符合ISO 26262的汽车电子软件开发流程

董淑成 Shucheng.dong@mathworks.cn
MathWorks 中国
高完整性软件开发标准和基于模型的设计
软件开发标准里出现基于模型的设计
为什么？
大纲

- ISO 26262软件开发项目的启动
- 符合ISO 26262的软件开发过程
ISO 26262定义的软件开发过程
ISO 26262的软件项目启动

1. 软件开发计划
2. 软件验证计划
3. 编程、建模语言的选择
4. 编码、建模标准
5. 工具的选择
6. 工具应用指南
建模/编程语言的选择及相关标准

- 建模或者编程语言的选择标准
  - 明确的定义
  - 支持嵌入式实时软件和运行时错误处理
  - 支持模块化、抽象及结构化

- 语言本身不能涵盖的上述标准应通过相应的指导或开发环境涵盖

- 通常，汽车电子软件选择C语言
  - 基础软件手工编写C代码
  - 控制策略软件通过Simulink建模并自动生成代码C代码

- 建模/编码标准要涵盖的内容

<table>
<thead>
<tr>
<th>Topics</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td>1a Enforcement of low complexity</td>
<td>++</td>
</tr>
<tr>
<td>1b Use of Language subsets</td>
<td>++</td>
</tr>
<tr>
<td>1c Enforcement of strong typing</td>
<td>++</td>
</tr>
<tr>
<td>1d Use of defensive implementation technique</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td>1e Use of established design principles</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td>1f Use of unambiguous graphical representation</td>
<td>+</td>
</tr>
<tr>
<td>1g Use of style guides</td>
<td>+</td>
</tr>
<tr>
<td>1h Use of naming conventions</td>
<td>++</td>
</tr>
</tbody>
</table>
Simulink/Stateflow 建模标准

- 汽车行业建模标准（MAAB）
  - 专门为汽车行业Simulink用户制定

- 高完整性系统建模标准
  - 专门为民航、火车、汽车等高完整性系统建模制定
设计工具/验证工具的选择

- 工具的分类及资质审核

注：ISO 26262要求对工具进行资质审核
TÜV SÜD认证的工具

- **Embedded Coder™**
  功能：生产针对嵌入式优化的C和C++代码

- **Simulink® Verification and Validation™**
  功能：验证模型和模型生成的代码

- **Simulink® Design Verifier™**
  功能：定位设计错误，生成测试用例，并根据需求对设计进行验证

- **Polyspace® Client™ for C/C++**
  功能：证明源代码没有运行期错误

- **Polyspace® Server™ for C/C++**
  功能：在计算机集群执行代码验证并发布度量
开发工具的应用指南

- 除了选择开发工具之外，还要提供开发工具的应用指南

- Embedded Coder等工具具有非常详实的用户手册
基于模型的嵌入式软件开发

需求分析
- 模型架构
- 可实现性
- 可测性
- 可追溯
- 可配置

模型建立
- 建模语言
- 建模标准
- 模型复杂度
- 平台化开发

模型验证
- 建模标准
- 模型评审
- 形式化方法验证
- 功能测试

代码实现
- 数据管理
- 等效性测试
- 代码验证
- 代码集成
基于模型的嵌入式软件开发

需求分析
- 模型架构
- 可实现性
- 可测性
- 可追溯
- 可配置

模型建立
- 建模语言
- 建模标准
- 模型复杂度
- 平台化开发

模型验证
- 建模标准
- 模型评审
- 形式化方法验证
- 功能测试

代码实现
- 数据管理
- 等效性测试
- 代码验证
- 代码集成
汽车电子软件的现状和复杂软件开发的困境

- GM汽车上的代码量
- 软件工程师的工作效率

解决复杂软件开发效率低下的途径
- 模块化开发
模块化的原则和目标

- 模块划分的一般原则
  - 从功能上
  - 高内聚
  - 低耦合

- 模块划分的目标
  - 简化设计
  - 便于分工
  - 便于测试
  - 便于后期维护
In order to avoid failures resulting from high complexity, the software architecture design shall exhibit the following properties,

- Modularity;
- Encapsulation; and
- Simplicity.

ISO 26262软件架构设计原则

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>++</td>
</tr>
<tr>
<td>1c</td>
<td>+</td>
</tr>
<tr>
<td>1d</td>
<td>+</td>
</tr>
<tr>
<td>1e</td>
<td>+</td>
</tr>
<tr>
<td>1f</td>
<td>++</td>
</tr>
<tr>
<td>1g</td>
<td>+</td>
</tr>
</tbody>
</table>
软件的层次化结构设计

- 模块如何划分
  - 从功能上划分组件
    - 以发动机为例，分为：点火、进气、油量计算、怠速、巡航等
    - 模型实现上 model reference
  - 对复杂组件进一步划分为单元模块
    - 以发动机的怠速控制为例，分为暖机怠速、闭环速度控制、扭矩请求等单元
    - 模型实现上 model reference
模块化设计和验证——模块划分举例
模块化设计和验证——模块划分举例

系统级
- ACD
  - AirChargeDetermination
- ISC
  - IdleSpeedControl
- IGN
  - IgnitionControl
- ETC
  - ElectronicThrottleControl

组件级
- WarmUpIdle
  - WarmupIdleControl
- CloseLoopIdleSpeed
  - ClosedLoopIdleSpeedControl
- IdleTorqueReq
  - IdleTorqueRequest

单元级
- subsystem1
- subsystem2
- s_Chat

单元模块的设计不建议使用Model Reference.
基于模型的嵌入式软件开发

需求分析
- 模型架构
- 可实现性
- 可测性
- 可追溯
- 可配置

模型建立
- 建模语言
- 建模标准
- 模型复杂度
- 平台化开发

模型验证
- 建模标准
- 模型评审
- 形式化方法验证
- 功能测试

代码实现
- 数据管理
- 等效性测试
- 代码验证
- 代码集成
Simulink建模语言

- 使用建模语言的子集

- Simulink和Stateflow之间的选择
  - 如果算法是复杂的逻辑运算，使用Stateflow；
  - 如果算法主要是数据运算，使用Simulink；

- Stateflow的flow chart和state chart之间的选择
  - 如果算法本质上是计算工作状态或者离散状态，使用state chart；
  - 如果算法本质上是if-then-else结构，使用flow chart或者真值表；
ISO 26262软件单元的设计原则

- 软件单元的设计和实现原则

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a One entry and one exit point in subprograms and functions</td>
<td>++</td>
</tr>
<tr>
<td>1b No dynamic objects or variables, or else online test during their creation</td>
<td>+</td>
</tr>
<tr>
<td>1c Initialization of variables</td>
<td>++</td>
</tr>
<tr>
<td>1d No multiple use of variable names</td>
<td>+</td>
</tr>
<tr>
<td>1e Avoid global variables or else justify their usage</td>
<td>+</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1h No hidden data flow or control flow</td>
<td>+</td>
</tr>
<tr>
<td>1j No recursions</td>
<td>+</td>
</tr>
</tbody>
</table>

- Example: Parallel states should not appear at the top level of a state-chart.
  -- Misra Modeling Guideline
模型复杂度监测

- 对单元模块进行复杂度监测
  - Model advisor

- 圈复杂度
Simulink模型的平台化开发

- Model Variants
  - 通过配置不同的参数选择不同的被引用模型
  - 比如，K_Param == CLASS_A，选择Model_A.mdl；K_Param == CLASS_B，选择Model_B.mdl
  - 支持生成条件编译的代码

- System Variants
基于模型的嵌入式软件开发

需求分析
- 模型架构
- 可实现性
- 可测性
- 可追溯
- 可配置

模型建立
- 建模语言
- 建模标准
- 模型复杂度
- 平台化开发

模型验证
- 建模标准
- 模型评审
- 形式化方法验证
- 功能测试

代码实现
- 数据管理
- 等效性测试
- 代码验证
- 代码集成
ISO 26262定义的软件开发过程
MAAB及相关规范的检查

- Model Advisor实现建模规范检查

- 定制检查集

- 定制检查项
模型评审

- 模型和需求的双向追溯
  - 模型→需求
  - 需求→模型

- Simulink Report Generator生成报告
  - 为非Simulink用户生成报告

- Simulink Report Generator实现不同版本模型比较
使用Simulink Design Verifier检查逻辑错误

- 设定生成测试用例目标为MC/DC 100%覆盖
- 生成测试用例
- 逻辑错误导致无法生成100%覆盖的测试用例，并提示错误逻辑
使用Simulink Design Verifier检查数据错误

- 通过算术运算分析定位错误
  - 数据溢出
  - 被零除
- 证明没有错误的运算
演示Simulink Design Verifier检查错误
单元模块的功能测试

- 仿真测试

- 覆盖率分析

<table>
<thead>
<tr>
<th>Model Hierarchy/Complexity</th>
<th>Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D1</td>
</tr>
<tr>
<td>1. cc_ref1</td>
<td>32 26%</td>
</tr>
<tr>
<td>2. <strong>.. Subsystem</strong></td>
<td>31 28%</td>
</tr>
<tr>
<td>3. <strong>.. TuroLampModeDetermination</strong></td>
<td>24 21%</td>
</tr>
<tr>
<td>4. <strong>.. SF: TuroLampModeDetermination</strong></td>
<td>23 21%</td>
</tr>
<tr>
<td>5. <strong>.. SF: TurnLight</strong></td>
<td>6 0%</td>
</tr>
<tr>
<td>6. <strong>.. SF: Comfort</strong></td>
<td>2 0%</td>
</tr>
<tr>
<td>7. <strong>.. TurnLightSwProcess</strong></td>
<td>7 60%</td>
</tr>
<tr>
<td>8. <strong>.. SF: TurnLightSwProcess</strong></td>
<td>6 60%</td>
</tr>
</tbody>
</table>
模型测试的覆盖率要求

- 对单元软件测试的结构覆盖率要求
  - 覆盖率达到分支覆盖率100%
  - MC/DC要求

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td>1a Statement coverage</td>
<td>++</td>
</tr>
<tr>
<td>1b Branch coverage</td>
<td>+</td>
</tr>
<tr>
<td>1c MC/DC (Modified Conditional/Decision Coverage)</td>
<td>+</td>
</tr>
</tbody>
</table>

- 对软件架构测试的覆盖率要求

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td>1a Function coverage</td>
<td>+</td>
</tr>
<tr>
<td>1b Call coverage</td>
<td>+</td>
</tr>
</tbody>
</table>
模型的集成测试

- 模型的组件级集成测试

- 模型的系统级测试
  - 模型在环测试
  - 快速原型

- 不同组件之间的接口测试

- 不同组件功能上是否冲突
基于模型的嵌入式软件开发

需求分析
- 模型架构
- 可实现性
- 可测性
- 可追溯
- 可配置

模型建立
- 建模语言
- 建模标准
- 模型复杂度
- 平台化开发

模型验证
- 建模标准
- 模型评审
- 形式化方法验证
- 功能测试

代码实现
- 数据管理
- 等效性测试
- 代码验证
- 代码集成
代码生成的前提条件

- 模型经过充分验证

- 模型符合建模标准

- 功能测试覆盖率足够高

- 模型不含有无效逻辑

- 模型不含有数据错误

- 模型经过充分验证
数据对象和数据字典

- 使用数据对象定义数据属性

  Package（包）
  
  Classes（类）
  
  Properties（属性）

- 使用数据字典管理数据对象

```matlab
modelName = 'f14';
dictionaryName = 'myNewDictionary.sldd';
dictionaryObj = Simulink.data.dictionary.create(dictionaryName);
set_param(modelName, 'DataDictionary', dictionaryName);
```
数据字典管理数据

- 按照组件划分进行数据管理
代码生成工具配置

- 软件工具除确定id和版本号之外，还需要确定配置

1. 通过系统目标文件设定回调函数

```matlab
%f--- Configure code generation settings %
%f---
rtwgensettings.DerivedFrom = 'ext.tlc';
rtwgensettings.BuildDirSuffix = '_myarduino';
rtwgensettings.Worker = nil;
rtwgensettings.SelectCallback = 'myarduino_select_callback_handler(hDlg, hSrc)';
END_RTW_OPTIONS
%f---
```

2. 在代码生成设置的回调函数里固化设置

```matlab
% Set the target language to C and disable modification
slConfigUISetVal(hDlg, hSrc, 'TargetLang', 'C');
slConfigUISetEnabled(hDlg, hSrc, 'TargetLang', 0);

% Set the TargetLibSuffix
slConfigUISetVal(hDlg, hSrc, 'TargetLibSuffix', '.a');
```

![代码生成工具配置界面](image)
等效性测试

- SIL测试/PIL测试都是等效性测试
  - 验证生成的代码和用于代码生成的模型具有相同的行为属性
  - PIL除等效性验证之外，还可以用来测量运行时间
- 等效性测试的测试用例
  - 功能测试的测试用例
  - Simulink Design Verifier自动生成

*NOTE 4*  For model-based development, software unit testing can be carried out at the model level followed by back-to-back comparison tests between the model and the object code. The back-to-back comparison tests are used to ensure that the behaviour of the models with regard to the test objectives is equivalent to the automatically-generated code.

- 模型覆盖率和代码覆盖率的比较

9.4.3 The software unit testing methods listed in Table 10 shall be applied to demonstrate that the software units achieve:

d) confidence in the absence of unintended functionality;
代码的集成和集成测试

- 代码集成的两种方式
  - 单元模型的代码生成，代码级别做集成
  - 模型级别集成，然后生成代码

- 软硬件的系统级集成
  - 硬件在环测试
  - 台架测试
  - 实车测试
基于模型的嵌入式软件开发

需求分析
- 模型架构
  - 可实现性
  - 可测性
  - 可追溯
  - 可配置

模型建立
- 建模语言
- 建模标准
- 模型复杂度
  - 平台化开发

模型验证
- 建模标准
- 模型评审
- 形式化方法验证
  - 功能测试

代码实现
- 数据管理
  - 等效性测试
  - 代码验证
  - 代码集成
Change the world by

Accelerating the pace of discovery, innovation, development, and learning

in engineering and science