

**中华人民共和国国家质量监督检验检疫总局**

**中国国家标准化管理委员会**

发布

201X-XX-XX实施

201X-XX-XX发布

农林拖拉机和机械 控制系统有关安全部件 第3部分：软硬件系列开发

**Tractors and machinery for agriculture and forestry—Safety-related parts of control system — Part 3: Series development，hardware and soft ware**

（ISO25119-3:2010,IDT）

（征求意见稿）

**GB/T**XXXX.3—201X/ISO25119-3:2010

中华人民共和国国家标准

**ICS**65.060.01

**B** 90

1. 前言

GB/TXXXX《农林拖拉机和机械控制系统有关安全部件》分为以下4个部分：

——第1部分：设计与开发通则

——第2部分：概念阶段

——第3部分：软硬件系列开发

——第4部分：生产、运行、修改与支持规程

本部分是GB/TXXXX的第3部分。

本部分按照GB/T 1.1—2009给出的规则起草。

1. 本部分使用翻译法等同采用ISO 25119-3:2010《农林拖拉机和机械控制系统与有关安全部件第3部分：软硬件的系列开发》（英文版）。

为便于使用，本部分还对ISO 25119-3:2010做了下列编辑性修改：

1. ——“ISO 25119的本部分”一词改为“本部分”；

——用小数点“.”代替作为小数点的逗号“，”；

1. ——删除 ISO 25119-3:2010的前言；

——本部分由中国机械工业联合会提出。

1. 本部分由全国农业机械标准化技术委员会（SAC/TC201）归口。
2. 本部分起草单位：
3. 本部分主要起草人：

农林拖拉机和机械 控制系统有关安全部件

第3部分：软硬件系列开发

1范围

GB/T XXXX的本部分规定了控制系统有关安全部件（SRP/CS）软硬件开发的通则，适用于农林拖拉机、农用自走机械、农用全挂及半挂机械、农用牵引机械。也可适用于市政机械（如：道路清扫机）。规定了实现安全功能的控制系统有关安全部件（SRP/CS）的特性及类别。

GB/T XXXX的本部分适用于电气/电子/可编程电子系统（E/E/PES）的有关安全部件。由于其涉及机械电子系统，本部分未规定用于特定场合的安全功能或类别。

GB/T XXXX的本部分不适用于非电气/电子/可编程电子系统（E/E/PES）（如：液压、机械及气压）。

2规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/TXXXX.1/ISO 25119-1:2010 农林拖拉机与机械控制系统有关安全部件第1部分：设计与开发通则

GB/TXXXX.2/ISO 25119-3:2010 农林拖拉机与机械控制系统有关安全部件第2部分：概念阶段

GB/TXXXX.4/ISO 25119-4:2010 农林拖拉机与机械控制系统有关安全部件第4部分：生产、运行、修改与支持规程

3术语和定义

GB/T XXXX.1给出的术语和定义适用于本部分。

4 缩略语

下列缩略语适用于本标准。

AgPL 农业性能等级(agricultural performance level)

AgPLr 农业性能等级要求(required agricultural performance level)

CAD 计算机辅助设计(computer-aided design)

Cat 硬件类别(hardware category)

CCF 共因失效 (common-cause failure)

DC 诊断覆盖率(diagnostic coverage)

DCavg 平均诊断覆盖率(average diagnostic coverage)

ECU 电子控制单元(electronic control unit)

ETA 事件树分析(event tree analysis)

E/E/PES 电气/电子/可编程电子系统(electrical/electronic/programmable electronic systems)

EMC 电磁兼容(electromagnetic compatibility)

EUC 可控设备（equipment under control）

FMEA 失效模式及影响分析 （failure mode and effects analysis）

FMECA 失效模式影响及危害度分析（failure mode effects and criticality analysis）

EPROM 可擦除可编程只读存储器（erasable programmable read-only memory）

FSM 功能安全管理（functional safety management）

FTA 故障树分析 （fault tree analysis）

HAZOP 危险和可操作性研究 （hazard and operability study）

HIL 硬件在环(hardware in the loop）

MTTF 平均失效时间（mean time to failure）

MTTFd 平均危险失效时间（mean time to dangerous failure）

MTTFdC 单通道平均危险失效时间(mean time to dangerous failure for each channel)

PES 可编程电子系统(programmable electronic system)

QM 质量措施(quality measures)

RAM 随机存取存储器(random-access memory)

SOP 开始生产(start of production)

SRL 软件要求等级(software requirement level)

SRP 有关安全部件(safety-related parts)

SRP/CS 控制系统有关安全部件(safety-related parts of control systems)

SRS 有关安全系统(safety-related system)

UML 统一建模语言(unified modeling language )

# 5系统设计

## 5.1 目的

目的是为整个有关安全系统完全满足安全要求的设计定义一个开发流程。

## 5.2概述

安全要求构成了达到和确保功能安全的所有要求。在安全寿命周期内，对安全要求按照层次等级进行了更详尽的阐述和规定。图1列出了不同等级的安全要求。安全要求开发的完整表述见5.4。推荐使用合适的要求管理工具以支持安全要求的管理。

## 5.3前提条件

在开始系统设计之前，应先定义有关安全功能要求、应用和操作环境。

## 5.4 要求

## 5.4.1安全要求构建

功能安全概念规定实现安全目标的有关安全系统的基本功能。系统架构功能安全的基本配置由技术安全概念以技术安全要求的形式规定。系统架构由软件和硬件组成。

硬件安全要求提炼并固化技术安全概念的要求。第6条详细描述了如何规定硬件要求。

软件安全要求源于技术安全概念和底层硬件的要求。应考虑第7条定义的软件要求。

本条规定了系统设计过程中应用于安全概念要求规范的方法，从而给无错系统设计提供了基础。



a 用斜线分开的两个数字的第一个数字代表GB/TXXXX的各自部分，第二个数字代表文档中的条数：如2/6是指GB/TXXXX.2/第6条，3/5是指GB/TXXXX.3/第5条,以此类推。

图1 安全要求构建

## 5.4.2 功能安全概念

## 5.4.2.1 功能安全概念通用要求

安全功能通常是在系统风险分析阶段标识，系统的功能安全概念文档包括功能安全要求。

每个安全概念要求的执行应考虑以下内容。

——可行性

在罗列功能安全要求时，应考虑要求的可行性及约束条件，如可用技术以及资金和时间资源。负责执行人员应理解和接受技术安全要求。

——无歧义性

功能安全要求应尽可能准确无误的表述。

注：功能安全要求应准确无误的表述，它只允许给预期读者一种解释。

——一致性

功能安全要求不应自相矛盾（内部一致性），也不应与其他要求相抵触（外部一致性）。有必要对不同的要求进行分析和比较，以确保外部一致性。这是要求管理任务。

——完整性

功能安全概念应考虑所有有关的规范、标准和法规。

功能安全概念应考虑来自于按照GB/TXXXX.2的风险分析的所有相关安全目标。

在系统设计期间，功能安全概念的完整性需反复补充。为保证完整性：

1）应指定功能安全概念的版本及有关的基本来源的版本；

2）应满足更改管理要求（见GB/TXXXX.2第10条），为此，应制定并明确表述功能安全要求，以支持更改过程；

3）应复查功能安全要求（见GB/TXXXX.4第6条）。

## 功能安全概念应考虑寿命周期的所有阶段（包括生产、客户运行、维护和报废）。

## 5.4.2.2功能安全概念规范

本条介绍了功能安全概念中要求的规定信息。功能安全概念可来自于风险分析期间对机器失效场景的评估。

每个失效场景描述应包括以下部分：

——环境状况（在结冰的道路上行驶，上坡，下坡，天气等）；

——机器情况（发动机运转，档位接合，停止等）；

——得到的AgPL；

——安全状态描述（发动机停机，阀关闭，传动系驻车，在性能降低下的功能保持等）。

## 5.4.3 技术安全概念

## 5.4.3.1 技术安全概念通用要求

技术安全概念文档包括系统的技术安全要求。

每个技术安全概念应与更高级别的安全要求关联（例如交叉引用），这可能是：

——其他技术安全要求；

——功能安全要求；

——安全目标和目的。

注：使用合适的要求管理工具可极大地促进可追溯性。

正如功能安全概念，每个技术安全概念要求的执行将考虑到可行性、无歧义性、一致性和完整性。

——可行性

在罗列技术安全要求时，应考虑到约束条件下要求的可行性，如可用技术以及资金和时间资源。负责执行人员应理解和接受技术安全要求。

——无歧义性

技术安全要求的制定应尽可能准确、无歧义。

注：技术安全要求的表述应准确无歧义，它只允许给预期读者一种解释。

——一致性

技术安全要求不应自相矛盾（内部一致性），也应不和其他要求相抵触（外部一致性）。

有必要对要求进行分析及对不同的要求进行比较，以确保外部一致性。这是要求管理任务。

——完整性

技术安全概念应考虑以下几点：

1）所有安全目标和功能安全要求；

2）所有相关规范、标准和法规；

3）从安全分析工具（FMEA，FTA等）得到的相关结果；安全分析提供了系统开发过程中的安全技术概念的迭代支持。

在系统设计期间，技术安全概念的完整性反复补充。以确保完整性：

4）应当指定技术安全概念的版本及有关基本来源的版本；

5）应满足来自更改管理的要求（见GB/TXXXX.2第10条），为此，应制定并明确表述技术安全要求以支持修改过程；

6）应复查技术安全要求（见GB/TXXXX.4第6条）。

技术安全概念应考虑寿命周期的所有阶段（包括生产，客户运行，维护和报废）

5.4.3.2技术安全概念规范

## 5.4.3.2.1概述

技术安全概念应包括足以满足观察单元设计的软硬件设计安全要求，并应按5.4.3.1确定。

## 5.4.3.2.2状态和时间

应为所有相关操作状态规定观察单元的行为、模块以及接口，包括

——启动；

——正常操作；

——停机；

——复位后重启；

——合理且可预见的不常用操作状态（例如降级的操作状态）。

特别地，应精确描述失效行为及要求的反应，也可以包含附加的应急操作功能。

技术安全概念应为每个功能安全要求、安全状态转换及安全状态保持规定一个安全状态。特别地，应规定是否立即关闭观察单元即为达到安全状态，或者是否只有通过控制停机才可达到安全状态。

技术安全概念应为每个功能安全要求规定从发生错误开始至进入安全状态的最大时间（响应时间）。在技术安全概念中应规定所有子系统和子功能的响应时间。

如果直接停机不能达到安全状态，则应为所有子系统和子功能的应急操作功能持续时间定义一个时间值。在技术安全概念中应记录此应急操作功能。

## 5.4.3.2.3安全架构、接口和边界条件

应描述安全架构及其子模块。特别地，应规定其技术措施。技术安全概念应分别描述以下模块（适用的范围）：

——传感器系统，对每个物理参数分别记录；

——混合的数字和模拟输入输出单元；

——处理，对每个算术单元/离散逻辑单元分别处理；

——执行器系统，对每个执行器分别执行；

——显示器，对每个指示单元分别显示；

——各种机电组件；

——模块之间的信号传输；

——与系统外部观察单元之间的信号传输；

——电源供电。

应规定观察单元模块间的接口、与机器的其他系统及功能间的接口以及用户接口。

应规定检测单元的限制及边界条件，包括在整个寿命周期的所有阶段的所有环境条件下的极值。

## 6硬件

## 6.1目的

为有关安全控制系统定义可接受的硬件架构。

## 6.2概述

改进控制系统有关安全部件的硬件架构可以提供避免故障、检测故障或容错的措施。实际操作措施包括冗余、多样处理和监控。

一般情况下，应考虑以下故障判据。

——由于一个故障的原因而导致组件失效，则首次出现的故障和所有后续故障为单一故障。

——由于一个共同原因而导致两个或多个单独故障，则为单一故障（即共因失效）。

——通常不可能同时出现两个独立故障。

## 6.3前提条件

前提条件是为硬件实现的每个安全功能确定的AgPLr。

## 6.4要求

硬件开发过程应在系统级别的安全功能及相关联要求标识后开始（见图2）。

硬件安全分析用于标识每个系统安全功能的性能等级（AgPLr）（见GB/TXXXX.2）。

设计者应将功能分组到与MTTFdC、DC和CCF关联的适当架构中（硬件类别）。

为便于开发可将系统分解成各种子系统。

在开发周期的每个阶段都应验证。

硬件系统架构的设计规程如下。

a）选择一个硬件类别（见GB/TXXXX.2，附录A）。

b）标识部件运行环境和应力等级。

c）选择组件。

d）MTTFdC满足要求等级的计算和验证（见GB/TXXXX.2，附录B）。

e）DC满足要求等级的确定和验证（见GB/TXXXX.2，附录C）。

f）考虑CCF（见GB/TXXXX.2，附录D）。

g）考虑系统性失效（见GB/TXXXX.2，附录E）。

h）考虑其他安全功能（见GB/TXXXX.2，附录F）。

注：上述步骤可要求迭代。

## 6.5硬件类别

控制系统有关安全部件的设计应该与GB/TXXXX.2中附录A所指定的五种分类中的一种或多种的要求相一致。

如果安全功能是由多硬件类别集成实现的，导致的安全功能的AgPL受限于全部硬件类别：MTTFdC，DC，SRL，CCF等。（见图3）。

总的SRL的确定，见7.3.4.7。



a 用斜线分开的两个数字的第一个数字代表GB/TXXXX的各自部分，第二个数字代表文档中的条数。

图2 V模式硬件开发



图3 类别2最大AgPL的集成系统

## 6.6工作产品

以下工作产品适用于硬件设计：

a）硬件安全确认测试计划；

b）硬件安全确认测试规范；

c）硬件安全确认测试结果；

d）硬件系统集成测试计划/硬件子系统测试计划；

e）硬件系统集成测试规范；

f）硬件系统集成测试结果/硬件子系统测试结果。

## 7 软件

## 7.1 软件开发计划

## 7.1.1 目的

是确定和计划软件开发的各阶段。包括本条款描述的软件开发过程以及GB/TXXXX.4第10条描述的必要支持过程。

## 7.1.2概述

图4详述了软件开发的过程。本条款以下的段落和表格详细解释了图中的各方框中的内容。

应根据要求的SRL选择适当的技术或措施。由于影响软件安全完整性的因素众多，故不可能存在一种技术和措施组合的算法对于任意特定应用均正确。对于特殊应用，技术或措施的适当组合应在安全计划制定期间予以阐述，并按照7.1.4的要求选择适当的技术或措施。

## 7.1.3 前提条件

这个阶段的前提条件是：

——每个安全功能要实现由AgPLr确定的要求的SRL；

——项目计划（包括系统开发计划）；

——系统验证计划；

——技术安全概念；

——系统设计规范；

——安全计划。

## 7.1.4 要求

## 7.1.4.1 阶段确定

对于软件开发过程，应确定要执行的软件开发阶段（见图4）。要充分考虑项目的大小和复杂程度。可参照图4无修改执行各阶段，或者如果生成了合并阶段的全部工作产品，则各阶段可以合并。

注：如果使用的方法难以明显地区别某些阶段，则可以将这些阶段合并。例如，软件架构和软件执行的设计可以连续地用同一个计算机辅助开发工具开发，则可以将其共同归类为基于模型的开发过程。

可以通过分配活动和任务增加其他阶段。

例子：在电子控制单元的安全确认之前可以插入一个数据应用阶段。ECU的安全确认可以根据其功能分布的不同分别进行，例如特定ECU测试或组合控制网络测试。可以在部件系统的测试地点或在试验车辆内进行。

## 7.1.4.2 过程灵活性

活动和任务可以在不同阶段之间移动。

## 7.1.4.3 过程时间表

应制定一个时间表，说明软件开发不同阶段和包括机器层级集成步骤在内的产品开发过程间关系。

## 7.1.4.4 适用性

在软件安全要求规范按照表1完成后，应确定各软件安全要求应用的集成步骤。

## 7.1.4.5 支持程序

支持程序应作为软件开发过程一部分来计划和实现：

a）工作产品应按GB/TXXXX.4，第12条的要求文档化；

b）软件更改按GB/TXXXX.4，第10条的要求进行；

c）工作产品应受配置管理程序的约束。

注: 以上的支持程序b），包括对由于更改造成的软件的不同分支的处理策略，包括合并这些分支。

## 7.1.4.6软件开发阶段

在软件开发的每个阶段，选择合适的开发方法、措施和相应工具，以及方法、措施和工具执行的指导方针，均应按照SRL进行。

这些选择应根据应用领域进行适当调整，且都应在每个开发阶段的启动阶段进行。

选择方法和措施时，应注意，除手动编码外，可以应用那些源代码或目标代码可以从模型自动生成的基于模型的开发。

注：协同方法和措施的选择有可能降低软件开发的复杂程度。

## 7.1.4.7 表格使用

对每个开发方法及措施，表1到表6用符号“+”或“o”给四个SRL分别进行标记。

+方法或措施应用于该SRL，除非有不适用的理由，在计划阶段应记录这些理由；

o不推荐或不允许这些方法或措施使用于该SRL。

在表格中，“o”也许出现在“+”右边。这表明对相同SRL可用更为严格的措施或技术。

应选择与各自SRL相一致的方法和措施。可选或等效的方法和措施用数字后的字母来标识。但至少要选择一项标记为“+”的可选择或等效的方法和措施。

如果一项特别的方法或措施未在表格中列出，则并非不允许使用该方法或措施。如果一项未列出的方法或措施替换表格里的方法或措施，则其应具有相等或更高值。

## 7.1.5 工作产品

适用于该阶段的工作产品是来自于从7.1.4.2到7.1.4.4的软件项目计划（参见GB/TXXXX.4第6条）。

## 7.2 软件安全要求规范

## 7.2.1目的

第一个目的是从技术安全要求中获得包括SRL在内的软件安全要求。

第二个目的是验证软件安全要求与技术安全概念的一致性。



图4 V模式软件开发

## 7.2.2 概述

软件安全要求规范应从系统的技术安全概念要求中导出，并标记为软件安全要求。应至少考虑以下几个方面：

a）软件中技术安全概念的适当执行；

b）系统配置和架构；

c） E/E/PES系统硬件的设计；

d）安全功能响应时间；

e）外部接口，例如通信；

f）影响软件的物理要求和环境条件；

g）安全软件修改要求。

注：系统的软硬件开发之间往往是迭代进行的。在进一步指定和细化软件安全要求和软件架构过程中，也许会影响硬件架构。因此，硬件和软件开发之间的紧密协调是必要的。

## 7.2.3 前提条件

下列是软件安全要求规范的前提条件：

——符合7.1.4.2至7.1.4.4的软件项目计划；

——符合5.4.3的技术安全概念；

——符合6.5的硬件类别。

## 7.2.4 要求

## 7.2.4.1 软件安全要求规范说明方法

软件安全要求规范与表1一致。

表1 软件安全要求规范

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 技术/措施a | 子条款 | SRL=B | SRL=1 | SRL=2 | SRL=3 |
| 1 自然语言的要求规范 | 7.2.4.1.1 | + | + | + | + |
| 2a 非形式化设计方法a | 7.2.4.1.2 | + | + | o | o |
| 2b 半形式化设计方法 | 7.2.4.1.3 | o | o | + | + |
| 2c 形式化设计方法 | 7.2.4.1.4 | o | o | + | + |
| 3 计算机辅助规范工具 | 7.2.4.1.5 | o | o | + | + |
| 4a 软件安全要求检查a | GB/TXXXX.1，3.28 | + | + | + | + |
| 4b 软件安全要求走查 | GB/TXXXX.1，3.56 | + | + | o | o |
| 见7.1.4.7关于表格使用说明 | | | | | |
| a 依照SRL选择适当的技术/措施。可选的或等效的技术/措施用数字后面的字母表示，仅需满足可选或等效的技术/措施中的一个。 | | | | | |
| 注1：在2c中概述了拥有完整语法定义和数学语义定义的建模方法。形式化方法允许用于形式化验证和自动测试用例的生成。示例包括与形式化验证相联系的状态机。  注2：在2b中概述了拥有不含运算完整语法定义和语义定义的建模方法。示例包括结构分析/设计和图形建模方法，如UML类图或方框图。  注3：在带有代码生成的基于模型的开发情况下，软件架构设计的方法和措施必须应用到功能模型，这是代码生成的基础。 | | | | | |

## 7.2.4.1.1 自然语言的要求规范

## 7.2.4.1.1.1 目标

在自然语言中应引入规范要求（即口语和书面语）。

## 7.2.4.1.1.2 描述

软件安全要求规范应包括用自然语言对问题的描述，如果需要，也可用非形式化方法，如图形和图表。

## 7.2.4.1.2 非形式化设计方法

## 7.2.4.1.2.1 目标

应使用自然语言表示一个完整概念。

## 7.2.4.1.2.2 描述

在系统开发的某些阶段，非形式化方法提供了一种系统描述的开发方法，即规格、设计或编码，可通过自然语言、图形、图表等典型方法。

## 7.2.4.1.3 半形式化设计方法

## 7.2.4.1.3.1 目标

半形式化设计方法应无歧义并一致表示概念、规格或者设计，这样可以检测出某些差错和遗漏。

## 7.2.4.1.3.2 描述

在某些开发阶段，软件开发的半形式化方法应用于提供系统描述的开发方法，即规格、设计或者编码。

例：数据流图、有限状态机/状态转换图。

在某些情况下，描述可用机器分析或使用动画来演示系统行为的不同方面。动画可提高系统满足要求的可信度。

## 7.2.4.1.4 形式化设计方法

## 7.2.4.1.4.1 目标

基于数学方法的软件开发应包括形式化设计和形式化编码技术。

## 7.2.4.1.4.2 描述

形式化方法应在某些阶段提供规范、设计或执行的系统描述的开发方法。产生的描述用严格的符号表示，应用数学分析的方法检测不一致性或不正确性的各种类型。此外，在某些情况下，描述应由机器进行分析，其严格程度类似于编译器对源程序进行语法检查；或使用动画来显示所描述系统行为的不同方面。动画可增强系统满足实际要求和形式化规定要求的可信度，这是由于它提高了人们对特定行为的认知。

形式化方法通常应提供标记（通常是使用离散数学的某种形式）、按照该标记进行描述的技术以及检查描述不同正确属性的各种分析形式。

## 7.2.4.1.5 计算机辅助规范工具

7.2.4.1.5.1 目标

为便于歧义性和完整性的自动检测，应使用形式化规范技术。

7.2.4.1.5.2 描述

该技术应以数据库的形式产生规范，可自动检查以评估其一致性和完整性。规范工具可以为使用者动画显示规定系统的各个方面。技术通常不仅支持规范的创建，也可用于项目寿命周期的设计及其他阶段。

7.2.4.2 非有关安全功能

如果附加于安全功能的功能由E/E/PES实现，则应详细说明或对其规范进行引用。

如果要求规范包括功能要求和安全要求两个方面，则后者应明确标识。

7.2.4.3 详细程度

软件安全要求规范应包含足够的细节以实现软件安全功能。

7.2.4.4 一致性

软件安全要求应是可追溯的，且和系统安全要求规范及系统架构具有一致性。

## 7.2.4.5 软硬件的互相依赖性

如果软硬件之间具有相关性，则软件安全规范应说明软硬件之间有关安全依赖性。

## 7.2.4.6 软件安全要求规范

如果相关，软件安全要求规范应描述下列软件安全要求：

——使系统能够实现或保持安全状态的功能。

——ECU、传感器、执行器和通信系统中相关故障检测、指示和处理的功能。

——软件自身的相关故障检测、指示和处理的功能（软件自监督）。

注1：包括操作系统中软件自监控和应用软件中针对系统故障检测软件的特定应用自监控。

——安全功能在线和离线测试相关的功能。

注2：在车辆启动和运行期间可实现自检。

注3：这里特指客户服务或其他E/E/PES系统对安全功能的检测能力。

——允许软件修改可安全执行的功能。

——非有关安全的功能接口。

——性能和响应时间。

——电子控制单元软件与硬件之间的接口

注4：接口也包括编程和配置。

软件安全完整性要求：

——上述每个功能的SRL

——软件安全要求的软件安全确认的接受准则。

## 7.2.4.7 软件安全要求验证

应检查软件安全要求，以确定是否符合7.2.4.1至7.2.4.6给出的要求。也应检查软件安全要求与技术安全概念的一致性。软件开发人员也应参加验证工作。确认方法可以为检查或走查（定义在GB/TXXXX.1中）两者之一。

## 7.2.5 工作产品

下列工作产品应用于本阶段：

a）依据7.2.4.1和7.2.4.3至7.2.4.6的软件安全要求规范；

b）依据7.2.4.2的非有关安全软件要求规范；

c）依据7.2.4.6的软件安全要求接受准则；

d）从7.2.4.7得到的关于软件安全要求规范的验证报告。

## 7.3 软件架构和设计

## 7.3.1 目的

软件架构的目的是用软件组件来实现和组织全部软件要求。并应确保他们的软件组件能够实现所有分配给他们的软件安全要求。

## 7.3.2 概述

软件架构是所有软件部件及其层次结构相互关系的表述。应描述静态方面，如所有软件组件的接口和数据路径，也应描述动态方面，如过程序列和时间行为。

## 7.3.3 前提条件

仅在软件安全要求规范达到足够成熟度后，软件架构才可开始。

## 7.3.4 要求

## 7.3.4.1 软件架构和设计方法

软件架构和设计的开发应与表2一致。

表2 软件架构和设计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 技术/措施a | 子条款 | SRL=B | SRL=1 | SRL=2 | SRL=3 |
| 1a 非形式化设计方法a | 7.2.4.1.2 | + | + | o | o |
| 1b 半形式化设计方法 | 7.2.4.1.3 | o | o | + | + |
| 1c 形式化设计方法 | 7.2.4.1.4 | o | o | + | + |
| 2 计算机辅助规范工具 | 7.2.4.1.5 | o | o | + | + |
| 3 自底向上的失效分析 | 7.3.4.1.1 | o | o | o | + |
| 4 自顶向下的失效分析 | 7.3.4.1.2 | o | o | + | + |
| 5a软件架构检查 | GB/TXXXX.1，3.28 | + | + | + | + |
| 5b 软件架构走查 | GB/TXXXX.1，3.56 | + | + | o | o |
| 本表格及其他表格的使用说明见7.1.4.7 | | | | | |
| a 依照SRL选择适当的技术/措施。可选或等效的技术/措施用数字后的字母表示，只能有一个可选或等效的技术/措施供选择。 | | | | | |

## 7.3.4.1.1失效分析——自底向上方法

## 7.3.4.1.1.1 目标

自底向上方法支持对导致危险或严重后果的事件或事件组合的分析。

## 7.3.4.1.1.2 描述

应通过树形路径从直接引起危险和严重后果（“底事件”）的事件开始分析。用逻辑运算符（和、或等）来描述组合原因。间接原因用同样的方法分析，返回到基本事件后分析停止。该方法是图形化的，且用标准符号组绘制故障树。

例如FMEA、HAZOP或FMECA方法。

## 7.3.4.1.2 故障分析——自顶向下方法

## 7.3.4.1.2.1 目标

在设计和运行中，用自顶向下方法来排列由于单点失效而导致伤害、损坏或系统退化的组件的危害度，以确定哪些部件需要特别关注和必要的控制措施。

## 7.3.4.1.2.2 描述

危害度可用多种方法排列。危险系数是多个参数的函数，且大多数必须被测定。确定危害系数的一种很简单的方法是将可能发生损害程度与组件失效概率相乘；这种方法类似于简单的风险因素评估。

注：这些技术主要用于硬件系统分析，不过也可以尝试应用于软件失效分析，例如FTA、ETA和“因果关系”图表。

## 7.3.4.2 设计方法特性

所选的设计方法具有的特性应支持以下：

1. 抽象化、模块化、封装和其他使复杂性可管理的特性，
2. 描述：

——功能性；

——组件间的信息流；

——过程控制和有关时间信息；

——时间限制；

——并发过程（如果相关）；

——数据结构及其特性；

——设计中的假设及其关系，

c）开发人员及其他相关者的理解，

d）软件修改的能力，

e）检验和确认。

## 7.3.4.3 软件架构构成

应开发基于软件安全要求分层次描述的全部有关安全软件组件的软件架构。

注：在顶级软件架构中，软件通常分为基础软件和应用软件。

## 7.3.4.4 详细程度

软件架构的结构应分层至软件模块最低级。

当开发软件架构时，有关安全组件的范围应保持尽可能小。

## 7.3.4.5 软件架构的可追溯性

在软件架构和软件安全要求之间应该实现双向可追溯性。

## 7.3.4.6 软件架构验证

应验证软件架构。应检查设计的架构是否满足软件安全要求。软件开发人员应参与验证活动。验证方法可以为检查或走查（定义在GB/TXXXX.1中）两者之一。

## 7.3.4.7有关安全软件组件组合

如果嵌入式软件必须实现不同的SRL软件组件或有关安全及非有关安全软件组件，那么总的SRL将受限于最低的SRL部件，除非能证明软件组件之间有足够的独立性（见附录B）。

## 7.3.5 工作产品

下列工作产品适用于这个阶段：

1. 依照7.3.4.1 至7.3.4.5的软件架构；
2. 来自7.3.4.6的软件架构验证报告；

## 7.4 软件模块设计和实现

## 7.4.1 目的

第一个目的是详细指定由软件架构规定的有关安全软件模块的行为。

第二个目的是生成一个可读的、可测试的、可维护的及可翻译成目标代码的源代码（代码，模型等）。

第三个目的是验证软件架构已被完全和正确的实现。

## 7.4.2 概述

## 7.4.3 前提条件

以下是软件模块设计及实现的前提条件：

——软件项目计划（见7.1.4.2至7.1.4.4）；

——软件要求[见7.2.5a）和b）]；

——软件架构（见7.3.4.1至7.3.4.5）；

——软件验证计划（见GB/TXXXX.4，第6条）。

## 7.4.4 要求

## 7.4.4.1 软件模块设计和实现方法

软件应按表3设计和开发。

表3 软件设计和开发——支持工具和编程语言

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 技术/措施a | 子条款 | SRL=B | SRL=1 | SRL=2 | SRL=3 |
| 1 工具和编程语言 | | | | | |
| 1.1 合适的编程语言 | 7.4.4.1.1 | + | + | + | + |
| 1.2 强类型编程语言 | 7.4.4.1.2 | o | + | + | + |
| 1.3 语言子集 | 7.4.4.1.3 | o | + | + | + |
| 1.4 工具和编译器：增强使用的可信度 | 7.4.4.1.4 | o | + | + | + |
| 1.5 可信的/经过验证的软件模块和组件的使用（如果可用） | 7.4.4.1.5 | o | o | + | + |
| 2 设计方法 | | | | | |
| 2.1a 非形式化设计方法a  2.1b 半形式化设计方法  2.1c 形式化设计方法 | 7.2.4.1.2  7.2.4.1.3  7.2.4.1.4 | +  o  o | +  o  o | o  +  + | o  +  + |
| 2.2 防御性编程 | 7.4.4.1.6 | o | o | o | + |
| 2.3 结构化编程 | 7.4.4.1.7 | o | + | o | o |
| 2.4 模块化方法 | 7.4.4.1.8 | o | o | o | + |
| 2.5可信的/经过验证的软件模块和组件库 | 7.4.4.1.9 | + | + | + | + |
| 2.6 计算机辅助设计工具 | 7.4.4.1.10 | o | o | o | + |
| 3 设计和编码标准 | | | | | |
| 3.1 编码标准的使用 | 7.4.4.1.11 | o | o | + | + |
| 3.2 无动态变量或对象 | 7.4.4.1.12 | o | o | o | + |
| 3.3 中断的使用限制 | 7.4.4.1.13 | o | o | o | + |
| 3.4指针的定义使用 | 7.4.4.1.14 | o | o | o | + |
| 3.5 递归的限制使用 | 7.4.4.1.15 | o | o | o | + |
| 4 设计和编码验证 | | | | | |
| 4a 软件设计和/或源代码检查 | GB/TXXXX.1，3.28 | + | + | + | + |
| 4b 软件设计和/或源代码走查 | GB/TXXXX.1，3.56 | + | + | o | o |
| 见7.1.4.7关于表格使用说明 | | | | | |
| a 依照SRL选择适当的技术/措施。可选或等效的技术/措施用数字后的字母表示。仅需满足可选或等效的技术/措施中的一个。 | | | | | |

## 7.4.4.1.1 合适的编程语言

## 7.4.4.1.1.1 目标

目标是选择一个尽可能支持GB/TXXXX要求的编程语言，尤其是防御式编程、结构化编程和可能指定的编程语言。编程语言的选择应易于验证代码，并有利于程序的开发、验证和维护。

## 7.4.4.1.1.2 描述

应充分和无歧义定义语言。语言应是面向用户或问题的，而不是面向机器的处理器/平台。应首选广泛使用的语言子集作为特殊目的语言。

除了已经引用的特征外，语言还应具有：

——块结构，

——转换时间校验，

——运行时的类型和数组边界检查。

语言应该支持：

——小的及易处理软件模块的使用，

——特定软件模块中访问数据的限定，

——变量子范围的定义，

——错误限制结构的其他类型。

如果系统的安全运行是依赖于实时约束，则语言还应提供异常/中断处理。一个好的语言要有合适的编译器、已存在的适当的软件模块库、调试器以及版本控制和开发的工具来支持。在GB/TXXXX的本部分制定时，未明确面向对象的语言是否是首选程序语言。

应避免导致验证困难的以下特征：

a）除子程序调用外的无条件跳转；

b）递归；

c）指针、堆栈或任意类型的动态变量或对象；

d）源代码级的中断处理；

e）循环、程序块或子程序的多个入口或出口；

f）隐式变量初始化或声明；

g）变体记录和等价记录；

h）程序参数。

低级语言尤其是汇编语言，由于面向机器的处理器/平台的性质会引起问题。一个理想语言的属性，其程序设计和使用结果的执行是可预见的。对于一个适当定义的可编程语言，其含有的子集可确保的程序执行是可预见的。尽管许多静态约束有助于确保可预见执行，而该子集不能（通常）静态确定。通常，需要证明数组索引在边界内，且不会引起数值溢出。

## 7.4.4.1.2 强类型编程语言或检验方针

## 7.4.4.1.2.1 目标

通过使用强类型语言或编程实践以降低故障率，该语言或编程实践允许使用编译器或静态分析工具进行高级检验。

## 7.4.4.1.2.2描述

当强类型的编程语言被编译或静态分析时，需检验变量类型如何使用（例如，过程调用和外部数据访问）。如果任何用法不符合预定义规则，则编译失败并产生一个错误消息。

注：这种语言通常允许源自基本语言数据类型定义的用户自定义数据类型（如整数类型，实数类型）。这些类型的用法与基本类型的用法完全相同。强制严格检验以确保正确类型的使用。即使分别进行的编译单元，仍需对整个程序进行强制校验。即使对编译的软件模块进行分别引用，仍需校验以确保程序参数的数值和类型相匹配。

## 7.4.4.1.3语言子集

## 7.4.4.1.3.1目标

语言子集的使用应减少引起编程错误的概率和增加遗留错误的检测概率。

## 7.4.4.1.3.2描述

应检查编程语言，以确定容易出错或难以分析的编程结构（例如使用静态分析方法）。然后应排除这些编程结构，定义一个语言子集。此外，应记录所使用的语言子集中结构安全的原因。

## 7.4.4.1.4工具和编译器——增加使用的可信度

## 7.4.4.1.4.1目标

为避免在软件包的开发、验证及维护过程中编译器故障而造成的困难，应使用经实践检验的工具和编译器。

## 7.4.4.1.4.2描述

应使用在之前的众多项目中已证明性能良好的编译器。应避免使用无运行经验或有已知严重故障的编译器，除非能证明编译器可正确运行。如果编译器有小缺陷，则应记录相关的语言结构，并在有关安全项目中尽量避免。

注1：该描述基于大量项目经验。已证明不成熟的编译器对软件开发非常不利，一般不适用于有关安全软件的开发。

注2：目前还没有方法能够证明所有工具或编译器部件的正确性。

## 7.4.4.1.5 可信的/经过验证的软件模块和组件的使用（如果可用）

## 7.4.4.1.5.1目标

应使用可信的/经过验证的软件模块和组件，以避免在每个新的应用中对软件模块和硬件组件进行大量的重新验证和重新设计。允许开发者使用尚未经过正式或严格验证，但已有大量运行历史的设计。

## 7.4.4.1.5.2 描述

该措施应验证软件模块和部件不受系统性设计故障和/或运行失效的影响。只有在极少数情况下，使用可信的软件模块和组件（即实践证明的）作为充分保证达到所需SRL的唯一措施。对于具有许多可能功能的复杂组件（如操作系统），重要的是确定哪些功能在实际中被充分证明在使用。例如，用自检程序检测硬件故障，如果在运行周期内没有发生硬件失效，则不能证明自检程序已被使用。

如果已经验证满足要求的SRL或者符合下列准则，则部件或软件模块是充分可信的：

——无更改的规范；

——不同应用的系统；

——至少有一年的运行历史；

——所有软件模块的运行经验应与已知需求配置有关，以确保增加的运行经验可以提高对软件模块行为的认识；

——无有关安全失效。

注：某些失效在一种环境中或许不是有关安全的，但在另一种环境中却是有关安全的，反之亦然。

为验证部件或软件模块是否符合上述准则，应记录以下内容：

a）每个系统及其组件的准确标识，包括版本号（软硬件）；

b）用户和应用时间的标识；

c）运行时间；

d）用户应用系统和应用案例选择的程序；

e）检测和记录失效、消除故障的程序。

## 7.4.4.1.6防御性编程

## 7.4.4.1.6.1目标

应使用防御性编程编制检测程序，该程序用于检测在执行期间出现的异常控制流、数据流或数值，并以一个预定和可接受方式做出反应。

## 7.4.4.1.6.2 描述

在编程期间可用多种技术检测控制或数据异常。所应用的技术应系统的应用到系统编程的全过程中，以减少错误数据处理的可能性。有两个重叠的防御技术领域。设计的固有错误安全软件用于容忍设计本身的缺陷。这些缺陷可能是由设计或编码错误，或错误要求造成的。包括以下技术：

——变量范围检测；

——数值合理性检测；

——程序入口参数的类型、维数和范围的检测。

第一套防御技术是依照程序功能和变量的物理意义确保程序使用数据的合理性。

应将只读和可读写参数分开，并对其访问进行检查。函数应将所有参数作为只读参数。文字常量不应被写访问。这有助于检测变量的意外覆盖或误用。设计容错软件的目的是“预计”本身环境中的失效，或在正常或预计条件之外使用，并以一个预定的方式运行。包括以下技术：

——利用物理意义检查输入变量和中间变量的合理性；

——最好通过直接观察相关系统状态变化，检查输出变量的效果；

——通过软件配置检查，包括预期硬件的存在性和可访问性、以及软件本身的完整性，尤其重要的是维护过程之后保持软件的完整性。

一些防御性编程技术，如控制流序列检查，也可处理外部失效。

## 7.4.4.1.7结构化编程

## 7.4.4.1.7.1目标

结构化编程用于设计及执行程序，以便在程序不执行时可以进行分析。

## 7.4.4.1.7.2 描述

为尽量减少结构的复杂性，应采取以下措施：

a）将程序分成适当小的软件模块，以确保其尽可能隔离，且所有的相互关系是明确的。

b）使用结构化设计方法编写软件模块的控制流，例如顺序、迭代和选择。

c）尽可能减少通过软件模块的可能通道数量，并尽可能使输入和输出参数之间的关系简单化。

d）避免复杂分支。尤其要避免在高级语言中的无条件跳转（go-to语句）。

e）在可能的情况下，将循环约束条件和分支关联到输入参数。

f）避免使用复杂的计算作为分支和循环判断的基础。与其他（据称）更有效率的特性相比，除非效率绝对优先，则应优先使用支持上述方法的编程语言特性。

## 7.4.4.1.8模块化方法

## 7.4.4.1.8.1 目标

为了管理系统的复杂性，应使用模块化方法将软件系统分解为小的易于理解的模块。

## 7.4.4.1.8.2 描述

模块化方法（模块化）是以软件项目的设计、编码和维护阶段的多个规则为前提的。这些规则根据使用设计方法的不同而改变。以下方法适用于GB/TXXXX的本部分：

——软件模块应执行单一的明确定义的任务或功能。

——软件模块之间的联系应受限和严格定义；一个软件模块必须具有较强的相关性。

——应建立子程序集，提供多级的软件模块。

——软件模块的大小应限制为一特定值，通常为2到4个屏幕大小。

——软件模块应有单入口和单出口。

——软件模块和其他软件模块间应通过接口通信。全局和共同变量的使用应结构良好，访问可控，在每个实例中的使用应合理。

——所有软件模块接口都应完全记录。

——任何软件模块的接口应只包含这个功能所需的参数。

## 7.4.4.1.9 可信的/经过验证的软件模块和组件库

## 7.4.4.1.9.1目标

应使用可信的/经过验证的软件模块和组件库，以避免在每个新的应用中进行大量的重新验证和重新设计。允许开发者使用尚未经过正式或严格验证，但已有运行历史的设计。

## 7.4.4.1.9.2 描述

为了进行良好的设计和构建，E/E/PES应由硬件组件、软件组件和明确区分的软件模块组成，软件模块以明确的特定方式相互作用。

E/E/PES的不同应用程序的设计中可以包含多个相同或非常相似的软件模块或组件。建立一个普遍适用的软件模块库，并允许确认设计所需要的资源能被多个应用共享。

此外，软件模块在多个应用中的使用提供了成功操作使用的经验性证据。这些经验性证据无疑增加了使用者对这些软件模块的信任。

## 7.4.4.1.10 计算机辅助设计工具

## 7.4.4.1.10.1 目标

CAD工具用于实现更系统化的程序设计，且包含可用的和经过测试的合适的自动构建元件。

## 7.4.4.1.10.2 描述

当系统复杂度合理且可接受时，在硬件和软件的设计中使用CAD工具。这种工具的正确性应通过特定测试、大量的历史使用满意度或有关安全系统输出的独立验证来证明。

## 7.4.4.1.11 编码标准的使用

## 7.4.4.1.11.1 目标

应使用编码标准以利于生成代码的验证。

## 7.4.4.1.11.2 描述

在编码之前应对详细规则达成一致。这些规则通常要求：

——模块化的细节，如接口形状，软件模块大小；

——在面向对象的语言情况下使用的封装，继承（深度限制）和多态性；

——限制使用或避免某些语言结构如“go-to语句”、“等价”、动态对象、动态数据、动态数据结构、递归、指针和退出；

——安全关键代码执行期间中断启用的限制；

——代码方案（列出）；

——高级语言程序中没有无条件跳转（例如“go-to语句”）。

这些规则使软件模块易于测试、验证、评估和维护。因此，可考虑使用特定分析仪中的可用工具。

## 7.4.4.1.12设计和编码标准——无动态变量或对象

## 7.4.4.1.12.1 目标

设计和编码标准应排除要避免的动态变量或对象，比如：

——不必要或未检测到的内存覆盖，

——（与安全有关的）运行中的资源瓶颈。

## 7.4.4.1.12.2 描述

为了达成该措施的目的，动态变量和动态对象是在运行时分配内存和确定绝对地址的变量和对象。内存分配值及地址值取决于分配时的系统状态，这意味着它不能由编译器或任何其他离线工具检查。

由于动态变量和对象的数量以及现有分配新动态变量或对象的可用内存空间，取决于在分配时的系统状态。因此，分配、使用变量或对象时，可能发生故障。例如，由系统分配地址时可用内存空间不足，另一变量在存储器中的内容可能在不经意间被覆盖。如果不使用动态变量或对象，则可避免这些故障。

## 7.4.4.1.13设计和编码标准——中断的使用限制

## 7.4.4.1.13.1 目标

软件开发人员应限制使用中断，以保持软件可验证性和可测试性。

## 7.4.4.1.13.2 描述

应限制使用中断。如果中断使系统简化则可使用中断。在关键软件的执行过程中应禁止软件处理中断。如果使用中断，则应为不能被中断的部分指定最大计算时间，以便于计算禁止中断的最大时间。中断的使用和禁止应完整记录。

## 7.4.4.1.14设计和编码标准——指针的定义使用

## 7.4.4.1.14.1 目标

指针的定义使用用来避免由于访问未预先检查范围和指针类型数据所造成的问题，以支持软件的模块化测试和验证并限制失效后果。

## 7.4.4.1.14.2 描述

在应用软件中，仅当指针数据类型和数值范围（确保指针引用在正确地址空间内）经过检查后，指针运算方可在源代码级使用。

## 7.4.4.1.15设计和编码标准——递归的限制使用

## 7.4.4.1.15.1 目标

递归的限制使用，用以避免子程序调用的无法验证和不稳定使用。

## 7.4.4.1.15.2 描述

若使用递归，应建立允许递归深度的明确准则。

7.4.4.2 软件模块设计和编码验证

应验证软件模块的设计及其编码。应检验设计和编码是否满足软件安全要求。软件开发者应参加验证活动。验证方法可能是检查或走查（在GB/TXXXX.1中有定义）两种类型之一。

## 7.4.5 工作产品

下列工作产品适用于本阶段：

a）依据7.4.4.1的软件详细设计；

b）依据7.4.4.1的软件；

c）来自于7.4.4.2的软件模块设计和编码验证报告。

## 7.5 软件模块测试

## 7.5.1 目的

软件模块测试的目的是验证设计和编码的软件模块正确地实现了软件要求。

## 7.5.2 概述

在本阶段，应建立按照要求进行的软件模块测试程序，并按照该程序进行测试。

## 7.5.3 前提条件

软件模块测试的前提条件如下：

——软件项目计划（见7.1.4.2至7.1.4.4）；

——软件要求[见7.2.5 a）和b）]；

——软件验证计划（见GB/TXXXX.4，第6条）；

——依据7.4.4.1的软件模块。

## 7.5.4 要求

## 7.5.4.1 软件模块测试方法

软件模块测试应依据表4进行。

## 7.5.4.1.1 动态分析和测试

## 7.5.4.1.1.1 目标

动态分析和测试用于软件开发完成前状态时，通过原型动态行为的检查来检测规范失效。

## 7.5.4.1.1.2 描述

有关安全系统的动态分析是通过对有关安全系统的近似操作原型输入典型的预定操作环境数据来进行的。如果观察到的有关安全系统行为与要求行为一致，则分析符合要求。任何有关安全系统的失效均应予以纠正，并应重新分析新运行版本。

## 7.5.4.1.2边界值分析的测试用例执行

## 7.5.4.1.2.1目标

边界值分析的测试用例执行用于检测在参数限制或边界上发生的软件错误。

表4 软件模块测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 技术/措施a | 子条款 | SRL=B | SRL=1 | SRL=2 | SRL=3 |
| 1 动态分析和测试7.5.4.1.1 | | | | | |
| 1.1 边界值分析的测试用例执行 | 7.5.4.1.2 | o | o | o | + |
| 1.2 基于结构的测试 | 7.5.4.1.3 | o | o | o | + |
| 2 静态分析 | | | | | |
| 2.1 边界值分析 | 7.5.4.1.4 | + | + | + | + |
| 2.2 审核清单 | 7.5.4.1.5 | + | + | + | + |
| 2.3 控制流分析 | 7.5.4.1.6 | o | o | + | + |
| 2.4 数据流分析 | 7.5.4.1.7 | o | o | + | + |
| 2.5 走查/设计复查 | 7.5.4.1.8 | o | o | + | + |
| 3 功能和黑盒测试 | | | | | |
| 3.1 等价类和输入分区测试 | 7.5.4.1.9 | o | o | + | o |
| 3.2 边界值分析 | 7.5.4.1.4 | o | o | o | + |
| 4 性能测试7.5.4.1.10 | | | | | |
| 4.1 资源预算测试 | 7.5.4.1.11 | o | + | o | o |
| 4.2 响应时间和内存约束 | 7.5.4.1.12 | o | o | + | + |
| 4.3 性能要求 | 7.5.4.1.13 | o | o | + | + |
| 4.4 崩溃/压力测试 | 7.5.4.1.14 | o | o | o | + |
| 5 接口测试 | 7.5.4.1.15 | o | o | o | + |
| 见7.1.4.7关于本表格及其他表格使用说明 | | | | | |
| a 依照SRL选择技术/措施。可选或等效技术/措施用数字后的字母表示，仅需满足可选或等效的技术/措施中的一个。 | | | | | |

## 7.5.4.1.2.2 描述

程序的输入域按照等价关系划分成多个输入类别（见7.5.4.1.9）。测试应包括类别的边界和极值。测试检验规范输入域的边界是否与程序中的相一致。零值的使用在直接和间接转化中易出错，且需特别注意以下几点：

——零除数；

——空ASCII字符；

——空堆栈或列表中的元素；

——全矩阵；

——零表格入口。

通常，输入的边界直接对应一个输出范围的边界。应写入测试用例以强制输出达到其极限值。也可考虑可能指定一个测试用例引起输出超过规范的边界值。

如果输出是一个数据序列（如打印表），应特别注意第一个和最后一个元素，以及包括空元素、一个元素和两个元素的列表。

## 7.5.4.1.3 基于结构的测试

## 7.5.4.1.3.1 目标

基于结构的测试用于对程序结构中的某些子集的测试。

## 7.5.4.1.3.2 描述

基于程序分析，选择一组输入数据以便于大部分（预先设定目标）的程序代码得以运行。根据严格要求程度，代码覆盖率措施的变化如下。

——语句

这是最不严格的测试，因为有可能在未运行条件语句分支的情况下执行所有的代码语句。

——分支

每个分支的两边均应进行检查。这对于防御性代码的某些类型可能是不切实际的。

——复合条件

复合条件分支中的每个条件（即和/或的链接）均应执行。

——线性代码序列和跳转

适用于任何代码语句的线性序列，其中包括被跳转终止的条件语句。早期代码的执行对输入数据施加的限制，使得许多潜在子路径不可行。

——数据流

基于数据使用选择执行路径，例如，同一变量的写入和读取的路径。

——调用图

程序由可以被其他子程序调用的子程序组成。调用图是程序中的子程序调用树。所设计的测试应覆盖树上的所有调用。

——基本路径

从开始到结束有限路径的最小集合之一，包括所有弧（在基本集合中路径的重叠组合可以形成通过程序该部分的任意路径）。所有基本路径的测试可以高效的定位错误。

## 7.5.4.1.4 边界值分析

## 7.5.4.1.4.1 目标

边界值分析用于检测软件在参数限制或边界上发生的错误。

## 7.5.4.1.4.2 描述

程序的输入域依据等价关系被分成若干个输入类（见7.5.4.1.9）。测试应覆盖输入类的边界和极值。测试检查规范规定的输入域的边界与程序中的一致。零值的使用在直接和间接转化中易出错，且需特别注意以下几点：

——零除数；

——空ASCII字符；

——空堆栈或列表中的元素；

——全矩阵；

——零表格输入。

通常，输入的边界直接对应一个输出范围的边界。测试用例应被写入以强制将输出达到其限制值。也可考虑可能指定一个测试用例引起输出超过规范的边界值。

如果输出是一个数据序列（如打印表），应特别注意第一个和最后一个元素，以及包括空元素、一个元素和两个元素的列表。

## 7.5.4.1.5 审核清单

## 7.5.4.1.5.1 目标

审核清单用于提醒注意和管理安全寿命周期阶段的各个重要方面的评价，以确保满足精确要求时各方面的全面覆盖。

## 7.5.4.1.5.2 描述

审核清单是由执行审核的人回答的一组问题。许多问题是通用的，且应由评审员给出最恰当的解释。清单应被用于全部E/E/PES的软件安全寿命周期的所有阶段，作为功能安全性评估的辅助工具时特别有用。为适应需验证系统的大范围的变化，多数清单包含适用于多种类型系统的问题。因此，清单中的一些问题是与正在处理的系统无关的且可以忽略的。同样地，对特定系统，也可以针对正处理的这个系统在标准审核清单中补充一些问题。在任何情况下，需要明确的是所使用的清单取决于选择以及应用清单的工程师的专业知识和评价。因此，工程师做出的有关选择清单的决定以及任何附加或多余问题，应被完整记录和注释。目的是确保应用清单可以被复查，且能达到可重复的结果，除非使用不同的准则。检查表中的全部对象应尽量简练。当需要扩展证明时，则需要引用附加文档。通过、不通过和未有结论，或一些受限的相似结论，应用于记录每个问题的结果。这一精简极大地简化了在完成清单的评估结果时得出整体结论的程序。

## 7.5.4.1.6静态分析——控制流分析

## 7.5.4.1.6.1 目标

控制流分析应用于检测欠佳的和潜在错误的程序结构。

## 7.5.4.1.6.2 描述

控制流分析是一种查找不遵循良好编程实践的可疑代码区域的静态测试技术。分析程序生成的有向图可供进一步分析：

——无法访问的代码，例如无条件跳转使得代码块不可达

——多节代码——相比于具有可归约控制图的良好结构性代码可依次简化为单个节点——结构性欠佳代码只能简化为由多个节点组成的结。

## 7.5.4.1.7 静态分析——数据流分析

## 7.5.4.1.7.1 目标

数据流分析应用于检测欠佳的和潜在错误的程序结构。

## 7.5.4.1.7.2 描述

数据流分析是一种静态测试技术，将从控制流分析获得的信息和在代码不同部分中进行读取或写入变量信息相结合。分析可以检查以下变量类型：

——那些赋值之前可以被读取的变量，则总可通过在声明一个新变量时赋值来避免。

——那些被多次写入而未被读取的变量，则可能表明是遗漏代码。

——那些被写入但从未被读取过的变量，则可能表明是冗余代码。

数据流异常并不总是直接对应于程序故障；然而，若避免异常，则代码不太可能包含故障。

## 7.5.4.1.8 静态分析——走查/设计复查

## 7.5.4.1.8.1 目标

走查/设计复查用于在开发阶段尽可能经济且快速的检测故障。

## 7.5.4.1.8.2 描述

对所有新产品/进程、新应用以及现有产品和制造工艺的修改应进行形式化设计复查，这些方面影响功能、性能、安全性、可靠性、可检查性、可维修性、可用性、成本和其他影响最终产品/进程（用户或旁观者）的特性。

代码走查包括给程序选择的一些纸质测试用例、典型的输入设置和相应的程序期望输出的走查小组。然后测试数据通过程序逻辑进行手动跟踪。

## 7.5.4.1.9 等价类和输入分区测试

## 7.5.4.1.9.1 目标

等价类和输入分区测试应利用最少的测试数据充分测试软件。应通过选择运行软件所需输入域的分区来获得测试数据。

## 7.5.4.1.9.2 描述

测试策略应基于输入的等价关系以确定输入域的分区。

测试用例的选择应涵盖所有事先指定的分区。至少要有一个测试用例来自每个等价类。

输入划分有两种基本的可能性：

——源于规范的等价类——规范的解释可以是面向输入（例如选择的值以同一种方式处理）或面向输出（例如一组值导致相同的功能结果）；

——源于程序内部结构的等价类——等价类结果由程序的静态分析确定（例如一组值导致执行相同路径）。

## 7.5.4.1.10 性能测试

## 7.5.4.1.10.1目标

性能测试应用于确保系统的工作能力充分满足指定的要求。

## 7.5.4.1.10.2 描述

要求规范须包括特定功能的吞吐量和响应时间要求，也可以结合整个系统资源在使用上的约束条件。建议的系统设计按照以下方式与声明的要求进行比较：

——搭建一个系统过程及其相互作用的模型；

——确定每个过程资源的使用（处理器时间、通信带宽、存储设备等）；

——确定系统在平均及最坏条件下的需求分配；

——计算单个系统功能吞吐量与响应时间的均值和最坏情况。

## 7.5.4.1.11 性能测试——资源预算测试

## 7.5.4.1.11.1 目标

资源预算测试应依据系统的复杂性进行。

——对于简单系统，一个解析解可能是足够的；

——对于更复杂系统，某种形式的仿真可能更适合于获得准确结果。

## 7.5.4.1.11.2描述

在详细建模前，简单的“资源预算”检查可以用于所有进程的资源要求的合计。如果要求超过设计的系统容量，则设计是不可行。即使设计通过检查，建模性能也可能显示由于资源匮乏出现过度延迟和响应时间。为避免这种情况，工程师设计系统时通常只使用资源总额的一部分（例如 50%），以减少资源匮乏的概率。

## 7.5.4.1.12 性能测试——响应时间和内存约束

## 7.5.4.1.12.1 目标

响应时间和内存约束用于确保系统满足时间和内存要求。

## 7.5.4.1.12.2 描述

系统和软件的要求规范包括特定功能的内存和响应要求，也可能结合整个系统资源使用的约束。通过分析确定在平均和最坏情况下的分配需求。该分析需要每个系统功能资源使用情况和所用时间的估计值。这些估计数据可以通过几种方式获得（例如与现有系统的比较或时延敏感系统的原型和基准）。

## 7.5.4.1.13性能测试——性能要求

## 7.5.4.1.13.1 目标

应建立用于证明软件系统性能要求的测试。

## 7.5.4.1.13.2 描述

应对系统和软件要求规范进行分析，以指定所有一般/特定和明确/隐含的性能要求。

为确定以下内容，应对每个性能要求进行检查：

——获得成功准则；

——是否有符合成功准则的失效；

——措施的潜在准确性；

——可估计措施的项目阶段；

——可制定措施的项目阶段。

应分析每个性能要求的实用性，以便获得性能要求、成功准则和潜在措施的列表。主要目标为：

a）每个性能要求至少关联一个措施；

b）在可能的情况下，在开发阶段应尽早地选择准确和有效的措施；

c）应规定必需的和可选的性能要求和成功的准则；

d）在可能的情况下，应优先考虑为多个性能要求使用单个措施的可能性。

## 7.5.4.1.14 性能测试——崩溃/压力测试

## 7.5.4.1.14.1 目标

崩溃/压力测试应用于将异常的高工作负载施加给测试对象，以表明测试对象能容易的在正常工作负载下工作。

## 7.5.4.1.14.2 描述

有多种测试条件适用于崩溃/压力测试，其中包括：

——如果在轮询模式下工作，那么测试对象在单位时间内可获得比正常条件下更多的输入变化；

——如果在需求模式下工作，那么在单位时间内施加给测试对象的需求数的增加超出正常条件；

——如果数据库的大小起重要作用，则其增加超出正常条件；

——有影响的设备要分别调到最高速度或最低速度；

——极端情况下，所有影响因素，尽可能的在同时放在边界条件中。

在这些测试条件下，可以评估测试对象的时间行为，观察负荷变化的影响以及检测内部缓冲区或动态变量、堆栈等的尺寸是否正确。

## 7.5.4.1.15 接口测试

## 7.5.4.1.15.1 目标

接口测试应用于检测子程序接口中的错误。

## 7.5.4.1.15.2 描述

细化多个等级或测试完整性是可行的。最重要的等级测试是：

——所有接口变量都为极值；

——所有接口变量分别为极值，其他接口变量为正常值；

——每个接口变量在值域内的所有值，其他接口变量为正常值；

——所有变量所有值的组合（仅接口数量少时可行）；

——和每个子程序的每次调用相关的指定测试条件。

如果接口不包含检测不正确参数值的声明，那么这些测试是特别重要的。在预先存在的子程序的新配置生成后，测试也很重要。

在此阶段中检测到的错误应消除。应对每个修改项进行影响分析。对影响任何前一阶段工作产品产生的所有修改都应启动返回到软件安全寿命周期的相应阶段。应按照GB/TXXXX的相关部分再执行后续阶段。

## 7.5.5 工作产品

以下工作产品适用于本阶段：

a）源于7.5.4.1的软件模块测试计划；

b）与7.5.4.1一致的软件模块测试规范；

c）源于测试性能的软件模块测试报告。

## 7.6 软件集成和测试

## 7.6.1 目的

软件集成和测试的第一个目的是把软件单元逐步集成为软件组件，直至完整的ECU嵌入式软件。

注：嵌入式软件可以由有关安全或非有关安全软件组件组成。

第二个目的是验证软件要求已被嵌入式软件正确实现。

## 7.6.2 概述

本阶段中，特殊的集成等级是根据软件要求来测试的。软件模块和/或软件组件之间的接口也要测试。软件组件集成及测试的步骤应直接与软件架构的层级相对应。

## 7.6.3 前提条件

以下是软件集成和测试的前提条件：

——软件项目计划（见7.1.4.2 至7.1.4.4）；

——软件要求[见7.2.5a）和b）]；

——软件架构（见7.3.4.1至7.3.4.5）；

——软件验证计划（见GB/TXXXX.4，第6条）；

——参照7.4.4.1测试的软件模块。

## 7.6.4 要求

## 7.6.4.1 软件集成和测试计划

软件集成和测试的计划应至少包含以下内容：

1. 软件集成策略；
2. 软件集成测试计划制定。

软件集成策略和软件测试计划应在软件构架和设计阶段开发。

## 7.6.4.2 软件集成策略

软件集成策略应至少描述以下内容：

1. 将单个软件模块分层整合为软件组件，直到完成整个ECU嵌入式软件的集成步骤；
2. 与软件集成相关的功能从属。

注1：如果在目标硬件上的软硬件集成和嵌入式软件测试还没有计划和执行，那么这应成为集成策略的一部分。该程序有时能够使软件集成和测试简单化。

注2：如果是基于模型开发，那么可以由模型级的集成和之后的集成模型的代码生成代替软件集成。

注3：依赖于约束条件，软件可以在一个主机环境、类似目标环境（例如一个评估板）或目标环境（ECU）内集成。

## 7.6.4.3 软件集成和测试程序

合适的测试程序应在软件集成测试计划制定阶段开发。

注：因为单个测试程序无法涵盖应考虑的所有方面，软件集成测试总是需要组合不同的程序。

## 7.6.4.4 软件集成和测试方法

硬件和软件集成测试应依照表5执行。

表5 集成测试（模块）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 技术/措施a | 子条款 | SRL=B | SRL=1 | SRL=2 | SRL=3 |
| 1 功能和黑盒测试7.6.4.4.1 | | | | | |
| 1.1等价类和输入分区测试 | 7.5.4.1.9 | o | o | + | o |
| 1.2边界值分析 | 7.5.4.1.4 | o | o | o | + |
| 2 性能测试 | | | | | |
| 2.1资源预算测试 | 7.5.4.1.11 | o | + | o | o |
| 2.2响应时间和内存约束 | 7.5.4.1.12 | o | o | + | + |
| 2.3性能要求 | 7.5.4.1.13 | o | o | + | + |
| 2.4崩溃/压力测试 | 7.5.4.1.14 | o | o | o | + |
| 见7.1.4.7本表和其他表格的使用说明 | | | | | |
| a 依照SRL选择技术/措施。可选或等效的技术/措施用数字后面的字母表示，仅需满足可选或等效的技术/措施中的一个。 | | | | | |

## 7.6.4.4.1 功能测试

## 7.6.4.4.1.1 目标

功能测试应用于发现规范和设计阶段的失效，以避免软硬件集成和执行期间出现的失效。

## 7.6.4.4.1.2 描述

功能测试期间，应执行复查以确定是否已经达到规定的系统特征，且是否给定能充分代表正常期望操作特征的系统输入数据。应观察输出，并将其响应与规范给定的响应相比较。应记录规范偏差和不完善规范的指示。为多通道架构设计的电子组件的功能测试通常包括用于预验证合作者组件测试的生产部件。

另外，推荐对已加工组件和其他合作者的同批组件联合检测，以发现共模故障，否则会一直隐藏。

## 7.6.4.5 消除缺陷

应消除在此阶段中检测到的错误。应对每个修改项进行影响分析。对前一阶段工作产品产生影响的所有修改，都应启动返回到软件安全寿命周期的相应阶段。应按照GB/TXXXX的相关部分再执行后续阶段。

## 7.6.5工作产品

以下工作产品适用于本阶段

1. 软件集成测试计划，包含来自于7.6.4.1至7.6.4.3的软件集成策略；
2. 7.6.4.3要求的软件集成测试规范；
3. 符合7.6.4.1的软件集成测试报告。

## 7.7 软件安全确认

## 7.7.1 目的

软件安全确认的第一个目的是表明软件要求已被嵌入式软件正确实现。

第二个目的是提供证据，以证明在机器级别上的技术安全概念的要求是合格的、完整的和完全可达的。

注1：软件安全确认是E/E/PES系统安全确认的一部分（见GB/TXXXX.4，第6条）。在完整E/E/PES系统安全确认的计划期间，应确定哪些安全目标在E/E/PES系统级测试，哪些安全目标在软件级别检测。在最简单的情况下，应考虑所有的安全目标由包含软件的E/E/PES系统的安全确认覆盖，因此不需要单独的软件安全确认。

注2：软件安全确认的前提条件是软硬件集成已经完成。

## 7.7.2 概述

## 7.7.3前提条件

以下是软件安全确认的前提条件：

——软件项目计划（见7.1.4.2至7.1.4.4）；

——软件要求[见7.2.5a）和b）]；

——软件架构（见7.3.4.1至7.3.4.5）；

——软件验证计划（见GB/TXXXX.4第6条）；

——集成软件；

——ECU。

## 7.7.4 要求

## 7.7.4.1软件安全确认方法

测试应是软件的主要验证方法；动画和建模可以作为验证活动的补充；应参照表6来选择适当的措施/技术。

表6 软件安全确认

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 技术/措施a | 子条款 | SRL=B | SRL=1 | SRL=2 | SRL=3 |
| 1 软件安全要求测试 | | | | | |
| 1.1测试接口 | 7.7.4.1.1 | + | + | + | + |
| 1.2aECU网络a内测试 | 7.7.4.1.2 | o | + | + | o |
| 1.2b硬件在环测试 | 7.7.4.1.3 | o | o | + | + |
| 1.2c 机器内测试 | 7.7.4.1.4 | o | o | o | + |
| 见7.1.4.7本表和其他表格的使用说明 | | | | | |
| 注：1.2a，1.2b，1.2c中的措施代表测试环境。 | | | | | |
| a 依照SRL选择技术/措施。可选或等效的技术/措施用数字后的字母表示，仅需满足可选或等效的技术/措施中的一个。 | | | | | |

## 7.7.4.1.1 测试接口

软件应和其相关联的ECU主机微处理器相集成。测试接口用以确定测试时ECU的内部状态，同时监控内部结果。

## 7.7.4.1.2 电子控制单元网络内测试

软件应和其相关联的ECU主机微处理器相集成，并且该ECU应与完整E/E/PES系统其他的ECU相集成。软件应在ECU网络的接口进行测试以证明软件按规范执行。

## 7.7.4.1.3 硬件在环测试

软件应与它相关ECU的主机微处理器相集成，与此同时仿真关联E/E/PES系统的剩余部分及其环境。应进一步在仿真环境中测试软件，以表明这个软件按规范执行。

## 7.7.4.1.4 机器内测试

软件及其相关联的E/E/PES系统应集成到相关联的机器架构中。系统应在机器中进行测试以表明软件按规范执行。

## 7.7.4.2 测试范围

软件应执行以下仿真：

——正常操作期间出现的输入信号；

——预期发生的事件；

——要求系统行为的非预期条件。

## 7.7.4.3 软件安全要求确认

测试程序及其他使用措施的有效性应依照安全概念在验证过程中的结论进行评估，以确认软件安全要求。

## 7.7.4.4 文档

供应商和/或开发人员应提供软件安全确认记录结果及所有相关文档给系统开发者，以使他们能满足GB/TXXXX.4的要求。

## 7.7.4.5 消除缺陷

应消除在本阶段中检测到的错误。应对每个修改项进行影响分析。对影响任何前一阶段工作产品的所有修改，都应启动返回到软件安全寿命周期的相应阶段。应按照GB/TXXXX的相关部分再执行后续阶段。

## 7.7.5 工作产品

以下工作产品适用于本阶段：

1. 源于7.7.4.1至7.7.4.4的软件确认计划；
2. 源于7.7.4.1至7.7.4.2的软件确认测试规范；
3. 按照7.7.4.3的软件确认测试报告。

## 7.8 基于软件的参数化

## 7.8.1 目的

基于软件的参数化指在开发完成后通过更改参数来修改软件的功能性，使软件系统能够适应不同的要求。

其目的是导出有关安全参数的需求。

## 7.8.2 概述

有关安全参数的基于软件的参数化应认为是SRP/CS设计的有关安全方面，在软件安全要求中进行描述。基于软件的参数包括

——变量编码（例如，国家代码，左置转向/右置转向）

——参数（例如，低怠速值，发动机特性图）

——标定数据（例如，车辆特性，为油门设定的停机限值）

## 7.8.3 前提条件

以下是基于软件参数化的前提条件：

——软件项目计划（见7.1.4.2 至7.1.4.4）

——软件要求[见7.2.5a）和b）]

——软件架构（参见7.3.4.1至7.3.4.5）

——软件验证计划（见GB/TXXXX.4，第6条）

——依照7.4.4.1测试的软件模块。

## 7.8.4 要求

## 7.8.4.1 数据完整性

应保持用于参数化的数据完整性，应阻止未经授权的修改。可以通过采取措施控制以下方面来实现

1. 有效输入范围，
2. 传输前后的数据损坏，包括

——检查配置数据的有效范围；

——对配置数据进行合理性检查；

——使用冗余数据存储；

——使用检错和纠错码；

1. 参数传输过程中的错误；
2. 不完整参数传输的影响；
3. 用于参数化工具的软硬件故障及失效影响。

## 7.8.4.2 参数数据的可执行代码

参数数据应不包含可执行代码。

## 7.8.4.3 配置管理

基于软件的参数化应是配置管理版本的一部分（参见GB/TXXXX.4，第6条）。

## 7.8.4.4 基于软件参数化的验证

基于软件的参数化验证应进行以下验证活动

——每个有关安全参数正确设定的验证（最小值、最大值和代表值）；

——有关安全参数合理性已检查的验证，通过无效值使用等；

——已阻止有关安全参数未经授权修改的验证；

——参数化的数据/信号应以故障不能导致安全功能遗失的方式产生及处理的验证。

## 7.8.5 工作产品

以下工作产品适用于本阶段：

1. 已确认的有关安全软件参数配置；
2. 源于7.7.4.1至7.7.4.3的软件确认计划；
3. 源于7.7.4.1的软件确认测试规范；
4. 源于7.7.4.3的软件确认测试报告。

# 

# 附录A

（资料性附录）

AgPL=e功能安全评估的议程范例

A.1 系统功能

A.1.1 系统、组件和功能性质的详细描述

A.1.2 电气组件的概述

A.1.3 功能的危险和风险分析

A.2 硬件

A.2.1 功能框图

A.2.2 布局和接线图

A.2.3 环境连接（接口）

A.3 安全概念

A.3.1 安全概念的基本原则

A.3.2 “安全状态”或“降级原则”的定义

A.3.3 安全概念的功能

A.3.4 故障条件（容错）下的安全功能的验证

A.3.5 安全概念与其他系统/功能的交互

A.4 安全分析和安全数据

A.4.1 失效分析（FMEA、FTA等）

A.4.2 内部监控功能

A.4.3 硬件失效率（对于不同的失效模式，例如来自数据库、 FTA、FMEA等）

A.4.4 用于发现内部故障的数据（例如，组件失效的诊断发现、失效识别次数等）

A.4.5 用于发现外部故障的数据（例如，网络、传感器、开关、电源失效等）

A.5 安全寿命周期各阶段的设计过程

A.5.1 项目管理

A.5.2 文档

A.5.3 规范阶段

A.5.4计划和开发阶段

A.5.5 集成阶段

A.5.6 通用确认/安全确认计划

A.5.7 通用确认/安全确认的结果

A.6 软件开发

A.6.1 软件安全概念

A.6.2 软件结构

A.6.3软件测试及文档

A.6.4 所用开发工具

A.6.5软件修改的识别和追踪（“版本控制”）

A.6.6 保护已执行的软件，防止未经授权的修改。

A.7 验证和测试

A.7.1 在无失效条件下系统功能的验证

A.7.2 在失效影响情况下系统功能的验证

A.8 文档和安全文档

A.8.1 完整性

A.8.2 一致性

A.9 汇总和评估

附录B

（资料性附录）

软件划分的独立性

B.1概述

在当前技术水平迅速发展的情况下，本附录提供了一种方法——软件划分及其相关的方法和措施——以帮助设计者证明软件模块的独立性。

通过排除违背此独立性的特别故障影响，保证软件组件的充分独立性的。为达到此目的，必要的方法和措施至少要执行至中等效力。对于本附录中给出的每种方法或措施，其效力推荐如下：（见表B.1和B.2）

“高” 意味着能有效阻止相应的故障影响；

“中” 意味着能部分阻止相应的故障影响；

“无” 意味着无助于阻止相应的故障影响。

B.2 术语，定义和缩略语

下列术语、定义和缩略语应用于本附录。

B.2.1

活动计数器 Alive counter

监控对象创建时，计数组件初始化为0。

注：只要对象是活动的，计数器从时刻 t-1至时刻 t递增。最后，活动计数器显示对象在网络中活动的持续时间。

B.2.2

黑盒测试Black-box test

不要求知道测试对象内部结构或具体执行的测试。

B.2.3

总线监护器 Bus guardian

位于节点和总线间的独立组件，仅当总线节点被允许传输时，才允许向总线传输数据。

注：监护器需要知道它的节点何时被允许访问总线——在事件触发系统中很难达到，但在时间触发系统中则很简单。

B.2.4

消息队列Message queue

（内部过程/内部任务通信）程序用于在异步过程或任务间传递和交换数据（消息），包括FIFO缓冲数据。

注：序列由操作系统管理，或由应用程序管理，消息排序能同步及互斥。

B.2.5

微时隙 Minislotting

总线调度技术使连接在总线中的每个节点需要等待一定的时间才能再次访问网络。

B.2.6

互斥Mutual exclusion

同步机制保护应分别执行的状态序列。

B.2.7

分区Partitions

资源实体，即带有像存储器配额、I / O设备及CPU时间等分配资源的一个虚拟机器环境。

注1：分区是静态指定的，并在系统启动时创建。

注2：在一个分区中运行的程序总是受限于由系统配置在系统启动时分配的资源。

B.2.8

微控制器分区Partition

（在单独的微控制器中）由固定配额的系统资源组成的子系统，每个分区包含一个或多个任务。

B.2.9

网络分区Partition

（在微控制器网络范围内）由可用内存、可用CPU和微控制器I/O功能组成的子系统。

B.2.10

分割 Partitioning

软件分割Software partitioning

故障隔离技术，排除存在于某个分区内的可能传播或者导致其他分区失效的技术。

注1：应用于软件时，分割的目的在于当一个软件分区与其他分区共享其资源或部分资源时（例如，处理器和/或外围设备），控制额外风险的产生。软件分割不是要防止单个软件分区的失效，而是要防止这些失效的传播。

注2：软件分割包含两个方面，一方面是分配外围设备未经授权的数据访和非法的命令给其他分区的空间分割。另一方面是分配时间定时的干扰给其他分区的时间分区（调度，执行命令等）。

B.2.11

管道Pipe

两个进程/任务之间的单管道通信，包括遵循FIFO原则的缓冲。

注：管道始建于消息队列的基础上，使用标准I/O接口，即一个进程/任务的输出用作另外一个的输入。

B.2.12

真实寻址Real addressing

绝对寻址Absolute dressing

存储单元或外围设备的明确标识。

参考相对寻址Relative addressing（B.2.14）

B.2.13

冗余Redundancy

倍增即大多数情况下的系统组件的副本用于提高系统的可靠性。

注：这一功能通常是在有必要使用备份或者自动防故障装置的情况下才使用的。

B.2.14

相对寻址Relative addressing

从其他地址的偏移确定一个内存位置或外围设备的标识。

参考真实寻址real addressing（B.2.12）

B.2.15

共享内存Shared memory

用于内部通信的内存。

注：共享内存是指定的多任务可直接访问的内存区域。信号量用于避免干扰并确保内存完整性。这种进程间/任务间通信的方法比通过操作系统服务交换数据更快。

B.2.16

信号量Semaphore

非负整数变量，其值在进入临界程序段时减小，而在离开该程序段时重新增加。

注：当多个任务访问同一个共用资源，例如一个共用数据空间时，信号量用于同步。

B.2.17

软件组件Software component

软件中一个或多个功能的执行。

注：软件组件是软件的一个逻辑可分的部分，包含一个或多个软件组件和/或软件单元。在软件架构内，软件组件是由分区和任务来实现的。

B2.18

软件单元 Software unit

软件最小的独立部分，它能够被独立翻译，可以用相关数据来测试它是否执行规范

注：软件单元是一个原子级软件组件。

B.2.19

软件分区Software partitions

具有分配的单独系统资源的运行时间环境。

B.2.20

系统资源System resources

软件操作要求的所有资源。

例如，CPU时间、I/O设备、内存。

B.2.21

任务Task

运行时间实体，在分区的资源预算内执行，每个任务有其自己的堆栈和优先级。

注：任务是根据分配给它和选定调度策略的任务优先级的调度器控制下执行。

B.2.22

软件的独立性Independence of software

排除软件组件之间的非有意的相互作用，也免于受到来自另外一个软件组件正确操作的影响，该影响源于软件组件错误的设计和/或执行。

B.2.23

用户模式User mode

CPU的两个执行模式中的一个，另外一个是内核模式。

注：用户模式是非特权的，即引用或访问内存和I / O空间将被检查授权。例如，在用户模式下，分配和访问其他进程的特定存储区域是被禁止的。相反，内核模式赋予特权访问和执行的权利。

B.2.24

看门狗Watchdog

定时器进程，如果在某一个周期内没有被一个软件组件重新设置，则认为该软件组件有错误。

注：软件组件应连续的重置定时器，以显示其功能正常。

CRC 循环冗余校验

CPU 中央处理器

MMU 内存管理单元

MPU 内存保护单元

B.3 目的

第一个目的是控制子系统内发生的危险，以使其不影响其他子系统。

注：软件危险的发生可归因于软件组件的设计和执行错误，会干扰与其分享资源的其他软件组件的正确操作。

第二个目的是规定如何证明通过软件分区来实现的软件组件足够的独立性。

B.4 概述

为了达到软件组件足够的独立性，系统资源应被分配给独立的子系统或分区，每个代表一个特别的运行时间环境。系统资源包括CPU、内存、总线、I/O-通道和诸如文件句柄等的操作系统资源。

软件分割的使用不受制于相同运行时间环境中的不同SRL的共存性。它也可支持：

a）没有未修改的软件分区的重新验证的分区更改

b）不同性质的软件的共存性（自身的，第三方）

无充足硬件支持的分割通常是不可能的。

为了隔离在一个共享资源环境内的多重分区，硬件应为每个单独的分区提供具有受限内存空间、处理时间和访问I/O能力的操作系统。

注：分区可以在一个单一的微控制器内分配，或分配给微控制器网络内的若干个微控制器。

根据所选择的架构，有两种方法可以使用：

1. 单个微控制器的若干分区
2. ECU网络范围内的若干分区

B.5 要求

B.5.1 一般要求

B.5.1.1 SRL

支持分区的软件的这个部分应该有与分区软件相关联的最高的SRL相同的或比其更高的SRL。

注：通常，提供或支持分割的软件是操作系统的一部分。

B.5.1.2 软件架构

应在规定软件架构时考虑软件分割概念。

B.5.2单个微控制器内的若干分区

B.5.2.1概述

见图B.1。

注：同一个分区内的任务应相互不独立。

B.5.2.2 软件分割的方法/措施

每个分区应执行“表B.1”内的方法和措施，以防止以下故障影响，并保证软件组件足够的独立性：

——内存破坏（意外写入其他分区的内存）；

——分区的阻塞（由于通信死锁）；

——处理器执行时间的错误分配；

——错误的对等通信（发送方发送消息给错误的接受方，或冒充别的发送方发送消息）；

——由于意外写入到其他分区的I/O接口而引起的I/O接口的破坏。

B.5.2.3 软件分割效力

表B.1中给出的方法和措施应至少应用至中等效力，以保证软件组件足够的独立性。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 微控制器 | | | | | | | |
|  |  |  |  |  |  |  |  |
|  |  | 任务A.1 |  |  | 任务B.1 |  |  |
|  |  | 任务A.2 |  |  | 任务B.2 |  |  |
|  |  | 任务A.n |  |  | 任务B.n |  |  |
|  |  | 分区A |  |  | 分区B |  |  |
|  | 操作系统 | | | | | |  |
|  | 硬件硬件 | | | | | |  |
|  | | | | | | |  |

图B.1 单个微控制器内的若干分区

B.5.3微控制器网络内的若干分区

B.5.3.1概述

软件组件在各自微控制器内各自分区中的执行，如图 B.2所示。

微控制网络可以由单个ECU上的若干处理器组成，通过内部数据总线（内部处理器通信）通信。如图 B.3所示。可用下列类似检查。

B.5.3.2多处理器的分割方法

为了保证微控制器网络内软件组件足够的独立性，应确保无以下阻止安全功能正确执行的故障影响：

——对等通信失效（对等通信不可用）；

——意外消息重复（相同的消息被意外重复发送）；

——消息丢失（传送中消息丢失）；

——消息插入（接收方意外收到一个额外的消息，可理解为正确来源和目的地址）

——重复排序（传输过程中数据排序被更改，即收到的消息排序与发出时不一致）；

——消息损坏（在消息传输过程中一个或多个数据位被更改）；

——消息延时（信息被正确接收，但是并未按时接收）；

——数据总线的访问阻塞（故障节点不坚持预期使用的模式，导致服务的过度请求，从而减少了其他节点的可用性，例如，无现有数据的错误等待。）

——消息的持续传输，被称为“babbling idiot”（故障节点持续传递，从而影响整个总线的运行）

表B.1 微控制器内的方法和措施

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 故障影响  方法/措施 | 内存破坏 | 分区阻塞 | 处理器执行时间的错误分配 | 错误的对等通信 | I/O接口的破坏 |
| 通信检查 | | | | | |
| 1.明确的双向通信目标a | 无 | 无 | 无 | 中 | 无 |
| 2.严格的两个单向通信目标b | 无 | 无 | 无 | 中 | 无 |
| 3.对标识的识别和确认c | 无 | 无 | 无 | 高 | 无 |
| 4.异步数据通信d | 无 | 高 | 无 | 无 | 无 |
| 处理器执行时间的分配 | | | | | |
| 5.不基于优先级的调度e | 无 | 无 | 中 | 无 | 无 |
| 6.时间分片方法f | 无 | 无 | 高 | 无 | 无 |
| 系统资源分配 | | | | | |
| 7.内存保护措施g | 高 | 无 | 无 | 无 | 无 |
| 8.安全临界数据的检查h | 高/中 | 无 | 无 | 无 | 无 |
| 9.静态分析i | 中 | 无 | 无 | 无 | 无 |
| 10静态分配j | 中 | 中 | 中 | 中 | 中 |
| 在条目1和2中的分区之间的通信目标是诸如管道、消息队列及共享内存等资源。这些不应用于同步分区。  使用共享内存的两个分区之间的通信访问应是同步的。例如，使用信号量。  当使用消息队列时，设计应阻止阻塞读取的访问。  内存管理单元MMU允许虚拟地址空间概念。这能阻止一个分区的任务通过意外的写入而破坏另外一个任务的内存空间，因为每个分区有其本身的内存空间。MMU内存管理单元的使用要求操作系统支持该特性。应制定规定，以避免MMU被忽略。因此，任务以所谓的用户模式运行，且真实寻址位模式是不使用的。 | | | | | |
| a. 一个严格的双向通信目标用于两个分区之间各自的数据交换。  b. 两个严格的单向通信目标用于两个分区之间各自的数据交换。  c. 使用唯一的数字以识别对等通信和/或确认对等通信收到消息。  d. 在使用本项所述的异步数据通信时，通信本身没有完成的等待状态。  e.应平等的考虑分区在处理器中分配的执行时间，并分配给他们相同的优先级。关于处理器在本条目中的时间分配，由于引入中断，在每个处理器周期中必须有一些空闲时间/缓冲。  f. 时间分片方法基于预定的固定时间表来指定调度算法，以固定周期重复。处理器执行时间的分配，使用时间分片方法通过静态分配表产生。因此，对于每一项任务，在固定的时间点是预先确定的激活任务。时间分片方法的使用排除基于优先级的调度。  g. 所指的内存保护机制指定诸如MMU或MPU的处理器。  h. 通过额外措施验证含有安全关键数据的RAM区域。例如，这可以通过使用CRC或冗余存储来完成。这项措施的有效性在很大程度上取决于验证质量。  i. 规定适当的静态分析方法，用于复查访问包含有关安全数据的内存区域的代码块。  j. 在初始化过程中资源是静态分配的。 | | | | | |

微控制器2

图B.2 微控制网络范围内的若干分区

数据总线

微控制器1

微控制器2

内部数据总线

电子控制单元元

微控制器1

图B.3 多处理器ECU范围内的若干分区

B.5.3.3 多元处理器分区效力

应执行表B.2中列出的方法和措施，使所有适当的故障影响可以被足够有效的处理，以确保软件组件足够的独立性。

表B.2 微控制器网络范围内的方法和措施

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 故障影响  方法/措施 | | 对等通信失效 | 意外消息重复 | 消息丢失 | 消息插入 | 重新排序 | 信息破坏和/或伪消息 | 消息延时 | 数据总线阻塞访问 | 消息的持续传输 |
| 通信检查 | | | | | | | | | | |
| 1 | 活动消息保持 | 高 | 无 | 无 | 无 | 无 | 中 | 无 | 无 | 无 |
| 2 | 活动的计数器 | 无 | 高 | 中 | 中 | 无 | 无 | 无 | 无 | 中 |
| 3 | CRC | 无 | 无 | 无 | 无 | 无 | 高 | 无 | 无 | 无 |
|  | 序列号 | 无 | 高 | 高 | 高 | 高 | 无 | 无 | 无 | 中 |
| 4 | 消息重复 | 无 | 无 | 高 | 无 | 中 | 中 | 无 | 无 | 无 |
| 5 | 看门狗 | 高 | 无 | 中 | 高 |  |  | 高 |  | 高 |
| 总线分配 | | | | | | | | | | |
| 6 | 时间触发的数据总线 | 高 | 高 | 无 | 高 | 无 | 无 | 高 | 无 |  |
| 7 | 总线监护器 | 无 | 无 | 无 | 无 | 无 | 中 | 无 | 无 | 高 |
| 8 | 微时隙 | 无 | 无 | 无 | 无 | 无 | 无 | 中 | 无 | 高 |
| 使用CRC（条目 3）时，应考虑在总线系统中执行的CRC残留错误率可能是不充分的，在此情况下，推荐在应用程序层面使用附加的CRC。  阻塞访问是由连接到总线的微控制器引起的，由于其持续不断地访问总线，从而阻止其他微控制器访问总线。 | | | | | | | | | | |

参考文献

[1]GB/T 9480-2001农林拖拉机和机械、草坪和园艺动力机械使用说明书编写规则（ISO3600:1996，IDT）

[2]GB/T 19001 质量管理体系 要求（ISO 9001，IDT）

[3]GB/T 15706 机械安全设计通则风险评估与风险减少（ISO12100，IDT）

[4]GB/T16856-2008 （所有部分）机械安全风险评价（ISO14121:2007，IDT）

[5]GB/T 25392农业工程电气和电子设备对环境条件的耐久试验（/ISO 15003，IDT）

[6]GB/T18305-2016质量管理体系汽车生产件及相关服务件组织应用GB/T19001-2008的特别要求（ISO/TS16949,2009 IDT）

[7]GB/T 17626 电磁兼容试验和测量技术抗扰度试验总论（IEC 61000-4-1，IDT）

[8] EN 61496-1，*Safety of machinery — Electro-sensitive protective equipment — Part 1: General requirements and tests*