTEM\_ENABLE == 0) { rtDW.Integrator1\_DSTATE[i] += tmp \* rtDW.Integrator1\_PREV\_U[i];**

**}**

**if (rtDW.Integrator2\_SYSTEM\_ENABLE == 0) { rtDW.Integrator2\_DSTATE[i] += tmp \* rtDW.Integrator2\_PREV\_U[i];**

**}**

**if (rtDW.Integrator3\_SYSTEM\_ENABLE != 0) { rtY.Out3[i] = rtDW.Integrator3\_DSTATE[i];**

**} else {**

**rtY.Out3[i] = tmp \* rtDW.Integrator3\_PREV\_U[i] + rtDW.Integrator3\_DSTATE[i];**

**}**

**rtDW.Integrator1\_PREV\_U[i] = rtDW.DetAndIntegF2S\_Buffer[i]; rtDW.Integrator2\_PREV\_U[i] = rtb\_IntegOnlyF2S[i]; rtDW.Integrator3\_DSTATE[i] = rtY.Out3[i]; rtDW.Integrator3\_PREV\_U[i] = rtU.In3[i]; rtDW.DetAndIntegS2F\_Buffer0[i] = rtDW.Integrator1\_DSTATE[i];**

**}**

**rtDW.Integrator1\_SYSTEM\_ENABLE = 0U; rtDW.Integrator2\_SYSTEM\_ENABLE = 0U; rtDW.Integrator3\_SYSTEM\_ENABLE = 0U; for (i = 0; i < 20; i++) {**

**rtDW.IntegOnlyS2F\_Buffer[i + (rtDW.IntegOnlyS2F\_ActiveBufIdx == 0) \* 20] = rtDW.Integrator2\_DSTATE[i];**

**}**

**rtDW.IntegOnlyS2F\_ActiveBufIdx = (int8\_T)(rtDW.IntegOnlyS2F\_ActiveBufIdx == 0); rtM->Timing.clockTick1++;**

**}**

**void rtwdemo\_ratetrans\_initialize(void)**

**{**

**rtDW.bitsForTID1.Algorithm\_RESET\_ELAPS\_T = true; rtDW.Integrator1\_SYSTEM\_ENABLE = 1U; rtDW.Integrator2\_SYSTEM\_ENABLE = 1U; rtDW.Integrator3\_SYSTEM\_ENABLE = 1U;**

**}**

代码内联在函数 **rtwdemo\_ratetrans\_step0** 和 **rtwdemo\_ratetrans\_step1** 中。

#### 限制

代码生成器不会将那些具有可变大小信号或位于 For Each Subsystem 模块内的 For Rate Transition 模块的代码和数据分离出来。

#### 另请参阅

* Rate Transition block code (Embedded Coder)

**bdclose('rtwdemo\_ratetrans'); rtwdemoclean; cd(currentDir)**

### 另请参阅详细信息

* [“基于时间的调度和代码生成” （第 4-2 页）](#_bookmark10)
* “子系统中的采样时间”
* “系统中的采样时间”
* “Modeling for Multitasking Execution”
* “Configure Time-Based Scheduling”
* “解决速率转移问题”
* “Time-Based Scheduling Example Models”

# Simulink Coder 中基于事件的调度

* [“异步事件” （第 5-2 页）](#_bookmark29)
* [“生成并同步 RTOS 任务的执行” （第 5-4 页）](#_bookmark33)

## 异步事件

### 异步支持

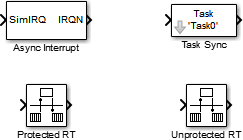
通常，您从周期中断源（例如硬件计时器）对计划从其生成代码的模型进行计时。采用周期性时钟的单速率模型中的模块以计时器中断速率（模型的基本速率）运行。采用周期性时钟多速率的模型中的模块以基本速率或该速率的倍数运行。

许多系统还必须支持模块的执行，以响应与系统的周期计时源有关的异步事件。例如，外围设备可能通过产生中断来发出信号通知输入操作已完成。系统必须处理这种中断，例如，通过从中断设备获取数据来处理。

本主题介绍如何使用模块为异步事件处理建模和生成代码，其中包括处理硬件生成的中断、计时器维护、异步读写操作以及实时操作系统 (RTOS) 下异步任务的生成。此模块库演示与示例 RTOS (VxWorks®) 的集成。虽然这些模块是针对示例 RTOS 的，但本章提供的源代码分析和其他信息可用于开发支持其他目标 RTOS 的异步事件处理的模块。1

### 用于调用示例实时操作系统的模块库

下图显示了 **vxlib1** 模块库中的模块。



库中的关键模块是 Async Interrupt 和 Task Sync 模块。这些模块是针对示例 RTOS (VxWorks) 的。您可以对它们进行修改，以将其用于您的 RTOS 应用程序。

**注意** 使用 Interrupt Templates 模块库中的模块（Async Interrupt 和 Task Sync）进行仿真和代码生

成。您可用使用这些模块提供的示例作为起点，帮助您为目标环境开发自定义模块。

要实现对示例 RTOS 以外的 RTOS 的异步支持，请使用这些规范和 “Create a Customized Asynchronous Library” 中的示例代码来针对您的 RTOS 调整 Interrupt Template 库模块。

Interrupt Template 库包括可用于以下目的的模块：

* 生成中断级代码 - Async Interrupt 模块
* 生成调用函数调用子系统的 RTOS 任务 - Task Sync 模块

1 VxWorks is a registered trademark of Wind River® Systems, Inc.

“Rate Transitions and Asynchronous Blocks”中讨论了在异步环境中使用受保护和不受保护的 Rate Transition 模块。有关速率转换的一般信息，请参阅“基于时间的调度和代码生成” [（第 4-2](#_bookmark10) 页）。

### 示例和其他信息

* 示例模型 **rtwdemo\_async** 使用 **tornado.tlc** 系统目标文件和 Interrupt Template 模块库。要打开模型，请在 MATLAB 命令提示符下键入 **rtwdemo\_async**。
* 示例模型 **rtwdemo\_async\_mdlreftop** 使用 **tornado.tlc** 系统目标文件和 Interrupt Template 模块库。要打开模型，请在 MATLAB 命令提示符下键入 **rtwdemo\_async\_mdlreftop**。
* [“基于时间的调度和代码生成” （第 4-2](#_bookmark10) 页）讨论周期性模型的一般多任务和速率转换问题。
* Embedded Coder 文档讨论了 **ert.tlc** 系统目标文件，包括任务执行和调度。
* 有关对本主题提及的示例 RTOS (VxWorks) 的系统调用的信息，请参阅 Wind River 网站上的 VxWorks 系统文档。

### 另请参阅详细信息

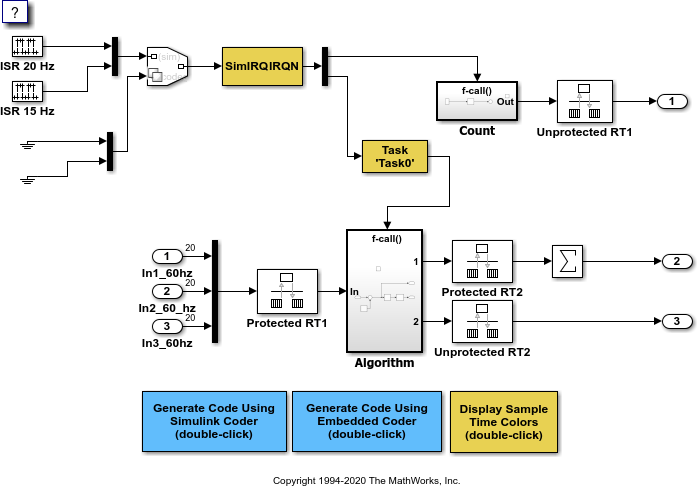
* [“基于时间的调度和代码生成” （第 4-2 页）](#_bookmark10)
* “Generate Interrupt Service Routines”
* [“生成并同步 RTOS 任务的执行” （第 5-4 页）](#_bookmark33)
* “Pass Asynchronous Events in RTOS as Input to a Referenced Model”
* “Timers in Asynchronous Tasks”
* “Import Asynchronous Event Data for Simulation”
* “Rate Transitions and Asynchronous Blocks”
* “Create a Customized Asynchronous Library”
* “Asynchronous Support Limitations”

## 生成并同步 RTOS 任务的执行

此示例说明如何在实时多任务系统上仿真和生成异步事件的代码。该模型包含 Async Interrupt 模块、 Task Sync 模块、异步执行子系统 **Count** 和 **Algorithm** 以及 Rate Transition 模块。Async Interrupt 模块会创建两个 Versa Module Eurocard (VME) 中断服务例程 (ISR)，它们将中断信号传递给子系统 **Count** 和 Task Sync 模块。**Count** 在中断级别执行。**Algorithm** 作为异步任务执行。代码生成器为子系统生成的代码是针对 VxWorks® 操作系统的。

#### 关于示例模型

打开示例模型 **rtwdemo\_async**。



您可以在仿真的中断源与下列模型元素之一之间放置一个 Async Interrupt 模块：

* 函数调用子系统
* Task Sync 模块
* 为函数调用输入事件配置的 Stateflow® 图
* 具有连接到前述模型元素之一的 Inport 模块的引用模型

Async Interrupt 和 Task Sync 模块使子系统能够异步执行。重新配置 Async Interrupt 和 Task Sync 模块，以便为另一个运行时环境生成代码。

**Count** 表示在中断级别执行的一个简单中断服务例程 (ISR)。最好让 ISR 尽可能简单。此子系统仅包含一个 Discrete-Time Integrator 模块。

**Algorithm** 包含更多内容。它包含多个模块并产生两个输出值。在中断级别执行较大的子系统会对系统中同等优先级和更低优先级中断的响应时间产生重大影响。对于较大的子系统，更好的解决方案是使用 Task Sync 模块来表示函数调用子系统的 ISR。

Async Interrupt 模块生成对 ISR 的调用。将该模块放在仿真的中断源与下列项之一之间：

* 函数调用子系统
* Task Sync 模块
* 为函数调用输入事件配置的 Stateflow 图

对于每个指定的中断级别，该模块生成一个 Versa Module Eurocard (VME) ISR，用于执行连接的子系统、Task Sync 模块或图。

在示例模型中，通过使用中断向量偏移 192 和 193，针对 VME Interrupt 1 和 2 配置 Async Interrupt 模块。Interrupt 1 直接连接到子系统 **Count**。Interrupt 2 连接到 Task Sync 模块，该模块充当 **Algorithm** 的 ISR。将 Task Sync 模块放在以下位置之一：

* Async Interrupt 模块与函数调用子系统或 Stateflow 图之间。
* 具有事件 **Output to Simulink**（该事件被配置为函数调用）的 Stateflow 图的输出端口上。

在示例模型中，Task Sync 模块位于 Async Interrupt 模块与函数调用子系统 **Algorithm** 之间。Task Sync 模块配置为使用任务名称 **Task()**，优先级为 50，堆栈大小为 8192，任务的数据传输与调用方任务同步。生成的任务使用信号量来同步任务执行。Async Interrupt 模块会触发任务信号量的释放。

四个 Rate Transition 模块会处理以不同速率工作的端口之间的数据传输。在两个实例中，受保护 Rate Transition 模块用于保护数据传输（防止它们被抢占和损坏）。在另外两个实例中，非受保护 Rate Transition 模块不会引入特殊行为。它们用于通知 Simulink® 发生速率转换。

为 Async Interrupt 和 Task Sync 模块生成的代码是专门用于示例 RTOS (VxWorks) 的。但是，您可以修改这些模块来生成特定于您的运行时环境的代码。

#### 数据传输假设

* 数据传输发生在一个读取任务与一个写入任务之间。
* 对以字节为单位的变量的读取或写入操作是原子级别的。
* 当两个任务交互时，只有其中一个任务可以抢占另一个任务。
* 对于周期性任务，速率较快的任务优先于速率较慢的任务。速率较快的任务可以抢占速率较慢的任务。
* 这些任务在单个处理器上运行。不允许分时。
* 进程不会停止和重启，尤其是在各任务间传输数据时。

#### 对模型进行仿真

对模型进行仿真。默认情况下，模型配置为以不同颜色显示采样时间。输入和输出的离散采样时间分别以红色和绿色显示。常量以红蓝色显示。异步中断和任务以紫色显示。具有混合速率的 Rate Transition 模块

（其输入和输出采样时间可以不同）以黄色显示。

#### 生成代码和报告

为模型生成代码和代码生成报告。为 Async Interrupt 和 Task Sync 模块生成的代码是针对示例 RTOS (VxWorks) 生成的。但是，您可以修改这些模块来为另一个运行时环境生成代码。

1.为编译和检查过程创建一个临时文件夹。

2.编译模型。

**### Starting build procedure for: rtwdemo\_async**

**Warning: Simulink Coder: The tornado.tlc target will be removed in a future release.**

**### Successful completion of code generation for: rtwdemo\_async Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**==================================================================================**

**rtwdemo\_async Code generated. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 16.982s**

#### 查看初始化代码

打开生成的源文件 **rtwdemo\_async.c**。初始化代码： 1.创建并初始化同步信号量 **Task0\_semaphore**。

**\*(SEM\_ID \*)rtwdemo\_async\_DW.SFunction\_PWORK.SemID = semBCreate(SEM\_Q\_PRIORITY, SEM\_EMPTY);**

**if (rtwdemo\_async\_DW.SFunction\_PWORK.SemID == NULL) { printf("semBCreate call failed for block Task0.\n");**

**}**

1. 生成任务 **task0** 并对任务分配优先级 50。

**rtwdemo\_async\_DW.SFunction\_IWORK.TaskID = taskSpawn("Task0", 50.0,**

**VX\_FP\_TASK, 8192.0, (FUNCPTR)Task0,**

**0, 0, 0, 0, 0, 0, 0,0, 0, 0);**

**if (rtwdemo\_async\_DW.SFunction\_IWORK.TaskID == ERROR) { printf("taskSpawn call failed for block Task0.\n");**

**}**

**/\* End of Start for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

**/\* Connect and enable ISR function: isr\_num1\_vec192 \*/**

**if (intConnect(INUM\_TO\_IVEC(192), isr\_num1\_vec192, 0) != OK) { printf("intConnect failed for ISR 1.\n");**

**}**

**sysIntEnable(1);**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

**/\* Connect and enable ISR function: isr\_num2\_vec193 \*/**

**if (intConnect(INUM\_TO\_IVEC(193), isr\_num2\_vec193, 0) != OK) { printf("intConnect failed for ISR 2.\n");**

**}**

**sysIntEnable(2);**

1. 为 Interrupt 1 连接并启用 ISR **isr\_num1\_vec192**，为 Interrupt 2 连接并启用 ISR

#### isr\_num2\_vec193。

**{**

**int32\_T i;**

**/\* InitializeConditions for RateTransition: '<Root>/Protected RT1' \*/ for (i = 0; i < 60; i++) {**

**rtwdemo\_async\_DW.ProtectedRT1\_Buffer[i] = 0.0;**

**}**

**/\* End of InitializeConditions for RateTransition: '<Root>/Protected RT1' \*/**

**/\* InitializeConditions for RateTransition: '<Root>/Protected RT2' \*/ for (i = 0; i < 60; i++) {**

**rtwdemo\_async\_DW.ProtectedRT2\_Buffer[i] = 0.0;**

**}**

**/\* End of InitializeConditions for RateTransition: '<Root>/Protected RT2' \*/**

**/\* SystemInitialize for S-Function (vxinterrupt1): '<Root>/Async Interrupt' incorporates:**

**\* SubSystem: '<Root>/Count'**

**\*/**

**/\* System initialize for function-call system: '<Root>/Count' \*/**

**/\* InitializeConditions for DiscreteIntegrator: '<S2>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_DSTATE\_l = 0.0;**

**rtwdemo\_async\_DW.Integrator\_PREV\_U\_o = 0.0;**

**/\* SystemInitialize for Outport: '<Root>/Out1' incorporates:**

**\* Outport: '<S2>/Out'**

**\*/**

**rtwdemo\_async\_Y.Out1 = 0.0;**

**/\* SystemInitialize for S-Function (vxinterrupt1): '<Root>/Async Interrupt' incorporates:**

**\* SubSystem: '<S3>/Subsystem'**

**\*/**

**/\* System initialize for function-call system: '<S3>/Subsystem' \*/**

**/\* SystemInitialize for S-Function (vxtask1): '<S4>/S-Function' incorporates:**

**\* SubSystem: '<Root>/Algorithm'**

**\*/**

**/\* System initialize for function-call system: '<Root>/Algorithm' \*/ rtwdemo\_async\_M->Timing.clockTick2 = rtwdemo\_async\_M->Timing.clockTick4;**

**/\* InitializeConditions for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_DSTATE = 0.0;**

**/\* InitializeConditions for Sum: '<S1>/Sum1' incorporates:**

**\* DiscreteIntegrator: '<S1>/Integrator'**

**\*/**

**rtwdemo\_async\_DW.Integrator\_PREV\_U = 0.0;**

**/\* SystemInitialize for Sum: '<S1>/Sum' incorporates:**

**\* Outport: '<S1>/Out1'**

**\*/**

**memset(&rtwdemo\_async\_B.Sum[0], 0, 60U \* sizeof(real\_T));**

**/\* SystemInitialize for Outport: '<Root>/Out3' incorporates:**

**\* Outport: '<S1>/Out2'**

**\*/**

**rtwdemo\_async\_Y.Out3 = 0.0;**

**/\* End of SystemInitialize for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* End of SystemInitialize for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

**/\* Enable for S-Function (vxinterrupt1): '<Root>/Async Interrupt' incorporates:**

**\* SubSystem: '<Root>/Count'**

**\*/**

**/\* Enable for function-call system: '<Root>/Count' \*/ rtwdemo\_async\_DW.Count\_RESET\_ELAPS\_T = true;**

**/\* Enable for DiscreteIntegrator: '<S2>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE\_h = 1U;**

**/\* Enable for S-Function (vxinterrupt1): '<Root>/Async Interrupt' incorporates:**

**\* SubSystem: '<S3>/Subsystem'**

**\*/**

**/\* Enable for function-call system: '<S3>/Subsystem' \*/**

**/\* Enable for S-Function (vxtask1): '<S4>/S-Function' incorporates:**

**\* SubSystem: '<Root>/Algorithm'**

**\*/**

**/\* Enable for function-call system: '<Root>/Algorithm' \*/**

**rtwdemo\_async\_M->Timing.clockTick2 = rtwdemo\_async\_M->Timing.clockTick4; rtwdemo\_async\_DW.Algorithm\_RESET\_ELAPS\_T = true;**

**/\* Enable for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE = 1U;**

**/\* End of Enable for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* End of Enable for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

**/\* Model terminate function \*/**

**static void rtwdemo\_async\_terminate(void)**

**{**

**/\* Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

**/\* Disable interrupt for ISR system: isr\_num1\_vec192 \*/ sysIntDisable(1);**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

**/\* Disable interrupt for ISR system: isr\_num2\_vec193 \*/ sysIntDisable(2);**

**/\* End of Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' incorporates:**

**\* SubSystem: '<S3>/Subsystem'**

**\*/**

**/\* Termination for function-call system: '<S3>/Subsystem' \*/**

**/\* Terminate for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* VxWorks Task Block: '<S4>/S-Function' (vxtask1) \*/**

**/\* Destroy task: Task0 \*/ taskDelete(rtwdemo\_async\_DW.SFunction\_IWORK.TaskID);**

**/\* End of Terminate for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* End of Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

**/\*========================================================================\***

**\* Start of Classic call interface \***

**\*========================================================================\*/**

**void MdlOutputs(int\_T tid)**

**{**

**rtwdemo\_async\_output(tid);**

**}**

**void MdlUpdate(int\_T tid)**

**{**

**rtwdemo\_async\_update(tid);**

**}**

**void MdlInitializeSizes(void)**

**{**

**}**

**void MdlInitializeSampleTimes(void)**

**{**

**}**

**void MdlInitialize(void)**

**{**

**}**

**void MdlStart(void)**

**{**

**rtwdemo\_async\_initialize();**

**}**

**void MdlTerminate(void)**

**{**

**rtwdemo\_async\_terminate();**

**}**

**/\* Registration function \*/ RT\_MODEL\_rtwdemo\_async\_T \*rtwdemo\_async(void)**

**{**

**/\* Registration code \*/**

**/\* initialize non-finites \*/ rt\_InitInfAndNaN(sizeof(real\_T));**

**/\* initialize real-time model \*/**

**(void) memset((void \*)rtwdemo\_async\_M, 0, sizeof(RT\_MODEL\_rtwdemo\_async\_T));**

**/\* Initialize timing info \*/**

**{**

**int\_T \*mdlTsMap = rtwdemo\_async\_M->Timing.sampleTimeTaskIDArray; mdlTsMap[0] = 0;**

**mdlTsMap[1] = 1;**

**/\* polyspace +2 MISRA2012:D4.1 [Justified:Low] "rtwdemo\_async\_M points to static memory which is guaranteed to be non-NULL" \*/**

**rtwdemo\_async\_M->Timing.sampleTimeTaskIDPtr = (&mdlTsMap[0]); rtwdemo\_async\_M->Timing.sampleTimes =**

**(&rtwdemo\_async\_M->Timing.sampleTimesArray[0]); rtwdemo\_async\_M->Timing.offsetTimes = (&rtwdemo\_async\_M->Timing.offsetTimesArray[0]);**

**/\* task periods \*/**

**rtwdemo\_async\_M->Timing.sampleTimes[0] = (0.016666666666666666);**

**rtwdemo\_async\_M->Timing.sampleTimes[1] = (0.05);**

**/\* task offsets \*/**

**rtwdemo\_async\_M->Timing.offsetTimes[0] = (0.0);**

**rtwdemo\_async\_M->Timing.offsetTimes[1] = (0.0);**

**}**

**rtmSetTPtr(rtwdemo\_async\_M, &rtwdemo\_async\_M->Timing.tArray[0]);**

**{**

**int\_T \*mdlSampleHits = rtwdemo\_async\_M->Timing.sampleHitArray;**

**int\_T \*mdlPerTaskSampleHits = rtwdemo\_async\_M->Timing.perTaskSampleHitsArray; rtwdemo\_async\_M->Timing.perTaskSampleHits = (&mdlPerTaskSampleHits[0]); mdlSampleHits[0] = 1;**

**rtwdemo\_async\_M->Timing.sampleHits = (&mdlSampleHits[0]);**

**}**

**rtmSetTFinal(rtwdemo\_async\_M, 0.5);**

**rtwdemo\_async\_M->Timing.stepSize0 = 0.016666666666666666;**

**rtwdemo\_async\_M->Timing.stepSize1 = 0.05;**

**rtwdemo\_async\_M->solverInfoPtr = (&rtwdemo\_async\_M->solverInfo); rtwdemo\_async\_M->Timing.stepSize = (0.016666666666666666);**

**rtsiSetFixedStepSize(&rtwdemo\_async\_M->solverInfo, 0.016666666666666666); rtsiSetSolverMode(&rtwdemo\_async\_M->solverInfo, SOLVER\_MODE\_MULTITASKING);**

**/\* block I/O \*/**

**rtwdemo\_async\_M->blockIO = ((void \*) &rtwdemo\_async\_B); (void) memset(((void \*) &rtwdemo\_async\_B), 0,**

**sizeof(B\_rtwdemo\_async\_T));**

**/\* states (dwork) \*/**

**rtwdemo\_async\_M->dwork = ((void \*) &rtwdemo\_async\_DW); (void) memset((void \*)&rtwdemo\_async\_DW, 0,**

**sizeof(DW\_rtwdemo\_async\_T));**

**/\* external inputs \*/**

**rtwdemo\_async\_M->inputs = (((void\*)&rtwdemo\_async\_U)); (void)memset(&rtwdemo\_async\_U, 0, sizeof(ExtU\_rtwdemo\_async\_T));**

**/\* external outputs \*/**

**rtwdemo\_async\_M->outputs = (&rtwdemo\_async\_Y); (void)memset(&rtwdemo\_async\_Y, 0, sizeof(ExtY\_rtwdemo\_async\_T));**

**/\* Initialize Sizes \*/**

**rtwdemo\_async\_M->Sizes.numContStates = (0);/\* Number of continuous states \*/ rtwdemo\_async\_M->Sizes.numY = (3); /\* Number of model outputs \*/ rtwdemo\_async\_M->Sizes.numU = (60); /\* Number of model inputs \*/ rtwdemo\_async\_M->Sizes.sysDirFeedThru = (1);/\* The model is direct feedthrough \*/ rtwdemo\_async\_M->Sizes.numSampTimes = (2);/\* Number of sample times \*/ rtwdemo\_async\_M->Sizes.numBlocks = (19);/\* Number of blocks \*/ rtwdemo\_async\_M->Sizes.numBlockIO = (6);/\* Number of block outputs \*/**

**return rtwdemo\_async\_M;**

**}**

**/\*========================================================================\***

**\* End of Classic call interface \***

**\*========================================================================\*/**

这些操作的顺序很重要。在代码生成器启用激活任务的中断之前，它必须生成该任务。

#### 查看任务和任务同步代码

在生成的源文件 **rtwdemo\_async.c** 中，查看任务和任务同步代码。

代码生成器从 Task Sync 模块为函数 **Task0** 生成代码。该函数包含少量中断级代码，并作为 RTOS 任务运行。

该任务在无限 **for** 循环中等待，直到系统释放一个同步信号量。如果系统释放该信号量，该函数将更新其任务计时器，并调用为 **Algorithm** 子系统生成的代码。

在该示例模型中，为 Task Sync 模块设置了**将此任务的数据传输与调用方任务同步**参数。该参数设置使用 Async Interrupt 模块维护的计时器 (**rtM->Timing.clockTick3**) 的值更新与 Task Sync 模块相关联的计时器 (**rtM->Timing.clockTick2**)。因此，**Algorithm** 子系统中模块的代码使用基于最近中断时间的计时器值，而不是最近激活 **Task0** 的时间。

**{**

**/\* Wait for semaphore to be released by system: rtwdemo\_async/Task Sync \*/ for (;;) {**

**if (semTake(\*(SEM\_ID \*)rtwdemo\_async\_DW.SFunction\_PWORK.SemID,NO\_WAIT) != ERROR) {**

**logMsg("Rate for Task Task0() too fast.\n",0,0,0,0,0,0);**

**#if STOPONOVERRUN**

**logMsg("Aborting real-time simulation.\n",0,0,0,0,0,0); semGive(stopSem);**

**return(ERROR); #endif**

**} else {**

**semTake(\*(SEM\_ID \*)rtwdemo\_async\_DW.SFunction\_PWORK.SemID, WAIT\_FOREVER);**

**}**

**/\* Use the upstream clock tick counter for this Task. \*/**

**rtwdemo\_async\_M->Timing.clockTick2 = rtwdemo\_async\_M->Timing.clockTick4;**

**/\* Call the system: '<Root>/Algorithm' \*/**

**{**

**int32\_T i; int32\_T i\_0;**

**/\* RateTransition: '<Root>/Protected RT1' \*/**

**i = rtwdemo\_async\_DW.ProtectedRT1\_ActiveBufIdx \* 60; for (i\_0 = 0; i\_0 < 60; i\_0++) { rtwdemo\_async\_B.ProtectedRT1[i\_0] =**

**rtwdemo\_async\_DW.ProtectedRT1\_Buffer[i\_0 + i];**

**}**

**/\* End of RateTransition: '<Root>/Protected RT1' \*/**

**/\* S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* Output and update for function-call system: '<Root>/Algorithm' \*/**

**{**

**real\_T Sum; int32\_T i;**

**rtwdemo\_async\_M->Timing.clockTick2 = rtwdemo\_async\_M->Timing.clockTick4; if (rtwdemo\_async\_DW.Algorithm\_RESET\_ELAPS\_T) { rtwdemo\_async\_DW.Algorithm\_ELAPS\_T = 0U;**

**} else { rtwdemo\_async\_DW.Algorithm\_ELAPS\_T = rtwdemo\_async\_M->Timing.clockTick2 - rtwdemo\_async\_DW.Algorithm\_PREV\_T;**

**}**

**rtwdemo\_async\_DW.Algorithm\_PREV\_T = rtwdemo\_async\_M->Timing.clockTick2; rtwdemo\_async\_DW.Algorithm\_RESET\_ELAPS\_T = false;**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/**

**if (rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE == 0) {**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_DSTATE += 0.016666666666666666 \* (real\_T) rtwdemo\_async\_DW.Algorithm\_ELAPS\_T**

**\* rtwdemo\_async\_DW.Integrator\_PREV\_U;**

**}**

**/\* End of DiscreteIntegrator: '<S1>/Integrator' \*/**

**/\* Outport: '<Root>/Out3' incorporates:**

**\* SignalConversion generated from: '<S1>/Out2'**

**\*/**

**rtwdemo\_async\_Y.Out3 = rtwdemo\_async\_DW.Integrator\_DSTATE;**

**/\* Sum: '<S1>/Sum1' \*/ rtwdemo\_async\_DW.Integrator\_PREV\_U = -0.0; for (i = 0; i < 60; i++) {**

**/\* Sum: '<S1>/Sum' incorporates:**

**\* Constant: '<S1>/Offset'**

**\*/**

**Sum = rtwdemo\_async\_B.ProtectedRT1[i] + 1.25; rtwdemo\_async\_B.Sum[i] = Sum;**

**/\* Sum: '<S1>/Sum1' \*/ rtwdemo\_async\_DW.Integrator\_PREV\_U += Sum;**

**}**

**/\* Update for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE = 0U;**

**}**

**/\* End of Outputs for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* RateTransition: '<Root>/Protected RT2' \*/ for (i = 0; i < 60; i++) { rtwdemo\_async\_DW.ProtectedRT2\_Buffer[i +**

**(rtwdemo\_async\_DW.ProtectedRT2\_ActiveBufIdx == 0) \* 60] = rtwdemo\_async\_B.Sum[i];**

**}**

**rtwdemo\_async\_DW.ProtectedRT2\_ActiveBufIdx = (int8\_T) (rtwdemo\_async\_DW.ProtectedRT2\_ActiveBufIdx == 0);**

**/\* End of RateTransition: '<Root>/Protected RT2' \*/**

**}**

**}**

**}**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/ void isr\_num1\_vec192(void)**

**{**

**int\_T lock;**

**FP\_CONTEXT context;**

**/\* Use tickGet() as a portable tick**

**counter example. A much higher resolution can be achieved with a hardware counter \*/**

**rtwdemo\_async\_M->Timing.clockTick3 = tickGet();**

**/\* disable interrupts (system is configured as non-preemptive) \*/ lock = intLock();**

**/\* save floating point context \*/ fppSave(&context);**

**/\* Call the system: '<Root>/Count' \*/**

**{**

**/\* S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* Output and update for function-call system: '<Root>/Count' \*/ if (rtwdemo\_async\_DW.Count\_RESET\_ELAPS\_T) { rtwdemo\_async\_DW.Count\_ELAPS\_T = 0U;**

**} else {**

**rtwdemo\_async\_DW.Count\_ELAPS\_T = rtwdemo\_async\_M->Timing.clockTick3 - rtwdemo\_async\_DW.Count\_PREV\_T;**

**}**

**rtwdemo\_async\_DW.Count\_PREV\_T = rtwdemo\_async\_M->Timing.clockTick3; rtwdemo\_async\_DW.Count\_RESET\_ELAPS\_T = false;**

**/\* DiscreteIntegrator: '<S2>/Integrator' \*/**

**if (rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE\_h == 0) {**

**/\* DiscreteIntegrator: '<S2>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_DSTATE\_l += 0.016666666666666666 \* (real\_T)**

**rtwdemo\_async\_DW.Count\_ELAPS\_T \* rtwdemo\_async\_DW.Integrator\_PREV\_U\_o;**

**}**

**/\* End of DiscreteIntegrator: '<S2>/Integrator' \*/**

**/\* Outport: '<Root>/Out1' incorporates:**

**\* SignalConversion generated from: '<S2>/Out'**

**\*/**

**rtwdemo\_async\_Y.Out1 = rtwdemo\_async\_DW.Integrator\_DSTATE\_l;**

**/\* Update for DiscreteIntegrator: '<S2>/Integrator' incorporates:**

**\* Constant: '<S2>/Constant'**

**\*/**

**rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE\_h = 0U; rtwdemo\_async\_DW.Integrator\_PREV\_U\_o = 1.0;**

**/\* End of Outputs for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

**/\* restore floating point context \*/ fppRestore(&context);**

**/\* re-enable interrupts \*/ intUnlock(lock);**

**}**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/ void isr\_num2\_vec193(void)**

**{**

**/\* Use tickGet() as a portable tick**

**counter example. A much higher resolution can be achieved with a hardware counter \*/**

**rtwdemo\_async\_M->Timing.clockTick4 = tickGet();**

**/\* Call the system: '<S3>/Subsystem' \*/**

**{**

**/\* S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* Output and update for function-call system: '<S3>/Subsystem' \*/**

**/\* S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* VxWorks Task Block: '<S4>/S-Function' (vxtask1) \*/**

**/\* Release semaphore for system task: Task0 \*/**

**semGive(\*(SEM\_ID \*)rtwdemo\_async\_DW.SFunction\_PWORK.SemID);**

**/\* End of Outputs for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* End of Outputs for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

**}**

代码生成器为 ISR **isr\_num1\_vec192** 和 **isr\_num2\_vec293** 生成代码。ISR **isr\_num2\_vec192** 将：

* 禁用中断。
* 保存浮点上下文。
* 调用为子系统生成的代码（该子系统连接到接收中断的引用模型的 Inport 模块）。
* 还原浮点上下文。
* 重新启用中断。

**void isr\_num1\_vec192(void)**

**{**

**int\_T lock;**

**FP\_CONTEXT context;**

**/\* Use tickGet() as a portable tick**

**counter example. A much higher resolution can be achieved with a hardware counter \*/**

**rtwdemo\_async\_M->Timing.clockTick3 = tickGet();**

**/\* disable interrupts (system is configured as non-preemptive) \*/ lock = intLock();**

**/\* save floating point context \*/ fppSave(&context);**

**/\* Call the system: '<Root>/Count' \*/**

**{**

**/\* S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* Output and update for function-call system: '<Root>/Count' \*/ if (rtwdemo\_async\_DW.Count\_RESET\_ELAPS\_T) { rtwdemo\_async\_DW.Count\_ELAPS\_T = 0U;**

**} else {**

**rtwdemo\_async\_DW.Count\_ELAPS\_T = rtwdemo\_async\_M->Timing.clockTick3 - rtwdemo\_async\_DW.Count\_PREV\_T;**

**}**

**rtwdemo\_async\_DW.Count\_PREV\_T = rtwdemo\_async\_M->Timing.clockTick3; rtwdemo\_async\_DW.Count\_RESET\_ELAPS\_T = false;**

**/\* DiscreteIntegrator: '<S2>/Integrator' \*/**

**if (rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE\_h == 0) {**

**/\* DiscreteIntegrator: '<S2>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_DSTATE\_l += 0.016666666666666666 \* (real\_T)**

**rtwdemo\_async\_DW.Count\_ELAPS\_T \* rtwdemo\_async\_DW.Integrator\_PREV\_U\_o;**

**}**

**/\* End of DiscreteIntegrator: '<S2>/Integrator' \*/**

**/\* Outport: '<Root>/Out1' incorporates:**

**\* SignalConversion generated from: '<S2>/Out'**

**\*/**

**rtwdemo\_async\_Y.Out1 = rtwdemo\_async\_DW.Integrator\_DSTATE\_l;**

**/\* Update for DiscreteIntegrator: '<S2>/Integrator' incorporates:**

**\* Constant: '<S2>/Constant'**

**\*/**

**rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE\_h = 0U; rtwdemo\_async\_DW.Integrator\_PREV\_U\_o = 1.0;**

**/\* End of Outputs for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

**/\* restore floating point context \*/ fppRestore(&context);**

**/\* re-enable interrupts \*/ intUnlock(lock);**

**}**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

ISR **isr\_num2\_vec293** 维护用于存储中断发生时的计时单元计数的计时器。更新计时器后，ISR 会释放用于激活 **Task0** 的信号量。

**/\* Spawned with priority: 50 \*/ void Task0(void)**

**{**

**/\* Wait for semaphore to be released by system: rtwdemo\_async/Task Sync \*/ for (;;) {**

**if (semTake(\*(SEM\_ID \*)rtwdemo\_async\_DW.SFunction\_PWORK.SemID,NO\_WAIT) != ERROR) {**

**logMsg("Rate for Task Task0() too fast.\n",0,0,0,0,0,0); #if STOPONOVERRUN**

**logMsg("Aborting real-time simulation.\n",0,0,0,0,0,0); semGive(stopSem);**

**return(ERROR); #endif**

**} else {**

**semTake(\*(SEM\_ID \*)rtwdemo\_async\_DW.SFunction\_PWORK.SemID, WAIT\_FOREVER);**

**}**

**/\* Use the upstream clock tick counter for this Task. \*/**

**rtwdemo\_async\_M->Timing.clockTick2 = rtwdemo\_async\_M->Timing.clockTick4;**

**/\* Call the system: '<Root>/Algorithm' \*/**

**{**

**int32\_T i; int32\_T i\_0;**

**/\* RateTransition: '<Root>/Protected RT1' \*/**

**i = rtwdemo\_async\_DW.ProtectedRT1\_ActiveBufIdx \* 60; for (i\_0 = 0; i\_0 < 60; i\_0++) { rtwdemo\_async\_B.ProtectedRT1[i\_0] =**

**rtwdemo\_async\_DW.ProtectedRT1\_Buffer[i\_0 + i];**

**}**

**/\* End of RateTransition: '<Root>/Protected RT1' \*/**

**/\* S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* Output and update for function-call system: '<Root>/Algorithm' \*/**

**{**

**real\_T Sum; int32\_T i;**

**rtwdemo\_async\_M->Timing.clockTick2 = rtwdemo\_async\_M->Timing.clockTick4; if (rtwdemo\_async\_DW.Algorithm\_RESET\_ELAPS\_T) { rtwdemo\_async\_DW.Algorithm\_ELAPS\_T = 0U;**

**} else { rtwdemo\_async\_DW.Algorithm\_ELAPS\_T = rtwdemo\_async\_M->Timing.clockTick2 - rtwdemo\_async\_DW.Algorithm\_PREV\_T;**

**}**

**rtwdemo\_async\_DW.Algorithm\_PREV\_T = rtwdemo\_async\_M->Timing.clockTick2; rtwdemo\_async\_DW.Algorithm\_RESET\_ELAPS\_T = false;**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/**

**if (rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE == 0) {**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_DSTATE += 0.016666666666666666 \* (real\_T) rtwdemo\_async\_DW.Algorithm\_ELAPS\_T**

**\* rtwdemo\_async\_DW.Integrator\_PREV\_U;**

**}**

**/\* End of DiscreteIntegrator: '<S1>/Integrator' \*/**

**/\* Outport: '<Root>/Out3' incorporates:**

**\* SignalConversion generated from: '<S1>/Out2'**

**\*/**

**rtwdemo\_async\_Y.Out3 = rtwdemo\_async\_DW.Integrator\_DSTATE;**

**/\* Sum: '<S1>/Sum1' \*/ rtwdemo\_async\_DW.Integrator\_PREV\_U = -0.0; for (i = 0; i < 60; i++) {**

**/\* Sum: '<S1>/Sum' incorporates:**

**\* Constant: '<S1>/Offset'**

**\*/**

**Sum = rtwdemo\_async\_B.ProtectedRT1[i] + 1.25; rtwdemo\_async\_B.Sum[i] = Sum;**

**/\* Sum: '<S1>/Sum1' \*/ rtwdemo\_async\_DW.Integrator\_PREV\_U += Sum;**

**}**

**/\* Update for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE = 0U;**

**}**

**/\* End of Outputs for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* RateTransition: '<Root>/Protected RT2' \*/ for (i = 0; i < 60; i++) { rtwdemo\_async\_DW.ProtectedRT2\_Buffer[i +**

**(rtwdemo\_async\_DW.ProtectedRT2\_ActiveBufIdx == 0) \* 60] = rtwdemo\_async\_B.Sum[i];**

**}**

**rtwdemo\_async\_DW.ProtectedRT2\_ActiveBufIdx = (int8\_T) (rtwdemo\_async\_DW.ProtectedRT2\_ActiveBufIdx == 0);**

**/\* End of RateTransition: '<Root>/Protected RT2' \*/**

**}**

**}**

**}**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/ void isr\_num1\_vec192(void)**

**{**

**int\_T lock;**

**FP\_CONTEXT context;**

**/\* Use tickGet() as a portable tick**

**counter example. A much higher resolution can be achieved with a hardware counter \*/**

**rtwdemo\_async\_M->Timing.clockTick3 = tickGet();**

**/\* disable interrupts (system is configured as non-preemptive) \*/ lock = intLock();**

**/\* save floating point context \*/ fppSave(&context);**

**/\* Call the system: '<Root>/Count' \*/**

**{**

**/\* S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* Output and update for function-call system: '<Root>/Count' \*/ if (rtwdemo\_async\_DW.Count\_RESET\_ELAPS\_T) { rtwdemo\_async\_DW.Count\_ELAPS\_T = 0U;**

**} else {**

**rtwdemo\_async\_DW.Count\_ELAPS\_T = rtwdemo\_async\_M->Timing.clockTick3 - rtwdemo\_async\_DW.Count\_PREV\_T;**

**}**

**rtwdemo\_async\_DW.Count\_PREV\_T = rtwdemo\_async\_M->Timing.clockTick3; rtwdemo\_async\_DW.Count\_RESET\_ELAPS\_T = false;**

**/\* DiscreteIntegrator: '<S2>/Integrator' \*/**

**if (rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE\_h == 0) {**

**/\* DiscreteIntegrator: '<S2>/Integrator' \*/ rtwdemo\_async\_DW.Integrator\_DSTATE\_l += 0.016666666666666666 \* (real\_T)**

**rtwdemo\_async\_DW.Count\_ELAPS\_T \* rtwdemo\_async\_DW.Integrator\_PREV\_U\_o;**

**}**

**/\* End of DiscreteIntegrator: '<S2>/Integrator' \*/**

**/\* Outport: '<Root>/Out1' incorporates:**

**\* SignalConversion generated from: '<S2>/Out'**

**\*/**

**rtwdemo\_async\_Y.Out1 = rtwdemo\_async\_DW.Integrator\_DSTATE\_l;**

**/\* Update for DiscreteIntegrator: '<S2>/Integrator' incorporates:**

**\* Constant: '<S2>/Constant'**

**\*/**

**rtwdemo\_async\_DW.Integrator\_SYSTEM\_ENABLE\_h = 0U; rtwdemo\_async\_DW.Integrator\_PREV\_U\_o = 1.0;**

**/\* End of Outputs for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

**/\* restore floating point context \*/ fppRestore(&context);**

**/\* re-enable interrupts \*/ intUnlock(lock);**

**}**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

#### 查看任务终止代码

Task Sync 模块会生成以下终止代码。

**static void rtwdemo\_async\_terminate(void)**

**{**

**/\* Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

**/\* Disable interrupt for ISR system: isr\_num1\_vec192 \*/ sysIntDisable(1);**

**/\* VxWorks Interrupt Block: '<Root>/Async Interrupt' \*/**

**/\* Disable interrupt for ISR system: isr\_num2\_vec193 \*/ sysIntDisable(2);**

**/\* End of Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**/\* Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' incorporates:**

**\* SubSystem: '<S3>/Subsystem'**

**\*/**

**/\* Termination for function-call system: '<S3>/Subsystem' \*/**

**/\* Terminate for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* VxWorks Task Block: '<S4>/S-Function' (vxtask1) \*/**

**/\* Destroy task: Task0 \*/ taskDelete(rtwdemo\_async\_DW.SFunction\_IWORK.TaskID);**

**/\* End of Terminate for S-Function (vxtask1): '<S4>/S-Function' \*/**

**/\* End of Terminate for S-Function (vxinterrupt1): '<Root>/Async Interrupt' \*/**

**}**

#### 相关信息

* Async Interrupt
* Task Sync
* “Generate Interrupt Service Routines”
* “Timers in Asynchronous Tasks”
* “Create a Customized Asynchronous Library”
* “Import Asynchronous Event Data for Simulation”
* “将数据加载到根级输入端口”
* [“异步事件” （第 5-2 页）](#_bookmark29)
* “Rate Transitions and Asynchronous Blocks”
* “Asynchronous Support Limitations”

### 另请参阅详细信息

* “Generate Interrupt Service Routines”
* “Pass Asynchronous Events in RTOS as Input to a Referenced Model”
* “Timers in Asynchronous Tasks”
* “Import Asynchronous Event Data for Simulation”
* “Rate Transitions and Asynchronous Blocks”

# Simulink Coder 中的子系统

# Simulink Coder 中的引用模型

# Simulink Coder 中的合并模型

## 生成代码中的函数重用

此示例说明如何配置原子子系统来生成可重用的代码。要指定为子系统生成的代码作为原子单元执行，请在“模块参数”对话框中选择**视为原子单元**参数。该参数会启用**代码生成**选项卡上的**函数打包**参数。**函数打包**参数有以下四种设置：

* **Inline**:内嵌子系统代码
* **Nonreusable function**:I/O 作为全局数据传递的函数
* **Reusable function**:I/O 作为函数参数传递的函数
* **Auto**:让 Simulink Coder 基于上下文进行优化

**Reusable function** 和 **Auto** 设置允许代码生成器重用子系统代码。**Reusable function** 和

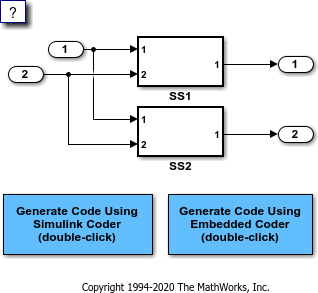
**Nonreusable function** 设置支持**函数名称选项**、**函数名称**以及**文件名选项**参数。

如果您有 Embedded Coder® 许可证，您可以配置一个不可重用子系统来接受参数。

#### 示例模型

**rtwdemo\_ssreuse** 模型包含两个相同的子系统，即 **SS1** 和 **SS2**。对于这些子系统，**函数打包**参数设置为 **Reusable function**，**函数名称**参数为 **myfun**。子系统是参数化的封装子系统。要查看封装子系统的内容，请右键点击子系统模块并选择**封装** > **查看封装内部**。

**model = 'rtwdemo\_ssreuse'; open\_system(model);**



#### 生成和检查代码

为编译和检查过程创建一个临时文件夹。打开 **Simulink Coder** 或 **Embedded Coder**。然后，生成并检查代码。

**currentDir=pwd; [~,cgDir]=rtwdemodir(); slbuild(model)**

**### Starting build procedure for: rtwdemo\_ssreuse**

**### Successful completion of build procedure for: rtwdemo\_ssreuse Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_ssreuse Code generated and compiled. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 11.517s**

**cfile=fullfile(cgDir, 'rtwdemo\_ssreuse\_grt\_rtw', 'rtwdemo\_ssreuse.c'); rtwdemodbtype(cfile, '/\* Model step', '/\* Model initialize', 1, 0);**

**/\* Model step function \*/**

**void rtwdemo\_ssreuse\_step(void)**

**{**

**/\* Outputs for Atomic SubSystem: '<Root>/SS1' \*/**

**/\* Inport: '<Root>/In1' incorporates:**

**\* Inport: '<Root>/In2'**

**\*/**

**myfun(rtwdemo\_ssreuse\_U.In1, rtwdemo\_ssreuse\_U.In2, rtwdemo\_ssreuse\_P.T1Data, rtwdemo\_ssreuse\_P.T1Break, &rtwdemo\_ssreuse\_B.SS1);**

**/\* End of Outputs for SubSystem: '<Root>/SS1' \*/**

**/\* Outport: '<Root>/Out1' \*/**

**rtwdemo\_ssreuse\_Y.Out1 = rtwdemo\_ssreuse\_B.SS1.LookupTable;**

**/\* Outputs for Atomic SubSystem: '<Root>/SS2' \*/**

**/\* Inport: '<Root>/In1' incorporates:**

**\* Inport: '<Root>/In2'**

**\*/**

**myfun(rtwdemo\_ssreuse\_U.In1, rtwdemo\_ssreuse\_U.In2, rtwdemo\_ssreuse\_P.T2Data, rtwdemo\_ssreuse\_P.T2Break, &rtwdemo\_ssreuse\_B.SS2);**

**/\* End of Outputs for SubSystem: '<Root>/SS2' \*/**

**/\* Outport: '<Root>/Out2' \*/**

**rtwdemo\_ssreuse\_Y.Out2 = rtwdemo\_ssreuse\_B.SS2.LookupTable;**

**}**

在模型单步函数中，有两个对可重用函数 **myfun** 的调用。封装参数 **T1Break**、**T1Data**、**T2Break** 和

**T2Data** 是函数参数。

将**函数打包**参数更改为 **Inline**。

**set\_param('rtwdemo\_ssreuse/SS1','RTWSystemCode','Inline') set\_param('rtwdemo\_ssreuse/SS2','RTWSystemCode','Inline')**

生成并检查代码。

**slbuild(model)**

**### Starting build procedure for: rtwdemo\_ssreuse**

**### Successful completion of build procedure for: rtwdemo\_ssreuse Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================**

**rtwdemo\_ssreuse Code generated and compiled. Generated code was out of date.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 10.564s**

**cfile=fullfile(cgDir, 'rtwdemo\_ssreuse\_grt\_rtw', 'rtwdemo\_ssreuse.c'); rtwdemodbtype(cfile, '/\* Model step', '/\* Model initialize', 1, 0);**

**/\* Model step function \*/**

**void rtwdemo\_ssreuse\_step(void)**

**{**

**real\_T Out1\_tmp;**

**/\* Outputs for Atomic SubSystem: '<Root>/SS2' \*/**

**/\* Outputs for Atomic SubSystem: '<Root>/SS1' \*/**

**/\* Sum: '<S1>/Sum' incorporates:**

* **Inport: '<Root>/In1'**
* **Inport: '<Root>/In2'**
* **Sum: '<S2>/Sum'**

**\*/**

**Out1\_tmp = rtwdemo\_ssreuse\_U.In1 + rtwdemo\_ssreuse\_U.In2;**

**/\* End of Outputs for SubSystem: '<Root>/SS2' \*/**

**/\* Outport: '<Root>/Out1' incorporates:**

* **Lookup\_n-D: '<S1>/Lookup Table'**
* **Sum: '<S1>/Sum'**

**\*/**

**rtwdemo\_ssreuse\_Y.Out1 = look1\_binlx(Out1\_tmp, rtwdemo\_ssreuse\_P.T1Break, rtwdemo\_ssreuse\_P.T1Data, 10U);**

**/\* End of Outputs for SubSystem: '<Root>/SS1' \*/**

**/\* Outputs for Atomic SubSystem: '<Root>/SS2' \*/**

**/\* Outport: '<Root>/Out2' incorporates:**

* **Lookup\_n-D: '<S2>/Lookup Table'**

**\*/**

**rtwdemo\_ssreuse\_Y.Out2 = look1\_binlx(Out1\_tmp, rtwdemo\_ssreuse\_P.T2Break, rtwdemo\_ssreuse\_P.T2Data, 10U);**

**/\* End of Outputs for SubSystem: '<Root>/SS2' \*/**

**}**

在模型单步函数中，子系统代码是内联的。

将**函数打包**参数更改为 **Nonreusable function**。对于 **SS2**，将**函数名称**参数更改为 **myfun2**。

**set\_param('rtwdemo\_ssreuse/SS1','RTWSystemCode','Nonreusable function') set\_param('rtwdemo\_ssreuse/SS2','RTWSystemCode','Nonreusable function') set\_param('rtwdemo\_ssreuse/SS2','RTWFcnName','myfun2')**

生成并检查代码。

**slbuild(model)**

**### Starting build procedure for: rtwdemo\_ssreuse**

**### Successful completion of build procedure for: rtwdemo\_ssreuse Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================**

**rtwdemo\_ssreuse Code generated and compiled. Generated code was out of date.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 8.8762s**

**cfile=fullfile(cgDir, 'rtwdemo\_ssreuse\_grt\_rtw', 'rtwdemo\_ssreuse.c'); rtwdemodbtype(cfile, '/\* Model step', '/\* Model initialize', 1, 0);**

**/\* Model step function \*/**

**void rtwdemo\_ssreuse\_step(void)**

**{**

**/\* Outputs for Atomic SubSystem: '<Root>/SS1' \*/ myfun();**

**/\* End of Outputs for SubSystem: '<Root>/SS1' \*/**

**/\* Outputs for Atomic SubSystem: '<Root>/SS2' \*/ myfun2();**

**/\* End of Outputs for SubSystem: '<Root>/SS2' \*/**

**}**

模型单步函数包含对函数 **myfun** 和 **myfun2** 的调用。这些函数具有 void-void 接口。将**函数打包**参数更改为 **Auto**。

**set\_param('rtwdemo\_ssreuse/SS1','RTWSystemCode','Auto') set\_param('rtwdemo\_ssreuse/SS2','RTWSystemCode','Auto')**

对于 **auto** 设置，Simulink Coder 会选择最佳格式。对于此模型，最佳格式是可重用函数。关闭模型并进行清理。

**bdclose(model) rtwdemoclean; cd(currentDir)**

# Simulink Coder 的代码重用

## 从顶层模型生成可重入代码

默认情况下，对于顶层模型，代码生成器生成不可重入的代码。入口函数具有 void-void 接口。代码通过共享对驻留在共享内存中的全局数据结构体的访问来与其他代码通信。

如果应用程序可以获益于代码重用，且代码的每次使用或每个实例需要保留其自己的独有数据，则可将模型配置为支持使用代码生成器生成可重入代码。要生成可重入代码，请将模型配置参数 Code interface packaging 设置为“可重用函数”。如果您正在使用 Embedded Coder 并生成 C++ 代码，您也可以将参数设置为 “**C++** 类”。在这两种情况下，代码生成器都会：

* 将模型数据（例如模块 I/O、DWork 向量和参数）打包到实时模型数据结构体 (**rtModel**) 中。
* 按引用将实时模型数据结构体作为输入参数传递给生成的模型入口函数。
* 将根级输入和输出参数作为单个参数传递给生成的模型入口函数。
* 为模型数据结构体静态分配内存。
* 将实时模型数据结构体导出到生成的头文件 **model.h** 中。

通过设置以下模型配置参数，应用其他诊断和代码生成控制：

* 要选择当模型不符合多实例代码要求时代码生成器显示的诊断消息的严重性级别，请将参数 Multi- instance code error diagnostic 设置为“无”、“警告”或“错误”。除非您需要更改当模型违反生成多实例代码的要求时显示的诊断的严重性级别。否则将参数设置为“错误”。
* 要控制生成的代码如何将根级模型输入和输出传递给可重用执行（单步）函数（需要 Embedded Coder），请将参数 Pass root-level I/O as (Embedded Coder) 设置为“单个参量”、“结构体引用”或“模型数据结构体的一部分”。

当您将 Code interface packaging 设置为“可重用函数”时，代码生成器将模型数据（如模块 I/O、 Dwork 和参数）打包到实时模型数据结构体中，并将模型结构传递给生成的模型入口函数。如果您将 Pass root-level I/O as (Embedded Coder) 设置为“模型数据结构体的一部分”，代码生成器还会将根级模型输入和输出打包到实时模型数据结构体中。

* 要通过从实时模型数据结构体中忽略错误状态字段来减少内存使用量（需要 Embedded Coder），请选择模型配置参数**删除实时模型数据结构体中的错误状态字段**。
* 要在生成的文件 **model.c** 中包含使用 **malloc** 为模型实例数据动态分配内存的函数（需要 Embedded Coder），请选择模型配置参数**使用动态内存分配进行模型初始化**。如果不选择此参数，生成的代码将为模型数据结构体静态分配内存。

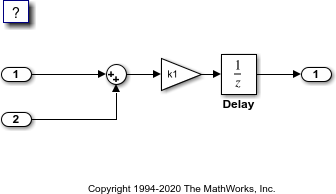
### 生成可重入的多实例代码

此示例说明如何配置模型以生成可重入的多实例代码。多个程序可以同时使用可重入代码。当您为实现可重入性而配置模型时，执行（单步）入口函数使用根级输入和输出参数而不是全局数据结构体。检查配置设置后，生成并查看生成的代码。

#### 打开模型

打开模型 **rtwdemo\_reusable**。该模型包含两个根 Inport 模块和一个根 Outport 模块。

**model='rtwdemo\_reusable'; open\_system(model);**



将模型的副本保存到可写位置。

**currentDir=pwd; [~,cgDir] = rtwdemodir();**

#### 检查相关模型配置参数设置

1.打开 **Embedded Coder**。 2.打开“模型配置参数”对话框。

1. 模型配置参数**系统目标文件**设置为 **ert.tlc**。虽然对于**系统目标文件**设置为 **grt.tlc** 的模型可生成可重入代码，但基于 ERT 和 ERT 的系统目标文件可以更好地控制代码如何通过根级 I/O。
2. 打开**代码生成 > 接口**窗格，并查看相关模型配置参数设置。
   * **代码接口打包**设置为 **Reusable function**。此参数设置指示代码生成器产生可重用的多实例代码。
   * **Reusable function** 的设置显示参数**多实例代码错误诊断**。该参数设置为 **Error**，表示如果模型违反生成多实例代码的要求，代码生成器将中止。
   * **根级 I/O 传递方式**设置为 **Part of model data structure**。此设置将根级别模型输入和输出打包到 实时模型数据结构体 (**rtModel**) 中，该结构体是一个经过优化的数据结构体，可代替 **SimStruct** 作为模型的顶级数据结构体。
   * **删除实时模型数据结构体中的错误状态字段**处于选中状态。此参数设置通过省略生成的实时模型数据结构体中的错误状态字段来减少内存使用量。

#### 生成和查看代码

**slbuild(model);**

**### Starting build procedure for: rtwdemo\_reusable**

**### Successful completion of build procedure for: rtwdemo\_reusable Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_reusable Code generated and compiled. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 17.491s**

#### 查看生成的代码

* + **ert\_main.c** 是模型的示例主程序（执行框架）。此代码通过调用入口函数

**rtwdemo\_reusable\_step** 来控制模型代码执行。使用此文件作为对执行框架进行编码的起始点。

* + **rtwdemo\_reusable.c** 包含实现模型算法的代码的入口函数。此文件包含速率调度代码。
  + **rtwdemo\_reusable.h** 声明模型数据结构体和对接模型入口函数和数据结构体的公共接口。
  + **rtwtypes.h** 定义生成的代码所需的数据类型、结构体和宏。

#### 代码接口

打开并查看代码接口报告。使用该报告中的信息编写用于您的执行框架的接口代码。

1.通过添加指令 **#include rtwdemo\_reusable.h** 来包含生成的头文件。 2.将输入数据写入模型的 Inport 模块的生成代码。

3.调用生成的入口函数。

1. 从为模型 Outport 模块生成的代码中读取数据。输入端口：
   * **<Root>/In1** 的数据类型为 **real\_T**，维度为 1
   * **<Root>/In2** 的数据类型为 **real\_T**，维度为 1

入口函数：

* + 初始化入口函数，**void rtwdemo\_reusable\_initialize(RT\_MODEL \*const rtM)**。在启动时，调用一次此函数。
  + 输出和更新（单步）入口函数，**void rtwdemo\_reusable\_step(RT\_MODEL \*const rtM)**。以模型中最快的速率定期调用此函数。对于此模型，每秒调用一次该函数。要实现实时执行，请将此函数附加到计时器。

输出端口：

* + **<Root>/Out1** 的数据类型为 **real\_T**，维度为 1

#### 检查单步函数

**Examine the |rtwdemo\_reusable\_step| function code in |rtwdemo\_reusable.c|.**

**cfile = fullfile(cgDir,'rtwdemo\_reusable\_ert\_rtw','rtwdemo\_reusable.c'); rtwdemodbtype(cfile,'/\* Model step function', '/\* Model initialize function ', 1, 0);**

**/\* Model step function \*/**

**void rtwdemo\_reusable\_step(RT\_MODEL \*const rtM)**

**{**

**D\_Work \*rtDWork = rtM->dwork;**

**ExternalInputs \*rtU = (ExternalInputs \*) rtM->inputs; ExternalOutputs \*rtY = (ExternalOutputs \*) rtM->outputs;**

**/\* Outport: '<Root>/Out1' incorporates:**

**\* UnitDelay: '<Root>/Delay'**

**\*/**

**rtY->Out1 = rtDWork->Delay\_DSTATE;**

**/\* Gain: '<Root>/Gain' incorporates:**

* **Sum: '<Root>/Sum'**
* **UnitDelay: '<Root>/Delay'**

**\*/**

**rtDWork->Delay\_DSTATE = (rtU->In1 + rtU->In2) \* rtP.k1;**

**}**

代码生成器将模型数据作为实时模型数据结构体的一部分传递给 **rtwdemo\_reusable\_step** 函数。尝试模型配置参数**代码接口打包**和**根级 I/O 传递方式**的不同设置，并重新生成代码。观察函数原型是如何变化的。

#### 关闭模型和报告

关闭模型和代码生成报告。

**bdclose(model) rtwdemoclean; cd(currentDir)**

### 在实例之间共享数据

当您的代码多次调用一个可重入模型入口函数时，每次调用都表示该模型的一个实例。默认情况下，代码生成器生成的代码假定每个实例分别读写模型中信号、模块状态和参数的单独副本。

* + 要在各实例之间共享一段参数数据（例如，共享可重用的 PID 控制算法的设定值），请使用参数对象，如 **Simulink.Parameter**。然后，将参数配置为使用 **Auto** 之外的存储类，或在代码映射编辑器中，将参数数据的对应类别的默认存储类“默认”（默认设置）设置为“模型默认”。参数对象在代码中显示为函数直接访问的全局符号，如全局变量。有关详细信息，请参阅“模型接口元素的 [C 代码生成配 置” （第 17-2](#_bookmark135) 页）。
  + 要在各实例之间共享一段非参数数据（例如，共享故障指示或累加器），请使用数据存储。您可以将数据存储配置为在代码中显示为函数直接访问的全局符号，例如全局变量。使用 **Simulink.Signal** 对象创建全局数据存储，或使用 Data Store Memory 模块并选择模块参数**跨模型实例共享**。有关详细信 息，请参阅“通过创建数据存储对全局数据建模”和 Data Store Memory。

### 另请参阅详细信息

* + “What Is Reentrant Code?” (Embedded Coder)
  + “Storage Classes and Reentrant, Multi-Instance Models and Components”

# 配置 Simulink Coder 的模型参数

* + [“配置运行时环境选项” （第 10-2 页）](#_bookmark43)
  + [“注册新硬件设备” （第 10-10 页）](#_bookmark56)

## 配置运行时环境选项

当您使用 Simulink 软件创建和执行模型并使用代码生成器生成 C 或 C++ 代码时，请针对最多三种运行时环境考虑您的配置：

* + 在应用程序开发期间运行 MathWorks® 软件的 MATLAB 开发计算机运行时环境。
  + 生产硬件运行时环境，在该环境中部署用于生产的应用程序。
  + 测试硬件运行时环境，在该环境中，您在部署之前测试正在开发的应用程序。

一个运行时环境可以提供多种功能，但各运行时环境在概念上仍然不同。通常，MATLAB 开发计算机是测试硬件。通常情况下，生产硬件与 MATLAB 开发或测试硬件不同，在功能上也不如后者强大。许多类型的生产硬件只能运行下载的可执行文件。

在下列情况下，会提供有关生产硬件板和配套使用的编译器的信息：

* + 使用 Simulink 软件来仿真稍后要生成代码的模型
  + 使用代码生成器生成用于在生产硬件上部署的代码

该软件使用硬件板和编译器信息来获得在仿真中和为生产硬件生成的代码中执行的整数和定点运算结果的位真一致性。代码生成器使用这些信息来创建以最高效率执行的代码。

生成用于在测试硬件上进行测试的代码时，请提供有关您使用的测试硬件板和编译器的信息。代码生成器使用此信息来创建代码，以提供以下结果之间的位真一致性：

* + 在仿真中执行的整数和定点运算
  + 在生产硬件上运行的生成代码
  + 在测试硬件上运行的生成代码

即使生产硬件和测试硬件不同，您也可以获得结果的位真一致性。在 C 标准没有完全定义行为的情况下，这两种硬件的编译器可以使用不同的默认值。

### 配置生产和测试硬件

您可以为特定硬件板及其设备类型指定模型仿真或代码生成。例如，您可以设置数据大小、字节顺序和编译器行为，如整数舍入。您可以配置：

* + [您使用的生产硬件和配套的编译器。这些信息会影响仿真和代码生成。请参阅“影响普通模式仿真的生产硬件设置示例” （第 10-7](#_bookmark55) 页）。
  + 您使用的测试硬件和配套的编译器。这些信息仅影响代码生成。

通过选择 **Configuration Parameters** > **Hardware Implementation** 来配置生产硬件。默认情况下， Hardware Implementation 窗格仅列出 **Hardware board**、**Device vendor** 和 **Device type** 参数字 段。除非您已安装硬件支持包，否则 **Hardware board** 会列出值 “**None**” 或 “**Determine by Code Generation system target file**” 和 “**Get Hardware Support Packages**”。安装硬件支持包

后，该列表还包含对应的硬件板名称。如果您选择一个硬件板名称，则显示该硬件板的参数。要设置设备详细信息（如数据大小和字节顺序），请点击 **Device details**。

#### 在 Configuration Parameters > Hardware Implementation > Advanced parameters 窗格中配置测试硬件。要启用配置测试硬件详细信息的参数，请禁用 Configuration Parameters > Hardware Implementation > Advanced parameters > Test hardware is the same as production

**hardware** 参数。为测试硬件生成的代码在由测试硬件参数指定的环境中执行。代码的行为就好像它在为生产硬件指定的环境中执行一样。有关详细信息，请参阅“测试硬件注意事项” [（第 10-7](#_bookmark54) 页）。

在以下情况下，默认值和属性在 **Hardware Implementation** 窗格中显示为初始值：

* + 您在 **Code Generation** 窗格中指定 **System target file**。
  + 系统目标文件指定默认微处理器及其硬件属性。

您不能更改只有一个可能值的参数。具有多个可能值的参数会提供有效值的列表。如果您在 **Hardware Implementation** 窗格中手动指定硬件属性，请确认这些值与系统目标文件一致。否则，生成的代码可能无法编译或执行，或者可能会执行但产生不正确的结果。

硬件实现参数向 MATLAB 软件描述硬件和编译器属性。代码生成器使用这些信息为运行时环境生成尽可能高效运行的代码。生成的代码为仿真、生产代码和测试代码中的整数和定点运算的结果提供位真一致性。

有关特定参数的详细信息，请参阅““硬件实现”窗格”。要查看 **Hardware Implementation** 窗格功能的示例，请参阅 **rtwdemo\_targetsettings** 示例模型。有关配置硬件实现的详细信息，请参阅：

* + [“指定硬件板” （第 10-3 页）](#_bookmark45)
  + [“指定设备供应商” （第 10-4 页）](#_bookmark46)
  + [“指定设备类型” （第 10-4 页）](#_bookmark47)
  + [“设置设备数据类型的位长度” （第 10-4 页）](#_bookmark48)
  + [“设置设备的字节顺序” （第 10-5 页）](#_bookmark49)
  + [“设置有符号整数除法的商舍入行为” （第 10-5 页）](#_bookmark50)
  + [“设置有符号整数的算术右移行为” （第 10-6 页）](#_bookmark51)
  + [“更新版本 14 的硬件配置” （第 10-6 页）](#_bookmark52)

#### 指定硬件板

指定运行从您的模型生成的代码的硬件板。为 **Configuration Parameters** > **Hardware Implementation** > **Hardware board** 选择值。

Hardware Implementation 窗格标识在 **Configuration Parameters** > **Code Generation** 上选择的系统目标文件。

要配置测试硬件，请使用 **Configuration Parameters** > **Hardware Implementation** > **Advanced parameters** 窗格。

要启用配置测试硬件详细信息的参数，请将 **ProdEqTarget** 设置为 **off**。

#### 如何指定硬件板

|  |  |
| --- | --- |
| **如果** | **选择** |
| 菜单包含您要使用的硬件板的名称。 | 该硬件板的名称。  如果您选择一个硬件板名称，则显示该硬件板的参数。 |
| 菜单不包含要使用的硬件板的名称。 | “**Get Hardware Support Packages**”。  该值将打开支持包安装程序。安装所需的支持包。安装支持包后，菜单包括相关硬件板的名称。 |
| 模型配置使用系统目标文件 **ert.tlc**、 **realtime.tlc** 或 **autosar.tlc**。 | “**None**”。  没有为硬件实现指定硬件板。 |
| 模型配置不使用系统目标文件 **ert.tlc**、 **realtime.tlc** 或 **autosar.tlc**。 | “**Determine by Code Generation system target file**”。  代码生成器使用指定的系统目标文件来确定硬件实现。 |

**指定设备供应商**

要指定硬件设备的微处理器供应商，请使用 **Device vendor** 参数。您的选择将决定 **Device type** 菜单中的可用微处理器。如果没有显示供应商名称，请选择 “**Custom Processor**”。然后，使用 **Device type** 参数指定微处理器。

* + 有关 **Device vendor** 和 **Device type** 值的完整列表，请参阅“Device vendor” 和“Device type”。
  + 要将 **Device vendor** 和 **Device type** 值添加到 **Hardware Implementation** 窗格中显示的默认设置，请参阅“注册新硬件设备” [（第 10-10](#_bookmark56) 页）。

#### 指定设备类型

要从 **Device vendor** 选项中列出的受支持设备中指定微处理器名称，请使用 **Device type** 参数。如果微处理器未出现在菜单中，请将 **Device vendor** 更改为 “**Custom Processor**”。然后，为您的自定义设备指定设备详细信息。

如果您选择系统目标文件为其指定默认硬件属性的设备类型，则属性将显示为初始值。您不能更改只具有一个可能选项的参数的值。具有多个可能值的参数会提供一个菜单。为您的硬件选择值。

#### 设置设备数据类型的位长度

**Number of bits** 参数描述微处理器的 **native word size** 以及 **char**、**short**、**int** 和 **long** 数据的位长度。要使代码生成成功，需要满足以下条件：

* + 位长度必须是：**char** <= **short** <= **int** <= **long**。
  + 位长度必须是 8 的倍数，最大为 32。
  + **long** 数据的位长度不能小于 32。

**rtwtypes.h** 文件定义整数类型名称。您提供的值必须与在编译器 **limits.h** 头文件中定义的字长一致。代码生成器将其整数类型名称映射到对应的 Simulink 整数类型名称。

如果没有具有匹配字长的 ANSI® C 类型，但有更大的 ANSI C 类型，则代码生成器对 **int8\_T**、

**uint8\_T**、**int16\_T**、**uint16\_T**、**int32\_T** 和 **uint32\_T** 使用更大的类型。当代码生成器使用更大的类型时，得到的记录值（例如 MAT 文件记录）可能具有与仿真的记录值不同的数据类型。

应用程序可以使用长度为从 1（无符号）或 2（有符号）位到 32 位的整数数据。如果整数长度与可用类型的长度匹配，则代码生成器将使用该类型。如果匹配类型不可用，则代码生成器使用可容纳数据的最小可用类型，从而生成不使用不必要的高阶位的代码。例如，在支持 8 位、16 位和 32 位整数的硬件上，对于指定为 24 位的信号，代码生成器将数据实现为 **int32\_T** 或 **uint32\_T**。

使用模拟整数数据的代码的执行效率并非最高。在应用程序开发过程中，以下代码可用于模拟仅在生产硬件上可用的整数长度。模拟不影响执行结果。

在代码生成过程中，软件会检查模型数据类型与您为生产硬件指定的数据类型的兼容性。

* + 如果您为生产硬件整数指定的长度都不是 32 位，则软件会生成错误。
  + 如果模型使用的数据类型的长度小于可用的生产硬件整数长度，则软件会生成警告。

#### 从代码生成器到 Simulink 的整数类型映射

|  |  |
| --- | --- |
| **代码生成器整数类型** | **Simulink 整数类型** |
| **boolean\_T** | **boolean** |
| **int8\_T** | **int8** |
| **uint8\_T** | **uint8** |
| **int16\_T** | **int16** |
| **uint16\_T** | **uint16** |
| **int32\_T** | **int32** |
| **uint32\_T** | **uint32** |

**设置设备的字节顺序**

**Byte ordering** 参数指定硬件是使用 “**Big Endian**”（最高有效位字节优先）还是 “**Little Endian**”

（最低有效位字节优先）字节顺序。如果保留为 “**Unspecified**”，则代码生成器会生成确定硬件字节顺序的代码。此设置的效率最低。

#### 设置有符号整数除法的商舍入行为

ANSI C 并不详尽定义当有符号整数除以另一个有符号整数时编译器使用的商舍入方法。因此，舍入行为依赖于实现。如果两个整数都是正数或都是负数，则商必须向下舍入。如果其中一个整数是正数而另一个是负数，则商既可以向上舍入也可以向下舍入。

**Signed integer division rounds to** 参数向代码生成器告知编译器如何舍入有符号整数除法的结果。提供这些信息不会更改编译器的操作。它仅向代码生成器描述该行为，该代码生成器使用这些信息来优化为有符号整数除法生成的代码。参数值包括：

* + “**Zero**” - 如果商介于两个整数之间，编译器将选择更接近零的整数作为结果。
  + “**Floor**” - 如果商介于两个整数之间，编译器将选择更接近负无穷的整数作为结果。
  + “**Undefined**” - 如果 “**Zero**” 或 “**Floor**” 不能描述编译器行为或者该行为是未知的，请选择此值。

应避免选择 “**Undefined**”。当代码生成器不知道编译器的有符号整数除法舍入行为时，模型编译会生成额外的代码。

编译器商舍入行为根据这些值而变化。

您可以从编译器文档获得有符号整数除法舍入的编译器实现。如果文档不可用，您可以通过试验确定此行为。

#### Zero、Floor 和 Undefined 的商舍入示例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **D** | **Ideal N/D** | **Zero** | **Floor** | **Undefined** |
| **33** | 4 | **8.25** | **8** | **8** | **8** |
| **-33** | **4** | **-8.25** | **-8** | **-9** | **-8** 或 **-9** |
| **33** | **-4** | **-8.25** | **-8** | **-9** | **-8** 或 **-9** |
| **-33** | **-4** | **8.25** | **8** | **8** | **8** 或 **9** |

**设置有符号整数的算术右移行为**

ANSI C 没有为编译器定义对负整数的右移行为。因此，舍入行为依赖于实现。**Shift right on a signed integer as arithmetic shift** 选项向代码生成器告知编译器如何对负整数实现右移。提供这些信息不会更改编译器的操作。它仅向代码生成器描述该行为，该代码生成器使用这些信息优化为算术右移生成的代 码。

如果 C 编译器通过算数右移实现符号整数右移，请选择此选项。否则，请清除该选项。算术右移用最高有效位的值填充右移空出的位，最高有效位表示 2 的补码记数法中数字的符号。该选项默认为选中状态。如果您的编译器将右移作为算术移位处理，则此设置为首选。

* + 当您选择该选项时，无论 Simulink 模型何时对有符号整数执行算术移位，代码生成器都会生成高效代码。
  + 当您清除该选项时，代码生成器会生成完全可移植但效率较低的代码来实现向右算术移位。

您可以从编译器文档中获得用于算术右移的编译器实现。如果文档不可用，您可以通过试验确定此行为。

#### 更新版本 14 的硬件配置

如果您的模型是在版本 14 之前创建的，并且您尚未更新模型，则 **Configure current execution hardware device** 参数 (**TargetUnknown**) 的值默认为 **'on'**。

要更新模型，请清除 **Configuration Parameters** > **Hardware Implementation** > **Advanced parameters** > **Test hardware** > **Configure test hardware** 复选框。或者在命令行窗口中键入：

**cs = getActiveConfigSet('your\_model\_name'); set\_param(cs,'TargetUnknown','off');**

对模型的这一更新将：

* + 启用 **Test Hardware is the same as production hardware** 参数 (**ProdEqTarget**)，将该参数设置为 **'on'**。

#### 将 Production device vendor and type 参数 (ProdHWDeviceType) 的值复制到 Test device vendor and type 参数 (TargetHWDeviceType)。

要完成更新，请执行下列操作：

1. 清除 **Configuration Parameters** > **Hardware Implementation** > **Advanced parameters** > **Test hardware** > **Test Hardware is the same as production hardware** 复选框。仅在您的生产硬件和测试硬件不同时才应用此步骤。
2. 在 **Configuration Parameters** > **Hardware implementation** > **Advanced parameters** 中设置参数，以匹配您的生产和测试系统。
3. 保存模型。

### 生产硬件注意事项

在配置生产硬件时，请考虑以下几点：

* + 生产硬件可以具有不同于 MATLAB 开发计算机的字长和其他硬件特性。您可以在不同于生产硬件或 MATLAB 开发计算机的硬件上对代码进行原型构建。在生成代码时，代码生成器会解决这些差异。
  + Simulink 产品会使用生产硬件配置中的一些信息。使用这些信息，无需生成代码即可执行仿真，从而得到与执行生成的代码相同的结果。例如，通过这些结果可以检测生产硬件上出现的错误情况，例如硬件溢出。
  + 代码生成器生成的代码提供与整数和定点运算的 Simulink 结果的位真一致性。模拟不可用数据长度的生成代码的运行效率低于非模拟情况。对于整数和定点结果，模拟不影响与 Simulink 的位真一致性。
  + 如果在应用程序开发期间更改运行时环境，则在生成或重新生成代码之前，请重新配置新运行时环境的硬件实现参数。如果执行代码所在的硬件并非生成该代码的硬件，则对于仿真、生产代码和测试代码中的整数和定点运算的结果，并不能始终实现位真一致性。
  + 要编译从模型生成的代码，请使用模型模块上的 **Integer rounding mode** 参数来仿真您要使用的 C编译器的舍入行为。此设置显示在可对有符号整数执行算术运算的模块（如 Product, Matrix Multiply和 n-D Lookup Table 模块）的参数对话框中的 **Signal Attributes** 窗格上。
  + 对于大多数模块来说，**Integer rounding mode** 的值完全定义舍入行为。对于支持定点数据和最简单舍入模式的模块，**Signed integer division rounds to** 的值也会影响舍入。有关详细信息，请参阅 “精度” (Fixed-Point Designer)。
  + 当模型包含 Model 模块时，请将它们引用的模型配置为使用相同的硬件设置。

### 测试硬件注意事项

默认情况下，测试硬件配置与生产硬件的配置相同。您可以使用生成的代码在与生产环境相同的环境中进行测试。

如果测试环境和生产环境不同，您可以通过以下设置，使生成的代码在测试硬件上运行时就像在生产硬件上运行一样：

1. 要启用测试硬件参数，请清除 **Configuration Parameters** > **Hardware Implementation** > **Advanced parameters** > **Test hardware** > **Test hardware is the same as production hardware** 复选框。或者，在命令行窗口中键入：

**cs = getActiveConfigSet('your\_model\_name'); set\_param(cs,'ProdEqTarget','off');**

1. 通过测试硬件 (**Target\***) 参数指定设备类型的详细信息。

如果选择指定默认微处理器及其硬件属性的系统目标文件，则这些默认值和属性将显示为初始值。

只有一个可能值的参数不能更改。如果修改硬件属性，请检查其值是否与系统目标文件一致。否则，生成的代码可能无法编译或执行，或者可能会执行但产生不正确的结果。

### 影响普通模式仿真的生产硬件设置示例

更改某些生产硬件设置（例如 **ProdLongLongMode** 和 **ProdIntDivRoundTo**）会影响普通模式的仿真结果。以下示例仿真具有四个输入的加法器。在第一次仿真中，**ProdLongLongMode** 处于禁用状

态。在第二次仿真中，**ProdLongLongMode** 处于启用状态。在仿真输出图中，您会发现时间步范围 125-175 之间的输出值之间存在一些小的差异。

**model = 'hwSettingEffect'; new\_system(model) open\_system(model)**

**% Create adder**

**pos = [140 140 200 340];**

**add\_block('simulink/Math Operations/Add', ... [model '/sum\_int32'], ...**

**'Inputs','++++', ...**

**'SaturateOnIntegerOverflow', ... 'on', ...**

**'Position', ... pos)**

**pos = [75 155 105 175];**

**add\_block('built-in/Inport',[model '/In1'],'Position',pos) set\_param([model '/In1'], 'OutDataTypeStr', ...**

**'int32','PortDimensions','1','SampleTime','1'); add\_line(model, 'In1/1','sum\_int32/1')**

**pos = [75 205 105 225];**

**add\_block('built-in/Inport',[model '/In2'],'Position',pos) set\_param([model '/In2'], 'OutDataTypeStr', ...**

**'int32','PortDimensions','1','SampleTime','1'); add\_line(model, 'In2/1','sum\_int32/2')**

**pos = [75 255 105 275];**

**add\_block('built-in/Inport',[model '/In3'],'Position',pos) set\_param([model '/In3'], 'OutDataTypeStr', ...**

**'int32','PortDimensions','1','SampleTime','1'); add\_line(model, 'In3/1','sum\_int32/3')**

**pos = [75 305 105 325];**

**add\_block('built-in/Inport',[model '/In4'],'Position',pos) set\_param([model '/In4'], 'OutDataTypeStr', ...**

**'int32','PortDimensions','1','SampleTime','1'); add\_line(model, 'In4/1','sum\_int32/4')**

**pos = [275 230 305 250];**

**add\_block('built-in/Outport',[model '/Out1'],'Position',pos) add\_line(model, 'sum\_int32/1','Out1/1')**

**% Specify input data t = 0:200;**

**peakValue = 1.5e9;**

**in1 = peakValue \* sin(t\*2\*pi/100); in2 = peakValue \* cos(t\*2\*pi/70); in3 = -peakValue \* sin(t\*2\*pi/40); in4 = -peakValue \* cos(t\*2\*pi/30);**

**set = Simulink.SimulationData.Dataset;**

**set = set.addElement(1, timeseries(int32(in1),t,'Name','sig1')); set = set.addElement(2, timeseries(int32(in2),t,'Name','sig2')); set = set.addElement(3, timeseries(int32(in3),t,'Name','sig3')); set = set.addElement(4, timeseries(int32(in4),t,'Name','sig4'));**

**set\_param(model, 'LoadExternalInput', 'on'); set\_param(model, 'ExternalInput', 'set');**

**set\_param(model, 'StopTime', '50');**

**% Disable production hardware setting and run first simulation set\_param(model, 'ProdLongLongMode', 'off');**

**[~, ~, y1] = sim(model, 200);**

**% Enable production hardware setting and run second simulation set\_param(model, 'ProdLongLongMode', 'on');**

**[~, ~, y2] = sim(model, 200);**

**plot([y1 y2]); figure(gcf);**

该行为差异归因于 Sum 模块中的累加器数据类型。**Accumulator data type** 模块参数设置为 “**Inherit:Inherit via internal rule**”。对于此示例，如果启用 C **long long** 数据类型，则生成的累加器数据类型的宽度为 64 位。否则，它的宽度为 32 位。视 Sum 模块的输入值不同，在 64 位累加器未进行饱和处理时，32 位累加器可能会进行饱和处理。因此，普通模式行为可能取决于 **ProdLongLongMode** 设置。在这两种情况下，普通模式行为和生产硬件行为会按位匹配。

### 另请参阅详细信息

* + ““硬件实现”窗格”
  + “Device vendor”
  + “Device type”
  + “精度” (Fixed-Point Designer)

## 注册新硬件设备

在**硬件实现**窗格中，您可以指定描述 MATLAB 软件的目标硬件和编译器属性的参数，以便：

* + 在模型仿真期间观测目标硬件。
  + 为生产或测试硬件生成优化的代码。
  + 在目标硬件上直接测试或部署生成的代码。

**硬件实现**窗格支持多个目标硬件。要扩展所支持的硬件范围，请使用 **target.Processor** 和

**target.LanguageImplementation** 类注册新硬件设备。

### 为新设备指定硬件实现

要注册新硬件设备，请执行以下操作：

1. 为新硬件设备创建一个 **target.Processor** 对象。

**myProc = target.create('Processor', ...**

**'Name', 'MyProcessor', ... 'Manufacturer', 'MyManufacturer');**

1. 为语言实现详细信息创建一个 **target.LanguageImplementation** 对象。

**myLanguageImplementation = target.create('LanguageImplementation', ...**

**'Name', 'MyProcessorImplementation');**

1. 指定语言实现详细信息。

**myLanguageImplementation.Endianess = target.Endianess.Little;**

**myLanguageImplementation.AtomicIntegerSize = 64;**

**myLanguageImplementation.AtomicFloatSize = 64;**

**myLanguageImplementation.WordSize = 64;**

**myLanguageImplementation.DataTypes.Char.Size = 8;**

**myLanguageImplementation.DataTypes.Short.Size = 16;**

**myLanguageImplementation.DataTypes.Int.Size = 32;**

**myLanguageImplementation.DataTypes.Long.Size = 64; myLanguageImplementation.DataTypes.LongLong.IsSupported = true; myLanguageImplementation.DataTypes.LongLong.Size = 64;**

**myLanguageImplementation.DataTypes.Float.Size = 32;**

**myLanguageImplementation.DataTypes.Double.Size = 64;**

**myLanguageImplementation.DataTypes.Pointer.Size = 32;**

**myLanguageImplementation.DataTypes.SizeT.Size = 64;**

**myLanguageImplementation.DataTypes.PtrDiffT.Size = 64;**

1. 将语言实现与硬件设备相关联。

**myProc.LanguageImplementations = myLanguageImplementation;**

1. 将 **target.Processor** 对象添加到内部数据库中。

**objectsAdded = target.add(myProc);**

在**硬件实现**窗格中，您现在可以将**设备供应商**和**设备类型**分别设置为 “**MyManufacturer**” 和 “**MyProcessor**”。

### 指定在各 MATLAB 会话之间持久保留的硬件实现

默认情况下，当您将目标对象添加到内部数据库时，目标数据仅可用于当前 MATLAB 会话。您可以指定是否在各 MATLAB 会话之间持久保留目标数据。

1. 为新硬件设备创建一个 **target.Processor** 对象。

**myProc = target.create('Processor', ...**

**'Name', 'MyProcessor', ... 'Manufacturer', 'MyManufacturer');**

**existingImplementation = target.get('LanguageImplementation', ... 'ARM Compatible-ARM Cortex');**

**myProc.LanguageImplementations = existingImplementation;**

1. 将 **target.Processor** 对象添加到一个内部数据库，并指定是否在各 MATLAB 会话之间持久保留目标数据。

**objectsAdded = target.add(myProc, 'UserInstall', true);**

1. 您可以从内部数据库中删除该对象。

**target.remove(objectsAdded);**

### 通过修改现有实现来创建硬件实现

如果某个现有硬件实现包含新硬件实现中所需的大部分值，则可以通过创建和修改该现有实现的副本来快速创建新实现。

1. 为新硬件设备创建一个 **target.Processor** 对象。

**myProc = target.create('Processor', ...**

**'Name', 'MyProcessor', ... 'Manufacturer', 'MyManufacturer');**

1. 创建一个用于复制现有语言实现的 **target.LanguageImplementation** 对象。

**myCopiedImplementation = target.create('LanguageImplementation', ...**

**'Name', 'MyCopiedImplementation', ... 'Copy', 'Atmel-AVR');**

1. 指定所需的语言实现详细信息。例如，字节顺序。

**myCopiedImplementation.Endianess = target.Endianess.Big;**

1. 将语言实现与硬件设备相关联。

**myProc.LanguageImplementations = myCopiedImplementation;**

1. 将 **target.Processor** 对象添加到内部数据库中。

**objectsAdded = target.add(myProc);**

### 通过重用现有实现来创建硬件实现

如果您的硬件设备需要与现有实现相同的硬件实现，您可以重用现有实现。

1. 为新硬件设备创建一个 **target.Processor** 对象。

**myProc = target.create('Processor', ...**

**'Name', 'MyProcessor', ... 'Manufacturer', 'MyManufacturer');**

1. 通过使用设备供应商的标识符和类型检索现有实现，例如 **'ARM Compatible-ARM Cortex'**。

**existingImplementation = target.get('LanguageImplementation', ... 'ARM Compatible-ARM Cortex');**

1. 将语言实现与硬件设备相关联。

**myProc.LanguageImplementations = existingImplementation;**

1. 将 **target.Processor** 对象添加到内部数据库中。

**objectsAdded = target.add(myProc);**

### 验证硬件设备数据

要验证目标对象的数据完整性，请使用 **target.Object** 基类的 **IsValid** 属性或 **validate** 方法。

假设有如下示例，您要在其中创建一个 **target.Processor** 对象，并将某个现有语言实现与该对象相关联。

**myProcessor = target.create('Processor');**

**myProcessor.LanguageImplementations = target.get('LanguageImplementation', ...**

**'ARM Compatible-ARM Cortex');**

要验证创建的对象，请运行 **myProcessor.IsValid** 或 **myProcessor.validate()**。

**myProcessor.IsValid**

**ans = logical 0**

**myProcessor.validate()**

**Error using target.Processor/validate Target data validation failed.**

* **Undefined property "Name" in "Processor" object.**
* **Undefined identifier in "Processor" object.**

验证失败，因为未指定以下 **target.Processor** 属性：

* **Name** - 处理器名称
* **Id** - 对象标识符

您可以指定处理器名称，该名称还指定对象标识符。

**myProcessor.Name = 'MyProcessor';**

检查 **myProcessor** 的有效性。

**myProcessor.IsValid**

**ans = logical 1**

该对象的有效性已建立。

**注意** 当您使用 **target.add** 函数注册目标对象时，软件也会检查该对象的有效性。

### 导出硬件设备数据

您可以在计算机和用户之间共享以前创建的硬件设备数据。对于此示例，指定硬件设备并将其添加到内部数据库中。

**myProc = target.create('Processor', ... 'Name', 'MyProcessor', ...**

**'Manufacturer', 'MyManufacturer'); existingImplementation = target.get('LanguageImplementation', ...**

**'ARM Compatible-ARM Cortex'); myProc.LanguageImplementations = existingImplementation;**

**objectsAdded = target.add(myProc);**

要创建共享硬件设备数据的函数，请运行：

**target.export(myProc, 'FileName', 'exportMyProcFunction')**

**target.export** 函数在当前工作文件夹中创建 **exportMyProcFunction.m**。

**function registeredObjects = exportMyProcFunction(varargin)**

**% This function was generated using target data export.**

**% Create target.Processor "MyManufacturer-MyProcessor" processor = target.create("Processor"); processor.LanguageImplementations(1) = ...**

**target.get("LanguageImplementation", "ARM Compatible-ARM Cortex"); processor.Manufacturer = "MyManufacturer";**

**processor.Name = "MyProcessor";**

**% Add the target objects to MATLAB memory registeredObjects = target.add(processor, varargin{:});**

现在，您可以使用生成的函数跨计算机和用户共享数据库中的硬件设备数据了。例如，在另一台计算机上运行以下命令。

**objectsAdded = exportMyProcFunction;**

生成的函数会重新创建 **target.Processor** 对象 **MyManufacturer-MyProcessor**，并将其添加到内部数据库中。

### 为目标对象创建替代标识符

要为目标对象创建替代标识符，请使用 **target.Alias** 类。

例如，如果 **target.Processor** 对象有较长的类标识符，您可以创建一个 **target.Alias** 对象，该对象可为 **target.Processor** 对象提供一个短标识符。

1. 检索 **target.Processor** 对象。

**proccesorObj = target.get('Processor', ...**

**'Analog Devices-ADSP-CM40x (ARM Cortex-M)');**

1. 使用 **target.create** 函数创建 **target.Alias** 对象。

**aliasProcessorObj = target.create('Alias');**

1. 使用 **target.Alias** 对象属性指定替代标识符和原始目标对象。

**aliasProcessorObj.Name = 'myShortName'; aliasProcessorObj.For = proccesorObj;**

1. 将 **target.Alias** 对象添加到内部数据库中。

**target.add(aliasProcessorObj);**

1. 要检索原始 **target.Processor** 对象，请运行：

**target.get('Processor', 'myShortName');**

### 升级硬件设备的数据定义

要升级通过 **rtwTargetInfo.m** 或 **sl\_customization.m** 文件指定的现有硬件设备定义，请使用

#### target.upgrade 函数。

**rtwTargetInfo.m 文件**

假设您在 **rtwTargetInfo.m** 文件中有硬件设备定义：

**function rtwTargetInfo(tr)**

**% Add registration function handle to the Target Registry tr.registerTargetInfo(@loc\_register\_hardware);**

**end**

**function hw = loc\_register\_hardware hw = RTW.HWDeviceRegistry; hw.Vendor = 'MyManufacturer'; hw.Type = 'MyDevice';**

**hw.Alias = {};**

**hw.Platform = {'Prod', 'Target'}; hw.setWordSizes([8 16 32 64 64 64 64 64 64 64 64]);**

**hw.Endianess = 'Little'; hw.IntDivRoundTo = 'Zero'; hw.ShiftRightIntArith = true; hw.LargestAtomicInteger = 'Long'; hw.LargestAtomicFloat = 'Double';**

**end**

要升级该文件中包含的数据定义，请运行：

**target.upgrade('rtwTargetInfo', 'myPathTo/rtwTargetInfo.m');**

在当前文件夹中，该函数会创建此 **registerUpgradedTargets.m** 文件：

**function processor = registerUpgradedTargets(varargin)**

**% This function was generated using target data export.**

**% Create target.LanguageImplementation 'MyManufacturer-MyDevice' languageimplementation = target.create('LanguageImplementation'); languageimplementation.AtomicFloatSize = 64;**

**languageimplementation.AtomicIntegerSize = 64;**

**languageimplementation.DataTypes.Char.Size = 8;**

**languageimplementation.DataTypes.Double.Size = 64;**

**languageimplementation.DataTypes.Float.Size = 64; languageimplementation.DataTypes.Half.IsSupported = false; languageimplementation.DataTypes.Half.Size = 16;**

**languageimplementation.DataTypes.Int.Size = 32;**

**languageimplementation.DataTypes.Long.Size = 64; languageimplementation.DataTypes.LongLong.IsSupported = false; languageimplementation.DataTypes.LongLong.Size = 64;**

**languageimplementation.DataTypes.Pointer.Size = 64;**

**languageimplementation.DataTypes.PtrDiffT.Size = 64;**

**languageimplementation.DataTypes.Short.Size = 16;**

**languageimplementation.DataTypes.SizeT.Size = 64; languageimplementation.Name = 'MyManufacturer-MyDevice'; languageimplementation.WordSize = 64;**

**% Create target.Processor 'MyManufacturer-MyDevice' processor = target.create('Processor');**

**processor.LanguageImplementations(1) = languageimplementation; processor.Manufacturer = 'MyManufacturer';**

**processor.Name = 'MyDevice';**

**% Add the target objects to MATLAB memory target.add(processor, varargin{:});**

**end**

要向 MATLAB 注册该硬件设备，请运行：

**registerUpgradedTargets()**

如果您需要该注册在各 MATLAB 会话之间持久保留，请运行：

**registerUpgradedTargets('UserInstall', true)**

#### sl\_customization.m 文件

假设您在一个 **sl\_customization.m** 文件中有多个硬件设备定义：

**function sl\_customization(cm)**

**% sl\_customization function to register a device**

**% vendor and type with Simulink.**

**% Copy or rename this file to sl\_customization.m. cm.registerTargetInfo(@loc\_register\_device); cm.registerTargetInfo(@loc\_register\_device2); cm.registerTargetInfo(@loc\_createConfig);**

**cm.registerTargetInfo(@locRegisterTfl); cm.CodeCoverageTools.add('DummyCoverageToolForTesting',...**

**'HDummyCovTool',...**

**'A Coverage Tool Vendor');**

**end**

**function thisDev = loc\_register\_device thisDev = RTW.HWDeviceRegistry; thisDev.Vendor = 'MyDevVendor'; thisDev.Type = 'MyDevType'; thisDev.Alias = {};**

**thisDev.Platform = {'Prod', 'Target'}; thisDev.setWordSizes([8 16 32 32 32]); thisDev.LargestAtomicInteger = 'Char'; thisDev.LargestAtomicFloat = 'None'; thisDev.Endianess = 'Unspecified'; thisDev.IntDivRoundTo = 'Undefined'; thisDev.ShiftRightIntArith = true; thisDev.setEnabled({'IntDivRoundTo'});**

**end**

**function thisDev = loc\_register\_device2 thisDev = RTW.HWDeviceRegistry; thisDev.Vendor = 'MyDevVendor'; thisDev.Type = 'MyDevType2'; thisDev.Alias = {};**

**thisDev.Platform = {'Prod', 'Target'}; thisDev.setWordSizes([8 16 32 32 32]); thisDev.LargestAtomicInteger = 'Char'; thisDev.LargestAtomicFloat = 'None'; thisDev.Endianess = 'Unspecified'; thisDev.IntDivRoundTo = 'Undefined'; thisDev.ShiftRightIntArith = true; thisDev.setEnabled({'IntDivRoundTo'});**

**end**

**% local function**

**function config = loc\_createConfig**

**config = rtw.connectivity.ConfigRegistry; config.ConfigName = 'Infineon->C16x, XC16x'; config.ConfigClass = 'pil\_slcust.HostDemoConfig1'; config.SystemTargetFile = {'custom\_target.tlc'}; config.TemplateMakefile = {'custom\_target.tmf'}; config.TargetHWDeviceType = {'Infineon->C16x, XC16x'};**

**end**

**function thisTfl = locRegisterTfl thisTfl(1) = RTW.TflRegistry; thisTfl(1).Name = 'myTFL1'; thisTfl(1).Description = 'Test'; thisTfl(1).TableList = {'tfl\_table\_Sum',...**

**'tfl\_table\_Product',...**

**}; % Sum includes Add and Subtract thisTfl(1).BaseTfl = 'ANSI\_C'; thisTfl(1).TargetHWDeviceType = {'\*'};**

**end**

要升级该文件中的 **RTW.HWDeviceRegistry** 数据定义，请运行：

**target.upgrade('sl\_customization', 'myPathTo/sl\_customization.m')**

在当前文件夹中，该函数会创建此 **registerUpgradedTargets.m** 文件：

**function targetObjects = registerUpgradedTargets(varargin)**

**% This function was generated using target data export.**

**% Create target.LanguageImplementation 'MyDevVendor-MyDevType' languageimplementation = target.create('LanguageImplementation'); languageimplementation.AtomicIntegerSize = 8;**

**languageimplementation.DataTypes.Char.Size = 8;**

**languageimplementation.DataTypes.Double.Size = 64;**

**languageimplementation.DataTypes.Float.Size = 32; languageimplementation.DataTypes.Half.IsSupported = false; languageimplementation.DataTypes.Half.Size = 16;**

**languageimplementation.DataTypes.Int.Size = 32;**

**languageimplementation.DataTypes.Long.Size = 32; languageimplementation.DataTypes.LongLong.IsSupported = false; languageimplementation.DataTypes.LongLong.Size = 64;**

**languageimplementation.DataTypes.Pointer.Size = 32;**

**languageimplementation.DataTypes.PtrDiffT.Size = 32;**

**languageimplementation.DataTypes.Short.Size = 16;**

**languageimplementation.DataTypes.SizeT.Size = 32; languageimplementation.Endianess = target.Endianess.Unspecified; languageimplementation.Name = 'MyDevVendor-MyDevType'; languageimplementation.WordSize = 32;**

**% Create target.Processor 'MyDevVendor-MyDevType' processor = target.create('Processor');**

**processor.LanguageImplementations(1) = languageimplementation; processor.Manufacturer = 'MyDevVendor';**

**processor.Name = 'MyDevType';**

**% Create target.LanguageImplementation 'MyDevVendor-MyDevType2' languageimplementation2 = target.create('LanguageImplementation'); languageimplementation2.AtomicIntegerSize = 8;**

**languageimplementation2.DataTypes.Char.Size = 8;**

**languageimplementation2.DataTypes.Double.Size = 64;**

**languageimplementation2.DataTypes.Float.Size = 32; languageimplementation2.DataTypes.Half.IsSupported = false; languageimplementation2.DataTypes.Half.Size = 16;**

**languageimplementation2.DataTypes.Int.Size = 32;**

**languageimplementation2.DataTypes.Long.Size = 32; languageimplementation2.DataTypes.LongLong.IsSupported = false; languageimplementation2.DataTypes.LongLong.Size = 64;**

**languageimplementation2.DataTypes.Pointer.Size = 32;**

**languageimplementation2.DataTypes.PtrDiffT.Size = 32;**

**languageimplementation2.DataTypes.Short.Size = 16;**

**languageimplementation2.DataTypes.SizeT.Size = 32; languageimplementation2.Endianess = target.Endianess.Unspecified; languageimplementation2.Name = 'MyDevVendor-MyDevType2'; languageimplementation2.WordSize = 32;**

**% Create target.Processor 'MyDevVendor-MyDevType2' processor2 = target.create('Processor');**

**processor2.LanguageImplementations(1) = languageimplementation2; processor2.Manufacturer = 'MyDevVendor';**

**processor2.Name = 'MyDevType2'; targetObjects = [processor, processor2];**

**% Add the target objects to MATLAB memory target.add(targetObjects, varargin{:});**

**end**

要向 MATLAB 注册硬件设备定义，请运行：

**registerUpgradedTargets()**

如果您需要该注册在各 MATLAB 会话之间持久保留，请运行：

**registerUpgradedTargets('UserInstall', true)**

### 另请参阅

**target.LanguageImplementation** | **target.Processor**

### 详细信息

* [“配置运行时环境选项” （第 10-2 页）](#_bookmark43)

# Simulink Coder 中的模型保护

## 保护模型以隐藏内容

当您要与第三方共享模型而又不能泄露知识产权时，请对模型进行保护。保护模型可以通过将原始模型编译到引用模型中来隐藏原始模型的实现细节。受保护模型包括派生文件，以支持您指定的可选功能。

在您保护模型时，可以允许受保护模型的用户执行以下操作：

* 打开模型的只读 Web 视图，包括模型内容和模块参数。
* 在快速（默认值）、快速加速和普通模式下对模型进行仿真。
* 为包含受保护模型的模型生成代码。
* 为包含受保护模型的模型生成 HDL 代码。
* 如果您有 Embedded Coder 并为模型指定了基于 ERT 的系统目标文件，可通过独立接口为受保护模型生成代码。

您可以选择使用密码保护每个选项。如果您为其中一个选项选择了密码保护，软件将通过使用 AES-256 加密来保护支持文件。

创建受保护模型时：

* Simulink 以与源模型同名的文件创建并存储模型的受保护版本，但文件扩展名为 **.slxp**。
* 扩展名为 **.slx** 的原始模型文件不会更改。如果您通过 Model 模块保护模型，则 Model 模块不会更改。
* （可选）Simulink 创建一个工程存档 (**.mlproj**)，其中包含受保护模型、受保护模型的框架模型及其他支持文件。

此示例说明如何基于引用模型创建受保护模型，以进行只读查看、仿真或代码生成。

### 准备父模型

将父模型中的 Model 模块配置为引用原始的被引用模型。此步骤可防止 Model 模块在您创建受保护模型时成为受保护的引用。

1. 打开引用要保护的模型的父模型。对于此示例，请打开模型 **sldemo\_mdlref\_basic**。
2. 要运行工作流，请创建要保护的 **sldemo\_mdlref\_counter** 模型的本地副本。然后，您可以创建父模型 **sldemo\_mdlref\_basic** 的本地副本。您必须将父模型保存在与引用模型相同的文件夹中。
3. 打开您在本地保存的 **sldemo\_mdlref\_basic** 模型。确保 Model 模块 **CounterA**、**CounterB** 和

**CounterC** 引用您在本地保存的 **sldemo\_mdlref\_counter** 模型。

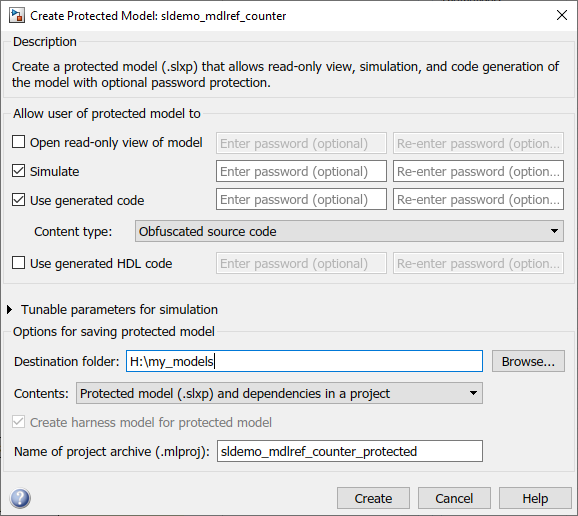
1. 对于每个 Model 模块，打开 Block Parameters 对话框，并在 **Model name** 字段中指定扩展

名 **.slx**。如果该模型和受保护模型在同一个文件夹中，则 **.slxp** 优先于 **.slx**。如果未指定扩展名，则模型中的原始 Model 模块将引用受保护模型，而不是原始模型。点击 **OK**。

### 保护引用的模型

1. 在 **sldemo\_mdlref\_basic** 模型中，点击三个 Model 模块中的任一个。在 Simulink 工具条的

**Model Block** 选项卡上，点击 **Protect**。



1. 在 Create Protected Model 对话框中，选中 **Simulate** 和 **Use generated code** 复选框。这些选项允许受保护模型的用户对引用了受保护模型的模型进行仿真并为其生成代码。如果要为受保护模型的功能设置密码保护，请输入至少包含四个字符的密码。每个选项都可以有唯一密码。
2. 如果您拥有 Embedded Coder，并为模型指定了基于 ERT 的系统目标文件（例如 **ert.tlc**），**Code interface** 字段将处于可见状态。

在此示例中，**sldemo\_mdlref\_basic** 未指定基于 ERT 的系统目标文件，因此 Create Protected Model 对话框中的 **Code interface** 选项不可用。

从 **Code interface** 下拉列表中，选择以下选项之一：

* + “**Model reference**” - 指定通过模型引用代码接口来访问代码，允许在模型引用层次结构中使用受保护模型。受保护模型的用户可以从包含受保护模型的父模型中生成代码。用户可以使用受保护模型运行 Model 模块 SIL/PIL 仿真。
  + “**Top model**” - 指定通过独立接口来访问代码。受保护模型的用户可以使用受保护模型来运行 Model 模块 SIL/PIL 仿真。

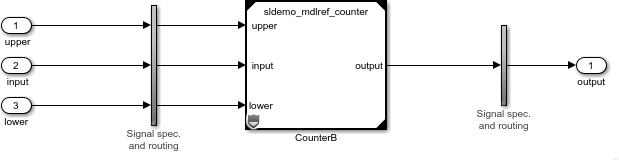
1. 从 **Content type** 列表中选择 “**Obfuscated source code**”，以隐藏受保护模型的源代码目的和逻辑。有关模型保护选项的详细信息，请参阅“Content type”。
2. 展开 **Tunable parameters for simulation** 部分，选择您需要受保护模型用户能够在仿真期间调节的参数。

对于代码生成，具有非自动存储类的参数是可调的。有关详细信息，请参阅“Storage Classes for Parameters and Signals Used in Model Blocks”。

1. 在 **Destination folder** 框中，指定受保护模型的文件夹路径。默认值为当前工作文件夹。
2. 要自动收集、创建和打包受保护模型的支持文件，请将 **Contents** 设置为 “**Protected Model (.slxp) and dependencies in a project**”。

**注意** 在共享工程之前，请检查工程是否包含必要的支持文件。如果缺失支持文件，对相关的框架模型进行仿真或生成代码有助于识别这些文件。将缺失的依存关系添加到工程中，并根据需要更新框架模型。

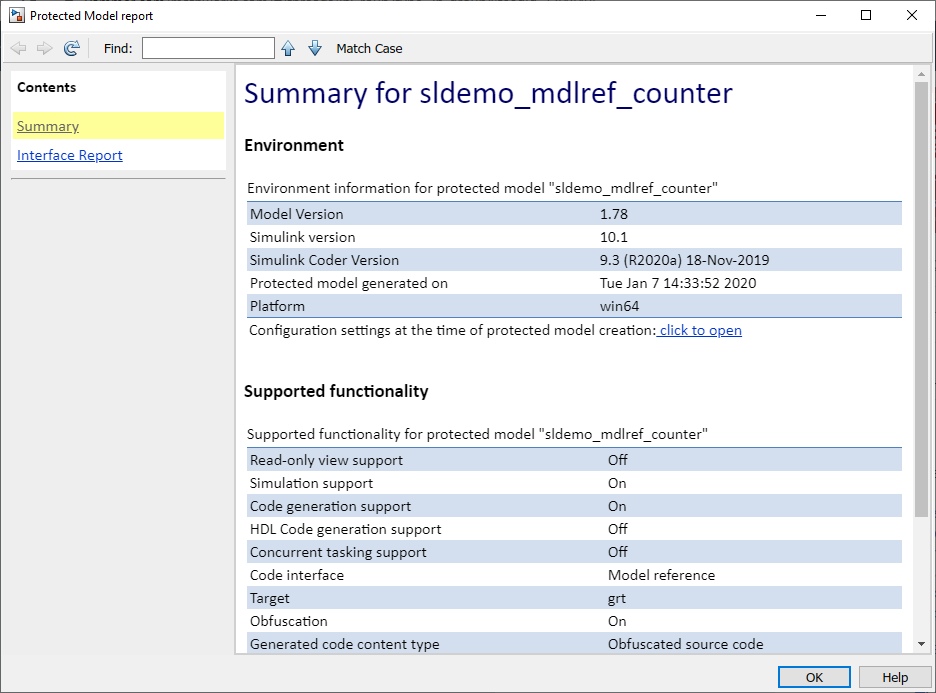
1. 将 **Contents** 设置为 “**Protected Model (.slxp) and dependencies in a project**” 会选中 **Create harness model for protected model** 复选框。框架模型包含在工程中，并为引用受保护模型的 Model 模块提供独立环境。



1. 要进一步自定义受保护模型，您可以：
   * 为代码混淆指定自定义混淆器。请参阅“Specify Custom Obfuscators for Protected Models”。
   * 指定多个代码生成目标。请参阅“Create Protected Models with Multiple Targets”。
   * 为受保护模型定义回调。请参阅“Define Callbacks for Protected Models”。
2. 点击 **Create**。

包含受保护模型、框架模型和支持文件的工程存档 (**.mlproj**) 将出现在指定的目标文件夹中。要测试受保护模型，请通过在当前文件夹浏览器中双击工程存档并指定目标文件夹来提取工程存档的内容。或者，右键点击工程存档，然后点击 **Extract Here**。工程将打开，提取的文件位于以工程存档命名的新文件夹中。

1. 从 Simulink Editor 创建受保护模型时，将生成受保护模型报告，并将其作为受保护模型的一部分。对于此示例，要查看受保护模型报告，请双击受保护模型或右键点击框架模型中模块上的受保护模型徽章图标，然后选择 **Display Report**。



该报告包含：

* + **Summary**，包括以下表：
    - **Environment**，提供 Simulink 版本和其他产品版本，以及用于创建受保护模型的平台。
    - **Supported functionality**，对受保护模型支持的每个可能功能报告 “**On**”、“**Off**” 或 “**On with password protection**”。如果您为多个目标配置受保护模型，此表将包含受支持目标的列表。
  + **Interface Report**，包括模型接口信息，如输入和输出设定、导出的函数信息、接口参数和数据存储。

该报告不包含代码生成报告中有关非受保护模型的报告部分，如代码指标报告和可追溯性报告，因为这些报告包含有关模型设计的详细信息。

1. 您可以测试受保护模型以将其与原始模型进行比较。有关详细信息，请参阅“Test Protected Models”。
2. 要在模型层次结构中使用该受保护模型，请通过 Model 模块（例如框架模型中包含的 Model 模块）引用它。引用受保护模型的 Model 模块的 **Simulation mode** 设置为 “**Accelerator**”。您不能更改此模式。有关详细信息，请参阅“引用来自第三方的受保护模型”。

#### 保护模型的其他方法

要基于当前模型创建受保护模型，请使用以下选项之一：

* + 在 Simulink 工具条的 **Simulation** 选项卡上，点击 **Save** 按钮箭头，然后选择 **Protected Model**。
  + 在 Simulink 工具条 **C Code** 选项卡上，点击 **Share** 按钮箭头，然后选择 **Generate Protected Model**。

要以编程方式创建受保护模型，请使用 **Simulink.ModelReference.protect** 函数。

### 受保护模型的要求和限制

创建受保护模型时，请考虑以下要求：

* + 您必须拥有 Simulink Coder 或 HDL Coder™ 许可证才能创建受保护模型。
  + 模型必须位于 MATLAB 路径上。
  + 模型不能有未保存的更改。
  + 模型使用在保护期间处于激活状态的配置。您无法更改受保护模型的配置。
  + 如果模型包含变体，则受保护模型仅包含在保护期间处于激活状态的变体。
  + 模型保护过程不保留任何回调。有关创建用于受保护模型的回调的详细信息，请参阅“Define Callbacks for Protected Models”。
  + 不要重命名受保护模型或更改其后缀。否则，模型将不可用，直到还原了其原始名称和后缀才可用。
  + 为模型及其引用的模型使用唯一名称。如果某受保护模型所引用模型的名称与另一个受保护模型的名称相同，或与另一个受保护模型的层次结构内的不同模型的名称相同，则使用该受保护模型存在限制。如果某顶层模型引用了存在上述名称冲突的两个受保护模型，则您无法保护该顶层模型、为该顶层模型生成代码或者在软件在环 (SIL)、处理器在环 (PIL) 或快速加速模式下仿真顶层模型。
  + 如果您的模型包含非内联的 S-Function，请对 .MEX 文件使用唯一名称。如果名称与 MATLAB 路径中的其他 .MEX 文件冲突，您无法仿真受保护模型。

该模型还必须满足“模型引用的要求和限制”中列出的要求。

### 代码生成要求和限制

要创建支持代码生成的受保护模型，您的模型必须满足以下要求：

* + 受保护模型必须使用普通、快速、软件在环 (SIL) 或处理器在环 (PIL) 模式和单个目标。
  + 受保护模型不能包含非内联的 S-Function。
  + 不要选中 **Code Generation > Verification > Measure function execution time** 复选框。如果在保护模型时选中了此选项，软件将关闭参数并显示警告。
  + 受保护引用模型必须支持代码生成且没有密码保护。
  + 受保护模型必须与每个受保护引用模型的 **Content type** 兼容。下表提供了兼容性信息。

#### 代码生成内容类型兼容性

|  |  |
| --- | --- |
| **受保护父模型内容类型** | **兼容的受保护引用模型内容类型** |
| “**Binaries**” | * “**Binaries**” * “**Obfuscated source code**” |
| “**Binaries**”（**'ObfuscateCode'** 设置为  **'false'**） | * “**Binaries**” * “**Binaries**”（**'ObfuscateCode'** 设置为   **'false'**）   * “**Obfuscated source code**” * “**Readable source code**” |
| “**Obfuscated source code**” | * “**Obfuscated source code**” |
| “**Readable source code**” | * “**Obfuscated source code**” * “**Readable source code**” |

当受保护父模型和受保护引用模型的 **Content type** 不匹配时，代码生成会应用它们之中保护级别较高的 **Content type**。例如，如果受保护父模型设置为 “**Binaries**” 类型，而受保护引用模型设置为 “**Obfuscated source code**” 类型，则该受保护引用模型会生成为 “**Binaries**” 类型的代码。如果受保护父模型设置为 “**Readable source code**” 类型，而受保护引用模型设置为 “**Obfuscated source code**” 类型，则该受保护引用模型会生成为 “**Obfuscated source code**” 类型的代码。

要避免在包含受保护模型的模型的代码生成过程中出现错误，请遵循下列各项：

* + 受保护模型名称必须与同一模型引用层次结构中的其他模型名称不同。
  + 接口必须匹配。
  + 参数必须兼容。

### 嵌套受保护模型的要求和限制

要使受保护模型的接收者能够保护引用它的模型，受保护模型必须：

* + 支持加速模式。您必须在 Create Protected Model 对话框中选择 **Simulate**，或者使用 **Simulink.ModelReference.protect** 函数将 **'Mode'** 设置为 **'Accelerator'** 或 **'CodeGeneration'**。
  + 使用默认的 **'CodeInterface'** 设置：**'Model reference'**。
  + 对仿真不使用密码保护。
  + 没有回调。
  + 在没有密码保护的情况下支持受保护的父模型要支持的操作。
  + 使用与受保护父模型的预期 **Content type** 兼容的 **Content type**（如果受保护父模型预期支持代码生成）。有关兼容的 **Content type** 选项的列表，请参阅上一节中的表。
  + 使用受保护的父模型预期要使用的编译器。
  + 引用模型的名称不能与另一个受保护模型的名称相同，或与另一个受保护模型的层次结构内的不同模型的名称相同。

如果要保护的模型引用受保护模型，被引用的受保护模型必须满足前面的要求。

要保护一个引用了受保护模型的模型，必须将被引用的受保护模型的可调参数指定为对正在保护的模型是可调的。

**注意** 如果您在选择了 **Use generated HDL code** 选项的情况下创建了一个受保护模型，则支持对引用了该受保护模型的模型进行仿真和 HDL 代码生成，并可对仿真和 HDL 代码生成进行加密。在使用此选项 时，您不能对 HDL 源代码进行混淆处理、使用回调或使用嵌套的受保护模型。要了解有关 HDL 代码生成

限制的更多信息，请参阅 “Protected Model Restrictions for HDL Code Generation” (HDL Coder)。

### 另请参阅

**详细信息**

* + “Package and Share Protected Models”
  + “Specify Custom Obfuscators for Protected Models”
  + “Create Protected Models with Multiple Targets”
  + “Define Callbacks for Protected Models”

# Simulink Coder 中的组件初始化、重置和终止

# Simulink Coder 中的 Stateflow 模块

# Simulink Coder 的模块编写和代码生成

* [“S-Function 和代码生成” （第 14-2 页）](#_bookmark75)
* [“使用代码继承工具在生成的代码中导入对外部代码的调用” （第 14-5 页）](#_bookmark79)
* [“集成以定点数据类型的形参方式传递输入和输出实参的外部 C 函数” （第 14-9 页）](#_bookmark86)
* [“集成实现 N 维表查找的外部 C 函数” （第 14-12 页）](#_bookmark87)
* [“集成外部 C++ 对象方法” （第 14-15 页）](#_bookmark88)
* [“编写非内联 S-Function” （第 14-18 页）](#_bookmark89)
* [“编写包装器 S-Function 和 TLC 文件” （第 14-20 页）](#_bookmark92)
* [“编写完全内联的 S-Function” （第 14-27 页）](#_bookmark98)

## S-Function 和代码生成

您可以使用 S-Function 扩展 Simulink 对仿真和代码生成的支持。例如，您可以使用它们：

* 表示自定义算法
* 将现有外部代码对接到 Simulink 模型和生成的代码中
* 表示与硬件对接的设备驱动程序
* 为嵌入式系统生成高度优化的代码
* 在 Simulink 仿真过程中验证为子系统生成的代码

通过 S-Function 的应用程序编程接口 (API)，您可以非常灵活地在 Simulink 环境中实现通用算法。如果您打算在模型中将 S-Function 用于代码生成，灵活度可能会有所不同。例如，需要生成代码的 S- Function 不能访问 MATLAB 工作区。本主题介绍使用 S-Function 需要注意的条件。但是，利用本主题介绍的技巧，可为大多数使用生成代码的应用程序创建 S-Function。

虽然 S-Function 提供了通用且灵活的解决方案，可在模型中实现复杂的算法，但基础 API 会在内存和计算资源方面产生一定的开销。通常，额外增加的资源开销对实时快速原型系统来说是可以接受的。不过，通常情况下实时嵌入式应用程序中并没有额外的可用资源。通过使用代码生成器附带的目标语言编译器技术来内联 S-Function，可最大程度降低内存和计算要求。如果您为现有的外部代码生成 S-Function，可以考虑使用代码继承工具生成 S-Function 和相关的 TLC 文件。

本主题假设您已理解以下概念：

* 2 级 S-Function
* 目标语言编译器 (TLC) 脚本编写
* 代码生成器如何生成和编译 C/C++ 代码

**注释** 此信息适用于代码生成器用户。即使您当前没有使用代码生成器，编写 S-Function 时也可以采用这些做法，尤其是在创建通用 S-Function 时。

### S-Function 的类型

以下情形适合用 S-Function 来实现仿真和代码生成：

1. “我不关心效率，我只想让我的算法能够自动在 Simulink 和代码生成器产品中工作。”
2. “我想在 Simulink 和代码生成器产品中实现一种高度优化的算法，它看起来就像是一个内置模块，并且能生成高效的代码。”
3. “我有很多人工代码需要集成。我希望高效地从 Simulink 和代码生成器产品中调用我的函数。”上述情况分别对应于以下 MathWorks 术语：
4. 非内联 S-Function
5. 内联 S-Function
6. 为外部代码自动生成的 S-Function

#### 非内联 S-Function

非内联 S-Function 是 C 或 C++ MEX S-Function，Simulink 引擎和生成的代码以相同的方式处理这种函数。一般情况下，您只需根据 S-Function API 实现一次算法即可。Simulink 引擎和生成的代码将在模型执行过程中调用 S-Function 例程（例如 **mdlOutputs**）。

非内联 S-Function 模块的每个实例都需要一些额外的内存和计算资源。但是，这种在模型和代码生成应用程序中嵌入算法的例程通常发生在工程的原型构建阶段，此时效率并不重要。牺牲效率所获得的好处是能够快速更改模型参数和结构。

编写非内联 S-Function 不涉及 TLC 编码。您在模型中内建一个 MEX S-Function 后，便可以直接按下 **Ctrl+B** 开始编译模型，而无需进行其他准备工作，从这个意义上讲，非内联 S-Function 是编译过程的默认情形。

生成联编文件时，关于非内联 S-Function [文件的名称和位置有一些限制。请参阅“编写非内联 S-](#_bookmark89)

[Function” （第 14-18](#_bookmark89) 页）。

#### 内联 S-Function

要使 S-Function 能够在 Simulink 环境中工作，需要生成一些开销代码。当代码生成器从包含 S- Function（不含 **sfunction.tlc** 文件）的模型中生成代码时，会在生成的代码中嵌入一些开销代码。如果要优化您的实时代码并消除一些开销代码，必须内联（或嵌入）您的 S-Function。这涉及到编写一个 TLC (**sfunction.tlc**) 文件，以便从生成的代码中消除开销代码。目标语言编译器通过处理 **sfunction.tlc** 文件来定义如何在生成的代码中内联您的 S-Function 算法。

**注意** 不要将内联一词与 C++ 语言中的 inline 关键字混淆。内联的意思是：通过指定文本来代替对通用 S- Function API 例程（如 **mdlOutputs**）的调用。例如，使用 TLC 文件内联 S-Function 时，生成的代码 将包含通常出现在 S-Function 例程中的 C/C++ 代码，而 S-Function 本身已从编译过程中删除。

完全内联的 S-Function 采用与内置模块相同的方式将算法（模块）嵌入到生成的代码中。通常，完全内联的 S-Function 需要实现两次算法：一次针对 Simulink 模型 (C/C++ MEX S-Function)，一次针对代码生成（TLC 文件）。TLC 文件的复杂程度取决于算法的复杂程度以及您要在生成的代码中实现的效率水平。 TLC 文件的结构具有不同的复杂程度。请参阅“Inlining S-Functions”。

#### 为原有代码或自定义代码自动生成的 S-Function

如果您需要在模型中调用手写的 C/C++ 代码，可以考虑使用 Simulink 代码继承工具。代码继承工具可以根据您在代码继承工具数据结构体中注册的信息自动生成完全内联的 S-Function 和对应的 TLC 文件。

有关详细信息，请参阅“Integrate C Functions Using Legacy Code Tool” [和“使用代码继承工具在生成的代码中导入对外部代码的调用” （第 14-5](#_bookmark79) 页）。

### 实现非内联和内联 S-Function 需要的文件

本主题简要介绍创建非内联和内联 S-Function 需要的文件和函数。

* 非内联 S-Function 需要 C 或 C++ MEX S-Function 源代码（**sfunction.c** 或 **sfunction.cpp**）。
* 完全内联的 S-Function 需要一个 **sfunction.tlc** 文件。完全内联的 S-Function 可为参数化 S- Function 生成最优代码。参数化 S-Function 是在特定模式下运行、依赖于固定的 S-Function 参数

（这些参数在模型执行过程中不会改变）的 S-Function。对于给定的操作模式，**sfunction.tlc** 文件指定为该模式实现算法而生成的确切代码。例如，“Write Fully Inlined S-Functions with mdlRTW

Routine”中的直接索引查找表 S-Function 包含两种操作模式 - 一种用于等间距 **x-data**，另一种用于非等间距 **x-data**。

**注意** 为调用原有代码或自定义 C/C++ 代码而生成的完全内联的 S-Function 还需要一个

**sfunction.tlc** 文件，此文件由代码继承工具生成。

完全内联的 S-Function 可能需要在 S-Function MEX 文件 **sfunction.c** 或 **sfunction.cpp** 中放置 **mdlRTW** 例程。**mdlRTW** 例程允许您将信息放在 **model.rtw** 中。**model.rtw** 是用来指定模型的记录文件，生成代码时，代码生成器将在执行 **sfunction.tlc** 之前调用目标语言编译器来处理此文件。

当您要在 TLC 文件中引入不可调参数时，包含一个 **mdlRTW** 例程很有用。此类参数用于确定在 S- Function 的给定实例中，哪个操作模式处于活动状态。基于此信息，S-Function 的 TLC 文件可为该操作模式生成高效的最优代码。

### 编写支持代码生成的 S-Function 的规范

* 可 C/C++ MEX、MATLAB 语言和 Fortran MEX S-Function 用于代码生成。
* 可通过提供内联 TLC 文件来内联 S-Function 进行代码生成。请参阅“Target Language Compiler”中的 S-Function 内联。MATLAB 和 Fortran MEX S-Function 必须是内联的。可以内联 C/C++ MEX S-Function 以提高代码效率，也可以不内联。
* 要自动生成完全内联的 C MEX S-Function 以调用现有代码或自定义代码，请使用代码继承工具。有关详细信息，请参阅“Integrate C Functions Using Legacy Code Tool” [和“使用代码继承工具在生成的代码中导入对外部代码的调用” （第 14-5](#_bookmark79) 页）。
* 如果代码效率不是最需要考虑的因素，例如您要快速进行原型构建，则可以选择不内联 C/C++ MEX S-Function。有关详细信息，请参阅“编写非内联 [S-Function” （第 14-18](#_bookmark89) 页）。

### 另请参阅详细信息

* “使用代码继承工具在生成的代码中导入对外部代码的调用” (Embedded Coder)

## 使用代码继承工具在生成的代码中导入对外部代码的调用

[“代码继承工具和代码生成” （第 14-5 页）](#_bookmark80)

[“生成内联 S-Function 文件以进行代码生成” （第 14-5 页）](#_bookmark81) [“对已有函数应用代码样式设置” （第 14-6 页）](#_bookmark82) [“解决对不同位置的文件的依存关系” （第 14-6 页）](#_bookmark83)

[“部署 S-Function 以进行仿真和代码生成” （第 14-7 页）](#_bookmark84) [“集成外部 C++ 对象” （第 14-7 页）](#_bookmark85)

**本节内容**

### 代码继承工具和代码生成

您可以使用 Simulink 代码继承工具为现有代码或自定义代码生成完全内联的 C MEX S-Function。S-

Function 针对设备驱动程序和查找表等嵌入式组件进行了优化，可以调用现有的 C 或 C++ 函数。

**注意** 代码继承工具可与 C++ 函数对接，但不能与 C++ 对象对接。要解决此问题，使该工具能够与 C++对象对接，请参阅“Legacy Code Tool Limitations”。

可以使用该工具：

* 编译和构造生成的 S-Function 以进行仿真。
* 生成封装的 S-Function 模块，此模块被配置为调用现有的外部代码。

如果要在打算为其生成代码的模型中包含这些类型的 S-Function，请使用该工具生成 TLC 模块文件。TLC模块文件指定为模型生成的代码如何调用现有的 C 或 C++ 函数。

如果 S-Function 依赖的是其他文件夹中的文件，而不是包含 S-Function 动态加载的可执行文件的文件夹中的文件，请使用该工具为 S-Function 生成 **sFunction\_makecfg.m** 或 **rtwmakecfg.m** 文件。生成这样的文件可在编译包含 S-Function 的模型时保持这些依存关系。例如，对于某些应用程序（如自定义目 标），您可能希望在目标特定的位置找到文件。编译过程将在与 S-Function 动态加载可执行文件相同的文件夹中查找 **sFunction\_makecfg.m** 或 **rtwmakecfg.m**，并调用此文件中的函数。

有关详细信息，请参阅“Integrate C Functions Using Legacy Code Tool”。

### 生成内联 S-Function 文件以进行代码生成

要为使用 S-Function 的模型生成代码，请根据应用程序的代码生成要求执行以下操作之一：

* 为内联 S-Function 生成一个 **.cpp** 文件。在从现有 C 函数生成 S-Function 源文件之前，请在代码继承工具数据结构体中将 **Options.singleCPPMexFile** 字段的值设置为 **true**。例如：

**def.Options.singleCPPMexFile = true; legacy\_code('sfcn\_cmex\_generate', def);**

* 为内联 S-Function 生成源文件和 TLC 模块文件。例如：

**def.Options.singleCPPMexFile = false; legacy\_code('sfcn\_cmex\_generate', def); legacy\_code('sfcn\_tlc\_generate', def);**

#### 有关 singleCPPMexFile 的限制

在下列情况下，不能将 **singleCPPMexFile** 字段设置为 **true**：

#### Options.language='C++'

* 您使用以下 Simulink 对象之一并将 **IsAlias** 属性设置为 **true**：
  + **Simulink.Bus**
  + **Simulink.AliasType**
  + **Simulink.NumericType**
* 代码继承工具函数规范中包含 **void\*** 或 **void\*\***，以表示状态参数的标量工作数据
* 代码继承工具结构体的 **HeaderFiles** 字段指定多个头文件

### 对已有函数应用代码样式设置

要将代码样式的模型配置参数应用于已有函数，请执行以下操作：

1. 初始化代码继承工具数据结构体。例如：

**def = legacy\_code('initialize');**

1. 在数据结构体中，将 **Options.singleCPPMexFile** 字段的值设置为 **true**。例如：

**def.Options.singleCPPMexFile = true;**

要检查设置，请输入：

**def.Options.singleCPPMexFile**

#### 有关 singleCPPMexFile 的限制

在下列情况下，不能将 **singleCPPMexFile** 字段设置为 **true**：

#### Options.language='C++'

* 您使用以下 Simulink 对象之一并将 **IsAlias** 属性设置为 **true**：
  + **Simulink.Bus**
  + **Simulink.AliasType**
  + **Simulink.NumericType**
* 代码继承工具函数规范中包含 **void\*** 或 **void\*\***，以表示状态参数的标量工作数据
* 代码继承工具结构体的 **HeaderFiles** 字段指定多个头文件

### 解决对不同位置的文件的依存关系

默认情况下，代码继承工具假定 S-Function 所依赖的文件与 S-Function 的动态加载可执行文件位于同一个文件夹中。如果您的 S-Function 所依赖的文件位于其他位置，而您使用的是模板联编文件编译过程，请为 S-Function 生成一个 **sFunction\_makecfg.m** 或 **rtwmakecfg.m** 文件。例如，如果您的代码继承工具数据结构体将编译资源定义为路径名称，则您可能需要生成此文件。

要生成 **sFunction\_makecfg.m** 或 **rtwmakecfg.m** 文件，请调用 **legacy\_code** 函数，并将 **'sfcn\_makecfg\_generate'** 或 **'rtwmakecfg\_generate'** 作为第一个参数，将代码继承工具数据结构体的名称作为第二个参数。例如：

**legacy\_code('sfcn\_makecfg\_generate', lct\_spec);**

如果您使用同一个文件夹中的多个注册文件，并通过调用一次 **legacy\_code** 为每个文件生成一个 S- Function，则指定 **'sfcn\_makecfg\_generate'** 或 **'rtwmakecfg\_generate'** 的 **legacy\_code** 调用必须对所有注册文件都是通用的。有关详细信息，请参阅“Handling Multiple Registration Files”。

例如，如果将 **defs** 定义为代码继承工具结构体数组，则可以使用 **'sfcn\_makecfg\_generate'** 调用一次

#### legacy\_code。

**defs = [defs1(:);defs2(:);defs3(:)]; legacy\_code('sfcn\_makecfg\_generate', defs);**

有关详细信息，请参阅“编译对 [S-Function 的支持” （第 30-11](#_bookmark248) 页）。

### 部署 S-Function 以进行仿真和代码生成

您可以部署使用代码继承工具生成的 S-Function，以便其他人也能使用它们。要部署 S-Function 进行仿真和代码生成，请共享以下文件：

* 注册文件
* 已编译的动态加载可执行文件
* TLC 模块文件

#### sFunction\_makecfg.m 或 rtwmakecfg.m 文件

* 生成的 S-Function 所依赖的头文件、源文件和 include 文件

使用这些部署的文件时：

* 在 Simulink 模型中使用部署的文件之前，请将包含 S-Function 文件的文件夹添加到 MATLAB 路径中。
* 如果代码继承工具数据结构体将所需的文件注册为绝对路径，而文件位置发生改变，请重新生成

**sFunction\_makecfg.m** 或 **rtwmakecfg.m** 文件。

### 集成外部 C++ 对象

代码继承工具可与 C++ 函数对接，但不能与 C++ 对象对接。以上一个示例为基础，下面的示例说明如何应对这一限制。

* 在新文件 **adder\_cpp.hpp** 中修改 **adder** 的类定义。添加三个新的宏，分别用于动态分配新的 **adder**对象，调用 **add\_one()** 方法和释放分配的内存。每个宏通过一个指针指向一个 **adder** 对象。因为代 码继承工具调用的每个函数必须具有一个与 C 语言类似的签名，所以该指针以 **void\*** 的形式进行缓存和传递。因此，您必须在宏中明确转换为 **adder\***。**adder** 的新类定义：

**#ifndef \_ADDER\_CPP\_ #define \_ADDER\_CPP\_**

**class adder { private:**

**int int\_state; public:**

**adder(): int\_state(0) {};**

**int add\_one(int increment);**

**int get\_val() {return int\_state;};**

**};**

**// Method wrappers implemented as macros #define createAdder(work1) \**

**\*(work1) = new adder**

**#define deleteAdder(work1) \ delete(static\_cast<adder\*>(\*(work1)))**

**#define adderOutput(work1, u1) \ (static\_cast<adder\*> ((work1)))->add\_one(u1)**

**#endif /\* \_ADDER\_CPP\_ \*/**

* 更新 **adder\_cpp.cpp**。采用类修改方式（而不是一个全局实例），生成的每个 S-Function 可以管理其自己的 **adder** 对象。

**#include "adder\_cpp.hpp"**

**int adder::add\_one(int increment)**

**{**

**int\_state += increment; return int\_state;**

**}**

* 使用以下更改更新 **rtwdemo\_sfun\_adder\_cpp.cpp**：
  + **StartFcnSpec** 调用的宏负责分配新的 **adder** 对象并缓存指针。

**def.StartFcnSpec = 'createAdder(void \*\*work1)';**

* + **OutputFcnSpec** 调用的宏负责调用 **add\_one()** 方法并提供 S-Function 特定的 **adder** 指针对象。

**def.OutputFcnSpec = 'int32 y1 = adderOutput(void \*work1, int32 u1)';**

* + **TerminateFcnSpec** 调用的宏负责释放内存。

**def.TerminateFcnSpec = 'deleteAdder(void \*\*work1)';**

### 另请参阅

**legacy\_code**

### 相关示例

* “Integrate C Functions Using Legacy Code Tool”
* “Legacy Code Tool Examples”
* “从模型和生成的代码调用外部 C 代码” (Embedded Coder)

## 集成以定点数据类型的形参方式传递输入和输出实参的外部 C 函数

使用代码继承工具，集成通过使用定点数据类型的参数来传递输入和输出的已有 C 函数。使用代码继承工具，您可以：

* 提供已有函数设定。
* 生成在仿真过程中调用现有代码的 C-MEX S-Function。
* 编译生成的 S-Function 以进行仿真。
* 生成 TLC 模块文件和可选的 rtwmakecfg.m 文件，该文件指定针对模型生成的代码如何调用现有代码。

#### 提供已有函数设定

代码继承工具函数接受特定的数据结构体或结构体数组作为参数。您可以通过调用函数 legacy\_code() 并使用 'initialize' 作为第一个输入来初始化数据结构体。初始化结构体之后，将其属性指定给与要集成的现有代码对应的值。此示例中调用的已有函数的原型为：

myFixpt timesS16(const myFixpt in1, const myFixpt in2, const uint8\_T fracLength) myFixpt 逻辑上是定点数据类型，实际上是 16 位整数的 typedef：

**myFixpt = Simulink.NumericType;**

**myFixpt.DataTypeMode = 'Fixed-point: binary point scaling'; myFixpt.Signed = true;**

**myFixpt.WordLength = 16;**

**myFixpt.FractionLength = 10; myFixpt.IsAlias = true; myFixpt.HeaderFile = 'timesFixpt.h';**

已有源代码位于文件 timesFixpt.h 和 timesS16.c 中。

**% rtwdemo\_sfun\_gain\_fixpt def = legacy\_code('initialize');**

**def.SFunctionName = 'rtwdemo\_sfun\_gain\_fixpt';**

**def.OutputFcnSpec = 'myFixpt y1 = timesS16(myFixpt u1, myFixpt p1, uint8 p2)'; def.HeaderFiles = {'timesFixpt.h'};**

**def.SourceFiles = {'timesS16.c'}; def.IncPaths = {'rtwdemo\_lct\_src'}; def.SrcPaths = {'rtwdemo\_lct\_src'};**

#### 生成 S-Function 进行仿真

要根据输入参数 'def' 提供的描述生成 C-MEX S-Function，请再次调用函数 legacy\_code() 并将第一个输入设置为 'sfcn\_cmex\_generate'。S-Function 将在仿真过程中调用已有函数。S-Function 的源代码位于文件 rtwdemo\_sfun\_gain\_fixpt.c 中。

**legacy\_code('sfcn\_cmex\_generate', def);**

#### 编译生成的 S-Function 进行仿真

在生成 C-MEX S-Function 源文件之后，要使用 Simulink® 编译 S-Function 进行仿真，请再次调用函数 legacy\_code() 并将第一个输入设置为 'compile'。

**legacy\_code('compile', def);**

**### Start Compiling rtwdemo\_sfun\_gain\_fixpt**

**mex('-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtwdemo\_lct\_src', '-I/tmp/Bdoc23a\_2 Building with 'gcc'.**

**MEX completed successfully.**

**mex('rtwdemo\_sfun\_gain\_fixpt.c', '-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtwdem Building with 'gcc'.**

**MEX completed successfully.**

**### Finish Compiling rtwdemo\_sfun\_gain\_fixpt ### Exit**

#### 生成 TLC 模块文件以进行代码生成

编译 S-Function 并将其用于仿真之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'sfcn\_tlc\_generate'，以生成 TLC 模块文件。该模块文件指定为模型生成的代码如何调用现有代码。如果您不生成 TLC 模块文件，则当您尝试为包含 S-Function 的模型生成代码时，代码生成将失败。S-

Function 的 TLC 模块文件为 rtwdemo\_sfun\_gain\_fixpt.tlc。

**legacy\_code('sfcn\_tlc\_generate', def);**

#### 生成 rtwmakecfg.m 文件以进行代码生成

创建 TLC 模块文件之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'rtwmakecfg\_generate'，以生成支持代码生成的 rtwmakecfg.m 文件。如果 S-Function 需要的源文件 和头文件与 S-Function 不在同一个文件夹中，而您要将这些相关文件添加到代码生成过程中生成的联编文件中，请生成 rtwmakecfg.m 文件。

**legacy\_code('rtwmakecfg\_generate', def);**

#### 生成封装的 S-Function 模块以调用生成的 S-Function

在编译 C-MEX S-Function 源之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'slblock\_generate'，以生成封装的 S-Function 模块来调用该 S-Function。软件将此模块放在新模型中。您可以将此模块复制到现有模型中。

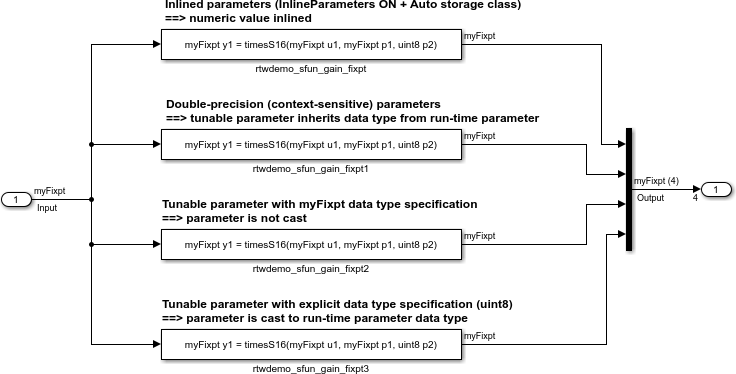
**legacy\_code('slblock\_generate', def);**



#### 显示生成的与现有代码的集成

模型 rtwdemo\_lct\_fixpt\_params 显示了模型与现有代码的集成。子系统 TestFixpt 的作用相当于通过生成的 S-Function 调用已有 C 函数的工具。

**open\_system('rtwdemo\_lct\_fixpt\_params') open\_system('rtwdemo\_lct\_fixpt\_params/TestFixpt') sim('rtwdemo\_lct\_fixpt\_params')**



### 另请参阅

**legacy\_code**

### 相关示例

* “Integrate C Functions Using Legacy Code Tool”
* “从模型和生成的代码调用外部 C 代码” (Embedded Coder)

## 集成实现 N 维表查找的外部 C 函数

通过使用代码继承工具，集成实现 N 维表查找的已有 C 函数。使用代码继承工具，您可以：

* 提供已有函数设定。
* 生成在仿真过程中调用现有代码的 C-MEX S-Function。
* 编译生成的 S-Function 以进行仿真。
* 生成 TLC 模块文件和可选的 rtwmakecfg.m 文件，该文件指定针对模型生成的代码如何调用现有代码。

#### 提供已有函数设定

代码继承工具函数接受特定的数据结构体或结构体数组作为参数。您可以通过调用函数 legacy\_code() 并使用 'initialize' 作为第一个输入来初始化数据结构体。初始化结构体之后，将其属性指定给与要集成的现有代码对应的值。此示例中调用的已有函数的原型为：

FLT directLookupTableND(const FLT \*tableND, const UINT32 nbDims, const UINT32 \*tableDims, const UINT32 \*tableIdx)

FLT 是浮点类型的 typedef，UINT32 是无符号 int32 类型的 typedef。已有源代码位于文件 your\_types.h、lookupTable.h 和 directLookupTableND.c 中。

**defs = [];**

**evalin('base','load rtwdemo\_lct\_data.mat')**

**% rtwdemo\_sfun\_dlut3D**

**def = legacy\_code('initialize'); def.SFunctionName = 'rtwdemo\_sfun\_dlut3D';**

**def.OutputFcnSpec = 'single y1 = DirectLookupTable3D(single p1[][][], uint32 p2[3], uint32 u1[3])'; def.HeaderFiles = {'lookupTable.h'};**

**def.SourceFiles = {'directLookupTableND.c'}; def.IncPaths = {'rtwdemo\_lct\_src'}; def.SrcPaths = {'rtwdemo\_lct\_src'};**

**defs = [defs; def];**

**% rtwdemo\_sfun\_dlut4D**

**def = legacy\_code('initialize'); def.SFunctionName = 'rtwdemo\_sfun\_dlut4D';**

**def.OutputFcnSpec = 'single y1 = DirectLookupTable4D(single p1[][][][], uint32 p2[4], uint32 u1[4])'; def.HeaderFiles = {'lookupTable.h'};**

**def.SourceFiles = {'directLookupTableND.c'}; def.IncPaths = {'rtwdemo\_lct\_src'}; def.SrcPaths = {'rtwdemo\_lct\_src'};**

**defs = [defs; def];**

#### 生成 S-Function 以进行仿真

要根据输入参数 'defs' 提供的描述生成 C-MEX S-Function，请再次调用函数 legacy\_code()。将第一个输入设置为 'sfcn\_cmex\_generate'。S-Function 将在仿真过程中调用已有函数。S-Function 的源代码位于文件 rtwdemo\_sfun\_dlut3D.c 和 rtwdemo\_sfun\_dlut4D.c 中。

**legacy\_code('sfcn\_cmex\_generate', defs);**

#### 编译生成的 S-Function 以进行仿真

在生成 C-MEX S-Function 源文件之后，要使用 Simulink® 编译 S-Function 进行仿真，请再次调用函数 legacy\_code()。将第一个输入设置为 'compile'。

**legacy\_code('compile', defs);**

**### Start Compiling rtwdemo\_sfun\_dlut3D**

**mex('-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtwdemo\_lct\_src', '-I/tmp/Bdoc23a\_2 Building with 'gcc'.**

**MEX completed successfully.**

**mex('rtwdemo\_sfun\_dlut3D.c', '-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtwdemo\_l Building with 'gcc'.**

**MEX completed successfully.**

**### Finish Compiling rtwdemo\_sfun\_dlut3D ### Exit**

**### Start Compiling rtwdemo\_sfun\_dlut4D**

**mex('-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtwdemo\_lct\_src', '-I/tmp/Bdoc23a\_2 Building with 'gcc'.**

**MEX completed successfully.**

**mex('rtwdemo\_sfun\_dlut4D.c', '-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtwdemo\_l Building with 'gcc'.**

**MEX completed successfully.**

**### Finish Compiling rtwdemo\_sfun\_dlut4D ### Exit**

#### 生成 TLC 模块文件以进行代码生成

编译 S-Function 并将其用于仿真之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'sfcn\_tlc\_generate' 以生成 TLC 模块文件。模块文件指定为模型生成的代码如何调用现有代码。如果您不生成 TLC 模块文件，则当您尝试为包含 S-Function 的模型生成代码时，代码生成将失败。S-Function 的 TLC 模块文件为 rtwdemo\_sfun\_dlut3D.tlc 和 rtwdemo\_sfun\_dlut4D.tlc。

**legacy\_code('sfcn\_tlc\_generate', defs);**

#### 生成 rtwmakecfg.m 文件以进行代码生成

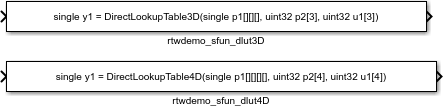
创建 TLC 模块文件之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'rtwmakecfg\_generate'，以生成支持代码生成的 rtwmakecfg.m 文件。如果 S-Function 需要的源文件 和头文件与 S-Function 不在同一个文件夹中，而您要将这些相关文件添加到代码生成过程中生成的联编文件中，请生成 rtwmakecfg.m 文件。

**legacy\_code('rtwmakecfg\_generate', defs);**

#### 生成封装的 S-Function 模块以调用生成的 S-Function

在编译 C-MEX S-Function 源之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'slblock\_generate'，以生成封装的 S-Function 模块来调用 S-Function。软件将模块放在新模型中。您可以将模块复制到现有模型中。

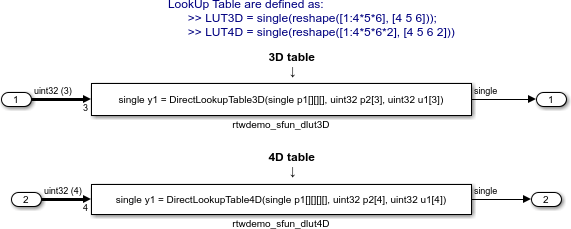
**legacy\_code('slblock\_generate', defs);**



#### 显示生成的与现有代码的集成

模型 rtwdemo\_lct\_lut 显示了模型与现有代码的集成。子系统 TestFixpt 的作用相当于调用已有 C 函数的工具，Display 模块将函数输出与内置 Simulink® 查找模块的输出进行比较。结果相同。

**open\_system('rtwdemo\_lct\_lut') open\_system('rtwdemo\_lct\_lut/TestLut1') sim('rtwdemo\_lct\_lut')**



### 另请参阅

**legacy\_code**

### 相关示例

* “Integrate C Functions Using Legacy Code Tool”
* “从模型和生成的代码调用外部 C 代码” (Embedded Coder)

## 集成外部 C++ 对象方法

通过使用代码继承工具，集成已有的 C++ 对象方法。使用代码继承工具，您可以：

* 提供已有函数设定。
* 生成在仿真过程中调用现有代码的 C++ MEX S-Function。
* 编译生成的 S-Function 以进行仿真。
* 生成模块 TLC 文件和可选的 rtwmakecfg.m 文件，用于在代码生成过程中调用现有代码。

#### 提供已有函数设定

代码继承工具附带的函数接受特定的数据结构体或结构体数组作为参数。可以通过调用函数 legacy\_code() 并使用 'initialize' 作为第一个输入来初始化数据结构体。初始化结构体之后，将其属性指定给与要集成的现有代码对应的值。此示例中已有 C++ 类的定义为：

**class adder { private:**

**int int\_state; public:**

**adder();**

**int add\_one(int increment); int get\_val();**

**};**

已有源代码位于文件 adder\_cpp.h 和 adder\_cpp.cpp 中。

**% rtwdemo\_sfun\_adder\_cpp def = legacy\_code('initialize');**

**def.SFunctionName = 'rtwdemo\_sfun\_adder\_cpp'; def.StartFcnSpec = 'createAdder()';**

**def.OutputFcnSpec = 'int32 y1 = adderOutput(int32 u1)'; def.TerminateFcnSpec = 'deleteAdder()'; def.HeaderFiles = {'adder\_cpp.h'};**

**def.SourceFiles = {'adder\_cpp.cpp'}; def.IncPaths = {'rtwdemo\_lct\_src'}; def.SrcPaths = {'rtwdemo\_lct\_src'}; def.Options.language = 'C++'; def.Options.useTlcWithAccel = false;**

#### 生成 S-Function 进行仿真

要根据输入参数 'def' 提供的描述生成 C-MEX S-Function，请再次调用函数 legacy\_code() 并将第一个输入设置为 'sfcn\_cmex\_generate'。S-Function 将在仿真过程中调用已有函数。S-Function 的源代码位于文件 rtwdemo\_sfun\_adder\_cpp.cpp 中。

**legacy\_code('sfcn\_cmex\_generate', def);**

#### 编译生成的 S-Function 进行仿真

在生成 C-MEX S-Function 源文件之后，要使用 Simulink® 编译 S-Function 进行仿真，请再次调用函数 legacy\_code() 并将第一个输入设置为 'compile'。

**legacy\_code('compile', def);**

**### Start Compiling rtwdemo\_sfun\_adder\_cpp**

**mex('-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtwdemo\_lct\_src', '-I/tmp/Bdoc23a\_2 Building with 'g++'.**

**MEX completed successfully.**

**mex('rtwdemo\_sfun\_adder\_cpp.cpp', '-I/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/rtw/rtwdemos/rtw Building with 'g++'.**

**MEX completed successfully.**

**### Finish Compiling rtwdemo\_sfun\_adder\_cpp ### Exit**

#### 生成 TLC 模块文件以进行代码生成

编译 S-Function 并将其用于仿真之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'sfcn\_tlc\_generate'，以生成支持通过 Simulink® Coder™ 生成代码的 TLC 模块文件。如果不创建 TLC模块文件，则当您尝试为包含 S-Function 的模型生成代码时，代码生成将失败。S-Function 的 TLC 模块文件为 rtwdemo\_sfun\_adder\_cpp.tlc。

**legacy\_code('sfcn\_tlc\_generate', def);**

#### 生成 rtwmakecfg.m 文件以进行代码生成

创建 TLC 模块文件之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'rtwmakecfg\_generate'，以生成支持通过 Simulink® Coder™ 生成代码的 rtwmakecfg.m 文件。如果 S-Function 需要的源文件和头文件与 S-Function 不在同一个文件夹中，而您要将这些相关文件添加到代码生成过程中生成的联编文件中，请生成 rtwmakecfg.m 文件。

**legacy\_code('rtwmakecfg\_generate', def);**

#### 生成封装的 S-Function 模块以调用生成的 S-Function

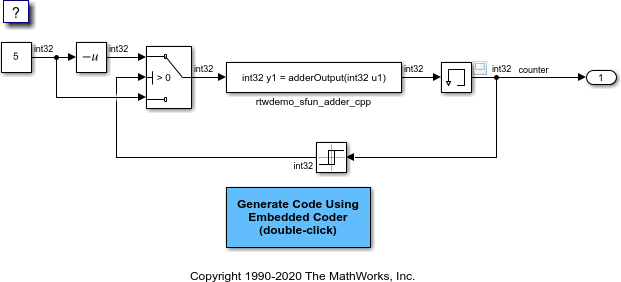
在编译 C-MEX S-Function 源之后，您可以再次调用函数 legacy\_code()。将第一个输入设置为 'slblock\_generate'，以生成封装的 S-Function 模块，此模块被配置为调用该 S-Function。软件将此模块放在新模型中。您可以将此模块复制到现有模型中。

**% legacy\_code('slblock\_generate', def);**

#### 显示生成的与现有代码的集成

模型 rtwdemo\_lct\_cpp 显示了与现有代码的集成。

**open\_system('rtwdemo\_lct\_cpp') sim('rtwdemo\_lct\_cpp')**



### 另请参阅

**legacy\_code**

### 相关示例

* “Integrate C Functions Using Legacy Code Tool”
* “从模型和生成的代码调用外部 C 代码” (Embedded Coder)

## 编写非内联 S-Function

非内联 S-Function 是 C 或 C++ MEX S-Function，Simulink 引擎和生成的代码以相同的方式处理这种函数。您只需根据 S-Function API 实现一次算法即可。Simulink 引擎和生成的代码将在模型执行过程中调用 S-Function 例程（例如 **mdlOutputs**）。

非内联 S-Function 的特点是您的 S-Function 缺少 **sfunction.tlc** 文件。文件名根据您的平台而有所不同。例如，在 64 位 Microsoft Windows 系统上，该文件名为 **sfunction.mexw64**。在 MATLAB 命令行窗口中键入 **mexext** 可查看系统使用的扩展名。

### 编写非内联 S-Function 的规范

* MEX 文件不能调用 MATLAB 函数。
* 如果 MEX 文件使用 MATLAB 外部接口库中的函数，请包括头文件 **cg\_sfun.h** 而不是 **mex.h** 或

**simulink.c**。对于头文件 **cg\_sfun.h**，在 S-Function 的末尾包括以下行：

**#ifdef MATLAB\_MEX\_FILE /\* Is this file being compiled as a MEX-file? \*/ #include "simulink.c" /\* MEX-file interface mechanism \*/**

**#else**

**#include "cg\_sfun.h" /\* Code generation registration function \*/ #endif**

* 仅使用代码生成器支持的 MATLAB API 函数。支持的 API 函数有：
  + **mxGetDoubles**
  + **mxGetEps**
  + **mxGetInf**
  + **mxGetM**
  + **mxGetN**
  + **mxGetNaN**
  + **mxGetPr**（不推荐。请改用 **mxGetDoubles**。）
  + **mxGetScalar**
  + **mxGetString**
  + **mxIsEmpty**
  + **mxIsFinite**
  + **mxIsInf**
* 生成的代码不支持 MEX Library 调用。要在 MEX 文件中而不是在生成的代码中使用此类调用，请添加以下条件：

**#ifdef MATLAB\_MEX\_FILE #endif**

* 只能使用仅包含实数数据的满矩阵。
* 不能为 **mxGetString** 的调用指定返回值。如果您指定返回值，MEX 文件将不会编译。在这种情况下，请使用函数的第二个输入参数，它返回指向字符向量的指针。
* 使用正确的 **#define s-function\_name** 语句。您指定的 S-Function 名称必须与 S-Function 的文件名完全一致。
* 尽量使用 **real\_T** 和 **int\_T** 数据类型，而不要使用 **double** 和 **int**。数据类型 **real\_T** 和 **int\_T** 更通用，可在多个环境中使用。
* 为编译过程提供用来编译 S-Function 的模块的名称。使用模板联编文件、**set\_param** 函数或 S- Function 模块参数对话框的 **S-function modules** 字段。例如，假设您使用以下命令编译 S-

Function：

**mex sfun\_main.c sfun\_module1.c sfun\_module2.c**

则可以使用以下 **set\_param** 调用来包含所需的模块：

**set\_param(sfun\_block, "SFunctionModules","sfun\_module1 sfun\_module2")**

准备好生成代码之后，强制代码生成器重新编译顶层模型。有关详细信息，请参阅“Control Regeneration of Top Model Code”。

### 非内联 S-Function 的参数类型限制

非内联 S-Function 的参数只能是以下类型：

* 双精度
* 标量、向量或二维矩阵中的字符

要想更灵活地选择提供给 S-Function 或 S-Function 中的运算的参数类型，请内联 S-Function 并考虑使用 **mdlRTW** S-Function 例程。

不支持使用 MATLAB **matrix.h** API 或其他 MATLAB API（如 **mex.h** 和 **mat.h**）中的其他函数。如果您从 S-Function 源文件中调用不支持的 API，将会发生编译器错误。有关支持的 MATLAB API 函数的详细信息，请参阅文件 **matlabroot/rtw/c/src/rt\_matrx.h** 和 **matlabroot/rtw/c/src/rt\_matrx.c**。

如果您对空矩阵使用 **mxGetPr**，该函数不会返回 **NULL**。它会返回一个随机值。因此，您必须使用 **mxIsEmpty** 保护对 **mxGetPr** 的调用。

### 另请参阅相关示例

* “Inlined and Noninlined S-Function Code”
* [“S-Function 和代码生成” （第 14-2 页）](#_bookmark75)

## 编写包装器 S-Function 和 TLC 文件

利用包装器的概念，创建能与 Simulink 和代码生成器产品无缝协作的 S-Function。您可以：

* 通过编写 MEX S-Function 包装器 (**sfunction.mex**)，在 Simulink 模型中引入您的算法。
* 通过创建 TLC S-Function 包装器 (**sfunction.tlc**)，指示代码生成器将您的算法插入到生成的代码中。

### MEX S-Function 包装器

使用 S-Function 包装器创建 S-Function 的优点在于，只需对您的原始 C/C++ 函数进行很少改动或根本不需要改动，即可将 C/C++ 代码算法插入到 Simulink 模型和生成的代码中。MEX S-Function 包装器是一个 S-Function，它调用位于另一个模块中的代码。

**注意** MEX S-Function 包装器只能在创建该包装器的 MATLAB 版本中使用。

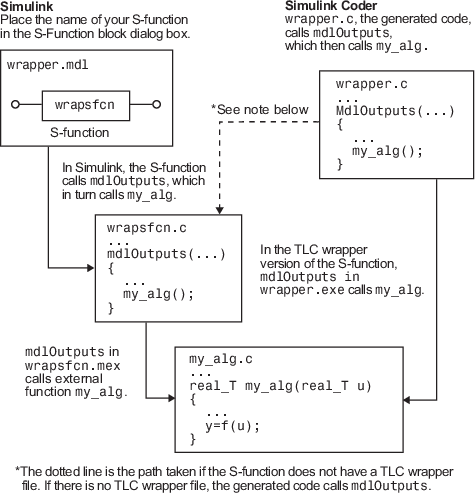
假设您有一个名为 **my\_alg** 的算法（即一个 C 函数），该算法位于 **my\_alg.c** 文件中。您可以通过创建 MEX S-Function 包装器（例如 **wrapsfcn.c**），将 **my\_alg** 集成到 Simulink 模型中。然后 Simulink 模型即可从 S-Function 模块中调用 **my\_alg**。Simulink S-Function 包含一组空函数，供 Simulink 引擎用来实现各种与 API 有关的目的。例如，虽然只有 **mdlOutputs** 调用 **my\_alg**，但引擎还会调用 **mdlTerminate**，即使此 S-Function 例程并不执行任何操作。

您可以通过创建 TLC S-Function 包装器（例如 **wrapsfcn.tlc**），在生成的代码中嵌入对 **my\_alg** 的调用。您可以消除空函数调用。您可以避免执行 **mdlOutputs** 函数的开销，然后可以消除 **my\_alg** 函数。

当您创建程序性的算法或将原有代码集成到 Simulink 模型中时，S-function 包装器很有用。如果要创建以下代码：

* 本质上是解释性代码（即按操作模式高度参数化）
* 高度优化的代码（即不需要通过额外的测试来决定代码的运行模式）

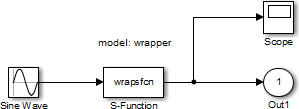
则必须为您的 S-Function 创建一个完全内联的 TLC 文件。下图显示了 S-Function 包装器的概念。



使用 S-Function 包装器将算法导入 Simulink 模型是指：将 S-Function 用作一个接口来从 **mdlOutputs**

中调用您的 C/C++ 算法。您可以将大型、独立 C/C++ 程序快速集成到模型中，而无需更改代码。

此示例模型包含一个 S-Function 包装器。



**wrapsfcn** 模块有两个关联文件：S-Function 包装器和包含算法的 C/C++ 代码。前三个语句：

1. 定义 S-Function 的名称（您在 Simulink S-Function 模块对话框中输入的内容）。
2. 指定 S-Function 使用 2 级格式。
3. 提供对 **SimStruct** 数据结构体的访问。**SimStruct** 结构体包含仿真和代码生成期间所用数据的指针，并定义在 **SimStruct** 中存储和检索数据的宏。

**#define S\_FUNCTION\_NAME wrapsfcn #define S\_FUNCTION\_LEVEL 2 #include "simstruc.h"**

**extern real\_T my\_alg(real\_T u); /\* Declare my\_alg as extern \*/**

**/\***

**\* mdlInitializeSizes - initialize the sizes array**

**\*/**

**static void mdlInitializeSizes(SimStruct \*S)**

**{**

**ssSetNumSFcnParams( S, 0); /\*number of input arguments\*/**

**if (!ssSetNumInputPorts(S, 1)) return; ssSetInputPortWidth(S, 0, 1);**

**ssSetInputPortDirectFeedThrough(S, 0, 1);**

**if (!ssSetNumOutputPorts(S,1)) return; ssSetOutputPortWidth(S, 0, 1);**

**ssSetNumSampleTimes( S, 1);**

**}**

**/\***

* **mdlInitializeSampleTimes - indicate that this S-function runs**
* **at the rate of the source (driving block)**

**\*/**

**static void mdlInitializeSampleTimes(SimStruct \*S)**

**{**

**ssSetSampleTime(S, 0, INHERITED\_SAMPLE\_TIME);**

**ssSetOffsetTime(S, 0, 0.0);**

**}**

**/\***

* **mdlOutputs - compute the outputs by calling my\_alg, which**
* **resides in another module, my\_alg.c**

**\*/**

**static void mdlOutputs(SimStruct \*S, int\_T tid)**

**{**

**InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0); real\_T \*y = ssGetOutputPortRealSignal(S,0);**

**\*y = my\_alg(\*uPtrs[0]); /\* Call my\_alg in mdlOutputs \*/**

**}**

**/\***

* **mdlTerminate - called when the simulation is terminated.**

**\*/**

**static void mdlTerminate(SimStruct \*S)**

**{**

**}**

**#ifdef MATLAB\_MEX\_FILE /\* Is this file being compiled as a MEX-file? \*/ #include "simulink.c" /\* MEX-file interface mechanism \*/**

**#else**

**#include "cg\_sfun.h" /\* Code generation registration function \*/ #endif**

有关详细信息，请参阅“Templates for C S-Functions”。

S-Function 例程 **mdlOutputs** 包含对 **my\_alg** 的函数调用，后者是包含 S-Function 执行的算法的 C 函数。对于 **my\_alg.c**，代码为：

**#ifdef MATLAB\_MEX\_FILE #include "tmwtypes.h" #else**

**#include "rtwtypes.h" #endif**

**real\_T my\_alg(real\_T u)**

**{**

**return(u \* 2.0);**

**}**

有关详细信息，请参阅“Manage Build Process File Dependencies”。

包装器 S-Function **wrapsfcn** 调用 **my\_alg** 来计算 **u \* 2.0**。要编译 **wrapsfcn.mex**，请使用以下命令：

**mex wrapsfcn.c my\_alg.c**

### TLC S-Function 包装器

TLC S-Function 包装器是一个 TLC 文件，它指定代码生成器如何调用您的代码。例如，您可以在生成的代码的 **mdlOutputs** 部分内联对 **my\_alg** 的调用。在 [“MEX S-Function 包装器” （第 14-20](#_bookmark93) 页）示例 中，对 **my\_alg** 的调用嵌入在 **mdlOutputs** 部分，如下所示：

**\*y = my\_alg(\*uPtrs[0]);**

创建 TLC S-Function 包装器时，目的也是在生成的代码中嵌入相同类型的调用。

观察代码生成器如何执行非内联 S-Function。非内联 S-Function 的特点是缺少 **sfunction.tlc** 文件，但存在 **sfunction.mex** 文件。为非内联 S-Function 生成代码时，代码生成器通过函数指针生成对 **mdlOutputs** 的调用，在此示例中，该函数指针将调用 **my\_alg**。

包装器示例包含一个 S-Function **wrapsfcn.mex**。您必须再编译一个模块 **my\_alg** 并将其与生成的代码链接。在 MATLAB 命令提示符下，输入：

**set\_param('wrapper/S-Function','SFunctionModules','my\_alg')**

### 非内联 S-Function 的代码开销

使用 **grt.tlc** 作为系统目标文件而没有 **wrapsfcn.tlc** 时生成的代码如下：

**<Generated code comments for wrapper model with noninlined wrapsfcn S-function>**

**#include <math.h> #include <string.h> #include "wrapper.h" #include "wrapper.prm"**

**/\* Start the model \*/ void mdlStart(void)**

**{**

**/\* (start code not required) \*/**

**}**

**/\* Compute block outputs \*/ void mdlOutputs(int\_T tid)**

**{**

**/\* Sin Block: <Root>/Sin \*/ rtB.Sin = rtP.Sin.Amplitude \***

**sin(rtP.Sin.Frequency \* ssGetT(rtS) + rtP.Sin.Phase);**

**/\* Level2 S-Function Block: <Root>/S-Function (wrapsfcn) \*/**

**{**

**/\* Noninlined S-functions create a SimStruct object and**

**\* generate a call to S-function routine mdlOutputs**

**\*/**

**SimStruct \*rts = ssGetSFunction(rtS, 0); sfcnOutputs(rts, tid);**

**}**

**/\* Outport Block: <Root>/Out \*/ rtY.Out = rtB.S\_Function;**

**}**

**/\* Perform model update \*/ void mdlUpdate(int\_T tid)**

**{**

**/\* (update code not required) \*/**

**}**

**/\* Terminate function \*/ void mdlTerminate(void)**

**{**

**/\* Level2 S-Function Block: <Root>/S-Function (wrapsfcn) \*/**

**{**

**/\* Noninlined S-functions require a SimStruct object and**

**\* the call to S-function routine mdlTerminate**

**\*/**

**SimStruct \*rts = ssGetSFunction(rtS, 0); sfcnTerminate(rts);**

**}**

**}**

**#include "wrapper.reg"**

**/\* [EOF] wrapper.c \*/**

生成的文件 **wrapper.reg** 还包含对包装器 S-Function 模块的 **SimStruct** 进行初始化。模型中的每个 S-Function 模块都有一个子级 **SimStruct**。通过为 S-Function 创建 TLC 包装器，可以显著减少这类开销。

### 内联包装器 S-Function

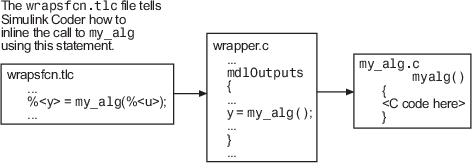
生成的代码使用以下代码在 **mdlOutputs** 中调用您的 S-Function **wrapsfcn.c**：

**SimStruct \*rts = ssGetSFunction(rtS, 0); sfcnOutputs(rts, tid);**

此调用会产生相应的计算开销。Simulink 引擎为 S-Function 模块创建一个 **SimStruct** 数据结构体。代码生成器通过函数指针构造一个调用来执行 **mdlOutputs**，然后 **mdlOutputs** 调用 **my\_alg**。通过内联对 C/C++ 算法 **my\_alg** 的调用，可以消除 **SimStruct** 和额外的函数调用，从而提高效率并减小生成的代码的大小。

内联包装器 S-Function 需要为 S-Function 创建一个 **sfunction.tlc** 文件。TLC 文件必须包含对 **my\_alg**

的函数调用。下图显示了算法、S-Function 包装器和 **sfunction.tlc** 文件之间的关系。



要内联对 **my\_alg** 的调用，请将函数调用放在与 S-Function 同名的 **sfunction.tlc** 文件中（本示例中为

**wrapsfcn.tlc**）。目标语言编译器会覆盖在生成的代码中放置对 S-Function 的调用的默认方法。

以下代码是用于内联 **wrapsfcn.c** 的 TLC 文件 **wrapsfcn.tlc**：

**%% File : wrapsfcn.tlc**

**%% Abstract:**

**%% Example inlined tlc file for S-function wrapsfcn.c**

**%%**

**%implements "wrapsfcn" "C"**

**%% Function: BlockTypeSetup ====================================================**

**%% Abstract:**

**%% Create function prototype in model.h as:**

**%% "extern real\_T my\_alg(real\_T u);"**

**%%**

**%function BlockTypeSetup(block, system) void**

**%openfile buffer**

**extern real\_T my\_alg(real\_T u); /\* This line is placed in wrapper.h \*/**

**%closefile buffer**

**%<LibCacheFunctionPrototype(buffer)>**

**%endfunction %% BlockTypeSetup**

**%% Function: Outputs ===========================================================**

**%% Abstract:**

**%% y = my\_alg( u );**

**%%**

**%function Outputs(block, system) Output**

**/\* %<Type> Block: %<Name> \*/**

**%assign u = LibBlockInputSignal(0, "", "", 0)**

**%assign y = LibBlockOutputSignal(0, "", "", 0)**

**%% PROVIDE THE CALLING STATEMENT FOR "algorithm"**

**%% The following line is expanded and placed in mdlOutputs within wrapper.c**

**%<y> = my\_alg(%<u>);**

**%endfunction %% Outputs**

这段代码的第一部分内联 **wrapsfcn** S-Function 模块并生成 C 代码：

**%implements "wrapsfcn" "C"**

下一个任务是通知代码生成器，必须在生成的 **wrapper.h** 文件中为模型中的 **wrapsfcn** S-Function 模块将例程 **my\_alg** 声明为外部例程。使用 **BlockTypeSetup** 函数可以一次为所有 **wrapsfcn** S- Function 模块执行此声明。在此函数中，您指示目标语言编译器创建一个缓冲区，并将 **my\_alg** 以 **extern** 的形式缓存在生成的头文件 **wrapper.h** 中。

最后一步是内联对 **my\_alg** 函数的调用。**Outputs** 函数内联调用。在此函数中，您将访问模块的输入和输出，并放置一个对 **my\_alg** 的直接调用。调用嵌入在 **wrapper.c** 中。

### 内联代码

内联 S-Function 包装器时生成的代码与默认生成的代码类似。**mdlTerminate** 函数不包含对空函数的调用，**mdlOutputs** 函数现在直接调用 **my\_alg**。

**void mdlOutputs(int\_T tid)**

**{**

**/\* Sin Block: <Root>/Sin \*/ rtB.Sin = rtP.Sin.Amplitude \***

**sin(rtP.Sin.Frequency \* ssGetT(rtS) + rtP.Sin.Phase);**

**/\* S-Function Block: <Root>/S-Function \*/**

**rtB.S\_Function = my\_alg(rtB.Sin); /\* Inlined call to my\_alg \*/**

**/\* Outport Block: <Root>/Out \*/ rtY.Out = rtB.S\_Function;**

**}**

**wrapper.reg** 不会为 S-Function 创建子级 **SimStruct**，因为生成的代码会直接调用 **my\_alg**，从而消除了 1 KB 以上的内存使用量。

### 另请参阅

**相关示例**

* [“编写非内联 S-Function” （第 14-18 页）](#_bookmark89)
* [“编写完全内联的 S-Function” （第 14-27 页）](#_bookmark98)

## 编写完全内联的 S-Function

完全内联的 S-Function 将您的算法（模块）嵌入到生成的代码中，您不能将其与内置模块区分开来。通常，完全内联的 S-Function 需要实现两次算法：一次针对 Simulink 模型 (C/C++ MEX S-Function)，一次针对代码生成（TLC 文件）。

使用“编写包装器 [S-Function 和 TLC 文件” （第 14-20](#_bookmark92) 页）中的示例，您可以通过在 **wrapsfcn.tlc** 中指定显式代码（即 **2.0 \* u**）完全消除对 **my\_alg** 的调用。虽然这样可以提高性能，但如果您要处理大量 C/C++ 代码，这项任务可能很耗时。您还必须在两个地方维护算法：C/C++ S-Function 本身和对应的 TLC 文件。需要考虑所得到的性能增益能否大于它的弊端。要内联本示例中使用的算法，则在 **wrapsfcn.tlc** 文件的 **Outputs** 部分，请不要编写：

**%<y> = my\_alg(%<u>);**

使用...

**%<y> = 2.0 \* %<u>;**

以下代码是 **mdlOutputs** 中生成的代码：

**void mdlOutputs(int\_T tid)**

**{**

**/\* Sin Block: <Root>/Sin \*/ rtB.Sin = rtP.Sin.Amplitude \***

**sin(rtP.Sin.Frequency \* ssGetT(rtS) + rtP.Sin.Phase);**

**/\* S-Function Block: <Root>/S-Function \*/**

**rtB.S\_Function = 2.0 \* rtB.Sin; /\* Explicit embedding of algorithm \*/**

**/\* Outport Block: <Root>/Out \*/ rtY.Out = rtB.S\_Function;**

**}**

Target Language Compiler 使用算法本身代替对 **my\_alg** 的调用。

### 多端口 S-Function

更高级的多端口内联 S-Function 示例是 **sfun\_multiport.c** 和 **sfun\_multiport.tlc**。此 S-Function 说明如何为包含多个端口的 S-Function 创建完全内联的 TLC 文件。

### 编写内联 S-Function 的规范

* 考虑使用模块属性 **RTWdata**（请参阅“S-Function RTWdata”）。此属性是可与模块关联的字符向量结构体。代码生成器将此结构体与模型一起保存在 **model.rtw** 文件中，并使 **.rtw** 文件更易于阅

读。例如，在 MATLAB 命令行窗口中，假设您输入以下命令：

**mydata.field1 = 'information for field1'; mydata.field2 = 'information for field2'; set\_param(sfun\_block, 'RTWdata', mydata);**

代码生成器为模块生成的 **.rtw** 文件包含结构体 **mydata** 中指定的注释。

* 考虑使用 **mdlRTW** 函数在生成的代码中内联 C MEX S-Function 以用于以下目的：
  + 重命名生成的代码中的可调参数。
  + 在 TLC 文件中引入不可调参数。

### 另请参阅

**相关示例**

* “Write Fully Inlined S-Functions with mdlRTW Routine”

# 基于消息的通信

## 生成用于在 Simulink 组件之间进行数据通信的 C 消息

要生成支持 Simulink 环境中的模型组件之间基于消息的通信的 C 代码，请使用 Simulink Messages & Events Library 模块 Send 和 Receive。要自定义通信，请使用 Queue 模块（来自同一个库）来设置容 量、排序策略（LIFO、FIFO 和优先级）和覆盖策略（队列超出容量时的行为）的参数。您可以使用 Simulink Coder 为基于 GRT 的系统目标文件生成 C 代码，或使用 Embedded Coder 为基于 ERT 的系统目标文件生成 C 代码。

对于可以在 Simulink 中建模的分布式复杂系统，消息是一种有效的通信方法。

### 如何在 Simulink 中为基于消息的通信准备模型

在 Simulink 中，您可以对模型组件之间基于消息的通信进行建模：

1. 创建一个包含 Send 模块的模型（引用模型）。
2. 创建一个包含 Receive 模块的模型（引用模型）。
3. 创建一个包含两个 Model 模块的模型（顶层模型）。
   * 将第一个 Model 模块设置为包含 Send 模块的模型（步骤 1 中的模型）。
   * 将第二个 Model 模块设置为包含 Receive 模块的模型（步骤 2 中的模型）。
4. 运行模型时，顶层模型中消息线上会生成一个队列。队列可显式或隐式控制消息通信。您可以使用生成的队列，也可以在顶层模型中添加 Queue 模块来显式指定通信参数。

### 模型组件如何传递消息，以及代码如何实现该行为

引用模型通过以下方式传递消息：

1. 在包含 Send 模块的模型中，Send 模块将信号转换为消息。
2. 包含队列的顶层模型根据定义容量、传送顺序和其他服务质量 (QoS) 指标的参数来管理消息。
3. 在包含 Receive 模块的模型中，Receive 模块将消息转换回信号。

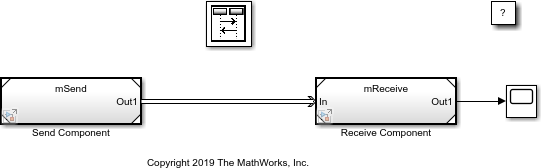
在生成的 C 代码中，顶层模型通过建立发送和接收引用模型可彼此独立访问的集合接口来支持这些引用模型之间的连接。

生成的 C 代码通过以下方式实现消息行为：

1. 在每个模型边界上创建一个服务。服务包含对顶层模型的引用和一个入口函数（称为服务函数），供引用模型将消息传递到顶层模型。
2. 顶层模型初始化每个服务，以创建与每个引用模型的连接。
3. 引用模型调用服务函数来将消息传递给顶层模型。

### C 代码生成示例

此示例基于“Establish Message Send and Receive Interfaces Between Software Components”中提供的模型生成并检查 C 代码。

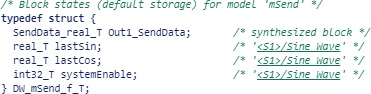


生成 C 代码：

1. 打开模型。
2. 在 App 库中，点击 **Embedded Coder**。
3. 对于每个模型（顶层模型和两个引用模型），在 Configuration Parameters 对话框的 Code Generation 窗格中，将 **Language** 设置为 **C** 并保存模型。
4. 生成代码。在 **C Code** 选项卡上，点击 **Build**。
5. 查看生成的代码。在 **C Code** 选项卡上，点击 **View Code**。

检查 C 代码：

1. 在每个模型边界上创建一个服务。在 C 语言中，引用模型将服务表示为 DWork。服务提供了指向顶层模型实例的指针，以及一个引用模型调用的服务函数，用于传递消息。
   1. 要查看发送消息的服务，请打开发送引用模型头文件 **mSend.h**。查看 DWork 分配。



* 1. 要查看发送消息数据类型，如果数据类型是可共享的（例如，内置数据类型、导入的总线类型或采用指定数据类型的导出总线类型），则信息位于共享头文件中。如果数据类型不可共享，则信息位于模型头文件 **mSend.h** 中。在此示例中，请通过打开共享头文件 **SendData\_real\_T.h** 查看可共享数据类型。



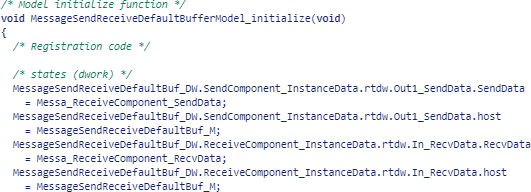
* 1. 要查看用于接收消息的服务，请打开接收引用模型头文件 **mReceive.h**。查看 DWork 分配。



* 1. 要查看收到的消息数据类型，如果数据类型是可共享的（例如，内置数据类型、导入的总线类型或采用指定数据类型的导出总线类型），则信息位于共享头文件中。如果数据类型不可共享，则信息位于模型头文件 **mReceive.h** 中。在此示例中，请通过打开共享头文件 **RecvData\_real\_T.h** 查看可共享数据类型。



1. 顶层模型初始化每个服务，以创建与每个引用模型的连接。在 C 语言中，顶层模型初始化每个引用模型 DWork。要查看初始化，请打开顶层模型 C 文件 **MessageSendReceiveDefaultBufferModel.c**。



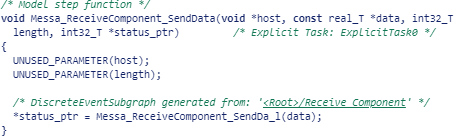
1. 引用模型调用服务函数来将消息传递给顶层模型。在 C 语言中，引用模型通过解引用服务函数指针并传递一个指向顶层模型实例的指针来调用服务函数。您可以在顶层模型头文件中查看服务函数的原 型。您可以在引用模型 C 文件中查看这些服务函数的引用模型调用。
   1. 要查看发送消息的服务函数的原型，请打开顶层模型头文件

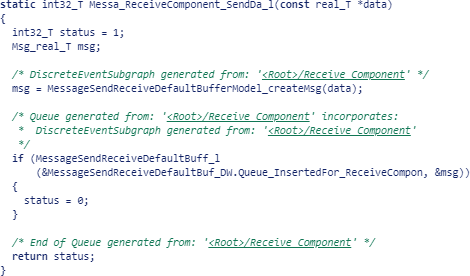
#### MessageSendReceiveDefaultBufferModel.h。



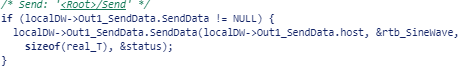
* 1. 要查看发送消息的服务函数的实现，请打开顶层模型 C 文件

#### MessageSendReceiveDefaultBufferModel.c。





* 1. 要查看发送消息的引用模型如何调用服务函数，请打开模型的 C 文件 **mSend.c**。在单步函数中，模型调用服务以向顶层模型发送实例数据和消息，并接收返回状态。



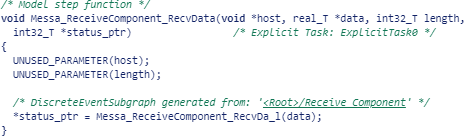
* 1. 要查看接收消息的服务函数的原型，请打开顶层模型头文件

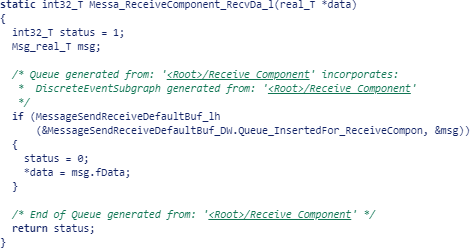
#### MessageSendReceiveDefaultBufferModel.h。



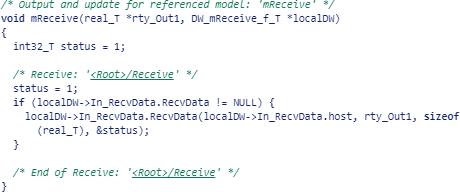
* 1. 要查看接收消息的服务函数的实现，请打开顶层模型 C 文件

#### MessageSendReceiveDefaultBufferModel.c。





* 1. 要查看接收消息的引用模型如何调用服务函数，请打开模型的 C 文件 **mReceive.c**。在单步函数中，模型调用服务来接收消息有效负载和返回状态。



### 注意事项和限制

* Simulink Coder 支持为基于 GRT 的系统目标文件生成 C 代码。
* Embedded Coder 支持为基于 ERT 的系统目标文件生成 C 代码。
* 要生成代码，对于顶层模型和引用模型，请选择相同的语言 (C) 和相同的系统目标文件。
* 您不能为具有根消息端口的顶层模型配置函数原型控制 (FPC)。
* 不支持外部模型和变体模型。
* 不支持软件在环 (SIL) 和处理器在环 (PIL) 仿真。

# 数据、函数和文件定义

# Simulink Coder 中的数据表示

* [“生成的代码如何存储内部信号、状态和参数数据” （第 16-2 页）](#_bookmark109)
* [“在生成的代码中使用枚举数据” （第 16-21 页）](#_bookmark116)
* [“在生成的代码中创建可调标定参数” （第 16-28 页）](#_bookmark123)
* [“在仿真和代码执行期间在参数值集之间切换” （第 16-37 页）](#_bookmark133)

## 生成的代码如何存储内部信号、状态和参数数据

为了根据输入计算输出，生成的代码会在全局内存中存储一些内部数据。未连接到根级别输入或输出

（Inport 或 Outport 模块）的信号是内部数据。

内部数据还可以包括：

* 模块状态，例如 Unit Delay 模块的状态。算法必须保留执行周期之间的状态值，因此生成的代码通常将状态存储在全局内存中（例如，作为全局变量或全局结构体变量的一个字段）。
* 模块参数，例如 Gain 模块的**增益**参数，代码生成器不能在代码中内联该参数的值。例如，代码生成器不能内联非标量参数的值。
* 条件执行子系统（例如使能子系统）的状态指示符。

要获得更高效的代码，您可以配置优化，例如 **配置参数** > **默认参数行为**和**配置参数** > **信号存储重用**，这些优化会尝试消除内部数据的存储。但是，优化不能消除某些数据的存储，这会占用生成的代码的内存使用量。

当您了解生成的代码存储内部数据的默认格式时，您可以：

* 默认情况下，使信号可访问且参数可调。然后，您可以在执行期间与代码进行交互并监视代码。
* 消除内部数据存储，并根据您的硬件和编译工具链来控制优化不能消除的数据在内存中的位置，从而生成高效的生产代码。
* 将内部数据段提升到模型接口，以便其他组件和系统可以访问这些数据。

有关生成的代码如何通过接口与调用方环境交换数据的信息，请参阅“How Generated Code Exchanges Data with an Environment”。

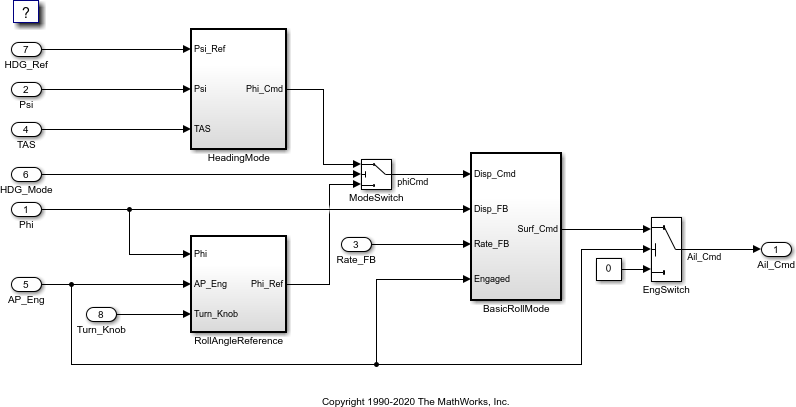
### 生成的代码中的内部数据

此示例说明生成的代码如何存储模块状态等内部数据。

#### 浏览模型示例

打开示例模型 **rtwdemo\_roll**。

**open\_system('rtwdemo\_roll')**



该模型包含不连接到根级别 Inport 或 Outport 模块的内部信号。某些信号具有名称，例如 **phiCmd** 信号。

该模型还包含一些维护状态数据的模块。例如，在 **BasicRollMode** 子系统中，标记为 **Integrator** 的 Discrete-Time Integrator 模块用于维护状态。

在模型中，将**配置参数 > 代码生成 > 系统目标文件**设置为 **grt.tlc**。

**set\_param('rtwdemo\_roll','SystemTargetFile','grt.tlc')**

检查**配置参数 > 代码生成 > 接口 > 代码接口打包**的设置。设置 **Nonreusable function** 表示生成的代码不可重用（可重入）。

对于此示例，通过清除**配置参数 > 代码生成 > 接口 > 高级参数 > Mat 文件记录**生成更简单的代码。

**set\_param('rtwdemo\_roll','MatFileLogging','off')**

#### 生成不可重用的代码

设置以下配置参数：

* 将**默认参数行为**设置为 **Tunable**。
* 清除**信号存储重用**。

**set\_param('rtwdemo\_roll','DefaultParameterBehavior','Tunable',... 'OptimizeBlockIOStorage','off')**

从模型中生成代码。

**slbuild('rtwdemo\_roll')**

**### Starting build procedure for: rtwdemo\_roll**

**### Successful completion of build procedure for: rtwdemo\_roll Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**=============================================================================================**

**rtwdemo\_roll Code generated and compiled Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 21.256s**

文件 **rtwdemo\_roll.h** 定义了几种表示内部数据的结构体类型。例如，模块输入和输出结构体为模型中的每个内部信号定义一个字段。每个字段名称都派生自生成该信号的模块的名称，或者派生自该信号的名称

（如果您指定了信号名称）。

**file = fullfile('rtwdemo\_roll\_grt\_rtw','rtwdemo\_roll.h'); rtwdemodbtype(file,...**

**'/\* Block signals (default storage) \*/','} B\_rtwdemo\_roll\_T;',1,1)**

**/\* Block signals (default storage) \*/ typedef struct {**

**real32\_T phiCmd; /\* '<Root>/ModeSwitch' \*/**

**real32\_T hdgError; /\* '<S2>/Sum' \*/**

**real32\_T DispGain; /\* '<S2>/DispGain' \*/**

**real32\_T Product; /\* '<S2>/Product' \*/ real32\_T Abs; /\* '<S3>/Abs' \*/**

**real32\_T FixPtUnitDelay1; /\* '<S4>/FixPt Unit Delay1' \*/ real32\_T Xnew; /\* '<S4>/Enable' \*/**

**real32\_T TKSwitch; /\* '<S3>/TKSwitch' \*/ real32\_T RefSwitch; /\* '<S3>/RefSwitch' \*/ real32\_T Integrator; /\* '<S1>/Integrator' \*/ real32\_T DispLimit; /\* '<S1>/DispLimit' \*/ real32\_T Sum; /\* '<S1>/Sum' \*/ real32\_T DispGain\_f; /\* '<S1>/DispGain' \*/ real32\_T RateLimit; /\* '<S1>/RateLimit' \*/ real32\_T Sum1; /\* '<S1>/Sum1' \*/**

**real32\_T RateGain; /\* '<S1>/RateGain' \*/**

**real32\_T Sum2; /\* '<S1>/Sum2' \*/ real32\_T CmdLimit; /\* '<S1>/CmdLimit' \*/**

**real32\_T IntGain; /\* '<S1>/IntGain' \*/ boolean\_T NotEngaged; /\* '<S3>/NotEngaged' \*/ boolean\_T TKThreshold; /\* '<S3>/TKThreshold' \*/**

**boolean\_T RefThreshold2; /\* '<S3>/RefThreshold2' \*/ boolean\_T RefThreshold1; /\* '<S3>/RefThreshold1' \*/ boolean\_T Or; /\* '<S3>/Or' \*/**

**boolean\_T NotEngaged\_e; /\* '<S1>/NotEngaged' \*/**

**} B\_rtwdemo\_roll\_T;**

该文件定义一种 DWork 结构体类型，用于表示模块状态，例如 Discrete-Time Integrator 模块的状态。

**rtwdemodbtype(file,...**

**'/\* Block states (default storage) for system','} DW\_rtwdemo\_roll\_T;',1,1)**

**/\* Block states (default storage) for system '<Root>' \*/**

**typedef struct {**

**real32\_T FixPtUnitDelay1\_DSTATE; /\* '<S4>/FixPt Unit Delay1' \*/ real32\_T Integrator\_DSTATE; /\* '<S1>/Integrator' \*/**

**int8\_T Integrator\_PrevResetState; /\* '<S1>/Integrator' \*/**

**} DW\_rtwdemo\_roll\_T;**

该文件定义一种表示参数数据的结构体类型。模型中的每个可调模块参数（例如 Gain 模块的**增益**参数）显示为此结构体的一个字段。如果模块参数从 MATLAB® 变量或 **Simulink.Parameter** 对象获取其参数值，则该变量或对象显示为字段，而不是模块参数。

该文件还定义一种结构体类型，即实时模型数据结构体，其单个字段表示一种运行时指示，用于指示生成的代码在执行期间是否遇到错误。

**rtwdemodbtype(file,'/\* Real-time Model Data Structure \*/',... '/\* Block parameters (default storage) \*/',1,0)**

**/\* Real-time Model Data Structure \*/ struct tag\_RTM\_rtwdemo\_roll\_T { const char\_T \*errorStatus;**

**};**

对于表示实时模型数据结构体的结构体类型，文件 **rtwdemo\_roll\_types.h** 会创建一个别名，生成的代码稍后将使用该别名为结构体分配内存。

**file = fullfile('rtwdemo\_roll\_grt\_rtw','rtwdemo\_roll\_types.h'); rtwdemodbtype(file,'/\* Forward declaration for rtModel \*/',...**

**'RT\_MODEL\_rtwdemo\_roll\_T;',1,1)**

**/\* Forward declaration for rtModel \*/**

**typedef struct tag\_RTM\_rtwdemo\_roll\_T RT\_MODEL\_rtwdemo\_roll\_T;**

**rtwdemo\_roll.c** 使用这些结构体类型来定义用于为生成的算法存储内部数据的全局结构体变量（为其分配内存）。该文件还定义表示实时模型数据结构体的变量和指向该结构体的指针。

**file = fullfile('rtwdemo\_roll\_grt\_rtw','rtwdemo\_roll.c'); rtwdemodbtype(file,'/\* Block signals (default storage) \*/',...**

**'= &rtwdemo\_roll\_M\_;',1,1)**

**/\* Block signals (default storage) \*/ B\_rtwdemo\_roll\_T rtwdemo\_roll\_B;**

**/\* Block states (default storage) \*/ DW\_rtwdemo\_roll\_T rtwdemo\_roll\_DW;**

**/\* External inputs (root inport signals with default storage) \*/ ExtU\_rtwdemo\_roll\_T rtwdemo\_roll\_U;**

**/\* External outputs (root outports fed by signals with default storage) \*/ ExtY\_rtwdemo\_roll\_T rtwdemo\_roll\_Y;**

**/\* Real-time model \*/**

**static RT\_MODEL\_rtwdemo\_roll\_T rtwdemo\_roll\_M\_; RT\_MODEL\_rtwdemo\_roll\_T \*const rtwdemo\_roll\_M = &rtwdemo\_roll\_M\_;**

模型 **step** 函数（表示主模型算法）使用 **void void** 接口（不带参数）。

**rtwdemodbtype(file,...**

**'/\* Model step function \*/','void rtwdemo\_roll\_step(void)',1,1)**

**/\* Model step function \*/**

**void rtwdemo\_roll\_step(void)**

在函数定义中，算法通过直接访问全局变量来执行计算并将中间结果存储在信号和状态结构体中。该算法还从对应的全局变量中读取参数数据。例如，在 **BasicRollMode** 子系统中，为 **Integrator** 模块生成的代码在结构体中读取和写入信号、状态和参数数据。

**rtwdemodbtype(file,'/\* DiscreteIntegrator: ''<S1>/Integrator'' \*',... '/\* End of DiscreteIntegrator: ''<S1>/Integrator'' \*/',1,1)**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/**

**if (rtwdemo\_roll\_B.NotEngaged\_e || (rtwdemo\_roll\_DW.Integrator\_PrevResetState**

**!= 0)) {**

**rtwdemo\_roll\_DW.Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_IC;**

**}**

**if (rtwdemo\_roll\_DW.Integrator\_DSTATE >= rtwdemo\_roll\_P.intLim) { rtwdemo\_roll\_DW.Integrator\_DSTATE = rtwdemo\_roll\_P.intLim;**

**} else if (rtwdemo\_roll\_DW.Integrator\_DSTATE <= rtwdemo\_roll\_P.Integrator\_LowerSat) {**

**rtwdemo\_roll\_DW.Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_LowerSat;**

**}**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/**

**rtwdemo\_roll\_B.Integrator = rtwdemo\_roll\_DW.Integrator\_DSTATE;**

**/\* Saturate: '<S1>/DispLimit' \*/ u0 = rtwdemo\_roll\_B.phiCmd;**

**u1 = rtwdemo\_roll\_P.DispLimit\_LowerSat; u2 = rtwdemo\_roll\_P.dispLim;**

**if (u0 > u2) {**

**/\* Saturate: '<S1>/DispLimit' \*/ rtwdemo\_roll\_B.DispLimit = u2;**

**} else if (u0 < u1) {**

**/\* Saturate: '<S1>/DispLimit' \*/ rtwdemo\_roll\_B.DispLimit = u1;**

**} else {**

**/\* Saturate: '<S1>/DispLimit' \*/ rtwdemo\_roll\_B.DispLimit = u0;**

**}**

**/\* End of Saturate: '<S1>/DispLimit' \*/**

**/\* Sum: '<S1>/Sum' incorporates:**

**\* Inport: '<Root>/Phi'**

**\*/**

**rtwdemo\_roll\_B.Sum = rtwdemo\_roll\_B.DispLimit - rtwdemo\_roll\_U.Phi;**

**/\* Gain: '<S1>/DispGain' \*/**

**rtwdemo\_roll\_B.DispGain\_f = rtwdemo\_roll\_P.dispGain \* rtwdemo\_roll\_B.Sum;**

**/\* Saturate: '<S1>/RateLimit' \*/ u0 = rtwdemo\_roll\_B.DispGain\_f;**

**u1 = rtwdemo\_roll\_P.RateLimit\_LowerSat; u2 = rtwdemo\_roll\_P.rateLim;**

**if (u0 > u2) {**

**/\* Saturate: '<S1>/RateLimit' \*/ rtwdemo\_roll\_B.RateLimit = u2;**

**} else if (u0 < u1) {**

**/\* Saturate: '<S1>/RateLimit' \*/ rtwdemo\_roll\_B.RateLimit = u1;**

**} else {**

**/\* Saturate: '<S1>/RateLimit' \*/ rtwdemo\_roll\_B.RateLimit = u0;**

**}**

**/\* End of Saturate: '<S1>/RateLimit' \*/**

**/\* Sum: '<S1>/Sum1' incorporates:**

**\* Inport: '<Root>/Rate\_FB'**

**\*/**

**rtwdemo\_roll\_B.Sum1 = rtwdemo\_roll\_B.RateLimit - rtwdemo\_roll\_U.Rate\_FB;**

**/\* Gain: '<S1>/RateGain' \*/**

**rtwdemo\_roll\_B.RateGain = rtwdemo\_roll\_P.rateGain \* rtwdemo\_roll\_B.Sum1;**

**/\* Sum: '<S1>/Sum2' \*/**

**rtwdemo\_roll\_B.Sum2 = rtwdemo\_roll\_B.Integrator + rtwdemo\_roll\_B.RateGain;**

**/\* Saturate: '<S1>/CmdLimit' \*/ u0 = rtwdemo\_roll\_B.Sum2;**

**u1 = rtwdemo\_roll\_P.CmdLimit\_LowerSat; u2 = rtwdemo\_roll\_P.cmdLim;**

**if (u0 > u2) {**

**/\* Saturate: '<S1>/CmdLimit' \*/ rtwdemo\_roll\_B.CmdLimit = u2;**

**} else if (u0 < u1) {**

**/\* Saturate: '<S1>/CmdLimit' \*/ rtwdemo\_roll\_B.CmdLimit = u1;**

**} else {**

**/\* Saturate: '<S1>/CmdLimit' \*/ rtwdemo\_roll\_B.CmdLimit = u0;**

**}**

**/\* End of Saturate: '<S1>/CmdLimit' \*/**

**/\* Gain: '<S1>/IntGain' \*/**

**rtwdemo\_roll\_B.IntGain = rtwdemo\_roll\_P.intGain \* rtwdemo\_roll\_B.Sum1;**

**/\* Update for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_roll\_DW.Integrator\_DSTATE += rtwdemo\_roll\_P.Integrator\_gainval \* rtwdemo\_roll\_B.IntGain;**

**if (rtwdemo\_roll\_DW.Integrator\_DSTATE >= rtwdemo\_roll\_P.intLim) { rtwdemo\_roll\_DW.Integrator\_DSTATE = rtwdemo\_roll\_P.intLim;**

**} else if (rtwdemo\_roll\_DW.Integrator\_DSTATE <= rtwdemo\_roll\_P.Integrator\_LowerSat) {**

**rtwdemo\_roll\_DW.Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_LowerSat;**

**}**

**rtwdemo\_roll\_DW.Integrator\_PrevResetState = (int8\_T) rtwdemo\_roll\_B.NotEngaged\_e;**

**/\* End of Update for DiscreteIntegrator: '<S1>/Integrator' \*/**

**/\* End of Outputs for SubSystem: '<Root>/BasicRollMode' \*/**

**/\* Switch: '<Root>/EngSwitch' incorporates:**

**\* Inport: '<Root>/AP\_Eng'**

**\*/**

**if (rtwdemo\_roll\_U.AP\_Eng) {**

**/\* Outport: '<Root>/Ail\_Cmd' \*/ rtwdemo\_roll\_Y.Ail\_Cmd = rtwdemo\_roll\_B.CmdLimit;**

**} else {**

**/\* Outport: '<Root>/Ail\_Cmd' incorporates:**

**\* Constant: '<Root>/Zero'**

**\*/**

**rtwdemo\_roll\_Y.Ail\_Cmd = rtwdemo\_roll\_P.Zero\_Value\_c;**

**}**

**/\* End of Switch: '<Root>/EngSwitch' \*/**

**}**

**/\* Model initialize function \*/**

**void rtwdemo\_roll\_initialize(void)**

**{**

**/\* Registration code \*/**

**/\* initialize error status \*/ rtmSetErrorStatus(rtwdemo\_roll\_M, (NULL));**

**/\* block I/O \*/**

**(void) memset(((void \*) &rtwdemo\_roll\_B), 0, sizeof(B\_rtwdemo\_roll\_T));**

**/\* states (dwork) \*/**

**(void) memset((void \*)&rtwdemo\_roll\_DW, 0, sizeof(DW\_rtwdemo\_roll\_T));**

**/\* external inputs \*/**

**(void)memset(&rtwdemo\_roll\_U, 0, sizeof(ExtU\_rtwdemo\_roll\_T));**

**/\* external outputs \*/ rtwdemo\_roll\_Y.Ail\_Cmd = 0.0F;**

**/\* SystemInitialize for Atomic SubSystem: '<Root>/RollAngleReference' \*/**

**/\* InitializeConditions for UnitDelay: '<S4>/FixPt Unit Delay1' \*/ rtwdemo\_roll\_DW.FixPtUnitDelay1\_DSTATE = rtwdemo\_roll\_P.LatchPhi\_vinit;**

**/\* End of SystemInitialize for SubSystem: '<Root>/RollAngleReference' \*/**

**/\* SystemInitialize for Atomic SubSystem: '<Root>/BasicRollMode' \*/**

**/\* InitializeConditions for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_roll\_DW.Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_IC; rtwdemo\_roll\_DW.Integrator\_PrevResetState = 0;**

**/\* End of SystemInitialize for SubSystem: '<Root>/BasicRollMode' \*/**

**}**

**/\* Model terminate function \*/**

**void rtwdemo\_roll\_terminate(void)**

**{**

**/\* (no terminate code required) \*/**

**}**

由于存在 **void void** 接口和直接数据访问，该函数不可重入。如果在一个应用程序中多次调用该函数，则每次调用都会将数据写入全局结构体变量，后续调用可以读取该数据，从而导致各次调用之间出现意外的干扰。

模型初始化函数 **rtwdemo\_roll\_initialize** 将所有内部数据初始化为零。该函数还通过调用专用宏函数对错误状态进行初始化。初始化函数直接访问全局变量，这意味着该函数不可重入。

**rtwdemodbtype(file,'/\* Model initialize function \*/',... 'sizeof(DW\_rtwdemo\_roll\_T));',1,1)**

**/\* Model initialize function \*/**

**void rtwdemo\_roll\_initialize(void)**

**{**

**/\* Registration code \*/**

**/\* initialize error status \*/ rtmSetErrorStatus(rtwdemo\_roll\_M, (NULL));**

**/\* block I/O \*/**

**(void) memset(((void \*) &rtwdemo\_roll\_B), 0, sizeof(B\_rtwdemo\_roll\_T));**

**/\* states (dwork) \*/**

**(void) memset((void \*)&rtwdemo\_roll\_DW, 0, sizeof(DW\_rtwdemo\_roll\_T));**

然后，该函数将 DWork 结构体中的模块状态初始化为模型中的模块参数指定的初始值。模型的三个状态中的两个具有可调初始值，因此代码通过从参数结构体中读取数据来初始化它们。

**rtwdemodbtype(file,...**

**'/\* SystemInitialize for Atomic SubSystem: ''<Root>/RollAngleReference'' \*/',... '/\* Model terminate function \*/',1,0)**

**/\* SystemInitialize for Atomic SubSystem: '<Root>/RollAngleReference' \*/**

**/\* InitializeConditions for UnitDelay: '<S4>/FixPt Unit Delay1' \*/ rtwdemo\_roll\_DW.FixPtUnitDelay1\_DSTATE = rtwdemo\_roll\_P.LatchPhi\_vinit;**

**/\* End of SystemInitialize for SubSystem: '<Root>/RollAngleReference' \*/**

**/\* SystemInitialize for Atomic SubSystem: '<Root>/BasicRollMode' \*/**

**/\* InitializeConditions for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_roll\_DW.Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_IC; rtwdemo\_roll\_DW.Integrator\_PrevResetState = 0;**

**/\* End of SystemInitialize for SubSystem: '<Root>/BasicRollMode' \*/**

**}**

#### 生成可重用的代码

您可以将生成的代码配置为可重入代码，这意味着您可以在一个应用程序中多次调用入口函数。使用此配置时，入口函数并不直接访问全局变量，而是通过形参（指针参数）接受内部数据。通过使用这些指针参数，每次调用都可以在一组单独的全局变量中维护内部数据，从而防止各次调用之间出现意外的交互。

在模型中，将**配置参数 > 代码生成 > 接口 > 代码接口打包**设置为 **Reusable function**。

**set\_param('rtwdemo\_roll','CodeInterfacePackaging','Reusable function')**

从模型中生成代码。

**slbuild('rtwdemo\_roll')**

**### Starting build procedure for: rtwdemo\_roll**

**### Successful completion of build procedure for: rtwdemo\_roll Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**============================================================================**

**rtwdemo\_roll Code generated and compiled Generated code was out of date.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 16.304s**

现在，在 **rtwdemo\_roll.h** 中，实时模型数据结构体包含指向错误指示的指针、内部数据以及 **ExtU** 和

**ExtY** 子结构体形式的主输入和输出数据（其字段表示模型根级别的 Inport 和 Outport 模块）。

**file = fullfile('rtwdemo\_roll\_grt\_rtw','rtwdemo\_roll.h'); rtwdemodbtype(file,'/\* Real-time Model Data Structure \*/',...**

**'/\* External data declarations for dependent source files \*/',1,0)**

**/\* Real-time Model Data Structure \*/ struct tag\_RTM\_rtwdemo\_roll\_T { const char\_T \*errorStatus; B\_rtwdemo\_roll\_T \*blockIO; ExtU\_rtwdemo\_roll\_T \*inputs; ExtY\_rtwdemo\_roll\_T \*outputs; DW\_rtwdemo\_roll\_T \*dwork;**

**};**

**/\* Block parameters (default storage) \*/ extern P\_rtwdemo\_roll\_T rtwdemo\_roll\_P;**

要在一个应用程序中多次调用生成的代码，您的代码必须在每次调用时为实时模型数据结构体分配内存。文件 **rtwdemo\_roll.c** 定义一个专用函数，它为新实时模型数据结构体分配内存并返回指向该结构体的指针。该函数还为模型数据结构体中的字段指向的子结构体（例如 DWork 结构体）分配内存。

**file = fullfile('rtwdemo\_roll\_grt\_rtw','rtwdemo\_roll.c'); rtwdemodbtype(file,'/\* Model data allocation function \*/',...**

**'RT\_MODEL\_rtwdemo\_roll\_T \*rtwdemo\_roll(void)',1,1)**

**/\* Model data allocation function \*/ RT\_MODEL\_rtwdemo\_roll\_T \*rtwdemo\_roll(void)**

模型 **step** 函数接受表示实时模型数据结构体的参数。

**rtwdemodbtype(file,'/\* Model step function \*/','void rtwdemo\_roll\_step',1,1)**

**/\* Model step function \*/**

**void rtwdemo\_roll\_step(RT\_MODEL\_rtwdemo\_roll\_T \*const rtwdemo\_roll\_M)**

在函数定义中，算法首先将每个指针从实时模型数据结构体中提取到局部变量中。

**rtwdemodbtype(file,'\*rtwdemo\_roll\_B =','rtwdemo\_roll\_M->outputs;',1,1)**

**B\_rtwdemo\_roll\_T \*rtwdemo\_roll\_B = rtwdemo\_roll\_M->blockIO; DW\_rtwdemo\_roll\_T \*rtwdemo\_roll\_DW = rtwdemo\_roll\_M->dwork; ExtU\_rtwdemo\_roll\_T \*rtwdemo\_roll\_U = (ExtU\_rtwdemo\_roll\_T \*) rtwdemo\_roll\_M->inputs;**

**ExtY\_rtwdemo\_roll\_T \*rtwdemo\_roll\_Y = (ExtY\_rtwdemo\_roll\_T \*) rtwdemo\_roll\_M->outputs;**

然后，为了访问存储在全局内存中的内部数据，该算法与这些局部变量交互。

**rtwdemodbtype(file,'/\* DiscreteIntegrator: ''<S1>/Integrator'' \*/',... '/\* End of DiscreteIntegrator: ''<S1>/Integrator'' \*/',1,1)**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/ if (rtwdemo\_roll\_B->NotEngaged\_e ||**

**(rtwdemo\_roll\_DW->Integrator\_PrevResetState != 0)) { rtwdemo\_roll\_DW->Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_IC;**

**}**

**if (rtwdemo\_roll\_DW->Integrator\_DSTATE >= rtwdemo\_roll\_P.intLim) { rtwdemo\_roll\_DW->Integrator\_DSTATE = rtwdemo\_roll\_P.intLim;**

**} else if (rtwdemo\_roll\_DW->Integrator\_DSTATE <= rtwdemo\_roll\_P.Integrator\_LowerSat) {**

**rtwdemo\_roll\_DW->Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_LowerSat;**

**}**

**/\* DiscreteIntegrator: '<S1>/Integrator' \*/**

**rtwdemo\_roll\_B->Integrator = rtwdemo\_roll\_DW->Integrator\_DSTATE;**

**/\* Saturate: '<S1>/DispLimit' \*/ u0 = rtwdemo\_roll\_B->phiCmd;**

**u1 = rtwdemo\_roll\_P.DispLimit\_LowerSat; u2 = rtwdemo\_roll\_P.dispLim;**

**if (u0 > u2) {**

**/\* Saturate: '<S1>/DispLimit' \*/ rtwdemo\_roll\_B->DispLimit = u2;**

**} else if (u0 < u1) {**

**/\* Saturate: '<S1>/DispLimit' \*/ rtwdemo\_roll\_B->DispLimit = u1;**

**} else {**

**/\* Saturate: '<S1>/DispLimit' \*/ rtwdemo\_roll\_B->DispLimit = u0;**

**}**

**/\* End of Saturate: '<S1>/DispLimit' \*/**

**/\* Sum: '<S1>/Sum' \*/**

**rtwdemo\_roll\_B->Sum = rtwdemo\_roll\_B->DispLimit - rtwdemo\_roll\_U->Phi;**

**/\* Gain: '<S1>/DispGain' \*/**

**rtwdemo\_roll\_B->DispGain\_f = rtwdemo\_roll\_P.dispGain \* rtwdemo\_roll\_B->Sum;**

**/\* Saturate: '<S1>/RateLimit' \*/**

**u0 = rtwdemo\_roll\_B->DispGain\_f;**

**u1 = rtwdemo\_roll\_P.RateLimit\_LowerSat; u2 = rtwdemo\_roll\_P.rateLim;**

**if (u0 > u2) {**

**/\* Saturate: '<S1>/RateLimit' \*/ rtwdemo\_roll\_B->RateLimit = u2;**

**} else if (u0 < u1) {**

**/\* Saturate: '<S1>/RateLimit' \*/ rtwdemo\_roll\_B->RateLimit = u1;**

**} else {**

**/\* Saturate: '<S1>/RateLimit' \*/ rtwdemo\_roll\_B->RateLimit = u0;**

**}**

**/\* End of Saturate: '<S1>/RateLimit' \*/**

**/\* Sum: '<S1>/Sum1' \*/**

**rtwdemo\_roll\_B->Sum1 = rtwdemo\_roll\_B->RateLimit - rtwdemo\_roll\_U->Rate\_FB;**

**/\* Gain: '<S1>/RateGain' \*/**

**rtwdemo\_roll\_B->RateGain = rtwdemo\_roll\_P.rateGain \* rtwdemo\_roll\_B->Sum1;**

**/\* Sum: '<S1>/Sum2' \*/**

**rtwdemo\_roll\_B->Sum2 = rtwdemo\_roll\_B->Integrator + rtwdemo\_roll\_B->RateGain;**

**/\* Saturate: '<S1>/CmdLimit' \*/ u0 = rtwdemo\_roll\_B->Sum2;**

**u1 = rtwdemo\_roll\_P.CmdLimit\_LowerSat; u2 = rtwdemo\_roll\_P.cmdLim;**

**if (u0 > u2) {**

**/\* Saturate: '<S1>/CmdLimit' \*/ rtwdemo\_roll\_B->CmdLimit = u2;**

**} else if (u0 < u1) {**

**/\* Saturate: '<S1>/CmdLimit' \*/ rtwdemo\_roll\_B->CmdLimit = u1;**

**} else {**

**/\* Saturate: '<S1>/CmdLimit' \*/ rtwdemo\_roll\_B->CmdLimit = u0;**

**}**

**/\* End of Saturate: '<S1>/CmdLimit' \*/**

**/\* Gain: '<S1>/IntGain' \*/**

**rtwdemo\_roll\_B->IntGain = rtwdemo\_roll\_P.intGain \* rtwdemo\_roll\_B->Sum1;**

**/\* Update for DiscreteIntegrator: '<S1>/Integrator' \*/**

**rtwdemo\_roll\_DW->Integrator\_DSTATE += rtwdemo\_roll\_P.Integrator\_gainval \* rtwdemo\_roll\_B->IntGain;**

**if (rtwdemo\_roll\_DW->Integrator\_DSTATE >= rtwdemo\_roll\_P.intLim) { rtwdemo\_roll\_DW->Integrator\_DSTATE = rtwdemo\_roll\_P.intLim;**

**} else if (rtwdemo\_roll\_DW->Integrator\_DSTATE <= rtwdemo\_roll\_P.Integrator\_LowerSat) {**

**rtwdemo\_roll\_DW->Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_LowerSat;**

**}**

**rtwdemo\_roll\_DW->Integrator\_PrevResetState = (int8\_T) rtwdemo\_roll\_B->NotEngaged\_e;**

**/\* End of Update for DiscreteIntegrator: '<S1>/Integrator' \*/**

**/\* End of Outputs for SubSystem: '<Root>/BasicRollMode' \*/**

**/\* Switch: '<Root>/EngSwitch' \*/ if (rtwdemo\_roll\_U->AP\_Eng) {**

**/\* Outport: '<Root>/Ail\_Cmd' \*/**

**rtwdemo\_roll\_Y->Ail\_Cmd = rtwdemo\_roll\_B->CmdLimit;**

**} else {**

**/\* Outport: '<Root>/Ail\_Cmd' incorporates:**

**\* Constant: '<Root>/Zero'**

**\*/**

**rtwdemo\_roll\_Y->Ail\_Cmd = rtwdemo\_roll\_P.Zero\_Value\_c;**

**}**

**/\* End of Switch: '<Root>/EngSwitch' \*/**

**}**

**/\* Model initialize function \*/**

**void rtwdemo\_roll\_initialize(RT\_MODEL\_rtwdemo\_roll\_T \*const rtwdemo\_roll\_M)**

**{**

**DW\_rtwdemo\_roll\_T \*rtwdemo\_roll\_DW = rtwdemo\_roll\_M->dwork;**

**/\* SystemInitialize for Atomic SubSystem: '<Root>/RollAngleReference' \*/**

**/\* InitializeConditions for UnitDelay: '<S4>/FixPt Unit Delay1' \*/ rtwdemo\_roll\_DW->FixPtUnitDelay1\_DSTATE = rtwdemo\_roll\_P.LatchPhi\_vinit;**

**/\* End of SystemInitialize for SubSystem: '<Root>/RollAngleReference' \*/**

**/\* SystemInitialize for Atomic SubSystem: '<Root>/BasicRollMode' \*/**

**/\* InitializeConditions for DiscreteIntegrator: '<S1>/Integrator' \*/ rtwdemo\_roll\_DW->Integrator\_DSTATE = rtwdemo\_roll\_P.Integrator\_IC; rtwdemo\_roll\_DW->Integrator\_PrevResetState = 0;**

**/\* End of SystemInitialize for SubSystem: '<Root>/BasicRollMode' \*/**

**}**

**/\* Model terminate function \*/**

**void rtwdemo\_roll\_terminate(RT\_MODEL\_rtwdemo\_roll\_T \* rtwdemo\_roll\_M)**

**{**

**/\* model code \*/ rt\_FREE(rtwdemo\_roll\_M->blockIO); rt\_FREE(rtwdemo\_roll\_M->inputs); rt\_FREE(rtwdemo\_roll\_M->outputs); rt\_FREE(rtwdemo\_roll\_M->dwork); rt\_FREE(rtwdemo\_roll\_M);**

**}**

**/\* Model data allocation function \*/ RT\_MODEL\_rtwdemo\_roll\_T \*rtwdemo\_roll(void)**

**{**

**RT\_MODEL\_rtwdemo\_roll\_T \*rtwdemo\_roll\_M;**

**rtwdemo\_roll\_M = (RT\_MODEL\_rtwdemo\_roll\_T \*) malloc(sizeof (RT\_MODEL\_rtwdemo\_roll\_T));**

**if (rtwdemo\_roll\_M == (NULL)) { return (NULL);**

**}**

**(void) memset((char \*)rtwdemo\_roll\_M, 0, sizeof(RT\_MODEL\_rtwdemo\_roll\_T));**

**/\* block I/O \*/**

**{**

**B\_rtwdemo\_roll\_T \*b = (B\_rtwdemo\_roll\_T \*) malloc(sizeof(B\_rtwdemo\_roll\_T)); rt\_VALIDATE\_MEMORY(rtwdemo\_roll\_M,b);**

**rtwdemo\_roll\_M->blockIO = (b);**

**}**

**/\* states (dwork) \*/**

**{**

**DW\_rtwdemo\_roll\_T \*dwork = (DW\_rtwdemo\_roll\_T \*) malloc(sizeof (DW\_rtwdemo\_roll\_T)); rt\_VALIDATE\_MEMORY(rtwdemo\_roll\_M,dwork);**

**rtwdemo\_roll\_M->dwork = (dwork);**

**}**

**/\* external inputs \*/**

**{**

**ExtU\_rtwdemo\_roll\_T \*rtwdemo\_roll\_U = (ExtU\_rtwdemo\_roll\_T \*) malloc(sizeof (ExtU\_rtwdemo\_roll\_T)); rt\_VALIDATE\_MEMORY(rtwdemo\_roll\_M,rtwdemo\_roll\_U);**

**rtwdemo\_roll\_M->inputs = (((ExtU\_rtwdemo\_roll\_T \*) rtwdemo\_roll\_U));**

**}**

**/\* external outputs \*/**

**{**

**ExtY\_rtwdemo\_roll\_T \*rtwdemo\_roll\_Y = (ExtY\_rtwdemo\_roll\_T \*) malloc(sizeof (ExtY\_rtwdemo\_roll\_T)); rt\_VALIDATE\_MEMORY(rtwdemo\_roll\_M,rtwdemo\_roll\_Y);**

**rtwdemo\_roll\_M->outputs = (rtwdemo\_roll\_Y);**

**}**

**{**

**B\_rtwdemo\_roll\_T \*rtwdemo\_roll\_B = rtwdemo\_roll\_M->blockIO; DW\_rtwdemo\_roll\_T \*rtwdemo\_roll\_DW = rtwdemo\_roll\_M->dwork; ExtU\_rtwdemo\_roll\_T \*rtwdemo\_roll\_U = (ExtU\_rtwdemo\_roll\_T \*) rtwdemo\_roll\_M->inputs;**

**ExtY\_rtwdemo\_roll\_T \*rtwdemo\_roll\_Y = (ExtY\_rtwdemo\_roll\_T \*) rtwdemo\_roll\_M->outputs;**

**/\* block I/O \*/**

**(void) memset(((void \*) rtwdemo\_roll\_B), 0, sizeof(B\_rtwdemo\_roll\_T));**

**/\* states (dwork) \*/**

**(void) memset((void \*)rtwdemo\_roll\_DW, 0, sizeof(DW\_rtwdemo\_roll\_T));**

**/\* external inputs \*/**

**(void)memset(rtwdemo\_roll\_U, 0, sizeof(ExtU\_rtwdemo\_roll\_T));**

**/\* external outputs \*/ rtwdemo\_roll\_Y->Ail\_Cmd = 0.0F;**

**}**

**return rtwdemo\_roll\_M;**

**}**

同样，模型初始化函数接受实时模型数据结构体作为参数。

**rtwdemodbtype(file,...**

**'/\* Model initialize function \*/','void rtwdemo\_roll\_initialize',1,1)**

**/\* Model initialize function \*/**

**void rtwdemo\_roll\_initialize(RT\_MODEL\_rtwdemo\_roll\_T \*const rtwdemo\_roll\_M)**

由于您对入口函数的每次调用都与一个单独的实时模型数据结构体交互，因此可以避免各次调用之间发生意外交互。

#### 使用代码生成优化消除内部数据

为了获得消耗更少内存的更高效代码，请选择优化，例如您在前面清除了的**默认参数行为**。

**set\_param('rtwdemo\_roll','DefaultParameterBehavior','Inlined',... 'OptimizeBlockIOStorage','on',...**

**'LocalBlockOutputs','on')**

在此示例中，为了获得更简单的代码，请将**代码接口打包**设置为 **Nonreusable function**。

**set\_param('rtwdemo\_roll','CodeInterfacePackaging','Nonreusable function')**

从模型中生成代码。

**slbuild('rtwdemo\_roll')**

**### Starting build procedure for: rtwdemo\_roll**

**### Successful completion of build procedure for: rtwdemo\_roll Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**============================================================================**

**rtwdemo\_roll Code generated and compiled Generated code was out of date.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 14.954s**

现在，**rtwdemo\_roll.h** 没有定义用于模块输入和输出的结构体。对于模型中的所有内部信号，优化要么已消除了存储，要么创建了局部函数变量而不是全局结构体字段。

优化未能消除三种模块状态的存储，因此文件继续定义 DWork 结构体类型。

**file = fullfile('rtwdemo\_roll\_grt\_rtw','rtwdemo\_roll.h'); rtwdemodbtype(file,...**

**'/\* Block states (default storage) for system','} DW\_rtwdemo\_roll\_T;',1,1)**

**/\* Block states (default storage) for system '<Root>' \*/ typedef struct {**

**real32\_T FixPtUnitDelay1\_DSTATE; /\* '<S4>/FixPt Unit Delay1' \*/**

**real32\_T Integrator\_DSTATE; /\* '<S1>/Integrator' \*/ int8\_T Integrator\_PrevResetState; /\* '<S1>/Integrator' \*/**

**} DW\_rtwdemo\_roll\_T;**

现在，为 Discrete-Time Integrator 模块生成的代码仅在 DWork 结构体中存储状态和输出数据。

**file = fullfile('rtwdemo\_roll\_grt\_rtw','rtwdemo\_roll.c'); rtwdemodbtype(file,'/\* Update for DiscreteIntegrator: ''<S1>/Integrator''',...**

**'/\* End of Update for DiscreteIntegrator: ''<S1>/Integrator'' \*/',1,1)**

**/\* Update for DiscreteIntegrator: '<S1>/Integrator' incorporates:**

**\* Gain: '<S1>/IntGain'**

**\*/**

**rtwdemo\_roll\_DW.Integrator\_DSTATE += 0.5F \* rtb\_TKSwitch \* 0.025F; if (rtwdemo\_roll\_DW.Integrator\_DSTATE >= 5.0F) { rtwdemo\_roll\_DW.Integrator\_DSTATE = 5.0F;**

**} else if (rtwdemo\_roll\_DW.Integrator\_DSTATE <= -5.0F) { rtwdemo\_roll\_DW.Integrator\_DSTATE = -5.0F;**

**}**

**rtwdemo\_roll\_DW.Integrator\_PrevResetState = (int8\_T)rtb\_NotEngaged\_f;**

优化还消除了模型中模块参数的存储。例如，在 Discrete-Time Integrator 模块中，**饱和上限**和**饱和下限**参数设置为 **intLim** 和 **-intLim**。**intLim** 是 **Simulink.Parameter** 对象，用于存储值 **5**。在为 Discrete-Time Integrator 生成的代码中，这些模块参数和 **intLim** 显示为内联字面数字 **5.0F** 和 **-5.0F**。

如果模型包含代码生成器不能直接内联的参数（例如数组参数），则代码会定义表示该数据的结构体类型。此常量参数结构体使用 **const** 存储类型限定符，因此某些编译工具链可以进一步优化程序集代码。

结构体 **ConstParam** **model\_** 的声明位于 **model.h** 中。

**/\* Constant parameters (auto storage) \*/ typedef struct {**

**/\* Expression: [1 2 3 4 5 6 7]**

**\* Referenced by: '<Root>/Constant'**

**\*/**

**real\_T Constant\_Value[7];**

**/\* Expression: [7 6 5 4 3 2 1]**

**\* Referenced by: '<Root>/Gain'**

**\*/**

**real\_T Gain\_Gain[7];**

**} ConstParam\_model;**

默认情况下，常量参数 **model\_ConstP** 的声明位于 **model\_data.c** 中。

**/\* Constant parameters (auto storage) \*/ const ConstParam\_model model\_ConstP = {**

**/\* Expression: [1 2 3 4 5 6 7]**

**\* Referenced by: '<Root>/Constant'**

**\*/**

**{ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 },**

**/\* Expression: [7 6 5 4 3 2 1]**

**\* Referenced by: '<Root>/Gain'**

**\*/**

**{ 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0 }**

**};**

**model\_ConstP** 作为参数传递给引用模型。

### 生成的代码中的局部变量

当您选择**配置参数** > **启用局部模块输出**优化时，代码生成器会尝试通过将内部信号表示为局部变量（而不是全局结构体的字段）来生成更高效的代码。如果局部变量消耗的内存有超过目标硬件上的可用堆栈空间的风险，请考虑通过设置**配置参数** > **最大堆栈大小(字节)** 来指示最大堆栈大小。有关详细信息，请参阅 Maximum stack size (bytes)。

### 生成的代码中测试点的外观

测试点是存储在唯一内存位置的信号。有关在模型中包含测试点的信息，请参阅“将信号配置为测试点”。

为包含测试点的模型生成代码时，编译过程会为每个测试点分配单独的内存缓冲区。默认情况下，测试点存储为标准数据结构体的成员，例如 **model\_B**。

如果您有 Embedded Coder：

* 通过在代码映射编辑器中为**内部数据**类别的数据指定代码生成设置，您可以控制测试点的默认表示形式

（请参阅“Configure Default Code Generation for Data” (Embedded Coder)）。

* 您可以使用 Ignore test point signals (Embedded Coder) 参数指定编译过程忽略模型中的测试点，从而允许最佳缓冲区分配。忽略测试点可加快从原型构建到部署的过程，并避免由于工作流的产物而导致生成的代码出现意外的性能降级。请参阅 Ignore test point signals (Embedded Coder)。

虚拟总线不显示在生成的代码中，即使与测试点相关联时也是如此。要在生成的代码中显示总线，请使用非虚拟总线或使用通过 Signal Conversion 模块转换为非虚拟总线的虚拟总线。

### 生成的代码中工作区变量的外观

您可以使用工作区变量来指定模型中的模块参数值。工作区变量包括您存储在工作区（例如基础工作区）或数据字典中的 MATLAB 数值变量和 **Simulink.Parameter** 对象。

将**默认参数行为**设置为“可调”时，默认情况下，工作区变量在生成的代码中显示为全局参数结构体的可调字段。如果使用一个这样的变量指定多个模块参数值，则该变量显示为全局参数结构体的单个字段。代码不会创建多个字段来表示模块参数。因此，在代码执行期间调整字段值会更改模型的数学行为，就像在仿真期间调整 MATLAB 变量或参数对象的值一样。

如果您有 Embedded Coder，则可以通过在代码映射编辑器中指定参数数据类别的代码生成设置来控制工作区变量的默认表示形式（请参阅“Configure Default Code Generation for Data” (Embedded Coder)）。

* **模型参数**类别适用于存储在模型工作区中的变量。
* **外部参数**类别适用于存储在基础工作区或数据字典中的变量。

### 将内部数据提升到接口

默认情况下，代码生成器假定应用程序中的其他系统和组件不需要访问内部数据。例如，代码生成器会对内部数据进行优化，以从生成的代码中消除它们。对于原型构建和测试目的，您可以通过清除优化或配置测试点并应用存储类来访问内部数据（请参阅“Preserve Variables in Generated Code”）。对于优化的生产代码，将单个数据项配置为作为模型接口的一部分显示在生成的代码中。

#### 您可以提升的数据

根据生成的代码的重入性，即您为**代码接口打包**选择的设置，您可以通过将模型中的每个数据项配置为在代码中显示为下列实体之一来参与接口：

* 全局符号，例如全局变量或对专用函数的调用
* 生成的入口函数的形参（参数）

下表显示了每个数据类别参与接口时可使用的机制。

|  |  |  |
| --- | --- | --- |
| **数据类别** | **显示为全局符号** | **显示为入口函数的参数** |
| 根级别 Inport 或 Outport 模块 | 仅适用于非重入模型。 | 是。 |
| 连接两个模块的信号 | 仅适用于非重入模型。 | 仅适用于可重入模型，且仅作为结构体字段。  或者，将信号连接到根级别 Outport 模块。 |
| 模块状态 | 仅适用于非重入模型。 | 仅适用于可重入模型，且仅作为结构体字段。 |
| 数据存储，例如 Data Store Memory 模块 | 是。 | 仅适用于可重入模型，且仅作为结构体字段。 |
| 模块参数或参数对象，例如  **Simulink.Parameter** | 是。 | 仅作为结构体的字段。 |

#### 单实例算法

对于单实例算法（将**代码接口打包**设置为“不可重用函数”），使用模型数据编辑器或属性检查器将存储类直接应用于单个数据项。使用直接应用的存储类时，数据项在代码中显示为全局符号，例如全局变量。存储类还可以防止优化消除数据项的存储。

您可以将存储类应用于信号、模块状态和模块参数。（对于模块参数，可以通过 **Simulink.Parameter**等参数对象间接应用存储类）。但是，对于信号，请考虑将信号连接到模型根级别的 Outport 模块。然后，您可以选择将存储类应用于模块。在模块图中，Outport 模块显示信号表示系统输出。

有关存储类的详细信息，请参阅“模型接口元素的 [C 代码生成配置” （第 17-2](#_bookmark135) 页）。

#### 可重入算法

对于可重入算法（将**代码接口打包**设置为“可重用函数”），可使用不同方法将数据项配置为在代码中体现为生成的入口函数的形参。

* 对于内部信号，可直接应用存储类 **Model default**（请参阅“模型接口元素的 [C 代码生成配置” （第](#_bookmark135) [17-2](#_bookmark135) 页））。如果您有 Embedded Coder，在代码映射编辑器中，对于**内部数据**类别，请将默认

存储类设置为“默认”或者设置为您在 Embedded Coder 字典中定义的结构化存储类（请参阅 “Create Code Definitions for Use in the Code Mappings Editor” (Embedded Coder)）。或

者，将信号配置为测试点（请参阅“生成的代码中测试点的外观” [（第 16-17](#_bookmark112) 页））。默认情况下，[信号显示为标准数据结构体之一的字段（请参阅“生成的代码如何存储内部信号、状态和参数数据”](#_bookmark109)

[（第 16-2](#_bookmark109) 页））。如果您不希望信号显示在生产代码中，请使用测试点，以便稍后选择模型配置参数

#### 忽略测试点信号。

或者，将信号连接到模型根级别的 Outport 模块。将信号连接到根级别 Outport 模块可防止优化从代码中消除信号。为了帮助在大型模型中进行信号路由，请使用 Goto 和 From 模块。

* 对于模块参数，请创建一个参数对象，例如 **Simulink.Parameter**，并直接将 **Auto** 之外的存储类应用于该对象。存储类可防止优化在代码中内联参数值。

使用存储类时，会在模型的实例之间共享该参数，这些实例是对入口函数的调用。这些函数以全局符号而不是实参形式直接访问参数数据。要使模型的每个实例能够使用不同的参数值，请将该参数配置为模型参量。有关详细信息，请参阅“Specify Instance-Specific Parameter Values for Reusable Referenced Model”。

有关应用存储类的信息，请参阅“模型接口元素的 [C 代码生成配置” （第 17-2](#_bookmark135) 页）。

### 控制内部数据的默认表示形式 (Embedded Coder)

默认情况下，代码生成器会将优化无法消除的内部数据（如大多数状态数据）聚合到 DWork 结构体等标准结构体中。使用 Embedded Coder，您可以控制生成的代码存储这些数据的方式。

#### 通过插入 pragma 指令控制数据在内存中的位置

使用代码映射编辑器为每个类别的数据（例如属于**内部数据**的状态和信号数据）指定默认内存段。在生成的代码中，您的自定义 pragma 指令或其他装饰元素位于数据定义和声明的前后位置。

您还可以根据模型中的原子子系统对结构体进行分区，以便为子例程和其他算法子组件的数据指定不同的默认内存段。

有关详细信息，请参阅“Control Data and Function Placement in Memory by Inserting Pragmas” (Embedded Coder)。

#### 控制标准数据结构体的类型、字段和全局变量的名称

您可以控制标准数据结构体的某些特性。有关详细信息，请参阅“Manage Replacement of Simulink Data Types in Generated Code” (Embedded Coder)。

要对结构体特性施加更多控制，例如在生成的代码文件中的位置，请使用 Embedded Coder 字典创建自己的结构化存储类。然后，使用代码映射编辑器将存储类应用于数据类别。存储类会删除标准结构体中的数据，从而创建您可以施加更多控制的其他结构体。有关将默认存储类应用于数据类别的详细信息，请参阅“Configure Default Code Generation for Data” (Embedded Coder)。有关创建存储类的详细信 息，请参阅“Define Service Interfaces, Storage Classes, Memory Sections, and Function Templates for Software Architecture” (Embedded Coder)。

#### 根据子组件将数据组织到结构体中

* 在标准数据结构体中，要创建包含单实例（非重入）子例程或子组件的数据的子结构体，请使用原子子系统来封装对应的模块。在子系统参数中，将**函数打包**设置为“可重用函数”。有关详细信息，请参阅 “为非虚拟子系统生成模块化函数代码” (Embedded Coder)。

或者，将模块封装在模型中并使用 Model 模块。在引用模型中，将**配置参数** > **模型引用** > **每个顶层模型允许的实例总数**设置为“多个”。有关详细信息，请参阅“Generate Code for Model Reference Hierarchy”。

* 要在模型中创建包含多实例（可重入）子例程或子组件的数据的单独独立结构体，请使用原子子系统来封装对应的模块。在子系统参数中，将**函数打包**设置为“不可重用函数”，然后选择**具有独立数据的函数**。有关详细信息，请参阅“为非虚拟子系统生成模块化函数代码” (Embedded Coder)。

或者，将模块封装在模型中并使用 Model 模块。在引用模型中，选择以下方法之一：

* + 将**配置参数** > **模型引用** > **每个顶层模型允许的实例总数**设置为“一个”。有关详细信息，请参阅 “Generate Code for Model Reference Hierarchy”。
  + 将**每个顶层模型允许的实例总数**设置为“多个”，并使用 Embedded Coder 字典创建一个结构化存储类。然后，使用代码映射编辑器将存储类应用于数据类别。要创建存储类，请参阅“Define Service Interfaces, Storage Classes, Memory Sections, and Function Templates for Software Architecture” (Embedded Coder)。要将存储类应用于数据类别，请参阅“Configure Default Code Generation for Data” (Embedded Coder)。

#### 将信号和参数数据组织成有意义的自定义结构体和子结构体

要将任意信号和参数组织成自定义结构体和子结构体，请创建非虚拟总线信号和参数结构体。（可选）要防止优化消除代码中的数据，请将总线信号或参数结构体的存储类设置为“自动”（默认设置）以外的 值。

在向模型添加模块时，必须将每个新信号和参数显式放入总线或结构体中。

有关详细信息，请参阅“Organize Data into Structures in Generated Code”。

#### 创建单独的全局变量而不是结构体字段

要使内部数据的类别在生成的代码中显示为单独的非结构化全局变量而不是标准数据结构体的字段，请使用代码映射编辑器将非结构化存储类应用于数据类别。例如，应用存储类 **ExportedGlobal**。但是，如果通过将配置参数**代码接口打包**设置为“不可重用函数”以外的值来生成多实例可重入代码，则不能将此方法用于某些数据类别（请参阅“Storage Classes and Reentrant, Multi-Instance Models and Components”）。

要使用代码映射编辑器将默认存储类应用于数据类别，请参阅“Configure Default Code Generation for Data” (Embedded Coder)。要选择存储类，请参阅“Choose Storage Class for Controlling Data Representation in Generated Code”。

### 另请参阅相关示例

* “Data Structures in the Generated Code”
* “How Generated Code Exchanges Data with an Environment”
* “模型接口元素的 C 代码生成配置” (Embedded Coder)

## 在生成的代码中使用枚举数据

[“枚举数据类型” （第 16-21 页）](#_bookmark117)

[“为枚举指定整数数据类型” （第 16-21 页）](#_bookmark118) [“自定义枚举数据类型” （第 16-22 页）](#_bookmark119) [“在生成的代码中控制枚举类型实现” （第 16-25 页）](#_bookmark120) [“枚举的类型转换” （第 16-26 页）](#_bookmark121) [“有关枚举类型的限制” （第 16-27 页）](#_bookmark122)

**本节内容**

### 枚举数据类型

枚举数据是指仅限于一组有限值的数据。枚举数据类型是一个 MATLAB 类，它定义一组枚举值。每个枚举值包括一个枚举名称和一个基础整数，该数字由软件在内部以及生成的代码中使用。下面是本节示例中使用的枚举数据类型 **BasicColors** 的 MATLAB 类定义。

**classdef BasicColors < Simulink.IntEnumType enumeration**

**Red(0) Yellow(1) Blue(2) end**

**end**

有关枚举数据类型及其在 Simulink 模型中的用法的基本信息，请参阅“在 Simulink 模型中使用枚举数据”。有关 Stateflow® 图中的枚举数据类型的信息，请参阅“Define Enumerated Data Types” (Stateflow)。

### 为枚举指定整数数据类型

为枚举指定数据类型时，您可以：

* 通过指定超类来控制生成的代码中枚举数据类型的大小。
* 减少 RAM/ROM 使用量。
* 提高代码的可移植性。
* 改善与现有代码的集成。

您可以指定以下整数数据类型：

* **int8**
* **uint8**
* **int16**
* **uint16**
* **int32**
* **Simulink.IntEnumType**。指定适合您的硬件平台的有符号整数范围内的值。

#### 在 MATLAB 文件中使用类定义

要指定整数数据类型大小，请从整数数据类型派生枚举类。

**classdef Colors < int8 enumeration Red(0)**

**Green(1) Blue(2) end**

**end**

代码生成器生成以下代码：

**typedef int8\_T Colors;**

**#define Red ((Colors)0) #define Green ((Colors)1) #define Blue ((Colors)2)**

#### 使用函数 Simulink.defineIntEnumType

要指定整数数据类型大小，请指定名称-值对组 **StorageType** 作为整数数据类型。

**Simulink.defineIntEnumType('Colors',{'Red','Green','Blue'},... [0;1;2],'StorageType','int8')**

代码生成器生成以下代码：

**typedef int8\_T Colors;**

**#define Red ((Colors)0) #define Green ((Colors)1) #define Blue ((Colors)2)**

### 自定义枚举数据类型

当您从使用枚举数据的模型中生成代码时，可以实现下面这些静态方法，在仿真期间和生成的代码中自定义该枚举数据类型的行为：

* **getDefaultValue** - 指定枚举数据类型的默认值。
* **getDescription** - 指定枚举数据类型的说明。
* **getHeaderFile** - 指定为生成的代码定义类型的头文件。
* **getDataScope** - 指定生成的代码是否将枚举数据类型定义导出或导入单独的头文件。
* **addClassNameToEnumNames** - 指定类名在生成的代码中是否变成前缀。

上述第一个方法 **getDefaultValue** 与仿真和代码生成均有关，“指定默认枚举值” 中对此方法进行了说明。其他方法仅与代码生成有关。要自定义枚举类型的行为，请在枚举类定义的 **methods(Static)** 部分包含该方法的某个版本。如果您不想自定义类型，则可以省略 **methods(Static)** 部分。下表总结了每一种方法和要提供的数据。

|  |  |  |  |
| --- | --- | --- | --- |
| **静态方法** | **用途** | **未实现方法时的默认值** | **自定义返回值** |
| **getDefaultValue** | 指定类的默认枚举成员。 | 枚举定义中指定的第一个成员 | 包含类中某个枚举成员的名称的字符向量（请参阅“对枚举进行实例化”）。 |
| **getDescription** | 指定枚举类的说明。 | **''** | 包含类型说明的字符向量。 |
| **getHeaderFile** | 指定头文件的名称。方法 **getDataScope** 决定该文件的重要性。 | **''** | 包含定义枚举类型的头文件的名称的字符向量。  默认情况下，生成的 **#include** 指令使用预处理器分隔符 **"**，而不是 **<** 和  **>**。要生成指令 **#include**  **<myTypes.h>**，请将自定义返回值指定为 **'<myTypes.h>'**。 |
| **getDataScope** | 指定生成的代码是导出还是导入枚举数据类型的定义。使用方法 **getHeaderFile**指定定义该类型的生成的或包含的头文件。 | **'Auto'** | 以下值之一：**'Auto'**、 **'Exported'** 或 **'Imported'**。 |
| **addClassNameToEnum Names** | 指定在生成的代码中是否为类名称添加前缀。 | **false** | **true** 或 **false**。 |

#### 指定描述信息

要为枚举数据类型指定描述信息，请在枚举类的 **methods(Static)** 部分包含此方法：

**function retVal = getDescription()**

**% GETDESCRIPTION Optional description of the data type. retVal = 'description';**

**end**

使用 MATLAB 字符向量替换 **description**。定义枚举类型的生成代码会包括指定的描述信息。

#### 在生成的代码中导入类型定义

为了防止在生成的代码中定义枚举数据类型，可在外部文件中提供定义，并在枚举类的 **methods(Static)**

部分包含这些方法：

**function retVal = getHeaderFile()**

**% GETHEADERFILE Specifies the file that defines this type in generated code.**

**% The method getDataScope determines the significance of the specified file. retVal = 'imported\_enum\_type.h';**

**end**

**function retVal = getDataScope()**

**% GETDATASCOPE Specifies whether generated code imports or exports this type.**

**% Return one of:**

**% 'Auto': define type in model\_types.h, or import if header file specified**

**% 'Exported': define type in a generated header file**

**% 'Imported': import type definition from specified header file**

**% If you do not define this method, DataScope is 'Auto' by default. retVal = 'Imported';**

**end**

这样，生成的代码将不会在 **model\_types.h** 中定义类型（此为默认行为），而是使用与下面类似的

**#include** 语句从指定的头文件中导入定义：

**#include "imported\_enum\_type.h"**

生成代码时不会创建导入的头文件。您必须使用方法 **getHeaderFile** 所指定的文件名提供定义枚举数据类型的头文件。

要创建与现有 C 代码枚举对应的 Simulink 枚举，请使用 **Simulink.importExternalCTypes** 函数。

#### 在生成的代码中导出类型定义

要生成单独的定义枚举数据类型的头文件，请在枚举类的 **methods(Static)** 部分包含这些方法：

**function retVal = getDataScope()**

**% GETDATASCOPE Specifies whether generated code imports or exports this type.**

**% Return one of:**

**% 'Auto': define type in model\_types.h, or import if header file specified**

**% 'Exported': define type in a generated header file**

**% 'Imported': import type definition from specified header file**

**% If you do not define this method, DataScope is 'Auto' by default. retVal = 'Exported';**

**end**

**function retVal = getHeaderFile()**

**% GETHEADERFILE Specifies the file that defines this type in generated code.**

**% The method getDataScope determines the significance of the specified file. retVal = 'exported\_enum\_type.h';**

**end**

生成的代码将枚举类型定义导出到生成的头文件 **exported\_enum\_type.h** 中。

#### 为类名添加前缀

默认情况下，在生成的代码中，枚举值的名称与枚举类定义中的名称相同。在您的代码中，也可以使用类名为枚举类中的每个枚举值添加前缀。您可以采用这种方法来防止标识符冲突或提高代码的可读性。要指定类名前缀，请在枚举类的 **methods(Static)** 部分包含此方法：

**function retVal = addClassNameToEnumNames()**

**% ADDCLASSNAMETOENUMNAMES Specifies whether to add the class name**

**% as a prefix to enumeration member names in generated code.**

**% Return true or false.**

**% If you do not define this method, no prefix is added. retVal = true;**

**end**

可以将返回值指定为 **true** 以启用类名前缀，或者指定为 **false** 以禁用前缀。如果指定 **true**，枚举类中的每个枚举值在生成的代码中都将以 **EnumTypeName\_EnumName** [的形式显示。对于“枚举数据类](#_bookmark117)

[型” （第 16-21](#_bookmark117) 页）中的示例枚举类 **BasicColors**，生成的代码中的数据类型定义可能如下所示：

**#ifndef \_DEFINED\_TYPEDEF\_FOR\_BasicColors\_ #define \_DEFINED\_TYPEDEF\_FOR\_BasicColors\_**

**typedef enum {**

**BasicColors\_Red = 0, /\* Default value \*/ BasicColors\_Yellow = 1,**

**BasicColors\_Blue = 2,**

**} BasicColors; #endif**

枚举类名称 **BasicColors** 显示为每个枚举名称的前缀。

#### 控制重复枚举成员名称的使用

从头文件导入枚举数据时，您可以控制在代码生成过程中重复枚举成员名称的使用。重复使用枚举成员名称可提高代码的可读性。使用模型配置参数 **Duplicate enumeration member names** 可在代码生成期

间允许对不同枚举类型使用重复枚举成员名称，或生成错误或警告消息。仅当两个枚举具有相同的

**StorageType** 并具有以下设定时，才能使用重复的枚举成员名称：

#### DataScope 设置为 'Imported'

* **StorageType** 设置为 **'int8'**、**'int16'**、**'int32'**、**'uint8'** 或 **'uint16'**
* **Value** 相同

例如：

**typedef int32\_T enum { Red = 0,**

**Yellow = 1,**

**Blue = 2,**

**}A;**

**typedef int32\_T enum { Black = 0,**

**Yellow = 1,**

**White = 2,**

**}B;**

您可以在枚举 **A** 和 **B** 中使用 **Yellow** 枚举成员，而无需在成员名称前面加上类名，从而提高代码的可读性。

### 在生成的代码中控制枚举类型实现

假设您定义了一个枚举类型 **BasicColors**。您可以指定生成的代码通过以下方式实现该类型定义：

* 使用 **enum** 模块。硬件的原生整数类型是枚举成员的基础整数类型。
* 使用 **typedef** 语句和一系列 **#define** 宏。**typedef** 语句基于特定的整数数据类型（例如 **int8**）定义枚举类型名称。宏将枚举成员与基础整数值相关联。

#### 使用 enum 模块实现枚举类型

要通过使用 **enum** 模块实现类型定义，请执行以下操作：

* 在 Simulink 中，使用 **classdef** 模块在脚本文件中定义枚举类型。从类型 **Simulink.IntEnumType**

派生枚举。

* 也可以使用函数 **Simulink.defineIntEnumType**。不要指定属性 **StorageType**。

生成代码时，类型定义显示在 **enum** 模块中。

**#ifndef \_DEFINED\_TYPEDEF\_FOR\_BasicColors\_ #define \_DEFINED\_TYPEDEF\_FOR\_BasicColors\_**

**typedef enum {**

**Red = 0, /\* Default value \*/ Yellow,**

**Blue,**

**} BasicColors;**

**#endif**

#### 使用特定的整数类型实现枚举类型

要使用 **typedef** 语句和 **#define** 宏实现类型定义，请执行以下操作：

* 在 Simulink 中，使用 **classdef** 模块在脚本文件中定义枚举类型。从特定的整数类型（如 **int8**）派生枚举。
* 也可以使用函数 **Simulink.defineIntEnumType**。使用特定的整数类型（例如 **int8**）指定属性 **StorageType**。

当您生成代码时，类型定义显示为 **typedef** 语句和一系列 **#define** 宏。

**#ifndef \_DEFINED\_TYPEDEF\_FOR\_BasicColors\_ #define \_DEFINED\_TYPEDEF\_FOR\_BasicColors\_**

**typedef int8\_T BasicColors;**

**#define Red ((BasicColors)0) /\* Default value \*/ #define Yellow ((BasicColors)1)**

**#define Blue ((BasicColors)2) #endif**

默认情况下，生成的文件 **model\_types.h** 包含枚举类型定义。

### 枚举的类型转换

#### 安全转换

Simulink Data Type Conversion 模块接受整数类型的信号。此模块可将输入转换为枚举类型的基础值之一。

如果输入值与枚举类型值的基础值不匹配，Simulink 将插入一个安全转换，用枚举类型的默认值替换输入值。

#### 启用和禁用安全转换

您可以在代码生成期间为 Simulink Data Type Conversion 模块或 Stateflow 模块启用或禁用枚举的安全转换。

要控制安全转换，请启用或禁用 **Saturate on integer overflow** 模块参数。此参数的工作原理如下：

* **启用**：Simulink 在仿真期间会使用枚举值的默认值替换不匹配的输入值。软件将在代码生成期间生成安全转换函数。
* **禁用**：Simulink 在仿真期间遇到不匹配的输入值时会产生一个错误。软件将在代码生成过程中忽略安全转换函数。这种情况下，代码的效率更高，但可能更容易出现运行时错误。

#### 生成的代码中的安全转换函数

此示例说明为 32 位硬件生成代码时，枚举 **BasicColors** 的安全转换函数 **int32\_T ET08\_safe\_cast\_to\_BasicColors** 在生成代码中如何显示。

**static int32\_T ET08\_safe\_cast\_to\_BasicColors(int32\_T input)**

**{**

**int32\_T output;**

**/\* Initialize output value to default value for BasicColors (Red) \*/**

**output = 0;**

**if ((input >= 0) && (input <= 2)) {**

**/\* Set output value to input value if it is a member of BasicColors \*/ output = input;**

**}**

**return output;**

**}**

通过此函数，当输入值与枚举类型值的基础值不匹配时，将使用枚举类型的默认值。

如果禁用了模块的 **Saturate on integer overflow** 参数，此函数将不会出现在生成的代码中。

### 有关枚举类型的限制

* 生成的代码不支持记录枚举数据。

### 另请参阅

**enumeration** | **Simulink.defineIntEnumType** | **Simulink.data.getEnumTypeInfo**

### 相关示例

* “在 Simulink 模型中使用枚举数据”
* “Simulink 枚举”
* “Exchange Structured and Enumerated Data Between Generated and External Code” (Embedded Coder)

## 在生成的代码中创建可调标定参数

标定参数是存储在全局内存中的一个值，供算法读取用于计算但不写入。标定参数是可调的，因为您可以在执行算法的过程中更改存储的值。创建标定参数可以：

* 通过在执行过程中调整参数和监控信号值，确定最佳参数值。
* 通过覆盖内存中存储的参数值，有效地调整算法以适应不同的执行条件。例如，通过在每个车辆的引擎控制单元存储不同的参数值，可对不同质量的多个车辆使用同一个控制算法。

在 Simulink 中，可以创建一个 **Simulink.Parameter** 对象来表示标定参数。您可以使用参数对象来设置模块参数值，例如 Gain 模块的 **Gain** 参数。要控制参数对象在生成的代码中的表示，请对该对象应用存储类。

要使模块参数在生成的代码中默认可访问（例如为了建立快速原型），请将 **Default parameter behavior**（请参阅 Default parameter behavior）设置为 “**Tunable**”。有关详细信息，请参阅 “Preserve Variables in Generated Code”。

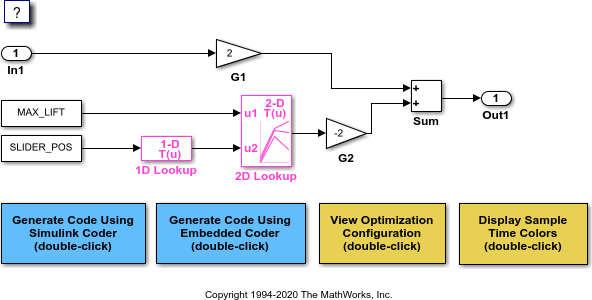
### 将模块参数表示为可调全局变量

此示例说明如何在生成的代码中通过将模块参数表示为全局变量来创建可调参数数据。

#### 使用参数对象配置模块参数

打开示例模型 **rtwdemo\_paraminline** 并将其配置为显示生成的模块名称。

**load\_system('rtwdemo\_paraminline') set\_param('rtwdemo\_paraminline','HideAutomaticNames','off') open\_system('rtwdemo\_paraminline')**



在 **Modeling** 选项卡上，点击 **Model Data Editor**。

在模型数据编辑器中，检查 **Parameters** 选项卡。

在模型中，点击 Gain 模块 **G1**。模型数据编辑器突出显示与模块的 **Gain** 参数对应的行。在模型数据编辑器的 **Value** 列，将增益值从 **2** 更改为 **myGainParam**。

在 **myGainParam** 旁边，点击操作按钮（含三个纵点），然后选择 **Create**。

在 Create New Data 模块对话框中，将 **Value** 设置为 **Simulink.Parameter(2)**。点击 **Create**。

**Simulink.Parameter** 对象 **myGainParam** 在模型工作区中存储参数值 **2**。

在 myGainParam 对话框的 **Code Generation** 选项卡上，点击 **Configure in Coder App**。

在 Code Mappings 编辑器中，将 **myGainParam** 的 **Storage Class** 设置为 **ExportedGlobal**。此存储类使参数对象在生成的代码中显示为可调全局变量。

或者，要创建参数对象并配置模型，请在命令提示符下使用以下命令：

**set\_param('rtwdemo\_paraminline/G1','Gain','myGainParam') mws = get\_param('rtwdemo\_paraminline', 'modelworkspace'); mws.assignin('myGainParam',Simulink.Parameter(2));**

**cm = coder.mapping.utils.create('rtwdemo\_paraminline'); setModelParameter(cm,'myGainParam','StorageClass','ExportedGlobal');**

使用模型数据编辑器为 Gain 模块 **G2** 创建参数对象 **myOtherGain**。应用存储类 **ExportedGlobal**。也可以在命令提示符下使用以下命令：

**set\_param('rtwdemo\_paraminline/G2','Gain','myOtherGain') mws.assignin('myOtherGain',Simulink.Parameter(-2)); setModelParameter(cm,'myOtherGain','StorageClass','ExportedGlobal');**

#### 生成和检查代码

从模型中生成代码。

**slbuild('rtwdemo\_paraminline')**

**### Starting build procedure for: rtwdemo\_paraminline**

**### Successful completion of build procedure for: rtwdemo\_paraminline Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_paraminline Code generated and compiled Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 9.502s**

生成的文件 **rtwdemo\_paraminline.h** 包含全局变量 **myGainParam** 和 **myOtherGain** 的 **extern**

声明。您可以包含 (**#include**) 此头文件，以使您的代码能够在执行过程中读写变量的值。

**file = fullfile('rtwdemo\_paraminline\_grt\_rtw','rtwdemo\_paraminline.h'); rtwdemodbtype(file,...**

**'extern real\_T myGainParam;','Referenced by: ''<Root>/G2''',1,1)**

**extern real\_T myGainParam; /\* Variable: myGainParam**

* **Referenced by: '<Root>/G1'**

**\*/**

**extern real\_T myOtherGain; /\* Variable: myOtherGain**

* **Referenced by: '<Root>/G2'**

文件 **rtwdemo\_paraminline.c** 为 **myGainParam** 和 **myOtherGain** 分配内存并进行初始化。

**file = fullfile('rtwdemo\_paraminline\_grt\_rtw','rtwdemo\_paraminline.c'); rtwdemodbtype(file,...**

**'/\* Exported block parameters \*/','Referenced by: ''<Root>/G2''',1,1)**

**/\* Exported block parameters \*/**

**real\_T myGainParam = 2.0; /\* Variable: myGainParam**

* **Referenced by: '<Root>/G1'**

**\*/**

**real\_T myOtherGain = -2.0; /\* Variable: myOtherGain**

* **Referenced by: '<Root>/G2'**

模型 **step** 函数中生成的代码算法使用 **myGainParam** 和 **myOtherGain** 进行计算。

**rtwdemodbtype(file,...**

**'/\* Model step function \*/','/\* Model initialize function \*/',1,0)**

**/\* Model step function \*/**

**void rtwdemo\_paraminline\_step(void)**

**{**

**/\* Outport: '<Root>/Out1' incorporates:**

* **Gain: '<Root>/G1'**
* **Gain: '<Root>/G2'**
* **Inport: '<Root>/In1'**
* **Sum: '<Root>/Sum'**

**\*/**

**rtwdemo\_paraminline\_Y.Out1 = myGainParam \* rtwdemo\_paraminline\_U.In1 + myOtherGain \* -75.0;**

**}**

### 当模块参数引用 MATLAB 数值变量时应用存储类

如果使用数值变量来设置模块参数的值，则不能将存储类应用于该变量。作为解决方法，您可以将该变量转换为参数对象，然后对该对象应用一个存储类。要将变量转换为参数对象，请选择下列方法之一：

* 在模型数据编辑器的 **Parameters** 选项卡上，将 **Change view** 设置为 “**Code**”，找到与该变量对应的行。在 **Storage Class** 列中，从下拉列表中选择 “**Convert to parameter object**”。模型数据编辑器会将变量转换为参数对象。然后，使用 **Storage Class** 列将存储类应用于该对象。

您也可以在模型资源管理器中使用这种方法。

* 使用 Data Object Wizard（请参阅“使用数据对象向导为模型创建数据对象”）。在该向导中，选中 **Parameters** 复选框。该向导将变量转换为对象。然后，使用模型数据编辑器或模型资源管理器对这些对象应用存储类。

### 创建表示标定参数的存储类 (Embedded Coder)

以下示例说明如何创建在生成的代码中生成标定参数的存储类。存储类使每个参数对象 (**Simulink.Parameter**) 显示为具有特殊装饰（例如关键字和 pragma 指令）的全局变量。

在生成的代码中，标定参数必须显示为在文件 **calPrms.c** 中定义并在 **calPrms.h** 中声明的全局变量。变量定义必须如下所示：

**#pragma SEC(CALPRM)**

**const volatile float param1 = 3.0F; const volatile float param2 = 5.0F; const volatile float param3 = 7.0F;**

**#pragma SEC()**

这些变量使用关键字 **const** 和 **volatile**。pragma 指令 **#pragma SEC(CALPRM)** 控制变量在内存中的位置。要实现这一 pragma 指令，变量定义必须出现在相邻的代码块中。

此外，生成的代码必须包含每个参数的 ASAP2 (**a2l**) 描述。

#### 创建用于存储存储类和内存段定义的包

现在，通过复制示例包 **+SimulinkDemos** 在当前文件夹中创建一个包。该包存储 **Parameter** 和 **Signal** 类的定义，稍后您将使用这些类将存储类应用于模型中的数据元素。稍后，该包还会存储存储类和关联内存段的定义。

1. 将当前 MATLAB 文件夹设置为可写位置。
2. 将 **+SimulinkDemos** 包文件夹复制到当前文件夹中。将副本命名为 **+myPackage**。

**copyfile(fullfile(matlabroot,... 'toolbox','simulink','simdemos','dataclasses','+SimulinkDemos'),... '+myPackage','f')**

1. 在 **+myPackage** 文件夹内导航到文件 **Signal.m** 以编辑 **Signal** 类的定义。
2. 对 **methods** 中定义 **setupCoderInfo** 方法的部分取消注释。在对函数 **useLocalCustomStorageClasses** 的调用中，用 **'myPackage'** 替换 **'packageName'**。完成后，该部分如下所示：

**methods**

**function setupCoderInfo(h)**

**% Use custom storage classes from this package useLocalCustomStorageClasses(h, 'myPackage'); end**

**end % methods**

1. 保存并关闭文件。
2. 在 **+myPackage** 文件夹内导航到文件 **Parameter.m** 以编辑 **Parameter** 类的定义。取消定义

**setupCoderInfo** 方法的 **methods** 部分的注释，并用 **'myPackage'** 替换 **'packageName'**。

1. 保存并关闭文件。

#### 创建存储类和内存段

1. 将当前文件夹设置为包含包文件夹 **+myPackage** 的文件夹。
2. 打开 Custom Storage Class Designer。

**cscdesigner('myPackage')**

1. 在 Custom Storage Class Designer 中的 **Memory Sections** 选项卡上，点击 **New**。
2. 对于新内存段，根据下表设置属性。

|  |  |
| --- | --- |
| **属性** | **值** |
| **Name** | **CalMem** |
| **Statements Surround** | “**Group of variables**” |
| **Pre Statement** | **#pragma SEC(CALPRM)** |
| **Post Statement** | **#pragma SEC()** |
| **Is const** | 选择 |
| **Is volatile** | 选择 |

1. 点击**应用**。
2. 在 **Custom Storage Class** 选项卡上，点击 **New**。
3. 对于新存储类，根据下表设置属性。

|  |  |
| --- | --- |
| **属性** | **值** |
| **Name** | **CalParam** |
| **For signals** | 清除 |
| **Data scope** | “**Exported**” |
| **Header File** | **calPrms.h** |
| **Definition File** | **calPrms.c** |
| **Memory Section** | “**CalMem**” |

1. 点击 **OK**。出现询问是否保存更改的消息时，点击 **Yes**。**将默认参数对象设置为 myPackage.Parameter**

为了更轻松地应用存储类，请使用模型资源管理器将默认参数对象从 **Simulink.Parameter** 更改为

#### myPackage.Parameter。

1. 在命令提示符下，打开模型资源管理器。

**daexplr**

1. 在模型资源管理器的 **Model Hierarchy** 窗格中，选择 **Base Workspace**。
2. 在模型资源管理器工具栏中，点击 **Add Simulink Parameter** 按钮旁边的箭头。在下拉列表中，选择 **Customize class lists**。
3. 在 **Customize class lists** 对话框的 **Parameter classes** 下，选中 **myPackage.Parameter** 旁边的复选框。点击 **OK**。
4. 在模型资源管理器工具栏中，点击 **Add Simulink Parameter** 按钮旁边的箭头。在下拉列表中，选择 **myPackage Parameter**。

基础工作区中将显示一个 **myPackage.Parameter** 对象。您可以删除此对象。现在，当您使用诸如模型数据编辑器之类的工具创建参数对象时，Simulink 将创建

#### myPackage.Parameter 对象而不是 Simulink.Parameter 对象。

**应用存储类**

在示例模型 **rtwdemo\_roll** 中，**BasicRollMode** 子系统表示 PID 控制器。将 P、I 和 D 参数配置为标定参数。

1. 打开模型。

**rtwdemo\_roll**

1. 在模型中，导航到 **BasicRollMode** 子系统中。
2. 在 App 库中，点击 **Embedded Coder**。
3. 在模块图下方，通过选择 **Model Data Editor** 选项卡打开模型数据编辑器。
4. 在模型数据编辑器中，选择 **Parameters** 选项卡并更新模块图。

现在，数据表包含与 Gain 模块使用的工作区变量（代表控制器的 P、I 和 D 参数）对应的行。

1. 在模型数据编辑器中，在 **Filter contents** 框的旁边，激活 **Filter using selection** 按钮。
2. 在模型中，选择三个 Gain 模块。
3. 在 **Filter contents** 框中，输入 **model workspace**。

Gain 模块使用的变量位于模型工作区中。

1. 在数据表中，选择三个行，并在一行的 **Storage Class** 列中选择 “**Convert to parameter object**”。

模型数据编辑器将工作区变量转换为 **myPackage.Parameter** 对象。现在，您可以对这些对象应用存储类。

1. 在一行的 **Storage Class** 列中，选择 “**CalParam**”。

#### 生成和检查代码

1. 从模型中生成代码。
2. 在代码生成报告中，检查文件 **calPrms.c**。该文件定义标定参数。

**/\* Exported data definition \*/ #pragma SEC(CALPRM)**

**/\* Definition for custom storage class: CalParam \*/ const volatile real32\_T dispGain = 0.75F;**

**const volatile real32\_T intGain = 0.5F; const volatile real32\_T rateGain = 2.0F;**

**#pragma SEC()**

文件 **calPrms.h** 声明参数。

1. 检查接口文件 **rtwdemo\_roll.a2l**。该文件包含有关每个参数的信息，例如 **dispGain**。

**/begin CHARACTERISTIC**

**/\* Name \*/ dispGain**

**/\* Long Identifier \*/ ""**

**/\* Type \*/ VALUE**

**/\* ECU Address \*/ 0x0000 /\* @ECU\_Address@dispGain@ \*/**

**/\* Record Layout \*/ Scalar\_FLOAT32\_IEEE**

**/\* Maximum Difference \*/ 0**

**/\* Conversion Method \*/ rtwdemo\_roll\_CM\_single**

**/\* Lower Limit \*/ -3.4E+38**

**/\* Upper Limit \*/ 3.4E+38**

**/end CHARACTERISTIC**

### 从系统常量或其他宏初始化参数值 (Embedded Coder)

您可以生成使用从某些系统常量（宏）计算的值初始化可调参数的代码。例如，您可以生成以下代码，这些代码使用从宏 **numVessels** 和 **vesInitVol** 计算出的值初始化可调参数 **totalVol**：

**#define numVessels 16**

**#define vesInitVol 18.2**

**double totalVol = numVessels \* vesInitVol;**

这种初始化方法会保留可调参数和系统常量之间的数学关系，从而使生成的代码更易读、更易于维护。要生成此代码，请执行下列操作：

1. 创建表示系统常量的参数对象。

**numVessels = Simulink.Parameter(16); vesInitVol = Simulink.Parameter(18.2);**

1. 将这些对象配置为使用存储类 **Define**，这会在生成的代码中生成一个宏。

**numVessels.CoderInfo.StorageClass = 'Custom'; numVessels.CoderInfo.CustomStorageClass = 'Define';**

**vesInitVol.CoderInfo.StorageClass = 'Custom'; vesInitVol.CoderInfo.CustomStorageClass = 'Define';**

1. 创建另一个表示可调参数的参数对象。将该对象配置为使用存储类 **ExportedGlobal**，这会在生成的代码中生成一个全局变量。

**totalVol = Simulink.Parameter; totalVol.CoderInfo.StorageClass = 'ExportedGlobal';**

1. 使用表达式 **numVessels \* vesInitVol** 设置 **totalVol** 的值。要指定生成的代码保留表达式，请使用 **slexpr** 函数。

**totalVol.Value = slexpr('numVessels \* vesInitVol');**

1. 使用 **totalVol** 在模型中设置模块参数值。您从模型生成的代码使用基于系统常量的值来初始化可调参数。

有关使用表达式设置 **Simulink.Parameter** 对象的值的详细信息和限制，请参阅“使用数学表达式设置变量值”。

### 存储位置对参数对象的代码生成的影响

您可以在基础工作区、模型工作区或数据字典中创建参数对象。但是，当您结束 MATLAB 会话时，基础工作区中的变量将被删除。要确定模型使用的参数对象和其他变量的存储位置，请参阅“确定在何处存储 Simulink 模型的变量和对象”。

参数对象的位置可能会影响生成的代码中对应数据定义的文件位置。

* + 如果您将一个参数对象放在基础工作区或数据字典中，代码生成器将假定对应的参数数据（例如全局变量）属于您从中生成代码的系统，而不是属于系统中的特定组件。例如，如果模型引用层次结构中的某个模型使用了一个参数对象，而该对象的存储类不是 **Auto**，则数据定义将显示在为层次结构中的顶层模型生成的代码中，而不是为使用该对象的模型生成的代码中。

但是，如果您拥有 Embedded Coder，则可以使用某些存储类指定数据片段所属模型的名称。指定所属模型后，为该模型生成的代码将定义数据。有关数据所属关系的详细信息，请参阅“Control Placement of Global Data Definitions and Declarations in Generated Files” (Embedded Coder)。

* + 如果您将参数对象放在模型工作区中，代码生成器将假定数据属于该模型。如果您从包含所属模型的引用层次结构中生成代码，数据定义将显示在为所属模型生成的代码中。

有关数据所属关系的详细信息，请参阅“Control Placement of Global Data Definitions and Declarations in Generated Files” (Embedded Coder)。

* + 如果您对参数对象应用除 **Auto** 以外的存储类，该对象在生成的代码中将显示为全局符号。因此，在模型引用层次结构中，位于不同模型工作区或字典中的两个这样的对象不能具有相同的名称。每个对象的名称在整个模型层次结构中必须唯一。

但是，如果您有 Embedded Coder，则可以使用存储类 **FileScope** 来防止不同模型工作区中参数对象之间的名称冲突。请参阅“使用 Struct 存储类将参数数据组织为结构体” (Embedded Coder)。

如果您将 **AUTOSAR.Parameter** 对象存储在模型工作区中，代码生成器将忽略您为该对象指定的存储类。

### 配置信号数据的可访问性

在算法执行过程中调整参数值时，可以监控或捕获输出信号值，以分析调整的结果。要在生成的代码中将信号表示为可访问数据，可以使用测试点和存储类等技术。请参阅“Preserve Variables in Generated Code”。

### 用于调整参数的编程接口

您可以配置生成的代码以包括：

* + C 应用程序编程接口 (API)，用来独立于外部模式对参数进行调整。生成的代码包括额外的代码，以便于您编写自己的代码来访问参数值。请参阅“使用 [C API 在生成的代码和外部代码之间交换数据”](#_bookmark348)

[（第 34-2](#_bookmark348) 页）。

* + Target Language Compiler API，用来独立于外部模式对参数进行调整。请参阅“Parameter Functions”。

### 设置可调参数的最小值和最大值

为可调参数指定最小值和最大值是一种不错的做法。您可以通过以下方式指定最小值和最大值：

* + 在使用参数对象的模块对话框中。使用这种方式将最小值和最大值信息存储在模型中。
  + 通过使用用来设置参数值的 **Simulink.Parameter** 对象的属性。使用这种方式将最小值和最大值信息存储在模型外。

有关详细信息，请参阅“指定模块参数的最小值和最大值”。

### 其他建模目的注意事项

|  |  |
| --- | --- |
| **目的** | **注意事项和更多信息** |
| 应用存储类型限定符 **const** 和  **volatile** | 如果您有 Embedded Coder，要生成存储类型限定符，请参阅“Protect Global Data with const and volatile Type Qualifiers” (Embedded Coder)。 |
| 通过应用关键字 **static** 防止在不同组件中的参数之间发生名称冲突 | 如果您有 Embedded Coder，请使用存储类 **FileScope** 或您创建的类似存储类。请参阅“Choose Storage Class for Controlling Data Representation in Generated Code” (Embedded Coder)。 |

|  |  |
| --- | --- |
| **目的** | **注意事项和更多信息** |
| 生成 ASAP2 (**a2l**) 描述 | 您可以生成一个 **a2l** 文件，以使用 ASAP2 标准说明您的标定参数。有关详细信息，请参阅“为参数和信号定义 [ASAP2 信息” （第 35-3](#_bookmark370) 页）。 |
| 生成 AUTOSAR XML (ARXML) 描述 | 如果您拥有 Embedded Coder，可以生成一个 ARXML 文件，以说明您为 AUTOSAR 标准配置的模型所使用的标定参数。请参阅“Model AUTOSAR Calibration Parameters and Lookup Tables” (AUTOSAR Blockset)。 |
| 存储查找表数据进行标定 | 要存储查找表数据以根据 ASAP2 或 AUTOSAR 标准（例如 STD\_AXIS、 COM\_AXIS 或 CURVE）进行标定，可以在查找表模块中使用 **Simulink.LookupTable** 和 **Simulink.Breakpoint** 对象。  但是，这存在一些限制。请参阅 **Simulink.LookupTable**。要绕过 **Simulink.LookupTable** 和 **Simulink.Breakpoint** 对象的限制，可以改用 **Simulink.Parameter** 对象。  有关详细信息，请参阅“为参数和信号定义 [ASAP2 信息” （第 35-3 页）](#_bookmark370)和“Configure Lookup Tables for AUTOSAR Calibration and Measurement” (AUTOSAR Blockset)。 |
| 使用 pragma 指令将参数数据存储在特定的内存位置 | 如果您拥有 Embedded Coder 许可证，要生成包含自定义 pragma 指令的代码，请使用存储类和内存段。请参阅“Control Data and Function Placement in Memory by Inserting Pragmas” (Embedded Coder)。 |

**另请参阅**

**Simulink.Parameter** | **Simulink.LookupTable** | **Simulink.Breakpoint**

### 相关示例

* + “Exchange Data Between External C/C++ Code and Simulink Model or Generated Code”
  + “Reuse Parameter Data in Different Data Type Contexts”
  + “Limitations for Block Parameter Tunability in Generated Code”
  + [“生成的代码如何存储内部信号、状态和参数数据” （第 16-2 页）](#_bookmark109)
  + [“模型接口元素的 C 代码生成配置” （第 17-2 页）](#_bookmark135)
  + “Access Structured Data Through a Pointer That External Code Defines” (Embedded Coder)
  + “Configure Lookup Tables for AUTOSAR Calibration and Measurement” (AUTOSAR Blockset)

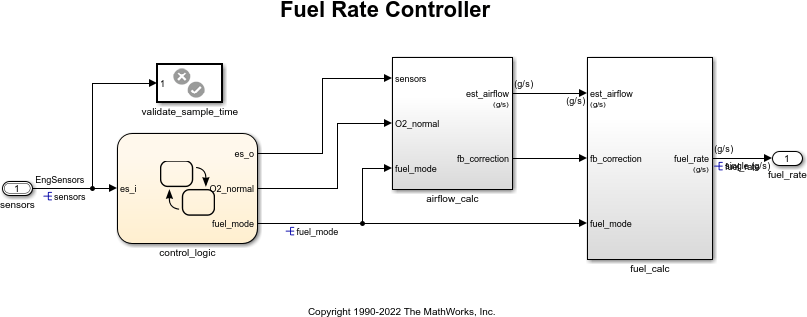
## 在仿真和代码执行期间在参数值集之间切换

要存储相同模块参数的多个独立值集，可以使用结构体数组。要在参数集之间切换，请创建一个变量作为数组的索引，并更改该变量的值。您可以在仿真期间更改变量值，如果变量可调，还可以在生成的代码执行期间更改变量值。

#### 浏览模型示例

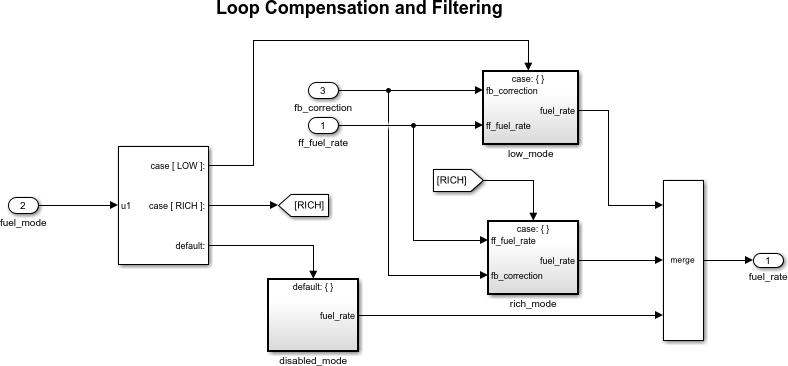
打开示例模型。

**open\_system('sldemo\_fuelsys\_dd\_controller')**



此模型代表一台汽油发动机的供油系统。模型的输出是流向引擎的燃油速率。导航到 **switchable\_compensation** 嵌套子系统。

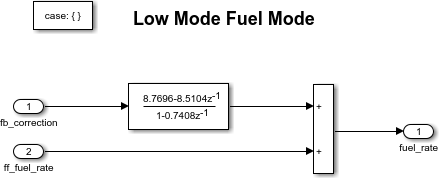
**open\_system(['sldemo\_fuelsys\_dd\_controller/fuel\_calc/',... 'switchable\_compensation'])**



此子系统对喷油量信号中的噪声进行校正和滤波。该子系统基于燃油供给模式使用不同滤波器系数，从而根据引擎中的传感器故障控制逻辑更改。例如，控制算法在正常运行期间激活 **low\_mode** 子系统。它激活 **rich\_mode** 子系统以响应传感器故障。

打开 **low\_mode** 子系统。

**open\_system(['sldemo\_fuelsys\_dd\_controller/fuel\_calc/',... 'switchable\_compensation/low\_mode'])**



Discrete Filter 模块对喷油量信号进行滤波。在模块对话框中，**分子**参数设置滤波器的分子系数。同级子系统 **rich\_mode** 还包含 Discrete Filter 模块，该模块使用不同系数。

更新模型图以显示信号数据类型。该模块的输入和输出信号使用单精度浮点数据类型 **single**。

在模型的左下角，点击模型数据标记，然后点击**数据字典**链接。此模型的数据字典

**sldemo\_fuelsys\_dd\_controller.sldd** 在模型资源管理器中打开。

在模型资源管理器的**模型层次结构**窗格中，选择**设计数据**节点。

在**目录**窗格中，查看 **Simulink.NumericType** 对象的属性，如 **s16En15**。所有这些对象当前都表示单精度浮点数据类型 **single**。该模型使用这些对象来设置信号数据类型，包括 Discrete Filter 模块的输入和输出信号。

假设在仿真和执行生成的代码的过程中，您需要其中每个子系统都基于您控制其值的变量在不同分子系数之间切换。

#### 将参数值存储在结构体数组中

将现有的分子系数集存储在其值为结构体的 **Simulink.Parameter** 对象中。该结构体的每个字段存储一个 Discrete Filter 模块的系数。

**lowBlock = ['sldemo\_fuelsys\_dd\_controller/fuel\_calc/'... 'switchable\_compensation/low\_mode/Discrete Filter'];**

**richBlock = ['sldemo\_fuelsys\_dd\_controller/fuel\_calc/'... 'switchable\_compensation/rich\_mode/Discrete Filter'];**

**params.lowNumerator = eval(get\_param(lowBlock,'Numerator')); params.richNumerator = eval(get\_param(richBlock,'Numerator')); params = Simulink.Parameter(params);**

将 **params** 的值复制到临时变量中。修改此临时结构体中的字段值，并将修改后的结构体指定为 **params**

的第二个元素。

**temp = params.Value;**

**temp.lowNumerator = params.Value.lowNumerator \* 2; temp.richNumerator = params.Value.richNumerator \* 2; params.Value(2) = temp;**

**clear temp**

**params** 的值是一个包含两个结构体的数组。每个结构体存储一个滤波器系数集。

#### 创建变量以在参数集之间切换

创建名为 **Ctrl** 的 **Simulink.Parameter** 对象。

**Ctrl = Simulink.Parameter(2); Ctrl.DataType = 'uint8';**

在 **low\_mode** 子系统的 Discrete Filter 模块对话框中，将**分子**参数设置为表达式

#### params(Ctrl).lowNumerator。

**set\_param(lowBlock,'Numerator','params(Ctrl).lowNumerator');**

在 **rich\_mode** 子系统的 Discrete Filter 模块中，将**分子**参数的值设置为

#### params(Ctrl).richNumerator。

**set\_param(richBlock,'Numerator','params(Ctrl).richNumerator');**

表达式使用变量 **Ctrl** 选择 **params** 中的一个结构体。然后表达式解引用结构体中的一个字段。该字段值设置分子系数的值。

要在系数集之间切换，只需将 **Ctrl** 的值更改为结构体数组中的对应索引。

#### 使用总线对象作为结构体数组的数据类型

（可选）创建一个 **Simulink.Bus** 对象，用作结构体数组的数据类型。您可以：

* 控制结构体的形状。
* 对于每个字段，控制数据类型和物理单位等特性。
* 在生成的代码中控制 **struct** 类型的名称。

使用函数 **Simulink.Bus.createObject** 创建对象，并将对象重命名为 **paramsType**。

**Simulink.Bus.createObject(params.Value) paramsType = slBus1;**

**clear slBus1**

您可以使用数据字典中的 **Simulink.NumericType** 对象来控制结构体字段的数据类型。在总线对象中，使用数据类型对象的名称来设置每个元素的 **DataType** 属性。

**paramsType.Elements(1).DataType = 's16En15'; paramsType.Elements(2).DataType = 's16En7';**

使用总线对象作为结构体数组的数据类型。 **params.DataType = 'Bus: paramsType';使用枚举类型切换变量**

（可选）使用枚举类型作为切换变量的数据类型。您可以将每个参数集与一个有意义的名称相关联，并限制切换变量的允许值。

创建一个名为 **FilterCoeffs** 的枚举类型。为 **params** 中的每个结构体创建一个枚举成员。将每个枚举成员的基础整数值设置为 **params** 中的对应索引。

**Simulink.defineIntEnumType('FilterCoeffs',{'Weak','Aggressive'},[1 2])**

使用枚举类型作为切换变量的数据类型。将变量的值设置为 **Aggressive**，它对应于索引 **2**。

**Ctrl.Value = FilterCoeffs.Aggressive;**

#### 向数据字典添加新对象

将您创建的对象添加到数据字典 **sldemo\_fuelsys\_dd\_controller.sldd**。

**dictObj = Simulink.data.dictionary.open('sldemo\_fuelsys\_dd\_controller.sldd'); sectObj = getSection(dictObj,'Design Data');**

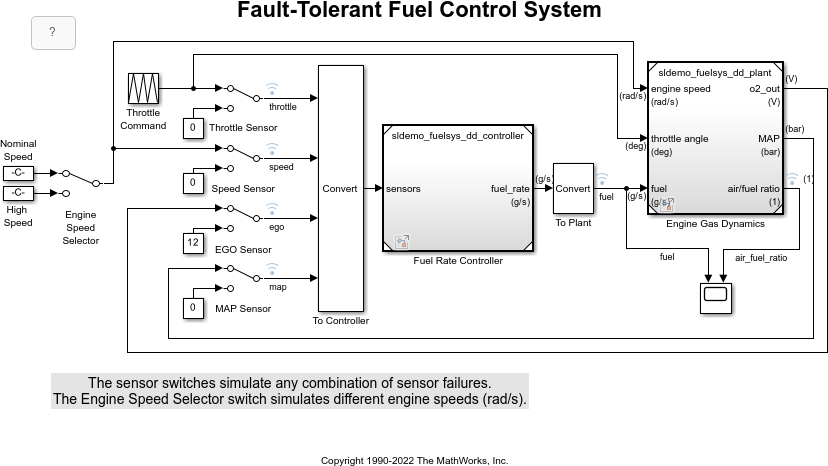
**addEntry(sectObj,'Ctrl',Ctrl) addEntry(sectObj,'params',params) addEntry(sectObj,'paramsType',paramsType)**

您也可以在数据字典中存储枚举类型。但在这种情况下，您无法导入枚举类型，因为您无法保存对 **sldemo\_fuelsys\_dd\_controller.sldd** 的更改。有关在数据字典中存储枚举类型的详细信息，请参阅 “Enumerations in Data Dictionary”。

#### 仿真期间在参数集之间切换

打开示例模型 **sldemo\_fuelsys\_dd**，它引用控制器模型 **sldemo\_fuelsys\_dd\_controller**。

**open\_system('sldemo\_fuelsys\_dd')**



将仿真停止时间设置为 **Inf**，以便您可以在仿真期间与模型交互。

开始仿真运行并打开 Scope 模块对话框。示波器显示，在引擎正常运行期间，燃油流量（**fuel** 信号）大幅波动。

在模型资源管理器中，查看数据字典 **sldemo\_fuelsys\_dd\_controller.sldd** 的内容。将 **Ctrl** 的值设置为 **FilterCoeffs.Weak**。

更新 **sldemo\_fuelsys\_dd** 模型图。示波器显示，喷油量振荡的振幅由于滤波器系数减弱而降低。停止仿真。

#### 生成和检查代码

如果您有 Simulink Coder 软件，您可以生成代码，使您能够在代码执行期间在参数集之间切换。

在模型资源管理器中，查看数据字典 **sldemo\_fuelsys\_dd\_controller.sldd** 的内容。在**目录**窗格中，将

**列视图**设置为 **Storage Class**。

使用 **StorageClass** 列将存储类 **ExportedGlobal** 应用于 **params**，以便结构体数组在生成的代码中显示为可调全局变量。将相同的存储类应用于 **Ctrl**，以便您可以在代码执行期间更改切换变量的值。

或者，要配置对象，请使用以下命令：

**tempEntryObj = getEntry(sectObj,'params'); params = getValue(tempEntryObj); params.StorageClass = 'ExportedGlobal'; setValue(tempEntryObj,params);**

**tempEntryObj = getEntry(sectObj,'Ctrl'); Ctrl = getValue(tempEntryObj); Ctrl.StorageClass = 'ExportedGlobal'; setValue(tempEntryObj,Ctrl);**

从控制器模型生成代码。

**slbuild('sldemo\_fuelsys\_dd\_controller')**

**### Starting build procedure for: sldemo\_fuelsys\_dd\_controller**

**### Successful completion of code generation for: sldemo\_fuelsys\_dd\_controller Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**sldemo\_fuelsys\_dd\_controller Code generated. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 33.598s**

在代码生成报告中，查看头文件 **sldemo\_fuelsys\_dd\_controller\_types.h**。代码定义枚举数据类型 **FilterCoeffs**。

**file = fullfile('sldemo\_fuelsys\_dd\_controller\_ert\_rtw',... 'sldemo\_fuelsys\_dd\_controller\_types.h');**

**rtwdemodbtype(file,'#ifndef DEFINED\_TYPEDEF\_FOR\_FilterCoeffs\_',... '/\* Forward declaration for rtModel \*/',1,0)**

**#ifndef DEFINED\_TYPEDEF\_FOR\_FilterCoeffs\_ #define DEFINED\_TYPEDEF\_FOR\_FilterCoeffs\_**

**typedef enum {**

**Weak = 1, /\* Default value \*/ Aggressive**

**} FilterCoeffs; #endif**

代码还定义结构体类型 **paramsType**，它对应于 **Simulink.Bus** 对象。这些字段使用模型中的单精度浮点数据类型。

**rtwdemodbtype(file,'#ifndef DEFINED\_TYPEDEF\_FOR\_paramsType\_',... '#ifndef DEFINED\_TYPEDEF\_FOR\_FilterCoeffs\_',1,0)**

**#ifndef DEFINED\_TYPEDEF\_FOR\_paramsType\_ #define DEFINED\_TYPEDEF\_FOR\_paramsType\_**

**typedef struct {**

**real32\_T lowNumerator[2]; real32\_T richNumerator[2];**

**} paramsType; #endif**

查看源文件 **sldemo\_fuelsys\_dd\_controller.c**。代码使用枚举类型来定义切换变量 **Ctrl**。

**file = fullfile('sldemo\_fuelsys\_dd\_controller\_ert\_rtw',... 'sldemo\_fuelsys\_dd\_controller.c');**

**rtwdemodbtype(file,'FilterCoeffs Ctrl = Aggressive;',... '/\* Block signals (default storage) \*/',1,0)**

**FilterCoeffs Ctrl = Aggressive; /\* Variable: Ctrl**

* **Referenced by:**
* **'<S12>/Discrete Filter'**
* **'<S13>/Discrete Filter'**

**\*/**

代码还定义结构体数组 **params**。

**rtwdemodbtype(file,'/\* Exported block parameters \*/',... '/\* Variable: params',1,1)**

**/\* Exported block parameters \*/ paramsType params[2] = { {**

**{ 8.7696F, -8.5104F },**

**{ 0.0F, 0.2592F }**

**}, { { 17.5392F, -17.0208F },**

**{ 0.0F, 0.5184F }**

**} } ; /\* Variable: params**

模型的 **step** 函数中的代码算法使用切换变量对结构体数组进行索引。

要在存储在结构体数组中的参数集之间切换，请在代码执行期间更改 **Ctrl** 的值。

关闭与数据字典的连接。此示例丢弃未保存的更改。要保存更改，请使用 **'-save'** 选项。

**Simulink.data.dictionary.closeAll('sldemo\_fuelsys\_dd\_controller.sldd','-discard')**

### 另请参阅相关示例

* “使用模块参数值进行调优和试验”
* [“在生成的代码中创建可调标定参数” （第 16-28 页）](#_bookmark123)
* “在结构体中组织相关的模块参数定义”
* “Access Structured Data Through a Pointer That External Code Defines” (Embedded Coder)

# 在生成的代码中配置数据和函数

* [“模型接口元素的 C 代码生成配置” （第 17-2 页）](#_bookmark135)
* [“为 C 代码生成配置参数” （第 17-7 页）](#_bookmark142)
* [“为模型入口函数配置生成的 C 函数接口” （第 17-15 页）](#_bookmark148)

## 模型接口元素的 C 代码生成配置

代码生成器为您提供工具来控制在生成代码中数据的表示，以及入口函数的表示（如果您有 Embedded Coder）。例如，您可以针对代码生成配置数据和函数以符合代码标准和规范，将生成的代码与外部代码集成，或满足内存要求。

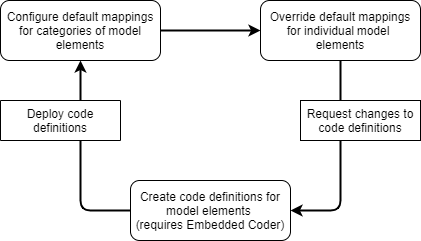
代码生成器为模型数据元素和函数生成代码：

* 应用程序代码可以读写的数据，如输入端口、信号、状态和参数。
* 入口函数，如 **model\_step**，您的应用程序代码调用这些函数来执行模型算法。

您可以使用**代码映射编辑器 - C** 或代码映射编程接口 (**coder.mapping.api.CodeMapping**) 为代码生成配置模型数据元素和函数。需要有 Embedded Coder 才可配置函数。对于数据元素，请使用 Code Mappings 编辑器或代码映射 API，而不是创建和配置 Simulink 数据对象；或者将二者相结合。编辑器和 API 使您能够配置：

* 适用于整个模型中某一类别数据元素的默认代码生成设置
* 适用于单个数据元素的配置设置

下图显示整个工作流：



### 迁移

当您打开在 R2020b 之前的版本中创建的模型时，Simulink® 会将在模型中指定的代码定义迁移到模型代码映射。

如果需要，在迁移过程中，Simulink 会配置模型使用的 Embedded Coder 字典。保存模型时，代码映射作为模型的一部分保存。

有关详细信息，例如迁移的影响以及迁移前的注意事项，请参阅“Migration of Model Data Configurations to Code Mappings”。

### 代码定义

代码定义包括存储类。存储类定义代码生成器在为关联的数据生成代码时使用的属性，如形式和位置。

内存段控制数据和函数定义在内存中的位置。在某些情况下，您可以在配置数据元素和函数时选择所需的特定内存段。

代码生成器为数据元素提供预定义的代码定义。如果您有 Embedded Coder，您可以使用 Embedded Coder 字典来创建和共享代码定义，例如，用于标准化您从多个模型生成的代码或为不同应用程序运行时环境创建定义。创建自定义定义使您能够实现预定义的定义无法满足的代码生成目标。对于数据元素，您还可以选择使用 Embedded Coder Custom Storage Class Designer 创建在包中创建的代码定义。

您使用 Embedded Coder 字典创建的代码定义会出现在**代码映射编辑器 - C** 中，并且可以使用代码映射 API 进行访问。对于使用 Custom Storage Class Designer 创建的一些定义，您可以通过配置 Embedded Coder 字典来引用它们，以使这些定义出现在 Code Mappings 编辑器中。

默认情况下，使用 Embedded Coder 字典为模型创建的代码定义存储在模型文件中。通过将代码定义存储在 Simulink 数据字典中，可以在模型和工程之间共享代码定义。使用数据字典时，要修改共享定义，只需在一个位置更改定义 - 即在 Embedded Coder 字典中。

有关详细信息，请参阅“Define Service Interfaces, Storage Classes, Memory Sections, and Function Templates for Software Architecture” (Embedded Coder)和“Create Storage Classes by Using the Custom Storage Class Designer” (Embedded Coder)。

### 数据默认配置

通过为整个模型中的数据元素的类别指定默认配置（例如，输入端口或模型工作区参数），减少为 C 代码生成准备模型的工作量。应用默认配置可以节省时间并降低在代码中引入错误的风险，尤其是对于大型模型和要从中生成多实例代码的模型更是如此。

默认配置应用于整个模型，但不包括引用模型。Simulink 将一组单独的代码映射保存为模型引用层次结构中每个模型的一部分。

如果一个模型包含某给定类别的大量元素（例如，超过 10 个），则更高效的方式是使用默认设置配置类别，然后在特殊情况下覆盖该设置。

默认设置可以帮助您减少手动数据输入。

* 向模型中添加模块和信号时，新数据元素会继承默认设置。
* 要一次性更改许多数据元素的代码生成设置，请在一个位置进行更改，即在默认设置中。
* 为了提高效率，在模型的其他位置进行更改时，请使用命名规则。

您可以使用**代码映射编辑器 - C** 的 **Data Defaults** 选项卡或 **setDataDefault** 函数来配置数据默认值。

#### 模型数据类别

您可以使用**代码映射编辑器 - C** 的 **Data Defaults** 选项卡或 **setDataDefault** 函数来配置数据默认值。在 **Data Defaults** 选项卡或函数调用中，您可以选择或指定这些数据元素类别。

|  |  |
| --- | --- |
| **模型元素类别** | **描述** |
| 输入端口 | 模型的根级输入端口，如 Inport 和 In Bus Element 模块。 |
| 输出端口 | 模型的根级输出端口，如 Outport 和 Out Bus Element 模块。 |
| 信号、状态和内部数据 | 模型内部的数据元素，如模块输出信号、离散模块状态、数据存储和过零信号。 |
| 共享局部数据存储 | 设置了 **Share across model instances** 模块参数的 Data Store Memory 模块。这些数据存储只在定义它们的模型中可访问。数据存储值在模型的实例之间共享。 |

|  |  |
| --- | --- |
| **模型元素类别** | **描述** |
| 全局数据存储 | 由基础工作区或数据字典中的信号对象定义的数据存储。一个应用程序中的多个模型可以使用这些数据存储。要在 Code Mappings 编辑器中查看和配置这些数据存储，请点击类别名称右侧的 **Refresh** 链接。点击此链接可更新模型图。 |
| 模型参数 | 在模型中定义的参数，例如模型工作区中的参数。不包括模型实参。 |
| 外部参数 | 在基础工作区或数据字典中定义为对象的参数。一个应用程序中的多个模型可以使用这些参数。要在 Code Mappings 编辑器中查看和配置这些参数，请点击类别名称右侧的 **Refresh** 链接。点击此链接可更新模型图。 |

#### 为数据配置默认代码生成设置的注意事项

在为数据元素配置默认代码生成设置之前，请考虑下表中列出的注意事项。

|  |  |
| --- | --- |
| **注意事项** | **更多信息** |
| 哪些类别与您的模型相关？ | 上表中的数据元素类别 |
| 该模型是否使用一个类别中的多个数据实例？如果答案是肯定的，则应用默认映射是有好处的。否 则，请考虑为每个数据元素单独配置代码生成。 | * “Configure Root-Level Inport Blocks for C Code Generation” * “Configure Root-Level Outport Blocks for C Code Generation” * “Configure Signal Data for C Code Generation” * [“为 C 代码生成配置参数” （第 17-7 页）](#_bookmark142) * “Configure Block States for C Code Generation” * “Configure Data Stores for C Code Generation” |
| 哪个存储类符合您对每个类别的代码生成要求？ | “Choose Storage Class for Controlling Data Representation in Generated Code” |
| 是否阻止优化从代码中消除特定数据？ | [“通过优化消除某些类别的数据” （第 17-4页）](#_bookmark139) |

在配置一个或多个模型元素类别后，设置内存段和 **Shared utilities identifier format** 模型配置参数对模型配置不起作用。在打开一个编码器时，Simulink 会将模型配置参数设置迁移到 Code Mappings 编辑 器。在迁移过程中，Simulink 会配置模型使用的 Embedded Coder 字典，如“Migration of Memory Section and Shared Utility Settings from Configuration Parameters to Code Mappings” (Embedded Coder)中所述。

#### 通过优化消除某些类别的数据

代码生成优化可以从代码中消除数据，这意味着您的应用程序代码无法与数据交互。有关通过优化消除数据的一般信息，请参阅“生成的代码如何存储内部信号、状态和参数数据” (Embedded Coder)。

优化只能消除下列类别的数据：

* **Model parameters**
* **Model parameter arguments**
* **External parameters**
* **Internal data**

在通过优化消除数据后，代码生成器会将代码映射中的默认代码生成设置应用于其余数据。对于参数和信号线，优化可以消除整个类别的数据。如果发生此类消除，则您为该类别指定的默认设置将不应用于数 据。

为了防止优化消除单个数据元素，请将存储类显式应用于单个元素。存储类控制生成代码中数据的外观。要强制数据元素使用您指定的默认存储类，请显式应用存储类 “**Model default**”。请参阅“Choose Storage Class for Controlling Data Representation in Generated Code”。

#### 共享字典数据默认值

如果您将模型链接到 Simulink 数据字典，其中包括为数据类别配置默认代码定义的编码器字典，则可以使用 Code Mappings 编辑器应用字典默认值。在 Code Mappings 编辑器的 **Data Defaults** 选项卡上，选择一个类别，并将存储类设置为 “**Dictionary Default**”。如果有人更改共享编码器字典中的默认设置，代码生成器会在为您的模型生成代码时应用更新后的默认设置。请参阅“Configure Default Code Mapping in a Shared Dictionary” (Embedded Coder)。

### 单个数据元素的配置

在配置模型范围的默认设置后，您可以覆盖单个数据元素的默认值。单个数据元素配置应用于模型中的一个特定数据元素。在下列情况下可以单独配置数据元素：

* 模型包括给定类别的一些元素，这些元素具有不同源、命名或位置要求。
* 您配置了默认设置，并希望覆盖单个元素的默认设置。

对于单个数据元素，最初存储类设置为 “**Auto**”，这意味着代码生成器可能出于优化目的而消除或更改相关代码的表示。如果无法进行优化，代码生成器将应用模型默认配置。

* 要避免优化并强制代码生成器使用默认配置，请将存储类设置为 “**Model default**”。
* 要覆盖默认配置，请指定满足该数据元素的代码生成要求的存储类。

如果模型中的一个数据元素未命名（例如，信号没有标签或状态没有名称），则在为 C 代码生成配置该数据元素时，必须为其指定代码标识符。通过为存储类属性 **Identifier** 指定值来设置代码标识符。

### 配置数据

1. 打开 Simulink Coder。
2. 为数据元素的类别配置默认映射。选择 **Code Interface** > **Default Code Mappings**。在 Code Mappings 编辑器的 **Data Defaults** 选项卡上，选择一个模型元素类别并设置存储类。在属性检查器中，设置存储类属性以符合模型要求。
3. 决定是否覆盖单个数据元素的默认配置设置。如果您选择不覆盖设置，请转至步骤 8。
4. 确定要单独配置的 MATLAB 变量，并将配置存储在模型文件中。在模型资源管理器中，将变量转换为

**Simulink.Parameter** 对象。生成的数据对象作为模型参数出现在 Code Mappings 编辑器中。

1. 标识那些表示在生成代码执行时要监控的算法数据的信号，并单独配置那些信号。对于您标识的每个信号，请执行以下操作之一：
   * 对于特定于某模型（非共享）的信号数据，将信号添加到 Code Mappings 编辑器中。
   * 对于共享信号数据，创建和配置 **Simulink.Signal** 对象。
2. 配置单个数据元素。在 Code Mappings 编辑器中，点击一个类别选项卡，选择数据元素，然后设置存储类。在属性检查器中，设置存储类属性值。要更改模型配置参数 **Default parameter behavior**的设置，请点击指示存储类设置为 “**Auto**” 时的行为的链接。在 Model Configuration Parameters 对话框中，将参数设置更改为 “**Tunable**” 或 “**Inline**”。

有关配置数据元素的特定类型的信息，请参阅：

1. 要查看和配置存储在基础工作区或数据字典中的外部数据对象，请在 Code Mappings 编辑器的

**Data Defaults** 选项卡上，点击类别名称右侧的 **Refresh** 链接。

1. 生成和查看代码。

有关详细信息和示例，请参阅：

#### 代码映射编辑器 - C

* “Configure Root-Level Inport Blocks for C Code Generation”
* “Configure Root-Level Outport Blocks for C Code Generation”
* “Configure Signal Data for C Code Generation”
* [“为 C 代码生成配置参数” （第 17-7 页）](#_bookmark142)
* “Configure Block States for C Code Generation”
* “Configure Data Stores for C Code Generation”

### 另请参阅

**代码映射编辑器 - C** | **coder.mapping.api.CodeMapping**

### 详细信息

* “Configure Root-Level Inport Blocks for C Code Generation”
* “Configure Root-Level Outport Blocks for C Code Generation”
* “Configure Signal Data for C Code Generation”
* [“为 C 代码生成配置参数” （第 17-7 页）](#_bookmark142)
* “Configure Block States for C Code Generation”
* “Configure Data Stores for C Code Generation”
* “Choose Storage Class for Controlling Data Representation in Generated Code”

## 为 C 代码生成配置参数

为代码生成配置模型时，您可以标识和配置参数以支持调整，例如标定。下表列出了您可以配置的参数的类型。

|  |  |
| --- | --- |
| **参数的类型** | **描述** |
| 模型参数 | 在模型中定义的参数，例如模型工作区中的参数。 |
| 外部参数 | 在基础工作区或数据字典中定义为对象的参数。 |

要为代码生成配置一个参数，必须将该参数与一个数据对象相关联。例如，在为代码生成配置 MATLAB 变量之前，在模型资源管理器中，将变量转换为参数对象。

当您在 **Simulink Coder** 中打开模型时，与数据对象相关联的模型参数会出现在 Code Mappings 编辑器中。在该编辑器中，您可以为代码生成配置这些参数。如果模型使用外部参数，您可以通过点击参数名称右侧的 **Refresh** 链接将这些参数添加到 Code Mappings 编辑器中。该链接启动更新图，并在编辑器视图中添加模型使用的外部参数。

将参数数据配置为：

* 在生成的代码执行时，可访问数据以进行交互。
* 最小化存储在内存中的数据量。
* 将参数数据提升到模型接口，以便其他组件和系统可以访问这些数据。
* 提高生成代码的可读性和可追溯性。

要控制整个模型中的参数是否可调，您可以使用 Code Mappings 编辑器的 **Data Defaults** 选项卡上的 **'Auto' will be tunable/inline** 链接来获得对模型配置参数 **Default parameter behavior** 的访问权限。例如，在以下情况下，您可以使用该链接使参数可调：

* 在快速原型过程中细化参数设置
* 标定参数
* 为生产代码优化参数

对于代码生成，示例说明如何为模型 **rtwdemo\_configrpinterface** 配置模型参数和模型实参。您可以使用**代码映射编辑器 - C** 或代码映射编程接口 (**coder.mapping.api.CodeMapping**) 来配置代码映射。

### 选择参数的自定义选项

默认情况下，模型中的参数在生成的代码中显示为名为 **model\_P** 的全局数据结构体的字段。根据您的代码接口要求，决定是否自定义参数数据的生成。如果不配置自定义，代码生成器将出于优化目的确定是消除还是更改生成代码中的参数表示。如果您配置自定义，请决定：

* 是否设置默认配置

如果一个模型包含大量必须可调的类别参数（例如，超过 10 个），则更高效的方式是使用默认设置配置该类别的参数，然后在特殊情况下覆盖该设置。如果模型包括一个给定类别的若干元素，且这些元素具有不同源、命名或位置要求，请考虑单独配置这些参数。

* 如何在生成的代码中声明和处理模型参数数据
  + 作为单独的全局变量
  + 从在外部代码中定义的全局变量读取模型参数数据
  + 作为对访问函数的调用。需要 Embedded Coder

有关这些选项的详细信息，请参阅“Control Data and Function Interface in Generated Code”。模型参数的其他考虑事项包括是否要进行以下操作：

* 通过使用模型中的参数名称或使用唯一代码标识符来命名生成代码中的参数。
* 支持由编译器标志或选项定义的预处理器条件句。需要 Embedded Coder。请参阅“使用变体模型生成使用 C 预处理器条件句的代码” (Embedded Coder)。
* 在全局变量定义和声明中包含 **static** 类型限定符（例如，用于防止名称冲突）。需要 Embedded Coder。请参阅“Prevent Name Clashes by Configuring Data Item as static” (Embedded Coder)。
* 在全局变量定义和声明中包含 **const**、**volatile** 或 **const** 和 **volatile** 类型限定符。需要 Embedded Coder。请参阅“Protect Global Data with const and volatile Type Qualifiers” (Embedded Coder)。
* 生成宏 (**#define**) 或使用在外部头文件中定义的宏的代码。需要 Embedded Coder。请参阅“宏定义 (#define)” (Embedded Coder)。
* 使用您指定的名称生成全局数据结构体。需要 Embedded Coder。请参阅“Organize Data into Structures in Generated Code” (Embedded Coder)。
* 将参数数据放入内存的特定区域。需要 Embedded Coder。请参阅“Control Data and Function Placement in Memory by Inserting Pragmas” (Embedded Coder)。

要了解具有对应存储类和存储类属性的参数的接口要求，请参阅“Choose Storage Class and Storage Class Properties for Data Stores”。

示例模型 **rtwdemo\_configrpinterface** 的参数要求是：

* 默认情况下，在生成的代码中保留模型参数以用于调整。不要通过内联参数来优化代码。
* 将前缀 **mp\_** 应用于表示模型参数的变量的名称。

对于此示例，请在 **rtwdemo\_configrpinterface** 中配置模型参数，以满足这些代码生成要求。

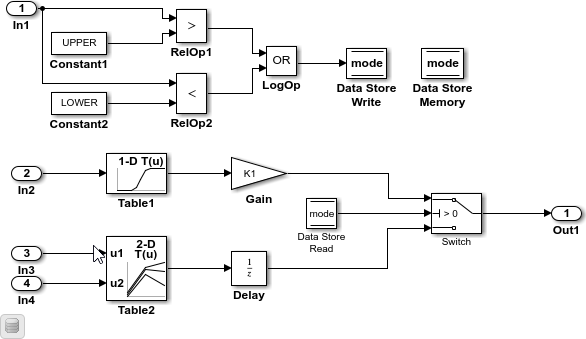
### 为参数配置默认代码生成设置

参数的默认代码生成设置可以减少为代码生成准备模型的工作量，特别是在模型有大量参数且您在生成代码执行时需要与之交互的情况下。选择一次配置设置，代码生成器即会将这些设置应用于整个模型中的参数。Simulink 将默认配置存储为模型的一部分。

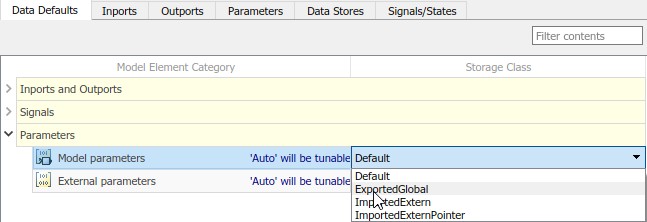
如果您的模型使用同一类别的多个参数且这些参数没有唯一性要求，请考虑为模型参数配置默认代码生成设置。

此示例说明如何使用 **Code Mappings 编辑器 - C** 为模型 **rtwdemo\_configrpinterface** 配置模型参数的默认设置。配置模型参数，使其可调并在生成代码中定义和声明为单独全局变量。

1. 打开模型 **rtwdemo\_configrpinterface**。将模型的副本保存到可写位置。



1. 打开 **Simulink Coder**。
2. 在 **C Code** 选项卡中，选择 **Code Interface** > **Default Code Mappings**。
3. 在 Code Mappings 编辑器中的 **Parameters** 下，选择类别 **Model parameters**。链接文本 **'Auto' will be inlined** 指示代码生成器默认配置为内联模型参数。此示例的一个要求是模型参数是可调的。点击 **'Auto' will be inlined**。
4. 在 Model Configuration Parameters 对话框中，将模型配置参数 **Default parameter behavior** 设置为 “**Tunable**”。保存更改并关闭对话框。在 Code Mappings 编辑器中，链接文本更改为 **'Auto' will be tunable**。
5. 在 Code Mappings 中，在 **Model parameters** 类别仍处于选中状态下，将存储类设置为 “**ExportedGlobal**”。



1. 保存模型。

### 为单个参数配置代码生成

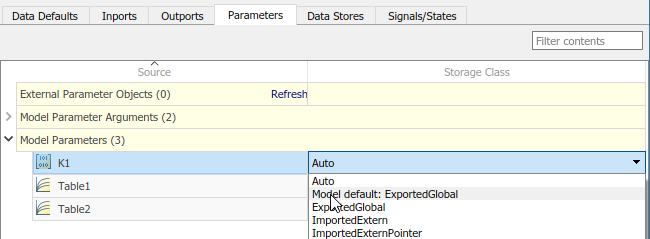
您可以为单个参数配置代码生成。例如，如果模型有属于同一类别的两个参数，且这两个参数具有不同的代码生成要求，请单独配置这些参数。或者，如果您为一类参数配置了默认设置，则可以覆盖特定参数的那些设置。

如果您的模型满足下列中的至少一个标准，请考虑为参数单独配置代码生成设置：

* 使用属于同一类别的多个参数，且这些参数具有不同源、命名或位置要求。
* 使用属于同一类别的若干参数。
* 某一类参数具有默认配置，您需要覆盖某些特定参数的该配置。

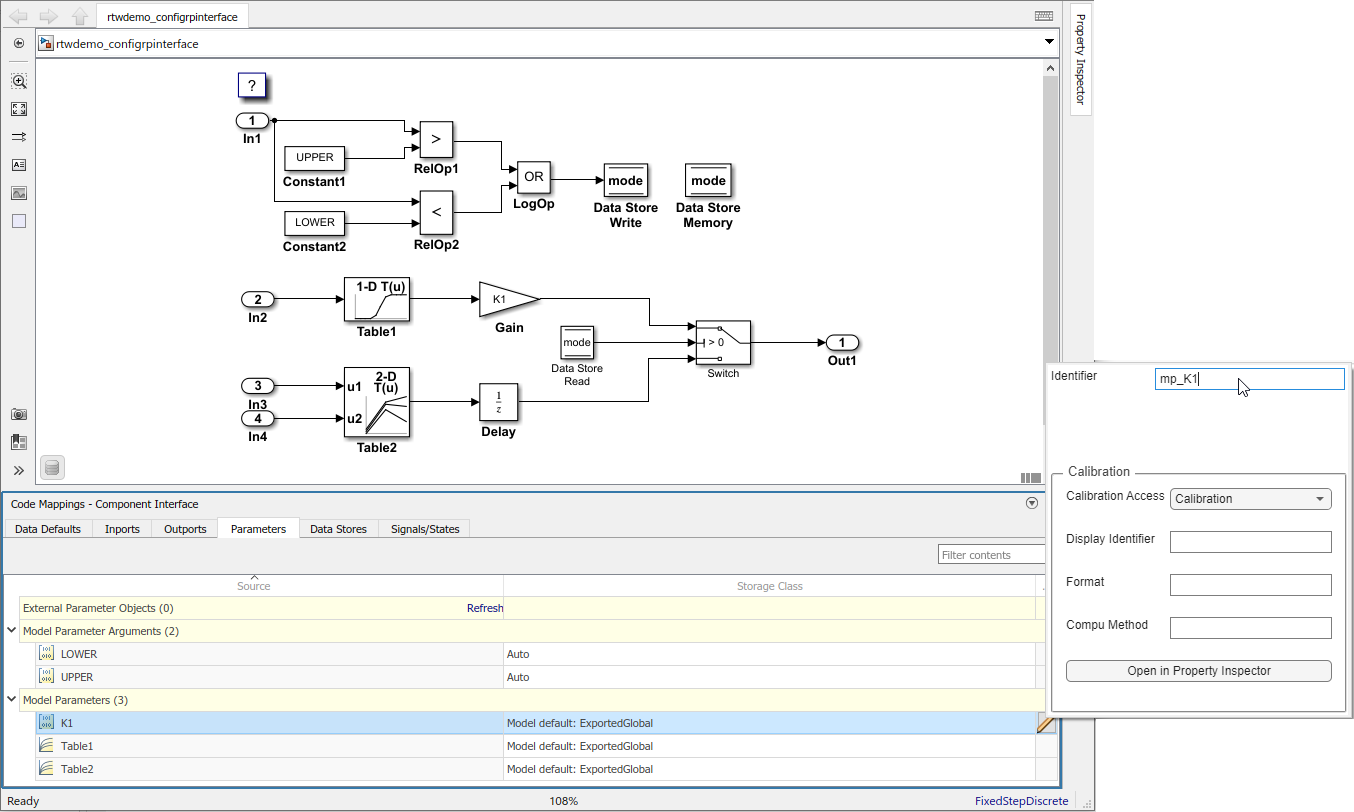
此示例说明如何使用 Code Mappings 编辑器将模型参数的默认存储类设置应用于模型 **rtwdemo\_configinterface** 中的参数 **K1**、**Table1** 和 **Table2**。为这些参数配置代码标识符。您可以指定代码生成标识符（例如用于集成），而无需修改模型设计。

1. 如果您尚未这样做，请完成“为参数配置默认代码生成设置” [（第 17-8](#_bookmark144) 页）中的步骤。
2. 展开 **Model Parameters**。默认情况下，每个参数的存储类都设置为 **Auto**。使用模型默认配置，该配置指定存储类 **ExportedGlobal**。
   * 要避免优化并强制代码生成器使用默认配置，请将存储类设置为 “**Model default**”。
   * 要覆盖默认配置，请指定满足该参数的代码生成要求的存储类。
3. 在 Code Mappings 编辑器中的 **Model Parameters** 下，选择参数 **K1**、**Table1** 和 **Table2**。将存储类设置为 “**Model default:ExportedGlobal**”。



1. 使用包含前缀 **mp\_** 的名称配置模型参数的代码标识符。在 Code Mappings 编辑器中，选择模型参

数 **K1**。点击  图标，并将存储类属性 **Identifier** 设置为 **mp\_K1**。对于参数 **Table1** 和 **Table2**，将 **Identifier** 设置为 **mp\_Table1** 和 **mp\_Table2**。



1. 保存模型。
2. 生成并查看代码。例如，在 **rtwdemo\_configrpinterface.c** 中查找模型参数 **mp\_K1** 的数据定义。

**int8\_T mp\_K1 = 2;**

查找参数在单步入口函数中的使用位置。

**if (mode) {**

**output = (real\_T)mp\_K1 \* dout\_Table1;**

**} else {**

**output = dstate\_X;**

**}**

### 以编程方式配置参数的代码生成设置

要自动化配置参数的代码生成，请使用代码映射的编程接口。例如，当创建自定义模块库或应用程序测试环境的一部分时，可使用编程接口来自动化数据配置。

此示例说明如何为模型 rtwdemo\_configrpinterface 的模型参数配置默认设置。配置模型参数，使其可调并在生成代码中定义和声明为单独全局变量。将模型参数的默认存储类设置应用于参数 **K1**、**Table1** 和 **Table2**。为这些参数配置代码标识符。

1. 打开示例模型。

**open\_system('rtwdemo\_configrpinterface')**

1. 将模型配置参数 **Default parameter behavior** 设置为 “**Tunable**”。

**model='rtwdemo\_configrpinterface';**

**set\_param(model,'DefaultParameterBehavior','Tunable');**

1. 通过调用函数 **coder.mapping.api.get** 创建对象 **cm**。该对象存储模型 **rtwdemo\_configrpdefaults** 中数据元素的代码生成配置。

**cm = coder.mapping.api.get('rtwdemo\_configrpinterface');**

1. 通过调用函数 **setDataDefault**，配置模型形参的默认设置。对于参数，请指定以下值：

#### coder.mapping.api.get 返回的对象

* + **ModelParameters** 作为默认类别
  + 属性名称 **StorageClass**，属性值 **ExportedGlobal**。

**setDataDefault(cm,'ModelParameters','StorageClass','ExportedGlobal');**

1. 验证模型参数的默认配置。发出对 **getDataDefault** 的调用，这些调用指定由

#### coder.mapping.api.get 返回的对象、类别 ModelParameters 和 StorageClass。

**getDataDefault(cm,'ModelParameters','StorageClass') ans =**

**'ExportedGlobal'**

1. 将模型参数的默认配置应用于参数 **K1**、**Table1** 和 **Table2**。

默认情况下，Simulink 将单个参数的存储类设置为 **Auto**。当存储类为 **Auto** 时，代码生成器将：

* + 确定是否出于优化目的从生成的代码中消除数据。
  + 如果保留数据，则确定如何在生成代码中高效地表示数据，同时考虑默认配置设置。

配置代码生成器，以将默认模型参数设置应用于参数 **K1**、**Table1** 和 **Table2**。对于每个参数，调用函数 **setModelParameter**。指定由 **coder.mapping.api.get** 返回的对象、参数名称、属性名称 **StorageClass** 和属性值 **Model default**。

**setModelParameter(cm,'K1','StorageClass','Model default'); setModelParameter(cm,'Table1','StorageClass','Model default'); setModelParameter(cm,'Table2','StorageClass','Model default');**

1. 通过调用函数 **getModelParameter**，验证您对参数 **K1**、**Table1** 和 **Table2** 的配置更改。

**getModelParameter(cm,'K1','StorageClass') ans =**

**'Model default' getModelParameter(cm,'Table1','StorageClass') ans =**

**'Model default'**

**getModelParameter(cm,'Table2','StorageClass') ans =**

**'Model default'**

1. 为模型参数配置代码标识符。对于每个参数，调用函数 **setModelParameter**。指定由 **coder.mapping.api.get** 返回的对象、参数名称、属性名称 **Identifier** 以及下列属性值之一。

|  |  |
| --- | --- |
| **模型参数** | **代码标识符** |
| **K1** | **mp\_K1** |
| **Table1** | **mp\_Table1** |
| **Table2** | **mp\_Table2** |

**setModelParameter(cm,'K1','Identifier','mp\_K1'); setModelParameter(cm,'Table1','Identifier','mp\_Table1'); setModelParameter(cm,'Table2','Identifier','mp\_Table2');**

1. 通过调用函数 **getModelParameter**，验证您对模型参数的配置更改。

**getModelParameter(cm,'K1','Identifier') ans =**

**'mp\_K1' getModelParameter(cm,'Table1','Identifier') ans =**

**'mp\_Table1'**

**getModelParameter(cm, 'Table2', 'Identifier') ans =**

**'mp\_Table2'**

1. 保存模型。
2. 生成并查看代码。例如，在 **rtwdemo\_configrpinterface.c** 中查找模型参数 **mp\_K1** 的数据定义。

**int8\_T mp\_K1 = 2;**

查找参数在单步入口函数中的使用位置。

**if (mode) {**

**output = (real\_T)mp\_K1 \* dout\_Table1;**

**} else {**

**output = dstate\_X;**

**}**

### 为模型参数选择存储类和存储类属性

根据您的代码生成要求，从这些存储类中进行选择，为模型参数配置代码生成。对于常量，仅 “**Default**” 适用。对于模型实参，仅 “**Auto**”、“**Default**” 和 “**Model default**” 适用。

|  |  |
| --- | --- |
| **要求** | **Storage Class** |
| 启用优化，以便能生成更高效的代码。 | “Auto”（仅限单个映射） |
| 对于无法优化的数据元素，将数据表示为标准数据结构体的一个字段。 | “Default”（仅限默认映射） |

|  |  |
| --- | --- |
| **要求** | **Storage Class** |
| 防止优化消除数据元素的存储，并使用数据元素类别的默认映射。 | “Model Default”（仅限单个映射）、  **Dictionary Default**（仅限单个映射） |
| 生成全局变量定义和声明。 | “ExportedGlobal” |
| 生成可读写在外部代码中定义的全局变量或全局变量指针的代码。 | “ImportedExtern, ImportedExternPointer” |

可用存储类的列表可能包括在 Embedded Coder 字典中定义的其他工程特定存储类。如果您有列出的存储类不能满足的特殊要求，并且您有 Embedded Coder 软件，您可以定义存储类。请参阅“Define Service Interfaces, Storage Classes, Memory Sections, and Function Templates for Software Architecture” (Embedded Coder)。

对于单个模型参数，使用 **Identifier** 存储类属性来配置在生成代码中表示参数的变量的名称。

### 另请参阅

**代码映射编辑器 - C** | **coder.mapping.api.CodeMapping**

### 详细信息

* [“模型接口元素的 C 代码生成配置” （第 17-2 页）](#_bookmark135)
* “Choose Storage Class for Controlling Data Representation in Generated Code”
* “Configure Model Data Elements for ASAP2 File Generation”

## 为模型入口函数配置生成的 C 函数接口

### 什么是入口函数？

入口是代码中发生程序控制权（执行）转移的位置。主函数 (**main()**) 是进入 C/C++ 程序的入口，在应用程序开始执行时调用。对其他函数的调用（例如从 **main** 函数调用）是进入函数代码的入口。程序控制权会转移到被调函数。函数代码随后执行，在完成后将控制权交还给 **main** 或其他主调函数。

为模型生成代码时，代码生成器会定义一组入口函数，您可以调用这些函数来执行生成的代码。您可以从外部代码或生成的主函数的修改版调用生成的函数。

代码生成报告的“代码接口报告”部分会列出代码生成器为模型生成的入口函数。有关详细信息，请参阅 “Analyze Generated Data Code Interface” (Embedded Coder)。

### 生成的入口函数的类型

代码生成器为下列类型的模型函数生成入口函数。

|  |  |  |  |
| --- | --- | --- | --- |
| **模型函数的类型** | **模型源函数名称** | **默认的生成函数名称** | **描述** |
| 导出函数（需要 Embedded Coder） | **ExportedFunction:slI dentifier**，其中 **slIdentifier** 是模型中函数调用 Inport 模块的名称 | **function-call-inport- block-name** 或 **signal- label**（如果指定） | 对于导出函数模型，即为子系统的导出函数。 |
| 初始化函数 | **Initialize** | **model\_initialize** | 模型的初始化代码。在应用程序代码的开始，调用函数一次。请不要使用此函数重置实时模型数据结构体 (**rtM**)。 |
| 分区函数 | **Partition:slIdentifier**  ，其中 **slIdentifier** 是从模型中的模块显式创建的一个分区并显示在 Simulink® 调度编辑器中  （例如 P1），或是“并行执行”对话框中的一个任务名称 | **model\_stepn**，其中 **n** 基于模型采样周期唯一标识生成的函数 | 对于模型分区，输出并更新代码。模型配置参数**单一输出/更新函数**处于选中状态（默认值）。 |
| 分区更新函数 | **PartitionUpdate:slIde ntifier**，其中 **slIdentifier** 是从模型中的模块显式创建的一个分区并显示在 Simulink® 调度编辑器中（例如 P1），或是“并行执行”对话框中的一个任务名称 | **model\_outputn** 和 **model\_updaten**，其中 **n** 基于模型采样周期唯一标识生成的函数 | 对于模型分区，使用分别的函数输出和更新代码。模型配置参数**单一输出/更新函数**处于未选中状 态。 |

|  |  |  |  |
| --- | --- | --- | --- |
| **模型函数的类型** | **模型源函数名称** | **默认的生成函数名称** | **描述** |
| 周期性多任务函数 | **Periodic:slIdentifier**，其中 **slIdentifier** 是一个注释，对应于多任务模型的一个周期性或连续速率的采样时间周期（例如， D1） | **model\_stepn**，其中 **n** 基于模型采样周期唯一标识生成的函数 | 对于为多任务配置的一个基于速率的模型中的模 块，输出和更新代码。代码生成器为每个采样周期生成一个函数。模型配置参数**单一输出/更新函数**处于选中状态（默认  值）。 |
| 周期性多任务更新函数 | **PeriodicUpdate:slIden tifier**，其中 **slIdentifier** 是一个注  释，对应于多任务模型的一个周期性或连续速率的采样时间周期（例如， D1） | **model\_outputn** 和 **model\_updaten**，其中 **n** 基于模型采样周期唯一标识生成的函数 | 对于为多任务配置的一个基于速率的模型中的模 块，分别在单独的函数中输出和更新代码。代码生成器为每个采样周期生成输出和更新函数。模型配置参数**单一输出/更新函数**处于未选中状态。 |
| 周期性单任务函数 | **Periodic** | **model\_step** | 对于为单任务配置的基于速率的模型中的模块，输出和更新代码。模型配置参数**单一输出/更新函数**处于选中状态（默认  值）。 |
| 周期性单任务更新函数 | **PeriodicUpdate** | **model\_output** 和  **model\_update** | 对于为单任务配置的基于速率的模型中的模块，分别在单独的函数中输出和更新代码。模型配置参数**单一输出/更新函数**处于未选中状态。 |
| 重置函数 | **Reset:slIdentifier**，其中 **slIdentifier** 是模型中重置函数的名称 | **model\_reset-function- name** | 如果模型包含 Reset Function 模块，则重置生成的代码。要重置条件或状态，请从应用程序代码中调用函数。 |
| Simulink 函数 | **Simulink Function:slIdentifier**  ，其中 **slIdentifier** 是模型中 Simulink Function模块的名称 | 对于全局 Simulink Function模块，为 **function-name**；对于限定作用域的 Simulink Function 模块，为  **model\_function-name** | 对于导出函数模型，是 Simulink Function 模块的导出函数。 |
| 终止函数 | **Terminate** | **model\_terminate** | 用于关闭系统的代码。对于基于 ERT 的模型，您可以通过清除模型配置参数**需要终止函数** (Embedded Coder)（默认设置）来隐藏此函数的生成。 |

如果您有 Embedded Coder 软件，为了便于集成外部代码和生成的代码并符合代码标准和规范，您可以配置代码生成器如何从模型或子系统生成函数接口。

### 配置入口函数是否可重用

默认情况下，对于顶层模型，代码生成器生成不可重用或不可重入的代码。入口函数有一个 **void-void** 接口。代码通过直接访问驻留在共享内存中的全局数据结构体来与其他代码通信。

如果您的应用程序需要可重用的多实例入口函数代码，您可以配置代码生成器使用唯一数据来调用每个函数（实例）。在这种情况下，代码是可重入的。

您可以使用模型配置参数**代码接口打包**和相关参数来配置入口函数是否可重用。您选择的参数设置取决于系统目标文件的配置选择、编程语言和参数接口等因素。

#### 单实例 C 入口函数的默认配置

默认情况下，对于基于 GRT 和基于 ERT 的系统目标文件，代码生成器生成单实例 C 入口函数。生成的代码：

* 创建不带参数的执行函数 (**void-void**)。
* 为模型数据结构体静态分配内存（在编译时）。

配置单实例入口函数代码的默认模型配置参数设置为：

* **语言**设置为 “**C**”。
* **代码接口打包**设置为“不可重用函数”。

#### 生成可重用的多实例 C 入口函数

您可以配置代码生成器，为基于 GRT 或基于 ERT 的系统目标文件生成可重用的 C 入口函数。但是，代码生成器默认生成的各函数接口有所不同。假设模型配置参数**语言**设置为 “**C**” 且**代码接口打包**设置为“可重用函数”，代码生成器为每个系统目标文件场景生成以下入口函数代码。

|  |  |
| --- | --- |
| **系统目标文件** | **接口** |
| 基于 GRT | * 可重用、可重入的多实例 C 入口函数。 * 将模型根级 Inport 模块和 Outport 模块的值打包到实时模型数据结构体中。按引用将该结构体作为参数传递给执行函数。 * 在运行时为模型实例的数据动态分配内存。通过调用函数（如 **malloc**）来分配内存。 |
| 基于 ERT | * 可重用、可重入的多实例 C 入口函数。 * 将每个模型根级 Inport 模块和 Outport 模块的值作为单独的参数传递给执行函数。 * 为模型数据结构体静态分配内存。 |

如果您使用的是基于 ERT 的系统目标文件，并希望生成可重用、可重入的多实例 C 入口函数，请考虑：

* 使用动态内存分配来初始化模型数据结构体。选择**使用动态内存分配进行模型初始化** (Embedded Coder)。
* 将模型根级 Inport 模块的值打包到一个结构体中，将根级 Outport 模块的值打包到另一个结构体中，并按引用将这些结构体作为参数传递给执行函数。将**根级 I/O 传递方式** (Embedded Coder)设置为

“结构体引用”。

* 将模型根级 Inport 模块和 Outport 模块的值打包到实时模型数据结构体中，并按引用将该结构体作为参数传递给执行函数。将**根级 I/O 传递方式** (Embedded Coder)设置为“模型数据结构体的一部

分”。

### 如何与生成的入口函数对接

1. 为模型生成代码后，使用**代码**视图查看生成的入口函数，以及表示外部输入和输出端口的变量（如果适用）。
2. 将 **#include** 语句添加到外部代码中，这些语句包括声明模型入口函数的生成的头文件。
3. 添加一条 **#include** 语句以包含生成的文件 **rtwtypes.h**。此文件提供类型定义、**#define** 语句和枚举。
4. 初始化目标特定的数据结构体和硬件，如 ADC 或 DAC。
5. 如果适用，为可重用模型的每个实例初始化数据。
6. 如果适用，将输入数据写入表示模型 Inport 模块的生成变量。
7. 调用生成的入口函数或设置 **rt\_OneStep** 函数的使用。
8. 如果适用，从表示模型 Outport 模块的生成变量中读取数据。

有关详细信息，请参阅“Deploy Applications to Target Hardware” (Embedded Coder)。

### C 函数接口自定义限制

生成的 C 函数接口的自定义有以下限制：

* 您必须选择模型配置参数**单一输出/更新函数**。
* 支持多速率模型，但您必须为单任务配置模型。
* 您必须配置根级输入端口和输出端口以使用 **Default** 存储类。
* 如果您选择自定义函数接口，您必须提供自己的自定义 **main** 程序。您无法使用 MathWorks 提供的静态 **rt\_main.c** 来配置函数接口。指定非默认的函数接口配置会导致生成的代码与默认静态 **rt\_main.c** 不匹配。
* 代码生成器会删除模型根输入端口的数据结构体，除非由不可重用函数实现的子系统使用一个或更多个输入端口的值。
* 代码生成器会删除模型的根输出端口的数据结构体，除非您启用 MAT 文件记录或者一个或更多个输出端口的采样时间不是基本速率（包括恒定速率）。
* 如果您复制子系统模块以在新模型或同一模型中创建模块，则原始子系统模块的函数接口信息不会复制到新子系统模块中。
* 如果您有 Stateflow，则对于使用模型根输入端口值或调用使用模型根输入端口值的子系统的 Stateflow 图，请执行以下操作之一来生成代码：
  + 在 Stateflow 图中，清除**初始化时执行(进入)图**复选框。
  + 将 Stateflow 函数设置为不可重用的函数。
  + 在根输入端口后立即插入 Simulink Signal Conversion 模块。在 Signal Conversion 模块参数对话框中，选择**从 ''模块简化'' 优化中排除此模块**。
* 如果模型根输入端口值连接到 Simscape™ 转换模块，请在根输入端口和 Simscape 转换模块之间插入 Simulink Signal Conversion 模块。在 Signal Conversion 模块参数对话框中，选择**从 ''模块简化'' 优化中排除此模块**。
* 当编译配置有函数接口的引用模型时，不要使用虚拟总线作为引用模型的输入或输出。请改用非虚拟总线。
* 如果 C 函数接口不是默认值，则忽略模型配置参数**为代码生成按值传递固定大小的标量根输入**的值。有关详细信息，请参阅 Pass fixed-size scalar root inputs by value for code generation。

### 另请参阅

Simulink Function | Function Caller

### 详细信息

* [“模型接口元素的 C 代码生成配置” （第 17-2 页）](#_bookmark135)
* “Simulink Function Blocks and Code Generation”

**Simulink Coder 中的入口函数和调度**

# 数组布局

# 在生成的代码中配置数据

# 代码生成

# Simulink Coder 的配置

## 以编程方式配置模型的代码生成参数

您可以在“配置参数”对话框中或从 MATLAB 命令行修改活动配置集的代码生成参数。使用命令行方法创建一个脚本，为已建立的模型配置自动设置参数。

### 修改参数以支持“执行效率”

在此示例中，您将修改配置参数以支持代码生成顾问应用程序目标“执行效率”。

#### 步骤 1.打开一个模型。

**openExample('slexAircraftExample')**

#### 步骤 2.获取活动配置集。

**cs = getActiveConfigSet(model);**

#### 步骤 3.选择一般实时 (GRT) 目标。

**switchTarget(cs,'grt.tlc',[]);**

#### 步骤 4.要优化执行速度，请修改参数。

如果您的应用程序目标是“执行效率”，请使用 **set\_param** 修改以下参数：

**set\_param(cs,'MatFileLogging','off'); set\_param(cs,'SupportNonFinite','off'); set\_param(cs,'RTWCompilerOptimization','on'); set\_param(cs,'OptimizeBlockIOStorage','on'); set\_param(cs,'EnhancedBackFolding','on'); set\_param(cs,'ConditionallyExecuteInputs','on') set\_param(cs,'DefaultParameterBehavior','Inlined'); set\_param(cs,'BooleanDataType','on'); set\_param(cs,'BlockReduction','on'); set\_param(cs,'ExpressionFolding','on'); set\_param(cs,'LocalBlockOutputs','on'); set\_param(cs,'EfficientFloat2IntCast','on'); set\_param(cs,'BufferReuse','on');**

#### 步骤 5.将模型配置保存到文件中。

将模型配置保存到 **'Exec\_efficiency\_cs.m'** 文件，并查看参数设置。

**saveAs(cs,'Exec\_Efficiency\_cs'); dbtype Exec\_Efficiency\_cs 1:50**

### 另请参阅详细信息

* “Code Generation Configuration”
* “Application Objectives Using Code Generation Advisor”

# 系统目标文件配置

* [“配置系统目标文件” （第 22-2 页）](#_bookmark161)
* [“比较各产品的系统目标文件支持” （第 22-5 页）](#_bookmark166)

## 配置系统目标文件

要对模型进行配置以用于代码生成，请按照“选择支持代码生成的求解器” [（第 22-2](#_bookmark162) 页）和“从 [STF浏览器中选择系统目标文件” （第 22-2](#_bookmark163) 页）中的步骤进行操作。选择系统目标文件时，其他模型配置参数会相应更改，以满足执行环境的要求。例如：

* 代码接口参数
* 编译过程参数，例如工具链或模板联编文件
* 目标硬件参数，例如字长和字节顺序

选择系统目标文件后，您可以修改模型配置参数设置。

您可以在单个工作流中切换系统目标文件，以实现不同的代码生成目的（例如，快速原型构建与生产代码部署）。要进行切换，请为同一模型设置不同配置集，并切换当前操作的活动配置集。有关如何设置配置集和更改活动配置集的详细信息，请参阅“管理模型的配置集”。

### 选择支持代码生成的求解器

要编译模型，模型配置必须指定与系统目标文件的代码生成兼容的求解器。对于使用变步长求解器的模型或具有非零开始时间的模型，很少有系统目标文件支持为其生成代码。

* 对于 GRT、ERT 和基于 ERT 的系统目标文件，将**类型**配置参数设置为“定步长”。
* 对于快速仿真 (Rsim) 或 S-Function (rtwsfcn) 系统目标文件，将**类型**配置参数设置为“定步长”或 “变步长”。

有关使用定步长求解器为实时系统目标文件生成代码的要求的详细信息，请参阅“基于时间的调度和代码生成” (Embedded Coder)。

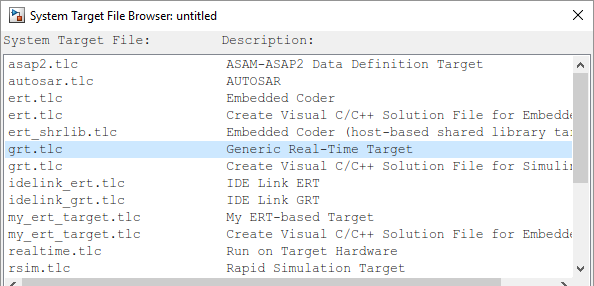
当**求解器选择**配置参数设置为“使用 **Simulink** 求解器模块”时，快速仿真 (RSim) 系统目标文件支持非零开始时间。其他系统目标文件不支持非零开始时间 - 代码生成器不生成代码且编译过程产生错误。

### 从 STF 浏览器中选择系统目标文件

选择求解器（请参阅“选择支持代码生成的求解器” [（第 22-2](#_bookmark162) 页））后，为**系统目标文件**配置参数指定值。直接在字段中输入系统目标文件的名称，或使用系统目标文件浏览器选择文件。

要使用系统目标文件浏览器，请执行以下操作：

1. 在配置参数对话框中，在**系统目标文件**字段旁边，点击**浏览**按钮。该浏览器显示可用的系统目标文件列表，包括自定义项。
2. 从列表中选择您的文件，例如，GRT 系统目标文件 (**grt.tlc**)。列表框的背景变为黄色，表示该选项未应用。点击**应用**或**确定**。



#### 系统目标文件浏览器

您还可以从 MATLAB [代码以编程方式选择系统目标文件，如“以编程方式选择系统目标文件” （第 22-](#_bookmark164) [3](#_bookmark164) 页）中所述。

选择系统目标文件后，您可以修改模型配置参数设置。为模型选择系统目标文件会选择工具链方法或模板联编文件方法进行编译过程控制。有关这些方法的详细信息，请参阅“配置工具链 [(ToolchainInfo) 或模板联编文件编译过程” （第 31-3](#_bookmark265) 页）。

如果要在单个工作流中切换系统目标文件以用于不同的代码生成目的，请为同一模型设置不同配置集。切换当前操作的活动配置集。此方法对于在快速原型构建和生产代码部署之间切换非常有用。有关如何设置配置集和更改活动配置集的详细信息，请参阅“管理模型的配置集”。

### 以编程方式选择系统目标文件

Simulink 模型在配置集中存储模型范围的参数和系统目标文件特定的数据。每个配置集都包含一个组件，用于定义特定系统目标文件的结构以及相关选项的当前值。Simulink 从您指定的系统目标文件中加载部分此类信息。您可以通过复制和修改旧配置集或添加新配置集并浏览以选择新系统目标文件来配置模型以生成备选代码。然后，您可以通过交互方式从这些配置集中选择活动配置（在一个给定时间只能有一个配置集处于活动状态）。

自动执行系统目标文件选择的脚本必须模拟此过程。要对系统目标文件选择进行编程，请执行下列操作：

1. 通过调用 **getActiveConfigSet** 函数获取活动配置集的句柄。
2. 定义对应于所需系统目标文件、工具链或模板联编文件和/或 **make** 命令设置的字符向量变量。例

如，对于 ERT 系统目标文件，您将为字符向量 **'ert.tlc'**、**'ert\_default\_tmf'** 和 **'make\_rtw'** 定义变量。

1. 通过调用 **switchTarget** 函数选择系统目标文件。在函数调用中，指定活动配置集的句柄和系统目标文件。
2. 将 **TemplateMakefile** 和 **MakeCommand** 配置参数设置为在步骤 2 中创建的对应变量。例如：

**cs = getActiveConfigSet(model); stf = 'ert.tlc';**

**tmf = 'ert\_default\_tmf'; mc = 'make\_rtw';**

**switchTarget(cs,stf,[]); set\_param(cs,'TemplateMakefile',tmf); set\_param(cs,'MakeCommand',mc);**

有关以编程方式选择系统目标文件的详细信息，请参阅 **switchTarget**。

### 开发自定义系统目标文件

您可以创建自己的系统目标文件，以对接外部代码或操作环境。

[有关如何使自定义系统目标文件出现在系统目标文件浏览器中并显示相关控制项的详细信息，请参阅“自定义目标” （第 46-2](#_bookmark413) 页）及其引用的主题。

### 另请参阅

**getActiveConfigSet** | **switchTarget**

### 详细信息

* “Configure STF-Related Code Generation Parameters”
* “Choose a Code Replacement Library”
* “Configure Standard Math Library for Target System”
* [“比较各产品的系统目标文件支持” （第 22-5 页）](#_bookmark166)
* [“自定义目标” （第 46-2 页）](#_bookmark413)

## 比较各产品的系统目标文件支持

在您选择系统目标文件（例如 **grt.tlc**）时，所做选择将定义运行时环境和代码生成功能。确定符合您的代码生成工作流目标的系统目标文件功能。

代码生成器使用系统目标文件来生成用于在特定目标硬件或特定操作系统上执行的代码。系统目标文件调[用其他运行时环境特定的文件。有关为目标硬件配置模型代码生成参数的详细信息，请参阅“配置运行时环境选项” （第 10-2](#_bookmark43) 页）。

不同类型的系统目标文件支持选择相应的生成代码的功能。在系统目标文件中，**CodeFormat** TLC 变量的值和对应的 **rtwgensettings.DerivedFrom** 字段值标识系统目标文件类型和生成代码的功能。这些选择会在代码生成过程中的几个点应用代码生成控制决策。您的选择包括模型编译是否以及如何生成：

* 特定数据结构体（例如，**SimStruct** 或 **rtModel**）
* 静态或动态内存分配代码
* 为生成的模型函数调用接口

对于自定义系统目标文件开发，**CodeFormat** 值在不同的代码生成目标中会有所不同：

* 如果系统目标文件不包含 **CodeFormat** TLC 变量的值，则对于一般实时目标 (GRT)，默认值为

**RealTime**。对应的 **rtwgensettings.DerivedFrom** 字段值为 **grt.tlc**（默认值）。

* 如果您正在开发自定义系统目标文件并且有 Embedded Coder 软件，则对于嵌入式实时目标 (ERT)，请考虑将 **CodeFormat** TLC 变量值设置为 **Embedded-C**。对应的 **rtwgensettings.DerivedFrom** 字段值为 **ert.tlc**。与 GRT 系统目标文件相比，ERT 系统目标文件支持更多的生成代码功能。

此示例说明如何在 **ert.tlc** 中设置 **CodeFormat** TLC 变量的值以及对应的

#### rtwgensettings.DerivedFrom 字段值。

**%assign CodeFormat = "Embedded-C"**

**/%**

**BEGIN\_RTW\_OPTIONS**

**rtwgensettings.DerivedFrom = 'ert.tlc'; END\_RTW\_OPTIONS**

**%/**

**注意** 使用 **CodeFormat** TLC 变量的值及其对应的 **rtwgensettings.DerivedFrom** 字段值来为模型生成代码。如果未显式选择值，将应用默认值。对于 **rtwgensettings.DerivedFrom** 字段，请使用 **'ert.tlc'** 或 **'grt.tlc'**。有关详细信息，请参阅“系统目标文件的结构[” （第 46-4 页）](#_bookmark417)。

有关针对 GRT 和 ERT [系统目标文件默认生成的优化调用接口的说明，请参阅“为模型入口函数配置生成的 C 函数接口” （第 17-15](#_bookmark148) 页）。

使用 GRT 和 ERT 系统目标文件的代码生成采用实时模型数据结构体 (**rtModel**)。此结构体封装模型特定信息的形式比仿真结构体 **SimStruct** 的封装形式紧凑得多。所生成代码的许多高效功能取决于 **rtModel**而非 **SimStruct** 的生成，包括：

* 整数绝对时间和已用时间计时服务
* 异步任务的独立计时器
* 为信号、状态和参数监测生成改进的 C API 代码
* 删减数据结构体以最小化其大小（仅限 ERT 派生的系统目标文件）

有关 **rtModel** 数据结构体的说明，请参阅“Real-Time Model Data Structure”。

### 比较产品系统目标文件

您可以使用系统目标文件浏览器从一系列系统目标文件中进行选择。通过此选择，您可以使用配置选项进行试验并保存具有不同配置的模型。

除非您的系统上有所需的软件，否则不能为非 GRT 系统目标文件编译或生成代码。例如，对于 ERT 系统目标文件，需要 Embedded Coder；对于 SLDRT 系统目标文件，需要 Simulink Desktop Real-Time

™，等等。

为模型选择系统目标文件时，会选择工具链方法或模板联编文件方法进行编译过程控制。有关这些方法的详细信息，请参阅“编译从 [Simulink 模型生成的代码的方法” （第 31-2](#_bookmark264) 页）。

#### 可从系统目标文件浏览器获得的系统目标文件

|  |  |  |
| --- | --- | --- |
| **支持的系统目标文件** | **文件名** | **参考** |
| Embedded Coder（适用于 PC 或 UNIX 平台） | **ert.tlc ert\_shrlib.tlc** | “配置系统目标文件” (Embedded Coder) |
| 为 Embedded Coder 创建 Visual C  ++® 解决方案文件 | **ert.tlc**  （需要 CMake 工具链或 **RTW.MSVCBuild** 模板联编文件。请参阅注释。） | “配置系统目标文件” (Embedded Coder) |
| 适用于 AUTOSAR 的 Embedded Coder | **autosar.tlc** | “开发符合 AUTOSAR 标准的模型” (Embedded Coder) |
| 一般实时（适用于 PC 或 UNIX 平台） | **grt.tlc** | [“按 STF 比较生成的代码的功能” （第 22-](#_bookmark170) [10 页）](#_bookmark170) |
| 创建 Visual C++ 解决方案文件 | **grt.tlc**  （需要 CMake 工具链或 **RTW.MSVCBuild** 模板联编文件。请参阅注释。） | [“按 STF 比较生成的代码的功能” （第 22-](#_bookmark170) [10 页）](#_bookmark170) |
| 快速仿真（默认适用于 PC 或 UNIX平台） | **rsim.tlc** | “Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File” |
| 适用于 LCC 编译器的快速仿真 | **rsim.tlc** | “Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File” |
| 适用于 UNIX 平台的快速仿真 | **rsim.tlc** | “Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File” |
| 适用于 Visual C++ 编译器的快速仿真 | **rsim.tlc** | “Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File” |
| 适用于 PC 或 UNIX 平台的 S- Function | **rtwsfcn.tlc** | [“将 S-Function 目标用于模型或子系统”](#_bookmark210)  [（第 28-2 页）](#_bookmark210) |
| 适用于 LCC 的 S-Function | **rtwsfcn.tlc** | [“将 S-Function 目标用于模型或子系统”](#_bookmark210)  [（第 28-2 页）](#_bookmark210) |
| 适用于 UNIX 平台的 S-Function | **rtwsfcn.tlc** | [“将 S-Function 目标用于模型或子系统”](#_bookmark210)  [（第 28-2 页）](#_bookmark210) |
| 适用于 Visual C++ 编译器的 S- Function | **rtwsfcn.tlc** | [“将 S-Function 目标用于模型或子系统”](#_bookmark210)  [（第 28-2 页）](#_bookmark210) |
| ASAM-ASAP2 数据定义 | **asap2.tlc**  （在以后的版本中将会删除。有关如何生成 A2L 文件的信息，请参阅“生成 [ASAP2 和 CDF 标定文件” （第 36-2](#_bookmark381) 页）） | [“生成 ASAP2 和 CDF 标定文件” （第 36-](#_bookmark381) [2 页）](#_bookmark381) |

|  |  |  |
| --- | --- | --- |
| **支持的系统目标文件** | **文件名** | **参考** |
| Simulink Desktop Real-Time | **sldrt.tlc sldrtert.tlc** | “Set Run in Kernel Mode Code Generation Parameters” (Simulink Desktop Real-Time) |
| Simulink Real-Time™ | **slrealtime.tlc** | “Simulink Real-Time Options Pane” (Simulink Real-Time) |

**注意** 要创建和编译具有调试配置的 Visual C++ 解决方案 (**.sln**) 文件，请使用以下选项之一：

* 在 Toolchain 字段中，指定 CMake 工具链，例如，“**Microsoft Visual Studio Project 2019 | CMake (64** 位 **Windows)**” 或 “**Microsoft Visual Studio Project 2017 | CMake (64** 位 **Windows)**”。有关 CMake 工具链的详细信息，请参阅“Configure CMake Build Process”。
* 在 Template makefile 字段中，指定 “**RTW.MSVCBuild**”。

### 比较代码样式和 STF 支持

代码生成器生成两种样式的代码。一种代码样式适用于快速原型构建（以及使用代码生成进行仿真）。另一种样式适用于嵌入式应用程序。下表将系统目标文件映射到对应的代码样式。

#### 系统目标文件列出的代码样式

|  |  |  |
| --- | --- | --- |
| **系统目标文件** | **代码样式** | **用途** |
| Embedded Coder 嵌入式实时 (ERT) | 嵌入式 | 以此为起点开发 C/C++ 生成代码的嵌入式应用程序。 |
| Simulink Coder 一般实时 (GRT) | 快速原型 | 以此为起点，创建不使用实时操作系统任务原语的快速原型目标硬件，以及在台式计算机上验证生成的 C/C++ 代码。 |
| 快速仿真 (RSim) | 快速原型 | 在台式计算机和高速或批量仿真工具上提供非实时仿真。 |
| S-Function | 快速原型 | 创建 C MEX S-Function，用于在另一个 Simulink模型中进行仿真。 |
| Simulink Desktop Real-Time | 快速原型 | 当台式计算机在后台运行 Microsoft Windows 时，在中断级别实时运行模型。 |
| Simulink Real-Time | 快速原型 | 在运行 Simulink Real-Time 内核的台式计算机上实时运行模型。 |

第三方供应商提供其他系统目标文件以支持其产品的代码生成。有关第三方产品的详细信息，请参阅供应商网站或 MathWorks Connections 计划网页：[**https://www.mathworks.com/products/**](https://www.mathworks.com/products/connections.html)[**connections**](https://www.mathworks.com/products/connections.html)。

### 按产品比较生成的代码的功能

实时系统目标文件（例如 GRT）的代码生成过程提供许多嵌入式代码优化。选择基于 ERT 的系统目标文件提供比 GRT 更广泛的功能。系统目标文件选择确定代码生成产品的可用功能。当您选择与开发过程匹配的代码生成目标时，请使用下表来比较随 Simulink Coder 提供的代码生成功能和随 Embedded Coder 提供的功能。

**比较 Simulink Coder 与 Embedded Coder 的代码生成功能**

|  |  |  |
| --- | --- | --- |
| **功能** | **Simulink Coder** | **Embedded Coder** |
| **rtModel** 数据结构体 | * 生成完整的 **rtModel** 结构体 * GRT 变量声明：**rtModel\_model model\_M\_;** | * 针对模型对 **rtModel** 进行优化 * 可选择隐藏错误状态字段和数据记录字段 * ERT 变量声明：**RT\_MODEL\_model model\_M\_;** |
| 自定义存储类 (CSC) | 代码生成会忽略 CSC。为对象分配一个 CSC（默认为 **Auto** 存储类）。 | 支持使用 CSC 生成代码 |
| HTML 代码生成报告 | 基本 HTML 代码生成报告 | 包含更多详细信息和模型超链接的增强报告 |
| 符号格式设置 | 根据硬编码默认值生成符号（用于信号、参数等） | 对生成的符号施加详细控制 |
| 用户定义的针对生成符号的最大标识符长度 | 支持 | 支持 |
| 生成终止函数 | 生成 | 可选择是否隐藏终止函数 |
| 组合的输出/更新函数 | 生成单独的输出/更新函数 | 可选择是否生成组合的输出/更新函数 |
| 优化的数据初始化 | 不适用 | 可选择是否为零值内存、I/O 端口等生成不必要的初始化代码 |
| 注释生成 | 可选择包含或禁止注释生成 | 可选择是否在注释中包含 Simulink 模块描述、Stateflow 对象描述和 Simulink 数据对象描述 |
| 模块打包功能 (MPF) | 不支持 | 广泛的代码自定义功能。请参阅“Manage Replacement of Simulink Data Types in Generated Code” (Embedded Coder)和 “MPT Data Object Properties” (Embedded Coder)。 |
| 针对系统目标文件优化的数据类型头文件 | 需要完整的 **tmwtypes.h** 头文件 | 生成优化的 **rtwtypes.h** 头文件，包括系统目标文件所需的定义 |
| 用户定义的类型 | 用户定义的类型默认为代码生成中的基类型 | 代码生成支持用户定义的数据类型别名 |
| 速率分组 | 不支持 | 支持 |
| 主程序模块自动生成 | 不支持。提供静态主程序模块。 | 支持自动和可自定义的主程序模块生成（也可使用静态主程序） |
| 可重用（多实例）代码生成 | 可选择是否使用动态内存分配生成可重用代码 | 可选择是否使用静态或动态内存分配生成可重用代码 |
| 软件约束选项 | 支持浮点数、复数和非有限数 | 可选择启用或禁用浮点数、复数和非有限数支持 |
| 应用程序寿命 | 默认为 **inf** | 用户指定。确定整数计时器的最有效字长 |
| ANSI -C/C++ 代码生成 | 支持 | 支持 |
| ISO® -C/C++ 代码生成 | 支持 | 支持 |
| GNU® -C/C++ 代码生成 | 支持 | 支持 |
| 生成标量内联参数作为  **#DEFINE** 语句 | 不支持 | 支持 |

|  |  |  |
| --- | --- | --- |
| **功能** | **Simulink Coder** | **Embedded Coder** |
| MAT 文件变量名称修饰符 | 支持 | 支持 |
| 数据交换：C API、 ASAP2、外部模式 | 支持 | 支持 |

### 按 STF 比较生成的代码的功能

代码生成器支持不同系统目标文件的生成代码的一系列功能。在每个系统目标文件中，**CodeFormat** TLC变量的值标识功能集。

下表总结了不同系统目标文件对应用程序的支持。

|  |  |
| --- | --- |
| **应用场景** | **系统目标文件 (STF)** |
| 固定或变步长加速 | RSIM、S-Function、模型引用 |
| 定步长实时部署 | GRT、ERT、 Simulink Real-Time、Simulink Desktop Real-Time... |

下表总结了对每个**系统目标文件**选项可用的各种选项，也说明了例外情况。

#### 为系统目标文件 (STF) 生成的代码所支持的功能

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **系统目标文件 (STF)** | | | | | | | |
| **功能** | **grt.tlc**  **（请参阅注释 1。）** | **ert.tlc**  **（请参阅注释 1。）** | **ert\_shrlib.tlc**  **（请参阅注释 1。）** | **rtwsfcn.tlc**  **（请参阅注释 1。）** | **rsim.tlc**  **（请参阅注释 1。）** | **sldrt.tlc**  **（请参阅注释 1。）** | **slrealtim e.tlc**  **（请参阅注释 1。）** | **其他**  **（请参阅注释 1。）** |
| 静态内存分配 | X | X |  |  |  | X | X | X |
| 动态内存分配 | X  （请参阅注释 4、  5。） | X  （请参阅注释 4、  5。） |  | X | X |  | X |  |
| 连续时间 | X | X |  | X | X | X | X |  |
| C/C++ MEX  S-Function  （非内联） | X | X |  | X | X | X | X |  |
| S-Function  （内联） | X | X |  | X | X | X | X | X |
| 最小化 RAM/ROM 使  用量 |  | X |  |  |  |  | X2 | X |
| 支持外部模式 | X | X |  |  | X | X | X |  |
| 快速原型 | X |  |  |  |  | X | X | X |
| 生产代码 |  | X |  |  |  |  | X2 | X  （请参阅注释  3。） |
| 批量参数调整和蒙特卡罗方法 |  |  | X |  | X |  |  |  |
| 系统级仿真器 |  |  | X |  |  |  |  |  |
| 硬实时执行 | X  （请参阅注释 3。） | X  （请参阅注释 3。） |  |  |  | X | X | X5 |
| 包括非实时可执行文件 | X | X |  |  | X |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **系统目标文件 (STF)** | | | | | | | |
| **功能** | **grt.tlc**  **（请参阅注释 1。）** | **ert.tlc**  **（请参阅注释 1。）** | **ert\_shrlib.tlc**  **（请参阅注释 1。）** | **rtwsfcn.tlc**  **（请参阅注释 1。）** | **rsim.tlc**  **（请参阅注释 1。）** | **sldrt.tlc**  **（请参阅注释 1。）** | **slrealtim e.tlc**  **（请参阅注释 1。）** | **其他**  **（请参阅注释 1。）** |
| 模型的多个实例 | X  （请参阅注释 4、  5。） | X  （请参阅注释 4、  5。） |  | X4 |  |  | X  （请参阅注释 4、 5。） | X  （请参阅注释 4、 5。） |
| 支持变步长求解器 |  |  |  | X | X |  |  |  |
| 支持 SIL/PIL | X | X |  |  |  |  |  | X |

**注释**

1. 系统目标文件：
   * **grt.tlc** - 一般实时目标
   * **ert.tlc** - 嵌入式实时目标
   * **ert\_shrlib.tlc** - 嵌入式实时目标共享库
   * **rtwsfcn.tlc** - S-Function
   * **rsim.tlc** - 快速仿真
   * **sldrt.tlc** - Simulink Desktop Real-Time
   * **slrealtime.tlc** - Simulink Real-Time
   * **其他** - Simulink Coder 中的嵌入式实时功能支持其他系统目标文件
2. 不适用于基于 GRT 的系统目标文件。仅适用于基于 ERT 的系统目标文件。
3. 默认的 GRT 和 ERT **rt\_main** 文件模拟硬实时执行。当显式连接到实时时钟时，它们会硬实时执行。
4. 您可以为 Stateflow 图的多个实例或包含图的子系统的多个实例生成代码，但当该图包含导出的图形函数或 Stateflow 模型包含机器级别事件时例外。
5. 在“配置参数”对话框中，对于**代码接口打包**字段，选择“可重用函数”。

### 另请参阅详细信息

* [“配置系统目标文件” （第 22-2 页）](#_bookmark161)
* “Configure STF-Related Code Generation Parameters”
* “Choose a Code Replacement Library”
* “Configure Standard Math Library for Target System”

# Simulink Coder 中的国际化支持

## 国际化和代码生成

软件开发工具中的国际化支持对于实现高效全球化至关重要。如果将来可能跨区域设置与他人协作，从工程开始就应考虑国际化。支持国际化可以避免后期修改或不得不开发新模型设计。相关要求涉及到区域设置。

### 区域设置

在计算机上，区域设置定义用户界面的语言（字符集编码）以及时间、日期和货币等信息的显示格式。编码规定区域可以显示的字符数。例如，US-ASCII 编码字符集（代码集）定义 128 个字符。Unicode® 代码集，如 UTF-8，定义超过 1100000 个字符。

对于代码生成，区域设置决定所生成文件内容的字符集编码。为避免文本乱码或字符显示不正确， MATLAB 会话的区域设置必须与编译器和操作系统的设置兼容。有关查找和更改操作系统设置的信息，请参阅“国际化” 或操作系统文档。

### 准备生成支持混合语言和区域设置的代码

要准备为模型生成代码，请确定：

* 操作系统区域设置。
* MATLAB 会话的区域设置。
* 以下各项的代码生成要求：
  + Target Language Compiler 文件
  + 包含注释的代码生成模板文件（需要 Embedded Coder）

### 字符集限制

Target Language Compiler 文件仅支持用户默认编码。要生成可移植的国际自定义生成代码，请使用 7位 ASCII 字符集。

### XML 转义序列替换

代码生成器用 XML 转义序列替换无法以模型的字符集编码表示的字符。对于出现在以下位置的模块、信号和 Stateflow 对象名称，会进行转义序列替换：

* 生成的代码注释
* 代码生成报告
* 记录到 MAT 文件的模块路径
* 记录到 C API 文件 **model\_capi.c**（或 **.cpp**）和 **model\_capi.h** 的模块路径

### 生成和检查混用多种语言和区域设置的代码

此示例说明如何使用代码生成器生成和查看要在具有多种语言和多种区域设置环境中使用的代码。

在使用本示例之前，请参阅“国际化和代码生成” [（第 23-2](#_bookmark172) 页）或“国际化和代码生成” (Embedded Coder)。

**rtwdemo\_unicode** 模型配置使用 Embedded Coder➅ **ert.tlc** 系统目标文件。要查看 Simulink➅ Coder™ 的国际化和本地化支持，请将模型配置为使用 **grt.tlc** 系统目标文件。该示例指示特定于 Embedded Coder 的支持（例如，代码生成模板）。

模型配置指定文件和设置，用于控制代码生成器如何处理以下内容的本地化：

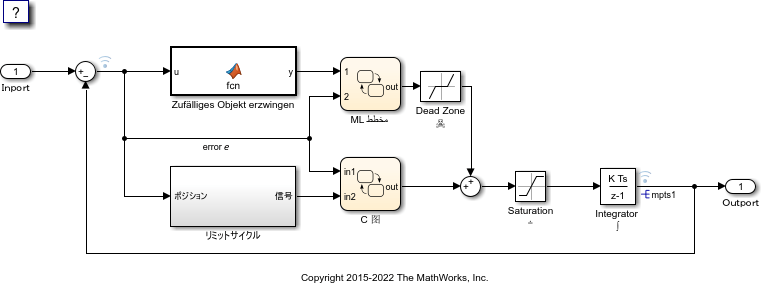
* C 和 C++ API 接口
* 代码生成模板 (CGT) 文件（需要 Embedded Coder）
* 应用代码自定义的 Target Language Compiler (TLC) 文件（需要 Embedded Coder）

#### 打开示例模型 rtwdemo\_unicode。

打开示例模型 **rtwdemo\_unicode**。模型中的标签显示为多种语言（阿拉伯语、中文、英语、德语和日语）和各种 Unicode 符号。

**model = 'rtwdemo\_unicode'; open\_system(model);**

**%**



#### 打开 Embedded Coder

在 Apps 选项卡中，选择 **Embedded Coder**。**验证区域设置**

验证 MATLAB➅ 软件的区域设置是否与您的编译器兼容。请参阅您的操作系统文档或以下 MATLAB 文档：

* “在 Microsoft Windows 平台上进行区域设置”
* “在 Linux 平台上进行区域设置”
* “在 macOS 平台上进行区域设置”

#### 代码生成模板文件

要在生成代码时使用包含 Unicode 字符的代码生成模板文件，请完成以下步骤（需要 Embedded Coder）。否则，请转到下一节。

1. 打开 Configuration Parameters 对话框。
2. 导航到 **Code Generation > Template** 窗格。模型配置为使用代码生成模板文件 **rtwdemo\_unicode.cgt**。该文件会在生成的代码文件的顶部添加注释。要使代码生成器对 **.cgt** 文件应用转义序列替换，请通过指定以下内容启用替换：

#### <encodingIn = "encoding-name">

1. 打开文件 **/toolbox/rtw/rtwdemos/rtwdemo\_unicode.cgt**。 **edit rtwdemo\_unicode.cgt**
2. 找到为字符集编码 **UTF-8** 启用转义序列替换的代码行。

**<encodingIn = "UTF-8">**

1. 关闭文件 **/toolbox/rtw/rtwdemos/rtwdemo\_unicode.cgt**。

**生成的文件自定义模板**

要在生成代码时使用包含 Unicode 字符的文件自定义模板，请完成以下步骤（需要 Embedded Coder）。否则，请转到下一节。

您可以使用 TLC 代码为生成的代码文件指定自定义。TLC 文件仅支持用户默认编码。要生成可移植的国际自定义生成代码，请使用 7 位 ASCII 字符集。

1. 打开 Configuration Parameters 对话框。
2. 导航到 **Code Generation > Template** 窗格。该模型配置为使用代码自定义文件 **example\_file\_process.tlc**。该文件在代码生成器写入代码文件之前对生成的代码进行自定义。例如，该文件添加一个 C 源文件、对应的 include 文件以及 **#define** 和 **#include** 语句。

#### 打开文件 /toolbox/rtw/rtwdemos/example\_file\_process.tlc。 edit example\_file\_process.tlc

4.在生成代码之前，取消注释以下代码行：

**%% %assign ERTCustomFileTest = TLC\_TRUE**%

5.关闭文件 **/toolbox/rtw/rtwdemos/example\_file\_process.tlc**。

**生成 C 代码和报告**

生成 C 代码和代码生成报告。 **evalc('slbuild(''rtwdemo\_unicode'')');查看生成的代码**

对于不在当前 MATLAB 字符集编码中的字符，代码生成器将使用转义序列替换在代码生成报告中正确显示这些字符。

1. 如果模型 **rtwdemo\_unicode** 的代码生成报告尚未打开，请在命令行窗口中键入：

**coder.report.open('rtwdemo\_unicode')**

1. 在 **rtwdemo\_unicode.c** 和 **rtwdemo\_unicode.h** 中查看生成的代码。模型元素的名称在代码注释中显示为本地语言的替换名称。
2. 打开 Traceability Report。此报告保存可追溯性信息，即使名称中包含当前编码未表示的字符时也是如此。模型元素的名称在报告中显示为本地语言的替换名称。
3. 向下滚动并点击第一个图 (**State 'Selection' <S2>:23**) 的代码位置链接。报告视图发生改变，以显示

**rtwdemo\_unicode.c** 中的相应代码。

1. 在代码注释中，点击 **<S2>:23** 链接。模型窗口将在新选项卡上显示该图。
2. 在模型窗口中，右键点击该图。从上下文菜单中，选择 **C/C++ Code > Navigate to C/C++ Code**。报告视图发生改变，以显示该图的命名常量代码段。

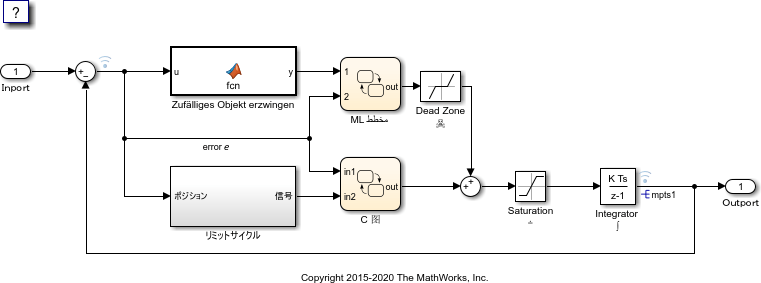
7.关闭代码生成报告、模型顾问和模型。在命令行窗口中键入：

**coder.report.close(); bdclose('all');**

#### 生成 C++ 代码

生成 C++ 代码和代码生成报告。 1.打开模型。

**model = 'rtwdemo\_unicode'; open\_system(model);**



1. 将模型配置参数 **Language** 设置为 **C++**。或者，在命令行窗口中键入：

**set\_param('rtwdemo\_unicode','TargetLang','C++');**

1. 将模型配置参数 **Code interface packaging** 设置为 **C++ class**。或者，在命令行窗口中键入：

**set\_param('rtwdemo\_unicode','CodeInterfacePackaging','C++ class');**

1. 生成 C++ 代码和代码生成报告。

**evalc('rtwrebuild(''rtwdemo\_unicode'')');**

1. 要查看国际化和本地化支持，请查看生成的代码。请参阅**查看生成的代码**。
2. 关闭代码生成报告和模型。在命令行窗口中键入：

**coder.report.close(); bdclose('all');**

### 另请参阅详细信息

* + “国际化的区域设置概念”

# Simulink Coder 基础代码生成

* + [“使用 Simulink Coder 生成代码” （第 24-2 页）](#_bookmark179)
  + [“飞机位置雷达模型” （第 24-8 页）](#_bookmark180)

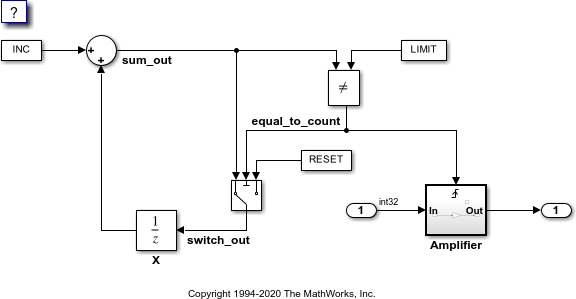
## 使用 Simulink Coder 生成代码

此示例说明如何为 Simulink➅ 模型选择系统目标文件，为实时仿真生成 C 代码，并查看生成的文件。

该模型代表为触发子系统馈送信号的一个 8 位计数器，该子系统由常量模块 **INC**、**LIMIT** 和 **RESET** 进行参数化。**Input** 和 **Output** 代表模型的 I/O。**Amplifier** 子系统按增益因子 **K** 放大输入信号，当信号 **equal\_to\_count** 为 **true** 时，增益因子将会更新。

1. 打开模型 **rtwdemo\_rtwintro**，并将副本保存到可写位置。

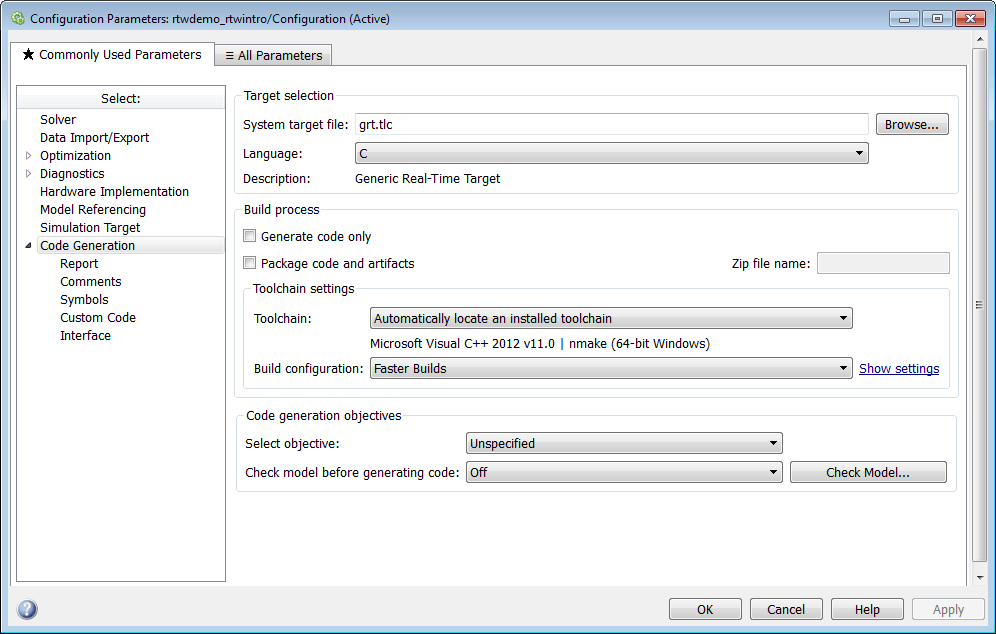
**model='rtwdemo\_rtwintro'; open\_system(model)**



1. 打开 **Simulink Coder**。
2. 打开 Model Configuration Parameters 对话框。在 **C Code** 选项卡中，点击 **Settings**。或者，从命令行中输入：

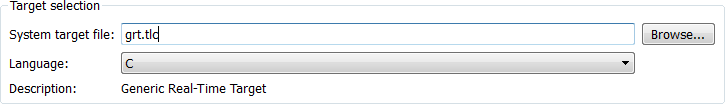
**cs = getActiveConfigSet(model); openDialog(cs);**

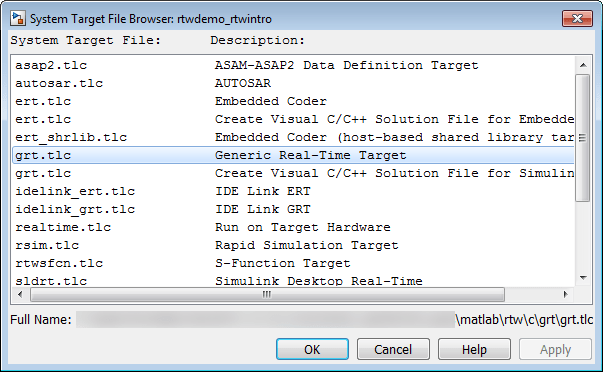
1. 选择 **Code Generation** 选项卡。



1. 在 **Target Selection** 窗格中，点击 **Browse** 以选择系统目标文件。

您可以为特定的目标计算机环境或目的生成代码。一些选项由内置系统目标文件提供，这些文件控制目标计算机环境的代码生成过程。



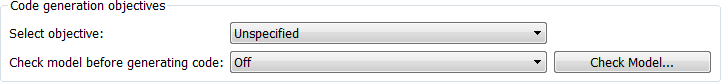


1. 选择 **Generic Real-Time (GRT)** 系统目标文件，然后点击 **Apply**。

（可选）在 **Code Generation Advisor** 窗格中，将参数 **Select objective** 设置为 **Execution efficiency** 或 **Debugging**。然后，要确定并系统地更改参数以满足您的目标，请点击 **Check model...**

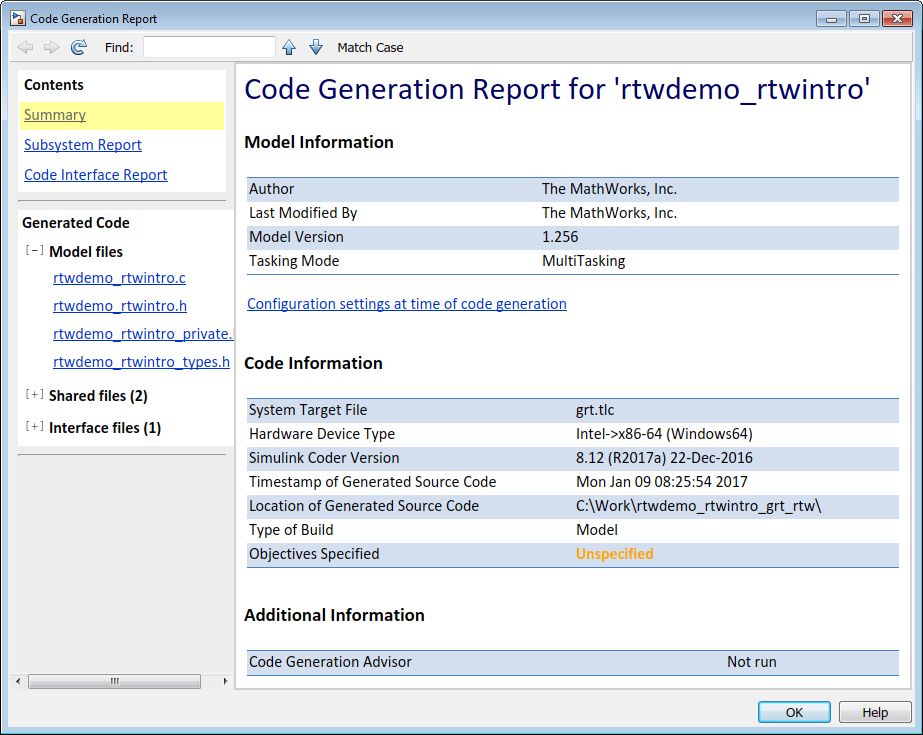
7.使用以下选项之一为模型生成代码：

* + 在 **C Code** 选项卡上，点击 **Build**。
  + 按 **Ctrl+B**。



8.查看代码生成报告。

该报告包括模型文件的链接，如 **rtwdemo\_rtwintro.c** 以及相关联的实用工具和头文件。



以下图窗包含 **rtwdemo\_rtwintro.c** 的一部分。

File: rtwdemo rtwintro.c

rtwdemo

uint5 T rtb sum out: boolean T rtb equal

step{void}

{uint6 T} {lU + {uint32 T}rtwdemo

equal

equal

!= PO5 ZCSIG}}

{rtwdemo

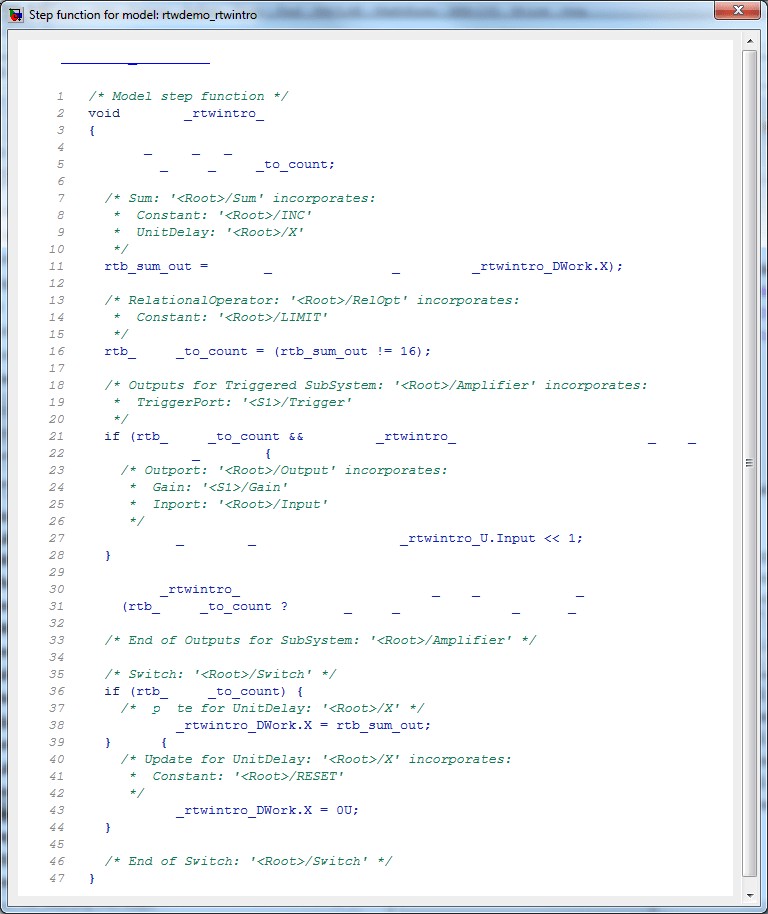
PrevZC5ig5tate.Amplifier Trig ZCE

rtwdemo rtwintro Y.Output = rtwdemo

rtwdemo

equal

PrevZC5ig5tate.Amplifier Trig ZCE = {uint5 T} int32 T}PO5 : int32 T}ZERO ZCSIG}:



equal da

rtwdemo else

rtwdemo

9.关闭模型。

**bdclose(model) rtwdemoclean;**

#### 相关主题

* [“配置模型、生成代码和仿真” （第 25-2 页）](#_bookmark182)
* [“配置系统目标文件” （第 22-2 页）](#_bookmark161)
* “使用 Embedded Coder 生成代码” (Embedded Coder)
* [“在工程中生成代码并进行模型仿真” （第 25-29 页）](#_bookmark199)
* [“使用工程以编程方式生成代码和仿真模型” （第 25-32 页）](#_bookmark200)

## 飞机位置雷达模型

此模型显示为包含 MATLAB➅ 脚本的 Simulink➅ 模型生成的代码。

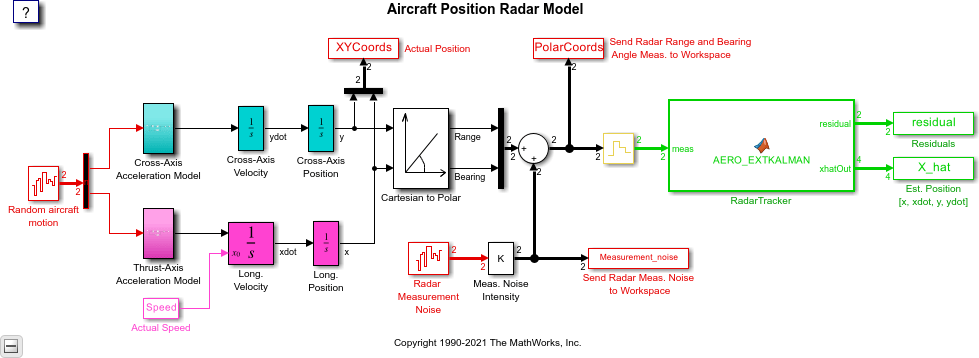
该模型包含一个扩展卡尔曼滤波器，用于根据雷达测量值估计飞机位置。MATLAB 脚本 rtwdemo\_eml\_aero\_radar.m 包含用于运行模型的数据。估计位置和实际位置将保存到工作区，并在仿真结束时由程序 rtwdemo\_aero\_radplot（自动从仿真中调用）进行绘图。

#### 查看和仿真模型

在本节中，您将查看模型并执行仿真。打开 Simulink 模型。

**model='rtwdemo\_eml\_aero\_radar'; open\_system(model) rtwdemo\_eml\_aero\_radar([],[],[],'compile');**

**rtwdemo\_eml\_aero\_radar([],[],[],'term');**



在 MATLAB 编辑器中打开 MATLAB Function 模块 **RadarTracker**。

**open\_system([model,'/RadarTracker'])**

[residual, xhatout] = ARRO EXTlUJ24AN(meas, deltat)

% AERO EXTKALMAN Radar Data Processing Tracker an Extended Kalman Filter. Radar update time deltat inherited workspace.

Initialization **persistent** persistent xhat

isempty(P) xhat =

zeros(4);

16 —

Compute Chi, and R deltat

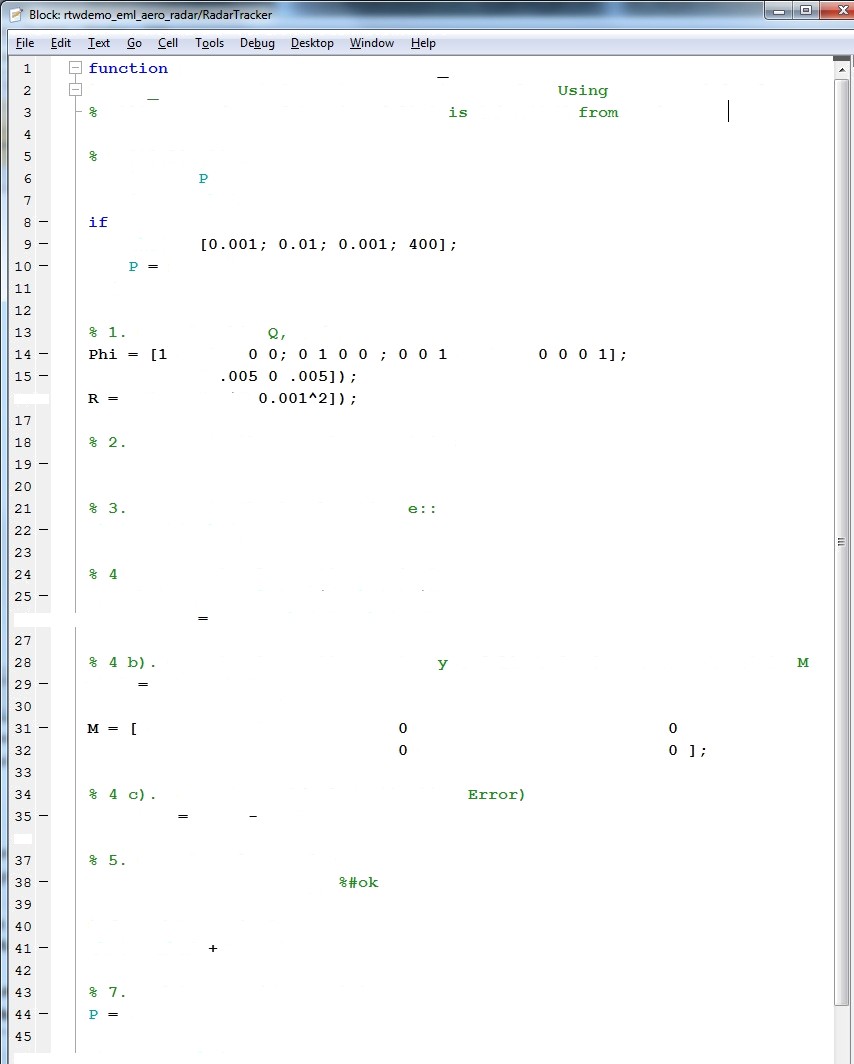
Q = diag([0

diag([300’2

Propagate the covariance matrix: P = Phi\*P\*Phi' + Q;

the track estimat xhat = Phi\*xhat;

deltat;



26 —

36

a). Compute observation estimates: Rangehat = sqrt(xhat(l)’2+xhat(3)’2); Bearinghat atan2(xhat(3),xhat(l));

Compute observation vector and linearized measurement matrix yhat [Rangehat;

Bearinghat];

cos(Bearinghat) sin(Bearinghat)

-sin(Bearinghat)/Rangehat cos(Bearinghat)/Rangehat

Compute residual (Estimation residual meas yhat;

Compute Kalman Gain: W = **P\*M'\*inv(M\*P\*M'+** R);

% 6. Update estimate

xhat = xhat W\*residual;

Update Covariance Matrix

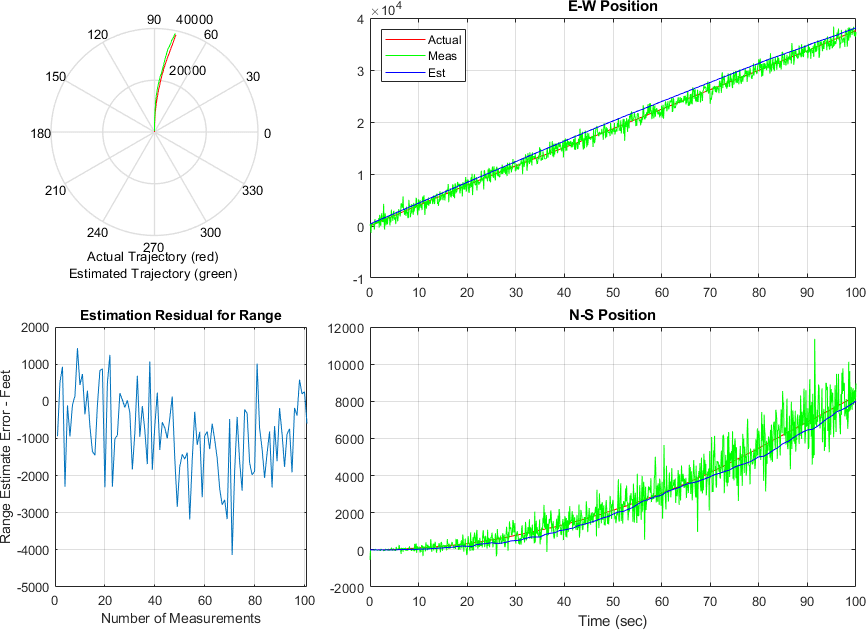
(eye(4)-W\*M)\*P\*(eye(4)-W\*M)' + W\*R\*W';

46 —

xhatout = xhat:

仿真模型并查看结果（自动显示）。

**sim(model)**



#### 为模型生成代码

在本节中，您将使用 Simulink Coder 提供的子系统编译功能为模型的卡尔曼滤波器部分生成代码。在第一个编译中，该模型配置为使用 Simulink Coder™ 生成代码。在第二个编译中，该模型配置为使用 Embedded Coder➅ 生成代码。

**% Create a temporary folder (in your system's temporary folder) for the**

**% build and inspection process. currentDir = pwd;**

**[~,cgDir] = rtwdemodir();**

使用 Simulink Coder 配置和编译模型。

**rtwconfiguredemo(model,'GRT') slbuild([model,'/RadarTracker'])**

**### Starting build procedure for: RadarTracker**

**### Successful completion of build procedure for: RadarTracker**

**Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**=============================================================================================**

**RadarTracker Code generated and compiled Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 19.709s**

使用 Embedded Coder 配置和编译模型。

**rtwconfiguredemo(model,'ERT') slbuild([model,'/RadarTracker'])**

**### Starting build procedure for: RadarTracker**

**### Successful completion of build procedure for: RadarTracker Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**=============================================================================================**

**RadarTracker Code generated and compiled Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 16.443s**

下面列出了 **RadarTracker.c** 的一部分。

**cfile = fullfile(cgDir,'RadarTracker\_ert\_rtw','RadarTracker.c'); rtwdemodbtype(cfile,'/\* Model step', '/\* Model initialize', 1, 0);**

**/\* Model step function \*/**

**void RadarTracker\_step(void)**

**{**

**real\_T P\_d\_tmp[16]; real\_T Phi\_0[16]; real\_T Q[16];**

**real\_T Q\_0[16]; real\_T M[8]; real\_T W[8]; real\_T tmp[8]; real\_T x\_tmp[8]; real\_T Phi\_1[4]; real\_T Bearinghat; real\_T M\_tmp; real\_T M\_tmp\_0; real\_T Rangehat; real\_T r;**

**real\_T tmp\_0; int32\_T Phi\_tmp;**

**int32\_T Phi\_tmp\_tmp; int32\_T i;**

**int32\_T j; int8\_T Phi[16];**

**static const real\_T e[4] = { 0.0, 0.005, 0.0, 0.005 }; static const real\_T c\_b[4] = { 90000.0, 0.0, 0.0, 1.0E-6 };**

**/\* MATLAB Function: '<Root>/RadarTracker' incorporates:**

**\* Inport: '<Root>/meas'**

**\*/**

**Phi[0] = 1;**

**Phi[4] = 1;**

**Phi[8] = 0;**

**Phi[12] = 0;**

**Phi[2] = 0;**

**Phi[6] = 0;**

**Phi[10] = 1;**

**Phi[14] = 1;**

**Phi[1] = 0;**

**Phi[3] = 0;**

**Phi[5] = 1;**

**Phi[7] = 0;**

**Phi[9] = 0;**

**Phi[11] = 0;**

**Phi[13] = 0;**

**Phi[15] = 1;**

**memset(&Q[0], 0, sizeof(real\_T) << 4U); for (j = 0; j < 4; j++) {**

**Q[j + (j << 2)] = e[j];**

**for (i = 0; i < 4; i++) { Phi\_tmp\_tmp = i << 2; Phi\_tmp = j + Phi\_tmp\_tmp; Phi\_0[Phi\_tmp] = 0.0;**

**Phi\_0[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp] \* (real\_T)Phi[j]; Phi\_0[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp + 1] \* (real\_T)Phi[j + 4]; Phi\_0[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp + 2] \* (real\_T)Phi[j + 8]; Phi\_0[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp + 3] \* (real\_T)Phi[j + 12];**

**}**

**}**

**for (i = 0; i < 4; i++) { Phi\_1[i] = 0.0;**

**for (j = 0; j < 4; j++) { Phi\_tmp\_tmp = (j << 2) + i;**

**rtDW.P\_d[Phi\_tmp\_tmp] = (((Phi\_0[i + 4] \* (real\_T)Phi[j + 4] + Phi\_0[i] \* (real\_T)Phi[j]) + Phi\_0[i + 8] \* (real\_T)Phi[j + 8]) + Phi\_0[i + 12] \* (real\_T)Phi[j + 12]) + Q[Phi\_tmp\_tmp];**

**Phi\_1[i] += (real\_T)Phi[Phi\_tmp\_tmp] \* rtDW.xhat[j];**

**}**

**}**

**rtDW.xhat[0] = Phi\_1[0]; rtDW.xhat[1] = Phi\_1[1]; rtDW.xhat[2] = Phi\_1[2]; rtDW.xhat[3] = Phi\_1[3];**

**Rangehat = sqrt(rtDW.xhat[0] \* rtDW.xhat[0] + rtDW.xhat[2] \* rtDW.xhat[2]); Bearinghat = atan2(rtDW.xhat[2], rtDW.xhat[0]);**

**M\_tmp = sin(Bearinghat); M\_tmp\_0 = cos(Bearinghat); M[0] = M\_tmp\_0;**

**M[2] = 0.0;**

**M[4] = M\_tmp;**

**M[6] = 0.0;**

**M[1] = -M\_tmp / Rangehat;**

**M[3] = 0.0;**

**M[5] = M\_tmp\_0 / Rangehat;**

**M[7] = 0.0;**

**rtY.residual[0] = rtU.meas[0] - Rangehat; rtY.residual[1] = rtU.meas[1] - Bearinghat; for (i = 0; i < 2; i++) {**

**for (j = 0; j < 4; j++) { Phi\_tmp\_tmp = (j << 1) + i;**

**x\_tmp[j + (i << 2)] = M[Phi\_tmp\_tmp]; W[Phi\_tmp\_tmp] = 0.0;**

**Phi\_tmp = j << 2;**

**W[Phi\_tmp\_tmp] += rtDW.P\_d[Phi\_tmp] \* M[i]; W[Phi\_tmp\_tmp] += rtDW.P\_d[Phi\_tmp + 2] \* M[i + 4];**

**}**

**}**

**for (i = 0; i < 2; i++) { for (j = 0; j < 2; j++) {**

**Phi\_tmp\_tmp = j << 2; Phi\_tmp = (j << 1) + i;**

**Phi\_1[Phi\_tmp] = (((x\_tmp[Phi\_tmp\_tmp + 1] \* W[i + 2] + x\_tmp[Phi\_tmp\_tmp]**

**\* W[i]) + x\_tmp[Phi\_tmp\_tmp + 2] \* W[i + 4]) + x\_tmp[Phi\_tmp\_tmp + 3] \* W[i + 6]) + c\_b[Phi\_tmp];**

**}**

**}**

**if (fabs(Phi\_1[1]) > fabs(Phi\_1[0])) { r = Phi\_1[0] / Phi\_1[1];**

**Rangehat = 1.0 / (r \* Phi\_1[3] - Phi\_1[2]); Bearinghat = Phi\_1[3] / Phi\_1[1] \* Rangehat; M\_tmp = -Rangehat;**

**M\_tmp\_0 = -Phi\_1[2] / Phi\_1[1] \* Rangehat; Rangehat \*= r;**

**} else {**

**r = Phi\_1[1] / Phi\_1[0];**

**Rangehat = 1.0 / (Phi\_1[3] - r \* Phi\_1[2]); Bearinghat = Phi\_1[3] / Phi\_1[0] \* Rangehat; M\_tmp = -r \* Rangehat;**

**M\_tmp\_0 = -Phi\_1[2] / Phi\_1[0] \* Rangehat;**

**}**

**for (i = 0; i < 4; i++) { for (j = 0; j < 2; j++) {**

**Phi\_tmp\_tmp = j << 2; Phi\_tmp = i + Phi\_tmp\_tmp; tmp[Phi\_tmp] = 0.0;**

**tmp[Phi\_tmp] += x\_tmp[Phi\_tmp\_tmp] \* rtDW.P\_d[i]; tmp[Phi\_tmp] += x\_tmp[Phi\_tmp\_tmp + 1] \* rtDW.P\_d[i + 4]; tmp[Phi\_tmp] += x\_tmp[Phi\_tmp\_tmp + 2] \* rtDW.P\_d[i + 8]; tmp[Phi\_tmp] += x\_tmp[Phi\_tmp\_tmp + 3] \* rtDW.P\_d[i + 12];**

**}**

**W[i] = 0.0;**

**W[i] += tmp[i] \* Bearinghat; r = tmp[i + 4];**

**W[i] += r \* M\_tmp;**

**tmp\_0 = W[i] \* rtY.residual[0]; W[i + 4] = 0.0;**

**W[i + 4] += tmp[i] \* M\_tmp\_0; W[i + 4] += r \* Rangehat;**

**rtDW.xhat[i] += W[i + 4] \* rtY.residual[1] + tmp\_0;**

**}**

**for (i = 0; i < 16; i++) { Phi[i] = 0;**

**}**

**Phi[0] = 1;**

**Phi[5] = 1;**

**Phi[10] = 1;**

**Phi[15] = 1;**

**memset(&Q[0], 0, sizeof(real\_T) << 4U); for (j = 0; j < 4; j++) {**

**Q[j + (j << 2)] = 1.0;**

**for (i = 0; i < 4; i++) { Phi\_tmp\_tmp = (i << 2) + j; P\_d\_tmp[Phi\_tmp\_tmp] = 0.0; Phi\_tmp = i << 1;**

**P\_d\_tmp[Phi\_tmp\_tmp] += M[Phi\_tmp] \* W[j]; P\_d\_tmp[Phi\_tmp\_tmp] += M[Phi\_tmp + 1] \* W[j + 4];**

**}**

**}**

**for (i = 0; i < 16; i++) { Q\_0[i] = Q[i] - P\_d\_tmp[i];**

**}**

**for (i = 0; i < 4; i++) { for (j = 0; j < 4; j++) {**

**Phi\_tmp\_tmp = j << 2; Phi\_tmp = i + Phi\_tmp\_tmp; Q[Phi\_tmp] = 0.0;**

**Q[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp] \* Q\_0[i]; Q[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp + 1] \* Q\_0[i + 4]; Q[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp + 2] \* Q\_0[i + 8]; Q[Phi\_tmp] += rtDW.P\_d[Phi\_tmp\_tmp + 3] \* Q\_0[i + 12];**

**Phi\_0[j + (i << 2)] = (real\_T)Phi[Phi\_tmp] - P\_d\_tmp[Phi\_tmp];**

**}**

**M[i] = 0.0;**

**M[i] += W[i] \* 90000.0; M[i + 4] = 0.0;**

**M[i + 4] += W[i + 4] \* 1.0E-6;**

**}**

**for (i = 0; i < 4; i++) { for (j = 0; j < 4; j++) {**

**Phi\_tmp\_tmp = j << 2; Phi\_tmp = i + Phi\_tmp\_tmp; Q\_0[Phi\_tmp] = 0.0;**

**Q\_0[Phi\_tmp] += Phi\_0[Phi\_tmp\_tmp] \* Q[i]; Q\_0[Phi\_tmp] += Phi\_0[Phi\_tmp\_tmp + 1] \* Q[i + 4]; Q\_0[Phi\_tmp] += Phi\_0[Phi\_tmp\_tmp + 2] \* Q[i + 8];**

**Q\_0[Phi\_tmp] += Phi\_0[Phi\_tmp\_tmp + 3] \* Q[i + 12]; P\_d\_tmp[Phi\_tmp] = 0.0;**

**P\_d\_tmp[Phi\_tmp] += M[i] \* W[j]; P\_d\_tmp[Phi\_tmp] += M[i + 4] \* W[j + 4];**

**}**

**}**

**for (i = 0; i < 16; i++) {**

**rtDW.P\_d[i] = Q\_0[i] + P\_d\_tmp[i];**

**}**

**/\* Outport: '<Root>/xhatOut' incorporates:**

**\* MATLAB Function: '<Root>/RadarTracker'**

**\*/**

**rtY.xhatOut[0] = rtDW.xhat[0]; rtY.xhatOut[1] = rtDW.xhat[1]; rtY.xhatOut[2] = rtDW.xhat[2]; rtY.xhatOut[3] = rtDW.xhat[3];**

**}**

您可以在详细的 HTML 报告中查看生成的完整代码，该报告可在模型和代码之间提供双向可追溯性。

**web(fullfile(cgDir,'RadarTracker\_ert\_rtw','html','index.html'))**

关闭模型并进行清理。

**bdclose(model) rtwdemoclean; cd(currentDir)**

# Simulink Coder 中的源代码生成

* [“配置模型、生成代码和仿真” （第 25-2 页）](#_bookmark182)
* [“管理编译过程文件夹” （第 25-11 页）](#_bookmark191)
* [“管理编译过程文件” （第 25-15 页）](#_bookmark194)
* [“使用跨版本工作流集成生成的代码” （第 25-21 页）](#_bookmark198)
* [“在工程中生成代码并进行模型仿真” （第 25-29 页）](#_bookmark199)
* [“使用工程以编程方式生成代码和仿真模型” （第 25-32 页）](#_bookmark200)

## 配置模型、生成代码和仿真

[“关于本示例” （第 25-2 页）](#_bookmark183)

[“模型的功能设计” （第 25-2 页）](#_bookmark184) [“查看顶层模型” （第 25-2 页）](#_bookmark185) [“查看子系统” （第 25-3 页）](#_bookmark186) [“仿真测试环境” （第 25-4 页）](#_bookmark187) [“运行仿真测试” （第 25-8 页）](#_bookmark188) [“关键点” （第 25-9 页）](#_bookmark189) [“了解更多信息” （第 25-10 页）](#_bookmark190)

**本节内容**

### 关于本示例

#### 学习目标

* 了解示例模型的功能性行为。
* 了解示例测试框架及其组件的作用。
* 对模型运行仿真测试。

#### 前提条件

* 能够打开和修改 Simulink 模型和子系统。
* 了解子系统以及如何查看子系统详细信息。
* 了解引用模型以及如何查看引用模型详细信息。
* 能够设置模型配置参数。

#### 需要的文件

在使用每个示例模型文件之前，将副本放在可写位置并将其添加到 MATLAB 路径中。

* **rtwdemo\_throttlecntrl** 模型文件
* **rtwdemo\_throttlecntrl\_testharness** 模型文件

### 模型的功能设计

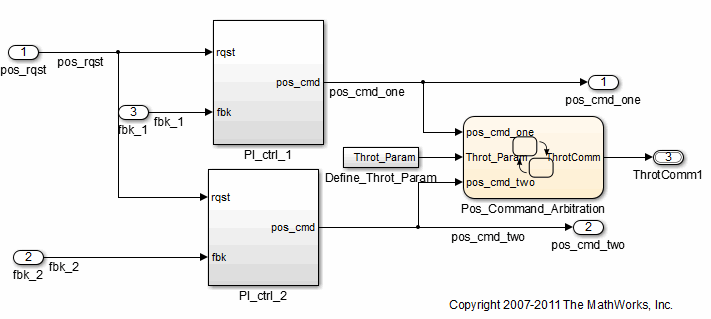
此示例使用简单但功能完整的节气门控制器示例模型。该模型采用冗余控制算法。该模型重点介绍算法设计中的一个标准模型结构和一组基本模块。

### 查看顶层模型

打开 **rtwdemo\_throttlecntrl** 并在 MATLAB 路径上的可写位置保存一个副本，将其命名为

#### throttlecntrl。

**注意** 此模型使用 Stateflow 软件。



模型的顶层包含以下元素：

|  |  |
| --- | --- |
| 子系统 | **PI\_ctrl\_1 PI\_ctrl\_2**  **Define\_Throt\_Param Pos\_Command\_Arbitration** |
| 顶层输入 | **pos\_rqst fbk\_1 fbk\_2** |
| 顶层输出 | **pos\_cmd\_one pos\_cmd\_two ThrotComm1** |
| 信号路由 |  |
| 省略了更改信号值的模块，例如 Sum 和 Integrator |  |

布局使用基本的模型架构风格：

* 计算与信号路由（信号线和总线）分离
* 划分为子系统

您可以将此风格应用于各种模型。

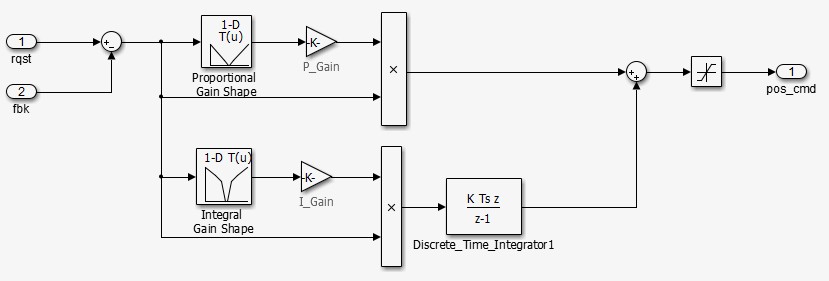
### 查看子系统

浏览顶层模型中的两个子系统。

1. 打开 **throttlecntrl**（如果尚未打开）。

顶层模型中的两个子系统表示比例积分 (PI) 控制器 **PI\_ctrl\_1** 和 **PI\_ctrl\_2**。在此阶段，这些相同的子系统使用相同的数据。

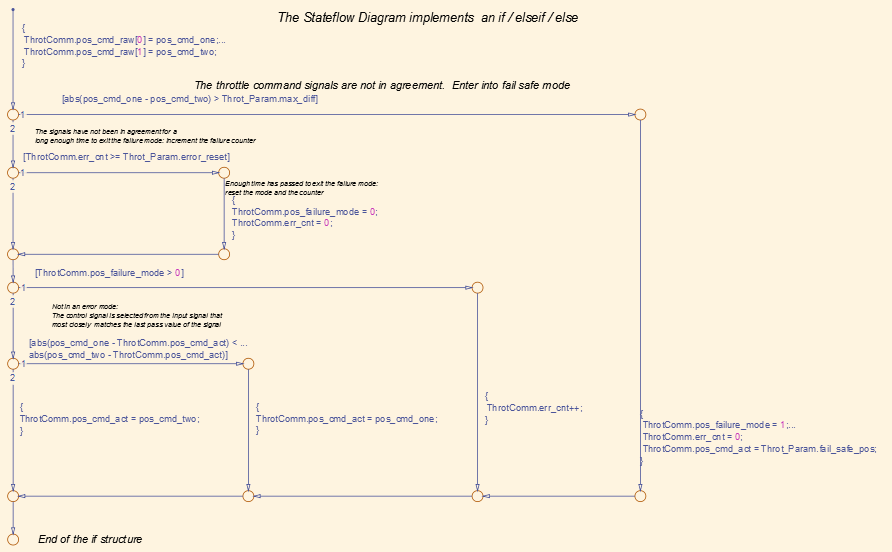
1. 打开 **PI\_ctrl\_1** 子系统。



模型中的 PI 控制器来自库，库是一组可重用的相关模块或模型。库提供包含和重用模型的两种方法之[一。“仿真测试环境” （第 25-4](#_bookmark187) 页）中介绍了第二种方法，即模型引用。您不能编辑从库中添加

到模型的模块。在库中编辑该模块，以便该模块的实例在不同的模型中保持一致。

1. 打开 **Pos\_Command\_Arbitration** 子系统。以下 Stateflow 图会对这两个命令信号执行基本的错误检查。如果这些命令信号相距太远，Stateflow 图会将输出设置为 **fail\_safe** 位置。



1. 关闭 **throttlecntrl**。

### 仿真测试环境

要测试节气门控制器算法，请将其合并到测试框架中。测试框架是一个用于评估控制算法的模型，它具有以下优势：

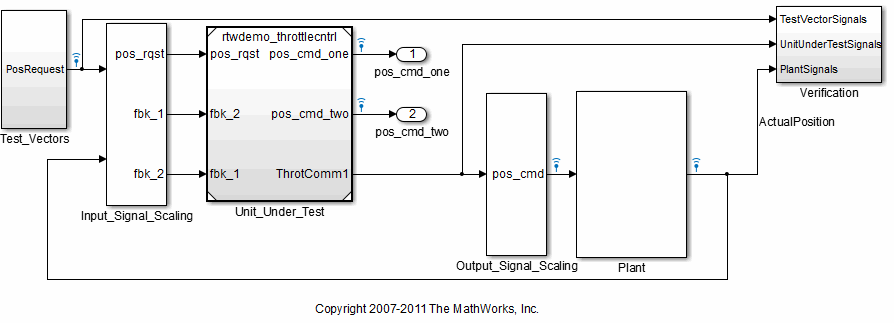
* 将测试数据与控制算法分开。
* 将被控对象或反馈模型与控制算法分开。
* 为控制算法的多个版本提供可重用的环境。

此示例的测试框架模型实现一个常见的仿真测试环境，包括以下各部分：

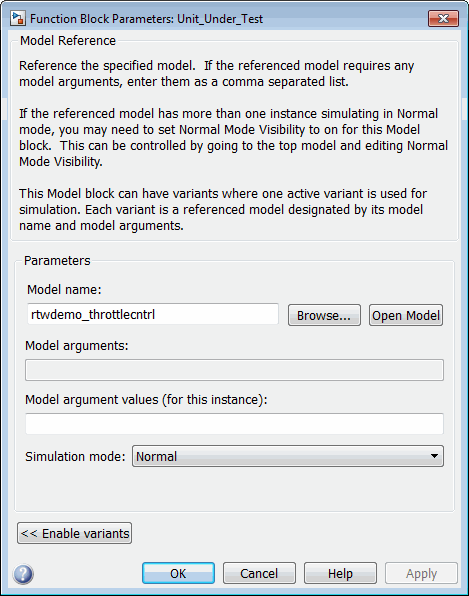
* 在测单元
* 测试向量源
* 计算和记录
* 被控对象或反馈系统
* 输入和输出缩放

浏览仿真测试环境。

1. 打开测试框架模型 **rtwdemo\_throttlecntrl\_testharness** 并在 MATLAB 路径上的可写位置保存一个副本，将其命名为 **throttlecntrl\_testharness**。



1. 将 **throttlecntrl** 模型设置为测试框架的控制算法。
   1. 打开 **Unit\_Under\_Test** 模块并查看控制算法。
   2. 右键点击 **Unit\_Under\_Test** 模块并选择 **Block Parameters (ModelReference)**，以查看模型引用参数。



**rtwdemo\_throttlecntrl** 显示为引用模型的名称。

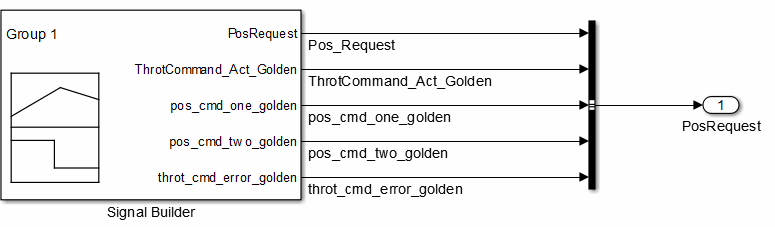
* 1. 将 **Model name** 的值更改为 **throttlecntrl**。
  2. 点击 **Simulation** > **Update Diagram** 以更新测试框架模型图。

由 Model 模块的名称 **Unit\_Under\_Test** 可以看出，控制算法是在测单元。

Model 模块提供重用组件的方法。它允许您在顶层模型中以编译函数形式（直接或间接）引用其他模型。默认情况下，Simulink 软件会在引用模型更改时重新编译模型。相对于库，编译函数有下列优 点：

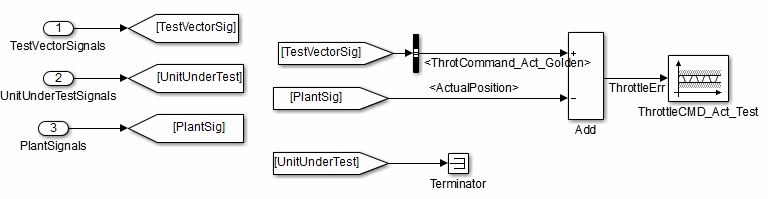
* + 大型模型的仿真时间更快。
  + 您可以直接仿真编译函数。
  + 仿真需要的内存更少。内存中只有一个已编译模型副本，即使模型被多次引用也是如此。

1. 打开测试向量源，它在此测试框架中实现为 **Test\_Vectors** 子系统。



该子系统对测试向量源使用 Signal Builder 模块。该模块具有驱动仿真的数据 (**PosRequest**)，并提供 **Verification** 子系统使用的预期结果。此示例测试框架仅使用一组测试数据。通常，需要创建一个完全覆盖系统运行的测试套件。

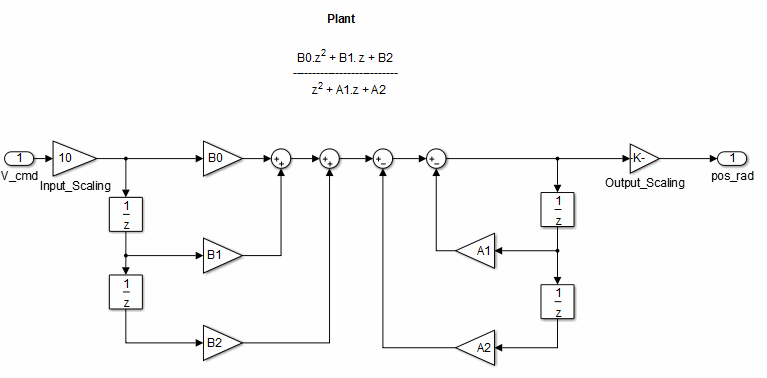
1. 打开计算和记录子系统，它在此测试框架中实现为子系统 **Verification**。



测试框架将控制算法仿真结果与黄金数据进行比较，黄金数据是一组体现经专家认证的控制算法的预期行为的测试结果。在 **Verification** 子系统中，Assertion 模块将来自被控对象的仿真节气门值位置与来自测试框架的黄金值进行比较。如果两个信号之间的差值大于 5%，则测试失败，Assertion 模块将停止仿真。

您也可以在仿真完成执行后计算仿真数据。使用 MATLAB 脚本或第三方工具执行计算。执行后的计算结果为数据分析提供了更大的灵活性。但是，它需要等待执行完成。结合使用这两种方法可以提供高度灵活和高效的测试环境。

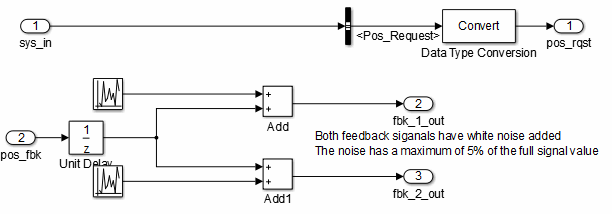
1. 打开被控对象或反馈系统，它在此测试框架中实现为 **Plant** 子系统。



**Plant** 子系统使用一种标准形式的传递函数对节气门动态特性进行建模。您可以创建不同保真度级别的被控对象模型。在不同测试阶段使用不同被控对象模型是很常见的。

1. 打开输入和输出定标子系统，它在此测试框架中实现为 **Input\_Signal\_Scaling** 和

#### Output\_Signal\_Scaling。





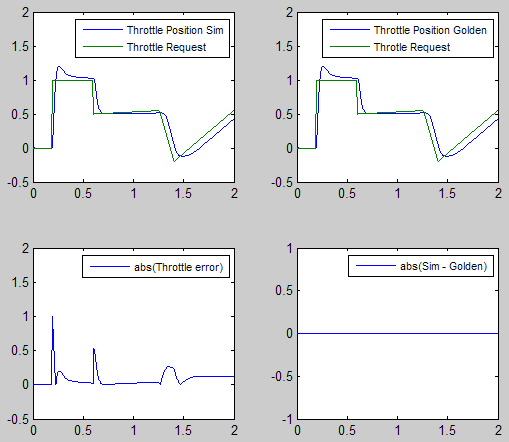
用于对输入和输出进行定标的子系统执行以下主要功能：

* 选择要传送到在测单元的输入信号。
* 选择要传送到被控对象的输出信号。
* 在工程单位和在测单元可写入的单位之间再次缩放信号。
* 处理被控对象与在测单元之间的速率转换。

1. 保存并关闭 **throttlecntrl\_testharness**。

### 运行仿真测试

1. 检查您的工作文件夹是否设置为可写文件夹，例如在其中放置了示例模型文件副本的文件夹。
2. 打开您保存的测试框架模型 **throttlecntrl\_testharness** 副本。
3. 开始进行测试框架模型的仿真。仿真完成后，将显示以下结果。



右下角的图显示预期（黄金）节气门位置和被控对象计算的节气门位置之间的差值。如果两个值之间的差值大于 ±0.05，将停止仿真。

1. 保存并关闭节气门控制器和测试框架模型。

### 关键点

* 基本模型架构将计算与信号路由分开，并将模型划分为多个子系统
* 模型重用的两个选项包括模块库和模型引用。
* 如果将测试框架中的控制算法表示为 Model 模块，请在 Model Reference Parameters 对话框中指定控制算法模型的名称。
* 测试框架是一个用于评估控制算法的模型。通常，框架包含在测单元、测试向量源、计算和记录、被控对象或反馈系统以及输入和输出缩放组件。
* 在测单元是待测试的控制算法。
* 测试向量源提供驱动仿真的数据，该仿真生成用于验证的结果。
* 在验证期间，测试框架会将控制算法仿真结果与黄金数据进行比较并记录结果。
* 测试框架的被控对象或反馈组件对受控环境进行建模。
* 在开发测试框架时，
  + 缩放输入和输出信号。
  + 选择要传送到在测单元的输入信号。
  + 选择要传送到被控对象的输出信号。
  + 在工程单位和在测单元可写入的单位之间再次缩放信号。
  + 处理被控对象与在测单元之间的速率转换。

### 了解更多信息

* “Support Model Referencing”
* “代码生成”
* “信号组”

## 管理编译过程文件夹

编译过程将从 Simulink 图更新和模型编译生成的文件放置到默认指定的文件夹层次结构中。在以下情况下，您可以更改编译过程文件夹的默认设定：

* 您的公司不使用当前工作文件夹作为代码生成文件夹或仿真缓存文件夹的位置。
* 您将代码生成文件夹置于版本控制下，但未将仿真缓存文件夹置于版本控制下。

MATLAB 会话参数 **CacheFolder**、**CodeGenFolder** 和 **CodeGenFolderStructure** 是文件生成控制参数，用于指定编译过程的文件夹位置。在 MATLAB 会话开始时，以下 Simulink 预设项确定这些参数的值：

**“Simulation cache folder”** – **CacheFolder**

**“Code generation folder”** – **CodeGenFolder**

**“Code generation folder structure”** – **CodeGenFolderStructure**

要在 MATLAB 会话中修改参数，请使用 **Simulink.fileGenControl**。您设置的参数值将在 MATLAB 会话结束时过期。

### 文件生成控制参数

下表提供有关如何使用参数来管理编译过程文件夹的信息。

|  |  |  |
| --- | --- | --- |
| **MATLAB 会话参数** | **Simulink 预设项** | **描述** |
| **CacheFolder** | **“Simulation cache folder”** | 编译过程中，基于 Simulink 图更新生成的文件以及用于仿真和代码生成的模型编译工件会被放置在仿真缓存文件夹 **simulationCacheFolder** 中。该文件夹是根文件夹。  默认情况下（即 **CacheFolder = ''** 时），  **simulationCacheFolder** 是当前工作文件夹 **pwd**。  您可以使用该参数指定其他文件夹。例如，如果您要：   * 将生成的文件与模型和其他资源分离。 * 重用或共享以前编译的仿真目标，而不用将当前工作文件夹设置回上次使用的工作文件夹。 |

|  |  |  |
| --- | --- | --- |
| **MATLAB 会话参数** | **Simulink 预设项** | **描述** |
| **CodeGenFolder** | **“Code generation folder”** | 编译过程使用系统目标文件从 Simulink 模型生成生产代码，并将生产代码放入代码生成文件夹 **codeGenerationFolder** 中。该文件夹是根文件夹。 |
|  |  | 如果您选择生成可执行程序文件，编译过程会将文件 **model.exe** (Windows) 或 **model** (UNIX) 写入该文件夹。 |
|  |  | 默认情况下（即 **CodeGenFolder = ''** 时），  **codeGenerationFolder** 是当前工作文件夹 **pwd**。 |
|  |  | 您可以使用该参数指定其他文件夹。例如，如果您要将生成的生产代码与以下各项分离： |
|  |  | * 模型和其他资源。 |
|  |  | * 生成的仿真和代码生成工件。 |
|  |  | 如果您将驱动器的根文件夹指定为代码生成文件夹，则编译过程不能为您的模型生成代码。例如，**C:\**。 |
| **CodeGenFolderStr ucture** | **“Code generation folder structure”** | 要指定代码生成文件夹内的文件夹结构，请使用该参 数。例如，如果您为不同目标环境配置模型，则可以为从每个模型生成的代码指定一个单独的子文件夹。 |

### 编译过程文件夹

下表提供有关 **CodeGenFolderStructure** 如何控制仿真缓存文件夹和代码生成文件夹内的文件夹结构的信息。

|  |  |  |
| --- | --- | --- |
| **当 CodeGenFolderStructure = 'ModelSpecific' 时的文件夹名称** | **当 CodeGenFolderStructure**  **=**  **'TargetEnvironmentSubfold er' 时的文件夹名称** | **描述** |
| **codeGenerationFolder/ model\_target\_rtw** | **codeGenerationFolder/ targetSpecific/model** | 编译文件夹，用于存储生成的源代码和由编译过程创建的其他文件。 |
| **target** 的默认值是所选系统目标文件的名称，例如 **grt**、**ert** 和 **rsim**。您可以使用系统目标文件中的  **rtwgensettings.BuildDirSuf fix** 字段更改 **target**。 | 编译过程使用系统目标文件和硬件设备的配置信息为子文件夹 **targetSpecific** 生成唯一标签。 | 包含生成的代码模块 **model.c** 和 **model.h** 以及生成的联编文件 **model.mk**。  **model.mk** 用于编译和链接从模型组件生成的代码。 |
|  |  | **model** 是源模型的名称。 |

|  |  |  |
| --- | --- | --- |
| **当 CodeGenFolderStructure = 'ModelSpecific' 时的文件夹名称** | **当 CodeGenFolderStructure**  **=**  **'TargetEnvironmentSubfold er' 时的文件夹名称** | **描述** |
| **codeGenerationFolder/ model\_target\_rtw/services** | **codeGenerationFolder/ targetSpecific/model/ services** | 为使用服务代码接口配置的组件模型生成代码时创建的文件夹。  包含 **services.h**，这是为目标平台服务指定函数原型的头文件。 |
|  |  | 有关详细信息，请参阅“Identify and Package Files for Deployment” (Embedded Coder)。 |
| **codeGenerationFolder/ model\_target\_rtw/ services/lib** | **codeGenerationFolder/ targetSpecific/model/ services/lib** | 包含 **buildInfo.mat**，用于构建组件模型库，该库表示根据 **services.h** 编译的生成代码。 |
|  |  | 有关详细信息，请参阅“Identify and Package Files for Deployment” (Embedded Coder)。 |
| **codeGenerationFolder/ model\_target\_rtw/html** | **codeGenerationFolder/ targetSpecific/model/html** | 包含编译过程生成的报告文件的代码生成报告文件夹。 |
| **codeGenerationFolder/ slprj/target/model** | **codeGenerationFolder/ targetSpecific/\_ref/model** | 模型引用目标文件。 |
| **codeGenerationFolder/ slprj/target/model/ referenced\_model\_includes** | **codeGenerationFolder/ targetSpecific/\_ref/model/ referenced\_model\_includes** | 来自 **model** 引用的模型的头文件。 |
| **codeGenerationFolder/ slprj/target/model/ tmwinternal** | **codeGenerationFolder/ targetSpecific/\_ref/model/ tmwinternal** | 代码生成期间使用的 MAT 文件。 |
| **codeGenerationFolder/ slprj/target/\_sharedutils** | **codeGenerationFolder/ targetSpecific/\_shared** | 模型引用系统目标文件的工具函数，在模型之间共享。 |
|  |  | 文件夹还包含用于编译生成的共享实用工具代码的 **rtwshared.mk** |
| **simulationCacheFolder/ slprj/sim/model** | **simulationCacheFolder/ slprj/sim/model** | 引用模型的仿真目标文件。 |
| **simulationCacheFolder/ slprj/sim/model/ tmwinternal** | **simulationCacheFolder/ slprj/sim/model/tmwinternal** | 代码生成期间使用的 MAT 文件。 |
| **simulationCacheFolder/ slprj/sim/\_sharedutils** | **simulationCacheFolder/ slprj/sim/\_sharedutils** | 仿真系统目标文件的工具函数，在模型之间共享。 |

如果系统目标文件是基于 ERT 的，则这些模型配置参数还控制共享实用工具代码的位置：

* 共享代码位置 (**UtilityFuncGeneration**)
* 现有共享代码 (Embedded Coder) (**ExistingSharedCode**)您可以使用 **RTW.getBuildDir** 来显示模型的编译文件夹信息。

### 另请参阅

**RTW.getBuildDir** | **Simulink.fileGenControl**

### 详细信息

* [“管理编译过程文件” （第 25-15 页）](#_bookmark194)
* “Manage Build Process File Dependencies”
* “Add Build Process Dependencies”
* “Build Process Support for File and Folder Names”
* “Build Process Workflow for Real-Time Systems”
* “Generate Code for Model Reference Hierarchy”
* “Cross-Release Shared Utility Code Reuse”
* “Cross-Release Code Integration”
* [“在工程中生成代码并进行模型仿真” （第 25-29 页）](#_bookmark199)
* [“使用工程以编程方式生成代码和仿真模型” （第 25-32 页）](#_bookmark200)

## 管理编译过程文件

要应用从编译过程生成的代码源和头文件，了解编译过程生成的文件以及控制文件生成的条件会很有帮助。这些信息提供对生成的代码资源的访问，例如：

* 模型入口点的公共接口
* 与内置数据类型对应的枚举类型
* 描述模型信号、状态和参数的数据结构体

代码生成器会在代码生成和编译过程中创建 **model.\*** 文件。您可以自定义生成的头文件、源文件和数据文件的文件名。有关详细信息，请参阅“Customize Generated File Names” (Embedded Coder)。代码生成器还会创建其他文件夹和依赖项文件以支持共享实用工具和模型引用。有关编译过程创建的文件夹的详细信息，请参阅“管理编译过程文件夹” [（第 25-11](#_bookmark191) 页）。有关说明如何使用工程来管理编译过程文件夹的示例，请参阅“在工程中生成代码并进行模型仿真” [（第 25-29](#_bookmark199) 页）。

根据模型架构和代码生成选项，在基于 GRT 的系统目标文件的编译过程中，可能会生成一些在基于 ERT的系统目标文件的编译过程中不会生成的文件。此外，对于基于 ERT 的系统目标文件，编译过程对生成的文件的打包方式也与基于 GRT 的系统目标文件不同。请参阅“Manage File Packaging of Generated Code Modules” (Embedded Coder)。

**注意** [默认情况下，编译过程会删除编译文件夹中的外部（非生成的）源文件。可以按照“在编译文件夹中保留外部代码文件” （第 30-10](#_bookmark247) 页）中的规范在编译文件夹中保留外部源文件。

下表描述了生成的主要文件。在表中显示的生成的文件名中，**model** 表示要为其生成代码的模型的名称。 **subsystem** 表示模型中子系统的名称。选择 **Create code generation report** 参数时，代码生成器会生成一组 HTML 文件。对于每个源文件以及编译文件夹内 **html** 子文件夹中的 **model\_contents.html** 索引文件，都有一个对应的 HTML 文件。表中的源文件和头文件具有依存关系。有关其他文件依存关系的描述，请参阅“Manage Build Process File Dependencies”和“Add Build Process Dependencies”。

|  |  |
| --- | --- |
| **文件** | **描述** |
| **builtin\_typeid\_types.h** | 定义与内置数据类型对应的枚举类型。  当满足下列一个或多个条件时，模型编译会生成此文件：   * 模型包含使用消息的 Stateflow 图。 * 模型配置启用 **MAT-file logging**。 * 模型配置在 **Code Generation** > **Interface** 中启用 C API 选项。 |
| **model.bat** | 包含 Windows 批处理文件命令，用于设置编译器环境和调用 **make** 实用工具。  有关使用此文件的详细信息，请参阅 [“model.bat” （第 25-18](#_bookmark195) 页）。 |

|  |  |
| --- | --- |
| **文件** | **描述** |
| **model.c model.cpp** | 对应于模型文件。  Target Language Compiler 生成此 C 或 C++ 源代码文件。该文件包含：   * include 文件 **model.h** 和 **model\_private.h** * 数据，但 **model\_data.c** 中的数据除外 * 特定于模型的调度器代码 * 特定于模型的求解器代码 * 模型注册码 * 算法代码 * 可选的 GRT 包装函数 |
| **model.exe**（Windows 平台）  **model**（UNIX 和 Macintosh 平台） | 可执行程序文件。  模型编译生成此文件，除非您显式指定代码生成器仅生成代码。编译会在开发系统的 **make** 实用工具的控制下，在当前文件夹（而不是编译文件夹）中生成可执行文件。 |
| **model.h** | 定义模型数据结构体和对接模型入口点和数据结构体的公共接口。通过访问宏提供与实时模型数据结构体 (**model\_rtM**) 的接口。  模型中的子系统 **.c** 或 **.cpp** 文件包括 **model.h**。该文件包括：   * 导出的 Simulink 数据符号 * 导出的 Stateflow 机器级别数据 * 模型数据结构体，包括 **rtM** * 模型入口函数   有关详细信息，请参阅 [“model.h” （第 25-19](#_bookmark196) 页）。 |
| **model.mk** | 生成的联编文件，它通过开发系统的 **make** 实用工具控制对生成的代码进行编译并将其链接到最终二进制文件中。  如果设置 **MAKEFLAGS** 环境变量，请不要选择使用此变量且与编译过程使用的当前 **make** 实用工具冲突的选项。 |
| **model.rtw** | 表示已编译的模型。  默认情况下，编译过程完成时，会删除此 ASCII 文件。您可以选择保留该文件以供检查。 |
| **model\_capi.h model\_capi.c** | （可选文件）包含描述模型信号、状态和参数的数据结构体（不使用外部模式）。  有关详细信息，请参阅“使用 [C API 在生成的代码和外部代码之间交换数据”](#_bookmark348)  [（第 34-2](#_bookmark348) 页）。 |

|  |  |
| --- | --- |
| **文件** | **描述** |
| **model\_data.c** | 包含参数数据结构体和常量模块 I/O 数据结构体的声明（在满足条件时生成），以及模型使用的结构体数据类型的零表示形式。  当模型使用这些数据结构体时，模型编译便会生成此文件。结构体的 **extern** 声明显示在 **model.h** 中。如果此文件存在，它会包含：   * 常量模块 I/O 参数 * include 文件 **model.h** 和 **model\_private.h** * 模型使用的用户定义结构体数据类型的零表示形式的定义 * 常量参数 |
| **model\_dt.h** | （可选文件）声明包含生成的模型数据结构体的数据类型和数据类型转换信息的结构体，以支持外部模式。 |
| **model\_private.h** | 包含模型和子系统的局部 **define** 常量和局部数据。  模型编译中生成的源文件包含此文件。将外部代码与从模型生成的代码集成时，请包含 **model\_private.h**。该文件包含：   * 导入的 Simulink 数据符号 * 导入的 Stateflow 机器级别数据 * Stateflow 入口点 * Simulink Coder 详细信息（代码专用的各种宏、**enum** 等）   有关详细信息，请参阅“Manage Build Process File Dependencies”。 |
| **model\_reference\_types.h** | 包含计时桥的类型定义。  模型编译为引用模型或包含模型引用模块的模型生成此文件。 |
| **model\_targ\_data\_map.m** | （可选文件）包含外部模式用于初始化外部模式连接的 MATLAB 语言命令。 |
| **model\_types.h** | 提供实时模型数据结构体和参数数据结构体的前向声明。  从模型编译中生成的头文件包含此文件。可重用函数的函数声明可以使用这些结构体。 |
| **multiword\_types.h** | 包含多字宽数据类型及其字长块的类型定义。如果您的代码使用多字数据类型，请包含此头文件。  当满足下列一个或多个条件时，模型编译会生成此文件：   * 模型使用多字数据类型。 * 模型配置启用 **MAT-file logging**。 * 模型配置启用 **Code Generation** > **Interface** > **External mode**。 |

|  |  |
| --- | --- |
| **文件** | **描述** |
| **rtGetInf.c rtGetInf.h rtGetNaN.c rtGetNaN.h rt\_nonfinite.c**  **rt\_nonfinite.h** | 为 **inf**、负 **inf** 和 **nan** 声明并初始化全局非有限值。提供非有限比较函数。  当满足下列一个或多个条件时，模型编译会生成这些文件：   * 模型包含 S-Function。 * 从模型生成的代码需要非有限数。 * 模型配置启用 **MAT-file logging**。 * 模型配置选择 “**grt.tlc**” 作为 **System target file** 并启用 **Classic call interface**。 |
| **rtmodel.h** | 包含静态主程序模块（如 **rt\_main.c**）所需的 **#include** 指令。  编译过程不会在代码生成时创建这些模块。模块包括 **rtmodel.h** 以访问特定于模型的数据结构体和入口函数。如果您创建自己的主程序模块，请确保包含 **rtmodel.h**。 |
| **rtwtypes.h** | 提供基本的类型定义、**#define** 语句和枚举。  对于不生成 GRT 接口且没有非内联 S-Function 的基于 ERT 的系统目标文件，  **rtwtypes.h** 不包括 **simstruc\_types.h**。  有关详细信息，请参阅“r[twtypes.h” （第 25-19](#_bookmark197) 页）和“Manage Build Process File Dependencies”。 |
| **rtw\_proj.tmw sl\_proj.tmw** | 标记文件。  编译过程会生成这些文件以帮助 **make** 实用工具确定何时重新编译和链接生成的代码。 |
| **rt\_defines.h** | 包含特殊数学常量（例如 π 和 e）的类型定义并定义  **UNUSED\_PARAMETER** 宏。  当生成的代码需要数学常量定义或函数体不访问所需的模型函数参数时，模型编译会生成此文件。 |
| **rt\_sfcn\_helper.h rt\_sfcn\_helper.c** | （可选文件）提供非内联 S-Function 在模型中使用的函数。  非内联 S-Function 使用函数 **rt\_CallSys**、**rt\_enableSys** 和 **rt\_DisableSys**  来调用下游函数调用子系统。 |
| **subsystem.c** | （可选文件）包含每个非内联非虚拟子系统的 C 源代码，或者在子系统配置为将代码放在单独的文件中时复制代码。 |
| **subsystem.h** | （可选文件）包含非内联非虚拟子系统的导出符号。 |

### model.bat

此文件包含 Windows 批处理文件命令，这些命令用于设置编译器环境和调用 **make** 实用工具。

如果您在编译过程中使用工具链方法，则还可以使用此批处理文件从生成的联编文件 **model.mk** 中提取信息。该信息包括出现在联编文件中的宏定义和值、例如 **CFLAGS**（C 编译器标志）和 **CPP\_FLAGS**（C

++ 编译器标志）。选择包含 **model.bat** 的文件夹作为当前工作文件夹，并在命令行窗口中键入：

**>> system('model.bat info')**

在 UNIX 和 Macintosh 平台上，代码生成器不会创建 **model.bat** 文件。要从在这些系统上生成的联编文件中提取工具链方法编译的信息，请在命令行窗口中键入：

**>> system('gmake -f model.mk info')**

### model.h

头文件 **model.h** 声明模型数据结构体和对接模型入口点和数据结构体的公共接口。此头文件还通过使用访问宏提供与实时模型数据结构体 (**model\_M**) 的接口。如果您的代码要与模型函数或模型数据结构体对

接，请包含 **model.h**：

* 导出的全局信号

**extern int32\_T INPUT; /\* '<Root>/In' \*/**

* 全局结构体定义

**/\* Block parameters (auto storage) \*/ extern Parameters\_mymodel mymodel\_P;**

* 实时模型 (RTM) 宏定义

**#ifndef rtmGetSampleTime**

**# define rtmGetSampleTime(rtm, idx) ((rtm)->Timing.sampleTimes[idx]) #endif**

* 模型入口函数（ERT 示例）

**extern void mymodel\_initialize(void); extern void mymodel\_step(void); extern void mymodel\_terminate(void);**

**main.c**（或 **.cpp**）文件包含 **model.h**。如果模型编译从 TLC 脚本生成 **main.c**（或 **.cpp**）文件，则 TLC 源可以包括 **model.h**。

**#include "%<CompiledModel.Name>.h"**

如果 **main.c** 是静态源文件，则可以使用固定的头文件名 **rtmodel.h**。该文件包含 **model.h** 头文件：

**#include "model.h" /\* If main.c is generated \*/**

或

**#include "rtmodel.h" /\* If static main.c is used \*/**

其他外部源文件可能要求包含 **model.h** 以对接模型数据，例如导出的全局参数或信号。由于生成的代码的要求，**model.h** 文件本身可能具有其他头文件依存关系。请参阅“System Header Files”和“Code Generator Header Files”。

要减少依存关系并减少包含的头文件的数目，请参阅“Manage Build Process File Dependencies”。

### rtwtypes.h

头文件 **rtwtypes.h** 定义生成的代码所需的数据类型、结构体和宏。对于 GRT 和 ERT 系统目标文件，需要包含 **rtwtypes.h**，而不是包含 **tmwtypes.h**。

通常，生成的代码要求整数运算在特定值上溢或下溢。例如，当代码需要 16 位整数时，代码不接受 8 位或 32 位整数类型。C 语言没有为 **char**、**int** 等类型中的位数设置标准。因此，C 语言中没有普遍接受的可表示特定字长整数的数据类型。

为了适应 C 语言的这一特性，生成的代码使用具有特定大小的整数类型，例如 **int8\_T**、**uint32\_T** 等，它们不是标准的 C 类型。在 **rtwtypes.h** 中，生成的代码使用配置参数的 **Hardware Implementation** 窗格中的信息将这些具有特定大小的整数类型映射到对应的 C 关键字基类型。

当满足以下条件时，代码生成器会为基于 ERT 的系统目标文件生成 **rtwtypes.h** 的优化版本：

#### 未选择 Configuration Parameters > Code Generation > Interface > Advanced parameters >

**Classic call interface**。

* 模型不包含非内联 S-Function。

包括 **rtwtypes.h**。例如，如果针对 GRT 系统目标文件包含该文件，则可以更轻松地将代码用于基于 ERT的系统目标文件。

对于 GRT 和 ERT 系统目标文件，**rtwtypes.h** 的位置取决于编译过程是否使用共享实用工具位置。如果它使用共享位置，则代码生成器将 **rtwtypes.h** 放在 **slprj/target/\_sharedutils** 中；否则，它将 **rtwtypes.h** 放在编译文件夹 (**model\_target\_rtw**) 中。请参阅“Specify Generated Code Interfaces”。

在源文件使用代码生成器类型名称或其他代码生成器定义时，源文件包括 **rtwtypes.h** 头文件。典型示例是使用代码生成器数据类型声明变量的文件，例如 **uint32\_T myvar**。

代码生成器和 S-Function 使用的源文件可以使用预处理器宏 **MATLAB\_MEX\_FILE**。宏定义来自 **mex**

函数：

**#ifdef MATLAB\_MEX\_FILE #include "tmwtypes.h"**

**#else**

**#include "rtwtypes.h" #endif**

代码生成器 **main.c**（或 **.cpp**）文件的源文件包含 **rtwtypes.h**，但不进行预处理器检查。

**#include "rtwtypes.h"**

Target Language Compiler 生成的自定义源文件还可以将这些 **include** 语句发送到其生成的文件中。请参阅“Control Placement of rtwtypes.h for Shared Utility Code”。

### 另请参阅

**详细信息**

* [“管理编译过程文件夹” （第 25-11 页）](#_bookmark191)
* “Manage Build Process File Dependencies”
* “Add Build Process Dependencies”
* “Build Process Support for File and Folder Names”
* “Build Process Workflow for Real-Time Systems”
* [“在工程中生成代码并进行模型仿真” （第 25-29 页）](#_bookmark199)
* [“使用工程以编程方式生成代码和仿真模型” （第 25-32 页）](#_bookmark200)

## 使用跨版本工作流集成生成的代码

此示例说明如何通过创建跨版本软件在环 (SIL) 模块并将该模块整合到集成模型中来重用以前生成的代码。有关工作流的详细信息，请参阅 “Cross-Release Code Integration” (Embedded Coder)。

#### 从模型中生成代码

在跨版本代码集成工作流中，您可以从以前生成的代码创建一个软件在环 (SIL) 或处理器在环 (PIL) 模块。此示例使用从当前版本中生成的代码。

**model = 'rtwdemo\_crossrelease\_counter'; close\_system(model,0) load\_system(model)**

**set\_param(model, 'SimulationCommand', 'update'); open\_system(model)**

**slbuild(model);**

**### Starting build procedure for: rtwdemo\_crossrelease\_counter**

**### Successful completion of code generation for: rtwdemo\_crossrelease\_counter Build Summary**

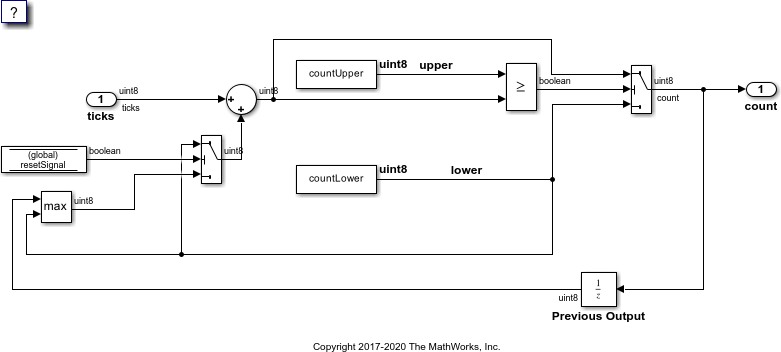
**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_crossrelease\_counter Code generated. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date) Build duration: 0h 0m 12.362s**



标识包含生成代码的编译文件夹。

**buildFolder = RTW.getBuildDir(model).BuildDirectory;**

标识包含与生成的代码相关联的共享实用工具代码的文件夹。

**previousSharedCodeFolder = RTW.getBuildDir(model).SharedUtilsTgtDir;**

要重用以前版本中生成的代码，请执行以下操作：

1. 在**仿真**选项卡的**文件**部分中，选择**保存 > 以前的版本**。
2. 在**另存为类型字段**中，指定版本和模型类型。
3. 点击**保存**。
4. 使用以前的版本打开保存的模型，然后生成代码。
5. 将 **buildFolder** 的值设置为上一版本中生成的代码的位置。
6. 将 **previousSharedCodeFolder** 的值设置为上一版本中生成的共享实用工具的位置。

#### 管理共享代码

将生成的共享源代码文件添加到集成模型使用的存储库文件夹中。

**sharedCodeRepo = 'SharedCodeRepo'; mkdir(sharedCodeRepo);**

**sharedCodeUpdate(previousSharedCodeFolder, sharedCodeRepo, 'Interactive', false);**

**The following files will be copied from slprj/ert/\_sharedutils to SharedCodeRepo/R2023a: rtwtypes.h**

**Files copied from slprj/ert/\_sharedutils to SharedCodeRepo/R2023a.**

#### 导入代码

加载集成模型。

**integrationModel = 'rtwdemo\_crossrelease\_integration'; close\_system(integrationModel, 0); load\_system(integrationModel);**

修改 Simulink 配置集，使其引用现有共享代码库。

**cs = getActiveConfigSet(integrationModel);**

**set\_param(cs, 'ExistingSharedCode', fullfile(pwd, sharedCodeRepo));**

创建跨版本 SIL 模块。

**blockHandle = crossReleaseImport(buildFolder, cs, 'SimulationMode', 'SIL');**

**### Starting import process for component: rtwdemo\_crossrelease\_counter\_R2023a ### Starting build process for SIL block: rtwdemo\_crossrelease\_counter\_R2023a**



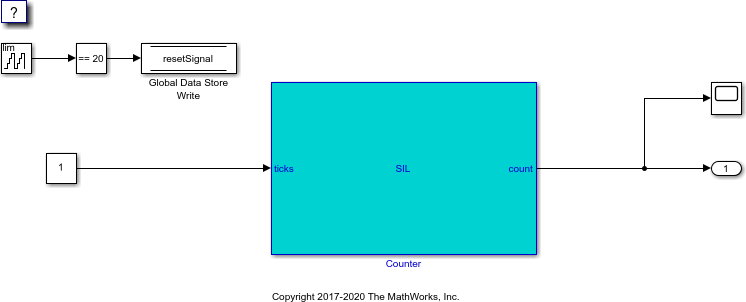
#### 将跨版本 SIL 模块合并到集成模型中

要将集成模型中的模块替换为跨版本模块，请使用 **pil\_block\_replace**。此函数保留模块大小、信号线连接和优先级。

**srcBlock = getfullname(blockHandle); dstBlock = [integrationModel, '/', 'Counter']; pil\_block\_replace(srcBlock, dstBlock) open\_system(integrationModel)**

**Successfully swapped the following blocks:**

**untitled/rtwdemo\_crossrelease\_counter\_R2023a\_sil rtwdemo\_crossrelease\_integration/Counter**



#### 集成模型的仿真

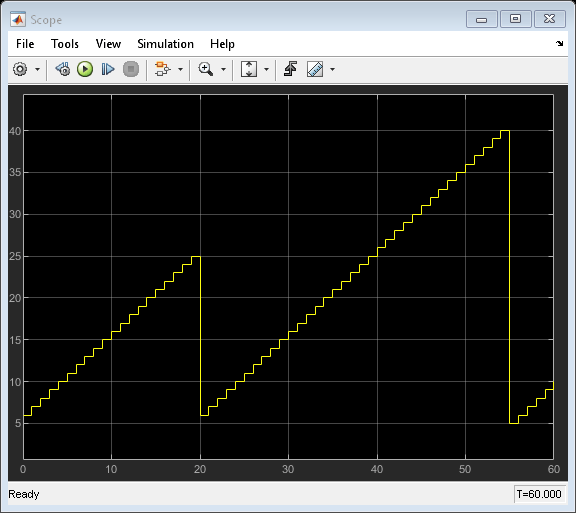
运行集成模型的仿真。

**sim(integrationModel)**

**### Preparing to start SIL block simulation: <a href="matlab: targets\_hyperlink\_manager('run',1);">rtwdemo\_cr ### Starting SIL simulation for component: rtwdemo\_crossrelease\_counter\_R2023a\_sil**

**### Application stopped**

**### Stopping SIL simulation for component: rtwdemo\_crossrelease\_counter\_R2023a\_sil**



#### 调整参数

跨版本 SIL 模块的源模型引用两个可调参数，这些参数由基础工作区中的 Simulink.Parameter 对象控制。使用这些参数修改 SIL 仿真的行为。

**countUpper.Value = 30;**

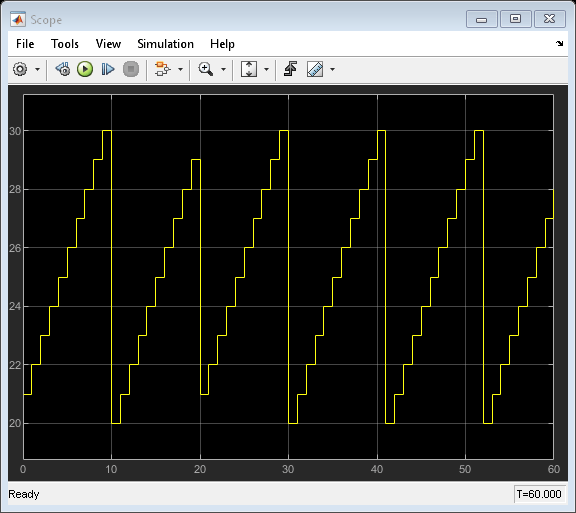
**countLower.Value = 20;**

**yout\_retuned = sim(integrationModel, 'ReturnWorkspaceOutputs', 'on');**

**### Preparing to start SIL block simulation: <a href="matlab: targets\_hyperlink\_manager('run',1);">rtwdemo\_cr ### Starting SIL simulation for component: rtwdemo\_crossrelease\_counter\_R2023a\_sil**

**### Application stopped**

**### Stopping SIL simulation for component: rtwdemo\_crossrelease\_counter\_R2023a\_sil**



#### 在集成模型中配置信号的存储类

配置：

* 集成模型中的信号名称，以匹配在导入代码中使用的名称。
* 补充存储类。

在本示例中，**ticks** 输入和 **count** 输出是通过导入的代码中的 **ImportedExtern** 存储类来实现的。如果连接到集成模型中跨版本模块的输入和输出端口的信号的存储类为 **ExportedGlobal**，则集成模型必须提供变量的定义。

如果信号名称不匹配，则集成模型将生成附加代码，以在由集成模型实现的信号和由导入的代码实现的信号之间复制数据。

**hLines = get\_param(dstBlock, 'LineHandles');**

**% Use the code mappings API to configure the storage class on the signals. cm = coder.mapping.api.get(integrationModel);**

**ticksPortHandle = get(hLines.Inport(1), 'SrcPortHandle'); set(ticksPortHandle, 'Name', 'ticks');**

**addSignal(cm, ticksPortHandle);**

**setSignal(cm, ticksPortHandle, 'StorageClass', 'ExportedGlobal');**

**countPortHandle = get(hLines.Outport(1), 'SrcPortHandle'); set(countPortHandle, 'Name', 'count');**

**addSignal(cm, countPortHandle);**

**setSignal(cm, countPortHandle, 'StorageClass', 'ExportedGlobal');**

#### 在集成模型中配置参数和数据存储内存的存储类

参数和数据存储是通过导入的代码中的 **ImportedExtern** 存储类实现的。如果您将参数和数据存储配置为使用 **ExportedGlobal** 存储类，则集成模型必须提供变量的定义。

**resetSignal.CoderInfo.StorageClass = 'ExportedGlobal'; countLower.CoderInfo.StorageClass = 'ExportedGlobal'; countUpper.CoderInfo.StorageClass = 'ExportedGlobal';**

#### 从集成模型中生成代码

在所需的共享代码保存到共享代码存储库中以后，请删除以前生成的共享实用工具文件夹。

**if isfolder(RTW.getBuildDir(integrationModel).SharedUtilsTgtDir) rmdir(RTW.getBuildDir(integrationModel).SharedUtilsTgtDir, 's');**

**end**

删除不影响代码生成的范围和连接线。

**scopeBlock = [integrationModel, '/', 'Scope']; hScopeLines = get\_param(scopeBlock, 'LineHandles'); hScopeLine = hScopeLines.Inport(1);**

**assert(strcmp(get(hScopeLine, 'SegmentType'), 'branch')); delete\_line(hScopeLine);**

**delete\_block(scopeBlock);**

生成代码。

**slbuild(integrationModel);**

**### Starting build procedure for: rtwdemo\_crossrelease\_integration**

**### Successful completion of code generation for: rtwdemo\_crossrelease\_integration Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_crossrelease\_integration Code generated. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 14.873s**

#### 检查对导入代码的调用

要检查集成模型代码中的跨版本模块代码，请使用 **rtwtrace** 实用工具。

**rtwtrace(dstBlock);**

#### 测试从集成模型生成的代码

运行集成模型的顶层模型软件在环 (SIL) 仿真。仿真运行从集成模型生成的代码，这会调用导入的代码。将仿真输出记录在工作区中。

**set\_param(integrationModel, 'SimulationMode', 'software-in-the-loop (sil)'); yout\_SIL = sim(integrationModel, 'ReturnWorkspaceOutputs', 'on'); plot(yout\_SIL.yout{1}.Values);**

**### Starting build procedure for: rtwdemo\_crossrelease\_integration**

**### Successful completion of build procedure for: rtwdemo\_crossrelease\_integration Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_crossrelease\_integration Code generated and compiled. S-function rtwdemo\_crossrelease\_counter\_R**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 10.972s**

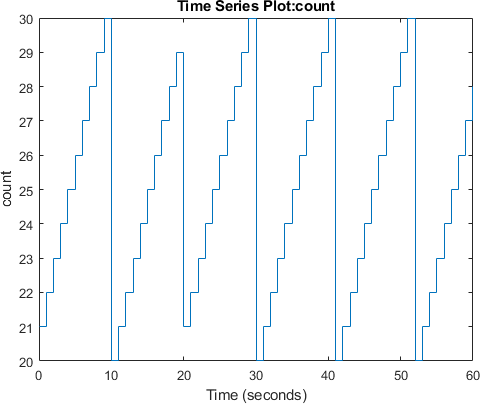
**### Preparing to start SIL simulation ... Building with 'gcc'.**

**MEX completed successfully.**

**### Updating code generation report with SIL files ...**

**### Starting SIL simulation for component: rtwdemo\_crossrelease\_integration ### Application stopped**

**### Stopping SIL simulation for component: rtwdemo\_crossrelease\_integration**



#### 比较仿真输出

比较以下两种仿真输出：在 SIL 模式下只运行导入代码的仿真输出，以及在 SIL 模式下作为顶层模型运行的集成模型的仿真输出。

**max(abs(yout\_SIL.yout{1}.Values.Data - yout\_retuned.yout{1}.Values.Data))**

**ans = uint8 0**

## 在工程中生成代码并进行模型仿真

此示例说明如何使用代码生成模板创建新的工程。Simulink 工程的代码生成工程模板包含多个模型。工程模板还提供可帮助您生成控制器代码并运行框架模型仿真的实用工具（.m 脚本）。

#### 创建工程

从代码生成工程模板中创建新的工程。

要从 Simulink➅ 起始页创建此工程，请在命令行窗口中键入：

#### simulink

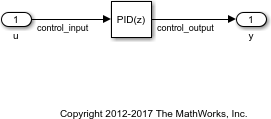
从起始页选择“代码生成”模板，然后创建 **exampleCodeGen** 工程。

#### 生成代码

为 **feedback\_control.slx** 模型生成控制器代码。

要生成控制器代码，请选择**工程快捷方式**选项卡，然后选择**生成控制器代码**快捷方式。

此快捷方式将运行工程文件夹 **utilities** 中的 **generate\_controller\_code.m** 脚本。此脚本将在工程文件夹 **controller** 中编译 **feedback\_control.slx** 模型。

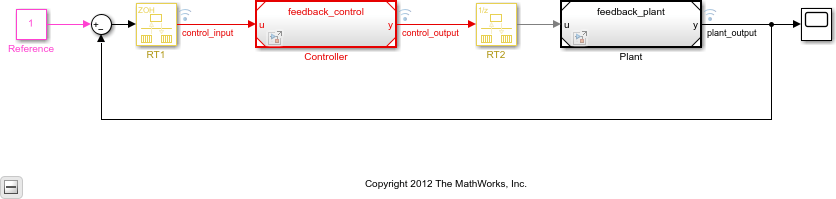


#### 对模型进行仿真

仿真顶层框架 **feedback\_harness.slx** 模型。

要打开框架模型进行仿真，请选择**工程快捷方式**选项卡，然后选择**反馈框架**快捷方式。此快捷方式将打开工程文件夹 **harnesses** 中的 **feedback\_harness.slx** 模型。

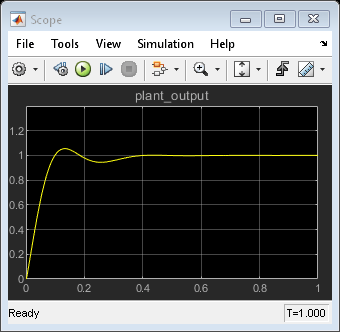
要仿真该模型，请点击**运行**。



#### 查看仿真输出

打开模型中的 **Scope** 模块，并查看仿真输出。

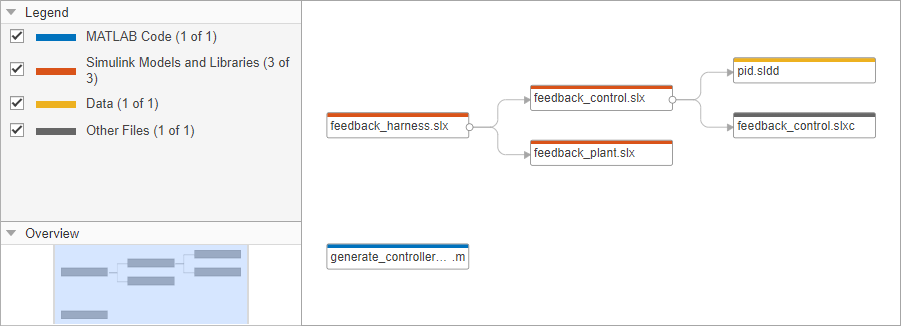
要打开示波器并查看仿真，请在 Simulink 窗口中双击 **Scope** 模块。



#### 使用依存关系分析器视图

要查看文件依存关系，请使用工程的**依存关系分析器**视图。从**依存关系分析器**视图中，您可以：

* 查看文件依赖项。
* 将鼠标悬停在依存关系箭头上，检查依存关系类型。
* 双击一个文件或模型将其打开。



#### 更多信息

* “Work with Project Files”

#### Simulink.createFromTemplate

* “Automate Startup Tasks”
* “Automate Shutdown Tasks”

#### 相关示例

* “从文件夹创建新工程”
* “从模型创建工程”

## 使用工程以编程方式生成代码和仿真模型

以下示例说明如何使用工程 API 创建新工程，并自动执行工程任务来操作文件。它讲述如何以编程方式创建空白工程、添加文件和文件夹、设置工程路径、定义工程快捷方式、生成代码以及对框架模型进行仿 真。

通过工程，您可以管理和共享文件与设置、查找必需文件以及与源代码管理进行交互，从而有助于您组织大型工程设计。

#### 管理基于模型的设计工程

示例模型提供了一个控制器，它将相机固定到跟踪绿色球的直流电机上。 MBD 方法旨在对控制器操作进行建模：

* 被控对象建模：被控对象是一台直流电机，它通过旋转重新定位相机。
* 控制器建模：该设计使用具有角位置反馈的 PID 控制器来定位直流电机。
* 高级球跟踪调度器建模：调度器检查绿色球位置的有效性和基准角的饱和度。调度器处理屏幕上无球可跟踪的实例。

该设计和工程包含以下几个文件夹。

**被控对象** - 被控对象提供：

* 包含使用基本 Simulink➅ 模块的连续时间动态系统的模型。
* 包含使用 LTI 系统模块的线性时不变 (LTI) 系统的模型。
* 在工作区中导入和导出仿真数据
* 将参数加载到使用数据字典和模型回调的基础工作区。

**控制器** - 控制器提供：

* 包含离散时间动态系统并使用基本 Simulink 模块的模型。
* 包含使用 PID Controller 模块的离散时间动态系统的模型。
* 强制执行使用 PWM Conversion 子系统的信号饱和。

**球跟踪** - 球跟踪提供：

* 基本球跟踪算法。
* 作为占位符的 Terminator 和 Ground 模块。
* 使用 Stateflow➅ 生成电机基准角的混合控制器。

**框架** - 框架提供整个系统的顶层仿真。

#### 创建空白工程并添加文件

1. 创建一个空白工程。在**主页**选项卡上，点击**新建** > **工程** > **空白工程**。

或者，从命令行窗口创建此工程并使用 currentProject 获取工程对象来操作工程：

**matlab.project.createProject('modelBasedDesignCodeGenProject'); proj\_modelBasedDesignCodeGen = currentProject**

**proj\_modelBasedDesignCodeGen = Project with properties:**

**Name: "modelBasedDesignCodeGenProject" SourceControlIntegration: ""**

**RepositoryLocation: "" SourceControlMessages: [1×0 string]**

**ReadOnly: 0**

**TopLevel: 1**

**Dependencies: [1×1 digraph]**

**Categories: [1×1 matlab.project.Category]**

**Files: [1×0 matlab.project.ProjectFile]**

**Shortcuts: [1×0 matlab.project.Shortcut]**

**ProjectPath: [1×0 matlab.project.PathFolder]**

**ProjectReferences: [1×0 matlab.project.ProjectReference]**

**StartupFiles: [1×0 string]**

**ShutdownFiles: [1×0 string] DefinitionFilesType: FixedPathMultiFile**

**Description: ""**

**RootFolder: "C:\Users\ktripp\MATLAB\Projects\modelBasedDesignCodeGenProject1" DependencyCacheFile: ""**

**SimulinkCodeGenFolder: ""**

**ProjectStartupFolder: "C:\Users\ktripp\MATLAB\Projects\modelBasedDesignCodeGenProject1" SimulinkCacheFolder: ""**

1. 将示例文件复制到工程内的文件夹中。或者，在命令行窗口中，键入：

**copyfile(fullfile(matlabroot,'toolbox','rtw','rtwdemos','slproject\_codegen\_demo'), ... fullfile(pwd),'f');**

1. 将现有文件添加到工程中：
   * 点击**工程文件视图**按钮并选择**所有文件视图**。
   * 选择文件夹 **ball\_tracking**、**controller**、**harnesses**、**plant** 和 **utilities**，右键点击其中一个文件夹的图标，然后选择**将文件夹添加到工程(包括子文件)**。

或者，从命令行窗口以编程方式将这些文件添加到工程中：

**addFolderIncludingChildFiles(proj\_modelBasedDesignCodeGen,'ball\_tracking'); addPath(proj\_modelBasedDesignCodeGen,'ball\_tracking'); addFolderIncludingChildFiles(proj\_modelBasedDesignCodeGen,'controller'); addPath(proj\_modelBasedDesignCodeGen,'controller'); addFolderIncludingChildFiles(proj\_modelBasedDesignCodeGen,'harnesses'); addPath(proj\_modelBasedDesignCodeGen,'harnesses'); addFolderIncludingChildFiles(proj\_modelBasedDesignCodeGen,'plant'); addPath(proj\_modelBasedDesignCodeGen,'plant'); addFolderIncludingChildFiles(proj\_modelBasedDesignCodeGen,'utilities'); addPath(proj\_modelBasedDesignCodeGen,'utilities');**

1. 选择 **work** 文件夹，右键点击其中一个文件夹的图标，然后选择**工程路径** > **添加到工程路径(包括子文件夹)**。如果选择将此文件夹添加到该路径中，则可以在工程路径中找到仿真和代码生成时生成的文件（工件）。如果选择不在工程中包含生成的文件，则依存关系图不显示这些文件。

或者，从命令行窗口将此文件夹添加到工程路径中：

**addPath(proj\_modelBasedDesignCodeGen,'work');**

1. 要使工程更容易访问，请定义工程快捷方式并为其设置属性。在**文件**视图中，选择**工程**选项卡。打开

**utilities** 文件夹节点。

1. 选择文件 **clean\_up\_project.m**、**generate\_controller\_code.m** 和 **set\_up\_project.m**，右键点击其中一个文件的图标，然后选择**创建快捷方式**。
2. 要在打开工程时设置当前工作文件夹、设置仿真缓存文件夹并运行其他设置脚本：
   * 选择 **set\_up\_project.m** 脚本。
   * 右键点击该文件。
   * 选择**启动时运行**。

或者，从命令行窗口将此设置脚本添加到工程启动任务中：

**addStartupFile(proj\_modelBasedDesignCodeGen,fullfile(pwd,'/utilities/set\_up\_project.m'));**

1. 要在关闭工程时还原以前的设置并运行其他清理脚本：
   * 选择 **clean\_up\_project.m** 脚本。
   * 右键点击该文件。
   * 选择**关闭时运行**。

或者，从命令行窗口以编程方式将清理脚本添加到工程关闭任务中：

**addShutdownFile(proj\_modelBasedDesignCodeGen,fullfile(pwd,'/utilities/clean\_up\_project.m'));**

1. 重新打开工程以运行**启动**快捷方式。

或者，重新加载工程并从命令行窗口运行 **set\_up\_project**：

**reload(proj\_modelBasedDesignCodeGen); run('set\_up\_project');**

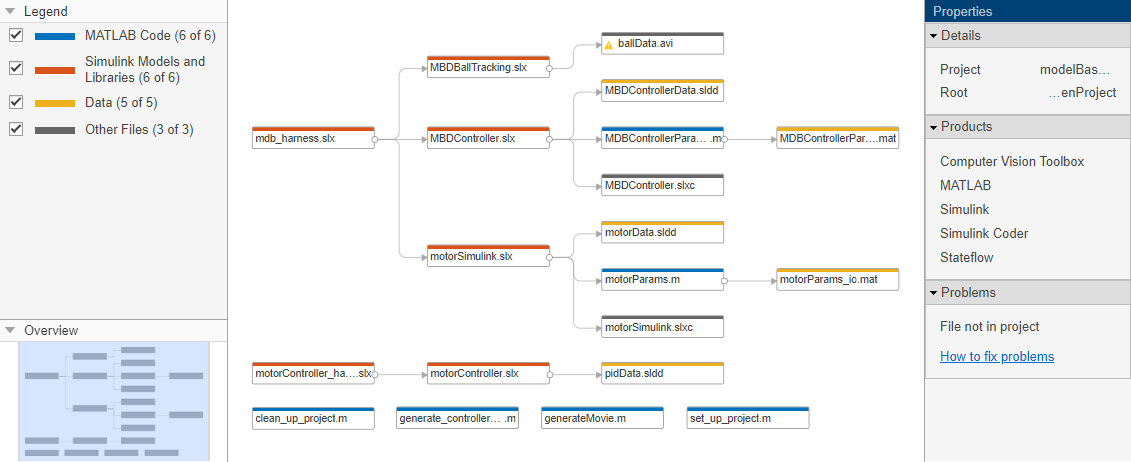
#### 从依存关系图中生成控制器的代码

1. 在**工程**选项卡上，点击**依存关系分析器**以对工程中的所有文件运行依存关系分析。

依存关系图显示您的工程结构和文件依存关系。它显示文件之间的相互关系。图中的每个项目表示一个文件，每个箭头表示一种依存关系。

**work** 文件夹的内容不会显示在依存关系图中。此设置简化了视图，以指示模型文件和 **.m** 脚本文件中的依存关系。

* 1. 检查右侧的**属性**窗格，了解有关您的工程的有用信息，如必需产品和问题文件。
  2. 使用**放大**按钮放大依赖项对应图。使用箭头键或者使用鼠标滚轮点击并拖动，在依赖项对应图上进行移动。
  3. 将鼠标悬停在依存关系箭头上，检查依存关系类型。例如，MATLAB➅ 脚本和函数正在由模型回调运行。



1. 在依存关系图上选择 **MBDController.slx** 模型。右键点击该模型并选择**打开**。
2. 当模型在 Simulink 中打开后，点击**编译**按钮为该模型生成代码。或者，使用工程中的

**generate\_controller\_code** 快捷方式生成代码。

或者，从命令行窗口为模型生成代码。为了演示此示例，请将写入权限设置为 **MBDController.slxc**，然后为模型生成代码：

**fileattrib('MBDController.slxc', '+w'); evalc('slbuild(''MBDController'')');**

1. 从工程窗口中关闭工程。

或者，关闭此工程并从命令行窗口运行 **clean\_up\_project**：

**run('clean\_up\_project'); close(proj\_modelBasedDesignCodeGen);**

**另请参阅**

**currentProject**, **openProject**

**更多信息**

* “Work with Project Files”

#### Simulink.createFromTemplate

* “Automate Startup Tasks”
* “Automate Shutdown Tasks”

#### 相关示例

* “从文件夹创建新工程”
* “从模型创建工程”

# Simulink Coder 中的报告生成

# Simulink Coder 中的代码替换

* [“什么是代码替换？” （第 27-2 页）](#_bookmark203)
* [“替换从 Simulink 模型生成的代码” （第 27-5 页）](#_bookmark207)

## 什么是代码替换？

代码替换是指更改代码生成器为函数和运算符生成的代码以满足应用程序代码要求的一种方法。例如，您可以替换生成的代码以满足以下要求：

* 针对特定运行时环境的优化，包括（但不限于）特定目标硬件。
* 与现有应用程序代码集成。
* 符合某种标准，例如 AUTOSAR。
* 修改代码行为，例如启用或禁用非有限或内联支持。
* 特定于应用程序或工程的代码要求，例如：
  + 消除 **math.h**。
  + 消除系统头文件。
  + 消除对 **memcpy** 或 **memset** 的调用。
  + 使用 BLAS。
  + 使用特定的 BLAS。

要应用此方法，请将代码生成器配置为在代码生成期间应用代码替换库 (CRL)。默认情况下，代码生成器不应用代码替换库。您可以从 MathWorks 提供的库以及您使用 Embedded Coder 产品创建和注册的库中进行选择。可用库的列表取决于：

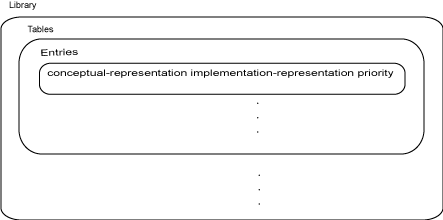
* 已安装的支持包。
* 系统目标文件、语言、语言标准和设备供应商配置。
* 您是否使用 Embedded Coder 产品创建并注册了库。

包含 GNU99 扩展的库旨在与 GCC 编译器结合使用。如果您将其中一个库与其他编译器结合使用，生成的代码可能无法编译。

### 代码替换库

代码替换库由一个或多个代码替换表组成，这些表指定函数和运算符的特定于应用程序的实现。例如，特定嵌入式处理器的库指定针对该处理器优化所生成代码的函数和运算符替换。

代码替换表包含一个或多个代码替换条目，每个条目表示一个函数或运算符的潜在替换。每个条目将函数或运算符的概念表示映射到实现表示和优先级。



|  |  |
| --- | --- |
| **表条目组成部分** | **说明** |
| 概念表示 | 标识表条目并包含代码生成器的匹配条件。包括：   * 函数名称或键。函数名称标识大多数函数。对于运算符和某些函数来说，使用称为键的一系列字符来标识函数或运算符。例如，函数名称 **'cos'** 和运算符键 **'RTW\_OP\_ADD'**。 * 遵守代码生成器命名的概念参数（**'y1'**、**'u1'**、**'u2'**...），具有对应的 I/O 类型   （输出或输入）和数据类型。   * 其他属性（如算法、定点饱和与舍入模式），用于标识函数或运算符的匹配条件。 |
| 实现表示 | 指定替换代码。包括：   * 函数名称。例如，**'cos\_dbl'** 或 **'u8\_add\_u8\_u8'**。 * 实现参数，具有对应的 I/O 类型（输出或输入）和数据类型。 * 提供附加实现细节的参数，如头文件和源文件名以及编译资源的路径。 |
| 优先级 | 定义相对于表中其他条目的条目优先级。该值的范围可以是从 0 到 100，0 表示最高优先级。如果多个条目具有相同的优先级，代码生成器将使用具有该优先级的第一个匹配项。 |

当代码生成器在代码替换库中查找匹配项时，它会创建一个调用位置对象并使用函数或运算符概念表示填充该对象。如果存在匹配项，代码生成器将使用用实现表示填充的匹配代码替换条目，并使用它来生成代码。

代码生成器按照各表在库中出现的顺序在代码替换库中搜索匹配项。如果代码生成器在一个表中找到多个匹配项，将根据优先级确定匹配项。代码生成器使用优先级较高的条目，而不是优先级较低的类似条目。

### 代码替换术语

|  |  |
| --- | --- |
| **术语** | **定义** |
| 缓存命中 | 对于在指定的代码替换库中定义的函数或运算符的代码替换条目，代码生成器找到与之匹配的条目。 |
| 缓存未命中 | 代码生成器找不到与函数或运算符的概念表示匹配的条目。 |
| 调用位置对象 | 代码生成器在遇到函数或运算符的调用位置时使用的函数或运算符的概念表示。代码生成器使用该对象来查询代码替换库，以获得概念表示匹配项。如果存在匹配项，代码生成器将返回一个代码替换对象，该对象由概念表示、实现表示和优先级完全填充，并使用该对象生成替换代码。 |
| 代码替换库 | 一个或多个代码替换表，这些表指定函数和运算符的特定于应用程序的实现。当配置为使用代码替换库时，代码生成器使用库中定义的条件来搜索匹配项。如果找到匹配项，代码生成器会将默认情况下生成的代码替换为库中定义的特定于应用程序的代码。 |
| 代码替换表 | 一个或多个代码替换表条目。提供一种将相关或共享的条目分组的方法，以便在不同库中使用。 |
| 代码替换条目 | 表示函数或运算符的潜在替换。将函数或运算符的概念表示映射到实现表示和优先级。 |

|  |  |
| --- | --- |
| **术语** | **定义** |
| 概念参数 | 表示被替换的函数或运算符的输入或输出参数。概念参数遵循命名约定（**'y1'**、**'u1'**、**'u2'**...）和代码生成器熟悉的数据类 型。 |
| 概念表示 | 表示代码生成器用于限定要替换的函数和运算符的匹配条件。包括：   * 函数或运算符名称或键 * 具有输入和输出的类型、维度和复/实性设定的概念参数 * 属性，如算法和定点饱和以及舍入模式 |
| 实现参数 | 表示 C 或 C++ 替换函数的输入或输出参数。实现参数遵循 C/C++ 名称和数据类型设定。 |
| 实现表示 | 指定 C 或 C++ 替换函数原型。包括：   * 函数名称（例如，**'cos\_dbl'** 或 **'u8\_add\_u8\_u8'**） * 指定函数输入和输出的类型、类型限定符和复/实性的实现参数 * 提供编译信息的参数，如头文件和源文件名、编译资源的路径以及编译和链接标志 |
| 键 | 标识将被替换的函数或运算符。函数名称或键出现在代码替换条目的概念表示中。**RTW\_OP\_ADD** 键表示加法运算符。 |
| 优先级 | 定义代码替换条目相对于代码替换库中具有相同名称和概念参数列表的其他条目的匹配优先级。优先级的范围可以是从 0 到 100，0 表示最高优先级。默认值为 100。如果库为一个函数或运算符提供两个实现，则优先级较高的实现会遮蔽优先级较低的实现。 |

### 代码替换限制

代码替换验证 - 代码替换的行为可能与您预期的不同。例如，您在代码生成器输入中观测到的数据类型可能与代码生成器在运算期间用作中间数据类型的数据类型不匹配。通过检查生成的代码来验证代码替换。

矩阵的代码替换 - 代码替换库不支持动态和符号大小的矩阵。

### 另请参阅相关示例

* “Choose a Code Replacement Library”
* [“替换从 Simulink 模型生成的代码” （第 27-5 页）](#_bookmark207)

## 替换从 Simulink 模型生成的代码

此示例说明如何使用代码替换库替换生成的代码。代码替换是指您可以用来更改代码生成器为函数和运算符生成的代码以满足应用程序代码要求的一种方法。例如，您可以替换生成的代码以满足以下要求：

* 针对特定运行时环境的优化，包括（但不限于）特定目标硬件。
* 与现有应用程序代码集成。
* 符合某种标准，例如 AUTOSAR。
* 修改代码行为，例如启用或禁用非有限或内联支持。
* 应用程序或工程特定的代码要求，例如去除 **math.h** 和系统头文件，或调用 **memcpy** 或 **memset**，或者使用 BLAS。

#### 准备代码替换

1. 确保在您的系统上安装了 MATLAB➅、Simulink➅、Simulink Coder™ 和 C 编译器。您的开发环境中某些代码替换库可能需要 Embedded Coder➅。

要安装 MathWorks➅ 产品，请参阅 MATLAB 安装文档。如果您已安装 MATLAB 并想查看还安装了哪些其他 MathWorks 产品，请在命令行窗口中输入 **ver**。

1. 标识一个现有 Simulink 模型，或创建一个您要代码生成器替换其代码的模型。

#### 选择代码替换库

默认情况下，代码生成器不应用代码替换库。您可以从 MathWorks➅ 提供的库以及使用 Embedded Coder➅ 产品创建和注册的库中进行选择。可用库的列表取决于：

* + 已安装的支持包。
  + 系统目标文件、语言、标准数学库和设备供应商配置。
  + 您是否使用 Embedded Coder➅ 产品创建并注册了库。

包含 GNU99 扩展的库旨在与 GCC 编译器结合使用。如果您将其中一个库与其他编译器结合使用，生成的代码可能无法编译。

根据您拥有的产品许可证，可能还有其他可用的库。如果您有 Embedded Coder 许可证，您可以查看和选择其他库，并且可以创建自定义代码替换库。

#### 配置代码生成器以使用代码替换库

1.配置代码生成器，以便在模型的代码生成过程中应用代码替换库。执行以下操作之一：

* + 在 Configuration Parameters 对话框的 **Code Generation > Interface** 窗格中，为 Code replacement library 参数选择一个库。
  + 在命令行中或以编程方式设置 **CodeReplacementLibrary** 参数。

2.将代码生成器配置为仅生成代码（不编译可执行程序），以便您可以在编译可执行程序之前验证代码替换。执行以下操作之一：

* + 在 Configuration Parameters 对话框的 **Code Generation** 窗格中，选择 Generate code only。
  + 在命令行中或以编程方式设置 **GenCodeOnly** 参数。

#### 在代码生成报告中包含代码替换信息

如果您有 Embedded Coder 许可证，您可以配置代码生成器，以便在代码生成报告中包含代码替换部分。这些附加信息可以帮助您验证代码替换。

1. 配置代码生成器以生成报告。在 Configuration Parameters 对话框的 **Code Generation > Report** 窗格中，选择 Create code generation report。考虑自动打开报告。选择自动打开报告。
2. 在报告中包含代码替换部分。选择 Summarize which blocks triggered code replacements (Embedded Coder)。

#### 生成替换代码

从模型生成 C/C++ 代码，如果相应地配置了代码生成器，还会生成代码生成报告。例如，在模型窗口中，按 **Ctrl+B**。

代码生成器生成代码并显示报告。

#### 验证代码替换。

通过检查生成的代码来验证代码替换。代码替换的行为可能与您预期的不同。例如，您在代码生成器输入中观测到的数据类型可能与代码生成器在运算期间用作中间数据类型的数据类型不匹配。

#### 有关详细信息

* [“什么是代码替换？” （第 27-2 页）](#_bookmark203)
* “Choose a Code Replacement Library”
* “Code Generation Configuration”

# 部署

# Simulink Coder 中的桌面

## 将 S-Function 目标用于模型或子系统

S-Function 是代码生成器可为其生成代码的一类系统目标文件。S-Function 目标可以封装子系统来提高其执行效率，促进代码重用，并保护其知识产权。

**注意** 虽然您可以使用 S-Function 目标以在部署应用组件用于重用的同时保护其内部逻辑免遭窥探和修改，但在分布式组件中，保护知识产权的首选解决方案是：

* [受保护模型，即隐藏模块和信号线信息的引用模型。有关详细信息，请参阅“保护模型以隐藏内容”](#_bookmark66)

[（第 11-2](#_bookmark66) 页）。

* 共享库系统目标文件，用于为模型或子系统生成共享库，以便在 Simulink 之外的系统仿真中使用。有关详细信息，请参阅“将生成的代码打包为共享库” (Embedded Coder)。

这些解决办法还可以提高执行效率，促进代码重用。

您可以将生成的 S-Function 目标与 Generated S-Function 模块结合使用。

S-Function 目标使用的 **CodeFormat** TLC 变量的值为 **'S-Function'** 时，会生成符合 Simulink C MEX S-Function 应用程序编程接口 (API) 规范的代码。

### S-Function 部署所需的文件

部署生成的 S-Function 模块用于仿真和用于代码生成所需要的文件是不同的。

要部署生成的 S-Function 模块以包含在其他模型中用于仿真，只需提供创建 S-Function 模块时在当前工作文件夹中生成的二进制 MEX 文件对象。所需的文件为：

#### subsys\_sf.mexext

其中 **subsys** 是子系统名称，**mexext** 是因平台而异的 MEX 文件扩展名（请参阅 **mexext**）。例如， **SourceSubsys\_sf.mexw64**。

要部署生成的 S-Function 模块以包含在其他模型中用于代码生成，请提供创建 S-Function 模块时在当前工作文件夹中生成的文件。所需的文件包括：

* **subsys\_sf.c** 或 **.cpp**，其中 **subsys** 是子系统名称（例如，**SourceSubsys\_sf.c**）

#### subsys\_sf.h

* **subsys\_sf.mexext**，其中 **mexext** 是因平台而异的 MEX 文件扩展名（请参阅 **mexext**）
* 子文件夹 **subsys\_sfcn\_rtw** 及其内容

本软件提供了 **rtwsfcn.tlc** 系统目标文件供 S-Function 目标使用。

生成的 S-Function 代码使用与创建函数时所在的主机系统匹配的**配置参数** > **硬件实现**参数值。当您使用模型中的 S-Function 来进行代码生成时，请确保模型的这些参数值与 S-Function 的参数值相匹配。

### 生成的 S-Function 中的采样时间传播

如果满足一定的条件，生成的 S-Function 模块可以从其所在的模型中继承采样时间。“引用模型的采样时间” 和“S-Functions That Specify Sample Time Inheritance Rules”介绍了决定 Model 模块和生成的 S-Function 模块的采样时间传播的条件。

要生成满足继承采样时间条件的 S-Function 模块，必须对从中生成 S-Function 模块的模型的求解器加以限制。将模型配置参数**类型**设置为“定步长”并将**周期性采样时间约束**设置“确保采样时间独立”。如果模型不能继承采样时间，此设置会导致 Simulink 软件在编译模型时显示错误消息。有关此选项的详细信息，请参阅“周期性采样时间约束”。

### 具有生成的 S-Function 的顶层模型的求解器类型

下表显示顶层模型求解器类型（这些类型与模型是具有离散还是连续采样时间有关）与生成的 S-Function的求解器类型的可能组合。

#### 顶层模型求解器选项和采样时间

|  |  |  |
| --- | --- | --- |
|  | **模型配置参数：顶层模型配置** | |
| **采样时间** | **求解器选项，类型：变步长** | **求解器选项，类型：定步长** |
| 离散 | 生成的 S-Function 需要变步长求解器 | 生成的 S-Function 可以具有变步长求解器或定步长求解器 |
| 连续 | 生成的 S-Function 需要变步长求解器 | 生成的 S-Function 需要定步长求解器 |

从子系统生成的 S-Function 的参数硬编码到模块中。Simulink 在生成模块时（而不是在仿真运行时）计算采样时间等参数。验证生成的 S-Function 模块是否能够在目标模型中按预期工作很重要。

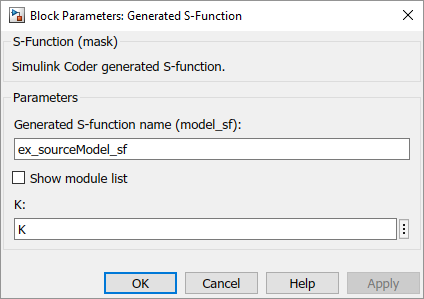
### 生成的 S-Function 中的可调参数

要在生成的 S-Function 中使用可调参数，请使用模型配置参数来声明所需的模块参数可调。请参阅 “Declare Workspace Variables as Tunable Parameters Using the Model Parameter Configuration Dialog Box”。

您在源模型中使用“自动”存储类声明为可调的模块参数将成为生成的 S-Function 的可调参数。这些参数不会像从其他系统目标文件生成的代码中那样，成为生成的 **model\_P**（以前称为 **rtP**）参数数据结构体的一部分。此时，生成的代码将使用 MEX API 调用（如 **mxGetPr** 或 **mxGetData**）来访问这些参数。您的代码应该以相同的方式访问这些参数。

有关 MEX API 调用的详细信息，请参阅“About C MEX S-Functions” 和“将 MATLAB 与外部编程语言和系统集成”。

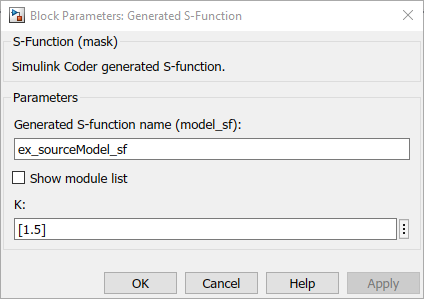
使用 S-Function 目标创建的 S-Function 模块将自动封装。封装将每个可调参数显示在一个编辑字段中。默认情况下，编辑字段按变量名称显示参数，如下图所示。



通过选择模型配置参数**使用可调参数的值**，您可以选择显示参数值，而不是其变量名称。



选择此参数后，编辑字段中将显示变量的值（在代码生成时）。



### 宏参数

假设您将存储类（如 **Define**）应用于 **Simulink.Parameter** 对象，以使该参数在生成的代码中显示为宏。如果在生成 ERT S-Function 的子系统中使用参数对象，则无法选择该参数对象作为可调参数。此 时，S-Function 代码生成器将自定义存储类应用于参数对象。S-Function 代码中的这种宏生成使您能够从包含变体元素（如 Variant Subsystem 模块）的子系统中生成 S-Function，您可以配置这些变体元素以在生成的代码中产生预处理器条件句。在仿真 S-Function 期间，无法更改该参数的值。

要选择该参数对象作为可调参数，请应用不同存储类或创建您自己的存储类。将参数视为宏的存储类包括 **Define**、**ImportedDefine**、**CompilerFlag** 以及通过在自定义存储类设计器中将**数据初始化**设置为 “宏”而创建的存储类。如果使用非宏存储类，则无法将参数对象用作变体控制项变量并生成预处理器条件句。

如果您应用的存储类将参数对象视为导入的宏，则在生成 ERT S-Function 之前，请提供宏定义。例如，假设您将存储类 **ImportedDefine** 应用于 **Simulink.Parameter** 对象，并将参数对象用作子系统中的变体控制项变量。如果将自定义属性 **HeaderFile** 设置为 **'myHdr.h'**，则在生成 S-Function 时，请将自定义头文件 **myHdr.h** 放在当前文件夹中。生成的 S-Function 使用头文件中的宏值，而不是参数对象的 **Value** 属性中的值。

要使用通过编译器选项定义的宏（例如通过应用存储类 **CompilerFlag**），请使用模型配置参数**代码生成**

> **自定义代码** > **代码信息** > **定义**来指定编译器选项。有关详细信息，请参阅“代码生成”窗格：自定义代码：附加编译信息：定义。

### 校验和与 S-Function 目标

代码生成器为模型创建一个校验和，并在编译过程中将校验和用于代码重用、模型引用和外部模式功能。代码生成器通过以下方式计算模型校验和

1. 计算模型中每个子系统的校验和。子系统的校验和是子系统模块的属性（数据类型、复/实性、采样时间、端口维度等）的合并。
2. 将子系统校验和与其他模型级信息合并。

S-Function 可以通过调用函数 **ssSetChecksumVal**，将模块属性分析时未捕获的其他信息添加到校验和中。对于 S-Function 目标来说，添加到校验和中的值是生成 S-Function 的模型或子系统的校验和。

代码生成器按以下方式应用子系统和模型的校验和：

* 代码重用 - 如果一个模型中的两个子系统具有相同的校验和，则代码生成器只为一个函数生成代码。
* 模型引用 - 如果当前模型校验和与编译模型时的校验和匹配，则编译过程不会重新编译引用模型。
* 外部模式 - 如果当前模型校验和与目标硬件上运行的代码的校验和不匹配，编译过程将产生错误。

### 生成的 S-Function 的兼容性

当您从模型编译 MEX S-Function 时，代码生成器将生成 2 级非内联 S-Function。跨版本使用生成的代码和二进制 MEX 文件（例如，**\*.mexw64**）时存在以下限制：

* 从旧版 MATLAB 软件中生成的 S-Function 目标代码与新版本不兼容。不要使用新版本的 MATLAB 软件重新编译从旧版本中生成的代码。请使用相同版本的 MATLAB 软件为 S-Function 目标生成代码并将代码编译成 MEX 文件。
* 您可以将从 MATLAB 早期版本生成的二进制 S-Function MEX 文件与相同版本或更新版本的软件一起使用，但兼容性注意事项与手写 S-Function 相同。有关详细信息，请参阅“S-Function Compatibility”。
* 代码生成器可从包含生成的 S-Function 的模型中生成代码并编译可执行文件。这项支持要求编译 S- Function 和编译模型时使用的 MATLAB 软件版本相同。您无法将从旧版 MATLAB 软件中生成的 S-

Function MEX 文件纳入某个模型，然后使用新版本软件来编译该模型。

### 有关 S-Function 目标的限制

* [“具有连续采样时间的 Inport 和 Outport 模块” （第 28-6 页）](#_bookmark219)
* [“表达式中的可调变量” （第 28-6 页）](#_bookmark220)
* [“参数调整” （第 28-6 页）](#_bookmark221)
* [“运行时参数和 S-Function 兼容性诊断” （第 28-6 页）](#_bookmark222)
* [“Goto 和 From 模块” （第 28-6 页）](#_bookmark223)
* [“编译和更新 S-Function” （第 28-7 页）](#_bookmark224)
* [“不支持的模块” （第 28-8 页）](#_bookmark225)
* [“代码生成不支持模型工作点” （第 28-8 页）](#_bookmark226)
* [“嵌套 S-Function” （第 28-8 页）](#_bookmark227)
* [“用户定义的数据类型” （第 28-9 页）](#_bookmark228)
* [“右键点击生成 S-Function 目标” （第 28-9 页）](#_bookmark229)
* [“总线输入和输出” （第 28-9 页）](#_bookmark230)
* [“具有函数调用输入和输出信号的子系统” （第 28-9 页）](#_bookmark231)
* [“函数调用子系统建模模式” （第 28-9 页）](#_bookmark232)
* [“数据存储访问” （第 28-10 页）](#_bookmark233)
* [“通过子系统封装指定 Inport 或 Outport 模块参数” （第 28-10 页）](#_bookmark234)
* [“MEX S-Function 包装器” （第 28-10 页）](#_bookmark235)

#### 具有连续采样时间的 Inport 和 Outport 模块

通过右键点击编译方式来生成 S-Function 时不会保留 Inport 和 Outport 模块的连续采样时间。由于采样时间没有保留，因此生成的 S-Function 模块的仿真可能会受到影响。

#### 表达式中的可调变量

在表达式中使用可调变量存在一些限制条件。当代码生成器在生成代码时遇到不受支持的表达式时，会显示警告并在代码中生成等效的数值。有关限制条件的列表，请参阅“Tunable Expression Limitations”。

#### 参数调整

S-Function 模块不支持调整具有以下值或类型的可调参数：

* 复数值。
* 转换为常量的值或数据类型（通过将模型配置参数**优化** > **默认参数行为**设置为“内联”）。
* 非内置的数据类型。
* 与内置类型不等效的浮点数据类型。

#### 运行时参数和 S-Function 兼容性诊断

如果您将模型配置参数**需要升级 S-Function** 设置为“警告”或“错误”，代码生成器会指示您升级从子系统生成的 S-Function。S-Function 系统目标文件没有注册运行时参数。只有内联 S-Function 才支持运行时参数，而生成的 S-Function 支持的功能阻止了它内联，例如，它可能调用或包含其他非内联 S-

Function。

要解决此限制，请将参数**需要升级 S-Function** 设置为“无”。

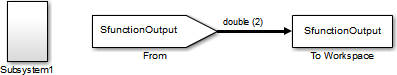
#### Goto 和 From 模块

当使用 S-Function 系统目标文件时，代码生成器将输入和输出限制为对应于根模型的 Inport 和 Outport模块（或生成 S-Function 目标的 Subsystem 模块的 Inport 和 Outport 模块）。代码生成器不会为 Goto 或 From 模块生成代码。

要绕过此限制，请使用所要求的 Inport 和 Outport 模块创建您的模型和子系统，而不要使用 Goto 和 From 模块在根模型和子系统之间传递数据。将生成的 S-Function 融入模型中之后，可以再添加 Goto 和 From 模块。

#### 未绕过限制的示例

* 根模型包括一个 From 模块和子系统 **Subsystem1**



* **Subsystem1** 包括具有全局可见性的 Goto 模块，并将其输入传递给根模型中的 From 模块



* 将 **Subsystem1** 替换为使用 S-Function 系统目标文件生成的 S-Function - 运行模型时生成警告，因为生成的 S-Function 未实现 Goto 模块



#### 绕过限制后的示例

用 Outport 模块替换 **Subsystem1** 中的 GoTo 模块。当您将生成的 S-Function 插入根模型时，其输出将直接连接到 To Workspace 模块。





#### 编译和更新 S-Function

以下限制适用于使用 S-Function 系统目标文件编译和更新 S-Function：

* 不能使用 S-Function 系统目标文件编译包含 Model 模块的模型。这也意味着，如果子系统包含 Model 模块，则无法使用右键点击上下文菜单编译子系统。此限制仅适用于使用 S-Function 目标生成的 S-Function，不适用于 ERT S-Function。
* 您可以使用以下工具编译模型：
  + 工具链 - 仅当工具链支持 MEX 文件生成时。
  + 模板联编文件 - 仅当模板联编文件与支持生成 MEX 文件的工具链关联 [（第 31-9](#_bookmark269) 页）时。
* 如果修改了生成 S-Function 模块的模型，编译过程不会自动重新编译包含生成的 S-Function 模块的模型。与此不同的是，对于 Model 模块引用的模型，根据模型参考**重新编译**配置的设置，在修改后可自动重新进行编译。
* 缺少对应 TLC 文件的手写 S-Function 必须包含无异常代码。有关无异常代码的详细信息，请参阅 “Exception Free Code”。

#### 不支持的模块

S-Function 格式不支持以下内置模块：

* Interpreted MATLAB Function 模块
* 包含以下函数的 S-Function 模块：
  + MATLAB 语言 S-Function（除非您提供 TLC 文件进行 C 代码生成）
  + Fortran S-Function（除非您提供 TLC 文件进行 C 代码生成）
  + 调用 MATLAB 环境的 C/C++ MEX S-Function
* Simulink Function 模块
* Function Caller 模块
* Scope 模块
* To Workspace 模块

S-Function 格式不支持 **embeddedtargetslib** 模块库中的模块。

#### 代码生成不支持模型工作点

您可以编写 C-MEX 和 2 级 MATLAB S-Function，能利用模型工作点保存和还原仿真状态，如 “S- Function Compliance with the ModelOperatingPoint” 中所述。模型工作点不支持代码生成，包括与 S-Function 系统目标文件一起使用的情况。

当您为指定默认工作点合规性或无工作点支持的 S-Function 生成代码时，生成的代码不包括模型工作点功能。您不能为实现自定义工作点功能的 S-Function 生成代码。

#### 嵌套 S-Function

在模型或子系统中嵌套一个生成的 S-Function 模块，再从中生成另一个 S-Function 时，适用以下限制：

* 本软件不支持对嵌套的 S-Function 使用非虚拟总线输入和输出。
* 避免将 S-Function 嵌套在与其同名（可能相隔几个层级）的模型或子系统中。否则，可能会对 S- Function 进行递归调用。本软件当前不会检测这种 S-Function 循环依赖关系，它可能会导致您的 MATLAB 会话终止或挂起。为了防止这种情况发生，请为子系统或模型指定的名称生成具有唯一性的 S-Function 目标，以防与 MATLAB 路径中的现有 MEX 文件重名。

#### 用户定义的数据类型

S-Function 系统目标文件不支持可在用户定义的数据类型上指定的 **HeaderFile** 属性，包括基于 **Simulink.AliasType**、**Simulink.Bus** 和 **Simulink.NumericType** 对象的属性。如果您的模型中用户定义的数据类型使用 **HeaderFile** 属性指定关联的头文件，则使用 S-Function 系统目标文件生成代码时将忽略该值，并且不会生成对应的 include 语句。

#### 右键点击生成 S-Function 目标

如果您通过右键点击 Function-Call Subsystem 模块来生成 S-Function 目标，则原始子系统和生成的 S- Function 可能不一致。当 Function-Call Subsystem 模块内的 Trigger Port 模块的**启用时的状态**参数设置为**继承**时，就会发生不一致。必须将**启用时的状态**参数设置为**重置**或**保持**，否则 Simulink 将报告错误。

#### 总线输入和输出

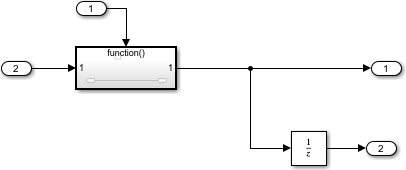
如果使用 S-Function 目标生成的 S-Function 具有总线输入或输出信号，则生成的总线数据结构体可能包含补白，以使总线元素的字段与仿真期间使用的 Simulink 表示形式对齐。但是，如果在模型中插入 S- Function 并使用模型目标（例如 **grt.tlc**）生成代码，则为模型编译生成的总线结构体对齐格式可能与为 S-Function 生成的补白不一致，这可能影响代码执行的数值结果。要使模型仿真和执行模型代码之间的结构体对齐格式保持一致，可以修改每个 **Simulink.Bus** 对象的 **HeaderFile** 属性，以删除未进行补白的总线结构体头文件。这样将在模型代码中重用为 S-Function 生成的总线类型定义。

#### 具有函数调用输入和输出信号的子系统

S-Function 目标不支持从具有函数调用触发器输入或函数调用输出的子系统创建 S-Function 模块。

#### 函数调用子系统建模模式

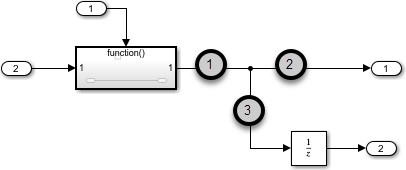
当您将生成的 S-Function 模块添加到模型中，然后仿真该模型时，如果生成的 S-Function 使用此函数调用子系统建模模式，MATLAB 可能会崩溃。



函数调用子系统直接连接到 Outport 模块。将子系统连接到 Outport 模块的信号线分支到具有 **Update**函数的模块（即在每个主时间步更新的模块，如 Unit Delay 和 Memory）。分支既可以在子系统的内 部，也可以在外部。

在包含函数调用子系统的模型中，这些位置之一的 Signal Conversion 模块可以使用生成的 S-Function解决问题：

1. 在子系统模块和分支之间
2. 在分支和根 Outport 模块之间
3. 在分支和具有 **Update** 函数的模块之间



解决办法中的 Signal Conversion 模块具有以下参数设置：

* **输出**设置为“信号副本”

#### 已选择从 '模块简化' 优化中排除此模块

**数据存储访问**

当模型中的 S-Function 模块在仿真期间访问数据存储时，Simulink 将禁用数据存储诊断。

* 如果该 S-Function 模块是从模型创建的，则会同时禁用对全局数据存储的诊断。
* 如果该 S-Function 模块是从子系统创建的，则会禁用对以下数据存储的诊断：
  + 全局数据存储
  + 放置在子系统之外但由 Data Store Read 或 Data Store Write 模块访问的数据存储。

#### 通过子系统封装指定 Inport 或 Outport 模块参数

当您尝试运行使用 S-Function 模块的仿真时，从具有子系统封装变量指定的 Inport 或 Outport 模块参数的子系统生成的 S-Function 会生成错误。

**Invalid setting in 'testSystem/Subsystem/** **OutputSSForSFun** **/Out2' for parameter 'PortDimensions'**

**...**

#### MEX S-Function 包装器

仅在创建包装的 MATLAB 版本中使用 MEX S-Function 包装器。

### 另请参阅详细信息

* [“保护模型以隐藏内容” （第 11-2 页）](#_bookmark66)
* “Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File”
* “C/C++ S-Function 基础知识”
* [“S-Function 和代码生成” （第 14-2 页）](#_bookmark75)
* [“使用代码继承工具在生成的代码中导入对外部代码的调用” （第 14-5 页）](#_bookmark79)
* “Use a Bus with S-Function Builder to Create an S-Function”

# Simulink Coder 中的实时系统

# Simulink Coder 中的外部代码集成

代码生成器包含了多种集成方法，可将现有代码或自定义代码与生成的代码集成。现有代码是指现有的人工代码，或者要与代码生成器生成的代码进行集成的环境中的代码。自定义代码是指您在代码生成器的编译过程中包含的现有代码或其他由用户指定的代码行。现有代码和自定义代码统称为外部代码。

要集成外部代码，您可以将现有外部代码导入到代码生成器生成的代码中、将生成的代码导出到现有外部代码库中，或者同时执行以上两种操作。例如，您可以通过调用外部函数或使用代码继承工具导入代码，或者在模型中包含 Custom Code 模块，从而将外部代码放置在生成的代码中的特定位置。导入外部代码时，生成的代码将与生成的调度代码进行对接。

您可以将生成的代码作为插件函数导出，以便在外部开发环境中使用。导出生成的代码时，您需要手动将该代码与您应用程序运行时环境中的调度机制进行对接。

要了解如何根据应用程序选择集成方法的指导原则，请参阅“Choose an External Code Integration Workflow”。

* [“调用可重用的外部算法代码进行仿真和代码生成” （第 30-2 页）](#_bookmark238)
* [“在 Simulink 环境中编译集成的代码” （第 30-9 页）](#_bookmark244)
* [“生成要导出到外部代码库的组件源代码” （第 30-15 页）](#_bookmark250)
* [“通过使用共享库对接在开发计算机上运行的仿真器” （第 30-27 页）](#_bookmark262)

## 调用可重用的外部算法代码进行仿真和代码生成

代码重用可带来业务和技术上的优势。从业务角度来看，代码重用可节省时间和资源。从技术角度来看，代码重用可提高一致性，减少内存要求。其他考虑因素包括：

* 模块化应用程序
* 重用优化的算法
* 与预定义数据集集成
* 开发应用程序变体

以下各项为独立于硬件且可重用的算法代码，可考虑导入 Simulink 环境中以进行仿真和代码生成：

* 工具函数
* 查找表
* 数字滤波器
* 专用积分器
* 比例积分导数 (PID) 控制模块

### 工作流

要调用可重用的外部算法代码进行仿真和代码生成，请依次执行下表中列出的任务。

|  |  |  |
| --- | --- | --- |
| **任务** | **操作** | **更多信息** |
| 1 | 检查您对外部代码特性和集成要求的评估。 | “Choose an External Code Integration Workflow” (Embedded Coder) |
| 2 | 基于外部代码的编程语言，选择一种集成方法以将外部代码添加到 Simulink 模型中。 | “选择集成方法” (Embedded Coder) |
| 3 | 通过仿真模型验证算法行为和性能。 | “仿真” |
| 4 | 为代码生成定义模型数据的表示。 | “代码接口配置和集成” 和 “Exchange Data Between External C/C++ Code and Simulink Model or Generated Code” (Embedded Coder) |
| 5 | 配置模型以进行代码生成。 | “Generate Code That Matches Appearance of External Code” (Embedded Coder) |
| 6 | 生成代码和代码生成报告。 | “代码生成” |
| 7 | 检查生成的代码接口和静态代码指标。 | “Analyze Generated Data Code Interface” (Embedded Coder) 和 “Static Code Metrics” (Embedded Coder) |
| 8 | 从模型中编译可执行程序。 | “在 Simulink 环境中编译集成的代码” (Embedded Coder) |
| 9 | 验证可执行程序的行为是否符合预期。 | “验证、测试和认证” (Embedded Coder) |

### 选择集成方法

有几种方法可以将可重用的算法代码集成到 Simulink 环境中进行代码生成。有些方法直接集成外部代码。其他方法先将外部代码转换为 Simulink 或 Stateflow 建模元素用于仿真，随后从建立的模型设计进行代码生成。您选择的集成方法取决于：

* 外部代码的编程语言 - MATLAB、C、C++ 或 Fortran
* 您的编程语言经验和偏好
* 性能要求
* 算法必须用于建立连续时间动态模型，还是将算法集成到使用离散和连续时间的应用程序中
* 是否希望利用基于模型的设计
* 所需的对代码生成器生成的代码的控制程度

要为可重用算法选择一种方法，请参阅与您的外部算法代码的编程语言相关的小节。

* “外部 MATLAB 代码的集成方法” (Embedded Coder)
* “外部 C 或 C++ 代码的集成方法” (Embedded Coder)
* “外部 Fortran 代码的集成方法” (Embedded Coder)

#### 外部 MATLAB 代码的集成方法

有多种方法可以将外部 MATLAB 代码集成到 Simulink 环境中。下表可帮助您根据集成要求为应用程序选择最佳集成方法。

|  |  |  |
| --- | --- | --- |
| **条件或要求** | **操作** | **更多信息** |
| * 外部代码支持 MATLAB 代码生成子集 * 您要从 Simulink 模型中调用 MATLAB 代码 | 将 MATLAB Function 模块添加到模型中。在该模块中嵌入 MATLAB代码。 | * “使用 MATLAB Function 模块集成 C 代码” * MATLAB Function |
| * 外部代码支持 MATLAB 代码生成子集 * 您要从 Simulink 模型中调用 MATLAB 代码 * 您的算法包括处理大量数据流的迭代计算 | 向模型中添加 MATLAB System 模块。将 MATLAB 代码作为 System object™ 嵌入该模块。 | * “使用 MATLAB Function 模块集成 C 代码” * MATLAB System |
| * 外部代码支持 MATLAB 代码生成子集 * 您要从 Simulink 模型中调用 MATLAB 代码 * 您的算法包括基于状态机和流程图的设计逻辑 | 向模型中添加 Stateflow 图。使用 MATLAB 作为动作语言，从图中调用外部代码。 | * “图编程基础知识” (Stateflow) * “将外部代码插入 Stateflow 图中” (Embedded Coder) |

|  |  |  |
| --- | --- | --- |
| **条件或要求** | **操作** | **更多信息** |
| 您想使用 **parfor** 函数进行并行计算或使用 MATLAB Coder、Simulink Coder 和 Embedded Coder 可用的接口数据类型。要使用 **parfor**，必须安装 Parallel Computing Toolbox™。 | 使用软件生成 C 代码。然后，以外部 C 代码形式调用生成的代码。 | * “使用 MATLAB Coder App 生成 C 代码” * “MATLAB Coder 快速入门” |
| 您拥有 C 或 C++ 编程经验，外部 MATLAB 代码非常简洁且主要使用 C 或 C++ 构造。 | 手动将 MATLAB 代码转换为 C 或 C++ 代码。为 C 或 C++ 代码选择一种集成方法。 | “外部 C 或 C++ 代码的集成方法” (Embedded Coder) |
| 外部 MATLAB 代码的有些部分可对应到内置模块。 | 使用适用的内置模块在模型中开发算法。 | * “交互式模型编辑” 和 “工程管理” * “Use Products and Blocks Supported for Code Generation” |
| 算法必须用于建立连续状态动态模型。 | 编写一个 MATLAB S-Function，并为生成代码编写对应的算法的 TLC 文件。将 S-Function 添加到您的模型中。 | * “创建和配置 MATLAB S-Function” * “S-Function” * “目标语言编译器” |

要将外部 MATLAB 代码嵌入 MATLAB Function 模块中或使用 MATLAB Coder 软件从 MATLAB 代码中生成 C 或 C++ 代码，MATLAB 代码必须使用 C/C++ 代码生成支持的函数和类。

* “C/C++ 代码生成支持的函数和对象”

#### 外部 C 或 C++ 代码的集成方法

有多种方法可以将外部 C 或 C++ 代码集成到 Simulink 环境中。下表可帮助您根据集成要求为应用程序选择最佳集成方法。

|  |  |  |
| --- | --- | --- |
| **条件或要求** | **操作** | **更多信息** |
| 您要将外部 C 代码与生成的 C  ++ 代码集成在一起，或者反之 | 通过修改外部代码的语言以与生成代码的语言选择一致。 | “修改外部代码的编程语言以匹配生成的代码” (Embedded Coder) |
| * 已有对自定义功能建模的现有 C 代码。您要将代码集   成到 Simulink 模块中。 | 对于用 C 语言编写的简单算法，考虑使用 Simulink C  Caller 模块。 | “使用 C Caller 模块集成 C 代码” |
| * 您的函数不读写全局/静态   变量。 |  |  |
| * 您要轻松地与 Simulink Coverage™、Simulink Test™ 和 Simulink Design Verifier™ 集成 |  |  |

|  |  |  |
| --- | --- | --- |
| **条件或要求** | **操作** | **更多信息** |
| * 已有对自定义功能建模的现有 C 代码。您要从一个 Simulink 模块中调用多个 C 函数，或有条件地调用 C   函数。 | 为您的模型添加一个 C Function 模块。 | “使用 MATLAB Function 模块集成 C 代码” |
| * 您要在调用 C 函数之前和之后使用 C 代码来预处理   和后处理数据。 |  |  |
| * 您要为仿真和代码生成指定不同代码。 |  |  |
| * 算法必须用于对离散和连续的状态动态建模以进行仿真和快速原型。 * 外部代码需要定点接口。 * 编程经验有限。您希望能够以基本的灵活性轻松控制代码生成器为快速原型生成的代码。 | 使用 S-Function Builder 生成 S-Function 和 TLC 文件。如有必要，手动细化生成的代码以满足应用程序要求。（如果您更改了生成的代码，则在您重新生成 S-Function 和 TLC 文件时，您将丢失这些更改。） | * “Use a Bus with S-Function Builder to Create an S-Function” |
| * 您要对离散时间应用程序进行仿真和生成外部代码。优化生成的代码至关重要。 * 您希望能够以适度的灵活性轻松控制代码生成器生成的代码。 * 您有 C 或 C++ 编程经验，但您更喜欢生成用于将代码添加到模型中的文件。 | 使用代码继承工具生成 S- Function 和 TLC 文件。如有必要，手动细化生成的代码以满足应用程序要求。（如果您更改了生成的代码，则在您重新生成  S-Function 和 TLC 文件时，您将丢失这些更改。） | * “Integrate C Functions Using Legacy Code Tool” * [“使用代码继承工具在生成的代码中导入对外部代码的调用” （第 14-5 页）](#_bookmark79) * “使用 C Caller 模块集成 C 代码” |
| * 应用程序需要的入口函数多于代码生成器通常生成的入口函数，例如，需要的入口函数不仅仅是 **model\_step**、 **model\_initialize** 和 **model\_terminate** 入口函数。 | 手动编写 S-Function 和 TLC 文件。 | * “C/C++ S-Function 基础知识” * “S-Function 和代码生成” (Embedded Coder) * “C S-Function Examples” 和 “C++ S- Function Examples” |
| 您需要最大的灵活性来控制代码生成器生成的代码。 |  |
| 您的算法包括基于状态机和流 | 向模型中添加 Stateflow 图。使 | “将外部代码插入 Stateflow 图中” (Embedded |
| 程图的设计逻辑。或者，要集 | 用 C 语言作为动作语言从图中 | Coder) |
| 成的函数必须使用全局变量与 | 调用外部代码。在图中，编写调 |
|  |
| 模型交换数据。该函数定义全 | 用外部函数并读写全局变量的代 |  |
| 局变量，并使用它们来写函数 | 码。要使用外部代码的输出来执 |  |
| 的输出，而不是返回值（返 | 行计算，模型必须在执行期间从 |  |
| 回）或将输出写至参数。 | 全局变量中读取数据。 |  |

|  |  |  |
| --- | --- | --- |
| **条件或要求** | **操作** | **更多信息** |
| 您要在 Stateflow 图中包含外部 C 或 C++ 代码，用于仿真和代码生成。 | 配置包含图的模型以应用外部 C或 C++ 代码。 | * “自定义代码” (Stateflow) * “在 C 语言状态图中调用 C 库函数” (Stateflow) * “将外部代码插入 Stateflow 图中” (Embedded Coder) |
| 您很快要在模型中嵌入对外部 C 或 C++ 代码的调用。性能不是问题。 | 从 MATLAB Function 模块中调用具有 **coder.ceval** 函数的 C或 C++ 代码。 | * **coder.ceval** 函数 * “使用 MATLAB Function 模块集成 C 代码” * MATLAB Function 模块 |

#### 修改外部代码的编程语言以匹配生成的代码

要将外部 C 代码与生成的 C++ 代码集成（或者反之），请修改外部代码的语言以与生成代码的编程语言选择一致。使编程语言匹配的选项包括：

* 用生成代码的语言选择编写或重写外部代码。
* 如果您正在生成 C++ 代码并且外部代码是 C 代码，则为每个 C 函数创建一个为该函数建立原型的头文件。使用以下格式：

**#ifdef cplusplus**

**extern "C" { #endif**

**int my\_c\_function\_wrapper();**

**#ifdef**

**}**

**#endif**

**cplusplus**

原型充当函数包装器。如果您的编译器支持 C++ 代码，则会定义值 **cplusplus**。链接设定 **extern "C"** 指定不带名称修饰的 C 链接。

* 如果您正在生成 C 代码并且外部代码是 C++ 代码，请在每个 **.cpp** 文件中包含 **extern "C"** 链接设定。例如，以下示例展示文件 **my\_func.cpp** 中的 C++ 代码：

**extern "C" {**

**int my\_cpp\_function()**

**{**

**...**

**}**

**}**

#### 外部 Fortran 代码的集成方法

要集成外部 Fortran 代码，请执行下列步骤：

1. 编写一个 S-Function 和对应的 TLC 文件。
2. 将 S-Function 添加到您的模型中。
3. 如有必要，在 Simulink 环境中添加支持文件和控制模型代码生成与编译。

请参阅 “C/C++ S-Function 基础知识”、“用 Fortran 代码实现算法”、“S-Function 和代码生成” (Embedded Coder)和 “Fortran S-Function Examples”。

### 将外部代码插入 Stateflow 图中

* [“为库图集成外部代码” （第 30-7 页）](#_bookmark242)
* [“为所有图集成外部代码” （第 30-7 页）](#_bookmark243)

#### 为库图集成外部代码

要集成仅应用于 Stateflow 库图的外部代码以进行代码生成，则对于为您的主模型提供了 Stateflow 图的每个库模型，请完成以下步骤。然后再生成代码。

1. 在 Stateflow 编辑器中，打开“模型配置参数”对话框。选择参数**使用本地自定义代码设置(不从主模型继承)**，

库模型在代码生成过程中会保留它自己的自定义代码设置。

1. 在子窗格中指定您的自定义代码。

请遵循“Specify Relative Paths to Your Custom Code” (Stateflow) 中的规范。

如果您为仿真指定了自定义代码设置，则可以将这些设置应用于代码生成。要避免两次输入相同的信息，请选择**使用与仿真目标相同的自定义代码设置**。

1. 点击**确定**。

为每个库模型完成上述步骤后，生成代码。

#### 为所有图集成外部代码

要集成应用于所有图的外部代码以进行代码生成，请执行以下操作：

1. 为主模型的代码生成指定自定义代码选项。
   1. 在“模型配置参数”对话框中，选择**代码生成** > **自定义代码**。
   2. 在自定义代码文本字段中，指定您的自定义代码。

请遵循“Specify Relative Paths to Your Custom Code” (Stateflow) 中的规范。

如果您为仿真指定了自定义代码设置，则可以将这些设置应用于代码生成。要避免两次输入相同的信息，请选择**使用与仿真目标相同的自定义代码设置**。

1. 为向主模型提供 Stateflow 图的每个库模型进行代码生成配置。在 Stateflow 编辑器中，打开“模型配置参数”对话框。清除参数**使用本地自定义代码设置(不从主模型继承)**。库图将继承主模型的自定义代码设置。点击**确定**。
2. 生成代码。

### 另请参阅详细信息

* “Integrate C Functions Using Legacy Code Tool”
* [“使用代码继承工具在生成的代码中导入对外部代码的调用” （第 14-5 页）](#_bookmark79)
* “自定义代码” (Stateflow)
* “使用 MATLAB Function 模块集成 C 代码”
* “C/C++ S-Function 基础知识”
* “S-Function 和代码生成” (Embedded Coder)

## 在 Simulink 环境中编译集成的代码

### 工作流

要编译集成生成的代码和外部 C 或 C++ 代码的可执行程序，请依次执行下表中的任务。

|  |  |  |
| --- | --- | --- |
| **任务** | **操作** | **更多信息** |
| 1 | 选择一种编译方法。 | “Approaches for Building Code Generated from Simulink Models” (Embedded Coder)  有关示例，请参阅“Build Process Workflow for Real-Time Systems” (Embedded Coder)。 |
| 2 | 为您的外部代码配置编译过程支持。 | “为集成的代码编译过程配置参数” (Embedded Coder) |
| 3 | 为您的外部代码配置 S-Function 编译支持。 | “编译对 S-Function 的支持” (Embedded Coder)  “Use makecfg to Customize Generated Makefiles for S-Functions” (Embedded Coder)  有关示例，请参阅“从模型和生成的代码调用外部 C代码” (Embedded Coder) 和“调用可重用的外部算法代码进行仿真和代码生成” (Embedded Coder)。 |
| 4 | 配置编译过程以查找外部代码源、库和头文件。 | “Manage Build Process File Dependencies” (Embedded Coder)  “Control Library Location and Naming During Build” (Embedded Coder) |
| 5 | 设置您的外部代码集成所需的自定义编译处理。 | 有关编译过程自定义工作流的信息，请参阅 “Customize Post-Code-Generation Build Processing” (Embedded Coder)。  要自动将编译自定义应用于工具链方法编译，请参阅 “Customize Build Process with sl\_customization.m” (Embedded Coder)。  要自动将编译自定义应用于模板联编文件方法编译，请参阅“使用 STF\_make\_rtw\_hook 文件自定义编译过程” (Embedded Coder)。 |

### 为集成的代码编译过程配置参数

下表说明如何配置编译过程的“模型配置参数”对话框的**代码生成** > **自定义代码**窗格中的模型配置参数，以支持在编译过程中集成外部代码。有关外部代码的文件夹的信息，请参阅“管理编译过程文件夹” (Embedded Coder)。如果您选择将外部代码放入“Code generation folder” 中，请参阅“在编译文件夹中保留外部代码文件” (Embedded Coder)。

|  |  |
| --- | --- |
| **要执行的操作** | **选择** |
| 将包含文件夹（其中包含头文件）添加到编译过程 | **代码信息** > **包含目录**，然后输入文件夹的绝对路径或相对路径。  如果指定相对路径，则路径必须相对于包含模型文件的文件夹，而不是相对于编译文件夹。您指定文件夹的顺序是在其中搜索头文件、源和库文件的顺序。 |
| 添加要编译和链接的源文件 | **代码信息** > **源文件**，然后输入文件的完整路径或仅输入文件名。  如果文件位于当前 MATLAB 文件夹或其中一个包含文件夹中，请只输入文件名。对于您指定的每个其他源，编译过程都会为源文件所在的文件夹在模板联编文件中展开一条通用规则。例如，如果源文件位于文件夹 **inc**中，则编译过程将添加一条类似于以下内容的规则：  **%.obj: buildir\inc\%.c**  **$(CC) -c -Fo$(@F) $(CFLAGS) $<**  编译过程按照列出源文件的顺序添加规则。 |
| 添加要链接的库 | **代码信息** > **库**，然后输入库的完整路径或仅输入文件名。  如果库位于当前 MATLAB 文件夹或其中某个包含文件夹中，请只输入文件名。 |
| 使用与针对 MATLAB Function 模块、Stateflow 图和 Truth Table 模块的仿真指定的相同的自定义代码设置 | **使用与仿真目标相同的自定义代码设置**  此参数引用“配置参数”对话框中的**仿真目标**窗格。 |
| 使库模型能够使用库所链接到的父模型中具有唯一性的自定义代码设置 | **使用本地自定义代码设置(不从主模型继承)**  此参数仅适用于包含 MATLAB Function 模块、Stateflow 图或 Truth Table 模块的库模型。 |

### 在编译文件夹中保留外部代码文件

默认情况下，编译过程将删除外部源文件。您可以按照下列规范保留外部源文件。

如果将 **.c/.cpp** 或 **.h** 源文件放入编译文件夹中，并且希望防止代码生成器在 TLC 代码生成过程中将其删除，请在 **.c/.cpp** 或 **.h** 文件的第一行中插入文本 **target specific file**。例如：

**/\* COMPANY-NAME target specific file**

**\***

* **This file is created for use with the**
* **COMPANY-NAME target.**
* **It is used for ...**

**\*/**

**...**

确保“target specific file”的拼写与前面示例中的完全相同，并且这段文本位于源文件的第一行。其他文字可以出现在这些文字之前或之后。

以这种方式标记用户文件可防止对这些文件进行后处理，避免用生成的源文件对这些文件的代码格式进行整理。在以前的版本中会发生自动缩进，以编译具有模式 **model\_\*.c/.cpp**（其中 **\*** 为文本）的名称的文件夹文件。格式整理不会损坏文件，但可能会导致源代码管理软件检测到差异，这些差异可能会触发不必要的更新。

### 编译对 S-Function 的支持

借助用户编写的 S-Function 模块，您可以将外部代码整合到 Simulink 开发环境中。在大多数情况下，您可以使用 S-Function 将现有外部代码与生成的代码进行集成。有几种编写 S-Function 的方法：

* [“编写非内联 S-Function” （第 14-18 页）](#_bookmark89)
* [“编写包装器 S-Function 和 TLC 文件” （第 14-20 页）](#_bookmark92)
* [“编写完全内联的 S-Function” （第 14-27 页）](#_bookmark98)
* “Write Fully Inlined S-Functions with mdlRTW Routine”
* “S-Functions for Code Reuse”
* “S-Functions for Multirate Multitasking Environments”

使用 S-Function，还可以最灵活有效地在编译过程中包含原有和自定义代码文件的编译信息。将 S-Function 添加到编译过程有多种不同的方法。

#### 隐式编译支持

在编译具有 S-Function 的模型时，编译过程向生成的联编文件添加规则、包含路径和源文件名。S- Function 的源文件（**.h**、**.c** 和 **.cpp**）必须与 S-Function MEX 文件位于同一文件夹中。无论是使用工具链方法还是使用模板联编文件方法进行编译，编译过程都会将此信息传播到工具链或模板联编文件中。

* 如果文件 **sfcnname.h** 与 S-Function MEX 文件位于同一文件夹中（例如 **sfcnname.mexext**），则该文件夹将被添加到包含路径中。
* 如果文件 **sfcnname.c** 或 **sfcnname.cpp** 与 S-Function MEX 文件位于同一文件夹中，则编译过程会添加一个联编文件规则，以编译来自该文件夹的文件。
* 当 S-Function 不与 TLC 文件内联时，编译过程必须编译 S-Function 源文件。为了确定要添加到要编译的文件列表的源文件的名称，编译过程会在 MATLAB 路径上搜索 **sfcnname.cpp**。如果找到源文件，则编译过程会将源文件名添加到联编文件。如果在路径上找不到 **sfcnname.cpp**，则编译过程会将文件名 **sfcnname.c** 添加到联编文件，无论它是否位于 MATLAB 路径上都是如此。

**注意** 为了使 Simulink 引擎找到用于仿真和代码生成的 MEX 文件，它必须存在于 MATLAB 路径上，或存在于我们的当前 MATLAB 工作文件夹中。

#### 为 S-Function 指定其他源文件

如果您的 S-Function 有其他源文件依存关系，则必须将其他模块的名称添加到编译过程。指定文件名：

* 在“S-Function 模块参数”对话框的 **S-Function 模块**字段中
* 在对 **set\_param** 函数的调用中使用 **SFunctionModules** 参数

例如，假设您编译具有多个模块的 S-Function。

**mex sfun\_main.c sfun\_module1.c sfun\_module2.c**

然后，您可以通过执行以下操作之一将模块添加到编译过程：

* 在“S-Function 模块”对话框中，在 **S-Function 模块**字段中指定 **sfun\_main**、**sfun\_module1** 和

#### sfun\_module2。

* 在 MATLAB 命令提示符下，输入：

**set\_param(sfun\_block,'SFunctionModules','sfun\_module1 sfun\_module2')**

您也可以定义一个变量来表示参数值。

**modules = 'sfun\_module1 sfun\_module2' set\_param(sfun\_block,'SFunctionModules', modules)**

**S-Function 模块**字段和 **SFunctionModules** 参数不支持完整的源文件路径设定。要使用该参数，代码生成器在执行联编文件时必须找到其他源文件。为了让代码生成器找到其他文件，请将它们放在与 S- Function MEX 文件相同的文件夹中。然后您可以利用“隐式编译支持” [（第 30-11](#_bookmark249) 页）中所述的隐式编译支持。

准备好生成代码之后，强制代码生成器重新编译顶层模型，如“Control Regeneration of Top Model Code”中所述。

对于更复杂的 S-Function 文件依存关系，例如在其他位置指定源文件或者指定库或目标文件，请使用

**rtwmakecfg.m** API，如“Use rtwmakecfg.m API to Customize Generated Makefiles”中所述。

#### 使用 TLC 库函数

如果通过编写 TLC 文件来内联 S-Function，则可以使用 TLC 库函数 **LibAddToModelSources** 将源文件名添加到编译过程。有关详细信息，请参阅“LibAddSourceFileCustomSection(file, builtInSection, newSection)”。

**注意** 此函数不支持完整的源文件路径设定。该函数假定代码生成器在执行联编文件时可以找到其他源文

件。

另一个有用的 TLC 库函数是 **LibAddToCommonIncludes**。在 **#include** 语句中使用此函数以将 S- Function 头文件包含在生成的 **model.h** 头文件中。有关详细信息，请参阅 “LibAddToCommonIncludes(incFileName)”。

对于更复杂的 S-Function 文件依存关系，例如在其他位置指定源文件或者指定库或目标文件，请使用

**rtwmakecfg.m** API，如“Use rtwmakecfg.m API to Customize Generated Makefiles”中所述。

#### 预编译 S-Function 库

您可以使用 MATLAB 语言函数 **rtw\_precompile\_libs** 为模型预编译新的或更新的 S-Function 库（MEX文件）。此函数使用指定的模型和库编译设定编译库并将其放置在预编译库文件夹中。

通过预编译 S-Function 库，可以优化系统编译。一旦您的预编译库存在，编译过程便可以在后续的编译中省略库编译。对于使用大量库的模型，可以显著节省编译处理的时间。

要使用 **rtw\_precompile\_libs**，请执行下列操作：

1. 根据您的系统平台设置库文件后缀，包括文件类型扩展名。

考虑确定平台类型，然后使用 **TargetLibSuffix** 参数对应地设置库后缀。例如，为 GRT 目标应用后缀时，可以将后缀设置为 **\_std.a**（对于 UNIX 平台）和 **\_vcx64.lib**（对于 Windows 平台）。

**if isunix**

**suffix = '\_std.a'; else**

**suffix = '\_vcx64.lib'; end**

**set\_param(my\_model,'TargetLibSuffix', suffix);**

有许多因素会影响预编译库后缀和扩展名。下表提供了通常选择的系统目标文件、编译器工具链以及影响您选择的后缀和扩展名的其他选项的示例。有关详细信息，请检查 **matlab/rtw/c/grt** 文件夹或 **matlab/rtw/c/ert** 文件夹中的模板联编文件。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TMF 文件** | **COMPILER**  **\_TOOL\_CHAIN**  **值** | **预编译器库 (PRECOMP\_LIBRARIES)** | | | |
| **库后缀 S- Function**  **（EXPAND**  **\_LIBRARY**  **\_NAME 值）** | **库后缀整数 - 仅代码（EXPAND**  **\_LIBRARY**  **\_NAME 值）** | **库后缀速度优化**  **（EXPAND**  **\_LIBRARY**  **\_NAME 值）** | **库扩展名**  **（EXPAND**  **\_LIBRARY**  **\_NAME 值）** |
| ert\_lcc64.tmf | lcc | \_rtwsfcn\_lcc | \_int\_ert\_lcc | \_ert\_lcc | .lib |
| ert\_vcx64.tmf | vcx64 | \_rtwsfcn\_vcx64 | \_int\_ert\_vcx64 | \_ert\_vcx64 | .lib |
| ert\_unix.tmf | unix | \_rtwsfcn | \_int\_ert | \_ert | .a |
| grt\_lcc64.tmf | lcc | 不适用 | 不适用 | \_lcc | .lib |
| grt\_vcx64.tmf | vcx64 | 不适用 | 不适用 | \_vcx64 | .lib |
| grt\_unix.tmf | unix | 不适用 | 不适用 | \_std | .a |

1. 设置预编译库文件夹。

使用以下方法之一来设置预编译库文件夹：

* + 设置 **TargetPreCompLibLocation** 参数，如“Specify the Location of Precompiled Libraries”中所述。
  + 在 **rtwmakecfg.m** 函数文件中设置 **makeInfo.precompile** 字段。（有关详细信息，请参阅 “Use rtwmakecfg.m API to Customize Generated Makefiles”。）

#### 如果设置 TargetPreCompLibLocation 和 makeInfo.precompile，则 TargetPreCompLibLocation 的设置优先。

以下命令将模型 **my\_model** 的预编译库文件夹设置为当前工作文件夹下的 **lib** 文件夹。

**set\_param(my\_model,'TargetPreCompLibLocation', fullfile(pwd,'lib'));**

**注意** 如果您为预编译库文件和目标库文件后缀都设置目标文件夹，则编译过程将在处理编译时检测是否缺少任何预编译库文件。

1. 定义编译设定。

设置定义编译设定的结构体。下表描述您可以在结构体中定义的字段。这些字段是可选的，但

#### rtwmakecfgDirs 除外。

|  |  |
| --- | --- |
| **字段** | **描述** |
| **rtwmakecfgDirs** | 字符向量元胞数组，用于命名包含要预编译的库的 **rtwmakecfg** 文件的文件夹。该函数使用 **rtwmakecfg** 返回的 **makeInfo.library** 的 **Name** 和 **Location** 元素来指定预编译库的名称和位置。如果您通过设置 **TargetPreCompLibLocation** 参数来指定库文件夹，则该设置将覆盖 **makeInfo.library.Location** 设置。  **注意：**指定的模型必须包含使用由 **rtwmakecfg** 文件指定的预编译库的模块，因为仅在编译过程使用这些库时，TMF 到联编文件的转换才会生成库规则。 |

|  |  |
| --- | --- |
| **字段** | **描述** |
| **libSuffix** | 字符向量，指定要附加到每个库名称的后缀（包括文件类型扩展名），例如 **.a** 或  **\_vc.lib**。字符向量必须包含句点 (.)。您必须使用此字段或 **TargetLibSuffix** 参数设置后缀。如果您同时使用这两种机制指定后缀，则 **TargetLibSuffix** 设置将覆盖此字段的设置。 |
| **intOnlyBuild** | 布尔标志。当设置为 true 时，该标志表示库将优化，以便它们仅从整数代码编译。此字段仅适用于 ERT 目标。 |
| **makeOpts** | 字符向量，指定要包含在 **rtwMake** 命令行中的选项。 |
| **addLibs** | 结构体的元胞数组，这些结构体指定要编译的、并非由 **rtwmakecfg** 函数指定的库。必须使用两个字符数组形式的字段来定义每个结构体：   * **libName** - 没有后缀的库名称 * **libLoc** - 预编译库的位置   目标联编文件 (TMF) 可以指定其他库以及这些库的编译方式。使用此字段来预编译这些库。 |

以下命令设置编译设定 **build\_spec**，它指示要编译的文件位于当前工作文件夹下的 **src** 文件夹中。

**build\_spec = [];**

**build\_spec.rtwmakecfgDirs = {fullfile(pwd,'src')};**

1. 发出对 **rtw\_precompile\_libs** 的调用。

该调用必须指定要为其编译预编译库的模型和编译设定。例如：

**rtw\_precompile\_libs(my\_model,build\_spec);**

### 另请参阅详细信息

* “调用可重用的外部算法代码进行仿真和代码生成” (Embedded Coder)
* “Place External C/C++ Code in Generated Code” (Embedded Coder)
* “Call External Device Drivers” (Embedded Coder)
* “Deploy Applications to Target Hardware” (Embedded Coder)
* “Deploy Generated Component Software to Application Target Platforms” (Embedded Coder)

## 生成要导出到外部代码库的组件源代码

[“建模选项” （第 30-15 页）](#_bookmark251)

[“要求” （第 30-15 页）](#_bookmark252) [“导出函数子系统的限制” （第 30-16 页）](#_bookmark253) [“工作流” （第 30-17 页）](#_bookmark254) [“选择集成方法” （第 30-17 页）](#_bookmark255)

[“生成导出函数模型的 C 函数代码” （第 30-18 页）](#_bookmark256) [“为导出函数模型生成 C++ 函数和类代码” （第 30-21 页）](#_bookmark257) [“为导出函数子系统生成代码” （第 30-25 页）](#_bookmark258)

**本节内容**

如果您有 Embedded Coder 软件，您可以从建模组件中生成函数源代码，以便在外部代码库中使用。生成的代码不包括相关调度代码（例如，单步函数）。Simulink 环境之外的控制逻辑会调用生成的函数代码。

### 建模选项

您可以为下列建模组件生成要导出的函数代码：

* 导出函数模型（包含函数模块的模型，这些函数模块只包含函数调用子系统、函数调用模型模块或其他导出函数模型，如“导出函数模型概述”中所述）
* 导出函数子系统（包含函数调用子系统的虚拟子系统）

要导出代码生成器为这些建模组件生成的代码，建模组件必须满足特定的要求 (Embedded Coder)。

对于在以前的版本中设计的模型，代码生成器可以从触发子系统中导出函数。对导出函数子系统的相关要求也适用于从触发子系统导出函数，但以下情况除外：

* 将要从中导出函数的触发子系统封装在一个顶层虚拟子系统中。
* 触发子系统不必满足针对包含函数调用子系统的虚拟子系统的要求和限制。
* “导出使用绝对时间或执行时长的函数” (Embedded Coder)不适用于从触发子系统导出函数。

### 要求

* 模型求解器必须为定步长离散求解器。
* 对于会触发函数调用子系统的每个根级 Inport 模块，必须将其配置为输出一个函数调用触发器。这些 Inport 模块不能连接到 Asynchronous Task Specification 模块。
* 模型或子系统，在根级必须仅包含以下模块：
  + 函数调用模块（如根级的 Function-Call Subsystem、Simulink Function、S-Functions 和 Function-Call Model 模块，如果求解器模型配置参数**任务和采样时间选项** > **周期性采样时间约束**设置为“确保采样时间独立”）
  + Inport 和 Outport 模块（端口）
  + Constant 模块（包括解析为常量的模块，如 Add）
  + 采样时间为 Inf 的模块
  + Merge 和 Data Store Memory 模块
  + 虚拟连接模块（例如，Function-Call Split、Mux、Demux、Bus Creator、Bus Selector、Signal Specification 以及包含这些模块的虚拟子系统）
  + 信号查看器模块，如 Scope 模块（仅导出函数子系统）
* 当模型或子系统的顶层出现常量模块时，您必须将模型或所在模型的配置参数**优化** > **默认参数行为**设置为“内联”。
* 模型或子系统中的模块必须支持代码生成。
* 使用绝对时间或已用时间的模块必须位于周期函数调用子系统内，且在对应的函数调用根级 Inport 模块上指定了离散采样时间。请参阅“导出使用绝对时间或执行时长的函数” (Embedded Coder)。
* 对于导出的系统，跨越其边界的数据信号不能为虚拟总线，也无法实现为 Goto-From 连接。跨导出边界的数据信号必须为标量、多路或非虚拟总线。

此外，对于导出函数模型，您不能为包含多个导出函数模型实例的基于速率的模型生成代码。例如，对于在仿真期间用于调度可重用的导出函数模型的测试框架模型，您不能为其生成代码。

对于导出函数子系统，还适用以下附加要求：

* 跨导出函数子系统边界的触发信号必须为标量。不作为触发信号的输入输出数据信号不必是标量。
* 当常量信号驱动导出函数子系统的输出端口时，该信号必须指定存储类。

#### 导出使用绝对时间或执行时长的函数

当建模组件具有使用绝对时间或执行时长的模块时，如果要为该建模组件导出函数代码，这些模块必须位于函数调用子系统内，该子系统需要满足以下条件：

* 配置为周期性执行
* 其根级 Inport 模块配置为使用离散采样时间

要将函数调用子系统配置为周期性执行，请执行以下操作：

1. 在函数调用子系统中，右键点击 Trigger 模块，并从上下文菜单中选择**模块参数**。
2. 将参数**采样时间类型**设置为“周期”。
3. 将**采样时间**设置为函数调用发起方中指定的相同粒度（直接指定或通过继承）。
4. 点击**确定**或**应用**。

有关详细信息，请参阅“Absolute and Elapsed Time Computation”。

### 导出函数子系统的限制

* 子系统模块参数不控制包含生成代码的文件的名称。文件名以导出的子系统的名称开头。
* 子系统模块参数不控制生成代码中顶层函数的名称。每个函数名称会反映触发该函数的信号的名称，如果信号未命名，则反映作为信号源的模块的名称。
* 仅当 C++ 类代码接口打包的函数设定设置为**默认单步方法**时，才能导出配置了 C++ 类代码接口打包的函数调用系统。请参阅“Interactively Configure C++ Interface” (Embedded Coder)。导出的函数与单线程执行兼容。为了避免共享信号的潜在数据竞争情况，请从同一个执行线程调用类的所有成 员。
* 仅当其函数调用发起方在加速模式下为非内联时，代码生成器才会在加速模式下支持 SIL 或 PIL 模块。 Stateflow 图就是一个非内联的发起方。
* 2 级 S-Function 发起方模块，如 Stateflow 图或内置 Function-Call Generator 模块，必须驱动 SIL模块。
* 您可以导出异步（采样时间）函数调用系统，但软件不支持异步系统的 SIL 或 PIL 模块。
* 导出函数子系统不再使用 TLC 函数 **LibIsFirstInit**。

### 工作流

要为导出的函数生成代码，请依次执行下表中列出的任务。

|  |  |  |
| --- | --- | --- |
| **任务** | **操作** | **更多信息** |
| 1 | 检查您对外部代码特性和集成要求的评估。 | “Choose an External Code Integration Workflow” (Embedded Coder) |
| 2 | 验证您正在导出的模型或子系统是否满足函数导出要求。 | “要求” (Embedded Coder) |
| 3 | 通过修改模型或子系统来满足数据接口要求。 | “Exchange Data Between External C/C++ Code and Simulink Model or Generated Code” (Embedded Coder) |
| 4 | 如有必要，配置函数原型。 | “Configure Entry-Point Function Interfaces for Simulink Function and Function Caller Blocks” (Embedded Coder)；对于基于速率的定步长模型，请参阅“为模型入口函数配置生成的 C 函数接口” (Embedded Coder)或“Interactively Configure C  ++ Interface” (Embedded Coder) |
| 5 | 如有必要，更新模型，将外部的应用程序特定代码放入生成的系统函数中。 | “Place External C/C++ Code in Generated Code” (Embedded Coder) |
| 6 | 通过创建和使用测试框架模型，验证函数在仿真期间的行为和性能是否符合预期。测试框架模型在仿真期间调度函数的执行。 | “配置模型、生成代码和仿真” (Embedded Coder)；如果您有 Simulink Test 软件，请参阅 “Test Authoring” (Simulink Test) |
| 7 | 为代码生成配置模型或子系统。 | “Generate Code That Matches Appearance of External Code” (Embedded Coder) |
| 8 | 生成代码和代码生成报告。 | “代码生成” |
| 9 | 检查生成的代码接口和静态代码指标。 | “Analyze Generated Data Code Interface” (Embedded Coder) 和 “Static Code Metrics” (Embedded Coder) |
| 10 | 编译包含导出的函数代码的可执行程序。 | “在 Simulink 环境之外编译集成的代码” (Embedded Coder) |
| 11 | 验证可执行程序的行为和性能是否符合预期。 | “验证、测试和认证” (Embedded Coder) |

### 选择集成方法

有多种方法可用于生成导出到外部开发环境的函数代码。下表比较了各种方法。您可选择最符合您的集成要求的方法。有关如何创建导出函数模型的详细信息，请参阅“导出函数模型概述”。有关为函数调用子系统生成代码的详细信息，请参阅“生成要导出到外部代码库的组件源代码” (Embedded Coder)。

|  |  |  |
| --- | --- | --- |
| **条件或要求** | **使用** | **更多信息** |
| * 建模元素和生成代码之间的可追溯性 * 本地输入（Inport 模块）和输出   （Outport 模块） | 函数调用子系统 | * “生成导出函数模型的 C 函数代码” (Embedded Coder) * “为导出函数模型生成 C++ 函数和类代码” (Embedded Coder) * “使用函数调用子系统” * Function-Call Subsystem |
| * 对生成的函数原型的控制 * 形式输入参数（Argument Inport模块）和输出参数（Argument Outport 模块） * 本地输入（Inport 模块）和输出   （Outport 模块） | Simulink Function 模块 | * “Configure Entry-Point Function Interfaces for Simulink Function and Function Caller Blocks” (Embedded Coder) * “Simulink Function Blocks and Code Generation” * “Simulink 函数” * Simulink Function |
| 代码响应初始化事件 | Initialize Function 模块 | * “生成导出函数模型的 C 函数代码” (Embedded Coder) * “为导出函数模型生成 C++ 函数和类代码” (Embedded Coder) * “Startup, Reset, and Shutdown Function Interfaces” * “使用初始化、重新初始化、重置和终止函数” |
| 代码响应重置事件 | Reset Function 模块 | * “生成导出函数模型的 C 函数代码” (Embedded Coder) * “为导出函数模型生成 C++ 函数和类代码” (Embedded Coder) * “Startup, Reset, and Shutdown Function Interfaces” * “使用初始化、重新初始化、重置和终止函数” |
| 代码包括的入口函数超出代码生成器默认生成的入口函数  （**model\_initialize**、**model\_step**和 **model\_terminate**） | S-Function | [“S-Function 和代码生成” （第 14-2 页）](#_bookmark75) |
| 单模型执行框架要用作测试框架并导出为模型部分生成的代码 | 导出函数子系统 | * “子系统” * “使用函数调用子系统” * Function-Call Subsystem |

### 生成导出函数模型的 C 函数代码

此示例说明如何为模型中的各个 Simulink➅ 函数模块和函数调用子系统生成函数代码，而不生成调度代码。

要生成要导出的函数代码，请执行下列操作：

1. 创建一个包含要导出的函数的模型。
2. 创建一个测试框架模型，用于在仿真期间调度函数的执行。
3. 使用该测试框架模型对包含这些函数的模型进行仿真。
4. 为包含这些函数的模型生成代码。

#### 创建包含要导出的函数的模型

具有要导出的函数的模型必须满足模型根级别的架构约束。在根级别，有效模块为：

* Inport
* Outport
* Function-Call Subsystem
* Simulink Function
* Goto
* From
* Merge

代码生成器为 Function-Call Subsystem、Simulink Function、Initialize Function 和 Reset Function 模块生成函数代码。对于 Function-call Subsystem 模块，您需要将模块输入端口连接到用于断言函数调用信号的根 Inport 模块。子系统会根据接收到的函数调用信号而执行。Simulink Function 模块会对相应 Function Caller 模块或 Stateflow 图的执行进行响应运行。在发生模型初始化事件时会执行 Initialize Function 模块，在发生用户定义的重置事件时会执行 Reset Function 模块。

为了导出函数，模型 **rtwdemo\_functions** 包含两个函数调用子系统（**f1\_alg** 和 **f2\_alg**）和一个 Simulink Function 模块 (**f3**)，它们均用于导出函数。该模型还包含一个 Initialize Function 模块 (**Initialize Function**) 和一个 Reset Function 模块 (**Reset Function**)。要计算在模型的其他部分中有状态的模块的初始条件，可在 Initialize Function 和 Reset Function 模块内部使用 State Writer 模块。

**open\_system('rtwdemo\_functions')**

#### 创建包含 Function Caller 模型

使用 Function Caller 模块来调用 Simulink Function 模块。Function Caller 模块可以与 Simulink Function 模块在同一模型中，也可以在不同模型中。

多个 Function Caller 模块可以调用一个 Simulink Function 模块。您可以将 Function Caller 模块放在函数调用子系统内。在代码生成期间，代码生成器从函数调用子系统导出函数。

模型 **rtwdemo\_caller** 导出包含 Function Caller 模块的函数调用子系统。

**open\_system('rtwdemo\_caller')**

#### 创建用于仿真的测试框架模型

导出函数时，生成的代码不包含调度器。创建一个测试框架模型来处理仿真期间的调度。不要使用该测试框架模型来生成您部署的代码。

模型 **rtwdemo\_export\_functions** 是一个测试框架。模型：

* 在 **rtwdemo\_caller** 中用 Function Caller 模块调度 Simulink Function 模块。
* 向此示例中的其他模型提供函数调用信号以调度模型内容，包括模型初始化和重置事件。

**open\_system('rtwdemo\_export\_functions')**

#### 对测试框架模型进行仿真

通过对测试框架模型进行仿真，验证包含要导出的函数的模型是否按预期执行。例如，对

#### rtwdemo\_export\_functions 进行仿真。

**sim('rtwdemo\_export\_functions')**

#### 生成函数代码

打开 **Embedded Coder**。然后，为要导出的函数生成代码。例如，为 **rtwdemo\_functions** 生成代码。

**slbuild('rtwdemo\_functions')**

**### Starting build procedure for: rtwdemo\_functions**

**### Successful completion of code generation for: rtwdemo\_functions Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**======================================================================================**

**rtwdemo\_functions Code generated. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 19.151s**

#### 查看生成的代码

查看生成的代码。

* **ert\_main.c** 是模型的示例主程序（执行框架）。此代码显示如何调用导出的函数。该代码还显示如何初始化和执行生成的代码。
* **rtwdemo\_functions.c** 调用初始化函数（包括 **Initialize Function**）和针对模型组件 **f1\_alg**、 **f2\_alg** 和 **f3** 导出的函数。
* **rtwdemo\_functions.h** 声明模型数据结构体和一个对接导出的入口函数和数据结构体的公共接口。
* **f3.h** 是共享文件，用于声明 Simulink 函数 **f3** 的调用接口。
* **rtwtypes.h** 定义生成的代码所需的数据类型、结构体和宏。

#### 编写接口代码

打开并查看代码接口报告。要编写用于您的执行框架的接口代码，请使用该报告中的信息。

1. 通过添加指令 **#include rtwdemo\_functions.h**、**#include f3.h** 和 **#include rtwtypes.h** 来包含生成的头文件。
2. 将输入数据写入模型的 Inport 模块的生成代码。
3. 调用生成的入口函数。
4. 从模型 Outport 模块的生成代码中读取数据。

输入端口：

* **rtU.U1** 的类型为 **real\_T**，维度为 1
* **rtU.U2** 的类型为 **real\_T**，维度为 1

入口函数：

* 初始化入口函数 **void rtwdemo\_functions\_initialize(void)**。在启动时，调用一次此函数。
* 重置入口函数 **void rtwdemo\_functions\_reset(void)**。根据需要调用此函数。
* 导出的函数 **void f1(void)**。根据需要调用此函数。
* 导出的函数 **void f2(void)**。根据需要调用此函数。
* Simulink 函数 **void f3(real\_T rtu\_u, real\_T \*rty\_y)**。根据需要调用此函数。

输出端口：

* **rtY.Accumulator1** 的类型为 **int8\_T**，维度为 [2]
* **rtY.Accumulator2** 的类型为 **int8\_T**，维度为 [2]
* **rtY.TicToc10** 的类型为 **int8\_T**，维度为 1

#### 有关详细信息

* “生成要导出到外部代码库的组件源代码” (Embedded Coder)
* “Deploy Applications to Target Hardware” (Embedded Coder)
* “Customize Code Organization and Format” (Embedded Coder)
* “为模型入口函数配置生成的 C 函数接口” (Embedded Coder)

#### 关闭示例模型

**bdclose('rtwdemo\_export\_functions') bdclose('rtwdemo\_functions') bdclose('rtwdemo\_caller')**

### 为导出函数模型生成 C++ 函数和类代码

此示例说明如何为包含函数调用子系统的导出函数模型生成函数代码。代码生成器生成不包括调度代码的函数和类代码。

要生成要导出的函数代码，请执行下列操作：

1. 创建一个包含要导出的函数的模型。
2. 创建一个测试框架模型，用于在仿真期间调度函数的执行。
3. 使用该测试框架模型对包含这些函数的模型进行仿真。
4. 为包含这些函数的模型生成代码。

#### 创建包含要导出的函数和 C++ 类接口的模型

包含带 C++ 模型类接口的导出函数的模型必须满足模型根级别的架构约束。对于 C++ 类的生成，在根级别有效的模块为：

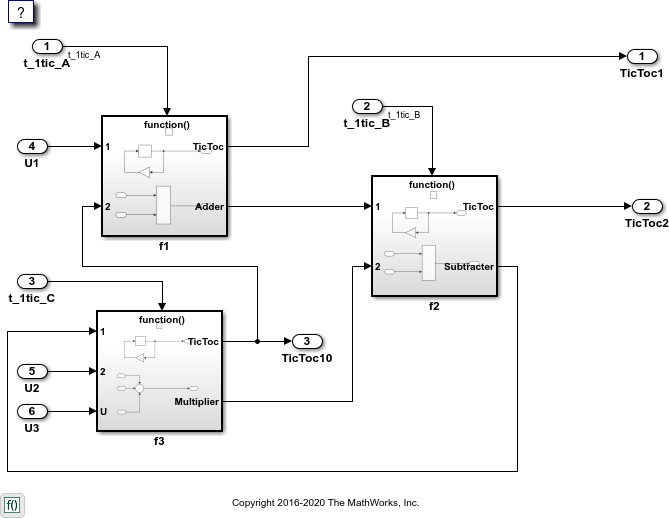
* Inport
* Outport
* Function-Call Subsystem
* Goto
* From
* Merge

**注意：**若要导出的 Function-Call Subsystem 采用 C++ 类接口，则不支持 Simulink Function 模块。

代码生成器为 Function-Call Subsystem 模块生成函数代码。对于 Function-call Subsystem 模块，您需要将模块输入端口连接到用于断言函数调用信号的根 Inport 模块。子系统会根据接收到的函数调用信号而执行。

模型 **rtwdemo\_cppclass\_functions** 包含用于导出函数的函数调用子系统 **f1**、**f2** 和 **f3**。

**open\_system('rtwdemo\_cppclass\_functions')**

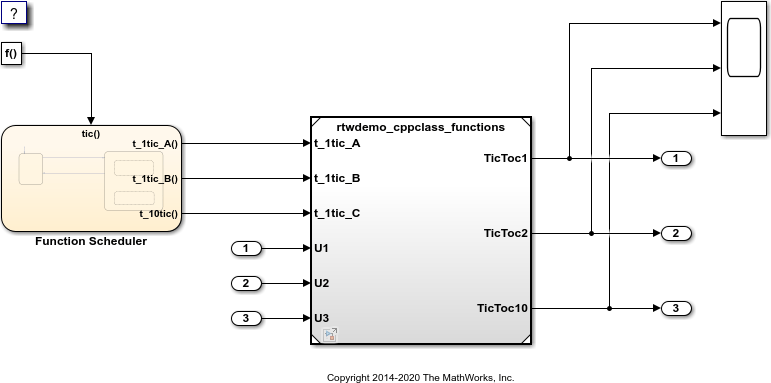


#### 创建用于仿真的测试框架模型

导出函数时，生成的代码不包含调度器。创建一个测试框架模型来处理仿真期间的调度。不要使用该测试框架模型来生成您部署的代码。

模型 **rtwdemo\_cppclass\_export\_functions** 是一个测试框架。该模型向此示例中的其他模型提供函数调用信号以调度模型内容。

**open\_system('rtwdemo\_cppclass\_export\_functions')**



#### 对测试框架模型进行仿真

通过对测试框架模型进行仿真，验证包含要导出的函数的模型是否按预期执行。例如，对

#### rtwdemo\_cppclass\_export\_functions 进行仿真。

**sim('rtwdemo\_cppclass\_export\_functions')**

#### 生成函数代码和报告

为要导出的函数生成代码和代码生成报告。例如，为 **rtwdemo\_cppclass\_functions** 生成代码。

**slbuild('rtwdemo\_cppclass\_functions')**

**### Starting build procedure for: rtwdemo\_cppclass\_functions**

**### Successful completion of build procedure for: rtwdemo\_cppclass\_functions Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_cppclass\_functions Code generated and compiled. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 25.009s**

#### 查看生成的代码

从代码生成报告中，查看生成的代码。

* **ert\_main.cpp** 是模型的示例主程序（执行框架）。此代码显示如何调用导出的函数。该代码还显示如何初始化和执行生成的代码。
* **rtwdemo\_cppclass\_functions.cpp** 调用初始化函数（包括 **Initialize Function**）和针对模型子系统组件 **f1**、**f2** 和 **f3** 导出的函数。
* **rtwdemo\_cppclass\_functions.h** 声明模型数据结构体和一个对接导出的入口函数和数据结构体的公共接口。
* **rtwtypes.h** 定义生成的代码所需的数据类型、结构体和宏。

#### 编写接口代码

打开并查看代码接口报告。要编写用于您的执行框架的接口代码，请使用该报告中的信息。

1. 通过添加指令 **#include rtwdemo\_cppclass\_functions.h** 和 **#include rtwtypes.h** 来包含生成的头文件。
2. 将输入数据写入模型的 Inport 模块的生成代码。
3. 调用生成的入口函数。
4. 从模型 Outport 模块的生成代码中读取数据。

输入端口：

* **rtU.U1** 的类型为 **real\_T**，维度为 1
* **rtU.U2** 的类型为 **real\_T**，维度为 1
* **rtU.U3** 的类型为 **real\_T**，维度为 1

入口函数：

* 初始化入口函数 **void initialize(void)**。在启动时，调用一次此函数。
* 导出的函数 **void t\_1tic\_A(void)**。根据需要调用此函数。
* 导出的函数 **void t\_1tic\_B(void)**。根据需要调用此函数。
* 导出的函数 **void t\_1tic\_C(void)**。根据需要调用此函数。

输出端口：

* **rtY.TicToc1** 的类型为 **int8\_T**，维度为 [2]
* **rtY.TicToc2** 的类型为 **int8\_T**，维度为 [2]
* **rtY.TicToc10** 的类型为 **int8\_T**，维度为 1

#### 有关详细信息

* “生成要导出到外部代码库的组件源代码” (Embedded Coder)
* “Deploy Applications to Target Hardware” (Embedded Coder)
* “Customize Code Organization and Format” (Embedded Coder)

#### 关闭示例模型

**bdclose('rtwdemo\_cppclass\_export\_functions') bdclose('rtwdemo\_cppclass\_functions')**

### 为导出函数子系统生成代码

* [“指定自定义初始化函数名称” （第 30-25 页）](#_bookmark259)
* [“指定自定义描述” （第 30-26 页）](#_bookmark260)
* [“优化为导出函数子系统生成的代码” （第 30-26 页）](#_bookmark261)

要为导出函数子系统生成代码，请执行以下操作：

1. 验证您正在为其生成代码的子系统是否满足导出要求 (Embedded Coder)。
2. 在“配置参数”对话框中，执行以下操作：
   1. 将参数**系统目标文件**设置为基于 ERT 的系统目标文件，如 **ert.tlc**。
   2. 如果您需要包含生成代码的 SIL 模块以用于验证目的，请将模型配置参数**创建模块**设置为 “**SIL**”。
   3. 点击**确定**或**应用**。
3. 右键点击子系统模块，并从上下文菜单中选择 **C/C++ 代码 > 导出函数**。

该操作创建并编译一个新模型 **subsystem.slx**，其中包含原始子系统的内容，并创建包含 Model 模块的 **ScratchModel**。此模块引用新创建的 **subsystem.slx** 模型。

代码生成器生成代码并将其放在工作文件夹中。

如果在步骤 2b 中将**创建模块**设置为 “**SIL**”，Simulink 将打开一个新窗口，其中包含表示生成代码的 S-Function 模块。此模块与原始子系统具有相同的大小、形状和连接线。

代码生成和模块创建现已完成。您可以像对待生成的 ERT 代码和 S-Function 模块一样，测试和使用这些[代码和模块（可选）。有关可选的工作流任务的信息，请参阅“指定自定义初始化函数名称” （第 30-](#_bookmark259)

[25](#_bookmark259) [页）和“指定自定义描述” （第 30-26](#_bookmark260) 页）。

#### 指定自定义初始化函数名称

您可以为导出函数的初始化函数指定自定义名称，以作为 **slbuild** 命令的参数。该命令为以下形式：

**blockHandle = slbuild('subsystem', 'Mode', 'ExportFunctionCalls',.. 'ExportFunctionInitializeFunctionName', 'fcnname')**

**fcnname** 指定函数名称。例如，如果您指定名称 **'myinitfcn'**，编译过程将生成类似以下内容的代码：

**/\* Model initialize function \*/ void myinitfcn(void){**

**...**

**}**

#### 指定自定义描述

您可以使用 Inport 模块的“模块属性”对话框输入导出函数的自定义说明。

1. 对要导出代码的子系统，右键点击驱动其控制端口的 Inport 模块。
2. 选择**属性**。
3. 在**常规**选项卡的**描述**字段中，输入您的描述性文本。

在函数导出过程中，您输入的文本将输出到 Inport 模块的生成代码的头部中。例如，如果您打开示例程序 **rtwdemo\_exporting\_functions** 并在端口 **t\_1tic\_A** 的“模块属性”对话框中输入描述，代码生成器将生成类似以下内容的代码：

**/\***

* + **Output and update for exported function: t\_1tic\_A**

**\***

* + **My custom description of the exported function**

**\*/**

**void t\_1tic\_A(void)**

**{**

**...**

**}**

#### 优化为导出函数子系统生成的代码

要优化为导出函数子系统生成的代码，请为每个跨子系统边界的输入信号和输出信号指定单独的存储类。对于要导出的每个函数调用子系统，请执行以下操作：

1. 右键点击子系统。
2. 从上下文菜单中，选择**模块参数(子系统)**。
3. 选择**代码生成**选项卡。
4. 将**函数打包**设置为“自动”。
5. 点击**确定**或**应用**。

### 另请参阅详细信息

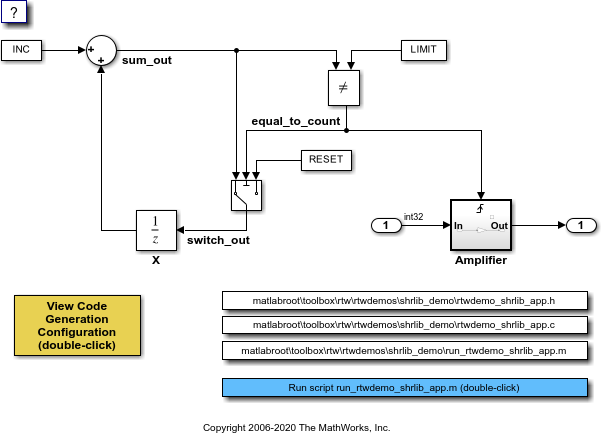
* + “Map Embedded System Architecture to Simulink Modeling Environment” (Embedded Coder)
  + “使用触发子系统”
  + “使用函数调用子系统”
  + “导出函数模型概述”

## 通过使用共享库对接在开发计算机上运行的仿真器

此示例生成一个共享库，用于对接在开发计算机上运行的仿真器。此示例使用系统目标文件

**ert\_shrlib.tlc** 来生成该共享库。

要从模型编译共享库并在应用程序中使用该库，请执行下列操作： 1.开发您的模型。对于此示例，请打开模型 **rtwdemo\_shrlib**。将模型的副本保存到可写位置。 **open\_system('rtwdemo\_shrlib');**



该模型是单速率的离散时间模型。8 位计数器为名为 **Amplifier** 的触发子系统馈送信号。参数 **INC**、 **LIMIT** 和 **RESET** 分别设置为常量值 1、4 和 0。当信号 **equal\_to\_count** 为 true 时，子系统按增益因子 **K=3** 放大其输入信号，并更新输出信号。

1. 打开 **Embedded Coder**。
2. 将模型配置参数**系统目标文件**设置为 **ert\_shrlib.tlc**。
3. 编译共享库文件。代码生成器生成的文件取决于您的开发平台。例如，在 Windows➅ 系统上，代码生成器生成库文件 **rtwdemo\_shrlib\_win64.dll**。

5.创建使用共享库的应用程序代码。此示例使用下列文件中提供的应用程序代码：

#### matlabroot\toolbox\rtw\rtwdemos\shrlib\_demo\rtwdemo\_shrlib\_app.h matlabroot

**\toolbox\rtw\rtwdemos\shrlib\_demo\rtwdemo\_shrlib\_app.c**

要查看这些文件中的源代码，请在模型中点击 **.h** 和 **.c** 文件的白色按钮。 6.编译并链接文件应用程序和共享库文件以生成可执行程序。以下脚本编译和运行程序。

#### matlabroot\toolbox\rtw\rtwdemos\shrlib\_demo\run\_rtwdemo\_shrlib\_app.m

要查看脚本代码，请在模型中点击 **.m** 文件的白色按钮。

要编译模型并运行使用生成的共享库的应用程序，请在模型中双击蓝色按钮。

有关使用共享库的详细信息，请参阅 “将生成的代码打包为共享库” (Embedded Coder)。

### 另请参阅详细信息

* “Generate Shared Library for Export to External Code Base” (Embedded Coder)

# Simulink Coder 中的程序编译、交互和调试

* [“编译从 Simulink 模型生成的代码的方法” （第 31-2 页）](#_bookmark264)
* [“配置工具链 (ToolchainInfo) 或模板联编文件编译过程” （第 31-3 页）](#_bookmark265)
* [“转移或共享生成的代码” （第 31-13 页）](#_bookmark272)
* [“使用 Microsoft Visual Studio 编译和调试生成的 C 代码” （第 31-17 页）](#_bookmark281)

## 编译从 Simulink 模型生成的代码的方法

使用以下方法之一来编译（编译和链接）从 Simulink 模型生成的代码：

* CMake - CMake 是用于编译过程管理的第三方开源工具，它使用配置 (CMakeLists.txt) 文件来生成用于本机编译环境的标准编译文件，例如联编文件、Ninja 文件或 Microsoft Visual Studio➅ 和 Xcode项目。您可以指定附带的 CMake 工具链定义，或使用 **target** 包提供自定义 CMake 工具链定义。请参阅“Create Custom CMake Toolchain Definition”。

MATLAB 附带 CMake 可执行文件。Simulink Coder 和 Embedded Coder 支持从 3.12.0 开始的 CMake 版本。

* **ToolchainInfo** - 编译过程生成联编文件并支持自定义工具链。您可以使用 MATLAB 脚本定义的工具链信息对象来控制编译过程。
* 模板联编文件 - 将模板联编文件与您指定的工具链结合使用的编译过程。您也可以通过工具链信息对象来控制编译过程。

每种编译方法都支持：

* Simulink 模型配置参数 - 请参阅“Configure CMake Build Pr[ocess”和“配置工具链](#_bookmark265) [(ToolchainInfo) 或模板联编文件编译过程” （第 31-3](#_bookmark265) 页）。

#### slbuild、Ctrl+B 和 codebuild

* “SIL 和 PIL 仿真” (Embedded Coder)
* “用于参数调节、信号监控和代码执行探查的外部模式仿真” (Embedded Coder)

有关自定义编译方法的信息，请参阅“编译过程自定义”。

### 另请参阅

**slbuild** | **codebuild**

### 相关示例

* “Configure CMake Build Process”
* “Create Custom CMake Toolchain Definition”
* [“配置工具链 (ToolchainInfo) 或模板联编文件编译过程” （第 31-3 页）](#_bookmark265)
* “Compile Code in Another Development Environment”
* “Build Library or Executable from AUTOSAR Adaptive Model” (AUTOSAR Blockset)

### 外部网站

* <https://cmake.org/>

## 配置工具链 (ToolchainInfo) 或模板联编文件编译过程

本主题说明如何指定工具链 (**ToolchainInfo**) 或模板联编文件方法来编译从模型生成的代码。如果将 System target file 配置参数设置为：

* “**ert.tlc**”、“**ert\_shrlib.tlc**”、“**grt.tlc**” 或任何与工具链兼容的系统目标文件，则编译过程将使[用工具链方法 （第 31-3](#_bookmark266) 页）。
* 任何不符合工具链的系统目标文件，编译过程将使用模板联编文件方法 (Embedded Coder)。您可以使用以下命令从工具链方法切换到模板联编文件方法：

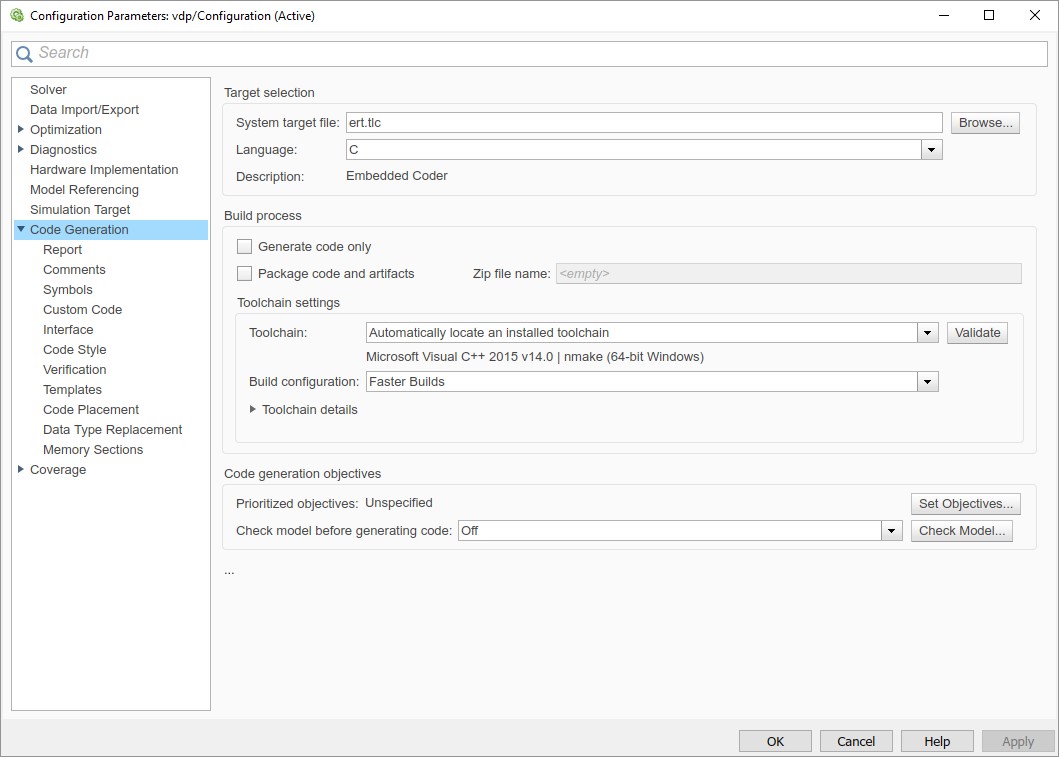
**set\_param(model, 'MakeCommand', 'make\_rtw TMF=1')**

### 工具链方法

当您将**系统目标文件**设置为以下项时，**工具链设置**将出现在**编译过程**下：

* “**grt.tlc —** 一般实时目标”
* “**ert.tlc — Embedded Coder**”（需要 Embedded Coder 产品）
* “**ert\_shrlib.tlc — Embedded Coder (**基于主机的共享库目标**)**”（需要 Embedded Coder）
* 任何与工具链兼容的系统目标文件（如果基于 ERT，则需要 Embedded Coder）

有关与工具链兼容的系统目标文件的详细信息，请参阅“Support Toolchain Approach with Custom Target”。



**工具链设置**包括以下配置参数：

* Toolchain - 指定一组第三方软件工具，用来编译生成的代码。工具链可以包括编译器、链接器和存档器以及其他预编译或编译后工具。

**工具链**的默认值为“自动定位已安装的工具链”。**工具链**参数显示“自动定位已安装的工具链”下方的工具链的名称。

要检查工具链是否存在，并验证代码生成器是否具有使用工具链所需的信息，请在**高级参数**部分中，点击**验证工具链**。生成的验证报告将指出所选工具链的通过/失败值，并显示需要解决的问题。

* Build configuration - 指定优化设置。默认情况下，**编译配置**设置为“快速编译”。您还可以选择“快速运行”、“调试”和“指定”。如果选择“指定”，您可以自定义工具链的选项。当您点击**应用**时，自定义工具链设置将应用于当前模型。

**注意** 使用模板联编文件方法的以下系统目标文件与使用工具链方法的系统目标文件名称相同但描述不同：

* “**ert.tlc —** 为 **Embedded Coder** 创建 **Visual C/C++** 解决方案文件”
* “**grt.tlc —** 为 **Simulink Coder** 创建 **Visual C/C++** 解决方案文件”

为了避免混淆，请点击**浏览**选择系统目标文件，并查看每个文件的描述。

### 升级模型以使用工具链方法

当您打开在 R2013b 之前创建的、使用以下系统目标文件的模型时，软件会尝试升级模型。升级将配置从使用模板联编文件设置更改为使用工具链设置：

#### “ert.tlc — Embedded Coder”

* “**ert\_shrlib.tlc — Embedded Coder (**基于主机的共享库目标**)**”
* “**grt.tlc —** 一般实时目标”

**注意** 要将使用自定义系统目标文件的模型升级为使用工具链方法，请参阅“Support Toolchain Approach with Custom Target”。

某些模型配置参数值会阻止软件将模型升级为使用工具链设置。以下说明指导您如何完成升级过程。

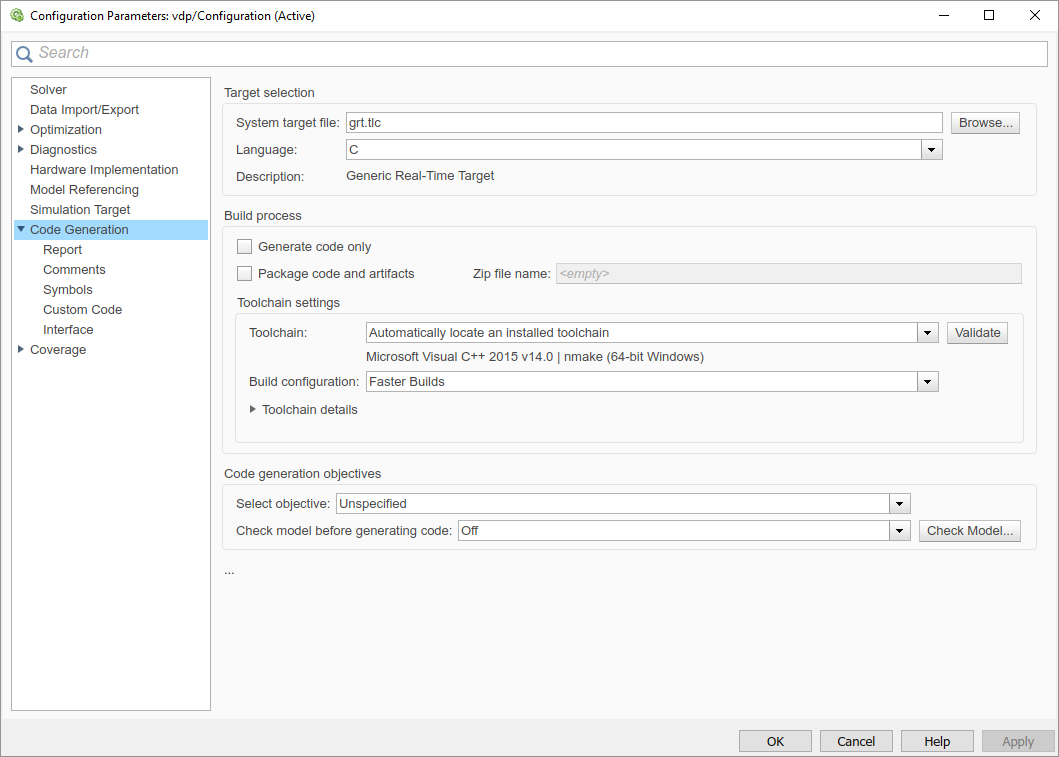
假设您要升级模型并使用工具链编译方法。这么做并不是必须的，您仍然可以从未升级的模型中生成代码。

**注意** 软件不会升级使用以下系统目标文件的模型：

* “**ert.tlc —** 为 **Embedded Coder** 创建 **Visual C/C++** 解决方案文件”
* “**grt.tlc —** 为 **Simulink Coder** 创建 **Visual C/C++** 解决方案文件”

要判断模型是否已升级，有如下方法：

1. 如果指定了**工具链**和**编译配置**配置参数，则该模型已升级。



如果指定了联编文件配置参数，如**生成联编文件**、**Make 命令**和**模板联编文件**，则该模型尚未升级。首先，使用**文件** > **另存为**为模型创建一个工作副本。此操作将保留原始模型和配置参数以供参考。

尝试使用升级顾问升级模型：

1. 在**建模**选项卡的**评估和管理**部分中，选择**升级顾问**。
2. 在升级顾问中，选择**检查并更新模型，以使用工具链方法编译生成的代码**，然后点击**运行此检查**。
3. 执行建议的操作和/或点击**更新模型**。

如果您不能使用升级顾问升级模型，则以下一个或多个参数未设置为如下默认值：

* **编译器优化级别** - “关闭优化**(**编译速度更快**)**”
* **生成联编文件** - 启用
* **模板联编文件** - 系统目标文件特定的模板联编文件
* **Make 命令** - **make\_rtw**（不带参数）

有时，模型确实不能升级。请尝试以下步骤：

* 如果**生成联编文件**被禁用，这种情况下则不能升级。但是，您可以尝试启用它，然后尝试使用升级顾问升级模型。
* 如果**编译器优化级别**设置为“打开优化**(**运行速度更快**)**”：
  1. 将**编译器优化级别**设置为“关闭优化**(**编译速度更快**)**”。
  2. 使用升级顾问升级模型。
  3. 将**编译配置**设置为“快速运行”。
* 如果**编译器优化级别**设置为“自定义”：

1. 将**自定义编译器优化标志**复制到文本文件中。
2. 将**编译器优化级别**设置为“关闭优化**(**编译速度更快**)**”。
3. 使用升级顾问升级模型。
4. 将**编译配置**设置为“指定”。
5. 要执行相同的优化，请编辑编译器选项。

* 如果**模板联编文件**使用自定义模板联编文件，这种情况下则不能升级。但是，您可以尝试以下步骤：

1. 更新**模板联编文件**以使用系统目标文件的默认联编文件。

**注意** 要获取默认的联编文件名称，请更改**系统目标文件**，点击**应用**，再更改回原来的名称，然后再次点击**应用**。

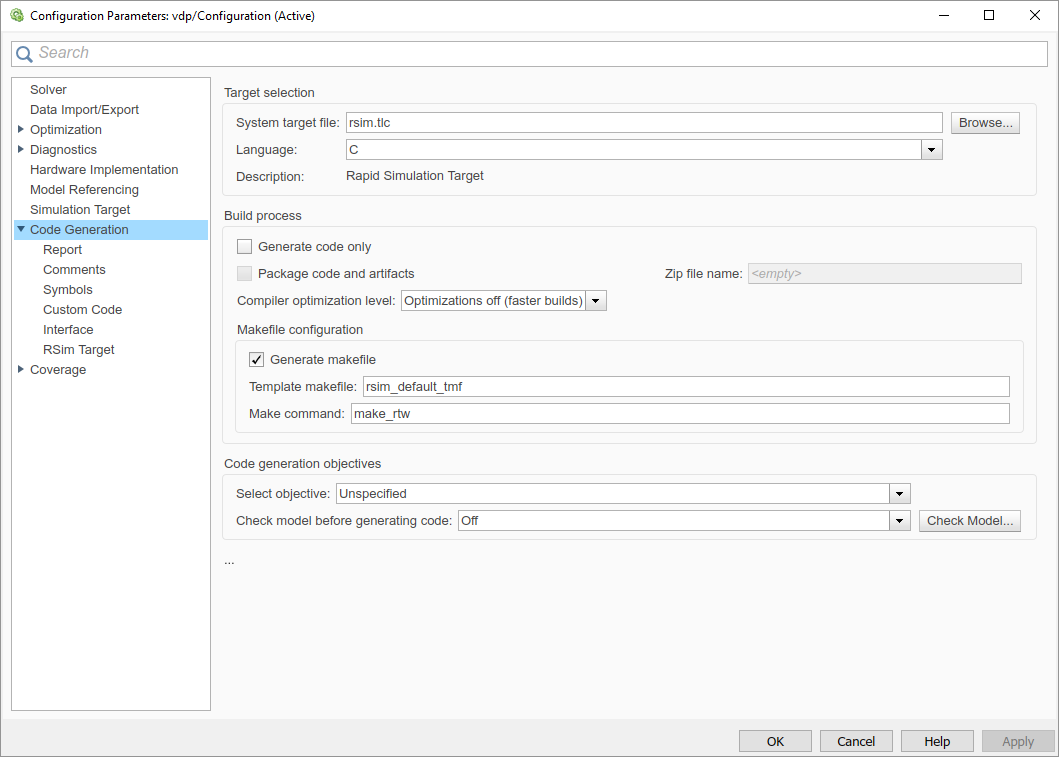
1. 使用升级顾问升级模型。
2. 如果模板联编文件包含编译工具选项（如编译器优化标志），请将**编译配置**设置为“指定”并更新选项。
3. 如果模板联编文件使用自定义编译工具，请创建并注册自定义工具链，如“Custom Toolchain Registration” 中所述。然后将**工具链**参数设置为使用自定义工具链。

**注意** 注册自定义工具链后，更新**工具链**以使用自定义工具链。

1. 如果模板联编文件包含自定义规则和逻辑，则这些自定义不能应用于已升级的模型。

### 模板联编文件方法

当**系统目标文件**设置为使用模板联编文件方法的 “**tlc**” 文件时，软件显示**编译器优化级别**、**生成联编文件**、**Make 命令**和**模板联编文件**参数。



#### 指定是否生成联编文件

**生成联编文件**选项指定编译过程是否为模型生成联编文件。默认情况下，编译过程会生成联编文件。但 是，举例来说，如果要支持不是基于联编文件的自定义编译处理，则可以取消选择**生成联编文件**，禁止生成联编文件。如果取消选择此参数：

* **Make 命令**和**模板联编文件**选项将不可用。
* 使用用户定义的命令来设置生成代码之后的编译处理，如“Customize Post-Code-Generation Build Processing”中所述。

#### 指定 Make 命令

每个基于模板联编文件的系统目标文件都有一个关联的 **make** 命令。代码生成器使用此内部 MATLAB 命令来控制编译过程。当您启动编译过程时，此命令将出现在 **Make 命令**字段中并开始运行。

大多数系统目标文件使用默认命令 **make\_rtw**。第三方系统目标文件可能提供了其他 **make** 命令。请参阅供应商提供的文档。

除了 **make** 命令的名称外，还可以在 **Make 命令**字段中提供联编文件选项。这些选项可能包括编译器特定的选项，包括路径和其他参数。当编译过程调用 **make** 工具函数时，会通过 **make** 命令行来传递这些选 项，将它们添加到传递给编译器的全部标志中。

“Template Makefiles and Make Options”列出了每个支持的编译器可以使用的 **Make 命令**选项。**指定模板联编文件**

**模板联编文件**字段具有以下功能：

* 如果您使用系统目标文件浏览器选择了系统目标文件，此字段将显示 MATLAB 语言文件的名称，为您的开发环境选择一个模板联编文件。例如，在“模型配置参数：代码生成”中，**模板联编文件**字段显示 **grt\_default\_tmf**，指示编译过程调用 **grt\_default\_tmf.m**。

“Template Makefiles and Make Options”详细介绍了编译过程选择模板联编文件所依据的逻辑。

* 您也可以在此字段中显式输入特定模板联编文件的名称（包括扩展名），或者输入用于返回模板联编文件的 MATLAB 语言文件的名称。如果您要使用系统目标文件浏览器中没有出现的系统目标文件，请使用此方法。例如，如果您为自定义系统目标文件编写了自己的模板联编文件，请使用此方法。

如果您指定自己的模板联编文件，请确保包含文件扩展名。如果您省略扩展名，编译过程将尝试查找并执行具有 **.m** 扩展名的文件（即 MATLAB 语言文件）。模板联编文件（或返回模板联编文件的 MATLAB 语言文件）必须位于 MATLAB 路径中。要确定该文件是否在 MATLAB 路径中，请在 MATLAB 命令行窗口输入以下命令：

**which tmf\_filename**

#### 将模板联编文件与工具链相关联

为模板联编文件编译过程指定工具链。以下示例提供最小工具链的定义，并说明如何将工具链与作为

**ert\_unix.tmf** 或 **ert\_vcx64.tmf** 副本的模板联编文件相关联。

1. 使用 **ert\_unix.tmf** 或 **ert\_vcx64.tmf** 的副本作为模板联编文件。

**copyfile(fullfile(matlabroot, 'toolbox', 'coder', ... 'compile', 'tmf', 'ert\_vcx64.tmf'), ... 'ert\_copy.tmf')**

1. 在模板联编文件（即 **ert\_copy.tmf**）中，编辑 **TOOLCHAIN\_NAME** 宏。

**TOOLCHAIN\_NAME = "Minimal Toolchain for TMF Build"**

**注意** 与 R2018a 之前版本的对应文件相比，当前版本的 **ert\_unix.tmf** 和 **ert\_vcx64.tmf** 文件已简化。软件通过检查变量 **TOOLCHAIN\_NAME** 是否存在来确定模板联编文件是否使用了此简化方

法。如果模板联编文件是基于 R2018a 之前版本的 **ert\_unix.tmf** 和 **ert\_vcx64.tmf** 文件，则没有指定 **TOOLCHAIN\_NAME** 变量。

此简化方法通过 **|>MODEL\_MODULES<|** 标记来指定 **main**、**model** 和其他需要编译的源文件。如果模板联编文件是基于 R2018a 之前版本的 **ert\_unix.tmf** 和 **ert\_vcx64.tmf** 文件，则会将 **main**和 **model** 源文件显式添加到需要编译的源文件列表中。

要更新源于 R2018a 之前版本的模板联编文件，请执行以下操作：

* 1. 删除对 **ert\_main.c**、**classic\_main.c** 或 **rt\_main.c** 的引用。例如，像 **ert\_main.c** 这样的字面值引用或像 **ert\_main.$(TARGET\_LANG\_EXT)** 这样的标记化引用。

#### 删除对 $(MODEL).c 或 $(MODEL).$(TARGET\_LANG\_EXT) 的引用。

1. 将工具链与模板联编文件相关联。
   1. 创建一个工具链定义文件。对于此示例，请使用以下代码：

#### 工具链定义文件

**function tc = minimalToolchainForTMF**

**% Create a toolchain object**

**tc = coder.make.ToolchainInfo('Name', 'Minimal Toolchain for TMF Build');**

**% Specify that the linker requires a response file instead of**

**% including all object file modules on the linker command line tc.addAttribute('RequiresCommandFile', true);**

**if ispc**

**objExt = '.obj'; else**

**objExt = '.o'; end**

**% Specify source file and object file extension used by the C compiler tool = tc.getBuildTool('C Compiler');**

**tool.setFileExtension( 'Source', '.c'); tool.setFileExtension( 'Header', '.h'); tool.setFileExtension( 'Object', objExt);**

**% Specify source file and object file extension used by the C++ compiler tool = tc.getBuildTool('C++ Compiler');**

**tool.setFileExtension( 'Source', '.cpp'); tool.setFileExtension( 'Header', '.hpp'); tool.setFileExtension( 'Object', objExt);**

* 1. 运行工具链定义文件，这将生成 **ToolchainInfo** 对象 **tc**。

**tc = minimalToolchainForTMF;**

* 1. 将 **ToolchainInfo** 对象保存到 MAT 文件中。

**save('tcMinimal','tc')**

* 1. 在 **RTW.TargetRegistry** 中注册工具链。
     1. 将以下代码放在 **rtwTargetInfo.m** 文件中。

**function rtwTargetInfo(tr) tr.registerTargetInfo(@loc\_createToolchain); end**

**function config = loc\_createToolchain config(1) = coder.make.ToolchainInfoRegistry;**

**config(1).Name = 'Minimal Toolchain for TMF Build'; config(1).FileName = fullfile(fileparts(mfilename('fullpath')),...**

**'tcMinimal.mat'); config(1).TargetHWDeviceType = {'\*'}; config(1).Platform = {computer('arch')}; end**

* + 1. 将该文件保存在当前工作文件夹中或保存在 MATLAB 搜索路径上的文件夹中。
    2. 重置 **TargetRegistry**。

**RTW.TargetRegistry.getInstance('reset');**

1. 将模板联编文件与您的模型相关联并编译模型。

**set\_param(model, 'TemplateMakefile', 'ert\_copy.tmf'); slbuild(model);**

### 指定 TLC 以用于代码生成

目标语言编译器 (TLC) 是代码生成器的重要组成部分。它使您能够自定义生成的代码。通过自定义，您可以生成特定于平台的代码，也可以融入您自己的算法更改，以改善执行速度、代码大小或保留与现有方法的兼容。有关详细信息，请参阅“Target Language Compiler Overview”。

您为代码生成指定的 TLC 选项显示在生成的 HTML 代码生成报告的摘要部分。

**注意** 指定 TLC 命令行选项不会在 make 命令行中添加标志。

您可以在 **set\_param** 函数调用中使用模型参数 **TLCOptions** 指定进行代码生成的目标语言编译器 (TLC)命令行选项和参量。例如，

**>> set\_param(gcs,'TLCOptions','-p0 -aWarnNonSaturatedBlocks=0')**

TLC 选项的一些常见用法包括：

* **-aVarName=1** 用于声明 TLC 变量和/或为其赋值
* **-IC:\Work** 用于指定包含路径
* **-v** 用于获取 TLC 处理的详细输出（例如，在进行调试时）

### 在生成代码的重新编译中使用校验和

当您在 **GenerateMakefile** 设置为 **'on'** 和 **GenCodeOnly** 设置为 **'off'** 的状态下编译模型时，软件会生成一个联编文件，并运行该联编文件来编译生成的代码。编译完成后，软件会创建并保存源文件和配置数据的校验和。

如果在编译模型时可以使用上一个版本的校验和，则：

1. 软件使用校验和来确定源文件和配置数据是否与之前的编译不同。
2. 如果软件检测到差异，则软件会：
   * 重新生成联编文件。
   * 重新编译生成的代码。

如果编译过程使用模板联编文件，您可以通过将以下文本添加到模板联编文件来禁用校验和：

**DO\_NOT\_USE\_CHECKSUMS=1**

在这种情况下，软件使用文件时间戳来确定是否应重新编译生成的代码。

### 另请参阅详细信息

* [“编译从 Simulink 模型生成的代码的方法” （第 31-2 页）](#_bookmark264)
* “Support Toolchain Approach with Custom Target”
* “Build Process Workflow for Real-Time Systems”
* “Custom Toolchain Registration”
* “Register Custom Toolchain and Build Executable”
* “Template Makefiles and Make Options”
* “Target Language Compiler Overview”
* “Executable Program Generation”
* “Configure CMake Build Process”

## 转移或共享生成的代码

要转移或共享静态代码和生成的代码文件，您可以使用 **packNGo** 函数。例如，将生成的代码转移到不提供 MATLAB 和 Simulink 产品的开发环境中时，请使用该函数。

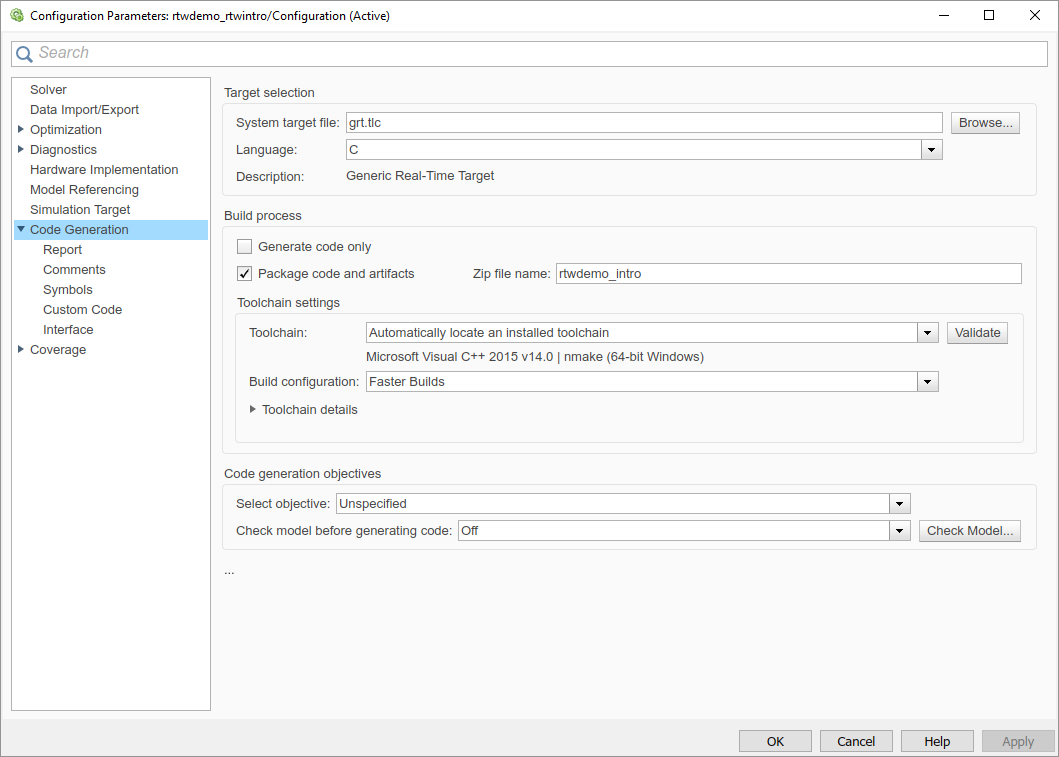
### 打包并转移生成的代码

要打包生成的代码，请执行以下操作：

1. 在“配置参数”系统对话框中，选择**代码和工件打包**。此选项将编译过程配置为在代码生成后运行

**packNGo** 函数。

1. 在 **Zip 文件名**字段中，输入要在其中打包生成的代码和工件的 ZIP 文件的名称。指定文件名时可以使用 **.zip** 扩展名，也可以不使用。如果您没有指定扩展名或指定 **.zip** 以外的扩展名，ZIP 实用工具会添加 **.zip** 扩展名。如果您未指定值，则编译过程将使用名称 **model.zip**，其中 **model** 是顶层模型的名称。



1. 应用此更改，然后为您的模型生成代码。
2. 使用 ZIP 工具检查生成的 ZIP 文件，以验证打包的代码是否已准备好进行转移。

将 ZIP 文件转移到目标开发环境中，并使用 ZIP 工具来解包该文件。要编译解包后的代码，请使用编译工具。

如果目标环境提供 Simulink Coder 或 Embedded Coder，可以使用 **codebuild** 来编译解包后的代码。

### 从命令行打包并转移生成的代码

要使用行命令打包和转移生成的代码，请使用以下工作流：

1. [选择 **zip** 文件的结构。 （第 31-14 页）](#_bookmark275)
2. [选择 **zip** 文件的名称。 （第 31-14 页）](#_bookmark276)
3. [将模型代码文件打包在一个 **zip** 文件中。 （第 31-14 页）](#_bookmark277)
4. [检查生成的 **zip** 文件。 （第 31-15 页）](#_bookmark278)
5. [转移并解压缩 **zip** 文件。 （第 31-15 页）](#_bookmark279)

#### 选择 ZIP 文件的结构

在为模型编译生成和打包文件之前，请决定将文件打包为扁平文件夹结构还是分层文件夹结构。默认情况下，**packNGo** 函数将文件打包在单一的扁平文件夹结构中。

|  |  |
| --- | --- |
| **如果...** | **则使用...** |
| 您将文件转移到不使用生成的联编文件的 IDE，或者代码不依赖于所需静态文件的相对位置 | 单一、扁平文件夹结构 |
| 目标开发环境必须保持与源环境相同的文件夹结 构，因为它使用生成的联编文件，或者代码依赖于文件的相对位置 | 分层结构 |

如果使用层次结构，则 **packNGo** 函数会创建两个层级的 **zip** 文件，即一个主 **zip** 文件包含以下次级 **zip**

文件：

* **mlrFiles.zip** - 您的 **matlabroot** 文件夹树中的文件
* **sDirFiles.zip** - 您的编译文件夹中的文件，您在该文件夹中为模型启动了代码生成
* **otherFiles.zip** - 不在 **matlabroot** 或 **start** 文件夹树中的必需文件

次级 **zip** 文件的路径相对于主 **zip** 文件的根文件夹，文件夹结构与源开发文件夹结构相同。

#### 选择 Zip 文件的名称

默认情况下，**packNGo** 函数将主 **zip** 文件命名为 **model**。您可以选择指定不同名称。如果指定文件名并省略文件类型扩展名，则该函数会将 **.** 追加到您指定的名称。

#### 在 Zip 文件中打包模型代码

使用 **PostCodeGenCommand** 配置参数、**packNGo** 函数和模型的编译信息对象对模型代码文件进行打包。您可以设置打包操作以使用：

* 系统生成的编译信息对象。

如果是这种情况，则在生成模型代码之前，请使用 **set\_param** 将配置参数

**PostCodeGenCommand** 设置为显式调用 **packNGo** 函数。例如：

**set\_param(bdroot, 'PostCodeGenCommand', 'packNGo(buildInfo);');**

在生成模型代码并写入磁盘后、但在生成联编文件前，此命令指示编译过程计算对 **packNGo** 的调用。此命令为当前选定的模型使用系统生成的编译信息对象。

* 以编程方式构造的编译信息对象。

在这种情况下，可使用其他编译信息函数有选择地将路径和文件包含在您随后使用 **packNGo** 函数指定的编译信息对象中。例如：

**.**

**.**

**.**

**myModelBuildInfo = RTW.BuildInfo; addSourceFiles(myModelBuildInfo, {'test1.c' 'test2.c' 'driver.c'});**

**.**

**.**

**.**

**packNGo(myModelBuildInfo);**

以下示例说明如何更改 **packNGo** 的默认行为。

|  |  |
| --- | --- |
| **如需执行以下操作...** | **指定...** |
| 将文件打包的结构更改为分层结构 | **packNGo(buildInfo, 'packType' 'hierarchical');** |
| 重命名主 **zip** 文件 | **packNGo(buildInfo, 'fileName' 'zippedsrcs');** |
| 将文件打包的结构更改为分层结构并重命名主  **zip** 文件 | **packNGo(buildInfo, 'packType' 'hierarchical'... 'fileName' 'zippedsrcs');** |
| 在 **zip** 文件中包括包含路径上的头文件 | **packNGo(buildInfo, 'minimalHeaders' false);** |
| 为解析错误和缺失文件生成警告 | **packNGo(buildInfo, 'ignoreParseError' true... 'ignoreFileMissing' true);** |

**注意 packNGo** 函数可以修改作为函数的第一个参数传递的 **RTW.BuildInfo** 对象中的编译信息。该函数可能会根据存储在模型的编译信息中的源和包含路径找到其他文件，并且将该信息添加到 **RTW.BuildInfo** 对象中。

#### 检查生成的 ZIP 文件

要验证生成的 ZIP 文件是否已准备好进行转移，请使用 ZIP 工具检查该文件。一些 ZIP 工具允许您在不解包文件的情况下查看文件内容。如果模型代码文件已打包为分层结构且需要解包，则需要解包主 ZIP 文件和次级 ZIP 文件。当您解包次级 ZIP 文件时，文件的相对路径将保留。

#### 转移并解包 ZIP 文件

将 ZIP 文件转移到目标开发环境中，并使用 ZIP 工具来解包该文件。要编译解包后的代码，请使用编译工具。

如果目标环境提供 Simulink Coder 或 Embedded Coder，可以使用 **codebuild** 来编译解包后的代码。

#### 代码打包示例

此示例说明如何对为示例模型 **rtwdemo\_rtwintro** 生成的代码文件进行打包：

1. 将您的工作文件夹设置为可写文件夹。
2. 打开模型 **rtwdemo\_rtwintro** 并在您的工作文件夹中保存一份副本。
3. 在命令行窗口中，输入：

**set\_param('rtwdemo\_rtwintro', 'PostCodeGenCommand',... 'packNGo(buildInfo, ''packType'' ''hierarchical'')');**

1. 为模型生成代码。
2. 检查生成的 ZIP 文件 **rtwdemo\_rtwintro.zip**。ZIP 文件包含两个次级 ZIP 文件，即 **mlrFiles.zip**

#### 和 sDirFiles.zip。

1. 检查 **mlrFiles.zip** 和 **sDirFiles.zip**。
2. 将 ZIP 文件转移到您的目标文件夹并解包。
3. 要编译解包的代码，请使用 **codebuild** 函数。

### 限制

#### packNGo 函数

有关适用于此函数的限制的信息，请参阅 **packNGo**。

#### 具有非默认扩展名的可执行文件

如果编译过程使用联编文件模板方式，则 **packNGo** 会使用链接器工具指定的可执行文件扩展名来确定需要打包的二进制工件。

如果生成具有非默认扩展名的可执行文件，请检查该扩展名是否保存在与联编文件模板关联的工具链中。有关详细信息，请参阅“将模板联编文件与工具链相关联” [（第 31-9](#_bookmark269) 页）。

如果编译过程生成的可执行文件的扩展名不同于工具链中保存的扩展名，**packNGo** 不会打包该可执行文件。

### 另请参阅

**packNGo** | **codebuild**

### 详细信息

* [“编译从 Simulink 模型生成的代码的方法” （第 31-2 页）](#_bookmark264)
* “Configure CMake Build Process”
* “Deploy Component Algorithm as Component Model Library by Using CMake” (Embedded Coder)
* “Compile Code in Another Development Environment”
* “Build Process Workflow for Real-Time Systems”
* “Executable Program Generation”

## 使用 Microsoft Visual Studio 编译和调试生成的 C 代码

此示例说明如何从模型生成代码并生成 Visual Studio➅ 解决方案。有关基本示例，请参阅

#### rtwdemo\_counter。

**关于本示例**

编译此示例模型会生成一个 Visual Studio 解决方案。以下模型配置参数设置用于控制此代码生成：

* 将**系统目标文件**设置为 **grt.tlc**。
* 将**模板联编文件**设置为 **RTW.MSVCBuild**。

当您使用此配置编译模型时，Simulink➅ Coder™ 会在 Visual Studio 解决方案中生成代码。将此解决方案添加到 Visual Studio C/C++ 项目中，以便将生成的代码与您的自定义 Visual Studio 代码集成。

#### 开始之前

* 此示例为在 Microsoft Windows➅ 平台上运行的 Microsoft➅ Visual Studio 编译代码。
* 此示例适用于 Microsoft Visual Studio，但不适用于 Visual Studio Express。
* Simulink Coder 使用 GRT 代码格式，该格式用于快速原型构建。
* Embedded Coder➅ 使用 ERT 代码格式，该格式用于生产部署。

#### 示例步骤

1. 打开示例模型 **rtwdemo\_counter\_msvc**。在命令行窗口中，键入： open\_system('rtwdemo\_counter\_msvc');。
2. 要在生成的 Visual Studio 解决方案中生成调试输出，请将模型配置参数 **Make 命令**设置为

**make\_rtw DEBUG\_BUILD=1**。（如果解决方案中不需要调试输出，请省略此步骤。）

1. 要生成代码并编译程序可执行文件，请启动编译过程。
2. 编译过程完成后，您可以看到系统生成了一个 Visual Studio 解决方案，并将该解决方案放在了

**rtwdemo\_counter\_msvc\_grt\_rtw** 文件夹下的 **msvc** 文件夹中。

1. 在 Microsoft Visual Studio 中，打开 **rtwdemo\_counter\_msvc.sln** 解决方案文件。

使用该解决方案在 Visual Studio 中编译并调试生成的代码。

### 另请参阅相关示例

* “Choose an External Code Integration Workflow”

# Simulink Coder 中的主机/目标通信

* [“用于参数调节、信号监控和代码执行探查的外部模式仿真” （第 32-2 页）](#_bookmark283)
* [“使用 TCP/IP 或串行通信的外部模式仿真” （第 32-6 页）](#_bookmark289)
* [“为 TCP/IP 或串行外部模式通信创建传输层” （第 32-39 页）](#_bookmark332)

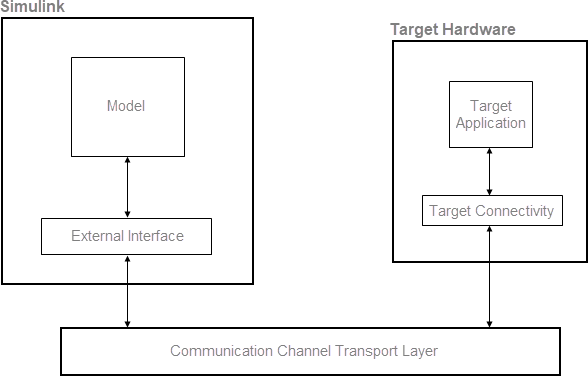
## 用于参数调节、信号监控和代码执行探查的外部模式仿真

您可以使用外部模式仿真进行快速原型构建。外部模式仿真会在开发计算机（主机）上的 Simulink 与运行由代码生成和编译过程创建的可执行文件的目标硬件之间建立通信信道。

通过该通信信道，您可以：

* 实时修改或调整模块参数。当您更改模型中的参数时，Simulink 会将新值下载到正在执行的目标应用程序。
* 监控并保存来自正在执行的目标应用程序的信号数据。
* 如果为外部模式仿真启用了代码执行探查，则检索执行时间指标。需要 Embedded Coder 许可证。

信道的低级传输层会处理消息的传输。Simulink 和生成的模型代码独立于此层。传输层及其接口代码隔离在用于格式化、传输和接收消息与数据包的单独模块中。



### 外部模式的通信机制

Simulink 支持两种外部模式仿真通信机制。要确定使用何种机制，请参阅下表中的信息。

|  |  |  |
| --- | --- | --- |
| **通信机制** | **支持** | **选择通信机制的原因** |
| XCP，通用测量标定协议 | ERT 和 GRT 系统目标文件。  一些 Simulink 支持包。有关详细信息，请参阅 [https://](https://www.mathworks.com/hardware-support.html?q&page=1) [www.mathworks.com/](https://www.mathworks.com/hardware-support.html?q&page=1) [hardware-support.html?](https://www.mathworks.com/hardware-support.html?q&page=1) [q=&page=1](https://www.mathworks.com/hardware-support.html?q&page=1)。 | XCP 外部模式：   * 使用标准通信协议。 * 只需要目标硬件上的轻量级通信软件堆栈。 * 支持 Dashboard 模块和仿真数据检查器的信号记录和流式传输。您可以流式传输来自引用模型层次结构的信号。 * 为外部模式目标连接提供记录的   **ext\_mode.h** API。   * 支持流式传输仿真数据检查器的执行时间指标，以实现基于主机的外部模式仿真和支持包外部模式仿真。   有关运行 XCP 外部模式仿真的信息，请参阅：   * “External Mode Simulation by Using XCP Communication” * “Customize XCP Server Software” |
| TCP/IP 和串行 (RS-232) | ERT、GRT 和 RSim 系统目标文件。  Simulink 支持包。 | 由于已知的限制，XCP 外部模式仿真不支持您的目标硬件。  有关运行 TCP/IP 和串行外部模式仿真的信息，请参阅：   * [“使用 TCP/IP 或串行通信的外部模式仿真” （第 32-6 页）](#_bookmark289) * [“为 TCP/IP 或串行外部模式通信创建传输层” （第 32-39 页）](#_bookmark332) |

### Simulink 功能对外部模式的支持

下表总结了对这两种外部模式仿真形式的功能支持。

|  |  |  |  |
| --- | --- | --- | --- |
| **功能** | | **XCP 支持** | **TCP/IP 和串行支持** |
| 参数调整 | 具有 Dashboard 模块 | 是 | 是 |
| 属于可调模块参数 | 是 | 是 |
| 仿真数据检查器 | | 是。在引用模型中包含信号。 | 否 |
| Logic Analyzer | | 是。在引用模型中包含信号。 | 否 |

|  |  |  |  |
| --- | --- | --- | --- |
| **功能** | | **XCP 支持** | **TCP/IP 和串行支持** |
| 接收和显示来自目标应用程序的信号的模块 | Dashboard 库 | 是 | 否 |
| Floating Scope，Scope | 是 | 是 |
| Spectrum Analyzer，Time Scope (DSP System Toolbox™) | 是 | 是 |
| Display | 是 | 是 |
| To Workspace | 是 | 是 |
| 用户编写的 S-Function。  S-Function API 内置了一种方法，该方法使用户编写的模块能够支持外部模式。  请参阅 **matlabroot/simulink/ include/simstruc.h**。 | 是 | 是 |
| Record, XY Graph | 是 | 是 |
| [信号查看子系统 （第 32-22 页）](#_bookmark313) | | 是 | 是 |

### 通过使用“在自定义硬件上运行”App 启用外部模式仿真

要在 MathWorks [支持包不支持的开发计算机或目标硬件上运行外部模式仿真，请使用“在自定义硬件上](https://www.mathworks.com/hardware-support.html?q&page=1)运行”App。对于外部模式仿真，您可以：

1. 在您的开发计算机上编译目标应用程序。
2. 将目标应用程序部署到目标硬件。
3. 将 Simulink 连接到在目标硬件上运行的目标应用程序。
4. 开始在目标硬件上执行生成的代码。

使用该工具，您可以分步执行这些步骤，也可以一键执行所有步骤。下表显示该 App 为您的目标硬件提供的工作流支持。



|  |  |  |  |
| --- | --- | --- | --- |
| **工作流功能或任务** | | **目标硬件** | |
| **开发计算机** | **自定义硬件** |
| 一键执行 | | 是 | 是，如果您在 Simulink 与目标硬件之间建立连接。 |
| 分步执行 | **编译以监控** | 是 | 是。软件通过使用您指定的工具链在开发计算机上编译目标应用程序。 |
| **部署** | 是 | 是，如果您在 Simulink 与目标硬件之间建立连接。 |
| **连接** | 是 | 是 |
| **开始** | 是 | 是 |

### 外部模式控制面板

使用“外部模式控制面板”，您可以执行“在自定义硬件上运行”App 提供的部分功能，例如**连接**、**开始**和**断开连接**。您还可以打开“外部信号和触发”对话框，该对话框使您能够：

* 选择要监控的记录信号。
* 配置用于上传目标应用程序数据的触发器。

有关详细信息，请参阅 XCP 面板或 [TCP/IP 和串行面板 （第 32-11](#_bookmark296) 页）。

### 外部模式仿真的安全性

确保对您的开发计算机和目标硬件的访问是安全的。

通过应用下表中列出的安全措施，在您的开发计算机和目标应用程序之间提供安全的通信信道。

|  |  |
| --- | --- |
| **通信协议** | **安全措施** |
| XCP on TCP/IP、TCP/IP | 在受信任的专用网络或虚拟专用网络中运行您的开发计算机和目标应用程序。 |
| XCP on serial、serial | 在开发计算机和目标应用程序之间使用点对点链接，例如串行电缆。 |

在生成用于生产的代码之前，为了降低意外访问目标应用程序的风险，请禁用外部模式设置。

### 另请参阅详细信息

* “External Mode Simulation by Using XCP Communication”
* “Set Up External Mode Connectivity Between Simulink and Target Hardware”
* “Customize XCP Server Software”
* [“使用 TCP/IP 或串行通信的外部模式仿真” （第 32-6 页）](#_bookmark289)
* [“为 TCP/IP 或串行外部模式通信创建传输层” （第 32-39 页）](#_bookmark332)
* “Create Execution-Time Profile for Generated Code” (Embedded Coder)

## 使用 TCP/IP 或串行通信的外部模式仿真

设置并运行外部模式仿真，该仿真使用 TCP/IP 或串行 (RS-232) 通信信道。

1. 创建和配置一个简单模型。
2. 编译目标可执行文件。
3. 运行目标应用程序。
4. 调整参数。

该示例使用 GRT 目标，它不需要外部硬件。生成的可执行文件可以通过以下方式运行：

* 在托管 Simulink 和 Simulink Coder 的开发计算机上运行。
* 作为独立于 MATLAB 和 Simulink 的进程运行。

### 创建和配置模型

在示例的此部分，您创建一个简单模型 **ex\_extModeExample**。您还创建名为 **ext\_mode\_example** 的文件夹来存储模型和生成的可执行文件。

要创建文件夹和模型，请执行以下操作：

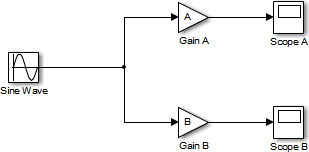
1. 在 MATLAB 命令行中，键入：

**mkdir ext\_mode\_example**

1. 将 **ext\_mode\_example** 作为您的工作文件夹：

**cd ext\_mode\_example**

1. 在 Simulink 中创建一个模型，其中包含一个用作输入信号的 Sine Wave 模块，两个并联的 Gain 模块，两个并联的 Scope 模块。请务必对 Gain 和 Scope 模块进行标记，如下图所示。

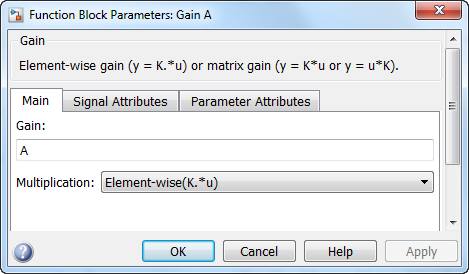


1. 定义并分配两个 MATLAB 工作区变量 **A** 和 **B**：

**A = 2;**

**B = 3;**

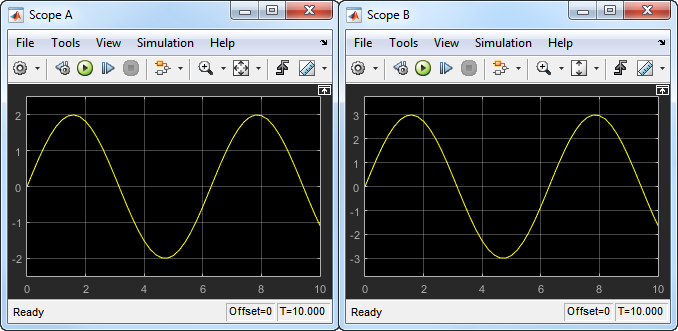
1. 打开 Gain 模块 A 并将其**增益**参数设置为变量 **A**。



1. 打开 Gain 模块 B 并将其**增益**参数设置为变量 **B**。

当目标应用程序在外部模式下编译并连接到 Simulink 时，您可以将新增益值下载到正在执行的目标应用程序。为此，您可以为工作区变量 **A** 和 **B** 赋予新值，或在模块参数对话框中编辑值。有关详细信 息，请参阅 [“调整参数” （第 32-9](#_bookmark293) 页）。

1. 验证模型的运行。打开 Scope 模块并运行模型。当 **A = 2** 且 **B = 3** 时，输出显示如下。

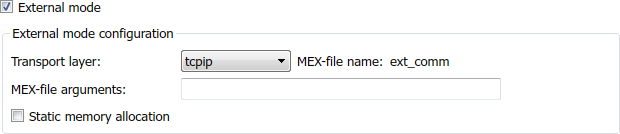


1. 将模型另存为 **ex\_extModeExample**。

### 编译目标可执行文件

设置外部模式目标应用程序所需的模型和代码生成参数。然后，生成代码并编译目标应用程序。

1. 在 Simulink 工具条的 **App** 选项卡上，在**为在硬件上运行进行设置**部分中，点击**在自定义硬件上运行**。
2. 在**硬件**部分，从**系统目标文件描述**列表中，选择“一般实时目标**(grt.tlc)**”。
3. 在**准备**部分中，点击**硬件设置**。将打开“配置参数”对话框，显示由系统目标文件确定的**硬件实现**设置。
4. 在**求解器**窗格上，请执行下列步骤：
   1. 在**类型**字段中，选择“定步长”。
   2. 在**求解器**字段中，选择“离散**(**无连续状态**)**”。
   3. 点击**求解器详细信息**。在**固定步长**字段中，指定 “**0.1**”。（否则，当您生成代码时， SimulinkCoder 编译过程会生成警告并提供一个值。）
   4. 点击**应用**。
5. 在**数据导入/导出**窗格中，清除**时间**和**输出**复选框。在此示例中，数据不会记录到工作区或 MAT 文件中。点击**应用**。
6. 在**代码生成** > **优化**窗格中，检查**默认参数行为**是否设置为“可调”。如果您进行了更改，请点击**应用**。
7. 在**代码生成** > **接口**窗格的**数据交换接口**部分中，选择**外部模式**。
8. 在**外部模式配置**部分，确保为**传输层**参数选择了默认值 “**tcpip**”。



外部模式支持通过 TCP/IP、串行和自定义传输协议进行通信。**MEX 文件名**指定实现主机-目标通信的 MEX 文件的名称。TCP/IP 的默认值为 “**ext\_comm**”，这是 Simulink Coder 软件附带的 MEX 文件。您可以通过提供其他文件来覆盖此默认值。如果您需要支持其他传输层，请参阅“为 [TCP/IP 或串行外部模式通信创建传输层” （第 32-39](#_bookmark332) 页）。

通过 **MEX 文件参数**字段，您可以指定要传递给外部接口程序的参数，如 TCP/IP 服务器端口号。这些参数特定于您正在使用的外部接口。有关设置这些参数的信息，请参阅“T[CP/IP 传输的 MEX 文件可选参数” （第 32-27](#_bookmark321) 页）和“串行传输的 [MEX 文件可选参数” （第 32-28](#_bookmark323) 页）。

此示例使用默认参数。将 **MEX 文件参数**字段留空。

**静态内存分配**复选框控制如何为目标中的外部模式通信缓冲区分配内存。对于此示例，不要选中该复选框。有关详细信息，请参阅 [“控制目标中通信缓冲区的内存分配” （第 32-11](#_bookmark295) 页）。

1. 点击**应用**以保存外部模式设置。
2. 保存模型。
3. 选择**代码生成**窗格。确保**仅生成代码**处于清除状态。
4. 要生成代码并创建目标应用程序，请在模型窗口中，按 **Ctrl+B**。或者，在**硬件**选项卡的**在硬件上运行**

部分中，点击**监控并调节**。然后，在**分步命令**下，点击**编译以监控**。

软件会在您的工作文件夹中创建 **ex\_extModeExample** 可执行文件。

### 运行目标应用程序

您现在运行 **ex\_extModeExample** 目标可执行文件，并使用 Simulink 作为正在运行的目标应用程序的交互式前端。可执行文件位于您的工作文件夹中。运行目标应用程序，并在 Simulink 和目标之间建立通信。

**注意** 像 **ex\_extModeExample** 这样的外部模式程序是基于主机的可执行程序。它的执行不依赖于实时操作系统 (RTOS) 或周期性计时器中断，它也不实时运行。程序只是尽可能快地运行，并且它所计数的时间 单位是仿真的时间单位，与程序外的世界中的时间并不一致。

要运行目标应用程序，请执行以下操作：

1. 在**硬件**选项卡的**在硬件上运行**部分中：
   1. 在**停止时间**字段中，指定 **inf**，这将使模型一直运行到目标应用程序收到来自 Simulink 的停止消息为止
   2. 点击**监控并调节**。然后，在**分步命令**下，点击**部署**。

目标应用程序开始执行，并进入等待状态。

1. 打开模型中的 Scope 模块。在开始执行模型之前，信号在示波器上不可见。
2. 在**硬件**选项卡的**在硬件上运行**部分中，点击**监控并调节**。然后，在**分步命令**下，点击**连接**。此操作启动 Simulink 和目标应用程序之间的握手。当 Simulink 和目标连接时，**连接**按钮变为**断开连接**。

**4**

在**在硬件上运行**部分中，点击 ，这将开始执行生成的模型代码。模型中的两个示波器显示 Gain模块 A 和 B 的输出。

您已在 Simulink 和正在运行的目标应用程序之间建立通信。现在，您可以在 Simulink 中调整模块参数，并观察参数变化的影响。

### 调整参数

下表说明如何在仿真期间调整可调模块参数。

|  |  |
| --- | --- |
| **方法** | **详细信息** |
| 模型数据编辑器 | 要通过模型数据编辑器调整参数，请执行以下操作：   1. 在 Simulink 编辑器的**硬件**选项卡上，在**准备**部分中，点击**调整参数**，这将打开模型数据编辑器。 2. 如果您要同时更新多个可调参数，请在**准备**部分中，通过切换操作打开**保持更新**。 3. 在模型数据编辑器的**参数**选项卡上，在**值**列中，为可调参数指定新值。 4. 通过切换操作关闭**保持更新**或点击**更新所有参数** (**Ctrl+D**)。Simulink 同时将新值下载到目标应用程序。   如果**保持更新**关闭，则在您指定新值后， Simulink 会立即将新值下载到目标应用程序。  有关详细信息，请参阅**模型数据编辑器**。 |
| “模块参数”对话框 | 要通过“模块参数”对话框调整参数，请执行下列步骤：   1. 对于要更新的每个模块，请执行以下操作：    1. 双击该模块，打开“模块参数”对话框。    2. 在参数字段中，指定新参数值。    3. 点击**应用**或**确定**。 2. 如果**保持更新**打开，请通过切换操作关闭**保持更新**或点击**更新所有参数** (**Ctrl+D**)。Simulink同时将新值下载到目标应用程序。   如果**保持更新**关闭，则在您点击**应用**或**确定**  后，Simulink 会立即将新模块值下载到目标应用程序。 |
| MATLAB 工作区 | 如果模块参数是 MATLAB 工作区变量，请执行以下操作：   1. 在命令行窗口中，为变量赋予新值。 2. 在 Simulink 编辑器的**硬件**选项卡上，在**准备**部分中，点击**更新所有参数** (**Ctrl+D**)。Simulink将新值下载到目标应用程序。   有关详细信息，请参阅“创建和编辑变量”。 |

在仿真过程中，您无法更改 Sine Wave 模块的采样时间。模块采样时间是模型结构定义的一部分，也是生成的代码的一部分。因此，如果要更改模块采样时间，您必须停止外部模式仿真，重置模块的采样时间，并重新编译可执行文件。

外部模式仿真期间的模块参数可调性取决于生成的代码表示模块参数的方式。

例如，在 Gain A 模块对话框中，您无法在仿真过程中更改**增益**参数中的表达式 **A**。在这种情况下，您必须在基础工作区中更改变量 **A** 的值。您无法更改表达式，因为生成的代码不会为**增益**参数分配内存存储空间。在这种情况下，代码在结构体中创建字段 **A**：

**/\* Parameters (auto storage) \*/ struct P\_ex\_extModeExample\_T\_ {**

**real\_T A; /\* Variable: A**

**\*/**

**real\_T B; /\* Variable: B**

**\*/**

**real\_T SineWave\_Amp; /\* Expression: 1**

**\*/**

**real\_T SineWave\_Bias; /\* Expression: 0**

**\*/**

**real\_T SineWave\_Freq; /\* Expression: 1**

**\*/**

**real\_T SineWave\_Phase; /\* Expression: 0**

**\*/**

**};**

生成的代码算法使用代码中这一表示模块 Gain A 的字段。在本例中，全局结构体变量

#### ex\_extModeExample\_P 使用 P\_ex\_extModeExample\_T\_ 类型：

**ex\_extModeExample\_B.GainA = ex\_extModeExample\_P.A \* rtb\_SineWave;**

当您在基础工作区中更改 **A** 的值时，仿真会将新值下载到目标应用程序中的字段 **A** 中。

您可以在仿真过程中更改 Sine Wave 模块参数中的表达式，因为生成的代码会在全局结构体 **ex\_extModeExample\_P** 中创建字段来表示模块中的每个参数。当您在模块对话框中更改表达式时，仿真首先计算新表达式。然后，仿真将结果数值下载到目标应用程序中对应的结构体字段。

[请参阅“在生成的代码中创建可调标定参数” （第 16-28](#_bookmark123) 页）。

### 停止目标应用程序

要同时断开 Simulink 与主机/目标通信的连接并结束目标应用程序的执行，请在**硬件**选项卡的**在硬件上运行**部分中，点击**停止**。

### 控制目标中通信缓冲区的内存分配

如果选中**代码生成** > **接口** > **静态内存分配**复选框（针对 GRT 和 ERT 目标），代码生成器将为仅使用静态内存分配的外部模式生成代码（"malloc-free" 代码）。选择**静态内存分配**会启用**静态内存缓冲区大小**参 数。使用此参数指定外部模式静态内存缓冲区的大小。默认值为 1,000,000 字节。如果您为程序输入的值太小，外部模式仿真会在尝试分配超出指定值的内存时发出内存不足错误。在这种情况下，请增大**静态内存缓冲区大小**字段中的值，并重新生成代码。

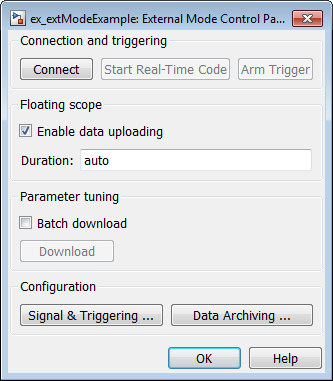
要确定分配多少内存，请对目标启用详细模式（通过在 **make** 命令行中包含 **OPTS="-DVERBOSE"**）。当它执行时，外部模式显示它尝试分配的内存量以及它每次尝试分配时可用的内存量。如果分配失败，您可以使用此控制台日志来确定**静态内存缓冲区大小**字段的规定值。

### TCP/IP 或串行外部模式控制面板

“外部模式控制面板”用于控制 TCP/IP 或串行外部模式操作，包括：

* [“连接、启动和停止” （第 32-12 页）](#_bookmark297)
* [“将目标应用程序信号数据上传到主机” （第 32-12 页）](#_bookmark298)
* [“将参数下载到目标应用程序” （第 32-13 页）](#_bookmark299)
* [“配置目标应用程序信号数据的主机监控” （第 32-14 页）](#_bookmark300)
* [“配置目标应用程序信号数据的主机存档” （第 32-19 页）](#_bookmark309)

要打开“外部模式控制面板”对话框，请在**硬件**选项卡的**准备**部分中，点击**控制面板**。



#### 连接、启动和停止

“外部模式控制面板”执行的连接/断开和启动/停止功能与 Simulink 工具条中**硬件**选项卡上的相同（请参阅“T[CP/IP 或串行外部模式仿真的图形控制项汇总” （第 32-21](#_bookmark310) 页））。

点击**连接**按钮会将您的模型连接到正在等待或正在运行的目标应用程序。完成连接后，该按钮将更改为**断开连接**按钮。**断开连接**将您的模型与目标环境断开连接，但不会停止目标环境中的实时代码运行。

点击**启动实时代码**按钮命令会指示目标开始运行实时代码。当实时代码在目标环境中运行时，该按钮变为

**停止实时代码**按钮。**停止实时代码**用于停止目标应用程序执行，并将您的模型与目标环境断开连接。

#### 将目标应用程序信号数据上传到主机

“外部模式控制面板”允许您触发和取消向主机上传数据。上传数据的目的地可以是 Scope 模块、 Display 模块、To Workspace [模块或“与外部模式兼容的模块和子系统” （第 32-22](#_bookmark311) 页）中列出的其他模块或子系统。

**装备触发器**和**取消触发器**按钮让您能够手动控制向兼容的模块或子系统上传数据，但不包括浮动示波器。

（对于浮动示波器，请使用“外部模式控制面板”的**浮动示波器**部分。）

* 要触发向兼容的模块或子系统上传数据，请点击**装备触发器**按钮。该按钮变为**取消触发器**。
* 要取消数据上传，请点击**取消触发器**按钮。该按钮恢复为**装备触发器**。

[您可以手动或自动触发数据上传。要配置数据上传的信号和触发器，请参阅“配置目标应用程序信号数据的主机监控” （第 32-14](#_bookmark300) 页）。

部分外部模式兼容模块（包括 Scope、Time Scope 和 To Workspace）允许您将上传的数据记录到磁盘。要配置数据存档，请参阅“配置目标应用程序信号数据的主机存档” [（第 32-19](#_bookmark309) 页）。

“外部模式控制面板”的**浮动示波器**部分控制数据何时上传到 Floating Scope 模块以及上传多长时间。当在外部模式下使用时，浮动示波器：

* 不出现在“外部信号和触发”对话框中。
* 不将数据记录到外部模式存档。
* 仅支持手动触发。

**浮动示波器**部分包含以下参数：

* **启用数据上传**选项，用作浮动示波器的**装备触发器**按钮。当目标断开连接时，该选项控制连接浮动示波器时是否装备触发器。当目标连接时，该选项充当装备或取消触发器的切换按钮。
  + 要触发向浮动示波器上传数据，请选中**启用数据上传**。
  + 要取消向浮动示波器上传数据，请清除**启用数据上传**。
* **持续时间**编辑字段，指定在触发事件后外部模式记录浮动示波器数据的基本速率步数。默认情况下，该字段设置为“自动”，这将使用在“外部信号和触发”对话框中设置的持续时间值。默认持续时间值是 1000 个基本速率步。

#### 将参数下载到目标应用程序

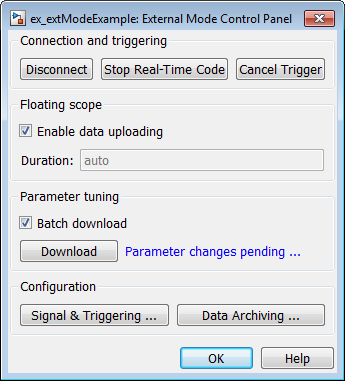
“外部模式控制面板”上的**批量下载**选项用于启用或禁用批处理参数更改。

默认情况下，批量下载被禁用。如果批量下载被禁用，当您点击**确定**或**应用**时，那些通过编辑模块参数对话框直接进行的模块参数更改将被发送到目标。执行**更新图**时，将发送对 MATLAB 工作区变量的更改。

如果选择**批量下载**，将启用**下载**按钮。在您点击**下载**之前，对模块参数所做的更改会存储在本地。当您点击**下载**时，所有更改将在一次传输中发送。

当参数更改等待批量下载时，“外部模式控制面板”在**下载**按钮右侧显示消息 **Parameter changes pending...**。此消息将一直可见，直到 Simulink 引擎收到新参数已安装在目标系统的参数向量中的通知。

下图显示了“外部模式控制面板”，其中**批量下载**选项已激活，参数更改尚未发送。



### 配置目标应用程序信号数据的主机监控

* [“触发器在信号数据上传中的作用” （第 32-14 页）](#_bookmark301)
* [“配置信号数据上传” （第 32-15 页）](#_bookmark302)
* [“默认触发器选项” （第 32-15 页）](#_bookmark303)
* [“选择要上传的信号” （第 32-16 页）](#_bookmark304)
* [“配置触发器选项” （第 32-16 页）](#_bookmark305)
* [“选择触发信号” （第 32-17 页）](#_bookmark306)
* [“设置触发条件” （第 32-18 页）](#_bookmark307)
* [“连接时修改信号和触发选项” （第 32-18 页）](#_bookmark308)

#### 触发器在信号数据上传中的作用

在外部模式下，是否向主机上传目标应用程序信号数据取决于触发器。触发器是开始数据上传所必须满足的一组条件。触发器可以装备也可以不装备。

* 如果装备了触发器，软件会检查允许数据上传开始的触发条件。
* 如果没有装备，软件不会检查触发条件，数据上传无法开始。
* 当主机连接到目标时会自动装备触发器，也可以通过点击“外部模式控制面板”上的**装备触发器**按钮手动装备触发器。

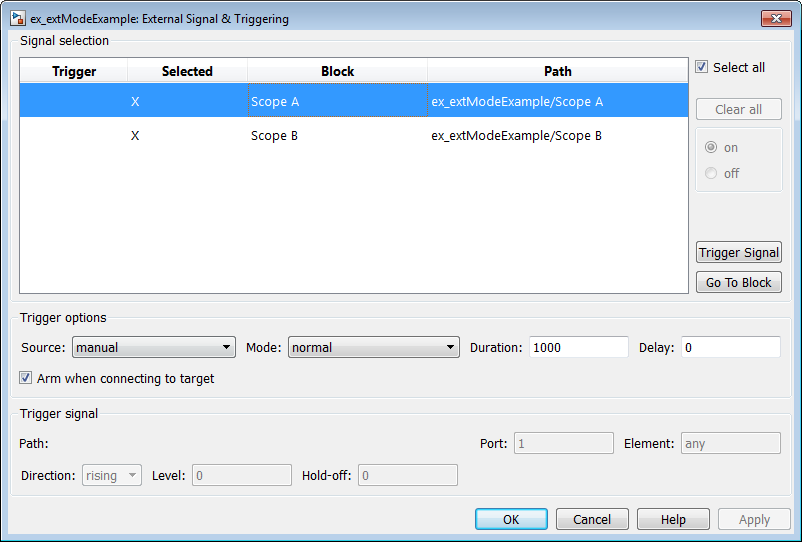
当触发器已装备并且满足触发条件时，触发器触发并且数据上传开始。

经过指定时长的数据采集后，触发事件完成，数据上传停止。然后，触发器可以重新装备，或保持未装备，直到您点击**装备触发器**按钮。

要选择要上传的目标应用程序信号并配置如何触发上传，请参阅“配置信号数据上传” [（第 32-15页）。](#_bookmark302)

#### 配置信号数据上传

点击“外部模式控制面板”的**外部信号和触发**按钮会打开“外部信号和触发”对话框。



“外部信号和触发”对话框显示模型中支持外部模式信号上传的模块和子系统列表。有关哪些类型的模块与外部模式兼容的信息，请参阅“与外部模式兼容的模块和子系统” [（第 32-22](#_bookmark311) 页）。

在“外部信号和触发”对话框中，您可以选择从目标系统采集并在外部模式下查看的信号。您还可以选择一个触发信号（该信号在满足特定信号条件时触发数据上传），并定义触发条件。

#### 默认触发器选项

上图显示了“外部信号和触发”对话框的默认设置。“外部信号和触发”对话框的默认操作简化了对目标应用程序的监控。如果使用默认设置，则不需要预配置信号和触发器。您启动目标应用程序，并将 Simulink 引擎连接到它。系统将选择外部模式兼容模块并装备触发器。在连接到目标应用程序后，信号上传立即开始。

触发器选项的默认配置是：

* “全选”：打开
* “源”：手动
* “模式”：普通
* “持续时间”：1000
* “延迟”：0
* “连接到目标时装备”：打开

#### 选择要上传的信号

模型中的外部模式兼容模块出现在“外部信号和触发”对话框的**信号选择**列表中。您可以使用此列表选择要查看的信号。在**选定**列中，对每个所选模块都会显示 **X**。

**全选**复选框用于选择所有信号。默认情况下，**全选**处于选中状态。

如果清除了**全选**，您可以使用**打开**和**关闭**选项选中或清除单个信号。要选择一个信号，请点击其列表条目，然后选择**打开**选项。要清除一个信号，请点击其列表条目，并选择**关闭**选项。

**全部清除**按钮用于清除所有信号。**配置触发器选项**

[如“触发器在信号数据上传中的作用” （第 32-14](#_bookmark301) 页）中所述，信号数据上传取决于触发器。触发器定义开始上传所必须满足的条件。而且，要开始上传数据，必须装备触发器。当触发器已装备并且满足触发条件时，触发器将触发并开始上传。经过指定时长的数据采集后，触发事件完成，数据上传停止。

要控制何时以及如何从目标系统采集（上传）信号数据，请在“外部信号和触发”对话框中配置以下**触发器选项**。

* **源**：“手动”或“信号”。控制按钮或信号是否触发数据上传。

选择“手动”会指示外部模式使用“外部模式控制面板”上的**装备触发器**按钮作为开始上传数据的触发器。当您点击**装备触发器**时，数据上传开始。

选择“信号”会指示外部模式使用触发信号作为开始上传数据的触发器。当触发信号满足触发条件时

（即，信号在指定方向越过触发电平），触发事件发生。（在**触发信号**部分指定触发条件。）如果触发器已装备，外部模式会监控触发事件的发生。当触发事件发生时，数据上传开始。

* **模式**：“普通”或“一次”。控制触发事件完成后触发器是否重新装备。

在“普通”模式下，外部模式会在每次触发事件后自动重新装备触发器。在触发器触发时，下一次数据上传开始。

在“一次”模式下，每次装备触发器时，外部模式仅采集一个缓冲区的数据。

有关**模式**设置的详细信息，请参阅“配置目标应用程序信号数据的主机存档” [（第 32-19](#_bookmark309) 页）。

* **持续时间**：指定触发事件后外部模式上传数据的基本速率步数（默认值为 1000）。例如，如果**持续时间**设置为 1000，并且模型的基本（最快）速率为一秒：
  + 对于以一秒的基本速率采样的信号（1.0 赫兹），外部模式在触发事件期间采集 1000 个连续样本。
  + 对于以两秒采样的信号（0.5 赫兹），外部模式在触发事件期间采集 500 个样本。
* **延迟**：指定要应用于数据采集的延迟。延迟表示触发事件和数据采集开始之间经过的时间。延迟以基本速率步数表示。它可以是正值或负值（默认值为 0）。负延迟对应于预触发。当延迟为负值时，将采集并上传触发事件之前的数据。
* **连接到目标时装备**：选中或清除。无论由按钮还是信号触发数据上传（由**源**定义），触发器都必须装备好才能允许开始数据上传。

如果选择此选项，将在连接到目标时装备触发器。

* + 如果触发**源**是“手动”，则立即开始数据上传。
  + 如果触发**源**是“信号”，则立即开始监控触发信号。当触发信号满足触发条件（如**触发信号**部分所定义）时，数据上传开始。

如果您清除了**连接到目标时装备**，可以通过点击“外部模式控制面板”上的**装备触发器**按钮手动装备触发器。

在外部模式下仿真时，模型中的每个速率都会在目标上创建一个缓冲区。缓冲器中的每个条目都足够大，可以将该速率下每个信号所需的所有数据保留一个时间步（时间+数据+识别信号的外部模式索引）。环形缓冲区中的条目数由外部模式触发器的**持续时间**参数 (**ExtModeTrigDuration**) 决定。分配给目标以用于缓冲信号的内存与**持续时间**和上传的信号数量成比例。**持续时间**还指示在外部模式下触发事件后日志数据的基本速率步数。

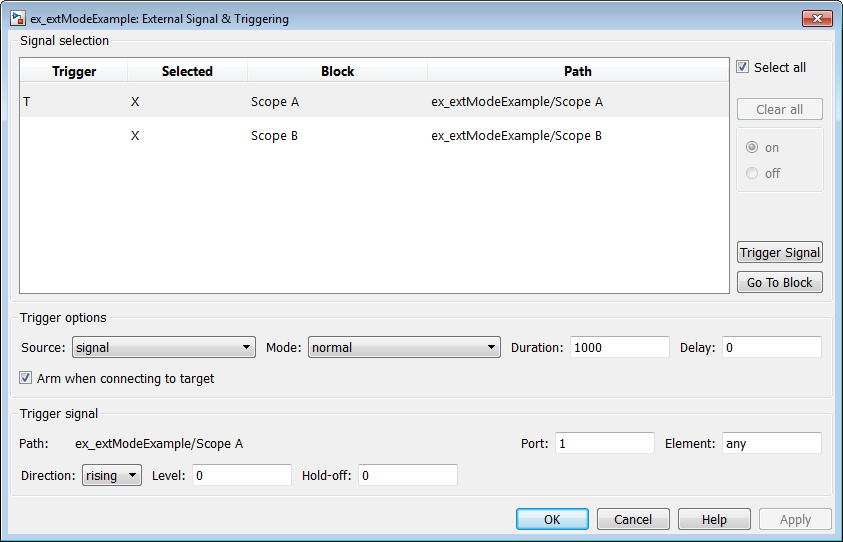
**持续时间**值指定在每个数据缓冲区中要采集的连续数据点数。您输入的**持续时间**值应等于您需要采集的连续采样点数，而不是依赖一系列缓冲区来保持连续。如果输入的值小于采样点总数，在将值从数据缓冲区传输到 MATLAB 工作区期间，您可能会丢失采样点。Simulink 软件仅在一个缓冲区内保持点连续性。在缓冲区之间，因为传输时间的原因，一些样本可能会被忽略。

**持续时间**值会影响 Scope 和 To Workspace 模块的**将数据点限制为最后**值。模块保存到 MATLAB 工作区的采样点数是这两个值中的较小者。要设置模块保存的采样点数，请清除**将数据点限制为最后**。然后，使用**持续时间**指定保存的采样点数。

#### 选择触发信号

您可以指定一个信号作为触发信号。要选择触发信号，请从**触发器选项**部分的**源**菜单中，选择“信号”。此操作将启用**触发信号**部分中的参数。然后，在**信号选择**列表中选择信号，并点击**触发信号**按钮。

当您选择一个信号作为触发信号时，**信号选择**列表的**触发**列中会相应出现一个 **T**。在下图中，**Scope A** 信号是触发信号。**Scope B** 也被选中以便查看，如**选定**列中的 **X** 所示。



选择触发信号后，您可以使用**触发信号**部分定义触发条件，并设置触发信号**端口**和**元素**参数。

#### 设置触发条件

使用“外部信号和触发”对话框的**触发信号**部分设置触发条件和属性。仅当**触发器选项**部分中的触发器参数**源**设置为“信号”时，才能启用**触发信号**参数。

默认情况下，触发器模块的第一个输入端口的任何元素都会导致触发器触发（即“端口”指定为 1，“元素”指定为“任何”）。您可以通过调整**触发信号**部分中的**端口**和**元素**值来修改此行为。**端口**字段接受数字或关键字 **last**。**元素**字段接受数字或关键字 **any** 或 **last**。

在**触发信号**部分，您还可以定义触发事件发生的条件。

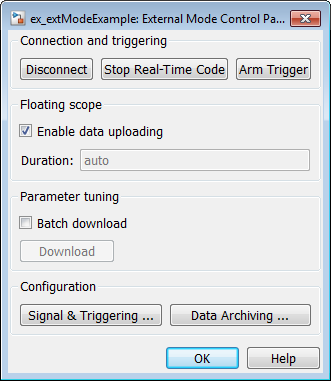
* **方向**：“上升沿”、“下降沿”或“任一沿”。当信号穿越阈值时应遵循的行进方向。默认值为“上升沿”。
* **电平**：一个值，指示信号必须沿指定方向穿越该阈值才能触发触发器。默认情况下，水平为 0。
* **释抑**：仅适用于“普通”模式。**释抑**以基本速率步数表示，是一个触发事件从终止到重新装备之间的时间。

#### 连接时修改信号和触发选项

在配置信号数据上传并将 Simulink 连接到正在运行的目标可执行文件后，您可以修改信号和触发选项，而无需断开与目标的连接。

如果触发器已装备（例如，触发器选项**连接到目标时装备**已选中，这也是默认设置），则无法修改“外部信号和触发”对话框。要修改信号和触发选项，请执行以下操作：

1. 打开“外部模式控制面板”。
2. 点击**取消触发器**。将停止触发且停止显示上传数据。
3. 打开“外部信号和触发”对话框，根据需要修改信号和触发器选项。例如，您可以在**信号选择**部分中启用或禁用范围，在**触发器选项**部分中更改触发器**模式**，例如从“普通”更改为“一次”。
4. 点击**装备触发器**。触发和上传数据的显示恢复，并应用了您的修改。



### 配置目标应用程序信号数据的主机存档

在外部模式下，您可以使用 Simulink 的 Scope 和 To Workspace 模块将数据存档到磁盘。

要了解存档功能如何工作，请设想存档未启用时的数据处理。有两种情况，一次模式和普通模式。

* 在一次模式下，触发事件发生后，每个所选模块都会将其数据写入工作区，就像在仿真结束时一样。如果触发了另一个一次模式，现有工作区数据将被覆盖。
* 在普通模式下，外部模式会在每次触发事件后自动重新装备触发器。因此，您可以将普通模式视为一系列一次模式。该系列中的每个一次（除了最后一个）都称为中间结果。由于触发器可以随时触发，因此将中间结果写入工作区会导致工作区变量发生不可预测的覆盖。因此，默认行为是只将最后一次 one- shot 的结果写入工作区。中间结果将被丢弃。如果您知道各次触发之间都有足够的时间来检查中间结果，您可以通过选择**将中间结果写入工作区**选项来覆盖默认行为。此选项无法防止工作区数据不被后续触发覆盖。

如果您使用 Simulink 的 Scope 模块将数据存档到磁盘，请打开“示波器”参数对话框，并选择选项**将数据记录到工作区**。该选项是必需的，原因如下：

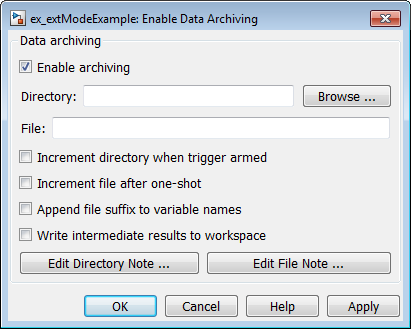
* 数据首先从示波器数据缓冲区传输到 MATLAB 工作区，再写入 MAT 文件。
* 在“示波器”参数对话框中输入的**变量名称**与 MATLAB 工作区和 MAT 文件中的相同。通过保存数据，可以将使用**变量名称**参数命名的变量保存到 MAT 文件中。

**注意** 如果没有选择 Scope 模块选项**将数据记录到工作区**，用于数据记录的 MAT 文件仍将创建，但是为 空。

“启用数据存档”对话框支持：

* 文件夹注释
* 文件注释
* 自动化数据存档

在“外部模式控制面板”上，点击**数据存档**按钮，打开“启用数据存档”对话框。如果您的模型连接到了目标环境，请在配置数据存档时断开连接。要启用对话框中的其他控制项，请选择**启用存档**。



“启用数据存档”对话框支持下列操作。

#### 文件夹注释

要为某一文件夹中的一批相关数据文件添加注释，请在“启用数据存档”对话框中，点击**编辑目录注释**。 MATLAB 编辑器随即打开。在此窗口中键入注释，这些注释将以文件形式保存到指定文件夹中。默认情况下，注释保存到数据存档最近一次写入的文件夹中。

#### 文件注释

要为单个数据文件添加注释，请在“启用数据存档”对话框中，点击**编辑文件注释**。文件查找器窗口随即打开，默认情况下，它将定位到您最近一次写入的文件。选择 MAT 文件会打开编辑窗口。在此窗口中，添加或编辑您要随单个 MAT 文件一起保存的注释。

#### 自动化数据存档

要配置将日志记录结果自动写入磁盘（可以选择是否包括中间结果），请使用**启用存档**选项及其启用的控制项。该对话框提供以下相关控制项：

* **目录**：指定保存数据的文件夹。如果选择**装备触发器时递增目录**，外部模式会追加后缀。
* **文件**：指定保存数据的文件的名称。如果选择**触发一次后递增文件**，外部模式会追加后缀。
* **装备触发器时递增目录**：每次点击**装备触发器**按钮，外部模式都会使用一个新文件夹来写入日志文件。文件夹以增量方式命名，例如，**dirname1**、**dirname2** 等。
* **触发一次后递增文件**：新数据缓冲区保存在增量命名的文件中：**filename1**、**filename2** 等。文件增量命名在普通模式下自动发生。
* **将文件后缀追加到变量名称**：外部模式以增量方式命名文件时，每个文件均包含具有相同名称的变量。如果选择**将文件后缀追加到变量名称**，每个文件将包含唯一的变量名称。例如，外部模式在以增量方式命名的文件中（**file\_1**、**file\_2** 等）保存名为 **xdata** 的变量时，将其命名为 **xdata\_1**、**xdata\_2** 等。这种方法支持将 MAT 文件加载到工作区并通过 MATLAB 命令提示符比较变量。如果没有唯一名称，在 MATLAB 工作区中，**xdata** 的每个实例都将覆盖它的前一个实例。
* **将中间结果写入工作区**：如果您希望 SimulinkCoder 软件将中间结果写入工作区，请选择此选项。

### TCP/IP 或串行外部模式仿真的图形控制项汇总

您可以通过以下各项控制 TCP/IP 或串行外部模式仿真：

* Simulink 工具条中的**硬件**选项卡。要显示**硬件**选项卡，请在 Simulink 工具条的 **App** 选项卡中，点击

#### 在自定义硬件上运行。

* “外部模式控制面板”。要打开此对话框，请在 Simulink 编辑器中**硬件**选项卡的**准备**部分中，点击 。然后在**信号监控和跟踪**下，点击**控制面板**。

下表列出了您可以使用**硬件**选项卡和“外部模式控制面板”中可用的控制项执行的外部模式操作。



|  |  |  |
| --- | --- | --- |
| **外部模式操作** | **“硬件”选项卡** | **外部模式控制面板** |
| 编译目标应用程序。 | **在硬件上运行** > **编译以监控** | 不适用 |
| 在目标硬件上运行应用程序。 | **在硬件上运行** > **部署** | 不适用 |
| 将 Simulink 连接到正在等待或运行的目标应用程序。 | **在硬件上运行** > **连接**  当 Simulink 连接到目标应用程序时，**连接**呈灰显状态，而**断开连接**  处于活动状态。 | **连接**  当 Simulink 连接到目标应用程序时，**连接**变为**断开连接**。 |
| 开始在目标环境中实时执行生成的代码。 | **在硬件上运行** > **开始** | **启动实时代码**  当生成的代码开始执行时，按钮变为**停止实时代码**。 |

|  |  |  |
| --- | --- | --- |
| **外部模式操作** | **“硬件”选项卡** | **外部模式控制面板** |
| 断开 Simulink 与目标环境的连接，但不停止代码的实时执行。 | **在硬件上运行** > **断开连接**  当 Simulink 与目标应用程序断开连接时，**断开连接**呈灰显状态，而  **连接**处于活动状态。 | **断开连接** |
| 停止目标应用程序的执行，并断开 Simulink 与目标环境的连接。 | 在**在硬件上运行**部分中，使用**停止**  按钮 。 | **停止实时代码** |
| 调整批量模块参数。 | 在**准备**部分，使用**保持更新**和**更新所有参数**  按钮。 | **批量下载**和**下载** |

### 与外部模式兼容的模块和子系统



* [“兼容模块” （第 32-22 页）](#_bookmark312)
* [“信号查看子系统” （第 32-22 页）](#_bookmark313)
* [“支持数据存档的模块” （第 32-24 页）](#_bookmark314)

#### 兼容模块

在外部模式下，您可以使用以下类型的模块来接收和查看从目标应用程序上传的信号：

* Floating Scope 和 Scope 模块
* DSP System Toolbox 产品中的 Spectrum Analyzer 和 Time Scope 模块
* Display 模块
* To Workspace 模块
* 用户编写的 S-Function 模块

S-function API 中内置了外部模式方法。此方法支持用户编写兼容外部模式的模块。请参阅

#### matlabroot/simulink/include/simstruc.h。

* Record, XY Graph 模块

您可以将某些子系统指定为信号查看子系统，并使用它们接收和查看从目标应用程序上传的信号。有关详细信息，请参阅 [“信号查看子系统” （第 32-22](#_bookmark313) 页）。

您可以使用“外部信号和触发”对话框选择外部模式兼容模块和子系统并装备触发器。默认情况下，模型中的此类模块处于选中状态，并且手动触发器设置为在连接到目标应用程序时装备。

#### 信号查看子系统

信号查看子系统是原子子系统，它封装了来自目标系统的信号的处理和查看。信号查看子系统仅在主机上运行，不在目标系统中生成代码。信号查看子系统以普通、快速、快速加速和外部仿真模式运行。

**注意** 如果信号查看子系统放置在 SIL 或 PIL 组件中，例如 SIL 或 PIL 模式的顶层模型、SIL 或 PIL 模式的 Model 模块或者 SIL 或 PIL 模块中，则它处于非激活状态。但是，SIL 或 PIL 组件可以为在支持模式下运行的信号查看子系统馈送信号。

如果您要在查看或记录信号之前处理或调节信号，但又不想在目标系统上执行这些任务，则信号查看子系统非常有用。通过使用信号查看子系统，您可以在目标系统上生成更小、更高效的代码。

与其他外部模式兼容模块一样，信号查看子系统显示在“外部信号和触发”对话框中。要将子系统声明为信号查看子系统，请执行以下操作：

1. 在“模块参数”对话框中，选择**视为原子单元**选项。

有关原子子系统的详细信息，请参阅“子系统”。

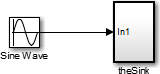
1. 要打开 SimViewingDevice 属性，请使用 **set\_param** 命令：

**set\_param('blockname', 'SimViewingDevice','on')**

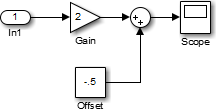
**'blockname'** 是子系统的名称。

1. 确保子系统满足以下要求：
   * 它必须为纯 Sink 模块。也就是说，不能包含 Outport 模块或 Data Store 模块。仅当对应的 From 模块包含在子系统边界内时，它才能包含 Goto 模块。
   * 它不能有连续状态。

以下模型 **sink\_examp** 包含原子子系统 **theSink**。



子系统 **theSink** 对其输入信号应用增益和偏移，并将其显示在 Scope 模块上。



如果 **theSink** 声明为信号查看子系统，则生成的目标应用程序仅包括 Sine Wave 模块的代码。如果在 “外部信号和触发”对话框中选择并装备了 **theSink**，目标应用程序将在仿真过程中向 **theSink** 上传正弦波信号。然后，您可以修改 **theSink** 中模块的参数，并观察上传的信号。

如果 **theSink** 未声明为信号查看子系统，其 Gain、Constant 和 Sum 模块将作为目标系统上的子系统代码运行。正弦波信号在经过这些模块处理后，将上传到 Simulink 引擎，并可在 **sink\_examp/theSink/ Scope2** 上查看。额外的信号处理和从主机下载模块参数变化，将增加对目标系统的处理要求。

#### 支持数据存档的模块

在外部模式下，您可以使用以下类型的模块将数据存档到磁盘：

* Scope 模块
* To Workspace 模块

[您可以使用“启用数据存档”对话框配置数据存档，如“配置目标应用程序信号数据的主机存档” （第](#_bookmark309) [32-19](#_bookmark309) 页）中所述。

### 用于下载可调参数的外部模式机制

* [“下载机制” （第 32-24 页）](#_bookmark316)
* [“内联和可调参数” （第 32-25 页）](#_bookmark317)

根据生成目标应用程序时**默认参数行为**选项的设置，处理参数更新的方式有所不同。“下载机制” [（第](#_bookmark316) [32-24](#_bookmark316) 页）说明了**默认参数行为** 设置为“可调”的外部模式通信的操作。“内联和可调参数” [（第 32-](#_bookmark317)

[25](#_bookmark317) 页）说明了**默认参数行为**设置为“内联”的外部模式的操作。

#### 下载机制

在外部模式下，Simulink 引擎不对模块图所示的系统进行仿真。默认情况下，启用外部模式时，Simulink引擎会将参数下载到目标系统。在初始下载后，引擎将保持等待模式，直到您更改了模块图中的参数，或直到引擎从目标接收到了数据。

当您更改模块图中的参数时，Simulink 引擎会调用外部接口 MEX 文件，将新参数值（及其他信息）作为参数传递。外部接口 MEX 文件包含用于实现进程间通信 (IPC) 信道一端的代码。该信道将 Simulink 进程

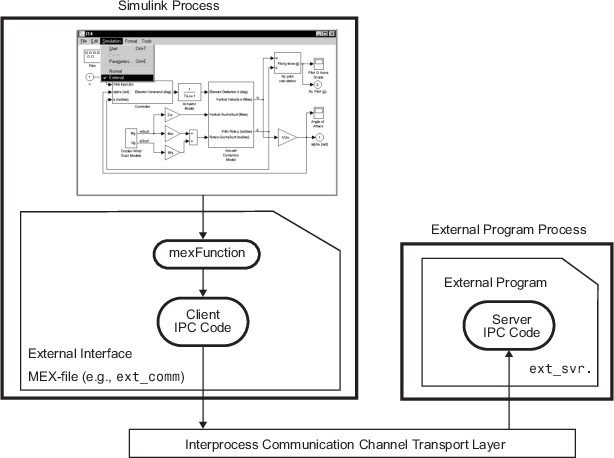
（MEX 文件在其中执行）连接到正在执行外部程序的进程。MEX 文件通过使用该信道向外部程序传输新参数值。

通信信道的另一端在外部程序中实现。此端将新参数值写入目标的参数结构体 (**model\_P**) 中。 Simulink 端通过向外部程序发送包含参数信息的消息来启动参数下载操作。在客户端/服务器计算的术语

中，Simulink 端是客户端，外部程序是服务器。这两个进程可以是远程进程，也可以是本地进程。当客户

端和服务器处于远程时，使用诸如 TCP/IP 之类的协议来传输数据。当客户端和服务器位于本地时，可以使用串行连接或共享内存来传输数据。

下图显示了这种关系。每当您更改模块图中的参数时，Simulink 引擎都会调用外部接口 MEX 文件。然后，MEX 文件通过使用通信信道将参数下载到外部程序。



#### 外部模式架构

**内联和可调参数**

默认情况下，外部模式程序中的参数（“T[CP/IP 和串行外部模式的限制” （第 32-36](#_bookmark331) 页）中列出的除外）是可调的；也就是说，您可以使用本节中所述的下载机制来更改它们。

如果您将**默认参数行为**设置为“内联”（通过“配置参数”对话框的**优化**窗格），则 Simulink Coder 代码生成器会在生成的代码中嵌入模型参数的数值（常量），而不是符号参数名称。内联参数有助于生成体积更小、运行更高效的代码。然而，内联参数实际上为常量，因此是不可调的。

Simulink Coder 软件可让您通过内联大多数参数来提高整体效率，同时保留对您的应用程序非常重要的所选参数的运行时调整灵活性。在内联参数时，您可以使用 **Simulink.Parameter** 对象从内联中删除单个参数，并声明它们是可调的。此外，您可以使用这些对象来控制参数在生成的代码中的表示方式。

有关可调参数的详细信息，请参阅“在生成的代码中创建可调标定参数” [（第 16-28](#_bookmark123) 页）。

**主机/目标连接上的自动参数上传**

每次 Simulink 引擎连接到在**默认参数行为**设置为“内联”状态下生成的目标应用程序时，目标应用程序都会将其可调参数的当前值上传到主机。这些值赋给对应的 MATLAB 工作区变量。此过程在主机和目标之间同步参数值。如果可调参数存储在模型工作区或数据字典中，则不会上传这些值，并且 Simulink 会产生警告。

模型所需的工作区变量必须在主机/目标连接时初始化。否则上传无法继续，并会导致错误。一旦建立了连接，这些变量将更新以反映目标系统上的当前参数值。

仅当在**默认参数行为**设置为“内联[”的情况下生成目标应用程序时，参数才会自动上传。“下载机制”](#_bookmark316)

[（第 32-24](#_bookmark316) 页）说明了**默认参数行为**设置为“可调”时外部模式通信的操作。

### 选择客户端和服务器的通信协议

* [“简介” （第 32-26 页）](#_bookmark319)
* [“使用 TCP/IP 实现” （第 32-26 页）](#_bookmark320)
* [“使用串行实现” （第 32-28 页）](#_bookmark322)
* [“运行外部程序” （第 32-29 页）](#_bookmark324)
* [“实现外部模式协议层” （第 32-30 页）](#_bookmark326)

#### 简介

SimulinkCoder 产品提供可实现基于 TCP/IP 或串行协议的外部模式通信的客户端和服务器端的代码。如果您的目标系统支持 TCP/IP，您可以将 SimulinkCoder 产品中基于套接字的外部模式实现与生成的代码结合使用。否则，请使用提供的串行传输层选项，或对其进行自定义。

低级传输层处理消息的物理传输。Simulink 引擎和模型代码都独立于此传输层。传输层和直接与传输层对接的代码被分别隔离在不同的模块中，这些模块负责格式化、发送和接收消息及数据包。

#### 使用 TCP/IP 实现

在 The Open Group UNIX 或 PC 系统上，您可以将基于 TCP/IP 的外部模式客户端/服务器实现与实时程序结合使用。有关自定义外部模式传输层的帮助，请参阅“为 [TCP/IP 或串行外部模式通信创建传输层”](#_bookmark332)

[（第 32-39](#_bookmark332) 页）。

要通过 TCP/IP 使用 Simulink 外部模式，请执行以下操作：

* 确保您已指定目标的 TCP/IP 传输的外部接口 MEX 文件。

MathWorks 提供的目标在 **matlabroot/toolbox/simulink/simulink/extmode\_transports.m** 中指定外部接口 MEX 文件的名称。接口的名称在“配置参数”对话框的**接口** 窗格的**外部模式配置**部分中显示为不可编辑的文本。TCP/IP 默认值为 “**ext\_comm**”。

要为自定义目标指定 TCP/IP 传输，您必须将以下形式的条目添加到 MATLAB 路径上的

#### sl\_customization.m 文件中：

**function sl\_customization(cm)**

**cm.ExtModeTransports.add('stf.tlc', 'transport', 'mexfile', 'Level1');**

**%end function**

* + **stf.tlc** 是您为其注册传输的系统目标文件的名称（例如，**'mytarget.tlc'**）
  + **transport** 是要在“配置参数”对话框的**接口**窗格的**传输层**菜单中显示的传输协议的名称（例如，

#### 'tcpip'）

* + **mexfile** 是传输协议的关联外部接口 MEX 文件的名称（例如，**'ext\_comm'**）

您可以通过增加 **cm.ExtModeTransports.add** 行来指定多个目标和/或传输，例如：

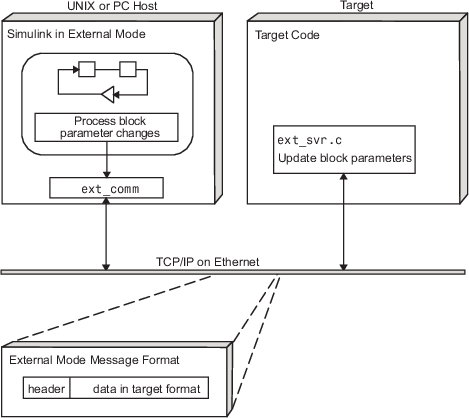
**function sl\_customization(cm) cm.ExtModeTransports.add('mytarget.tlc', 'tcpip', 'ext\_comm', 'Level1');**

**cm.ExtModeTransports.add('mytarget.tlc', 'serial', ... 'ext\_serial\_win32\_comm', 'Level1');**

**%end function**

* 请确保适当配置模板联编文件以链接 TCP/IP 服务器代码的源文件，并确保该文件在编译生成的代码时定义编译器标志。
* 编译外部程序。
* 运行外部程序。
* 将 Simulink 模型设置为外部模式，并连接到目标。

下图显示基于 TCP/IP 的实现的结构。



#### 基于 TCP/IP 的外部模式客户端/服务器实现

**TCP/IP 传输的 MEX 文件可选参数**

在“外部目标接口”对话框中，您可以指定传递给外部模式接口 MEX 文件的可选参数，以便与正在执行的目标通信。

* 目标网络名称：运行外部程序的计算机的网络名称。默认情况下，这是运行 Simulink 产品的计算机，例如 **'myComputer'**。您也可以使用 IP 地址，例如 **'148.27.151.12'**。
* 详细级别：控制数据传输过程中显示的信息的详细级别。值为 **0** 或 **1**，含义如下：

1. - 无信息
2. - 详细信息

* TCP/IP 服务器端口号：默认值为 **17725**。您可以将端口号更改为 **256** 和 **65535** 之间的值，以避免端口冲突。

参数为位置参数，必须按以下顺序指定：

**<TargetNetworkName> <VerbosityLevel> <ServerPortNumber>**

例如，如果您要指定详细级别（第二个参数），则您还必须指定目标网络名称（第一个参数）。参数可以用空白或逗号分隔。例如：

**'148.27.151.12' 1 30000**

您可以在启动外部程序时为其指定命令行选项。请参阅“运行外部程序” [（第 32-29](#_bookmark324) 页）。

#### 使用串行实现

控制串行通道上的主机/目标通信类似于控制 TCP/IP 通道上的主机/目标通信。要在串行通道上使用 Simulink 外部模式，您必须：

* 确保您已指定目标的串行传输的外部接口 MEX 文件。

MathWorks 提供的目标在 **matlabroot/toolbox/simulink/simulink/extmode\_transports.m** 中指定外部接口 MEX 文件的名称。接口的名称在“配置参数”对话框的**接口** 窗格的**外部模式配置**部分中显示为不可编辑的文本。串行默认值为 **serial**。

要为自定义目标指定串行传输，您必须将以下形式的条目添加到 MATLAB 路径上的

#### sl\_customization.m 文件中：

**function sl\_customization(cm)**

**cm.ExtModeTransports.add('stf.tlc', 'transport', 'mexfile', 'Level1');**

**%end function**

* + **stf.tlc** 是您为其注册传输的系统目标文件的名称（例如，**'mytarget.tlc'**）
  + **transport** 是要在“配置参数”对话框的**接口**窗格的**传输层**菜单中显示的传输协议的名称（例如，

#### 'serial'）

* + **mexfile** 是传输协议的关联外部接口 MEX 文件的名称（例如，**'ext\_serial\_win32\_comm'**）

您可以通过增加 **cm.ExtModeTransports.add** 行来指定多个目标和/或传输，例如：

**function sl\_customization(cm) cm.ExtModeTransports.add('mytarget.tlc', 'tcpip', 'ext\_comm', 'Level1'); cm.ExtModeTransports.add('mytarget.tlc', 'serial', ...**

**'ext\_serial\_win32\_comm', 'Level1');**

**%end function**

* 请确保适当配置模板联编文件以链接串行服务器代码的源文件，并确保该文件在编译生成的代码时定义编译器标志。
* 编译外部程序。
* 运行外部程序。
* 将 Simulink 模型设置为外部模式，并连接到目标。

**串行传输的 MEX 文件可选参数**

在“配置参数”对话框的**接口**窗格的 **MEX 文件参数**字段中，您可以指定传递给外部模式接口 MEX 文件的参数，以便与执行目标通信。对于串行传输，**ext\_serial\_win32\_comm** 的可选参数如下：

* 详细级别：此参数控制数据传输期间显示的信息的详细级别。此参数的值是：
  + **0**（无信息），或
  + **1**（详细信息）
* 串行端口 ID：主机的端口 ID，指定为整数或字符向量。例如，将 USB 转串行转换器的端口 ID 指定为 **'/dev/ttyusb0'**。Simulink Coder 会对整数端口 ID 加上前缀 **\\.\COM**（在 Windows 上）或 **/dev/ ttyS**（在 Unix 上）。

使用串行连接启动目标应用程序时，必须指定用于将其连接到主机的端口 ID。为此，请包含 **-port** 命令行选项。例如：

**mytarget.exe -port 2 -w**

* 波特率：指定整数值。默认值为 **57600**。

MEX 文件选项参数为位置参数，必须按以下顺序指定：

**<VerbosityLevel> <SerialPortID> <BaudRate>**

例如，如果您要指定串行端口 ID（第二个参数），则您还必须指定详细级别（第一个参数）。参数可以用空白或逗号分隔。例如：

**1 '/dev/ttyusb0' 57600**

启动外部程序时，可以指定命令行选项。

#### 运行外部程序

要在外部模式下使用 Simulink 产品，外部程序必须处于运行状态。

如果目标应用程序与主机在同一台计算机上执行，并且通信是通过环回串行电缆进行的，则目标的端口 ID不能与主机的端口 ID 相同（如 **MEX 文件参数**编辑字段所指定）。

要运行外部程序，请键入以下形式的命令：

**model -opt1 ... -optN**

**model** 是外部程序的名称，**-opt1 ... -optN** 是选项。（请参阅 [“外部程序的命令行选项” （第 32-30页）。）在本节的示例中，外部程序的名称是](#_bookmark325) **ext\_example**。

**在 Windows 环境中运行外部程序**

在 Windows 环境中，您可以通过以下任一方式运行外部程序：

* 打开命令提示符窗口。在命令提示符下，键入目标可执行文件的名称，后跟可能的选项，例如：

**ext\_example -tf inf -w**

* 您也可以从 MATLAB 命令行窗口启动目标可执行文件。命令必须以感叹号 (**!**) 开头，且以“与”符号 (**&**) 结尾，如下例所示：

**!ext\_example -tf inf -w &**

“与”符号 (**&**) 使操作系统生成另一个进程来运行目标可执行文件。如果不包含“与”符号，程序仍会运行，但您无法在 MATLAB 命令提示符下输入命令或手动终止可执行文件。

**在 UNIX 环境中运行外部程序**

在 UNIX 环境中，您可以通过以下任一方式运行外部程序：

* 打开一个 **Xterm** 窗口。在命令提示符下，键入目标可执行文件的名称，后跟可能的选项，例如：

**./ext\_example -tf inf -w**

* 您也可以从 MATLAB 命令行窗口启动目标可执行文件。您必须在后台运行它，以便仍可访问 Simulink环境。命令必须以感叹号 (**!**) 和点斜杠（./ 表示当前目录）开头，且以“与”符号 (**&**) 结尾，如下例所示：

**!./ext\_example -tf inf -w &**

“与”符号 (**&**) 使操作系统生成另一个进程来运行目标可执行文件。

**外部程序的命令行选项**

由 Simulink Coder 代码生成器生成的外部模式目标可执行文件支持以下命令行选项：

#### -tf n

**-tf** 选项会覆盖在 Simulink 模型中设置的停止时间。参数 **n** 指定程序将运行的秒数。值 **inf** 指示模型无限期运行。在本例中，模型代码将一直运行，直到目标应用程序收到来自 Simulink 引擎的停止消息。

以下示例将停止时间设置为 10 秒。

**ext\_example -tf 10**

当在外部模式下编译和执行纯整数 ERT 目标时，目标会将停止时间参数 (**-tf**) 解释为基本速率步数，而不是执行秒数。

#### -w

指示目标应用程序进入等待状态，直到收到来自主机的消息。此时，目标正在运行，但没有执行模型代码。当您从**仿真**菜单中选择**启动实时代码**或点击“外部模式控制面板”中的**启动实时代码**按钮时，将发送开始消息。

如果要查看从时间步 0 的目标应用程序执行数据，或要在目标应用程序开始执行模型代码之前修改参数，请使用 **-w** 选项。

#### -port n

指定目标应用程序的 TCP/IP 端口号或串行端口 ID **n**。目标应用程序的端口号必须与用于 TCP/IP 传输的主机的端口号相匹配。端口号取决于传输类型。

* + 对于 TCP/IP 传输：端口号是 **256** 和 **65535** 之间的整数，默认值为 **17725**。目标硬件上运行的另一个 TCP/IP 服务不能使用所选端口。
  + 对于串行传输：端口 ID 是整数或字符向量。例如，将 USB 转串行转换器的端口 ID 指定为 **'/dev/ ttyusb0'**

#### -baud r

指定为整数时，此选项仅适用于串行传输。

#### 实现外部模式协议层

如果您要实现自己的外部模式通信传输层，您必须修改 SimulinkCoder 产品提供的某些代码模块，并创建新外部接口 MEX 文件。请参阅“为 [TCP/IP 或串行外部模式通信创建传输层” （第 32-39](#_bookmark332) 页）。

### 以编程方式使用外部模式

您可以从 MATLAB 命令行或在脚本中以编程方式运行外部模式仿真。使用 **get\_param** 和 **set\_param** 命令检索和设置模型仿真命令行参数（如 **SimulationMode** 和 **SimulationCommand**）的值以及外部模式命令行参数（如 **ExtModeCommand** 和 **ExtModeTrigType**）的值。

以下模型仿真命令假设 Simulink 模型已打开，该模型将通过外部模式连接的目标应用程序也已加载。

1. 将 Simulink 模型更改为外部模式：

**set\_param(gcs,'SimulationMode','external')**

1. 将打开的模型连接到加载的目标应用程序：

**set\_param(gcs,'SimulationCommand','connect')**

1. 开始运行目标应用程序：

**set\_param(gcs,'SimulationCommand','start')**

1. 停止运行目标应用程序：

**set\_param(gcs,'SimulationCommand','stop')**

1. 断开目标应用程序与模型的连接：

**set\_param(gcs,'SimulationCommand','disconnect')**

使用 **'SimulationCommand'** 参数的 **set\_param** 命令是异步的。如果从脚本连续运行这些命令，每个命令都会启动，而不会等待前一个命令运行完毕才启动下一个命令。要检查每个命令是否完成，请在脚本中使用带 **'SimulationStatus'** 参数的 **get\_param** 命令。例如，对于步骤 1 至 3，在脚本中指定以下命令：

**set\_param(gcs,'SimulationMode','external'); set\_param(gcs,'SimulationCommand','connect');**

**isExternalSimulationActive = false; while ~isExternalSimulationActive**

**simStatus = get\_param(gcs, 'SimulationStatus'); isExternalSimulationActive = strcmp(simStatus, 'external');**

**end set\_param(gcs,'SimulationCommand','start');**

有关详细信息，请参阅“以编程方式运行仿真”。

诊断查看器显示由 **get\_param** 和 **set\_param** 命令产生的错误消息。

要调整工作区参数，请在命令提示符下更改其值。如果工作区参数是 **Simulink.Parameter** 对象，请将新值赋给 **Value** 属性。

**myVariable = 5.23;**

**myParamObj.Value = 5.23;**

在外部模式下，要下载工作区参数，需要更新模型图。以下模型仿真命令将启动模型更新：

**set\_param(gcs,'SimulationCommand','update')**

要触发和取消向示波器上传数据，请使用 **ExtModeCommand** 值 **armFloating** 和 **cancelFloating**，或 **armWired** 和 **cancelWired**。例如，要触发然后取消向有线（非浮动）示波器上传数据，请执行以下命令：

**set\_param(gcs,'ExtModeCommand','armWired') set\_param(gcs,'ExtModeCommand','cancelWired')**

下表列出了您可以在 **get\_param** 和 **set\_param** 命令中使用的外部模式命令行参数。该表包含简短描 述、有效值（粗体突出显示了默认值）以及“外部模式”对话框中与之对应的等效项。有关等效于“配置参数”对话框中**接口**窗格选项的外部模式参数，请参阅“模型配置参数：代码生成接口”。

**外部模式命令行参数**

|  |  |  |
| --- | --- | --- |
| **参数和值** | **对话框等效项** | **描述** |
| **ExtModeAddSuffixToVar off**, **on** | 启用数据存档：**将文件后缀追加到变量名称**复选框 | 为每个增量命名的文件增量命名变量名称。 |
| **ExtModeArchiveDirName character vector** | 启用数据存档：**目录**文本字段 | 将数据保存在指定的文件夹中。 |
| **ExtModeArchiveFileName character vector** | 启用数据存档：**文件**文本字段 | 将数据保存在指定文件中。 |
| **ExtModeArchiveMode**  **character vector** - **off**, **auto**, **manual** | 启用数据存档：**启用存档**复选框 | 激活自动数据存档功能。  要指定 **manual**，请运行 **set\_param(gcs,**  **'ExtModeArchiveMode', 'manual')**。 |
|  |  | 请注意，如果您指定 **auto**， **ExtModeAutoIncOneShot** 将设置为 **on**。 |
| **ExtModeArmWhenConnect off**, **on** | 外部信号和触发：**连接到目标时装备**复选框 | 一旦 Simulink Coder 软件连接到目标，立即装备触发器。 |
| **ExtModeAutoIncOneShot off**, **on** | 启用数据存档：**触发一次后递增文件**复选框 | 在增量命名的文件中保存新数据缓冲区。 |
| **ExtModeAutoUpdateStatusClock**（仅限 Windows 平台）  **off**、**on** | 不适用 | 连续上传并在模型窗口状态栏上显示目标时间。 |
| **ExtModeBatchMode off**, **on** | 外部模式控制面板：**批量下载**复选框 | 在批处理模式下启用或禁用参数下载。 |
| **ExtModeChangesPending off**, **on** | 不适用 | 当启用 **ExtModeBatchMode** 时，指示参数是否保留在要下载到目标的参数队列中。 |
| **ExtModeCommand**  **character vector** - **armFloating**, **armWired**, **cancelFloating**, **cancelWired** | * **armFloating** 和 **cancelFloating** 等效于选中和清除 “外部模式控制面板”中的**浮动示波器** > **启用数据上传**   复选框 | 向目标应用程序发出外部模式命令。 |
|  | * **armWired** 和 **cancelWired**等效于“外部模式控制面板”中的**装备触发器**和**取消触发器**按钮 |  |
| **ExtModeConnected off**, **on** | 外部模式控制面板：**连接/断开连接**按钮 | 指示与目标应用程序的连接状态。 |
| **ExtModeEnableFloating off**, **on** | 外部模式控制面板：**启用数据上传**复选框 | 当与浮动示波器建立连接时，启用或禁用触发器的装备和取消。 |
| **ExtModeIncDirWhenArm off**, **on** | 启用数据存档：**装备触发器时递增目录**复选框 | 每次装备触发器时，将日志文件写入增量命名的文件夹中。 |

|  |  |  |
| --- | --- | --- |
| **参数和值** | **对话框等效项** | **描述** |
| **ExtModeLogAll off**, **on** | 外部信号和触发：**全选**复选框 | 将可用信号从目标上传到主机。 |
| **ExtModeParamChangesPending off**, **on** | 不适用 | 当 Simulink Coder 软件连接到目标并且 **ExtModeBatchMode** 启用时，指示参数是否仍保留在要下载到目标的参数队列中。比 **ExtModeChangesPending** 更高效，因为它检查与目标的连接。 |
| **ExtModeSkipDownloadWhenConnect off**, **on** | 不适用 | 无需下载参数即可连接到目标应用程序。 |
| **ExtModeTrigDelay integer** (**0**) | 外部信号和触发：**延迟**文本字段 | 指定发生触发事件和开始数据采集之间经过的时间量（以基本速率步数表示）。 |
| **ExtModeTrigDirection character vector** - **rising**, **falling**, **either** | 外部信号和触发：**方向**菜单 | 指定当信号穿越阈值时规定的行进方向。 |
| **ExtModeTrigDuration integer** (**1000**) | 外部信号和触发：**持续时间**文本字段 | 指定触发事件后外部模式记录数据的基本速率步数。 |
| **ExtModeTrigDurationFloating character vector** - **integer** (**auto**) | 外部模式控制面板：**持续时间**文本字段 | 指定浮动示波器的持续时间。如果指定“自动”，则使用 **ExtModeTrigDuration** 的值。 |
| **ExtModeTrigElement**  **character vector** - **integer**, **any**, **last** | 外部信号和触发：**元素**文本字段 | 指定属于指定触发器模块输入端口的、可导致触发器触发的元素。 |
| **ExtModeTrigHoldOff integer** (**0**) | 外部信号和触发：**释抑**文本字段 | 指定触发事件终止和触发器重新装备之间的基本速率步数。 |
| **ExtModeTrigLevel integer** (**0**) | 外部信号和触发：**电平**文本字段 | 指定触发信号要触发触发器所必须穿越的阈值。 |
| **ExtModeTrigMode**  **character vector** - **normal**, **oneshot** | 外部信号和触发：**模式**菜单 | 指定触发器是在每次触发事件后自动重新装备，还是每次装备时只采集一个缓冲区的数据。 |
| **ExtModeTrigPort**  **character vector** - **integer** (**1**), **last** | 外部信号和触发：**端口**文本字段 | 指定属于指定触发器模块的、其元素可导致触发器触发的输入端口。 |
| **ExtModeTrigType**  **character vector** - **manual**, **signal** | 外部信号和触发：**源**菜单 | 指定是在触发器装备时还是在指定的触发器信号满足触发器条件时开始记录数据。 |
| **ExtModeUploadStatus**  **character vector** - **inactive**, **armed**, **uploading** | 不适用 | 返回外部模式上传机制的状态 - 非活动、已装备或上传。 |
| **ExtModeWriteAllDataToWs off**, **on** | 启用数据存档：**将中间结果写入工作区**复选框 | 将中间结果写入工作区。 |

### 在外部模式下以动画方式显示 Stateflow 图

如果您有 Stateflow，您可以在外部模式下以动画方式显示图。在外部模式下，您可以以动画方式显示 Stateflow 图中的状态，并在浮动示波器或信号查看器中查看测试点信号。

* [“在外部模式下以动画方式显示仿真期间的状态” （第 32-35 页）](#_bookmark329)
* [“在浮动示波器和信号查看器中查看测试点数据” （第 32-35 页）](#_bookmark330)

#### 在外部模式下以动画方式显示仿真期间的状态

要在外部模式下以动画方式显示图中的状态，请执行以下操作：

1. 将您要以动画方式显示的图加载到目标计算机上。
2. 在“配置参数”对话框中，选中**外部模式**复选框。
3. 打开“外部模式控制面板”对话框。
4. 点击**外部信号和触发**。
5. 在“外部信号和触发”对话框中，设置以下参数。

|  |  |
| --- | --- |
| **位置：** | **选择：** |
| **信号选择**窗格 | 要以动画方式显示的图 |
| **触发器选项**窗格 | **连接到目标时装备**复选框 |
| **触发器选项**窗格 | **模式**字段的下拉菜单中的“普通” |

1. 编译模型以生成可执行文件。
2. 部署目标应用程序。
3. 将 Simulink 连接到目标应用程序。
4. 开始执行生成的模型代码。图会在执行时突出显示状态。

#### 在浮动示波器和信号查看器中查看测试点数据

在外部模式下对 Stateflow 图进行仿真时，可以将本地示波器的图数据指定为测试点，并在浮动示波器和信号查看器中查看测试点数据。

要在外部模式下的仿真过程中查看测试点数据，请执行以下操作：

1. 打开模型资源管理器，对于要查看的每个数据，请按照以下步骤操作：
   1. 在中间的**内容**窗格中，选择感兴趣的状态或局部数据。
   2. 在右侧的**对话框**窗格中，选择**记录**选项卡，然后选中**测试点**复选框。
2. 从浮动示波器或信号查看器中，点击信号选择按钮：



将打开“信号选择器”对话框。

1. 在“信号选择器”的**模型层次结构**窗格中，选择图。
2. 在“模型层次结构”的**列出内容**菜单中，选择**仅测试点/记录信号**，然后选择您要查看的信号。
3. [按照“在外部模式下以动画方式显示仿真期间的状态” （第 32-35](#_bookmark329) 页）中所述，在外部模式下对模型进行仿真。

当仿真运行时，示波器或查看器将显示测试点信号的值。

有关详细信息，请参阅 “示波器和查看器在快速加速模式下的行为”。

### TCP/IP 和串行外部模式的限制

|  |  |
| --- | --- |
| **功能** | **详细信息** |
| 服务代码接口 | 当生成的代码在目标环境中执行时，对于从配置了基于 ERT 的系统目标文件和服务代码接口的组件模型生成的代码，不支持使用外部模式来调整参数和监控信号。 |
| 更改参数 | 通常，如果更改某参数会导致模型结构发生变化，则不能更改该参数。例如，您无法更改以下参数：   * 模块的状态、输入或输出的数量 * 采样时间或采样时间的数量 * 连续系统的积分算法 * 模型或模块的名称   如果您对模块图的上述参数进行了更改，则您必须用新生成的代码重新编译程序。  您可以通过特定方式更改传递函数和状态空间表示模块中的参数：   * Transfer Fcn（连续和离散）和 Discrete Filter 模块的参数（分子和分母多项式）可以更改（只要状态数不变）。 * 一旦外部仿真开始，将无法更改 State-Space 和 Zero Pole（包括连续和离散）模块中由用户指定或计算的参数（即在零极点到状态空间变换过程中得到的 A、B、C 和 D 矩阵）中的零值元。 * 在 State-Space 模块中，如果在可控标准实现中指定矩阵，则允许在保留此实现和矩阵维数的前提下对 A、B、C、D 矩阵进行更改。   如果 Simulink 模块图与外部程序不匹配，Simulink 会产生错误，指出校验和不匹配。校验和会考虑顶层模型，但不考虑引用模型。使用更新后的模块图重新编译目标应用程序。 |
| 上传数据 | 对于具有定点值的参数，不支持从目标应用程序上传参数值。 |
| 上传可变大小的信号 | 下列目标不支持上传可变大小的信号：   * Simulink Real-Time * Texas Instruments™ C2000™ |
| 仿真中的信号值显示 | 模型中信号值的图形显示（如“在模型图中显示信号值”所述）不受支持。例如，您无法使用**仿真中的数据显示**菜单选项**悬停时显示值标签**、**点击时切换值标签**和**显示所选端口的值标签**。 |
| 可调结构体参数 | 不支持上传或下载可调结构体参数。 |
| C++ 代码 | 不支持生成的 C++ 代码。如果指定以下配置参数设置，仿真会生成错误：   * **语言** - “**C++**” * **代码接口打包** - “**C++** 类” |

|  |  |
| --- | --- |
| **功能** | **详细信息** |
| 纯整数代码 | 支持纯整数代码。  如果您没有在执行命令中指定 **-tf finalTime**，目标应用程序将无限期运行生成的模型代码，而忽略 **StopTime**。  如果在执行命令中指定 **-tf finalTime**：   * **finalTime** 值表示基本速率步数，而不是秒数。 * **finalTime** 的最大值（以计时单元为单位）是 **MAX\_int32\_T**。 * 当 16 位或 32 位计时单元计数器溢出时，Scope 模块中的仿真时间返回到零。 |
| 半精度 | 不支持半精度数据类型。 |
| **Simulink.ImageType** | 不支持 **Simulink.ImageType** 数据类型。 |
| 存档数据 | Scope 和 To Workspace 模块支持将数据存档到磁盘。但是，其他示波器不支持数据存档。例如，您无法使用 Floating Scope 模块或查看器和生成器管理工具查看器对象来存档数据。 |
| 引用模型中的示波器 | 在模型层次结构中，如果顶层模型在外部模式下仿真，而引用模型在普通模式或加速模式下仿真，则不会显示引用模型中的示波器。  但是，如果顶层模型更改为在普通模式下仿真，则引用模型中示波器的行为在普通模式和加速模式下会有所不同。在普通模式下仿真的引用模型中的示波器会显示，而在加速模式下仿真的引用模型中的示波器不会显示。 |
| Scope 和 Display 模块连接到 Simulink 消息模块 | 在外部模式仿真期间，连接到 Simulink 消息模块的 Scope 和 Display 模块不提供显示画面。 |
| 仿真的开始时间 | 不支持非零仿真开始时间。在“配置参数”对话框的**求解器**窗格中，将**开始时间**设置为默认值 “**0.0**”。 |
| 中间步长值 | 一些 Simulink 模块可以在一个仿真时间步内生成多个值。例如：   * 在一个时间步内多次调用的 Simulink Function 或 Function-Call Subsystem 模块。 * 在一个时间步内执行多次的迭代子系统模块。   对于外部模式仿真中的每个时间步，Simulink 从目标应用程序仅上传这些模块的最终值。Simulink 不上传在时间步中生成的中间值。 |
| 作用域为文件的数据 | 不支持作用域为文件的数据，例如，应用内置自定义存储类 **FileScope** 的数据项。作用域为文件的数据无法从外部访问。 |
| 具有自定义存储类的信号 | 不支持上传具有自定义存储类 (CSC) 的信号。 |
| 使用 **printf** 语句 | 要在目标硬件显示器上显示目标应用程序错误和信息性消息，您可以使用 **printf** 调用。对于某些目标硬件，使用 **printf** 语句会增加外部模式二进制文件的大小。要禁用 **printf** 调用，请为目标应用程序编译器指定预处理器宏定义 **EXTMODE\_DISABLEPRINTF**。 |

|  |  |
| --- | --- |
| **功能** | **详细信息** |
| 命令行参数 | 您可以使用命令行参数来运行目标应用程序。但存在以下限制：   * 命令行参数的解析需要 **sscanf** 函数，这会增加一些目标硬件的程序大小。 * 一些目标应用程序不接受命令行参数。   如果目标硬件不支持命令行参数的解析，请为目标应用程序编译器指定预处理器宏定义 **EXTMODE\_DISABLE\_ARGS\_PROCESSING=1**。  要替换 **-w** 选项，您可以使用以下命令指定目标应用程序进入并保持等待状态，直到它收到来自 Simulink 的连接消息：  **set\_param(modelName, 'OnTargetWaitForStart', 'on');**  编译过程向编译器提供所需的选项 (**- DON\_TARGET\_WAIT\_FOR\_START=1**)。 |
| 行优先代码生成 | 不支持以行优先格式生成的代码。 |

**另请参阅**

**详细信息**

* “External Mode Simulation by Using XCP Communication”
* [“为 TCP/IP 或串行外部模式通信创建传输层” （第 32-39 页）](#_bookmark332)
* “Customize Connectivity for TCP/IP or Serial External Mode Simulations”

## 为 TCP/IP 或串行外部模式通信创建传输层

[“外部模式的设计” （第 32-39 页）](#_bookmark333)

[“外部模式通信概述” （第 32-41 页）](#_bookmark334) [“外部模式源文件” （第 32-42 页）](#_bookmark335) [“实现自定义传输层” （第 32-44 页）](#_bookmark339)

**本节内容**

本节帮助您使用自己的底层通信层通过外部模式连接到自定义目标。本节主题包括：

* 外部模式的设计和操作概述
* 外部模式源文件说明
* 修改外部模式源文件和编译可执行文件以处理默认 **ext\_comm** MEX 文件任务的规范

本节假设您熟悉 Simulink Coder 程序的执行过程和外部模式的基本操作。

### 外部模式的设计

Simulink 引擎与目标系统之间的外部模式通信基于一种客户端/服务器架构。客户端（Simulink 引擎）发送消息，请求服务器（目标）接受参数更改或上传信号数据。服务器通过执行请求来作出响应。

低级传输层处理消息的物理传输。Simulink 引擎和模型代码都独立于此传输层。传输层和直接与传输层对接的代码被分别隔离在不同的模块中，这些模块负责格式化、发送和接收消息及数据包。

此设计使不同的目标可以使用不同的传输层。GRT、ERT 和 RSim 目标通过使用 TCP/IP 和 RS-232（串行）通信来支持主机/目标通信。Simulink Desktop Real-Time 目标支持共享内存通信。Wind River Systems Tornado➅ 目标仅支持 TCP/IP。

Simulink Coder 产品为客户端和服务器端外部模式模块提供了完整的源代码，如 GRT、ERT、快速仿真和 Tornado 目标以及 Simulink Desktop Real-Time 和 Simulink Real-Time 产品使用的源代码。客户端主模块为 **ext\_comm.c**。服务器端主模块为 **ext\_svr.c**。

这两个模块通过以下源文件调用指定的传输层。

#### 内置传输层实现

|  |  |  |
| --- | --- | --- |
| **协议** | **客户端还是服务器？** | **源文件** |
| TCP/IP | 客户端（主机） | * **matlabroot/toolbox/coder/simulinkcoder\_core/ext\_mode/host/common/ rtiostream\_interface.c** * **matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/ rtiostream\_tcpip.c** |
| 服务器（目标） | * **matlabroot/rtw/c/src/ext\_mode/common/rtiostream\_interface.c** * **matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/ rtiostream\_tcpip.c** |
| 串行 | 客户端（主机） | * **matlabroot/toolbox/coder/simulinkcoder\_core/ext\_mode/host/serial/ ext\_serial\_transport.c** * **matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/ rtiostream\_serial.c** |
| 服务器（目标） | * **matlabroot/rtw/c/src/ext\_mode/serial/ext\_svr\_serial\_transport.c** * **matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/ rtiostream\_serial.c** |

对于串行通信，模块 **ext\_serial\_transport.c** 和 **rtiostream\_serial.c** 实现客户端传输功能，模块 **ext\_svr\_serial\_transport.c** 和 **rtiostream\_serial.c** 实现对应的服务器端功能。对于 TCP/IP 通信，模块 **rtiostream\_interface.c** 和 **rtiostream\_tcpip.c** 同时实现客户端和服务器端功能。您可以编辑这些文件的副本（但不能修改原件）。您可以使用以下模板创建类似的文件，从而使用自己的底层通信层来支持外部模式：

#### 客户端（主机）：matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/ rtiostream\_tcpip.c (TCP/IP) 或 matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/ rtiostream\_serial.c（串行）

* 服务器端（目标）：**matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/ rtiostream\_tcpip.c** (TCP/IP) 或 **matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/ rtiostream\_serial.c**（串行）

**rtiostream\_interface.c** 文件是外部模式协议和 **rtiostream** 通信信道之间的接口。有关实现和测试

**rtiostream** 通信信道的详细信息，请参阅：

* “Communications rtiostream API” (Embedded Coder)

#### rtiostreamtest

可以使用文档中介绍的接口实现您的 **rtiostream** 通信信道，以免更改文件 **rtiostream\_interface.c** 或与外部模式有关的其他文件。

**注意** 不能修改工作源文件。请使用 **/custom** 或 **/rtiostream** 文件夹中提供的模板作为起点并遵照其中的说明进行操作。

您只需要提供实现底层通信的代码，不需要担心主机与目标之间的数据转换或消息的格式等问题。这些工作由 Simulink Coder 软件完成。

在客户端（Simulink 引擎），通信由 **ext\_comm**（对于 TCP/IP 通信）和 **ext\_serial\_win32\_comm**

（对于串行通信）MEX 文件来处理。

在服务器端（目标），外部模式模块链接到目标可执行文件。如果您选择 **External mode**，这将在代码生成时自动发生，并且是基于您指定的 **Transport layer** 选项。从主程序和模型执行引擎中调用的模块独立于生成的模型代码。

实现您自己的客户端底层传输协议的一般步骤如下：

1. 编辑模板 **rtiostream\_tcpip.c**，将底层通信调用替换为您自己的通信调用。
2. 为您的自定义传输生成 MEX 文件可执行文件。
3. 在 Simulink 软件中注册您的新传输层，以便能够使用 Configuration Parameters 对话框的

**Interface** 窗格为模型选择该传输。

有关详细信息，请参阅“创建自定义客户端（主机）传输协议” [（第 32-45](#_bookmark341) 页）。实现您自己的服务器端底层传输协议的一般步骤如下：

1. 编辑模板 **rtiostream\_tcpip.c**，将底层通信调用替换为您自己的通信调用。通常这涉及到为您的目标硬件编写或集成设备驱动程序。
2. 修改模板联编文件以支持新传输。

有关详细信息，请参阅“创建自定义服务器（目标）传输协议” [（第 32-47](#_bookmark344) 页）。

### 外部模式通信概述

本节从更高的层次概述 Simulink Coder 生成的程序如何与 Simulink 外部模式进行通信。这些说明基于 Simulink Coder 产品附带的 TCP/IP 版外部模式。

要进行通信，服务器（目标）程序和 Simulink 软件都必须处于正在执行的状态。这并不是说服务器系统中的模型代码必须处于正在执行的状态。服务器可以处于等待 Simulink 引擎发出命令以开始执行模型的状 态。

客户端和服务器使用携带数据包的双向套接字进行通信。数据包由消息（命令、参数下载和响应）或数据

（信号上传）构成。

如果使用 **-w** 命令行选项调用目标应用程序，程序将进入等待状态，直到它从主机收到消息为止。否则，程序将开始执行模型。当目标应用程序处于等待状态时，Simulink 引擎可以将参数下载到目标并配置数据上传。

当用户从 **Simulation** 菜单中选择 **Connect to Target** 选项时，主机将通过发送 **EXT\_CONNECT message** 来启动握手。服务器使用与自身相关的信息来作出响应。这些信息包括：

* 校验和。主机使用模型校验和来确定目标代码是否准确表达了当前的 Simulink 模型。
* 数据格式信息。当要下载格式设置数据或解释已上传的数据时，主机会使用此信息。

至此，主机和服务器就建立了连接。服务器要么正在执行模型，要么处于等待状态。（在后一种情况下，用户可从 **Simulation** 菜单中选择 **Start Real-Time Code** 开始执行模型。）

在执行模型的过程中，消息服务器作为后台任务运行。此任务负责接收和处理消息，如参数下载。

数据上传包括信号数据包的前台执行和后台服务。当目标计算模型输出时，它也会将信号值复制到数据上传缓冲区中。这是与每个任务标识符 (**tid**) 关联的任务的一部分。因此，数据收集发生在前台，而收集到的数据的传输发生在后台。后台任务使用数据包将数据收集缓冲区中的数据发送给 Simulink 引擎。

主机以消息的形式启动大多数交换。目标通常会发送响应，确认它已收到并处理了消息。消息和命令的示例包括：

* 连接消息/连接响应
* 开始目标仿真/开始响应
* 参数下载/参数下载响应
* 装备数据上传触发器/装备触发器响应
* 终止目标仿真/目标关闭响应

当模型到达最终时间、主机发送终止命令或 Stop Simulation 模块终止模型执行时，模型执行将终止。终止后，服务器将通知主机模型执行已停止并关闭其套接字。主机也关闭其套接字并退出外部模式。

### 外部模式源文件

* [“客户端（主机）MEX 文件接口源文件” （第 32-42 页）](#_bookmark336)
* [“服务器（目标）源文件” （第 32-43 页）](#_bookmark337)
* [“服务器文件夹中的其他文件” （第 32-44 页）](#_bookmark338)

#### 客户端（主机）MEX 文件接口源文件

除非另有说明，MEX 文件接口组件的源文件保存在 **matlabroot/toolbox/coder/simulinkcoder\_core/ ext\_mode/host** 文件夹（打开）中：

#### common/ext\_comm.c

此文件是外部模式通信的核心。它相当于目标和 Simulink 引擎之间的一个中继站。**ext\_comm.c** 使用一个共享的数据结构体 **ExternalSim** 与 Simulink 引擎通信，并通过调用传输层与目标通信。

**ext\_comm.c** 执行的任务包括建立与目标的连接、下载参数以及终止与目标的连接。

#### common/rtiostream\_interface.c

此文件是外部模式协议和 **rtiostream** 通信信道之间的接口。有关实现 **rtiostream** 通信信道的详细信息，请参阅“Communications rtiostream API” (Embedded Coder)。可以使用文档中介绍的接口实现您的 **rtiostream** 通信信道，以免更改文件 **rtiostream\_interface.c** 或与外部模式有关的其他文件。

#### matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/rtiostream\_tcpip.c

此文件实现必需的 TCP/IP 传输层函数。Simulink Coder 软件附带的 **rtiostream\_tcpip.c** 版本使用的 TCP/IP 函数包括 **recv()**、**send()** 和 **socket()**。

#### matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/rtiostream\_serial.c

此文件实现必需的串行传输层函数。Simulink Coder 软件附带的 **rtiostream\_serial.c** 版本使用的串行函数包括 **ReadFile()**、**WriteFile()** 和 **CreateFile()**。

#### serial/ext\_serial\_transport.c

此文件实现必需的串行传输层函数。**ext\_serial\_transport.c** 包括 **matlabroot/rtw/c/src/ ext\_mode/serial** 文件夹（打开）中的 **ext\_serial\_utils.c**，还包含一些客户端和服务器端通用的函数。

#### common/ext\_main.c

此文件是外部模式的 MEX 文件包装器。**ext\_main.c** 使用标准 **mexFunction** 调用与 Simulink 引擎对接。（有关详细信息，请参阅 **mexFunction** 参考页和“将 MATLAB 与外部编程语言和系统集

成”。）**ext\_main.c** 包含函数调度程序 **esGetAction**，它将来自 Simulink 引擎的请求发送给

#### ext\_comm.c。

* **common/ext\_convert.c** 和 **ext\_convert.h**

此文件包含用于将数据从主机格式转换为目标格式（或反之）的函数。它们实现的功能包括字节交换

（从 big-endian 到 little-endian）、从非 IEEE➅ 浮点转换为 IEEE 双精度以及其他转换。这些函数可由 **ext\_comm.c** 和 Simulink 引擎调用，后者通过函数指针直接调用这些函数。

**注意** 您不需要通过自定义 **ext\_convert** 来实现自定义传输层，但可能需要为所需的目标自定义 **ext\_convert**。例如，如果目标以 Texas Instruments 格式表示 **float** 数据类型，则必须修改 **ext\_convert** 以执行从 Texas Instruments 到 IEEE 的转换。

#### common/extsim.h

此文件定义 **ExternalSim** 数据结构体和访问宏。该结构体用于在 Simulink 引擎和 **ext\_comm.c** 之间通信。

#### common/extutil.h

此文件只包含编译 **assert** 宏的条件句。

#### common/ext\_transport.h

此文件定义传输层必须实现的函数。

#### 服务器（目标）源文件

这些文件链接到 **model.exe** 可执行文件。除非另有说明，它们保存在 **matlabroot/rtw/c/src/ ext\_mode** 文件夹（打开）中。

#### common/ext\_svr.c

**ext\_svr.c** 类似于主机上的 **ext\_comm.c**，但它通常负责执行更多任务。它相当于主机和生成的代码之间的一个中继站。就像 **ext\_comm.c** 一样，**ext\_svr.c** 也执行诸如建立和终止主机连接之类的任务。 **ext\_svr.c** 还包含后台任务函数，用于将下载的参数写入目标模型，或从目标数据缓冲区中提取数据并发送回主机。

#### common/rtiostream\_interface.c

此文件是外部模式协议和 **rtiostream** 通信信道之间的接口。有关实现 **rtiostream** 通信信道的详细信息，请参阅“Communications rtiostream API” (Embedded Coder)。可以使用文档中介绍的接口实现您的 **rtiostream** 通信信道，以免更改文件 **rtiostream\_interface.c** 或与外部模式有关的其他文件。

#### matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/rtiostream\_tcpip.c

此文件实现必需的 TCP/IP 传输层函数。Simulink Coder 软件附带的 **rtiostream\_tcpip.c** 版本使用的 TCP/IP 函数包括 **recv()**、**send()** 和 **socket()**。

#### matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/rtiostream\_serial.c

此文件实现必需的串行传输层函数。本软件附带的 rtiostream\_serial.c 版本使用的串行函数包括

#### ReadFile()、WriteFile() 和 CreateFile()。

* **matlabroot/toolbox/coder/rtiostream/src/rtiostream.h**

此文件定义 **rtiostream\_tcpip.c** 中实现的 **rtIOStream\*** 函数。

#### serial/ext\_svr\_serial\_transport.c

此文件实现必需的串行传输层函数。**ext\_svr\_serial\_transport.c** 包括 **serial/ext\_serial\_utils.c**，它包含客户端和服务器端通用的函数。

#### common/updown.c

**updown.c** 处理与目标模型交互的细节。在下载参数的过程中，**updown.c** 负责将新参数安置到模型的参数向量中。对于数据上传，**updown.c** 包含的函数可从模型的 **blockio** 向量中提取数据，并将数据写入上传缓冲区。**updown.c** 同时为 **ext\_svr.c** 和模型代码（例如 **grt\_main.c**）提供服务。它包含使用 **ext\_svr.c** 的后台任务调用的代码，还包含作为高优先级模型执行的一部分调用的代码。

* **matlabroot/rtw/c/src/dt\_info.h**（包含在生成的模型编译文件 **model.h** 中）

这些文件包含用于在不同的计算机架构之间访问多数据类型结构体的数据类型转换信息。此信息用于在主机格式和目标格式之间进行数据转换。

#### common/updown\_util.h

此文件只包含编译 **assert** 宏的条件句。

#### common/ext\_svr\_transport.h

此文件定义必须由服务器（目标）传输层实现的 **Ext\*** 函数。

#### 服务器文件夹中的其他文件

* **common/ext\_share.h**

包含主机和目标模块需要的消息代码定义和其他定义。

#### serial/ext\_serial\_utils.c

包含串行协议传输层的主机和目标模块需要的 MEX 链接、生成的代码以及用于通信的函数和数据结构体。

* 串行传输实现包括以下附加文件：
  + **serial/ext\_serial\_pkt.c** 和 **ext\_serial\_pkt.h**
  + **serial/ext\_serial\_port.h**

### 实现自定义传输层

* [“自定义传输层的要求” （第 32-44 页）](#_bookmark340)
* [“创建自定义客户端（主机）传输协议” （第 32-45 页）](#_bookmark341)
* [“用于重新编译 ext\_comm 和 ext\_serial\_win32 MEX 文件的 MATLAB 命令” （第 32-45 页）](#_bookmark342)
* [“注册自定义客户端（主机）传输协议” （第 32-46 页）](#_bookmark343)
* [“创建自定义服务器（目标）传输协议” （第 32-47 页）](#_bookmark344)
* [“小于 64 字节的串行接收缓冲区” （第 32-48 页）](#_bookmark345)

#### 自定义传输层的要求

* 虽然有用于预分配静态内存的选项，但默认情况下，**ext\_svr.c** 和 **updown.c** 使用 **malloc** 在目标内存中分配缓冲区用于消息、数据收集和其他用途。如果您的目标使用其他内存分配方案，则必须修改这些模块。
* 假设目标同时支持 **int32\_T** 和 **uint32\_T** 数据类型。

#### 创建自定义客户端（主机）传输协议

要实现底层传输协议的客户端（主机），请执行以下操作：

1. 编辑模板文件 **matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/ rtiostream\_tcpip.c**，将底层通信调用替换为您自己的通信调用。
   1. 复制该文件并将其重命名为 **rtiostream\_name.c**（将 **name** 替换为对您有意义的名称）。
   2. 用相同的名称将函数 **rtIOStreamOpen**、**rtIOStreamClose**、**rtIOStreamSend** 和 **rtIOStreamRecv** 替换为调用您的底层通信基元的函数。您可以通过 **rtiostream\_interface.c**从其他外部模式模块中调用这些函数。有关详细信息，请参阅“Communications rtiostream API” (Embedded Coder)。
   3. 将您的 **rtiostream** 实现编译成共享库，以导出 **rtIOStreamOpen**、**rtIOStreamClose**、 **rtIOStreamRecv** 和 **rtIOStreamSend** 函数。
2. 使用 MATLAB **mex** 函数编译自定义的 MEX 文件可执行文件。有关 **mex** [调用的示例，请参阅“用于重新编译 ext\_comm 和 ext\_serial\_win32 MEX 文件的 MATLAB 命令” （第 32-45](#_bookmark342) 页）。

如果要保留现有 **ext\_comm** MEX 文件的功能，则不要替换该文件。在这种情况下，请使用 **-output**

选项命名新的可执行文件，例如，**my\_ext\_comm**。有关详细信息，请参阅 **mex**。

1. 在 Simulink 软件中注册您的新客户端传输层，以便能够使用 Configuration Parameters 对话框的

**Interface** [窗格为模型选择该传输。有关详细信息，请参阅“注册自定义客户端（主机）传输协议”](#_bookmark343)

[（第 32-46](#_bookmark343) 页）。

[“用于重新编译 ext\_comm 和 ext\_serial\_win32 MEX 文件的 MATLAB 命令” （第 32-45](#_bookmark342) 页）列出了用于重新编译外部模式 MEX 文件的示例命令。

#### 用于重新编译 ext\_comm 和 ext\_serial\_win32 MEX 文件的 MATLAB 命令

下表列出了在 PC 和 UNIX 平台上编译标准 **ext\_comm** 和 **ext\_serial\_win32** 模块的命令。

|  |  |
| --- | --- |
| **平台** | **命令** |
| Windows，TCP/IP | **cd (matlabroot)**  **mex toolbox/coder/simulinkcoder\_core/ext\_mode/host/common/ext\_comm.c ... toolbox/coder/simulinkcoder\_core/ext\_mode/host/common/ext\_convert.c ... toolbox/coder/simulinkcoder\_core/ext\_mode/host/common/rtiostream\_interface.c ... toolbox/coder/simulinkcoder\_core/ext\_mode/host/common/ext\_util.c ...**  **-R2018a ...**  **-Itoolbox/coder/rtiostream/src ...**  **-Itoolbox/coder/rtiostream/src/utils\_host ...**  **-Itoolbox/coder/simulinkcoder\_core/ext\_mode/host/common/include ...**  **-Irtw/c/src/ext\_mode/common ...**  **-lmwrtiostreamutils -lmwsl\_services ...**  **-DEXTMODE\_TCPIP\_TRANSPORT ...**  **-DSL\_EXT\_DLL -output my\_ext\_comm** |
| **注意 rtiostream\_interface.c** 函数将 RTIOSTREAM\_SHARED\_LIB 定义为 **libmwrtiostreamtcpip**，并动态加载 MathWorks TCP/IP **rtiostream** 共享库。如果需要加载其他 **rtiostream** 共享库，请修改此文件。 |
| Linux➅，TCP/IP | 使用 Windows 命令并进行以下更改：   * 将 **-DSL\_EXT\_DLL** 更改为 **-DSL\_EXT\_SO**。 * 用正斜杠替换反斜杠。 |

|  |  |
| --- | --- |
| **平台** | **命令** |
| Mac，TCP/IP | 使用 Windows 命令并进行以下更改：   * 将 **-DSL\_EXT\_DLL** 更改为 **-DSL\_EXT\_DYLIB**。 * 用正斜杠替换反斜杠。 |
| Windows，串行 | **cd (matlabroot)**  **mex toolbox\coder\simulinkcoder\_core\ext\_mode\host\common\ext\_comm.c ... toolbox\coder\simulinkcoder\_core\ext\_mode\host\common\ext\_convert.c ... toolbox\coder\simulinkcoder\_core\ext\_mode\host\serial\ext\_serial\_transport.c ... toolbox\coder\simulinkcoder\_core\ext\_mode\host\serial\ext\_serial\_pkt.c ... toolbox\coder\simulinkcoder\_core\ext\_mode\host\serial\rtiostream\_serial\_interface.c ... toolbox\coder\simulinkcoder\_core\ext\_mode\host\common\ext\_util.c ...**  **-R2018a ...**  **-Itoolbox\coder\rtiostream\src ...**  **-Itoolbox\coder\rtiostream\src\utils\_host ...**  **-Itoolbox\coder\simulinkcoder\_core\ext\_mode\host\common ...**  **-Itoolbox\coder\simulinkcoder\_core\ext\_mode\host\common\include ...**  **-Irtw\c\src\ext\_mode\common ...**  **-Irtw\c\src\ext\_mode\serial ...**  **-lmwrtiostreamutils -lmwsl\_services ...**  **-DEXTMODE\_SERIAL\_TRANSPORT -DSL\_EXT\_DLL ...**  **-output my\_ext\_serial\_comm** |
| **注意 rtiostream\_interface.c** 函数将 RTIOSTREAM\_SHARED\_LIB 定义为 **libmwrtiostreamserial**，并动态加载 MathWorks 串行 **rtiostream** 共享库。如果需要加载其他 **rtiostream** 共享库，请修改此文件。 |
| Linux，串行 | 使用 Windows 命令并进行以下更改：   * 将 **-DSL\_EXT\_DLL** 更改为 **-DSL\_EXT\_SO**。 * 用正斜杠替换反斜杠。 |
| Mac，串行 | 使用 Windows 命令并进行以下更改：   * 将 **-DSL\_EXT\_DLL** 更改为 **-DSL\_EXT\_DYLIB**。 * 用正斜杠替换反斜杠。 |

**注意 mex** 要求使用 MATLAB API 支持的编译器。有关 **mex** 函数的详细信息，请参阅 **mex** 参考页和

“将 MATLAB 与外部编程语言和系统集成”。

#### 注册自定义客户端（主机）传输协议

要在 Simulink 软件中注册自定义客户端传输协议，必须将以下形式的条目添加到 MATLAB 路径上的

#### sl\_customization.m 文件中：

**function sl\_customization(cm)**

**cm.ExtModeTransports.add('stf.tlc', 'transport', 'mexfile', 'Level1');**

**% -- end of sl\_customization**

其中

* **stf.tlc** 是要注册传输协议的系统目标文件的名称（例如，**'grt.tlc'**）
* **transport** 是要在 Configuration Parameters 对话框的 **Interface** 窗格的 **Transport layer** 菜单中显示的传输协议的名称（例如，**'mytcpip'**）
* **mexfile** 是传输协议的关联外部接口 MEX 文件的名称（例如，**'ext\_mytcpip\_comm'**）

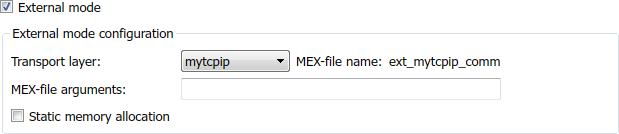
您可以通过增加 **cm.ExtModeTransports.add** 行来指定多个目标和/或传输，例如：

**function sl\_customization(cm)**

**cm.ExtModeTransports.add('grt.tlc', 'mytcpip', 'ext\_mytcpip\_comm', 'Level1'); cm.ExtModeTransports.add('ert.tlc', 'mytcpip', 'ext\_mytcpip\_comm', 'Level1');**

**% -- end of sl\_customization**

如果您将包含传输协议注册信息的 **sl\_customization.m** 文件放在 MATLAB 路径上，将在后续每个 Simulink 会话中注册您的自定义客户端传输协议。传输协议的名称将显示在 Configuration Parameters对话框的 **Interface** 窗格的 **Transport layer** 菜单中。当您为模型选择传输协议时，关联的外部接口 MEX文件的名称将显示在不可编辑的 **MEX-file name** 字段中，如下图所示。



#### 创建自定义服务器（目标）传输协议

**matlabroot/toolbox/coder/rtiostream/src/rtiostream.h** 中的 **rtIOStream\*** 函数原型可为服务器端（目标）和客户端（主机）传输层函数定义调用接口。

#### TCP/IP 实现在 matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/ rtiostream\_tcpip.c 中。

* 串行实现在 **matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/rtiostream\_serial.c**

中。

#### 注意 matlabroot/rtw/c/src/ext\_mode/common/ext\_svr\_transport.h 中的 Ext\* 函数原型在 matlabroot/rtw/c/src/ext\_mode/common/rtiostream\_interface.c 或 matlabroot/rtw/c/src/ ext\_mode/serial/rtiostream\_serial\_interface.c 中实现。大多数情况下，您不需要为自定义 TCP/IP或串行传输层修改 rtiostream\_interface.c 或 rtiostream\_serial\_interface.c。

要实现底层 TCP/IP 或串行传输协议的服务器端（目标），请执行以下操作：

#### 编辑模板 matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/rtiostream\_tcpip.c 或 matlabroot/toolbox/coder/rtiostream/src/rtiostreamserial/rtiostream\_serial.c，将底层通信调用替换为您自己的通信调用。

* 1. 复制该文件并将其重命名为 **rtiostream\_name.c**（将 **name** 替换为对您有意义的名称）。
  2. 用相同的名称将函数 **rtIOStreamOpen**、**rtIOStreamClose**、**rtIOStreamSend** 和

**rtIOStreamRecv** 替换为调用您的底层通信驱动程序的函数。

您必须实现 **rtiostream.h** 中定义的函数，而且您的实现必须符合该文件中定义的原型。有关指导信息，请参阅原始 **rtiostream\_tcpip.c** 或 **rtiostream\_serial.c**。

1. 将传输层的外部模式源文件合并到模型编译过程中。使用编译过程机制（如生成代码之后执行的命令或 **before\_make** 钩子函数）使传输文件在编译过程中可用。有关编译过程机制的详细信息，请参阅 “Customize Post-Code-Generation Build Processing”、“使用 [STF\_make\_rtw\_hook 文件自定义编译过程” （第 44-2](#_bookmark403) 页）和“Customize Build Process with sl\_customization.m”。

例如：

* 将上一步创建的文件添加到编译信息中：

**path/rtiostream\_name.c**

* 对于 TCP/IP 传输协议，请将以下文件添加到编译信息中：

**matlabroot/rtw/c/src/ext\_mode/common/rtiostream\_interface.c**

* 对于串行传输协议，请将以下文件添加到编译信息中：

**matlabroot/rtw/c/src/ext\_mode/serial/ext\_serial\_pkt.c matlabroot/rtw/c/src/ext\_mode/serial/rtiostream\_serial\_interface.c matlabroot/rtw/c/src/ext\_mode/serial/ext\_svr\_serial\_transport.c**

**注意** 对于外部模式，请检查并确保 **rtIOStreamRecv** 不是一个阻碍因素。否则，可能会导致外部模式服务器被阻拦，直到主机通过 **comm** 层发送数据为止。

#### 小于 64 字节的串行接收缓冲区

对于串行通信，如果目标的串行接收缓冲区小于 64 字节，请：

1. 使用实际的目标缓冲区大小更新以下宏：

**#define TARGET\_SERIAL\_RECEIVE\_BUFFER\_SIZE 64**

在以下文件中实现更改：

**matlabroot/rtw/c/src/ext\_mode/serial/ext\_serial\_utils.c matlabroot/toolbox/coder/simulinkcoder\_core/ext\_mode/host/serial/ext\_serial\_utils.c**

1. 运行命令以重新编译 **ext\_serial\_win32** MEX 文件。请参阅“用于重新编译 [ext\_comm 和](#_bookmark342) [ext\_serial\_win32 MEX 文件的 MATLAB 命令” （第 32-45](#_bookmark342) 页）。

### 另请参阅相关示例

* “How Generated Code Exchanges Data with an Environment”
* [“生成的代码如何存储内部信号、状态和参数数据” （第 16-2 页）](#_bookmark109)
* [“使用 TCP/IP 或串行通信的外部模式仿真” （第 32-6 页）](#_bookmark289)
* “Customize Connectivity for TCP/IP or Serial External Mode Simulations”
* [“用于参数调节、信号监控和代码执行探查的外部模式仿真” （第 32-2 页）](#_bookmark283)
* “Customize XCP Server Software”

# Simulink Coder 中的记录功能

# 使用 Simulink Coder 中的 C API 进行数据交换

## 使用 C API 在生成的代码和外部代码之间交换数据

一些 Simulink Coder 应用程序必须与信号、状态、根级输入/输出或模型生成代码中的参数进行交互。例如，校准应用程序会监控和修改参数。信号监测或数据记录应用程序会与信号、状态和根级输入/输出数据对接。使用 Simulink Coder C API，您可以编译目标应用程序，以在生成的代码执行时记录信号、状态和根级输入/输出，监控信号、状态和根级输入/输出，并调整参数。

C API 可以通过较小的结构体共享信号、状态、根级输入/输出和参数的通用信息，从而最小化其内存占 用。信号、状态、根级输入/输出和参数结构体包含结构体映射的索引，从而允许多个信号、状态、根级输入/输出或参数共享数据。

要从某个示例开始，请参阅“Use C API to Access Model Signals and States”或“Use C API to Access Model Parameters”。

[“生成的 C API 文件” （第 34-2 页）](#_bookmark349)

[“生成 C API 文件” （第 34-4 页）](#_bookmark350) [“C API 文件的说明” （第 34-5 页）](#_bookmark353)

[“生成用于与目标系统交换数据的 C API 数据定义文件” （第 34-15 页）](#_bookmark363) [“C API 的限制” （第 34-16 页）](#_bookmark364)

**本节内容**

### 生成的 C API 文件

当您将模型配置为使用 C API 时，Simulink Coder 代码生成器会生成两个附加文件 **model\_capi.c**

（或 **.cpp**）和 **model\_capi.h**，其中 **model** 是模型的名称。代码生成器根据 Configuration Parameters 对话框中的设置，将两个 C API 文件放置在编译文件夹中。C API 源代码文件包含在模型源代码的生成代码中定义的全局模块输出信号、状态、根级输入/输出以及全局参数的信息。C API 头文件是模型源代码和生成的 C API 之间的接口头文件。您可以使用这些 C API 文件中的信息来创建您的应用程序。生成的文件如下图所示。

**选择 C API 时生成的文件**

*'*

*model* . h

modey . c

—> Generate code

*model* . md1

\*

uodeJ\_capi. c

nodeJ\_capi. h

**注意** 如果您配置代码生成器使之生成支持 C API 接口和数据记录的代码，代码生成器可以在记录到 C API文件 **model\_capi.c**（或 **.cpp**）和 **model\_capi.h** 的模块路径中包含模块名称的文本。如果该文本包含模型的字符集编码中未表示的字符，代码生成器会用 XML 转义序列替换这些字符。例如，代码生成器用转义序列 **&#x30A2;** 替换日语全角片假名字母 ア。有关详细信息，请参阅 [“国际化和代码生成” （第 23-2页）](#_bookmark172)。

### 生成 C API 文件

要为您的模型生成 C API 文件，请执行以下操作：

1. 为您的模型选择 C API 接口。有两种方法可以为您的模型选择 C API 接口，如以下各节中所述。
   * [“使用 Configuration Parameters 对话框选择 C API” （第 34-4 页）](#_bookmark351)
   * [“从命令行中选择 C API” （第 34-4 页）](#_bookmark352)
2. 为您的模型生成代码。

生成代码后，您可以检查模型编译文件夹中的文件 **model\_capi.c**（或 **.cpp**）和 **model\_capi.h**。

#### 使用 Configuration Parameters 对话框选择 C API

1. 打开您的模型，并打开 Configuration Parameters 对话框。
2. 在 **Code Generation** > **Interface** 窗格中，在 **Data exchange interface** 子组中，选择一个或多个 “**C API**” 选项。根据您选择的选项，C API 生成的代码可支持访问信号、参数、状态和根级 I/O。
   * 如果您要为全局模块输出信号生成 C API 代码，请选择 **Generate C API for: signals**。
   * 如果要为全局模块参数生成 C API 代码，请选择 **Generate C API for: parameters**。
   * 如果要为离散和连续状态生成 C API 代码，请选择 **Generate C API for: states**。
   * 如果要为根级输入和输出生成 C API 代码，请选择 **Generate C API for: root-level I/O**。

#### 从命令行中选择 C API

从 MATLAB 命令行中，您可以使用 **set\_param** 函数来选中或清除 C API 模型配置参数。在 MATLAB 命令行中，输入以下一个或多个命令，其中 **modelname** 是您的模型名称。

要选中 **Generate C API for: signals**，请输入：

**set\_param('modelname','RTWCAPISignals','on')**

要清除 **Generate C API for: signals**，请输入：

**set\_param('modelname','RTWCAPISignals','off')**

要选中 **Generate C API for: parameters**，请输入：

**set\_param('modelname','RTWCAPIParams','on')**

要清除 **Generate C API for: parameters**，请输入：

**set\_param('modelname','RTWCAPIParams','off')**

要选中 **Generate C API for: states**，请输入：

**set\_param('modelname','RTWCAPIStates','on')**

要清除 **Generate C API for: states**，请输入：

**set\_param('modelname','RTWCAPIStates','off')**

要选中 **Generate C API for: root-level I/O**，请输入：

**set\_param('modelname','RTWCAPIRootIO','on')**

要清除 **Generate C API for: root-level I/O**，请输入：

**set\_param('modelname','RTWCAPIRootIO','off')**

### C API 文件的说明

* [“关于 C API 文件” （第 34-5 页）](#_bookmark354)
* [“在 C API 文件中生成的结构体数组” （第 34-7 页）](#_bookmark355)
* [“生成示例 C API 文件” （第 34-8 页）](#_bookmark356)
* [“C API 信号” （第 34-9 页）](#_bookmark357)
* [“C API 状态” （第 34-11 页）](#_bookmark358)
* [“C API 根级输入和输出” （第 34-12 页）](#_bookmark359)
* [“C API 参数” （第 34-13 页）](#_bookmark360)
* [“将 C API 数据结构体映射到实时模型” （第 34-14 页）](#_bookmark361)

#### 关于 C API 文件

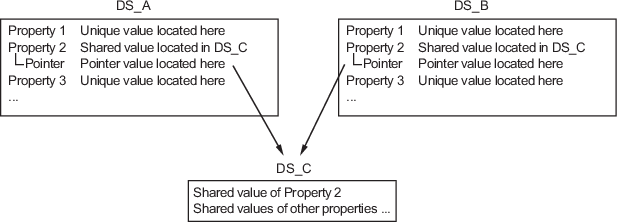
**model\_capi.c**（或 **.cpp**）文件为外部应用程序提供一致的模型数据接口。根据您的配置设置，数据可以是信号、状态、根级输入/输出或者参数。在本文档中，数据项可指信号、状态、根级输入/输出或者参

数。C API 使用提供数据项属性接口的结构体。该接口将每个数据项的属性打包到一个数据结构体中。如果模型包含多个数据项，接口会生成一个数据结构体数组。数据结构体的成员映射到数据属性。

为了与数据项对接，应用程序要求每个数据项具有以下属性：

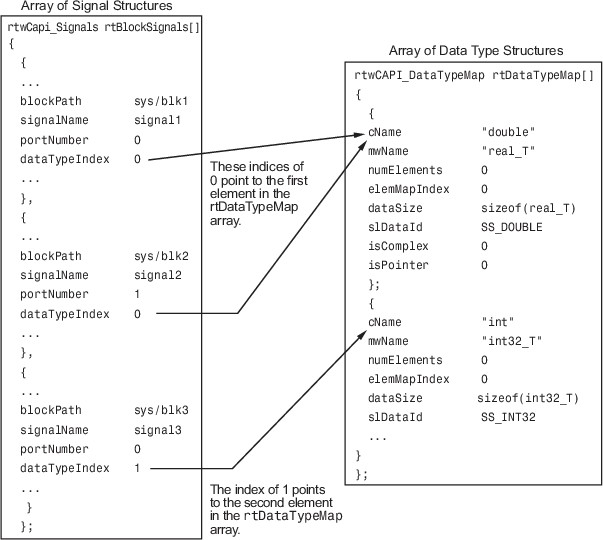
* 名称
* 模块路径
* 端口号（仅适用于信号和根级输入/输出）
* 地址
* 数据类型信息：本机数据类型、数据大小、复/实性和其他属性
* 维度信息：行数、列数和数据方向（标量、向量、矩阵或 n 维）
* 定点信息：斜率、偏置、定标类型、字长、指数和其他属性
* 采样时间信息（仅适用于信号、状态和根级输入/输出）：采样时间、任务标识符、帧

如下图中的示例所示，数据项 A 的属性位于数据结构体 DS\_A 中。数据项 B 的属性位于数据结构体 DS\_B中。



一些属性值对于每个数据项是唯一的，一些属性值则可在几个数据项之间通用。例如，名称对于每个数据项都有唯一的值。接口将唯一属性值直接放置在数据项的结构体中。数据项 A 的名称值在 DS\_A 中，数据项 B 的名称值在 DS\_B 中。

但是，数据类型这一属性的值可在几个数据项之间通用。由于一些数据项能够共享属性，因此 C API 具有重用特性。在本例中，接口仅在 DS\_A 和 DS\_B 中分别放置一个索引值。这些索引指向另一个数据结构体 DS\_C，其中包含实际数据类型值。下图更详细地展示了此方案。



该图显示三个信号。**signal1** 和 **signal2** 都具有数据类型 **double**。接口在结构体中只提供索引值 0，而不是在每个信号数据结构体中指定该数据类型值。**"double"** 由 **rtDataTypeMap** 数组中的条目 0 说明，两个信号都引用该条目。此外，属性值可以在信号、状态、根级输入/输出和参数之间共享，因此状态、根级输入/输出和参数也可以引用 **rtDataTypeMap** 数组中的 **double** 条目。这种信息重用可减少生成的接口所需的内存大小。

#### 在 C API 文件中生成的结构体数组

接口像处理数据类型一样，将其他通用属性（如地址、维度、定点定标和采样时间）映射到单独的结构体中，并在数据项的结构体中提供相应索引。有关结构体定义的完整列表，请参考文件 **matlabroot/rtw/c/src/rtw\_capi.h**。该文件还说明了结构体中的每个成员。**model\_capi.c**（或 **.cpp**）文件中生成的结构体数组具有在 **rtw\_capi.h** 文件中定义的结构体类型。以下是在 **model\_capi.c**

（或 **.cpp**）中生成的结构体数组的简要说明：

* **rtBlockSignals** 是一个结构体数组，包含关于模型中全局模块输出信号的信息。数组中的每个元素都具有 **struct rtwCAPI\_Signals** 类型。该结构体的成员提供信号名称、模块路径、模块端口号、地址以及指向数据类型、维度、定点和采样时间这些结构体数组的索引。
* **rtBlockParameters** 是结构体数组，包含关于模型中的可调模块参数的信息，按模块名称和参数名称列出。数组中的每个元素都具有 **struct rtwCAPI\_BlockParameters** 类型。该结构体的成员提供参数名称、模块路径、地址以及指向数据类型、维度和定点这些结构体数组的索引。
* **rtBlockStates** 是一个结构体数组，包含关于模型中离散和连续状态的信息。数组中的每个元素都具有 **struct rtwCAPI\_States** 类型。该结构体的成员提供状态名称、模块路径、类型（连续或离散）以及指向地址、数据类型、维度、定点和采样时间这些结构体数组的索引。
* **rtRootInputs** 是一个结构体数组，包含关于模型中根级输入的信息。数组中的每个元素都具有 **struct rtwCAPI\_Signals** 类型。该结构体的成员提供根级输入名称、模块路径、模块端口号、地址以及指向数据类型、维度、定点和采样时间这些结构体数组的索引。
* **rtRootOutputs** 是一个结构体数组，包含关于模型中根级输出的信息。数组中的每个元素都具有 **struct rtwCAPI\_Signals** 类型。该结构体的成员提供根级输出名称、模块路径、模块端口号、地址以及指向数据类型、维度、定点和采样时间这些结构体数组的索引。
* **rtModelParameters** 是一个结构体数组，其中包含由模型中的一个或多个模块或 Stateflow 图引用为模块参数的工作区变量的信息。数组中的每个元素都具有数据类型 **rtwCAPI\_ModelParameters**。该结构体的成员提供变量名称、地址以及指向数据类型、维度和定点这些结构体数组的索引。
* **rtDataAddrMap** 是一个数组，包含出现在 **rtBlockSignals**、**rtBlockParameters**、 **rtBlockStates** 和 **rtModelParameters** 数组中的信号、状态、根级输入/输出和参数的基址。 **rtDataAddrMap** 数组的每个元素均为指向 **void (void\*)** 的指针。
* **rtDataTypeMap** 是一个结构体数组，包含关于模型中各种数据类型的信息。该数组的每个元素都具有 **struct rtwCAPI\_DataTypeMap** 类型。该结构体的成员提供数据类型的名称、数据类型的大小以及关于数据是否为复数的信息。
* **rtDimensionMap** 是一个结构体数组，包含关于模型中各种数据维度的信息。该数组的每个元素都具有 **struct rtwCAPI\_DimensionMap** 类型。该结构体的成员提供关于数据中的维数、数据的方向

（标量、向量还是矩阵）以及数据实际维度的信息。

* **rtFixPtMap** 是一个结构体数组，包含关于信号、状态、根级输入/输出和参数的定点信息。该数组的每个元素都具有 **struct rtwCAPI\_FixPtMap** 类型。该结构体的成员提供关于数据定标、偏置、指数以及定点数据是否有符号的信息。如果模型没有定点数据（信号、状态、根级输入/输出或参数）， Simulink Coder 软件会将 **NULL** 或零值赋给 **rtFixPtMap** 数组的元素。
* **rtSampleTimeMap** 是一个结构体数组，包含关于模型中全局信号、状态和根级输入/输出的采样信息。（该数组不包含关于参数的信息。）该数组的每个元素都具有 **struct rtwCAPI\_SampleTimeMap** 类型。该结构体的成员提供关于采样周期、偏移以及数据基于帧还是基于采样的信息。

#### 生成示例 C API 文件

子主题 [“C API 信号” （第 34-9](#_bookmark357) 页）、“C [API 状态” （第 34-11](#_bookmark358) 页）、“C [API 根级输入和输](#_bookmark359)

[出” （第 34-12](#_bookmark359) 页）和 [“C API 参数” （第 34-13](#_bookmark360) 页）讨论使用示例模型 **rtwdemo\_capi** 生成的 C API 结构体。要从示例模型生成代码，请执行以下操作：

1. 通过点击上面的 **rtwdemo\_capi** 链接或在 MATLAB 命令行中键入 **rtwdemo\_capi** 打开模型。
2. 如果您要为 **rtwdemo\_capi** 中的根级输入/输出生成 C API 结构体，请选择模型配置参数 **Generate C API for: root-level I/O**。

此参数的设置必须在顶层模型和引用模型之间匹配。如果修改了参数设置，请将顶层模型和引用模型保存到同一可写工作文件夹中。

1. 为模型生成代码。

下一个子主题中的 C API 代码示例是用 C 语言作为目标语言生成的。

该模型有三个全局模块输出信号，这些信号将出现在 C API 生成代码中：

* **top\_sig1**，它是顶层模型中 Gain1 模块输出的测试点
* **sig2\_eg**，它出现在顶层模型中，并在基础工作区中定义为具有存储类 **ExportedGlobal** 的

#### Simulink.Signal 对象

* **bot\_sig1**，它出现在引用模型 **rtwdemo\_capi\_bot** 中，并定义为具有存储类 **Model default** 的

#### Simulink.Signal 对象

该模型还具有两个离散状态，它们将出现在 C API 生成代码中：

* **top\_state**，它是为顶层模型中的 Delay1 模块定义的
* **bot\_state**，它是为引用模型中的 Discrete Filter 模块定义的

该模型具有根级输入/输出，如果您选择模型配置参数 **Generate C API for: root-level I/O**，这些输入/输出将出现在 C API 生成代码中：

* 四个根级输入，**In1** 至 **In4**
* 六个根级输出，**Out1** 至 **Out6**

此外，该模型有五个全局模块参数，这些参数将出现在 C API 生成代码中：

* **Kp**（顶层模型 **Gain1** 模块和引用模型 **Gain2** 模块共享）
* **Ki**（引用模型 **Gain3** 模块）
* **p1**（查找表 **lu1d**）
* **p2**（查找表 **lu2d**）
* **p3**（查找表 **lu3d**）

#### C API 信号

**rtwCAPI\_Signals** 结构体捕获信号信息，包括信号名称、地址、模块路径、输出端口号、数据类型信息、维度信息、定点信息和采样时间信息。

以下是 **rtwdemo\_capi\_capi.c** 中的代码节，它提供关于 **rtwdemo\_capi** 中顶层模型的 C API 信号的信息：

**/\* Block output signal information \*/**

**static const rtwCAPI\_Signals rtBlockSignals[] = {**

**/\* addrMapIndex, sysNum, blockPath,**

**\* signalName, portNumber, dataTypeIndex, dimIndex, fxpIndex, sTimeIndex**

**\*/**

**{ 0, 0, "rtwdemo\_capi/Gain1",**

**"top\_sig1", 0, 0, 0, 0, 0 },**

**{ 1, 0, "rtwdemo\_capi/lu2d",**

**"sig2\_eg", 0, 0, 1, 0, 0 },**

**{**

**0, 0, (NULL), (NULL), 0, 0, 0, 0, 0**

**}**

**};**

**注意** 为了更好地理解代码，请阅读文件中的注释。例如，请注意前面代码中从第三行开始的注释。此注释按顺序列出 **rtwCAPI\_Signals** 结构体的成员。这将告诉您信号中每个成员的赋值出现的顺序。在此示例中，注释告诉您，**signalName** 是该结构体的第四个成员。以下几行代码描述了第一个信号：

**{ 0, 0, "rtwdemo\_capi/Gain1",**

**"top\_sig1", 0, 0, 0, 0, 0 },**

从这些代码中，您可以推断出第一个信号的名称是 **top\_sig1**。

除最后一个数组元素外，每个数组元素描述模块信号的一个输出端口。最后一个数组元素是哨兵值，所有元素都设置为空值。例如，检查由以下代码描述的第二个信号：

**{ 1, 0, "rtwdemo\_capi/lu2d",**

**"sig2\_eg", 0, 0, 1, 0, 0 },**

该信号名为 **sig2\_eg**，是 **rtwdemo\_capi/lu2d** 模块第一个端口的输出信号。（此端口是第一个端口，因为第二行显示的从零开始的 **portNumber** 索引赋值为 **0**。）

该信号的地址由 **addrMapIndex** 给出，在此示例中，它显示在第一行，值为 **1**。该值提供指向 **rtDataAddrMap** 数组的索引，该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到：

**/\* Declare Data Addresses statically \*/ static void\* rtDataAddrMap[] = {**

**&rtwdemo\_capi\_B.top\_sig1, /\* 0: Signal \*/ &sig2\_eg[0], /\* 1: Signal \*/ &rtwdemo\_capi\_DWork.top\_state, /\* 2: Discrete State \*/ &rtP\_Ki, /\* 3: Model Parameter \*/**

**&rtP\_Kp, /\* 4: Model Parameter \*/**

**&rtP\_p1[0], /\* 5: Model Parameter \*/**

**&rtP\_p2[0], /\* 6: Model Parameter \*/**

**&rtP\_p3[0], /\* 7: Model Parameter \*/**

**};**

索引 **1** 指向 **rtDataAddrMap** 数组中的第二个元素。从 **rtDataAddrMap** 数组中，您可以推断出该信号的地址是 **&sig2\_eg[0]**。

这种间接指定可用于同一模型中的多个代码实例。对于多个实例，信号信息保持不变，但地址除外。在本例中，模型是单个实例。因此，代码静态声明 **rtDataAddrMap**。如果您选择生成可重用代码，将生成初[始化函数，该函数会动态地初始化每个实例的地址。有关生成可重用代码的详细信息，请参阅“为模型入口函数配置生成的 C 函数接口” （第 17-15](#_bookmark148) 页）和“Configure Code Reuse Support” (Embedded Coder)。

**dataTypeIndex** 提供指向 **rtDataTypeMap** 数组的索引（该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到），指示信号的数据类型：

**/\* Data Type Map - use dataTypeMapIndex to access this structure \*/ static const rtwCAPI\_DataTypeMap rtDataTypeMap[] = {**

**/\* cName, mwName, numElements, elemMapIndex, dataSize, slDataId, \***

**\* isComplex, isPointer \*/**

**{ "double", "real\_T", 0, 0, sizeof(real\_T), SS\_DOUBLE, 0, 0 }**

**};**

由于 **sig2\_eg** 的索引是 **0**，因此该索引指向数组中的第一个结构体元素。您可以推断信号的数据类型是 **double**。**isComplex** 的值是 **0**，表明信号不是复信号。此处没有直接在 **rtwCAPI\_Signals** 结构体中提供数据类型信息，而是采用了间接指定。间接指定允许具有相同数据类型的多个信号指向同一个映射结构体，从而为每个信号节省内存。

**dimIndex**（维度索引）提供指向 **rtDimensionMap** 数组的索引（该数组可在 **rtwdemo\_capi\_capi.c**的后续部分中找到），指示信号的维度。由于 **sig2\_eg** 的索引是 **1**，因此该索引指向 **rtDimensionMap**数组中的第二个元素：

**/\* Dimension Map - use dimensionMapIndex to access elements of ths structure\*/ static const rtwCAPI\_DimensionMap rtDimensionMap[] = {**

**/\* dataOrientation, dimArrayIndex, numDims, vardimsIndex \*/**

**{ rtwCAPI\_SCALAR, 0, 2, 0 },**

**{ rtwCAPI\_VECTOR, 2, 2, 0 },**

**...**

**};**

从这个结构体中，您可以推断这是维度为 **2** 的非标量信号。**dimArrayIndex** 值 2 提供指向

**rtDimensionArray** 的索引，该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到：

**/\* Dimension Array- use dimArrayIndex to access elements of this array \*/ static const uint\_T rtDimensionArray[] = {**

**1, /\* 0 \*/**

**1, /\* 1 \*/**

**2, /\* 2 \*/**

**...**

**};**

**fxpIndex**（定点索引）提供指向 **rtFixPtMap** 数组的索引（该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到），指示关于信号的定点信息。您的代码可以使用定标信息来计算信号的真实值，所用的公式为 **V=SQ+B**，其中 V 是“真实”（即基数为 10）的值，S 是用户指定的斜率，Q 是“量化的定点值”或“存储的整数”，而 B 是用户指定的偏置。有关详细信息，请参阅“定标” (Fixed-Point Designer)。

由于 **sig2\_eg** 的该索引是 **0**，因此信号没有定点信息。定点映射索引为零意味着信号没有定点信息。 **sTimeIndex**（采样时间索引）提供 **rtSampleTimeMap** 数组的索引（该数组可在

**rtwdemo\_capi\_capi.c** 的后续部分中找到），指示信号的任务信息。如果记录多速率信号或条件执行信

号，采样信息会很有帮助。

**注意 model\_capi.c**（或 **.cpp**）包括 **rtw\_capi.h**。引用 **rtBlockSignals** 数组的源文件还必须包括

**rtw\_capi.h**。

#### C API 状态

**rtwCAPI\_States** 结构体捕获状态信息，包括状态名称、地址、模块路径、类型（连续或离散）、数据类型信息、维度信息、定点信息和采样时间信息。

以下是 **rtwdemo\_capi\_capi.c** 中的代码节，它提供关于 **rtwdemo\_capi** 中顶层模型的 C API 状态的信息：

**/\* Block states information \*/**

**static const rtwCAPI\_States rtBlockStates[] = {**

**/\* addrMapIndex, contStateStartIndex, blockPath,**

* **stateName, pathAlias, dWorkIndex, dataTypeIndex, dimIndex,**
* **fixPtIdx, sTimeIndex, isContinuous**

**\*/**

**{ 2, -1, "rtwdemo\_capi/Delay1",**

**"top\_state", "", 0, 0, 0, 0, 0, 0 },**

**{**

**0, -1, (NULL), (NULL), (NULL), 0, 0, 0, 0, 0, 0**

**}**

**};**

除了最后一个数组元素，每个数组元素都描述模型中的一个状态。最后一个数组元素是哨兵值，所有元素都设置为空值。在此示例中，顶层模型的 C API 代码显示一个状态：

**{ 2, -1, "rtwdemo\_capi/Delay1",**

**"top\_state", "", 0, 0, 0, 0, 0, 0 },**

该状态名为 **top\_state**，是为 **rtwdemo\_capi/Delay1** 模块定义的。**isContinuous** 的值为零，表明该状态是离散的，而不是连续的。其他字段对应于 [“C API 信号” （第 34-9](#_bookmark357) 页）中具有类似命名的等效信号项，如下所示：

* 信号地址由 **addrMapIndex** 给出，在此示例中，该地址为 **2**。这是指向 **rtDataAddrMap** 数组的索引，该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到。由于该索引从 0 开始，因此 **2** 对应于 **rtDataAddrMap** 中的第三个元素，即 **&rtwdemo\_capi\_DWork.top\_state**。
* **dataTypeIndex** 提供指向 **rtDataTypeMap** 数组的索引（该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到），指示参数的数据类型。值 0 对应于双精度非复参数。
* **dimIndex**（维度索引）提供指向 **rtDimensionMap** 数组的索引，该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到。值 0 对应于第一个条目，即 **{ rtwCAPI\_SCALAR, 0, 2, 0 }**。
* **fixPtIndex**（定点索引）提供指向 **rtFixPtMap** 数组的索引（该数组可在 **rtwdemo\_capi\_capi.c**的后续部分中找到），指示关于参数的定点信息。与对应的信号属性一样，定点映射索引为零意味着参数没有定点信息。

#### C API 根级输入和输出

**rtwCAPI\_Signals** 结构体捕获根级输入/输出信息，包括输入/输出名称、地址、模块路径、端口号、数据类型信息、维度信息、定点信息和采样时间信息。（此结构体也用于模块输出信号，如前面在 [“C API信号” （第 34-9](#_bookmark357) 页）中所述。）

以下是 **rtwdemo\_capi\_capi.c** 中的代码节，它提供关于 **rtwdemo\_capi** 中顶层模型的 C API 根级输入/输出的信息：

**/\* Root Inputs information \*/**

**static const rtwCAPI\_Signals rtRootInputs[] = {**

**/\* addrMapIndex, sysNum, blockPath,**

**\* signalName, portNumber, dataTypeIndex, dimIndex, fxpIndex, sTimeIndex**

**\*/**

**{ 3, 0, "rtwdemo\_capi/In1",**

**"", 1, 0, 0, 0, 0 },**

**{ 4, 0, "rtwdemo\_capi/In2",**

**"", 2, 0, 0, 0, 0 },**

**{ 5, 0, "rtwdemo\_capi/In3",**

**"", 3, 0, 0, 0, 0 },**

**{ 6, 0, "rtwdemo\_capi/In4",**

**"", 4, 0, 0, 0, 0 },**

**{**

**0, 0, (NULL), (NULL), 0, 0, 0, 0, 0**

**}**

**};**

**/\* Root Outputs information \*/**

**static const rtwCAPI\_Signals rtRootOutputs[] = {**

**/\* addrMapIndex, sysNum, blockPath,**

**\* signalName, portNumber, dataTypeIndex, dimIndex, fxpIndex, sTimeIndex**

**\*/**

**{ 7, 0, "rtwdemo\_capi/Out1",**

**"", 1, 0, 0, 0, 0 },**

**{ 8, 0, "rtwdemo\_capi/Out2",**

**"", 2, 0, 0, 0, 0 },**

**{ 9, 0, "rtwdemo\_capi/Out3",**

**"", 3, 0, 0, 0, 0 },**

**{ 10, 0, "rtwdemo\_capi/Out4",**

**"", 4, 0, 0, 0, 0 },**

**{ 11, 0, "rtwdemo\_capi/Out5",**

**"sig2\_eg", 5, 0, 1, 0, 0 },**

**{ 12, 0, "rtwdemo\_capi/Out6",**

**"", 6, 0, 1, 0, 0 },**

**{**

**0, 0, (NULL), (NULL), 0, 0, 0, 0, 0**

**}**

**};**

**注意** 生成 C++ 代码时，代码生成器不提供关于 C API 根级输入/输出的信息。

有关解释 **rtwCAPI\_Signals** 结构体中的值的信息，请参阅前一节 [“C API 信号” （第 34-9](#_bookmark357) 页）。

#### C API 参数

**rtwCAPI\_BlockParameters** 和 **rtwCAPI\_ModelParameters** 结构体捕获参数信息，包括参数名称、模块路径（对于模块参数）、地址、数据类型信息、维度信息和定点信息。

**rtModelParameters** 数组包含工作区变量的条目，这些变量被引用为可调的 Simulink 模块参数或具有 Stateflow 状态机作用域的数据。例如，可调参数包括使用 **Auto** 以外的存储类的 **Simulink.Parameter**对象。如果没有此种数据，Simulink Coder 软件仅为该数组元素赋予 **NULL** 或零值。

您为模型配置参数 **Default parameter behavior** 选择的设置将决定以何种方式将信息生成为

**model\_capi.c**（或 **.cpp**）中的 **rtBlockParameters** 数组。

* 如果您将 **Default parameter behavior** 设置为 “**Tunable**”，则模型中每个模块的每个可修改参数在 **rtBlockParameters** 数组中都有一个对应的条目。但是，如果使用 MATLAB 变量或可调参数来指定模块参数，则模块参数不会出现在 **rtBlockParameters** 中。取而代之，变量或可调参数会出现在 **rtModelParameters** 中。
* 如果您将 **Default parameter behavior** 设置为 “**Inlined**”，则 **rtBlockParameters** 数组为空。 Simulink Coder 软件仅为其元素赋予 **NULL** 或零值。

每个数组的最后一个成员是哨兵值，所有元素都设置为空值。

这是在 **rtwdemo\_capi\_capi.c** 中默认生成的 **rtBlockParameters** 数组：

**/\* Individual block tuning is not valid when inline parameters is \***

* **selected. An empty map is produced to provide a consistent \***
* **interface independent of inlining parameters. \***

**\*/**

**static const rtwCAPI\_BlockParameters rtBlockParameters[] = {**

**/\* addrMapIndex, blockPath,**

**\* paramName, dataTypeIndex, dimIndex, fixPtIdx**

**\*/**

**{**

**0, (NULL), (NULL), 0, 0, 0**

**}**

**};**

在此示例中，只生成最后一个作为哨兵值的数组元素，结构体 **rtwCAPI\_BlockParameters** 的所有成员都设置为 **NULL** 和零值。这是因为对于 **rtwdemo\_capi** 示例模型，默认情况下 **Default parameter behavior** 设置为 “**Inlined**”。如果将 **Default parameter behavior** 设置为 “**Tunable**”，则将在 **rtwCAPI\_BlockParameters** 结构体中生成模型参数。但是，MATLAB 变量和可调参数出现在 **rtwCAPI\_ModelParameters** 结构体中。

这是在 **rtwdemo\_capi\_capi.c** 中默认生成的 **rtModelParameters** 数组：

**/\* Tunable variable parameters \*/**

**static const rtwCAPI\_ModelParameters rtModelParameters[] = {**

**/\* addrMapIndex, varName, dataTypeIndex, dimIndex, fixPtIndex \*/**

**{ 2, TARGET\_STRING("Ki"), 0, 0, 0 },**

**{ 3, TARGET\_STRING("Kp"), 0, 0, 0 },**

**{ 4, TARGET\_STRING("p1"), 0, 2, 0 },**

**{ 5, TARGET\_STRING("p2"), 0, 3, 0 },**

**{ 6, TARGET\_STRING("p3"), 0, 4, 0 },**

**{ 0, (NULL), 0, 0, 0 }**

**};**

在此示例中，**rtModelParameters** 数组包含的各条目对应于可调 Simulink 模块参数引用的每个变量。

例如，第四个参数的 **varName**（变量名称）是 **p2**。其他字段对应于 [“C API 信号” （第 34-9](#_bookmark357) 页）中具有类似命名的等效信号项，如下所示：

* 第四个参数的地址由 **addrMapIndex** 给出，在此示例中，它是 **5**。这是指向 **rtDataAddrMap** 数组的索引，该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到。由于该索引从 0 开始，因此 **5** 对应于 **rtDataAddrMap** 中的第六个元素，即 **rtP\_p2**。
* **dataTypeIndex** 提供指向 **rtDataTypeMap** 数组的索引（该数组可在 **rtwdemo\_capi\_capi.c** 的后续部分中找到），指示参数的数据类型。值 0 对应于双精度非复参数。
* **dimIndex**（维度索引）提供指向 **rtDimensionMap** 数组的索引，该数组可在

**rtwdemo\_capi\_capi.c** 的后续部分中找到。值 3 对应于第四个条目，即

#### { rtwCAPI\_MATRIX\_COL\_MAJOR, 6, 2, 0 }。

* **fixPtIndex**（定点索引）提供指向 **rtFixPtMap** 数组的索引（该数组可在 **rtwdemo\_capi\_capi.c**的后续部分中找到），指示关于参数的定点信息。与对应的信号属性一样，定点映射索引为零意味着参数没有定点信息。

[有关生成代码中可调参数存储的详细信息，请参阅“生成的代码如何存储内部信号、状态和参数数据”](#_bookmark109)

[（第 16-2](#_bookmark109) 页）。

#### 将 C API 数据结构体映射到实时模型

实时模型数据结构体封装了模型数据和全面描述模型的关联信息。当您选择 C API 功能并生成代码时， Simulink Coder 代码生成器会向 **model.h** 中生成的实时模型数据结构体添加另一个成员：

**/\***

* **DataMapInfo:**
* **The following substructure contains information regarding**
* **structures generated in the model's C API.**

**\*/ struct {**

**rtwCAPI\_ModelMappingInfo mmi;**

**} DataMapInfo;**

该成员定义 **struct rtwCAPI\_ModelMappingInfo** 类型的 **mmi**（表示模型映射信息）。该结构体位于 **matlabroot/rtw/c/src/rtw\_modelmap.h** 中。**mmi** 子结构体定义模型和 C API 文件之间的接口。更具体地说，**mmi** 的成员将实时模型数据结构体映射到 **model\_capi.c**（或 **.cpp**）中的结构体。

将 **mmi** 成员的值初始化为数组即完成映射，如“将模型映射到各个 [C API 结构体数组” （第 34-15](#_bookmark362)

[页）中所示。每个成员都指向生成的](#_bookmark362) C API 文件中的一个结构体数组。例如，使用 **rtw\_modelmap.h** 文件中的以下代码，会将 **rtBlockSignals** 结构体数组的地址分配给 **model.c**（或 **.cpp**）中 **mmi** 子结构体的第一个成员：

**/\* signals \*/ struct {**

**rtwCAPI\_Signals const \*signals; /\* Signals Array \*/ uint\_T numSignals; /\* Num Signals \*/ rtwCAPI\_Signals const \*rootInputs; /\* Root Inputs array \*/ uint\_T numRootInputs; /\* Num Root Inputs \*/**

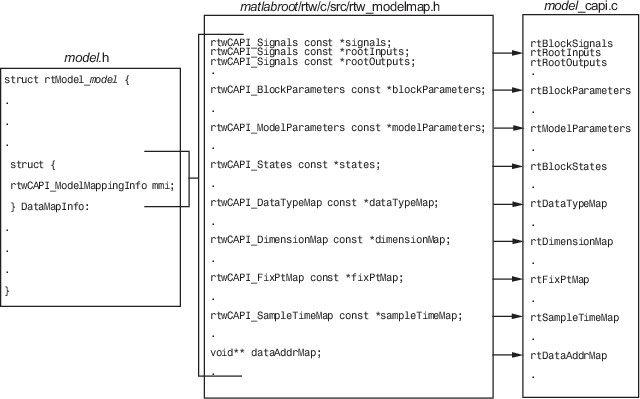
**rtwCAPI\_Signals const \*rootOutputs; /\* Root Outputs array \*/ uint\_T numRootOutputs;/\* Num Root Outputs \*/**

**} Signals;**

**model.c**（或 **.cpp**）中的模型初始化函数通过调用 C API 初始化函数来执行初始化。例如，示例模型

**rtwdemo\_capi** 的模型初始化函数中包含以下生成代码：

**/\* Initialize DataMapInfo substructure containing ModelMap for C API \*/ rtwdemo\_capi\_InitializeDataMapInfo(rtwdemo\_capi\_M);**



#### 将模型映射到各个 C API 结构体数组

**注意** 该图将数组按其结构体在 **rtw\_modelmap.h** 中的出现顺序列出，这与它们在 **model\_capi.c** 中生成的顺序略有不同。

### 生成用于与目标系统交换数据的 C API 数据定义文件

此示例说明如何使用基于目标的 C API 与表示信号、状态、参数和根级 I/O 的生成代码对接。

#### 打开示例模型

打开示例模型 **rtwdemo\_capi**。

**open\_system('rtwdemo\_capi');**

C API 可用于与生成的代码中的应用程序数据进行交互，而不必停止程序执行或重新编译生成的代码。要使用 C API 接口，对于顶层模型及其引用模型，请执行以下操作：

1. 在开发计算机和目标计算机之间设置客户端/服务器协议（如 TCP/IP 或双端口内存连接）。
2. 选择至少一个 C API 模型配置参数：**signals**、**parameters**、**states** 和 **root-level I/O**。 3.用可寻址的存储类配置您要使用 C API 访问的数据元素。

顶层模型和引用模型的 C API 配置设置必须匹配。

代码生成器将 C API 接口放在文件 **model\_capi.c** 中。根据您的配置设置，数据可以表示使用可寻址存储类配置的信号、状态、参数和根级 I/O。该文件包括提供数据属性接口的结构体。

### C API 的限制

C API 功能具有以下限制。

* + C API 不支持 **CodeFormat** TLC 变量的下列值：

#### S-Function

* + - **Accelerator\_S-Function**（用于加速仿真）
  + 对于基于 ERT 的目标，C API 要求启用对浮点代码的支持。
  + 不支持局部模块输出信号。
  + 不支持局部 Stateflow 参数。
  + 不支持以下自定义存储类对象：
    - 没有包 **csc\_registration** 文件的对象
    - 已分组的自定义存储类
    - 使用宏定义的对象

#### BitField 对象

* + - **FileScope** 对象
  + 使用 C API 时，自定义的数据放置被禁用。该接口在 **model.h** 和 **model\_private.h** 中查找全局数据声明。当自定义数据放置将声明放置于任何其他文件中，将导致代码无法编译。

**注意** 仅当您使用 ERT 系统目标文件并清除模型配置参数 **Ignore custom storage classes** 时，自定义存储类对象才能在代码生成中工作。

### 另请参阅

**相关示例**

* + “Access Signal, State, and Parameter Data During Execution”
  + “How Generated Code Exchanges Data with an Environment”
  + [“生成的代码如何存储内部信号、状态和参数数据” （第 16-2 页）](#_bookmark109)

# Simulink Coder 中的 ASAP2 数据测量和标定

## 导出 ASAP2 文件用于数据测量和标定

ASAM MCD-2 MC 标准，也称为 ASAP2，是由自动化和测量系统标准化协会 (ASAM) 提出的数据定义标准。ASAP2 是用于测量、标定和诊断系统的数据的非面向对象的描述。有关 ASAM 和 ASAM MCD-2 MC (ASAP2) 标准的详细信息，请参阅 ASAM 网站，网址为 [**www.asam.net**](https://www.asam.net/)。

通过代码生成器，您可以在代码生成过程中导出包含模型信息的 ASAP2 文件。

Simulink Coder 对 ASAP2 文件生成的支持与版本无关。默认情况下，软件生成 ASAP2 版本 1.31 格式，但是，生成的模型信息通常与其他 ASAP2 版本兼容。ASAP2 文件生成过程也与 ASAP2 测量和标定工具的特定需求无关。该软件提供自定义 API，您可以使用这些 API 自定义 ASAP2 文件生成，以生成某一 ASAP2 版本并满足您的 ASAP2 工具的特定需求。

[“基础知识” （第 35-2 页）](#_bookmark367)

[“支持 ASAP2 的系统目标文件” （第 35-2 页）](#_bookmark368) [“定义 ASAP2 信息” （第 35-2 页）](#_bookmark369)

[“生成 ASAP2 文件” （第 35-6 页）](#_bookmark374) [“ASAP2 文件的结构” （第 35-7 页）](#_bookmark379)

**本节内容**

### 基础知识

要使用 ASAP2 文件生成，您应该熟悉以下各项：

* ASAM 和 ASAP2 标准和术语。请访问 ASAM 网站，网址为 [**www.asam.net**](https://www.asam.net/)。
* 为代码生成配置模型数据元素。请参阅**代码映射编辑器 - C**。
* 生成代码中信号和参数的存储和表示。请参阅“原型构建和调试中的数据访问”。
* 如果您有 Embedded Coder，请参阅“数据表示” (Embedded Coder)。

### 支持 ASAP2 的系统目标文件

您可以基于不同的系统目标文件配置来生成 ASAP2 文件。例如，基于 GRT 和基于 ERT 的配置使您能够在代码生成和编译过程中生成 ASAP2 文件。

对于自适应 AUTOSAR 目标和 Simulink Real-Time 目标，可以生成 C++ 语言的 ASAP2 文件。对于 ERT和 GRT 目标，只能生成 C 语言的 ASAP2 文件，不支持 C++ 语言。

有关使用系统目标文件配置生成 ASAP2 文件的过程，请参阅“生成 [ASAP2 文件” （第 35-6](#_bookmark374) 页）。

### 定义 ASAP2 信息

* [“为参数和信号定义 ASAP2 信息” （第 35-3 页）](#_bookmark370)
* [“内存地址属性” （第 35-3 页）](#_bookmark371)
* [“ASAP2 文件的自动 ECU 地址替换 (Embedded Coder)” （第 35-4 页）](#_bookmark372)
* [“为查找表定义 ASAP2 信息” （第 35-4 页）](#_bookmark373)

#### 为参数和信号定义 ASAP2 信息

ASAP2 文件生成过程需要关于模型中参数和信号的信息。其中一些信息包含在模型本身中。使用工作区或数据字典中的 Simulink 数据对象来完成 ASAP2 文件生成过程。在某些情况下，工作区对象的使用是可选的。

使用模型数据编辑器和内置的 Simulink 数据对象来提供信息，例如使用 **Simulink.Signal** 对象来提供 MEASUREMENT（测量）信息，使用 **Simulink.Parameter** 对象来提供 CHARACTERISTIC（特征）信息。您可以使用从 **Simulink.Signal** 和 **Simulink.Parameter** 派生的数据类中的数据对象来提供信

息。请参阅 “数据对象” 和**模型数据编辑器**。

下表包含生成 ASAP2 文件时必需的数据属性。一些数据属性在模型中定义。其他属性在对象的属性中提供。对于在 **Simulink.Signal** 或 **Simulink.Parameter** 对象中定义的属性，下表列出了相关联的属性名称。

|  |  |  |
| --- | --- | --- |
| **数据属性** | **定义位置** | **属性名称** |
| 名称（符号） | 模型和数据对象 | 从作为参数或信号名称解析目标的数据对象的句柄继承 |
| 描述 | 数据对象 | **Description** |
| 数据类型 | 模型或数据对象 | **DataType** |
| 定标  （如果是定点数据类型） | 模型或数据对象 | **DataType** |
| 允许的最小值 | 模型或数据对象 | **Min** |
| 允许的最大值 | 模型或数据对象 | **Max** |
| 单位 | 模型或数据对象 | **Unit** |
| 内存地址（可选） | 模型或数据对象 | **MemoryAddress\_ASAP2**（可[选；请参阅“内存地址属性” （第](#_bookmark371) [35-3](#_bookmark371) 页）。） |

#### 内存地址属性

如果在代码生成之前内存地址属性未知，代码生成器将在生成的 ASAP2 文件中插入 **ECU Address** 占位符文本。您可以在后期对生成的文件进行处理，用实际地址代替占位符。有关示例，请参阅文件 **matlabroot/toolbox/rtw/targets/asap2/asap2/asap2post.m**。**asap2post.m** 解析您提供的链接器映射文件，并用实际内存地址替换 ASAP2 文件中的 **ECU Address** 占位符。由于链接器映射文件因编译器而异，您可能需要修改 **asap2post.m** 中的正则表达式代码，以匹配您使用的链接器映射的格式。

如果您的系统上安装了 Embedded Coder，并且您正在为嵌入式目标生成可执行与可链接格式 (ELF) 或程序数据库 (PDB) 文件，则可以使用 **rtw.asap2SetAddress** 函数自动执行 ECU 地址替换。有关详细信 息，请参阅 [“ASAP2 文件的自动 ECU 地址替换 (Embedded Coder)” （第 35-4](#_bookmark372) 页）。

如果在代码生成之前内存地址属性是已知的，可以在数据项或数据对象中定义它。默认情况下， **Simulink.Signal** 或 **Simulink.Parameter** 数据对象类中不存在 **MemoryAddress\_ASAP2** 属性。如果要添加属性，请将名为 **MemoryAddress\_ASAP2** 的属性添加到自定义类中，该类是 **Simulink** 或 **ASAP2** 类的子类。有关子类化 Simulink 数据类的信息，请参阅“定义数据类”。

下表显示的 Simulink 对象属性取代了以前版本的 ASAP2 对象属性。

#### ASAP2 和 Simulink 参数和信号对象属性之间的差异

|  |  |
| --- | --- |
| **ASAP2 对象属性（以前）** | **Simulink 对象属性（当前）** |
| LONGID\_ASAP2 | 描述 |
| PhysicalMin\_ASAP2 | Min |
| PhysicalMax\_ASAP2 | Max |
| Units\_ASAP2 | 单位 |

**ASAP2 文件的自动 ECU 地址替换 (Embedded Coder)**

如果您的系统中已安装 Embedded Coder，并且您要为嵌入式目标生成可执行与可链接格式 (ELF) 或程序数据库 (PDB) 文件，则可以使用 **rtw.asap2SetAddress** 函数自动地在生成的 ASAP2 文件中用实际地址替换 **ECU Address** 占位符内存地址值。

如果在代码生成之前内存地址属性未知，代码生成器将在生成的 ASAP2 文件中插入 **ECU Address** 占位符文本，如下所示：

**/begin CHARACTERISTIC**

**/\* Name \*/ Ki**

**/\* Long Identifier \*/ ""**

**/\* Type \*/ VALUE**

**/\* ECU Address \*/ 0x0000 /\* @ECU\_Address@Ki@ \*/**

要用实际地址代替 **ECU Address** 占位符，请使用 **rtw.asap2SetAddress** 函数处理生成的 ASAP2 文件。语法是：

**rtw.asap2SetAddress(ASAP2File,InfoFile)**

参数是字符向量，用于指定生成的 ASAP2 文件的名称和生成的可执行 ELF 文件、来自 Microsoft 工具链的 PDB 文件或模型的 DWARF 调试信息文件的名称。调用时，**rtw.asap2SetAddress** 从指定的 ELF、 PDB 或 DWARF 文件中提取实际 ECU 地址，并用实际地址替换 ASAP2 文件中的占位符，例如：

**/begin CHARACTERISTIC**

**/\* Name \*/ Ki**

**/\* Long Identifier \*/ ""**

**/\* Type \*/ VALUE**

**/\* ECU Address \*/ 0x40009E60**

#### 为查找表定义 ASAP2 信息

Simulink Coder 软件为查找表数据及其断点生成 ASAP2 描述信息。软件将一维表数据表示为 **CURVE**信息，将二维表数据表示为 **MAP** 信息，将断点表示为 **AXIS\_DESCR** 和 **AXIS\_PTS** 信息。您可以使用以下 Simulink Lookup Table 模块之一来建立查找表模型：

* Direct Lookup Table (n-D) - 一维和二维
* Interpolation Using Prelookup - 一维和二维
* 1–D Lookup Table
* 2–D Lookup Table
* n-D Lookup Table - 一维和二维

软件支持以下类型的查找表断点（轴点）。

|  |  |
| --- | --- |
| **断点类型** | **生成的信息** |
| 可调整并在多个表轴中共享（公共轴） | **COM\_AXIS** |
| 固定和不可调（固定轴） | **FIX\_AXIS** 的以下形式之一：   * **FIX\_AXIS\_PAR**（如果断点是具有等距间隔的整数并且等距间隔是 2 的幂） * **FIX\_AXIS\_PAR\_DIST**（如果断点是具有等距间隔的整数） * **FIX\_AXIS\_PAR\_LIST**（如果断点是具有非等距间距的整数） |
| 可调但不在多个表中共享（标准轴） | **STD\_AXIS** |

当您为 ASAP2 代码生成配置模块时：

* 对于表数据，请使用具有非 **Auto** 存储类的 **Simulink.Parameter** 数据对象。
* 对于在多个表轴中共享的可调断点数据 (**COM\_AXIS**)，请使用配置了非 **Auto** 存储类的

#### Simulink.Parameter 数据对象。

* 对于固定的、不可调的断点数据 (**FIX\_AXIS**)，请使用模块参数对话框中指定的工作区变量或数组。断点必须作为整数存储在代码中。数据类型必须为内置整数类型（**int8**、**int16**、**int32**、**uint8**、 **uint16** 或 **uint32**）、定点数据类型或等效别名类型。
* 对于不在多个表轴中共享的可调断点数据 (STD\_AXIS)：

1. 创建 **Simulink.Bus** 对象来定义 **struct** 打包（字段的名称和顺序）。参数结构体的字段必须对应于查找表数据和查找表模块的每个轴。例如，在二维的 n-D Lookup Table 模块中，结构体必须仅包含三个字段。此总线对象说明查找特性的记录布局。
2. 创建一个表示可调参数的 **Simulink.Parameter** 对象。
3. 创建表和轴值。
4. 您也可以为参数对象指定**单位**、**最小值**和**最大值**属性。

以下是 n-D Lookup Table 记录生成为标准轴格式的 ASAP2 文件的示例：

**/begin CHARACTERISTIC**

**/\* Name \*/ STDAxisParam**

**...**

**/\* Record Layout \*/ Lookup1D\_X\_WORD\_Y\_FLOAT32\_IEEE**

**...**

**begin AXIS\_DESCR**

**/\* Description of X-Axis Points \*/**

**/\* Axis Type \*/ STD\_AXIS**

**...**

**/end AXIS\_DESCR**

**/end CHARACTERISTIC**

**/begin RECORD\_LAYOUT Lookup1D\_X\_WORD\_Y\_FLOAT32\_IEEE AXIS\_PTS\_X 1 WORD INDEX\_INCR DIRECT**

**FNC\_VALUES 2 FLOAT32\_IEEE COLUMN\_DIR DIRECT**

**/end RECORD\_LAYOUT**

### 生成 ASAP2 文件

* [“生成 ASAP2 文件” （第 35-6 页）](#_bookmark375)
* [“使用基于 GRT 或基于 ERT 的系统目标文件” （第 35-6 页）](#_bookmark376)
* [“为引用模型生成 ASAP2 文件” （第 35-6 页）](#_bookmark377)
* [“合并顶层模型和引用模型的 ASAP2 文件” （第 35-7 页）](#_bookmark378)

#### 生成 ASAP2 文件

在代码生成和编译过程中，使用一般实时系统目标文件或 Embedded Coder 系统目标文件来生成 ASAP2文件。

使用具有内置 ASAP2 支持的系统目标文件生成 ASAP2 文件。

#### 使用基于 GRT 或基于 ERT 的系统目标文件

要使用一般实时 (GRT) 系统目标文件或 Embedded Coder 系统目标文件为模型生成 ASAP2 数据定义，请执行下列步骤：

1. 使用**代码映射编辑器 - C** 将存储类应用于信号、模块状态和模型参数，如“模型接口元素的 [C 代码生成配置” （第 17-2](#_bookmark135) 页）中所述。

如果存在以下一个或多个条件，则 ASAP2 文件中不会表示参数：

* + 您应用了存储类“自动”、“文件作用域”、“结构体”或“默认”（通过“模型默认”）。
  + 您通过 “**Embedded Coder** 字典”应用了存储类。
  + 您应用了一个存储类，而该类会导致代码生成器生成宏或不可寻址的变量。

1. 使用属性检查器为每个数据元素配置其余的存储类属性。
2. 将模型配置参数**系统目标文件**设置为基于 GRT 或基于 ERT 的系统目标文件。
3. 使用以下命令启用配置参数 **ASAP2 接口**

**set\_param('model\_Name','GenerateASAP2',true);**

**注意** 您无法再从“模型配置参数”对话框中启用 ASAP2 文件的生成。如 [“生成 ASAP2 和 CDF 标定文件” （第 36-2 页）](#_bookmark381) 所述，请使用 **Generate Calibration Files** 工具，或使用命令行模型

配置参数 **GenerateASAP2**。

1. 选择参数**仅生成代码**。
2. 点击**应用**。
3. 生成代码。

代码生成器将 ASAP2 文件写入编译文件夹。默认情况下，文件命名为 **model.a2l**，其中 **model** 是模型的名称。ASAP2 设置文件控制 ASAP2 文件名。有关详细信息，请参阅“Customize Generated ASAP2 File”。

#### 为引用模型生成 ASAP2 文件

编译过程可以为模型引用层次结构中的每个引用模型生成一个 ASAP2 文件。在生成的 ASAP2 文件中， MEASUREMENT 记录表示引用模型内的信号和状态。

要为引用模型生成 ASAP2 文件，请为顶层模型和引用层次结构中的每个引用模型选择 ASAP2 文件生成。如果您正在使用一般实时目标或 Embedded Coder [目标，请对顶层模型和每个引用模型按照“使用基于](#_bookmark376) [GRT 或基于 ERT 的系统目标文件” （第 35-6](#_bookmark376) 页）中所述的过程操作。

#### 合并顶层模型和引用模型的 ASAP2 文件

使用函数 **rtw.asap2MergeMdlRefs** 合并为顶层模型和引用模型生成的 ASAP2 文件。此函数具有以下语法：

**[status,info] = rtw.asap2MergeMdlRefs(topModelName,asap2FileName)**

* **topModelName** 是包含一个或多个引用模型的模型的名称。
* **asap2FileName** 是您为合并的 ASAP2 文件指定的名称。
* 可选：如果合并完成，**status** 返回 false（逻辑值 0），否则返回 true（逻辑值 1）。
* 可选：如果 **status** 为 true，则 **info** 返回有关合并失败的其他信息。否则，它将返回空字符向量。

请参考以下示例。

**[status,info] = rtw.asap2MergeMdlRefs('myTopMdl','merged.a2l')**

该命令将把为顶层模型 **myTopMdl** 及其引用的模型生成的 ASAP2 文件合并到文件 **merged.a2l** 中。

### ASAP2 文件的结构

下表概述了 ASAP2 文件的基本结构，并说明用于创建该文件的每个部分的目标语言编译器 (TLC) 函数和文件：

* ASAP2 文件的静态部分以**粗体**显示。
* 函数调用由 **%<FunctionName()>** 指示。

|  |  |  |
| --- | --- | --- |
| **文件部分** | **asap2main.tlc 的内容** | **包含函数定义的 TLC 文件** |
| 文件头 | **%<ASAP2UserFcnWriteFileHead()>** | **asap2userlib.tlc** |
| **/begin PROJECT ""** | **/begin PROJECT "%<ASAP2ProjectName>"** | **asap2setup.tlc** |
| **/begin HEADER "" HEADER** 内容 | **/begin HEADER"%<ASAP2HeaderName>"**  **%<ASAP2UserFcnWriteHeader()>** | **asap2setup.tlc asap2userlib.tlc** |
| **/end HEADER** | **/end HEADER** | **asap2userlib.tlc** |
| **/begin MODULE "" MODULE** 内容： | **/begin MODULE "%<ASAP2ModuleName>"}** | **asap2setup.tlc asap2userlib.tlc** |
| * **A2ML** * **MOD\_PAR** * **MOD\_COMMON**   ... | **%<ASAP2UserFcnWriteHardwareInterface()>** | **asap2userlib.tlc** |
| 依赖模型的 **MODULE** 内容： | **%<SLibASAP2WriteDynamicContents()>**  调用用户定义的函数： | **asap2lib.tlc** |
| * **RECORD\_LAYOUT** * **CHARACTERISTIC** * **ParameterGroups** * **ModelParameters** | **...WriteRecordLayout\_TemplateName()**  **...WriteCharacteristic\_TemplateName()**  **...WriteCharacteristic\_Scalar()** | **user/templates/...** |

|  |  |  |
| --- | --- | --- |
| **文件部分** | **asap2main.tlc 的内容** | **包含函数定义的 TLC 文件** |
| * **MEASUREMENT** * **ExternalInputs** * **BlockOutputs** | **...WriteMeasurement()** | **asap2userlib.tlc** |
| **- COMPU\_METHOD** | **...WriteCompuMethod()** | **asap2userlib.tlc** |
| **/end MODULE** | **/end MODULE** |  |
| 文件尾部 | **%<ASAP2UserFcnWriteFileTail()>** | **asap2userlib.tlc** |

### 另请参阅

**相关示例**

* [“生成 ASAP2 和 CDF 标定文件” （第 36-2 页）](#_bookmark381)
* [“在生成的代码中创建可调标定参数” （第 16-28 页）](#_bookmark123)
* [“生成的代码如何存储内部信号、状态和参数数据” （第 16-2 页）](#_bookmark109)

# 生成 ASAP2 和 CDF 标定文件

## 生成 ASAP2 和 CDF 标定文件

使用**生成标定文件**工具为模型生成和自定义一个 ASAP2 文件和一个 CDF（标定数据格式）文件。默认情况下，该工具根据 ASAM MCD-2 MC 标准生成 A2L 文件的最新支持版本。您可以选择生成不同的版本。您可以自定义标定文件的选项，例如 A2L 文件的版本和 CDF 文件的架构类型。有关 ASAM MCD-2 MC (ASAP2) 标准和 ASAM CDF 标准的详细信息，请参阅 ASAM 网站 [**www.asam.net**](https://www.asam.net/)。

### 打开生成标定文件工具

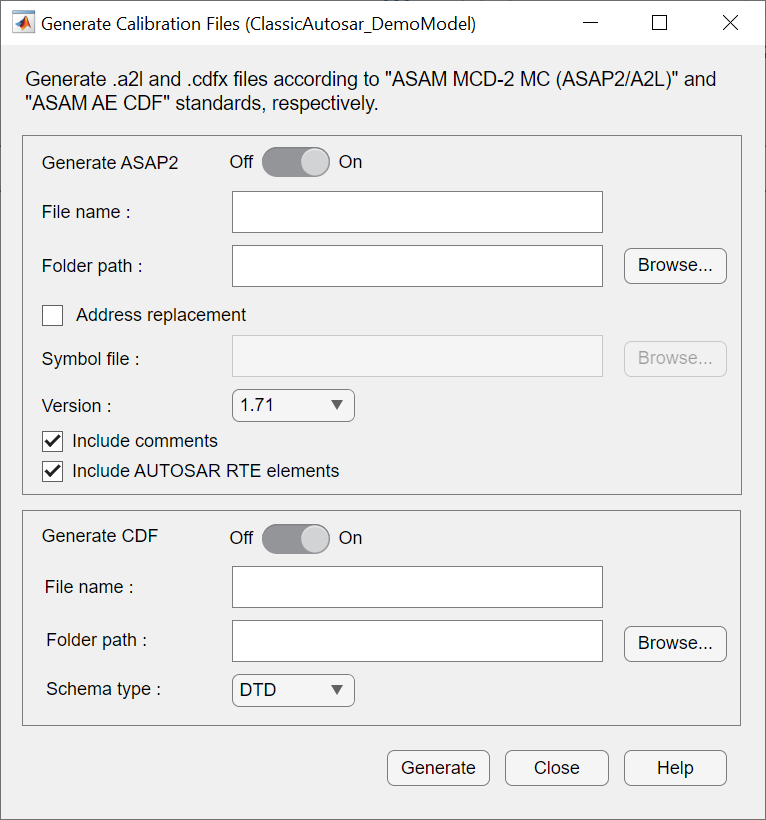
1. 打开模型。
2. 打开 Embedded Coder。
3. 为模型生成代码。
4. 在 **C 代码**选项卡上，选择**共享** > **生成标定文件**。

**注意** 对于 AUTOSAR 模型，选择 **Autosar** > **共享** > **生成标定文件**。

使用该工具，您可以同时为一个模型生成 A2L 和 **cdfx** 文件。您可以将该工具用于使用以下系统目标文件配置的模型：基于 ERT、基于 GRT、Simulink Real-Time、经典 AUTOSAR 或自适应 AUTOSAR 系统目标文件。该工具会记住上次使用的设置，如 ASAP2 文件的版本，包含或排除注释，关闭或打开 ASAP2 文件和 CDF 文件生成。Simulink 将这些设置保存在 MATLAB 预设项中。这些设置将覆盖默认设置。

对于自适应 AUTOSAR 和 Simulink Real-Time 系统目标文件，可以生成 C++ 语言的 ASAP2 文件。对于基于 GRT 和 ERT 的系统目标文件，只能生成 C 语言的 ASAP2 文件。

对于经典 AUTOSAR 模型，该工具允许您使用 **Include AUTOSAR RTE elements** 选项在 ASAP2 文件中包含或排除 RTE 元素。



### ASAP2 文件生成

ASAP2 定义当应用程序执行时用于调整参数和测量信号数据的内部 ECU 变量的描述格式。要生成 ASAP2文件，请执行以下操作：

1. 默认情况下，**生成 ASAP2** 选项处于启用状态。如果未启用，请将**生成 ASAP2** 滑动到打开位置以启用 A2L 生成。
2. 要更改 A2L 文件的名称，请在**文件名**字段中指定名称。
3. 默认情况下，代码生成器使用编译文件夹作为所选模型的文件夹路径。如果您要为 A2L 文件指定不同的文件夹路径，可以在**文件夹路径**字段中指定。
4. 要替换 A2L 文件中的 ECU 地址，请在**符号文件**字段中提供与应用程序可执行文件相关联的符号文件。否则，请取消选中**地址替换**字段。

该工具支持 ELF、PDB、DWARF 或可执行文件作为替换 ECU 地址的符号文件。

1. 将**版本**设置为要生成的 A2L 文件的所需版本。默认版本是 1.71。
2. 要排除 A2L 文件中的注释，请取消选中**包括注释**。
3. 对于经典 AUTOSAR 模型，要排除 A2L 文件中的 RTE 元素，请清除**包括 AUTOSAR RTE 元素**。
4. 点击**生成**。

要以编程方式生成 A2L 文件，请使用 **coder.asap2.export** 函数。

### CDF 文件生成

ASAM CDF 定义用于存储 ECU 参数值和相关联的元数据的数据描述格式。ASAM CDF 文件使用 XML 格式，您可以通过标定工具和 XML 编辑器轻松地验证、编辑、导入和导出 XML 格式的文件。要使用该工具生成 CDF 文件，请执行以下操作：

1. 将**生成 CDF** 滑块滑动到打开位置。
2. 要更改 CDF 文件的名称，请在**文件名**字段中指定更改后的名称。
3. 默认情况下，代码生成器选择编译文件夹作为所选模型的文件夹路径。如果要为 CDF 文件指定不同文件夹路径，请在**文件夹路径**字段中指定路径。
4. 使用**架构类型**字段设置为 **XSD**（XML 架构定义）或 **DTD**（文档类型定义）。**DTD** 是默认架构类型。

要以编程方式生成 CDF 文件，请使用 **coder.cdf.export** 函数。

**注意** CDF 文件生成对 Simulink Real-Time 系统目标文件不可用。

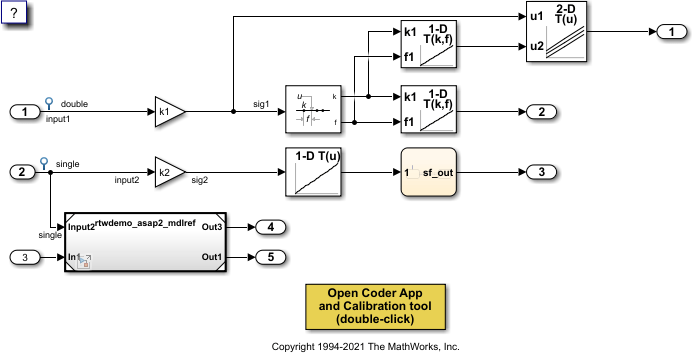
### 为数据测量和标定创建基于主机的 ASAM-ASAP2 数据定义文件

此模型显示 ASAP2 数据导出。ASAP2 是由自动化和测量系统标准化协会 (ASAM) 提出的数据定义标准。

#### 打开示例模型

打开示例模型 **rtwdemo\_asap2**。

**open\_system('rtwdemo\_asap2');**



请参阅 [“ASAP2 文件生成” （第 36-3 页）](#_bookmark383) 以自定义和生成 ASAP2 文件。

### 另请参阅

**coder.cdf.export** | **coder.asap2.export**

### 详细信息

* “External Mode Simulation by Using XCP Communication”
* “Customize XCP Server Software”
* “Configure AUTOSAR Adaptive Data for Run-Time Calibration and Measurement” (AUTOSAR Blockset)

**对 Simulink Coder 生成的代码进行直接内存访问**

**自定义生成的 ASAP2 文件**

**为 ASAP2 文件生成配置模型数据元素**

**性能**

**在 Simulink Coder 中优化生成的代码**

* [“生成代码的执行探查” （第 40-2 页）](#_bookmark391)
* [“控制 MATLAB Function 模块中可变大小数组的内存分配” （第 40-4 页）](#_bookmark392)
* [“优化时间计数器的内存使用量” （第 40-6 页）](#_bookmark396)

## 生成代码的执行探查

利用代码执行探查可以：

* 确定生成的代码是否满足在目标硬件上实时部署的执行时间要求。
* 确定需要提高执行速度的代码节。

以下任务代表使用代码执行探查的一般工作流：

1. 使用 Simulink 模型，设计并优化您的算法。
2. 为代码执行探查配置模型，并生成代码。
3. 在目标硬件上执行生成的代码。例如，您可以：
   * 在开发计算机上运行软件在环 (SIL) 仿真。
   * 使用目标支持包或自定义 PIL 目标运行处理器在环 (PIL) 仿真。
   * 在开发计算机或自定义硬件上运行基于 XCP 的外部模式仿真。
   * 使用 Simulink Real-Time 或目标支持包实时执行。
4. 通过代码执行探查图和报告分析执行速度。例如，检查算法代码是否满足实时部署的执行时间要求：
   * 如果算法代码很容易就能满足要求，请考虑增强您的算法以利用可用的处理能力。
   * 如果代码无法实时执行，请设法减少执行时间。

确定耗时最长的任务。对于这些任务，研究是否可以在功能和速度之间进行权衡。如果您的目标是多核处理器，请将算法代码的执行分布在可用的多个核上。

1. 如果需要，请细化模型并返回到步骤 2。

要查找有关 Simulink 产品的代码执行探查的信息，请使用下表。

|  |  |  |  |
| --- | --- | --- | --- |
| **执行类型** | **目标** | **相关产品** | **参阅** |
| 为并发执行配置的模型 | 开发计算机 | Simulink Coder | * “Optimize and Deploy on a Multicore Target” |
| 软件在环 (SIL) | 开发计算机 | Embedded Coder | * “Create Execution-Time Profile for Generated Code” (Embedded Coder) * “View and Compare Code Execution Times” (Embedded Coder) * “Analyze Code Execution Data” (Embedded Coder) |

|  |  |  |  |
| --- | --- | --- | --- |
| **执行类型** | **目标** | **相关产品** | **参阅** |
| 处理器在环 (PIL) | 嵌入式硬件或指令集仿真器 | Embedded Coder | * “Create Execution-Time Profile for Generated Code” (Embedded Coder) * “View and Compare Code Execution Times” (Embedded Coder) * “Analyze Code Execution Data” (Embedded Coder) |
| XCP 外部模式： | 开发计算机或自定义硬件 | Embedded Coder | * “Create Execution-Time Profile for Generated Code” (Embedded Coder) * “Analyze Code Execution Data” (Embedded Coder) |
| 实时执行 | Simulink Real-Time | Simulink Coder, Simulink Real-Time | * 实时应用程序的执行探查 (Simulink Real-Time) |

### 另请参阅

**相关示例**

* “Design Techniques to Optimize Models for Efficient Code Generation”
* “SIL 和 PIL 仿真” (Embedded Coder)

## 控制 MATLAB Function 模块中可变大小数组的内存分配

动态内存分配在运行时根据需要在堆上分配内存，而不是在堆栈上静态分配内存。您可以对 MATLAB Function 模块内的数组使用动态内存分配。

您不能将动态内存分配用于以下各项：

* 输入和输出信号。可变大小的输入和输出信号必须具有上界。
* 参数或全局变量。参数和全局变量必须为固定大小。
* 总线数组的字段。总线数组不能具有可变大小字段。
* 与 MATLAB System 模块关联的 System object 的离散状态属性。

在以下情况下，动态内存分配是有益的：

* 您不知道数组的上界。
* 您不想在堆栈上为大型数组分配内存。

动态内存分配和释放此内存会导致所生成代码的执行速度变慢。要控制 MATLAB Function 模块中可变大小数组的动态内存分配的使用，您可以：

* [为可变大小数组提供上界 （第 40-4](#_bookmark393) 页）。
* [禁用 MATLAB Function 模块的动态内存分配 （第 40-4](#_bookmark394) 页）。
* [修改动态内存分配阈值 （第 40-4](#_bookmark395) 页）。

### 为可变大小数组提供上界

对于无界的可变大小数组，代码生成器在堆上动态分配内存。对于有界的可变大小数组，如果大小（以字节为单位）小于动态内存分配阈值，代码生成器将在堆栈上静态分配内存。要避免动态内存分配，请提供数组维度的上界，以使数组的大小（以字节为单位）小于动态内存分配阈值。请参阅“Specify Upper Bounds for Variable-Size Arrays”。

### 为 MATLAB Function 模块禁用动态内存分配

默认情况下，对基于 GRT 的目标启用 MATLAB Function 模块的动态内存分配，而对基于 ERT 的目标禁用。要更改设置，请在“配置参数”对话框中，清除或选中**在 MATLAB 函数中使用动态内存分配**。

如果禁用动态内存分配，则必须为可变大小数组提供上界。

### 修改动态内存分配阈值

您可以使用动态内存分配阈值来指定代码生成器何时使用动态内存分配，而不是为所有可变大小数组禁用动态内存分配。

使用动态内存分配阈值可以：

* 对较小的数组禁用动态内存分配。对于较小的数组，静态内存分配可以加快所生成代码的执行速度。但是，静态内存分配会导致未使用的存储空间。对于较小的数组，未使用的存储空间可能不是重要考虑因素，具体由您决定。
* 对较大的数组启用动态内存分配。对于较大的数组，使用动态内存分配可以显著降低存储要求。

动态内存分配阈值的默认值为 64 KB。要更改阈值，请在“配置参数”对话框中设置 **MATLAB 函数中的动态内存分配阈值**参数。

要对所有可变大小数组使用动态内存分配，请将阈值设置为 **0**。

### 另请参阅

MATLAB Function

### 相关示例

* “可变大小数组的代码生成”
* “Specify Upper Bounds for Variable-Size Arrays”
* “Use Dynamic Memory Allocation for Variable-Size Arrays in a MATLAB Function Block”

## 优化时间计数器的内存使用量

此示例说明如何优化代码生成器为时间计数器分配的内存量。该示例优化存储经过时间（两个事件之间的时间间隔）的内存。

代码生成器将时间计数器表示为无符号整数。时间计数器的字长基于模型配置参数**应用程序生命周期(天)**的设置，该参数指定应用程序运行的预期最长持续时间。您可以使用此参数来防止时间计数器溢出。默认大小为 64 位。

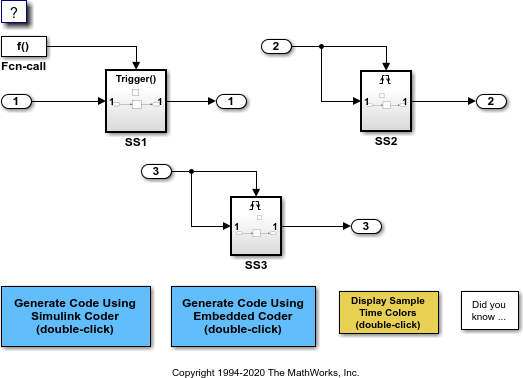
时间计数器使用的位数取决于**应用程序生命周期(天)** 参数的设置。例如，如果时间计数器以 1 kHz 的速率递增，为了避免溢出，计数器的位数如下：

* 生命周期 < 0.25 秒：8 位
* 生命周期 < 1 分钟：16 位
* 生命周期 < 49 天：32 位
* 生命周期 > 50 天：64 位

64 位时间计数器在寿命超过 5.9 亿年后才会溢出。

#### 打开示例模型

打开示例模型 **rtwdemo\_abstime**。



该模型由三个子系统 **SS1**、**SS2** 和 **SS3** 组成。打开“模型配置参数”对话框。在**数学和数据类型**窗格中，**应用程序生命周期(天)** 参数的设置为默认值，即 **inf**。

这三个子系统包含一个离散时间积分器，它需要经过的时间作为输入来计算其输出值。子系统的变化如下：

* SS1 - 时钟频率为 1 kHz。不需要时间计数器。触发端口的**采样时间类型**参数设置为 **periodic**。历时为 0.001。
* SS2 - 时钟频率为 100 Hz。需要时间计数器。基于 1 天的生命周期，32 位计数器存储经过的时间。
* SS3 - 时钟频率为 0.5 Hz。需要时间计数器。基于 1 天的生命周期，16 位计数器存储经过的时间。

#### 对模型进行仿真

对模型进行仿真。默认情况下，模型配置为以不同颜色显示采样时间。三个子系统的离散采样时间显示为红色、绿色和蓝色。触发子系统是蓝绿色。

#### 生成代码和报告

1.为编译和检查过程创建一个临时文件夹。

2.将代码生成器的模型配置为使用 GRT 系统目标文件，生命周期为 **inf** 天。

3.编译模型。

**### Starting build procedure for: rtwdemo\_abstime**

**### Successful completion of build procedure for: rtwdemo\_abstime Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**================================================================================================**

**rtwdemo\_abstime Code generated and compiled. Code generation information file does not exist.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 12.1s**

#### 查看生成的代码

打开生成的源文件 **rtwdemo\_abstime.h**。

**struct tag\_RTM\_rtwdemo\_abstime\_T { const char\_T \*errorStatus;**

**/\***

* **Timing:**
* **The following substructure contains information regarding**
* **the timing information for the model.**

**\*/ struct {**

**uint32\_T clockTick1; uint32\_T clockTickH1; uint32\_T clockTick2; uint32\_T clockTickH2; struct {**

**uint16\_T TID[3]; uint16\_T cLimit[3];**

**} TaskCounters;**

**} Timing;**

**};**

**/\* Block states (default storage) \*/**

**extern DW\_rtwdemo\_abstime\_T rtwdemo\_abstime\_DW;**

**/\* Zero-crossing (trigger) state \*/**

**extern PrevZCX\_rtwdemo\_abstime\_T rtwdemo\_abstime\_PrevZCX;**

**/\* External inputs (root inport signals with default storage) \*/ extern ExtU\_rtwdemo\_abstime\_T rtwdemo\_abstime\_U;**

**/\* External outputs (root outports fed by signals with default storage) \*/ extern ExtY\_rtwdemo\_abstime\_T rtwdemo\_abstime\_Y;**

**/\* Model entry point functions \*/**

**extern void rtwdemo\_abstime\_initialize(void); extern void rtwdemo\_abstime\_step0(void); extern void rtwdemo\_abstime\_step1(void); extern void rtwdemo\_abstime\_step2(void); extern void rtwdemo\_abstime\_terminate(void);**

**/\* Real-time Model object \*/**

**extern RT\_MODEL\_rtwdemo\_abstime\_T \*const rtwdemo\_abstime\_M;**

**/\*-**

* **The generated code includes comments that allow you to trace directly**
* **back to the appropriate location in the model. The basic format**
* **is <system>/block\_name, where system is the system number (uniquely**
* **assigned by Simulink) and block\_name is the name of the block.**
* **Use the MATLAB hilite\_system command to trace the generated code back**
* **to the model. For example,**
* **hilite\_system('<S3>') - opens system 3**
* **hilite\_system('<S3>/Kp') - opens and selects block Kp which resides in S3**
* **Here is the system hierarchy for this model**
* **'<Root>' : 'rtwdemo\_abstime'**
* **'<S1>' : 'rtwdemo\_abstime/SS1'**
* **'<S2>' : 'rtwdemo\_abstime/SS2'**
* **'<S3>' : 'rtwdemo\_abstime/SS3'**

**\*/**

**#endif /\* RTW\_HEADER\_rtwdemo\_abstime\_h\_ \*/**

四个 32 位无符号整数 **clockTick1**、**clockTickH1**、**clockTick2** 和 **clockTickH2** 是用于存储子系统

**SS2** 和 **SS3** 的经过的时间的计数器。

#### 启用优化和重新生成代码

1. 重新配置模型以将生命周期设置为 1 天。

2.编译模型。

**### Starting build procedure for: rtwdemo\_abstime**

**### Successful completion of build procedure for: rtwdemo\_abstime**

**Build Summary**

**Top model targets built:**

**Model Action Rebuild Reason**

**==============================================================================**

**rtwdemo\_abstime Code generated and compiled. Incremental checksum changed.**

**1 of 1 models built (0 models already up to date)**

**Build duration: 0h 0m 8.0106s**

#### 查看重新生成的代码。

**struct tag\_RTM\_rtwdemo\_abstime\_T { const char\_T \*errorStatus;**

**/\***

* **Timing:**
* **The following substructure contains information regarding**
* **the timing information for the model.**

**\*/ struct {**

**uint32\_T clockTick1; uint16\_T clockTick2; struct {**

**uint16\_T TID[3]; uint16\_T cLimit[3];**

**} TaskCounters;**

**} Timing;**

**};**

**/\* Block states (default storage) \*/**

**extern DW\_rtwdemo\_abstime\_T rtwdemo\_abstime\_DW;**

**/\* Zero-crossing (trigger) state \*/**

**extern PrevZCX\_rtwdemo\_abstime\_T rtwdemo\_abstime\_PrevZCX;**

**/\* External inputs (root inport signals with default storage) \*/ extern ExtU\_rtwdemo\_abstime\_T rtwdemo\_abstime\_U;**

**/\* External outputs (root outports fed by signals with default storage) \*/ extern ExtY\_rtwdemo\_abstime\_T rtwdemo\_abstime\_Y;**

**/\* Model entry point functions \*/**

**extern void rtwdemo\_abstime\_initialize(void); extern void rtwdemo\_abstime\_step0(void); extern void rtwdemo\_abstime\_step1(void); extern void rtwdemo\_abstime\_step2(void); extern void rtwdemo\_abstime\_terminate(void);**

**/\* Real-time Model object \*/**

**extern RT\_MODEL\_rtwdemo\_abstime\_T \*const rtwdemo\_abstime\_M;**

**/\*-**

* + **The generated code includes comments that allow you to trace directly**
  + **back to the appropriate location in the model. The basic format**
  + **is <system>/block\_name, where system is the system number (uniquely**
  + **assigned by Simulink) and block\_name is the name of the block.**
  + **Use the MATLAB hilite\_system command to trace the generated code back**
  + **to the model. For example,**
  + **hilite\_system('<S3>') - opens system 3**
  + **hilite\_system('<S3>/Kp') - opens and selects block Kp which resides in S3**
  + **Here is the system hierarchy for this model**
  + **'<Root>' : 'rtwdemo\_abstime'**
  + **'<S1>' : 'rtwdemo\_abstime/SS1'**
  + **'<S2>' : 'rtwdemo\_abstime/SS2'**
  + **'<S3>' : 'rtwdemo\_abstime/SS3'**

**\*/**

**#endif /\* RTW\_HEADER\_rtwdemo\_abstime\_h\_ \*/**

**应用程序生命周期(天)** 参数的新设置指示代码生成器为时间计数器留出更少的内存。重新生成的代码包括：

* 32 位无符号整数 **clockTick1**，用于存储 **SS2** 的任务的经过的时间
* 16 位无符号整数 **clockTick2**，用于存储 **SS3** 的任务的经过的时间

#### 相关信息

* “模型配置参数：代码生成优化”
* “Timers in Asynchronous Tasks”
* [“基于时间的调度和代码生成” （第 4-2 页）](#_bookmark10)

### 另请参阅详细信息

* “Design Techniques to Optimize Models for Efficient Code Generation”
* “Control Memory Allocation for Time Counters”
* “Access Timers Programmatically”
* “Generate Code for an Elapsed Time Counter”
* “Absolute Time Limitations”

**Simulink Coder 中的高吞吐量应用**

# 验证

# Simulink Coder 中的仿真和代码比较

# 模型和生成的代码之间的数值一致性

# 自定义

# Simulink Coder 中的编译过程集成

## 使用 STF\_make\_rtw\_hook 文件自定义编译过程

在编译过程中，您可以在代码生成和联编过程中的指定点执行的钩子方法中提供可选的自定义代码。您可以使用钩子方法将特定于目标的操作添加到编译过程中。

### STF\_make\_rtw\_hook 文件

您可以在通常称为 **STF\_make\_rtw\_hook.m** 的文件中修改钩子方法，其中 **STF** 是系统目标文件的名称，例如 **ert** 或 **mytarget**。此文件实现一个 **STF\_make\_rtw\_hook** 函数，它根据传入的 **hookMethod** 参数调度给特定的操作。

编译过程通过调用 **STF\_make\_rtw\_hook** 来传入 **hookMethod** 参数和其他参数。您只需实现编译过程需要的那些钩子方法。

如果您的模型包含引用模型，则可以根据需要为每个引用模型实现一个 **STF\_make\_rtw\_hook.m**。编译过程逐个调用引用模型的 **STF\_make\_rtw\_hook**，以递归方式（按依存顺序）处理这些文件。

### STF\_make\_rtw\_hook 文件的使用约定

要使编译过程调用 **STF\_make\_rtw\_hook**，请检查是否满足以下条件：

* **STF\_make\_rtw\_hook.m** 文件位于 MATLAB 路径中。
* 文件名是系统目标文件 (**STF**) 的名称，后跟文本 **\_make\_rtw\_hook.m**。例如，如果您使用自定义系统目标文件 **mytarget.tlc** 生成代码，请将钩子文件命名为 **mytarget\_make\_rtw\_hook.m**，并将文件中实现的钩子函数命名为 **mytarget\_make\_rtw\_hook**。
* 文件中实现的钩子函数使用“STF\_mak[e\_rtw\_hook.m 函数原型和参数” （第 44-2](#_bookmark406) 页）中介绍的函数原型。

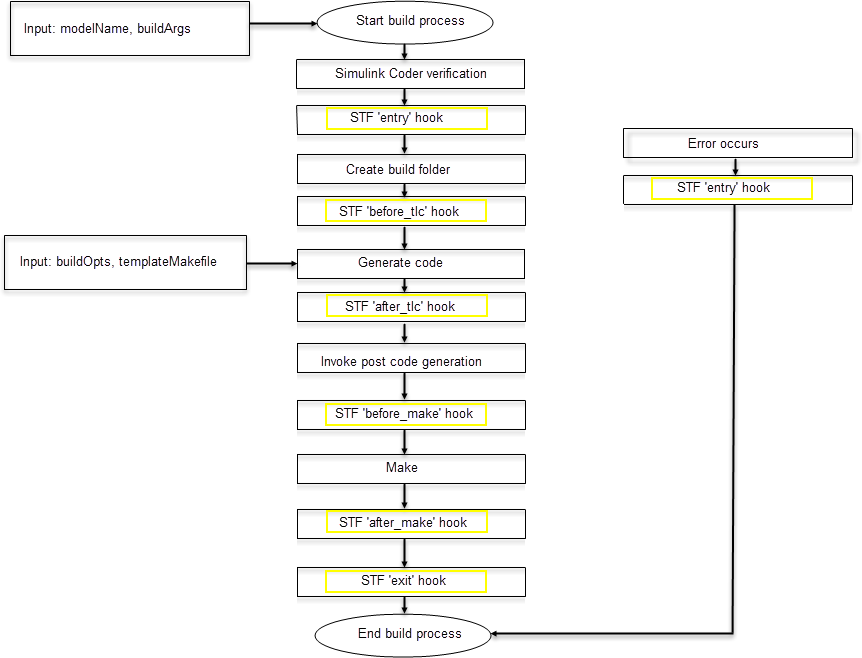
### STF\_make\_rtw\_hook.m 函数原型和参数

**STF\_make\_rtw\_hook** 的函数原型是：

**function STF\_make\_rtw\_hook(hookMethod, modelName, rtwRoot, templateMakefile, buildOpts, buildArgs, buildInfo)**

参数定义为：

* **hookMethod**:指定调用 **STF\_make\_rtw\_hook** 函数的编译过程阶段的字符向量。下面的流程图总结了编译过程，并突出显示了钩子点。**hookMethod** 的有效值为 **'entry'**、**'before\_tlc'**、**'after\_tlc'**、 **'before\_make'**、**'after\_make'**、**'exit'** 和 **'error'**。**STF\_make\_rtw\_hook** 函数使用 **switch** 语句调度给相关代码。



* **modelName**:指定模型名称的字符向量。在编译过程的所有阶段都有效。
* **rtwRoot**:保留。
* **templateMakefile**:模板联编文件的名称。
* **buildOpts**:包含布尔字段 **codeWasUpToDate** 的 MATLAB 结构体。仅在 **'before\_make'**、 **'after\_make'** 和 **'exit'** 阶段有效。
* **buildArgs**:包含 **make\_rtw** 的参数的字符向量。当您调用编译过程时，将复制**配置参数** + **代码生成**

+ **Make 命令**字段中 **"make\_rtw"** 后面的参数，以此作为 **buildArgs**。

例如，**Make 命令**字段中的以下 make 参数

**make\_rtw VAR1=0 VAR2=4**

将生成以下内容：

**% make -f untitled.mk VAR1=0 VAR2=4**

**buildArgs** 参数不适用于工具链方法编译，因为这些编译不允许在 **make\_rtw** 调用中添加 make 参数。在编译器命令行中，要提供同时适用于 TMF 方法和工具链方法编译的自定义定义（例如 **VAR1=0 VAR2=4**），请使用**配置参数** > **代码生成** > **自定义代码** > **定义**字段。

* **buildInfo**:包含编译和链接所生成代码的信息的 **RTW.BuildInfo** 对象。仅可用于 **'after\_tlc'**、 **'before\_make'**、**'after\_make'** 和 **'exit'** 阶段。**'after\_tlc'** 阶段结束时对象中的信息可能不完整。在后面的阶段中，**'before\_make'** 和 **'after\_make'** 钩子方法也可以向该对象添加信息。有关使用 **RTW.BuildInfo** 对象的详细信息，请参阅“Code Compilation Customization”。

### 应用 STF\_make\_rtw\_hook.m

本节说明如何使用 **STF\_make\_rtw\_hook.m** 钩子方法。

通常，使用 **'entry'** 钩子来初始化编译过程，例如，在生成代码之前更改或验证设置。**'entry'** 钩子的用途之一是重新运行最初在目标选择时间运行的自动配置脚本，从而对执行脚本前后的模型参数进行比较，以实现验证目的。

其他钩子点（**'before\_tlc'**、**'after\_tlc'**、**'before\_make'**、**'after\_make'**、**'exit'** 和 **'error'**）可用来与外部工具链、源代码管理工具和其他环境工具对接。

例如，您可以在 **'entry'** 之后的阶段使用 **STF\_make\_rtw\_hook.m** 文件来获取编译文件夹的路径。之后在 **'exit'** 阶段，您就可以在编译文件夹中找到生成的代码文件，并将它们签入到您的版本控制系统。在代码生成或编译过程中发生错误时，您可以使用 **'error'** 清理钩子函数使用的静态数据或全局数据。

**注意** 编译过程会暂时将 MATLAB 工作文件夹更改为 **'before\_make'**、**'after\_make'**、**'exit'** 和 **'error'**阶段的编译文件夹。您的 **STF\_make\_rtw\_hook.m** 文件绝对不可以对编译文件夹的位置做出错误假设。在 **'entry'** 阶段之后，您可以获取编译文件夹的路径。在下面的 MATLAB 代码示例中，编译文件夹路径以字符向量的形式返回给变量 **buildDirPath**。

**buildDirPath = rtwprivate('get\_makertwsettings',gcs,'BuildDirectory');**

**注意** 不要使用 **STF\_make\_rtw\_hook.m** 文件来更改模型配置。例如，不要使用钩子方法进行以下操作：

* 在模型变体之间切换。
* 调用 **set\_param** 函数。

更改模型配置可能会产生意外的代码生成结果。

### 使用 STF\_make\_rtw\_hook.m 控制代码重新生成

重新编译模型时，默认情况下编译过程将执行一些检查，以确定对模型或相关设置所做的更改是否需要重新生成顶层模型代码。（有关判定标准的详细信息，请参阅“Control Regeneration of Top Model Code”。）如果检查确定需要生成顶层模型代码，编译过程将重新生成并编译所有模型代码。如果检查发现为顶层模型生成的代码对模型来说是最新的代码，而且模型设置不要求完全重新生成，则编译过程将不会重新生成顶层模型代码。

无论是否重新生成顶层模型代码，编译过程随后都会调用编译过程钩子，包括 **STF\_make\_rtw\_hook** 函数和生成代码之后执行的命令。以下机制允许您在 **STF\_make\_rtw\_hook** 函数中执行与重新生成代码有关的操作：

* 要强制重新生成代码，请从 **'entry'** 钩子中使用以下函数调用：

**rtw.targetNeedsCodeGen('set', true);**

* 在从 **'before\_tlc'** 到 **'exit'** 的各个钩子中，传递给钩子的 **buildOpts** 结构体有一个布尔字段 **codeWasUpToDate**。如果模型代码是最新代码并且没有重新生成代码，则此字段设置为 **true**；如果代码不是最新代码并且重新生成了代码，则此字段设置为 **false**。您可以根据此字段的值自定义钩子操作。例如：

**...**

**case 'before\_tlc'**

**if buildOpts.codeWasUpToDate**

**%Perform hook actions for up to date model else**

**%Perform hook actions for full code generation end**

**...**

### 将 STF\_make\_rtw\_hook.m 用于您的编译过程

要为您的编译过程创建自定义 **STF\_make\_rtw\_hook** 钩子文件，请按以下步骤复制并编辑位于文件夹

#### matlabroot/toolbox/coder/embeddedcoder（打开）中的 ert\_make\_rtw\_hook.m 示例文件：

1. 将 **ert\_make\_rtw\_hook.m** 复制到一个位于 MATLAB 路径上的文件夹中。根据 [“STF\_make\_rtw\_hook 文件的使用约定” （第 44-2](#_bookmark405) 页）中介绍的命名约定重命名此文件。例如，要将文件用于 GRT 目标 **grt.tlc**，请将其重命名为 **grt\_make\_rtw\_hook.m**。
2. 重命名此文件中的 **ert\_make\_rtw\_hook** 函数以便与文件名匹配。
3. 通过在 **switch hookMethod** 语句的 case 语句中添加代码，实现您需要的钩子。

### 钩子方法 after\_tlc

**after\_tlc** 钩子方法仅为向后兼容而提供。

在 **after\_tlc** 阶段生成的代码的格式不是最终格式。

### 另请参阅相关示例

* “Customize Build Process with sl\_customization.m”

# Simulink Coder 中的运行时数据接口扩展

# Simulink Coder 中的自定义目标开发

* [“自定义目标” （第 46-2 页）](#_bookmark413)
* [“自定义系统目标文件” （第 46-4 页）](#_bookmark414)
* [“设备驱动程序” （第 46-26 页）](#_bookmark448)

## 自定义目标

自定义目标是您创建的软件包，用于将 MATLAB 和 Simulink 设计部署到目标硬件。使用该包，您可以执行任务来优化、原型化、验证应用程序并将其部署到您的硬件上。下表列出了您可以使用 **target** 类和函数执行的任务。

|  |  |
| --- | --- |
| **任务** | **参阅** |
| 为**硬件实现**设置注册新硬件设备和硬件板。 | [“注册新硬件设备” （第 10-10 页）](#_bookmark56)  **target.Board target.Processor**  **target.LanguageImplementation** |
| 使用基于 CMake 的工具链编译目标应用程序。 | “Create Custom CMake Toolchain Definition”  “Generated Code Compilation” |
| 在目标硬件上部署、下载和运行应用程序。 | **target.ExecutionTool**  **target.SystemCommandExecutionTool** |
| 在外部模式仿真期间，调节参数并监控目标应用程序信号。 | “Set Up External Mode Connectivity Between Simulink and Target Hardware”  “External Mode Simulation by Using XCP Communication”  “Customize XCP Server Software” |
| 通过运行处理器在环 (PIL) 仿真测试您的 MATLAB代码或 Simulink 模型与生成的代码之间的数值等效性。 | “Set Up PIL Connectivity by Using target Package” (Embedded Coder)  “Configure and Run PIL Simulation” (Embedded Coder) |
| 通过 PIL 仿真执行代码执行探查。 | “Set Up PIL Connectivity by Using target Package” (Embedded Coder)  “Create Execution-Time Profile for Generated Code” (Embedded Coder) |

下表列出了您可以执行的其他任务以及由 **target** 类和函数定义的自定义目标。

|  |  |
| --- | --- |
| **任务** | **参阅** |
| 生成或注册主文件来调度生成的代码 | “Deploy Applications to Target Hardware” (Embedded Coder)  “Model Multicore Concurrent Tasking Application” (Embedded Coder)  Target operating system (Embedded Coder)  “Generate Source and Header Files with a Custom File Processing (CFP) Template” (Embedded Coder)  “Support C Function Prototype Control” “Support C++ Class Interface Control” |
| 注册编译钩子。 | “Customize Build Process with sl\_customization.m”  “Customize Post-Code-Generation Build Processing” |
| 通过使用特定于处理器的代码替换库优化生成的代码。 | “Code Replacement” |
| 集成实时操作系统和设备驱动程序。 | [“设备驱动程序” （第 46-26 页）](#_bookmark448) |
| 使用钩子和自定义设置定义自定义系统目标文件，以将特定于目标的操作添加到编译过程中。 | “Target Platform Device Customization”  [“自定义系统目标文件” （第 46-4 页）](#_bookmark414)  [“使用 STF\_make\_rtw\_hook 文件自定义编译过程” （第 44-2 页）](#_bookmark403) |

您还可以为 ARM➅ 处理器创建基于 MathWorks 引用目标的自定义软件。例如，请参阅：

* “Embedded Coder Support Package for ARM Cortex-A Processors”
* “Embedded Coder Support Package for ARM Cortex-M Processors”

## 自定义系统目标文件

本节提供关于系统目标文件结构的信息、自定义系统目标文件的规范以及帮助您生成框架性系统目标文件的基本教程。

### 使用系统目标文件控制代码生成

系统目标文件控制编译过程的代码生成阶段。系统目标文件还使您能够控制以什么样的表示方式将您的目标显示给最终用户。系统目标文件提供

* 对编译过程至关重要的变量定义，例如 **CodeFormat** TLC 变量的值
* 生成代码的顶层 TLC 程序的主要入口点
* 要显示在 System Target File Browser 中的目标信息
* 用于定义目标特定的代码生成选项（以及与编译过程相关的其他参数）并在 Configuration Parameters 对话框中显示这些选项的机制
* 从其他目标（例如嵌入式实时 (ERT) 目标）继承选项的机制

请注意，虽然系统目标文件是 Target Language Compiler (TLC) 文件，但它包含嵌入式 MATLAB 代码。在创建或修改系统目标文件之前，您应该掌握 TLC 和 MATLAB 语言的相关知识。“目标语言编译器”和 “脚本与函数” 中介绍了 TLC 和 MATLAB 语言的特点和语法。

阅读本节内容时，您可能需要参考代码生成器提供的系统目标文件。这些文件大部分存储在 **matlabroot/rtw/c** 文件夹（打开）下的目标特定的文件夹中。其他系统目标文件存储在 **matlabroot/ toolbox/rtw/targets**（打开）下。

### 系统目标文件命名和位置约定

系统目标文件必须位于 MATLAB 路径上的文件夹中，这样才能使目标显示在 System Target File Browser中并在编译过程中得到调用。请遵循“Folder and File Naming Conventions”中提供的系统目标文件和相关目标文件的位置和命名约定。

### 系统目标文件的结构

* [“概述” （第 46-4 页）](#_bookmark418)
* [“标头注释” （第 46-6 页）](#_bookmark419)
* [“TLC 配置变量” （第 46-6 页）](#_bookmark420)
* [“TLC 程序入口点和相关的 %include 指令” （第 46-7 页）](#_bookmark421)
* [“RTW\_OPTIONS 部分” （第 46-7 页）](#_bookmark422)
* [“rtwgensettings 结构体” （第 46-8 页）](#_bookmark423)
* [“其他代码生成选项” （第 46-9 页）](#_bookmark424)
* [“模型引用注意事项” （第 46-9 页）](#_bookmark425)

#### 概述

本节介绍系统目标文件的结构和内容。下面的列表显示了系统目标文件的一般结构。请注意，这不是系统目标文件的完整代码列表，只是从构成系统目标文件的每个部分中摘录的片段。

**%%----------------------------**

**%% Header Comments Section**

**%%----------------------------**

**%% SYSTLC: Example Real-Time Target**

**%% TMF: my\_target.tmf MAKE: make\_rtw**

**%% Initial comments contain directives for STF Browser.**

**%% Documentation, date, copyright, and other info may follow.**

**...**

**%selectfile NULL\_FILE**

**...**

**%%----------------------------**

**%% TLC Configuration Variables Section**

**%%----------------------------**

**%% Assign code format, language, target type.**

**%%**

**%assign CodeFormat = "Embedded-C"**

**%assign TargetType = "RT"**

**%assign Language = "C"**

**%%**

**%%----------------------------**

**%% TLC Program Entry Point**

**%%----------------------------**

**%% Call entry point function.**

**%include "codegenentry.tlc"**

**%%**

**%%----------------------------**

**%% (OPTIONAL) Generate Files for Build Process**

**%%----------------------------**

**%include "mytarget\_genfiles.tlc"**

**%%----------------------------**

**%% RTW\_OPTIONS Section**

**%%----------------------------**

**/%**

**BEGIN\_RTW\_OPTIONS**

**%% Define rtwoptions structure array. This array defines target-specific**

**%% code generation variables, and controls how they are displayed. rtwoptions(1).prompt = 'example code generation options';**

**...**

**rtwoptions(6).prompt = 'Show eliminated blocks'; rtwoptions(6).type = 'Checkbox';**

**...**

**%** **%**

**% Configure RTW code generation settings %**

**%** **%**

**...**

**%%----------------------------**

**%% rtwgensettings Structure**

**%%----------------------------**

**%% Define suffix text for naming build folder here. rtwgensettings.BuildDirSuffix = '\_mytarget\_rtw'**

**%% Callback compatibility declaration rtwgensettings.Version = '1';**

**%% (OPTIONAL) target inheritance declaration rtwgensettings.DerivedFrom = 'ert.tlc';**

**%% (OPTIONAL) other rtwGenSettings fields...**

**...**

**END\_RTW\_OPTIONS**

**%/**

**%%----------------------------**

**%% targetComponentClass - MATHWORKS INTERNAL USE ONLY**

**%% REMOVE NEXT SECTION FROM USER\_DEFINED CUSTOM TARGETS**

**%%----------------------------**

**/%**

**BEGIN\_CONFIGSET\_TARGET\_COMPONENT**

**targetComponentClass = 'Simulink.ERTTargetCC'; END\_CONFIGSET\_TARGET\_COMPONENT**

**%/**

如果您基于现有系统目标文件创建自定义目标，则必须删除 **targetComponentClass** 部分（指令

#### BEGIN\_CONFIGSET\_TARGET\_COMPONENT 和

**END\_CONFIGSET\_TARGET\_COMPONENT** 之间的部分）。此部分预留给 MathWorks 内部开发的目标使用。

#### 标头注释

文件开头的这几行采用 TLC 注释的格式。它们为 System Target File Browser 和编译过程提供必要的信息。请注意，必须将浏览器注释放在文件头部，位于其他注释或 TLC 语句之前。

注释的存在使代码生成器能够检测到系统目标文件。打开 System Target File Browser 时，代码生成器将扫描 MATLAB 路径中是否存在具有格式化标头注释的 TLC 文件。注释中包含以下指令：

* SYSTLC：显示在浏览器中的描述符。
* TMF：要在编译过程中使用的模板联编文件 (TMF) 的名称。选择目标后，此文件名将显示在 Configuration Parameters 对话框的 **Code Generation** 窗格的 Template makefile 字段中。
* MAKE：要在编译过程中使用的 **make** 命令。选择目标后，此命令将显示在 Configuration Parameters 对话框的 **Code Generation** 窗格的 **Make command** 字段中。

以下标头注释来自 **matlabroot/rtw/c/ert/ert.tlc**。

**%% SYSTLC: Embedded Coder TMF: ert\_default\_tmf MAKE: make\_rtw**

**%% SYSTLC: Create Visual C/C++ Solution File for Embedded Coder\**

**%% TMF: RTW.MSVCBuild MAKE: make\_rtw**

**.**

**.**

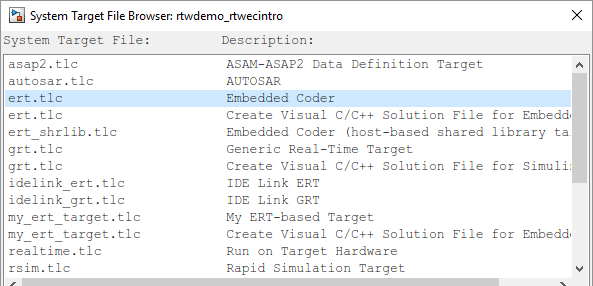
**.**

每个注释最多只能包含两行。

如果不指定系统目标文件中的 **TMF** 字段，该文件仍然有效。要更改 **TemplateMakefile** 参数的值，您可以改用由 **rtwgensettings.SelectCallback** 指定的回调函数。

您也可以使用 **rtwgensettings.SelectCallback** 指定的回调函数来更改外部模式参数 **ExtMode**、 **ExtModeTransport**、**ExtModeMexFile** 或 **ExtModeIntrfLevel** 的值。

您可以在标头注释中指定多组指令。每一组指令在 System Target File Browser 中显示为一种不同的目标配置。在上面的示例中，前两行代码指定 ERT 目标的默认配置。接下来两行指定创建和编译 Microsoft Visual C++ 解决方案 (**.sln**) 文件的配置。下图显示了这些配置在 System Target File Browser 中的显示方式。



有关自定义标头注释的示例，请参阅“创建自定义目标配置” [（第 46-17](#_bookmark439) 页）。

#### TLC 配置变量

系统目标文件中的这一节介绍与整个代码生成过程有关的为全局 TLC 变量赋值的内容。

对于嵌入式目标，大多数情况下只需使用 ERT 目标 (**ert.tlc**) 使用的全局 TLC 变量设置。特别重要的是，您的系统目标文件必须为 **CodeFormat** TLC 变量使用 **'Embedded-C'** 值，并在 TLC 文件的

**RTW\_OPTIONS** 部分使用对应的 **rtwgensettings.DerivedFrom = 'ert.tlc'**。验证并确认已为以下变量赋值：

* **CodeFormat**：**CodeFormat** TLC 变量选择生成的代码的功能。此变量的 **'Embedded-C'** 值由 ERT目标使用。基于 ERT 的目标应指定 **'Embedded-C'** 作为 **CodeFormat** 的值。此选项的作用是实现生产代码、最小化内存使用量、静态内存分配以及为生成的代码提供简化接口。

要了解 **CodeFormat** TLC [变量的其他值的相关信息，请参阅“比较各产品的系统目标文件支持”](#_bookmark166)

[（第 22-5](#_bookmark166) 页）。

* **Language**：唯一有效的值是 **C**，它支持由配置参数 **TargetLang** 指定的 **C** 或 **C++** 代码生成。
* **TargetType**：代码生成器定义预处理器符号 **RT** 和 **NRT**，以区分仿真代码和实时代码。这两个符号在条件编译中使用。**TargetType** 变量确定定义的是 **RT** 还是 **NRT**。

大多数目标都设计用来生成实时代码。它们按如下方式为 **TargetType** 赋值。

**%assign TargetType = "RT"**

而有些目标（比如模型引用仿真目标、加速仿真目标、RSim 目标和 S-Function 目标）生成的代码则仅在非实时时间使用。这些目标按如下方式为 **TargetType** 赋值。

**%assign TargetType = "NRT"**

#### TLC 程序入口点和相关的 %include 指令

代码生成过程通常以 **codegenentry.tlc** 开始。系统目标文件调用 **codegenentry.tlc**，如下所示。

**%include "codegenentry.tlc"**

**注意 codegenentry.tlc** 和底层 TLC 文件都假定已为 **CodeFormat**、**TargetType** 和 **Language** 赋 值。请在包含 **codegenentry.tlc** 之前设置这些变量。

如果您需要实现目标特定的代码生成功能，应在您的系统目标文件中包含 TLC 文件 **mytarget\_genfiles.tlc**。此文件提供了在调用 **codegenentry.tlc** 前后执行自定义 TLC 代码的机制。有关此机制的信息，请参阅：

* [“使用 mytarget\_genfiles.tlc” （第 46-15](#_bookmark435) 页），查看在主代码生成入口点后执行的自定义 TLC 代码的示例。
* “Target Development and the Build Process”，了解有关编译过程的一般信息以及有关其他编译过程自定义钩子的信息。

自定义代码生成过程的另一种方法是直接调用底层函数（通常由 **codegenentry.tlc** 调用），并在代码生成过程的每个阶段包含您自己的 TLC 函数。采用这种方法应慎重。有关详细信息，请参阅“TLC Files”。

**codegenentry.tlc** 调用的底层函数包括：

* **genmap.tlc**：将模块名称映射到对应的语言特定的模块目标文件。
* **commonsetup.tlc**：设置全局变量。
* **commonentry.tlc**：开始代码生成过程。

#### RTW\_OPTIONS 部分

**RTW\_OPTIONS** 部分须置于以下指令之间：

**/%**

**BEGIN\_RTW\_OPTIONS**

**.**

**.**

**.**

**END\_RTW\_OPTIONS**

**%/**

**RTW\_OPTIONS** 的第一部分定义 **rtwoptions** 结构体数组。“使用 [rtwoptions 显示自定义目标选项”](#_bookmark427)

[（第 46-10](#_bookmark427) 页）中介绍了这种结构体。

**RTW\_OPTIONS** 的第二部分定义 **rtwgensettings** 结构体，它定义代码生成过程的编译文件夹名称和其他设置。有关 **rtwgensettings** 的信息，请参阅 [“rtwgensettings 结构体” （第 46-8](#_bookmark423) 页）。

#### rtwgensettings 结构体

系统目标文件的最后一部分定义 **rtwgensettings** 结构体。此结构体存储写入 **model.rtw** 文件并由编译过程使用的信息。目标开发人员最常用到的 **rtwgensettings** 字段包括

* **rtwgensettings.Version**：使用此属性启用 **rtwoptions** 回调，并在

**rtwgensettings.SelectCallback** 中使用回调 API。

**注意** 要使用回调，您必须做如下设置：

**rtwgensettings.Version = '1';**

将上面的语句添加到系统目标文件的 **Configure RTW code generation settings** 部分。

* **rtwgensettings.DerivedFrom**[：此结构体字段定义要从中继承选项的系统目标文件。请参阅“继承目标选项” （第 46-14](#_bookmark431) 页）。
* **rtwgensettings.SelectCallback**：此结构体字段指定 **SelectCallback** 函数。您必须设置 **rtwgensettings.Version = '1';**，否则您的回调将被忽略。**SelectCallback** 与目标关联，而不是与目标的任何选项关联。当您执行以下操作时会触发 **SelectCallback** 函数：
  + 加载模型。
  + 更新 Configuration Parameters 对话框中的任何配置设置。
  + 编译模型。

**SelectCallback** 函数用于设置（或禁用）特定于目标的配置参数。以下代码安装 **SelectCallback** 函数：

**rtwgensettings.SelectCallback = 'my\_select\_callback\_handler(hDlg,hSrc)';**

**SelectCallback** 函数的参数 **(hDlg, hSrc)** 是回调 API 函数使用的私有数据的句柄。

**注意** 如果您开发了自定义目标并希望它与模型引用兼容，则必须实现 **SelectCallback** 函数来声明模型引用兼容性。请参阅“Support Model Referencing”。

* **rtwgensettings.ActivateCallback**：此属性指定 **ActivateCallback** 函数。当模型的活动配置集发生更改时，将触发 **ActivateCallback** 函数。这可能会在加载模型的过程中发生，也可能会在用户更改活动配置集时发生。

以下代码安装 **ActivateCallback** 函数：

**rtwgensettings.ActivateCallback = 'my\_activate\_callback\_handler(hDlg,hSrc)';**

**ActivateCallback** 函数的参数 **(hDlg,hSrc)** 是回调 API 函数使用的私有数据的句柄。

* **rtwgensettings.PostApplyCallback**：此属性指定 **PostApplyCallback** 函数。当用户在 Configuration Parameters 对话框中编辑选项后点击 **Apply** 或 **OK** 按钮时，将触发 **PostApplyCallback** 函数。当更改应用于配置集后，将调用 **PostApplyCallback** 函数。

以下代码安装 **PostApplyCallback** 函数：

**rtwgensettings.PostApplyCallback = 'my\_postapply\_callback\_handler(hDlg,hSrc)';**

**PostApplyCallback** 函数的参数 **(hDlg, hSrc)** 是回调 API 函数使用的私有数据的句柄。

* **rtwgensettings.BuildDirSuffix**：为了标识由编译过程创建的编译文件夹，大多数系统目标文件都会定义一个文件夹名称后缀。为了形成编译文件夹名称，编译过程会将 **rtwgensettings.BuildDirSuffix** 字段中定义的后缀追加到模型名称的后面。您可以通过为 **rtwgensettings.BuildDirSuffix** 指定新字符串值来修改后缀。例如：

**rtwgensettings.BuildDirSuffix = '\_mytarget\_rtw'**

编译过程会创建一个名为 **model\_mytarget\_rtw** [的编译文件夹。有关详细信息，请参阅“编译过程文件夹” （第 25-12](#_bookmark193) 页）。

不要使用函数来指定新字符串值。例如：

**rtwgensettings.BuildDirSuffix = my\_get\_build\_suffix(bdroot);**

#### 其他代码生成选项

“Configure Generated Code with TLC”介绍了其他 TLC 代码生成变量。目标的最终用户可以通过输入以下形式的 MATLAB 命令为这些变量赋值

**set\_param(modelName,'TLCOptions','-aVariable=val');**

（有关详细信息，请参阅“指定 [TLC 以用于代码生成” （第 31-11](#_bookmark270) 页）。）但是，最好还是使用以下形式的语句在系统目标文件中为这些变量赋值：

**%assign Variable = val**

为了增强可读性，建议您在系统目标文件中 **Configure RTW code generation settings** 注释的下方添加这些赋值。

#### 模型引用注意事项

有关系统目标文件以及为了支持代码生成器模型引用功能而需要进行的其他修改的重要信息，请参阅 “Support Model Referencing”。

### 定义和显示自定义目标选项

* [“使用 rtwoptions 显示自定义目标选项” （第 46-10 页）](#_bookmark427)
* [“包含自定义 rtwoptions 的示例系统目标文件” （第 46-13 页）](#_bookmark430)
* [“继承目标选项” （第 46-14 页）](#_bookmark431)

#### 使用 rtwoptions 显示自定义目标选项

通过自定义系统目标文件中的 **rtwoptions** 结构体，可以控制 Configuration Parameters 对话框的

**Code Generation** 窗格中显示的选项。

**rtwoptions** 结构体的字段定义要在 Configuration Parameters 对话框中显示的变量和关联的用户界面元素。使用 **rtwoptions** 结构体数组，您可以定义该对话框中显示的目标特定的选项，并将选项按类别进行组织。还可以编写回调函数，以指定如何处理这些选项。

当 **Code Generation** 窗格打开时，系统将扫描 **rtwoptions** 结构体数组，并显示列出的选项。每个选项由一个指定的用户界面元素（复选框、编辑字段、菜单或按钮）表示，此界面元素显示该选项的当前值。

用户界面元素可以处于启用或禁用（灰显）状态。如果选项处于启用状态，则用户可以更改选项值。如果某个选项被禁用，该选项将使用默认值，并且用户无法更改该选项值。

您也可以使用 **rtwoptions** 结构体数组来定义触发回调函数执行但不显示在 **Code Generation** 窗格中的特殊的 NonUI 元素。请参阅 [“NonUI 元素” （第 46-13](#_bookmark429) 页）。

**rtwoptions** 结构体数组的元素被分成几个组。每个组都以 **Category** 类型的头元素开始。**Category** 头元素的 default 字段必须包含该类别中其余元素的计数。

**Category** 头元素的下方是要显示在 **Code Generation** 窗格上的选项。每个类别中的头后面是一个或多个选项定义元素。

每个类别的目标选项都对应于 Configuration Parameters 对话框中 **Code Generation** 下面列出的选项。

[rtwoptions](#_bookmark428) 结构体字段摘要表总结了 **rtwoptions** 结构体的字段。

**示例 rtwoptions 结构体**

下面的 **rtwoptions** 结构体摘自示例系统目标文件 **matlabroot/toolbox/rtw/rtwdemos/ rtwoptions\_demo/usertarget.tlc**。这段代码定义了一个 **rtwoptions** 结构体数组。第一个（头）元素的 default 字段设置为 **4**，指示头后面的元素数。

**rtwoptions(1).prompt = 'userPreferred target options (I)'; rtwoptions(1).type = 'Category';**

**rtwoptions(1).enable = 'on';**

**rtwoptions(1).default = 4; % number of items under this category**

**% excluding this one. rtwoptions(1).popupstrings = ''; % At the first item, user has to rtwoptions(1).tlcvariable = ''; % initialize all supported fields rtwoptions(1).tooltip = '';**

**rtwoptions(1).callback = ''; rtwoptions(1).makevariable = '';**

**rtwoptions(2).prompt = 'Execution Mode'; rtwoptions(2).type = 'Popup'; rtwoptions(2).default = 'Real-Time';**

**rtwoptions(2).popupstrings = 'Real-Time|UserDefined'; rtwoptions(2).tlcvariable = 'tlcvariable1'; rtwoptions(2).tooltip = ['See this text as tooltip'];**

**rtwoptions(3).prompt = 'Log Execution Time'; rtwoptions(3).type = 'Checkbox'; rtwoptions(3).default = 'on'; rtwoptions(3).tlcvariable = 'RL32LogTETModifier'; rtwoptions(3).tooltip = ['']; % no tooltip**

**rtwoptions(4).prompt = 'Real-Time Interrupt Source'; rtwoptions(4).type = 'Popup';**

**rtwoptions(4).default = 'Timer';**

**rtwoptions(4).popupstrings = 'Timer|5|6|7|8|9|10|11|12|13|14|15'; rtwoptions(4).tlcvariable = 'tlcvariable3';**

**rtwoptions(4).callback = 'usertargetcallback(hDlg, hSrc, ''tlcvariable3'')'; rtwoptions(4).tooltip = [''];**

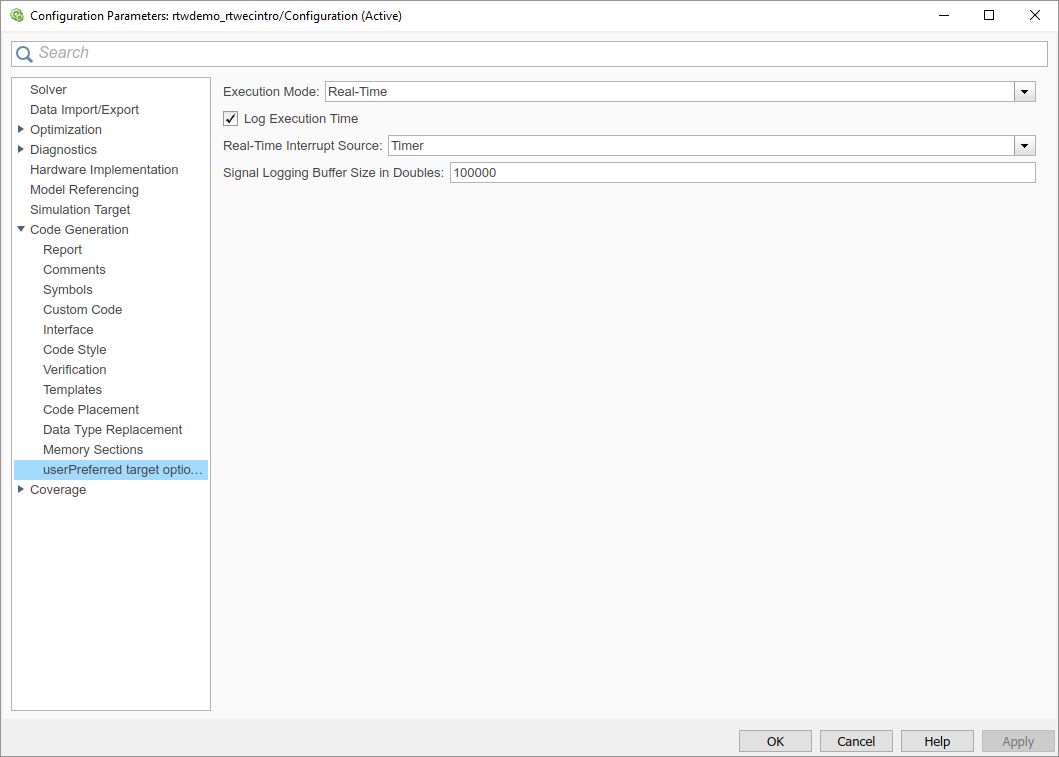
**rtwoptions(4).tooltip = ['See TLC file for how to use reserved '... ' keyword ''hDlg'', and ''hSrc''.'];**

**...**

**rtwoptions(5).prompt = 'Signal Logging Buffer Size in Doubles'; rtwoptions(5).type = 'Edit';**

**rtwoptions(5).default = '100000'; rtwoptions(5).tlcvariable = 'tlcvariable2'; rtwoptions(5).tooltip = [''];**

第一个元素在 Configuration Parameters 对话框中的 **Code Generation** 下添加了一个 **userPreferred target options (I)** 窗格。该窗格显示了 **rtwoptions(2)**、**rtwoptions(3)**、**rtwoptions(4)** 和 **rtwoptions(5)** 中定义的选项。



如果要定义很多选项，可在一个系统目标文件中定义多个 **Category** 组。

请注意，**rtwoptions** 结构体和回调是使用 MATLAB 代码编写的，虽然它们嵌入在 TLC 文件中。要验证 **rtwoptions** 结构体定义和代码的语法，可将命令复制并粘贴到 MATLAB 命令行窗口，在 MATLAB 提示符下执行这些命令。

要了解有关 **usertarget.tlc** 和附带的示例回调文件的详细信息，请参阅“包含自定义 [rtwoptions 的示例系统目标文件” （第 46-13](#_bookmark430) 页）。要查看更多目标特定的 **rtwoptions** 定义示例，请参阅 **matlabroot/rtw/c** 文件夹（打开）下的 **target.tlc** 文件。

[rtwoptions](#_bookmark428) 结构体字段摘要中列出了 **rtwoptions** 结构体的字段。

#### rtwoptions 结构体字段摘要

|  |  |
| --- | --- |
| **字段名称** | **描述** |
| **callback** | 有关回调用法的示例，请参阅“包含自定义 [rtwoptions 的示例系统目标文件” （第 46-13](#_bookmark430) 页）。 |
| **closecallback (obsolete)** | 请勿使用 **closecallback**。  应改用 **rtwgensettings.PostApplyCallback**（请参阅 [“rtwgensettings结构体” （第 46-8](#_bookmark423) 页））。 |
|  | **closecallback** 将被忽略。 |
|  | 有关回调用法的示例，请参阅“包含自定义 [rtwoptions 的示例系统目标文件” （第 46-13](#_bookmark430) 页）。 |
| **default** | 选项的默认值（如果 **type** 为 **Pushbutton**，则为空）。 |
| **enable** | 必须是 **'on'** 或 **'off'**。如果是 **'on'**，选项显示为一个启用的项目；否则，显示为一个禁用的项目。 |
| **makevariable** | 与选项关联的模板联编文件标记（如果有）。在处理模板联编文件期间，  **makevariable** 将被展开。请参阅“Template Makefile Tokens”。 |
| **modelReferenceParameter- Check** | 指定选项在引用模型和父模型中是否必须具有相同的值。如果未指定此字段或值为 **'on'**，则选项值必须相同。如果指定此字段而值为 **'off'**，则选项值可以不同。请参阅“Controlling Configuration Option Value Agreement”。 |
| **NonUI** | 不显示在界面上，但会用来调用 close 或 open 回调的元素。请参阅 [“NonUI元素” （第 46-13](#_bookmark429) 页）。 |
| **opencallback (obsolete)** | 请勿使用 **opencallback**。  应改用 **rtwgensettings.SelectCallback**（请参阅 [“rtwgensettings 结构体” （第 46-8](#_bookmark423) 页））。 |
|  | 有关回调用法的示例，请参阅“包含自定义 [rtwoptions 的示例系统目标文件” （第 46-13](#_bookmark430) 页）。 |
| **popupstrings** | 如果 **type** 是 **Popup**，则 **popupstrings** 定义菜单中的项目。项目之间用 "|"  （竖线）分隔。下面的示例定义 GRT 目标使用的 **MAT-file variable name modifier** 菜单中的项目。  **'rt\_|\_rt|none'** |
| **prompt** | 选项的标签。 |
| **tlcvariable** | 与选项关联的 TLC 变量的名称。 |
| **tooltip** | 将鼠标指针悬停在项目上时显示的帮助文本。 |
| **type** | 元素的类型：**Checkbox**、**Edit**、**NonUI**、**Popup**、**Pushbutton** 或  **Category**。 |

**NonUI 元素**

**rtwoptions** 数组中类型为 **NonUI** 的元素仅用于调用回调。**NonUI** 元素不会显示在 Configuration Parameters 对话框中。如果要在打开或关闭对话框时执行与用户界面元素无关联的回调，则可以使用 **NonUI** 元素。有关示例，请参阅下一节，“包含自定义 [rtwoptions 的示例系统目标文件” （第 46-13页）。](#_bookmark430)

**注意** 类型为 **NonUI** 或 **Edit** 的元素的默认值决定该元素的允许值的集合。

* 如果默认值为 **'0'** 或 **'1'**：
  + 对于类型 **NonUI**，元素存储布尔值。
  + 对于类型 **Edit**，元素存储 **int32** 类型的值。
* 如果默认值包含除 **'0'** 或 **'1'** 以外的整数，则元素中存储 **int32** 类型的值。
* 如果默认值不包含整数，则元素被理解为字符向量。

#### 包含自定义 rtwoptions 的示例系统目标文件

本软件提供了一个工作系统目标文件（包含 MATLAB 文件回调函数），以此为例说明如何使用

**rtwoptions** 结构体来显示和处理 **Code Generation** 窗格上的自定义选项。这些示例与回调 API 兼容。

示例目标文件位于下面的文件夹（打开）中：

**matlabroot/toolbox/rtw/rtwdemos/rtwoptions\_demo**

示例目标文件包括：

* **usertarget.tlc**：示例系统目标文件。此文件说明如何定义自定义菜单、复选框和编辑字段，还说明如何使用回调。
* **usertargetcallback.m**：菜单调用的 MATLAB 文件回调。

阅读本节内容时请参考示例文件。示例系统目标文件 **usertarget.tlc**：说明如何使用 **rtwoptions** 显示以下自定义目标选项：

#### Execution Mode 菜单。

* **Log Execution Time** 复选框。
* **Real-Time Interrupt Source** 菜单。此菜单执行外部文件 **usertargetcallback.m** 中定义的回调。与菜单关联的 TLC 变量传递给回调，回调将显示菜单的当前值。

#### Signal Logging Buffer Size in Doubles 编辑字段。

与 Configuration Parameters 对话框中的示例目标选项交互时，请尝试研究一下示例代码。要与示例目标文件交互，请执行以下操作：

1. 将 **matlabroot/toolbox/rtw/rtwdemos/rtwoptions\_demo**（打开）设置为工作文件夹。
2. 打开您选择的模型。
3. 打开 Configuration Parameters 对话框并选择 **Code Generation** 窗格。
4. 点击 **Browse**。将打开 System Target File Browser。选择 **usertarget.tlc**。然后点击 **OK**。
5. 注意 **Code Generation** 窗格包含一个自定义子选项卡：**userPreferred target options (I)**。
6. 当您与此类别中的选项交互以及打开和关闭 Configuration Parameters 对话框时，请观察 MATLAB命令行窗口显示的消息。这些消息或来自系统目标文件中的代码，或来自从系统目标文件所调用的回调。

#### 继承目标选项

**ert.tlc** 提供一组基本的 Embedded Coder 代码生成选项。如果您的目标基于 **ert.tlc**，则您的系统目标文件通常应继承 ERT 中定义的选项。

可以使用 **rtwgensettings** 结构体中的 **rtwgensettings.DerivedFrom** 字段定义要从中继承选项的系统目标文件。您应按以下说明转换您的自定义目标，以使用此机制。

按以下示例所示设置 **rtwgensettings.DerivedFrom** 字段值：

**rtwgensettings.DerivedFrom = 'stf.tlc';**

其中 **stf** 是要从中继承选项的系统目标文件的名称。例如：

**rtwgensettings.DerivedFrom = 'ert.tlc';**

当 Configuration Parameters 对话框执行这行代码时，它会自动包含 **stf.tlc** 中的选项。如果 **stf.tlc** 是已转换为新布局的 MathWorks 内部系统目标文件，对话框将使用新布局显示继承的选项。

**处理不支持的选项**

如果您的目标不能完全支持从 **ert.tlc** 继承的选项，您应该检测不支持的选项设置并显示警告或错误消息。在某些情况下，如果用户选择了您的目标不支持的选项，您可能需要中止编译过程。例如，如果您的目标不支持 **Generate an example main program** 选项，而此选项被选中，则不应允许编译过程继续下去。

即使您的目标可能无法全部支持继承的 ERT 选项，ERT 选项也要保留在 Configuration Parameters 对话框的 **Code Generation** 窗格中。不要简单地从系统目标文件中的 **rtwoptions** 结构体中删除不支持的选项。当代码生成器执行优化时，这些选项必须存在于对话框中才能被扫描到。

例如，您可能想要阻止用户关闭 **Single output/update function** 选项。那么，从对话框中删除此选项并将 TLC 变量 **CombineOutputUpdateFcns** 简单地指定为 **on** 似乎是合理的做法。但是，如果该选项不包含在对话框中，代码生成器会认为输出函数和更新函数不需要合并。这样将生成效率较低的代码。

### 自定义系统目标文件的提示和技巧

* [“简介” （第 46-14 页）](#_bookmark433)
* [“必需的和推荐的 %include 指令” （第 46-15 页）](#_bookmark434)
* [“处理目标选项值的别名” （第 46-15 页）](#_bookmark436)
* [“支持多个开发环境” （第 46-16 页）](#_bookmark437)
* [“更新您的自定义系统目标文件” （第 46-17 页）](#_bookmark438)

#### 简介

以下各节介绍自定义系统目标文件的各种技巧，包括：

* 如何从系统目标文件中调用自定义 TLC 代码
* 支持多个开发环境的方法
* 更新系统目标文件时的注意事项

#### 必需的和推荐的 %include 指令

如果您需要实现目标特定的代码生成功能，我们建议在您的系统目标文件中包含 TLC 文件

#### mytarget\_genfiles.tlc。

当您的系统目标文件设置必需的 TLC 环境后，您必须包含 **codegenentry.tlc** 才能启动标准代码生成过程。

**mytarget\_genfiles.tlc** 提供了一种在主代码生成入口点之后执行自定义 TLC [代码的机制。请参阅“使用](#_bookmark435) [mytarget\_genfiles.tlc” （第 46-15](#_bookmark435) 页）。

**使用 mytarget\_genfiles.tlc**

**mytarget\_genfiles.tlc**（可选）是一个很有用的中心文件，您可从中调用目标特定的 TLC 文件来生成其他文件，作为目标编译过程的一部分。例如，您的目标可能会为开发环境创建子联编文件或工程文件，或者为调试器创建命令脚本以便执行自动下载。

编译过程然后可以直接从 make 进程中调用这些生成的文件，或者在创建可执行文件后调用这些文件。这是通过 **STF\_make\_rtw\_hook.m** 机制完成的，如“使用 [STF\_make\_rtw\_hook 文件自定义编译过程”](#_bookmark403)

[（第 44-2](#_bookmark403) 页）所述。

以下 TLC 代码显示了示例 **mytarget\_genfiles.tlc** 文件。

**%selectfile NULL\_FILE**

**%assign ModelName = CompiledModel.Name**

**%% Create Debugger script**

**%assign model\_script\_file = "%<ModelName>.cfg"**

**%assign script\_file = "debugger\_script\_template.tlc"**

**%if RTWVerbose**

**%selectfile STDOUT**

**### Creating %<model\_script\_file>**

**%selectfile NULL\_FILE**

**%endif**

**%include "%<script\_file>"**

**%openfile bld\_file = "%<model\_script\_file>"**

**%<CreateDebuggerScript()>**

**%closefile bld\_file**

#### 处理目标选项值的别名

本节介绍了一些工具函数，在测试用户为目标设备类型 (**ProdHWDeviceType**) 和代码替换库 (**CodeReplacementLibrary**) 所指定的值时，您可以使用这些函数来检测和解析别名值或已有值。

**RTW.isHWDeviceTypeEq**

要测试两个目标设备类型值是否表示同一台硬件设备，请调用以下函数：

**result = RTW.isHWDeviceTypeEq(type1,type2)**

其中 **type1** 和 **type2** 是包含目标设备类型值或别名的字符向量。

如果 **type1** 和 **type2** 是表示同一台硬件设备的字符向量，则 **RTW.isHWDeviceTypeEq** 函数返回 true。例如，以下调用返回 true：

**RTW.isHWDeviceTypeEq('Specified','Generic->Custom')**

有关目标设备类型选项 **ProdHWDeviceType** 的说明，请参阅 **Hardware Implementation** 窗格参数 “Device vendor” 和 “Device type” 的命令行信息。

**RTW.resolveHWDeviceType**

要返回硬件设备的设备类型值（假设该值可能是别名或已有值），请调用以下函数：

**result = RTW.resolveHWDeviceType(type)**

其中 **type** 是包含目标设备类型值或别名的字符向量。

**RTW.resolveHWDeviceType** 函数返回设备的设备类型值。例如，以下调用都返回 **'Generic-**

#### >Custom'：

**RTW.resolveHWDeviceType('Specified') RTW.resolveHWDeviceType('Generic->Custom')**

有关目标设备类型选项 **ProdHWDeviceType** 的说明，请参阅 **Hardware Implementation** 窗格参数 “Device vendor” 和 “Device type” 的命令行信息。

**RTW.isTflEq**

要测试两个代码替换库 (CRL) 名称是否表示同一个 CRL，请调用以下函数：

**result = RTW.isTflEq(name1,name2)**

其中 **name1** 和 **name2** 是包含 CRL 值或别名的字符向量。

如果 **name1** 和 **name2** 是表示同一个代码替换库的字符向量，则 **RTW.isTflEq** 函数返回 true。例如，以下调用返回 true：

**RTW.isTflEq('GNU','GNU C99 extensions')**

有关 **CodeReplacementLibrary** 参数的说明，请参阅 Code replacement library。

**RTW.resolveTflName**

要返回代码替换库的 CRL 值（假设该值可能是别名或已有值），请调用以下函数：

**result = RTW.resolveTflName(name)**

其中 **name** 是包含 CRL 值或别名的字符向量。

**RTW.resolveTflName** 函数返回引用的代码替换库的值。例如，以下调用都返回 **'GNU C99 extensions'**：

**RTW.resolveTflName('GNU') RTW.resolveTflName('GNU C99 extensions')**

有关 **CodeReplacementLibrary** 参数的说明，请参阅 Code replacement library。

#### 支持多个开发环境

您的目标可能需要支持多个开发环境（例如，两个或多个交叉编译器）或多个代码生成模式（例如，是为编译器生成一个二进制可执行文件还是生成一个工程文件）。

要满足这项要求，一种方法是实现多个系统目标文件。每个系统目标文件为开发环境调用一个模板联编文件。这相当于提供两个单独的目标。

#### 更新您的自定义系统目标文件

更新自定义系统目标文件可能会影响使用更新后的系统目标文件加载的模型的选项值。如果禁用某个选 项，更新后的系统目标文件将使用该选项的默认值。如果模型对该选项具有不同值，则当您用更新后的系统目标文件加载模型时，模型中的值将被丢弃，而系统目标文件将使用默认值。

### 创建自定义目标配置

* [“简介” （第 46-17 页）](#_bookmark440)
* [“my\_ert\_target 概述” （第 46-17 页）](#_bookmark441)
* [“创建目标文件夹” （第 46-19 页）](#_bookmark443)
* [“创建基于 ERT 的、与工具链兼容的系统目标文件” （第 46-19 页）](#_bookmark444)
* [“创建基于 ERT 的模板联编文件” （第 46-23 页）](#_bookmark445)
* [“创建测试模型和 S-Function” （第 46-24 页）](#_bookmark446)
* [“验证目标操作” （第 46-25 页）](#_bookmark447)

#### 简介

本教程为“Sample Custom Targets”中介绍的示例目标指南提供补充信息。要查看简介和示例文件，请先尝试创建示例目标。

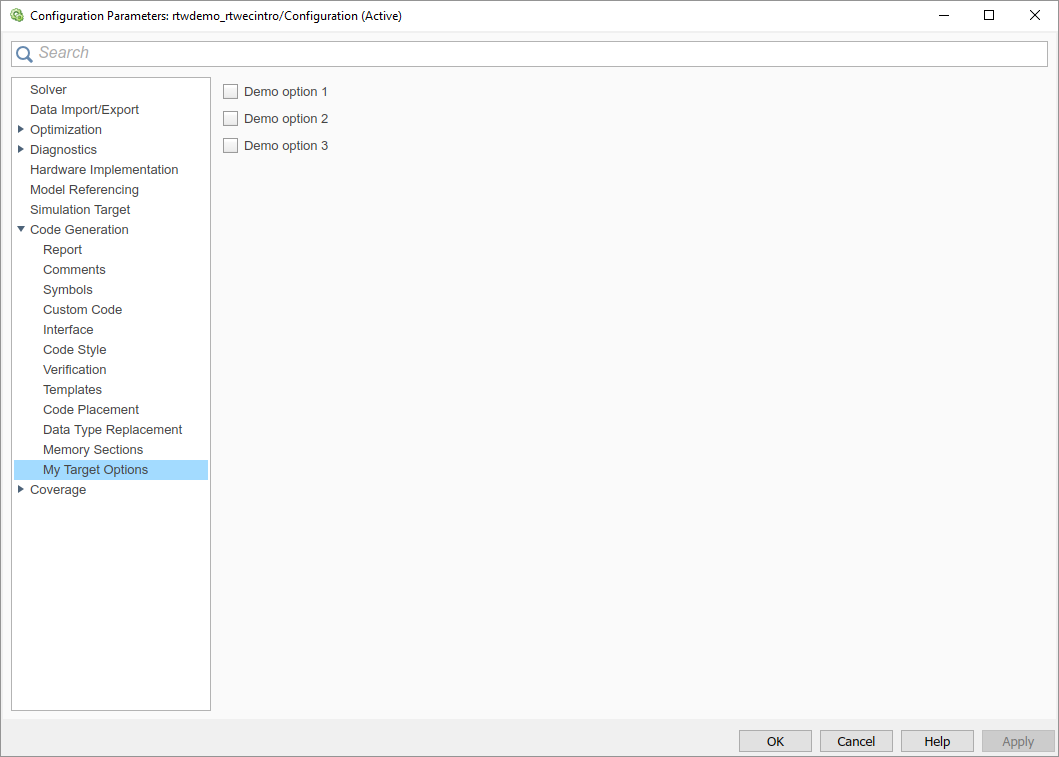
本教程将指导您创建一个基于 ERT 的目标 **my\_ert\_target**。此练习说明了以下几个任务，它们是创建自定义目标的典型任务：

* 设置目标文件夹并修改 MATLAB 路径。
* 修改标准系统目标文件和 TMF，使自定义目标在 System Target File Browser 中可见、继承 ERT 选项、显示目标特定的选项并使用基于主机的默认编译器生成代码。
* 使用自定义目标和包含一个内联 S-Function 的简单模型测试编译过程。

在本练习中，您将实现一个可操作的、但较为基础的基于 ERT 的目标。此目标可作为完整实现自定义嵌入式目标的一个起点。

#### my\_ert\_target 概述

在以下各节，您将创建一个较为基础的目标 **my\_ert\_target**。此目标继承并支持 ERT 目标的标准选项，还在 Configuration Parameters 对话框中显示目标特定的附加选项（请参阅 [“my\_ert\_target 的目标特定的选项” （第 46-18](#_bookmark442) 页））。



#### my\_ert\_target 的目标特定的选项

**my\_ert\_target** 支持基于工具链的编译，生成可在主机系统上运行的代码和可执行文件。 **my\_ert\_target** 在 Microsoft Windows 平台上使用 **lcc** 编译器。此编译器随时可用，并与代码生成器一起分发。在 Microsoft Windows 平台上，如果您使用其他编译器，可以通过以下 MATLAB 命令将 **lcc** 暂时设置为您的默认编译器：

**mex -setup**

软件会显示您计算机上安装的支持的编译器的链接。请点击 **lcc** 的链接。

**注意** 在 Linux 系统上，请确保您已安装 C 编译器。如果已经安装，则可以使用 Linux 文件夹语法来完成此练习。

**my\_ert\_target** 也可以支持基于模板联编文件的编译。有关将此目标与模板联编文件方法一起使用的详细信息，请参阅“创建基[于 ERT 的模板联编文件” （第 46-23 页）](#_bookmark445)。

您可以使用与 ERT 目标兼容的模型（请参阅“配置系统目标文件” (Embedded Coder)）来测试

**my\_ert\_target**。生成的程序与 ERT 生成的程序以相同的方式运行。

要简化您的目标测试，请使用 **targetmodel** [进行测试，这是一个非常简单的定步长模型（请参阅“创建测试模型和 S-Function” （第 46-24](#_bookmark446) 页））。**targetmodel** 中的 S-Function 模块使用 **timestwo** 示例中的源代码，并生成完全内联的代码。有关 **timestwo** 示例 S-Function 的进一步讨论，请参阅 “S-

Function Examples”和“Inline S-Functions with TLC”。

#### 创建目标文件夹

遵循建议的约定创建存储目标文件的文件夹，并将它们添加到 MATLAB 路径中（请参阅“Folder and File Naming Conventions”）。还要创建一个文件夹来存储测试模型、S-Function 和生成的代码。

此示例假定您的目标和模型文件夹位于文件夹 **c:/work** 中。不要将您的目标和模型文件夹放在 MATLAB文件夹树中（即，在 **matlabroot** 文件夹中或在此文件夹下）。

要创建文件夹并使它们可访问，请执行以下操作：

1. 创建目标根文件夹 **my\_ert\_target**。在 Windows 平台上，从 MATLAB 命令行窗口中输入：

**cd c:/work**

**mkdir my\_ert\_target**

1. 在目标根文件夹中创建一个子文件夹，以存储您的目标文件。

**mkdir my\_ert\_target/my\_ert\_target**

1. 将这些文件夹添加到您的 MATLAB 路径中。

**addpath c:/work/my\_ert\_target**

**addpath c:/work/my\_ert\_target/my\_ert\_target**

1. 创建文件夹 **my\_targetmodel**，以存储测试模型、S-Function 和生成的代码。

**mkdir my\_targetmodel**

#### 创建基于 ERT 的、与工具链兼容的系统目标文件

通过复制和修改 ERT 目标的标准系统目标文件，为您的目标创建一个系统目标文件。然后通过查看 System Target File Browser 和 Configuration Parameters 对话框中的新目标来验证该系统目标文件。

**编辑系统目标文件**

要编辑系统目标文件，请按以下步骤操作：

1. 将您的工作文件夹更改为您在“创建目标文件夹” [（第 46-19](#_bookmark443) 页）中创建的文件夹。

**cd c:/work/my\_ert\_target/my\_ert\_target**

#### 将 matlabroot/rtw/c/ert/ert.tlc 复制到 c:/work/my\_ert\_target/my\_ert\_target 中并重命名为

**my\_ert\_target.tlc**。文件 **ert.tlc** 是 ERT 目标的系统目标文件。

1. 在您选择的文本编辑器中打开 **my\_ert\_target.tlc**。
2. 自定义系统目标文件，将标头注释行替换为能让您的系统目标文件在 System Target File Browser 中可见的指令以及其他用来定义关联的 TMF、**make** 命令和外部模式接口文件（如果有）的指令。有关这些指令的详细信息，请参阅“标头注释” [（第 46-6](#_bookmark419) 页）。

用以下标头注释替换 **my\_ert\_target.tlc** 中的标头注释。

**%% SYSTLC: My ERT-based Target TMF: my\_ert\_target\_lcc.tmf MAKE: make\_rtw**

1. 文件 **my\_ert\_target.tlc** [按照“继承目标选项” （第 46-14](#_bookmark431) 页）中介绍的机制继承标准 ERT 选项。因此，现有的 **rtwoptions** 结构体定义是多余的。编辑 RTW\_OPTIONS 部分，使其仅包含以下代

码。

**/%**

**BEGIN\_RTW\_OPTIONS**

**%** **%**

**% Configure RTW code generation settings %**

**%** **%**

**rtwgensettings.BuildDirSuffix = '\_ert\_rtw'; END\_RTW\_OPTIONS**

**%/**

1. 删除 **RTW\_OPTIONS** 部分结束之后的代码，即指令

#### BEGIN\_CONFIGSET\_TARGET\_COMPONENT and

**END\_CONFIGSET\_TARGET\_COMPONENT** 所包围的部分。这段代码仅供 MathWorks 内部开发人员使用。

1. 根据“r[twgensettings 结构体” （第 46-8](#_bookmark423) 页）中介绍的约定修改 **rtwgenSettings** 结构体中的编译文件夹后缀。

要将 **\_my\_ert\_target** 自定义目标的编译文件夹后缀设置为字符向量，请将代码行

**rtwgensettings.BuildDirSuffix = '\_ert\_rtw'**

更改为

**rtwgensettings.BuildDirSuffix = '\_my\_ert\_target\_rtw'**

1. 按照“r[twgensettings 结构体” （第 46-8](#_bookmark423) 页）中的说明修改 **rtwgenSettings** 结构体，以便从 ERT 目标继承选项并声明与 14 版或更高版本的兼容性。将以下代码添加到 **rtwgenSettings** 定义中：

**rtwgensettings.DerivedFrom = 'ert.tlc'; rtwgensettings.Version = '1';**

1. 在 **BEGIN\_RTW\_OPTIONS** 指令之后添加一个 **rtwoptions** 结构体，它定义一个目标特定的选项类别，其中包含三个复选框。以下代码显示了完整的 **RTW\_OPTIONS** 部分，包括之前的 **rtwgenSettings** 更改。

**/%**

**BEGIN\_RTW\_OPTIONS**

**rtwoptions(1).prompt = 'My Target Options'; rtwoptions(1).type = 'Category'; rtwoptions(1).enable = 'on';**

**rtwoptions(1).default = 3; % number of items under this category**

**% excluding this one. rtwoptions(1).popupstrings = ''; rtwoptions(1).tlcvariable = ''; rtwoptions(1).tooltip = ''; rtwoptions(1).callback = ''; rtwoptions(1).makevariable = '';**

**rtwoptions(2).prompt = 'Demo option 1'; rtwoptions(2).type = 'Checkbox'; rtwoptions(2).default = 'off'; rtwoptions(2).tlcvariable = 'DummyOpt1'; rtwoptions(2).makevariable = '';**

**rtwoptions(2).tooltip = ['Demo option1 (non-functional)']; rtwoptions(2).callback = '';**

**rtwoptions(3).prompt = 'Demo option 2'; rtwoptions(3).type = 'Checkbox'; rtwoptions(3).default = 'off'; rtwoptions(3).tlcvariable = 'DummyOpt2'; rtwoptions(3).makevariable = '';**

**rtwoptions(3).tooltip = ['Demo option2 (non-functional)']; rtwoptions(3).callback = '';**

**rtwoptions(4).prompt = 'Demo option 3'; rtwoptions(4).type = 'Checkbox';**

**rtwoptions(4).default = 'off'; rtwoptions(4).tlcvariable = 'DummyOpt3'; rtwoptions(4).makevariable = '';**

**rtwoptions(4).tooltip = ['Demo option3 (non-functional)']; rtwoptions(4).callback = '';**

**%** **%**

**% Configure RTW code generation settings %**

**%** **%**

**rtwgensettings.BuildDirSuffix = '\_my\_ert\_target\_rtw'; rtwgensettings.DerivedFrom = 'ert.tlc'; rtwgensettings.Version = '1';**

**rtwgensettings.SelectCallback = 'enableToolchainCompliant(hSrc, hDlg)';**

**%SelectCallback provides toolchain approach support, but requires custom function**

**%Omit this SelectCallback if using the template makefile approach**

**END\_RTW\_OPTIONS**

**%/**

1. 将更改保存到 **my\_ert\_target.tlc** 并关闭该文件。

**创建与工具链兼容的函数**

要使编译使用工具链方法，需要在自定义系统目标文件接近末尾的位置创建一个与 **SelectCallback** 对应的函数。此函数用来设置工具链兼容性属性。

**function enableToolchainCompliant(hSrc, hDlg)**

**% The following parameters enable toolchain compliance. slConfigUISetVal(hDlg, hSrc, 'UseToolchainInfoCompliant', 'on'); slConfigUISetVal(hDlg, hSrc, 'GenerateMakefile','on');**

**% The following parameters are not required for toolchain compliance.**

**% But, it is recommended practice to set these default values and**

**% disable the parameters (as shown).**

**slConfigUISetVal(hDlg, hSrc, 'RTWCompilerOptimization','off'); slConfigUISetVal(hDlg, hSrc, 'MakeCommand','make\_rtw'); slConfigUISetEnabled(hDlg, hSrc, 'RTWCompilerOptimization',false); slConfigUISetEnabled(hDlg, hSrc, 'MakeCommand',false); hSrc.refreshDialog;**

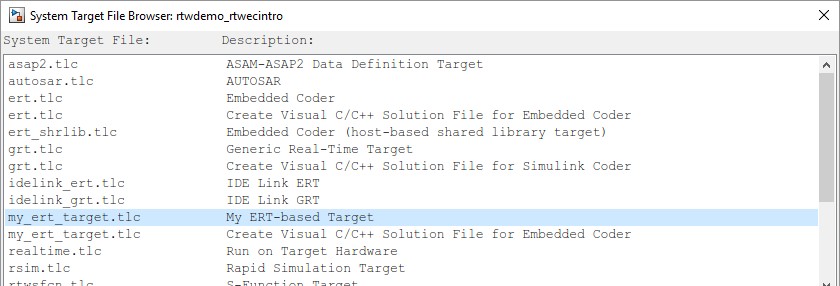
**end**

**注意** 如果您使用模板联编文件方法，则可省略从系统目标文件中调用该函数以实现工具链兼容性的步骤，而改用“创建基[于 ERT 的模板联编文件” （第 46-23](#_bookmark445) 页）中的信息。

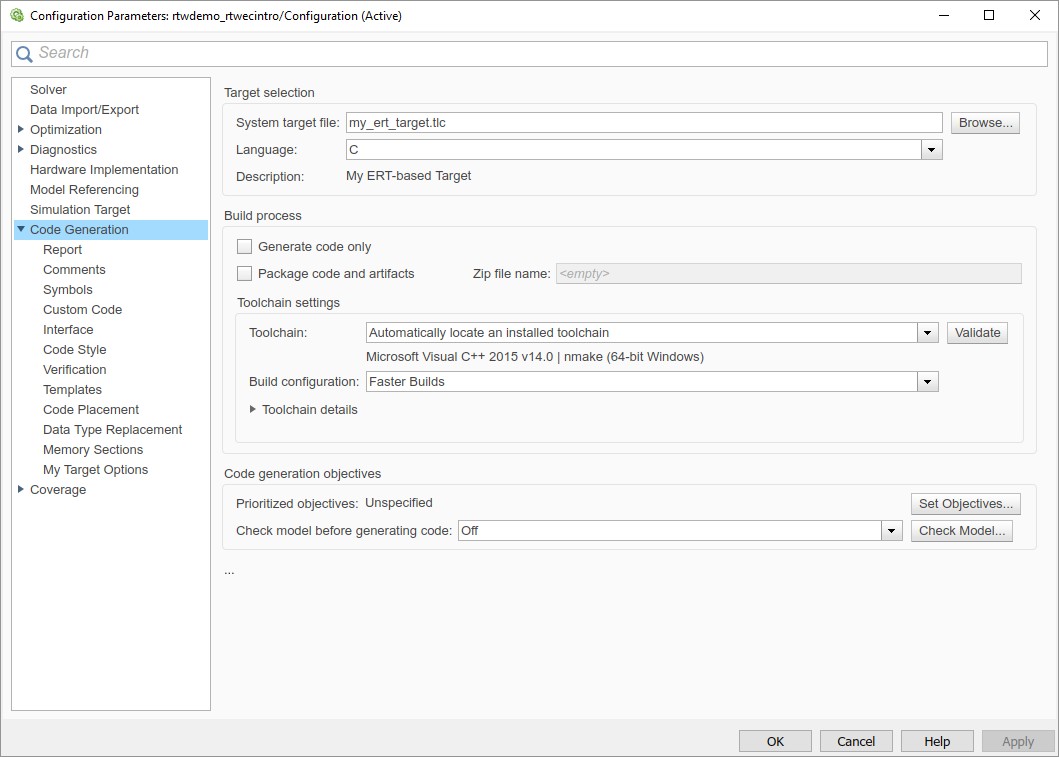
**查看系统目标文件**

此时，您可以按以下步骤验证目标是否继承并显示 ERT 选项：

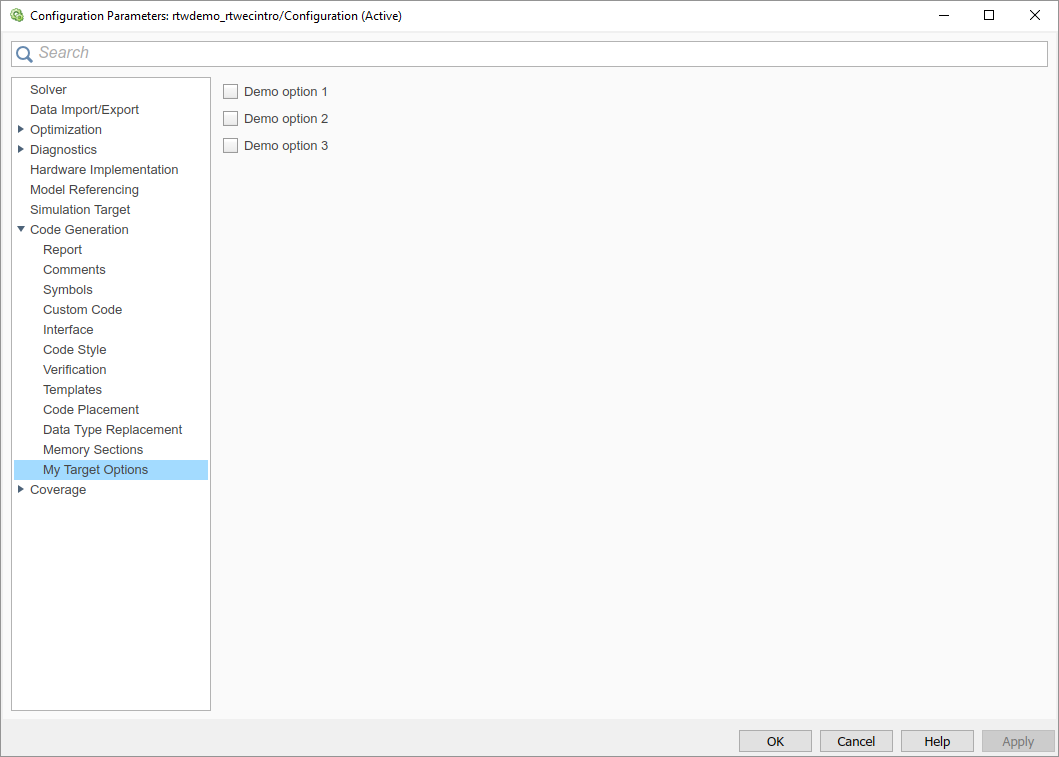
1. 创建一个新模型。
2. 打开模型资源管理器或 Configuration Parameters 对话框。
3. 选择 **Code Generation** 窗格。
4. 点击 **Browse** 以打开 System Target File Browser。
5. 在文件浏览器中，滚动目标列表以查找新目标 **my\_ert\_target.tlc**。（此步骤假设您的 MATLAB 路径中包含 **c:/work/my\_ert\_target/my\_ert\_target**，如之前在“创建目标文件夹” [（第 46-19 页）中所做的设置。）](#_bookmark443)
6. 选择 **My ERT-based Target**，然后点击 **OK**。



1. **Code Generation** 窗格现在显示已针对 **my\_ert\_target.tlc** 目标对模型进行了配置。**System target file**、**Language**、**Toolchain** 和 **Build configuration** 字段应如下所示：



1. 选择 **My Target Options** 窗格。目标显示在 **rtwoptions** 结构体中定义的三个复选框选项。



1. 选择 **Code Generation** 窗格，然后重新打开 System Target File Browser。
2. 选择 Embedded Coder 目标 (**ert.tlc**)。目标显示标准 ERT 选项。
3. 关闭模型。您不需要保存它。

至此，基础目标的系统目标文件已完成。如果您使用工具链方法，现在即可为您的目标调用编译过程。

如果您更喜欢使用模板联编文件方法，则在系统目标文件标头注释中引用 TMF **my\_ert\_target\_lcc.tmf**将会阻止您为目标调用编译过程，直到该 TMF 文件就绪为止。因此，您必须先创建 **my\_ert\_target\_lcc.tmf** 文件。

#### 创建基于 ERT 的模板联编文件

如果您使用工具链联编文件方法以及与工具链兼容的自定义目标，则可以省略适用于模板联编文件方法的步骤。（请跳过本节内容。）

如果您使用模板联编文件方法，请按照适用于模板联编文件的步骤进行操作，并省略从系统目标文件中调用函数以实现工具链兼容性的步骤，如“创建基于 [ERT 的、与工具链兼容的系统目标文件” （第 46-19页）](#_bookmark444) 中所述。

通过复制和修改提供的 ERT 模板联编文件，为您的目标创建一个模板联编文件。选择最适合您的目标的模板联编文件。此示例使用 **ert\_lcc64.tmf**，但您也可以使用 **ert\_vcx64.tmf** 或 **ert\_unix.tmf**。

1. 确保您的工作文件夹仍然设置为您之前在“创建目标文件夹” [（第 46-19](#_bookmark443) 页）中创建的目标文件夹。

**c:/work/my\_ert\_target/my\_ert\_target**

1. 将 **matlabroot/toolbox/coder/compile/tmf/ert\_lcc64.tmf** 复制到 **c:/work/my\_ert\_target/ my\_ert\_target** 中并重命名为 **my\_ert\_target\_lcc.tmf**。文件 **ert\_lcc64.tmf** 是适合 LCC 编译器的 ERT 编译器特定的模板联编文件。
2. 在文本编辑器中打开 **my\_ert\_target\_lcc.tmf**。
3. 更改 **SYS\_TARGET\_FILE** 参数，以便在 **make** 文件中生成对您的 **.tlc** 文件的文件引用。将以下代码行

**SYS\_TARGET\_FILE = any**

更改为

**SYS\_TARGET\_FILE = my\_ert\_target.tlc**

1. 将更改保存到 **my\_ert\_target\_lcc.tmf** 并关闭该文件。

现在，您的目标即可生成代码并编译基于主机的可执行文件。在下面几节，您将创建一个测试模型并使用

**my\_ert\_target** 来测试编译过程。

#### 创建测试模型和 S-Function

在本节中，您将构建一个简单的测试模型，以便在后面的代码生成中继续使用：

1. 将您的工作文件夹设置为 **c:/work/my\_targetmodel**。

**cd c:/work/my\_targetmodel**

在本教程的其余部分，假定 **my\_targetmodel** 就是工作文件夹。您的目标将代码生成过程的输出文件写入工作文件夹内的编译文件夹中。为 **timestwo** S-Function 生成内联代码时，编译过程将在工作文件夹中查找 S-Function 的 TLC 实现。

1. 将 **timestwo** S-Function 的以下 C 和 TLC 文件复制到工作文件夹中：

#### matlabroot/toolbox/simulink/simdemos/simfeatures/src/timestwo.c

* + **matlabroot/toolbox/simulink/simdemos/simfeatures/tlc\_c/timestwo.tlc**

1. 在 **c:/work/my\_targetmodel** 中生成 **timestwo** MEX 文件。

**mex timestwo.c**

1. 使用 Simulink User-Defined Functions 库中的 S-Function 模块创建以下模型。将该模型保存到工作文件夹中并命名为 **targetmodel**。



1. 双击 S-Function 模块以打开 Block Parameters 对话框。输入 S-Function 名称 **timestwo**。点击

**OK**。该模块现在绑定到 **timestwo** MEX 文件。

1. 打开模型资源管理器或 Configuration Parameters 对话框，然后选择 **Solver** 窗格。
2. 将求解器的 **Type** 设置为 “**fixed-step**”，然后点击 **Apply**。
3. 保存模型。
4. 打开示波器并运行仿真。验证并确认 **timestwo** S-Function 将其输入乘以 2.0。

将 **targetmodel** 模型保持打开状态供下一节使用，下一节将使用测试模型生成代码。

#### 验证目标操作

在本节中，您将为 **my\_ert\_target** 自定义目标配置 **targetmodel**，然后使用该目标生成代码并编译可执行文件：

1. 在 Configuration Parameters 对话框中，选择 **Code Generation** 窗格。
2. 点击 **Browse** 以打开 System Target File Browser。
3. 在该浏览器中，选择 **My ERT-based Target**，然后点击 **OK**。
4. Configuration Parameters 对话框现在显示 **my\_ert\_target** 的 **Code Generation** 窗格。
5. 选择 **Code Generation** > **Report** 窗格，然后选择 **Create code generation report** 选项。
6. 点击 **Apply** 并保存模型。这样即为 **my\_ert\_target** 配置了该模型。
7. 编译模型。如果编译成功，MATLAB 命令行窗口将显示以下消息。

**### Created executable: ../targetmodel.exe**

**### Successful completion of build procedure for model:**

**targetmodel**

您的工作文件夹包含 **targetmodel.exe** 文件和编译文件夹 **targetmodel\_my\_ert\_target\_rtw**，编译文件夹中包含生成的代码和其他文件。工作文件夹中还包含供编译过程在内部使用的 **slprj** 文件

夹。

代码生成器还创建代码生成报告。

1. 要查看生成的代码，请在 **C Code** 选项卡上，点击 **View Code**。在代码生成报告的 **Contents** 窗格中，点击 **targetmodel.c** 链接。
2. 在 **targetmodel.c** 中，找到模型阶跃函数 **targetmodel\_step**。观察以下代码。

**/\* S-Function Block: <Root>/S-Function \*/**

**/\* Multiply input by two \*/**

**targetmodel\_B.SFunction = targetmodel\_B.SineWave \* 2.0;**

这段代码的存在证实了 **my\_ert\_target** 自定义目标已为模型中的 S-Function 模块生成了内联输出计算。

### 另请参阅详细信息

* + [“自定义目标” （第 46-2 页）](#_bookmark413)
  + “Support Toolchain Approach with Custom Target”
  + “Support Model Referencing”
  + “Support Compiler Optimization Level Control”
  + “Support C Function Prototype Control”
  + “Support C++ Class Interface Control”
  + “Support Concurrent Execution of Multiple Tasks”

## 设备驱动程序

对于许多实时开发项目来说，与目标硬件通信的设备驱动程序是必不可少的。

您可以使用代码继承工具将现有 C（或 C++）设备驱动程序函数集成到 Simulink 模型中。当您使用代码生成器从模型生成代码时，代码继承工具可以将对 C 函数的调用插入到生成的代码中。有关详细信息，请[参阅“使用代码继承工具在生成的代码中导入对外部代码的调用” （第 14-5](#_bookmark79) 页）和“Integrate C Functions into Simulink Models with Legacy Code Tool”。

### 另请参阅详细信息

* + [“自定义目标” （第 46-2 页）](#_bookmark413)
  + “Sample Custom Targets”
  + [“自定义系统目标文件” （第 46-4 页）](#_bookmark414)