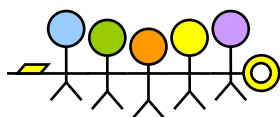


嵌入式软件测试指南

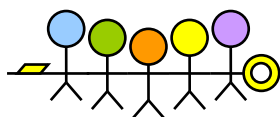
工作流程 · 核心技能 · 工具链 · 完整实战案例（AI 辅助版）

含被测 C 代码、单元测试、集成测试与系统测试全套脚本

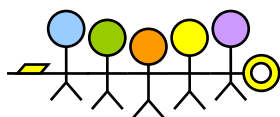


目 录

目 录.....	2
第一章 嵌入式软件测试概述.....	5
1.1 什么是嵌入式软件测试.....	5
1.2 与通用软件测试的六大差异.....	5
1.3 测试层次与 V 模型.....	6
1.4 AI 在嵌入式测试中的定位	6
第二章 嵌入式软件测试工作流程.....	8
2.1 全流程概览.....	8
2.2 阶段一：需求与风险分析（左移）	8
2.3 阶段二：静态分析与规范检查.....	9
2.4 阶段三到五：单元、集成、系统测试.....	9
2.5 阶段六：持续集成与回归.....	9
2.6 阶段七到八：缺陷管理与放行.....	10
第三章 核心工作技能.....	11
3.1 技能地图.....	11
3.2 用例设计基本功.....	11
3.3 可测性设计（DFT）能力	12
3.4 AI 协作能力	12
第四章 工具全景与选型.....	13
4.1 开源工具链（推荐起点）	13
4.2 商业工具链（认证与高安全场景）	14
4.3 编译器与调试工具.....	14
4.4 最小可用环境搭建命令.....	15
第五章 完整实战案例：温控风扇控制器.....	16



5.1 被测对象与需求.....	16
5.2 工程结构与可测性设计要点.....	16
5.3 被测 C 代码.....	17
5.3.1 硬件抽象层: hal_adc.h / hal_io.h.....	17
5.3.2 温度换算模块: temp_sensor	18
5.3.3 风扇控制状态机: fan_ctrl	19
5.3.4 应用调度层: app_controller	22
5.3.5 目标机入口: main.c.....	23
5.4 单元测试 (Ceedling + Unity + CMock)	24
5.4.1 Ceedling 配置 project.yml.....	24
5.4.2 test_temp_sensor.c —— 换算与错误处理	25
5.4.3 test_fan_ctrl.c —— 状态机与滞回	26
5.4.4 test_app_controller.c —— 应用层 (Mock HAL)	28
5.4.5 运行与结果.....	30
5.5 集成测试 (真实模块 + Fake 驱动)	31
5.6 系统测试.....	34
5.6.1 主机仿真固件 (SIL) + 串口风格 CLI.....	34
5.6.2 系统测试脚本: 驱动仿真固件 (pytest)	35
5.6.3 系统测试脚本: 真实硬件串口 (HIL)	38
5.6.4 Robot Framework 与 Renode 版本 (可选)	39
5.7 CI 流水线配置.....	40
第六章 AI 辅助嵌入式测试实战	42
6.1 分阶段用法矩阵.....	42
6.2 可直接复用的提示词模板.....	42



模板一：从需求生成用例.....	42
模板二：生成可编译的单元测试代码.....	42
模板三：覆盖率缺口补齐.....	43
6.3 风险与边界.....	43
第七章 工具使用指南与下载链接.....	44
7.1 单元测试与 Mock	44
7.2 静态分析与代码规范.....	44
7.3 覆盖率.....	45
7.4 编译器、调试与仿真.....	45
7.5 自动化与 CI.....	46
7.6 商业工具.....	46
7.7 缺陷与用例管理.....	47
附录.....	48
附录 A 嵌入式测试准入准出检查清单.....	48
附录 B 常见坑与规避.....	48
附录 C 参考标准与扩展阅读.....	48

第一章 嵌入式软件测试概述

1.1 什么是嵌入式软件测试

嵌入式软件测试，是针对运行在微控制器（MCU）、数字信号处理器（DSP）、片上系统（SoC）等资源受限硬件上的固件（Firmware）所开展的验证与确认活动。它既包含通用软件测试的全部内容——功能、性能、健壮性、安全性，还必须额外回答三个通用软件测试通常不关心的问题：

- 软件是否与真实物理世界（传感器、执行器、电源、电磁环境）正确交互？
- 在内存、算力、功耗都被严格约束的条件下，软件是否仍然满足实时性要求？
- 在异常输入、硬件老化、环境极端变化等场景下，软件是否进入可控的安全状态？

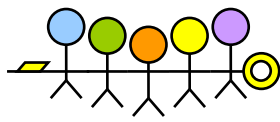
因此，嵌入式软件测试的完整目标可以概括为：在尽可能早的阶段，以尽可能低的成本，证明固件在其生命周期内所有可预见的工况下都行为正确、失效安全。

1.2 与通用软件测试的六大差异

理解差异是建立正确测试策略的前提。下表列出嵌入式测试相对于 Web / 桌面软件测试的典型特殊性，以及由此推导出的应对手段。

差异维度	具体表现	对测试的影响	应对手段
硬件强耦合	代码直接读写寄存器、中断、DMA	PC 上无法直接运行，测试依赖硬件	硬件抽象层（HAL）+ 打桩 / Mock，使逻辑层可在主机编译运行
可观测性差	无屏幕、无文件系统，只有 LED 和串口	故障复现与定位困难	SWO/ITM 跟踪、日志缓冲、断言、Tracealyzer/SystemView 等可视化工具
强实时约束	中断延迟、任务周期、最坏执行时间（WCET）有硬指标	功能正确但超时即失效	时序测试、WCET 分析、逻辑分析仪/示波器测量、RTOS trace
资源高度受限	几十 KB RAM、几百 KB Flash，无 MMU	测试代码、覆盖率插桩会撑爆目标机	主机侧（Host）执行测试，目标机只做关键回归；按模块分批插桩
交叉编译	在 x86 主机编译，在 ARM/RISC-V 运行	主机与目标机数据类型、对齐、字节序可能不同	静态断言检查类型宽度；主机与目标机双跑（Host/Target 双执行）
召回成本极高	汽车、医疗、工业设备召回代价巨大	缺陷必须左移，越晚发现成本越高	静态分析前置、单元测试强制覆盖率门禁、需求—用例双向追溯

一句话总结：嵌入式测试的核心矛盾是「硬件依赖」与「自动化、可重复、可度量」之间的矛盾，而解决这个矛盾的钥匙就是可测性设计（Design for Testability）。



1.3 测试层次与 V 模型

嵌入式领域普遍采用 V 模型组织验证活动。V 的左侧是需求分解与设计，右侧是与之对应的验证层级，二者一一对应，形成天然的可追溯关系。

落地到测试执行，通常划分为五个层次，越靠下越接近硬件、成本越高：

1. 静态分析与代码规范检查：不运行程序，通过扫描源码发现潜在缺陷和规范违反（MISRA C、CERT C）。成本最低，应最先做。
2. 单元测试：以函数/模块为最小单位，在主机上编译运行，依赖项全部用 Mock/桩替代。要求语句、分支、MC/DC 覆盖率达标。
3. 集成测试：把若干真实模块组合起来，验证接口契约、数据流、时序与错误处理链路。通常只 Mock 最底层的 HAL。
4. 系统测试：在目标机、仿真器（QEMU/ReNode）或硬件在环（HIL）台架上，验证完整需求，包含启动、通信、功耗、老化、EMC 相关场景。
5. 验收与现场验证：小批量试产、现场试运行、OTA 灰度，验证真实工况下的长期稳定性。

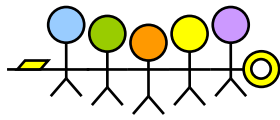
与之配套的三个常见执行环境缩写：

- SIL（Software-In-the-Loop）：纯主机仿真，被测代码在 PC 上运行，速度最快，用于单元/集成测试。
- PIL（Processor-In-the-Loop）：代码编译到目标芯片上运行，主机通过调试通道喂数据、取结果，用于验证编译器与芯片相关行为。
- HIL（Hardware-In-the-Loop）：接入真实或仿真的传感器/执行器台架，用于系统级与回归测试。

1.4 AI 在嵌入式测试中的定位

大模型（LLM）不会替代测试工程师，但它显著改变了工作量分布。在嵌入式测试这个「脚本多、样板代码多、协议繁琐、日志冗长」的领域，AI 的增益尤为明显：

活动	传统耗时	AI 辅助后	可靠性要求
编写 Mock / 测试脚手架	高（机械劳动占比大）	自动生成初稿，人工校正	必须编译通过并人工核对断言
设计边界值、异常用例	依赖个人经验	按需求条目批量生成候选清单	人工评审，AI 易遗漏领域约束
阅读芯片手册/协议文档	数小时	分钟级问答与要点提炼	必须与原文核对寄存器地址与时序
分析崩溃日志/断言信息	依赖资深工程师	快速给出候选根因与排查路径	结论必须用实验证实
补齐覆盖率缺口	逐个分支人工补	按未覆盖分支自动生成用例	需人工确认期望值是否合理



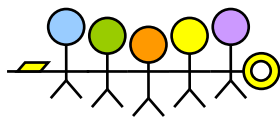
生成 CI 脚本/报告

易出错的重复劳动

一次性生成并复用

需在流水线中实测验证

贯穿全文的原则：AI 负责「产量」，工程师负责「判断」。凡是涉及安全需求、硬件边界、认证证据的内容，AI 输出只能作为草稿，不能作为结论。



第二章 嵌入式软件测试工作流程

2.1 全流程概览

一套可落地的嵌入式测试流程由 8 个阶段构成。每个阶段都应有明确的输入、活动、产出和准出条件（Exit Criteria），否则很容易退化成「凭感觉测一测」。

#	阶段	主要活动	关键产出	典型工具	AI 辅助点
1	需求与风险分析	需求可测性评审、失效模式分析（FMEA/FTA）、定义安全等级	可测试需求条目、风险清单、测试重点	DOORS/Polarion、Jira、FMEA 表	自动检查需求二义性、生成评审问题清单
2	静态分析与规范检查	编译告警清零、MISRA/CERT 扫描、复杂度度量	静态分析报告、圈复杂度、违规清单	Cppcheck、PC-lint Plus、clang-tidy、SonarQube	解释违规原因、批量给出修改建议
3	单元测试	模块隔离、Mock 依赖、边界/异常/状态迁移用例	测试代码、覆盖率报告（语句/分支/MC-DC）	Ceedling(Unity+CMock)、GoogleTest、gcovr	生成测试骨架与参数化用例初稿
4	集成测试	模块间接口、数据流、错误传播、时序	集成测试用例、接口契约验证结果	同上 + 自定义 Harness、Robot Framework	根据接口定义生成契约测试
5	系统测试	目标机/HIL 台架上的端到端场景、性能、可靠性	系统测试报告、缺陷清单、放行建议	pytest、pyserial、CANoe、Renode、QEMU	生成脚本框架、解析设备日志
6	回归与持续集成	提交即构建、即测试、即出覆盖率门禁	流水线状态、趋势报告	Jenkins、GitLab CI、GitHub Actions	自动生成/修复 CI 配置，定位失败用例
7	缺陷管理	缺陷定位、复现、修复验证、防复发	缺陷单、根因分析、回归用例	Jira、禅道、GitLab Issues	日志根因推断、相似缺陷聚类
8	报告与放行	覆盖率汇总、需求追溯矩阵、放行评审	测试报告、追溯矩阵、签核记录	TestRail、Polarion、自研报告脚本	自动汇总数据、生成报告初稿

2.2 阶段一：需求与风险分析（左移）

这是最容易被跳过、但收益最高的阶段。对每一条需求，必须确认它「可测」——即存在明确的输入、可判定的期望输出，且期望值可以用仪器或代码断言观测到。

需求可测性检查清单：

- 是否包含可量化的阈值？（避免「响应迅速」，改为「从温度越限到风扇全速不超过 500 ms」）
- 是否定义了异常与失效后的期望行为？（传感器开路时，风扇应保持全速并点亮报警）
- 是否明确了精度与容差？（温度测量误差 $\pm 1.5\text{ }^{\circ}\text{C}$ ）

- 是否存在隐含的物理约束？（PWM 频率必须在 20 kHz 以上以避免可听噪声）

风险分析推荐用简化 FMEA：对每个功能列出「失效模式 → 影响 → 严重度 → 现有检测手段 → 补充测试」。AI 在这一步很适合做「反方向检查」：把需求喂给它，要求它列出所有可能的歧义、缺失的异常分支、以及可能被误用的接口。

2.3 阶段二：静态分析与规范检查

在写任何测试用例之前，先让代码通过静态检查。这一步的投入产出比最高，因为大量缺陷（空指针解引用、数组越界、未初始化变量、有符号/无符号混用、volatile 缺失）在运行前就能被发现。

推荐的准入门禁（可作为 CI 的失败条件）：

- 编译告警级别 -Wall -Wextra -Werror，所有告警清零。
- MISRA C:2012 强制规则（Required/Mandatory）零违反；建议规则（Advisory）需有书面豁免。
- 单函数圈复杂度 ≤ 10 ；超过则需重构或补充论证。
- 禁止动态内存分配（安全关键项目常见要求）、禁止递归（无 MMU 的栈空间受限）。

2.4 阶段三到五：单元、集成、系统测试

三个层次的分工原则可以概括为「隔离度递减，真实性递增」：

- 单元测试：只测一个模块，其余全部 Mock。测的是逻辑正确性，尤其是分支与边界。
- 集成测试：真实模块 + 真实数据流，只 Mock 最底层 HAL。测的是接口契约与错误传播。
- 系统测试：真实固件 + 真实/仿真硬件。测的是需求满足度与时序性能。

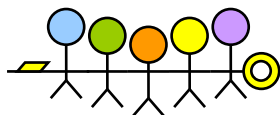
每一层的准出条件示例：

层次	覆盖率/通过率门禁（示例）	缺陷放行标准
单元测试	语句 100%，分支 $\geq 95\%$ ；安全等级高时要求 MC/DC 100%	不允许遗留致命/严重缺陷
集成测试	接口调用组合覆盖 100%，错误注入用例 100% 通过	接口类缺陷清零
系统测试	需求覆盖率 100%，全部用例通过；关键场景连续 72 h 无异常	不允许遗留与安全相关的缺陷

2.5 阶段六：持续集成与回归

嵌入式 CI 的特殊之处在于「双轨执行」：同一份测试代码，既要在主机（Host）上快速跑一遍作为门禁，也要定期在目标机/仿真器上跑一遍作为可信验证。

推荐的流水线分层：



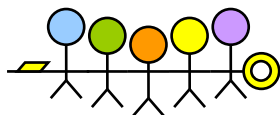
6. 提交级 (<3 分钟)：编译 + 静态分析 + 主机单元测试 + 覆盖率门禁。
7. 合并级 (<15 分钟)：集成测试 + 目标机编译 + 固件尺寸检查 (Flash/RAM 占用对比基线)。
8. 夜间级 (数小时)：HIL 台架系统测试 + 老化/压力测试 + 静态分析全量扫描。

一个实用技巧：把固件的 Flash/RAM 占用、栈最大使用深度也纳入 CI 度量。资源溢出在嵌入式项目中是高频回归缺陷，且往往在集成了新功能后才暴露。

2.6 阶段七到八：缺陷管理与放行

缺陷单应至少包含：复现环境（固件版本号、硬件版本、编译器版本）、最小复现步骤、期望与实际、日志/波形附件、影响分析。嵌入式领域特别强调「环境三要素」（固件版本 + 硬件版本 + 编译器版本），缺一个都可能导致无法复现。

放行评审的核心交付物是需求追溯矩阵：每一条需求 → 对应的测试用例编号 → 执行结果 → 覆盖率证据。安全标准（ISO 26262、DO-178C、IEC 61508）审计时，这张表是必查项。



第三章 核心工作技能

3.1 技能地图

能力域	具体技能点	熟练度要求
语言与系统	C 语言（指针、位操作、volatile、结构体对齐、定点运算）、C++ 基础、汇编基础、编译链接与启动流程、Makefile/CMake、链接脚本与内存布局	能独立定位编译/链接/运行时间问题
硬件与体系结构	MCU 内核（ARM Cortex-M/RISC-V）、中断与异常、DMA、时钟树、看门狗、Flash/RAM 特性、常见总线（UART/SPI/I2C/CAN/RS-485）	能读懂原理图与芯片手册关键章节
测试工程	用例设计方法、覆盖率度量与解读、Mock/桩/打桩技术、错误注入、边界与时序测试、测试自动化框架	能独立设计模块级测试方案
仪器与调试	示波器、逻辑分析仪、万用表、可编程电源/电子负载、J-Link/ST-Link、GDB、SEGGER RTT/SWO、总线分析仪（CANoe/USB 分析仪）	能独立抓取并解读波形与总线报文
流程与标准	V 模型、ASPICE、ISO 26262 / DO-178C / IEC 61508、MISRA C、需求追溯、变更与配置管理	理解证据要求，能产出合规文档
脚本与自动化	Python（pytest、pyserial、paramiko）、Shell、Robot Framework、CI 配置（YAML）、数据处理与可视化	能独立搭建自动化台架
AI 协作	提示词设计、结果校验、知识库/RAG 使用、代码生成的边界认知	能用 AI 提效且不引入错误结论

3.2 用例设计基本功

嵌入式测试中最常用、也最有效的四类用例设计方法：

方法	适用场景	嵌入式中的典型例子
等价类划分	输入域可分区	ADC 原始码分「低于下限 / 正常区间 / 高于上限 / 恰好边界」四类
边界值分析	阈值判断密集	温度阈值 45.0 ℃：取 44.9 / 45.0 / 45.1 ℃ 三档，另加滞回回差边界
状态迁移	存在状态机	风扇 OFF→LOW→MID→HIGH→FAULT，覆盖所有迁移边 + 非法迁移 + 自迁移
错误推测/注入	异常处理路径	ADC 返回负值、传感器断线、堵转转速为 0、I2C NACK、栈溢出

此外还有两个嵌入式专属的高价值手段：

- 故障注入（Fault Injection）：在代码里预留注入点（编译期宏或运行时钩子），主动制造硬件故障，验证安全状态是否达成。仅靠真实硬件很难稳定复现这类场景。
- 长时间压力与老化：72 h 连续运行、随机掉电、看门狗复位循环，用于暴露内存泄漏、计数器溢出、状态机卡死等「慢性」缺陷。

3.3 可测性设计 (DFT) 能力

这是区分「会写测试」和「能做好嵌入式测试」的分水岭。写代码时就要为测试留口子：

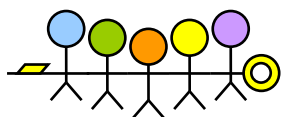
- 硬件访问收敛到 HAL：业务逻辑不直接读写寄存器，HAL 是唯一的硬件边界，也是唯一的 Mock 边界。
- 依赖注入而非硬编码：模块通过结构体指针接收配置与依赖，而不是在内部调用全局函数。
- 状态与输入显式化：把「当前时间」「传感器值」作为函数参数传入，而不是在函数内部读全局时钟——这样测试才能精确控制时序。
- 错误码统一且可判：每个模块定义明确的正/负返回值语义，避免用「魔数」表示错误。
- 预留观测接口：版本字符串、状态快照结构体、运行计数器，便于系统测试断言内部状态。

3.4 AI 协作能力

把 AI 用好的关键是「给足上下文、限定输出形态、强制可验证」。一个高质量的嵌入式测试提示词通常包含五要素：

- 角色与任务：你是资深嵌入式测试工程师，请为以下 C 模块设计单元测试。
- 完整上下文：贴入头文件、接口契约、配置参数含义、编译器与目标平台。
- 输出形态约束：使用 Unity + CMock 语法，每个用例一个 TEST 函数，包含 setUp/tearDown。
- 覆盖要求：覆盖所有分支、边界值 ± 1 、错误返回路径，并说明每条用例对应的需求编号。
- 验证方式：给出编译命令与期望通过结果，禁止使用未定义行为。

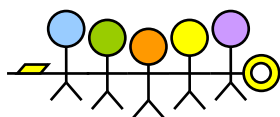
三条不可逾越的红线：① AI 生成的断言期望值必须人工核算，它可能「编造」一个看起来合理但错误的数值；② 涉及安全需求与认证证据的内容不得直接采信；③ 不得把未公开的项目代码、芯片手册、缺陷详情上传至外部服务。



第四章 工具全景与选型

4.1 开源工具链（推荐起点）

类别	工具	作用	许可	获取方式
单元测试框架	Unity	极轻量 C 单元测试框架，仅几个 .c/.h，适合裸机与深度嵌入式	MIT	github.com/ThrowTheSwitch/Unity
Mock 生成	CMock	根据头文件自动生成 Mock（打桩）代码，支持参数校验与返回值队列	MIT	github.com/ThrowTheSwitch/CMock
构建/测试系统	Ceedling	整合 Unity + CMock + CException 的 Ruby/Rake 构建与测试系统	MIT	<code>gem install ceedling</code>
单元测试框架	CppUTest	C/C++ 单元测试框架，自带内存泄漏检测，xUnit 风格	BSD	cpputest.github.io
单元测试框架	GoogleTest + GoogleMock	C++ 主流框架，适合有 C++ 组件的项目	BSD-3	github.com/google/googletest
静态分析	Cppcheck	开源 C/C++ 静态分析，支持 MISRA 规则集（需附加规则文件）	GPL-3	cppcheck.sourceforge.io
静态分析	clang-tidy / Clang 静态分析器	编译器前端级检查，可扩展自定义检查项	Apache-2	clang.llvm.org/extra/clang-tidy
质量平台	SonarQube + SonarQube for C/C++	质量门禁、趋势追踪、代码异味可视化	社区版 LGPL	sonarsource.com
覆盖率	gcov + gcovr / lcov	GCC 内置覆盖率 + 报告生成，支持语句/分支与 HTML 报告	GPL/BSD	<code>pip install gcovr</code>
覆盖率	llvm-cov	Clang 生态覆盖率工具	Apache-2	随 LLVM 发布
仿真	QEMU	系统级模拟器，支持 ARM/RISC-V 等，可跑裸机固件	GPL-2	qemu.org/download
仿真	Renode	面向 IoT/SoC 的仿真框架，支持外设建模与多节点网络，集成 Robot Framework	Apache-2	renode.io/builds.renode.io
调试/烧录	OpenOCD + GDB	开源调试服务器，配合 ST-Link/J-Link 使用	GPL-2	openocd.org
自动化框架	pytest + pyserial	Python 驱动的串口/仪器自动化测试	MIT/BSD	<code>pip install pytest pyserial</code>
自动化框架	Robot Framework	关键字驱动的验收测试框架，Renode 原生支持	Apache-2	robotframework.org
CI	GitLab CI / GitHub Actions	流水线编排，配合 Docker 镜像固定工具链版本	—	docs.gitlab.com/features/actions



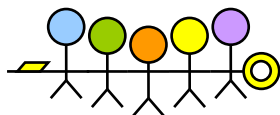
4.2 商业工具链（认证与高安全场景）

工具	厂商	核心能力	适用标准
VectorCAST/C++	Vector	自动生成测试用例与桩、语句/分支/MC-DC 覆盖率、主机+目标机双执行、需求追溯	ISO 26262、DO-178C、IEC 61508、EN 50128、IEC 62304
Parasoft C/C++test	Parasoft	单元测试 + 静态分析 + 覆盖率 + 合规报告一体化，深度集成 CI 与构建产物	MISRA C/C++、CERT C、ISO 26262、DO-178C
LDRA Testbed / TBvision	LDRA	静态/动态分析、需求与代码与测试的双向追溯视图（TBvision）	MISRA、ISO 26262、DO-178C、IEC 61508
Cantata	QA Systems	从 C/C++ 接口自动生成测试资产、桩与可执行 Harness	ISO 26262、DO-178C
TESSY	Hitex	嵌入式专用单元测试，自动生成目标机驱动与依赖替换	ISO 26262、IEC 61508
PC-lint Plus	Gimpel	业界事实标准级深度静态分析，MISRA 覆盖全面	MISRA C/C++
BullseyeCoverage	Bullseye	C/C++ 覆盖率，对嵌入式交叉编译支持成熟	DO-178C、ISO 26262
Coverity	Synopsys	大规模代码库深度静态分析，缺陷检出率高	MISRA、CERT、安全编码

选型建议：项目未做认证、团队规模小 → 优先 Unity + CMock + Ceedling + Cppcheck + gcovr，零成本且完全够用；项目需要功能安全认证 → 直接上 VectorCAST / LDRA / Parasoft，虽然昂贵，但其自带的工具鉴定包（TQP）能省下大量认证工作量。

4.3 编译器与调试工具

工具	用途	获取方式
Arm GNU Toolchain (arm-none-eabi-gcc)	ARM Cortex-M/R/A 交叉编译，含 gcc/gdb/newlib	developer.arm.com/downloads/-/arm-gnu-toolchain-downloads
RISC-V GNU Toolchain	RISC-V 交叉编译	github.com/riscv-collab/riscv-gnu-toolchain
IAR Embedded Workbench / Arm Keil MDK	商业编译器与 IDE，代码密度与调试体验优秀，自带 C-STAT 静态分析	iar.com / keil.arm.com
SEGGER J-Link + Ozone + SystemView	调试探针、独立调试器、RTOS 运行时可视化	segger.com/downloads
Percepio Tracealyzer	RTOS 与裸机运行时行为可视化，定位时序与优先级问题	percepio.com
Lauterbach TRACE32	高端调试与跟踪，支持多核、复杂 SoC	lauterbach.com



Vector CANoe/CANalyzer	CAN/LIN/车载以太网总线仿真与分析	vector.com
---------------------------	----------------------	------------

4.4 最小可用环境搭建命令

以下命令可在一台干净的 Ubuntu 主机上搭建完整的开源嵌入式测试环境，复制即可执行：

```
# 1) 基础编译与调试工具
sudo apt-get update
sudo apt-get install -y build-essential cmake git python3 python3-pip \
    gcc-arm-none-eabi gdb-multiarch openocd

# 2) 主机侧单元测试三件套 (Unity + CMock + Ceedling)
sudo apt-get install -y ruby ruby-dev
gem install ceedling --no-document          # Unity 与 CMock 随 Ceedling 一并安装

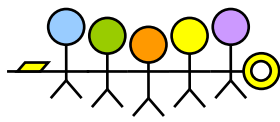
# 3) 覆盖率与静态分析
pip3 install gcovr                          # 覆盖率报告 (也可 pip install gcovr==8.x)
sudo apt-get install -y cppcheck clang-tidy

# 4) 系统测试自动化
pip3 install pytest pyserial robotframework pexpect

# 5) 验证安装
ceedling version
gcovr --version
cppcheck --version
```

Windows 用户建议直接使用 Ceedling 官方 Docker 镜像，可省去 Ruby 环境配置的麻烦：

```
docker pull throwtheswitch/madsciencelab
docker run -it --rm -v ${PWD}:/work -w /work throwtheswitch/madsciencelab bash
```



第五章 完整实战案例：温控风扇控制器

5.1 被测对象与需求

本章提供一个可直接运行的完整案例：一款温控风扇控制器（Thermal Fan Controller）。被测代码全部为 C 语言，配套单元测试、集成测试与系统测试脚本，可在普通 PC 上完整复现。

本案例的全部代码均已在 Ubuntu + GCC + Python 环境下实测通过：单元测试 24 例、集成测试 5 例、系统测试 9 例，共 38 个用例全部通过，被测源码在 -Wall -Wextra -Werror 下零告警。此外还做了变异测试：注入 21 个变异体（阈值比较符、回差、占空比映射、堵转判定、错误返回等），最终变异得分 $21/21 = 100\%$ 。第一轮曾有一个变异体存活——「堵转统计不判断占空比」，暴露出「OFF 档下转速为 0 持续 2 s 会误报堵转」的真实缺陷，补充 test_no_false_stall_when_fan_off 后才达到 100%。这说明：覆盖率 100% 不等于测试有效，变异测试才是判断测试能否抓到缺陷的手段。

需求条目（作为后续所有用例的追溯源头）：

编号	需求描述	验证层级
REQ-01	以 100 ms 为周期采集 NTC 温度，测量范围 $-40.0\text{ }^{\circ}\text{C} \sim +150.0\text{ }^{\circ}\text{C}$ ，误差 $\leq \pm 1.5\text{ }^{\circ}\text{C}$	单元 + 系统
REQ-02	温度 $\geq 45.0\text{ }^{\circ}\text{C}$ 风扇低速启动（占空比 30%）	单元 + 集成
REQ-03	温度 $\geq 60.0\text{ }^{\circ}\text{C}$ 中速（60%）， $\geq 75.0\text{ }^{\circ}\text{C}$ 高速（100%）	单元 + 集成
REQ-04	档位切换需有 $5.0\text{ }^{\circ}\text{C}$ 回差（滞回），避免在阈值附近频繁跳变	单元
REQ-05	温度 $\geq 90.0\text{ }^{\circ}\text{C}$ 进入过温保护并锁存，风扇强制 100%，点亮报警灯；温度降至 $85.0\text{ }^{\circ}\text{C}$ 以下才解除	单元 + 集成 + 系统
REQ-06	风扇已命令转动但转速为 0 且持续 2 s，判定为堵转，进入故障态	单元 + 集成
REQ-07	ADC 返回硬件错误或采样值越界时，进入故障安全态（风扇 100% + 报警灯），错误计数加一	单元 + 集成 + 系统
REQ-08	连续运行 30 min 无状态机卡死、无计数器溢出	系统

5.2 工程结构与可测性设计要点

```
thermal_fan/
├── CMakeLists.txt           # 主机仿真 / 覆盖率构建
├── project.yml              # Ceedling 单元测试配置
├── src/
│   ├── hal/
│   │   ├── hal_adc.h / hal_adc.c  # ADC 读取（唯一的硬件边界a）
│   │   ├── hal_io.h / hal_io.c    # PWM 输出、转速读取、报警灯
│   │   └── middleware/
│   │       ├── temp_sensor.h / .c  # 原始码 -> 毫摄氏度
│   │       └── fan_ctrl.h / .c     # 温度 -> 档位状态机 + 占空比
│   └── app/
│       └── app_controller.h / .c   # 周期调度：采样 -> 决策 -> 输出
```

└─ main.c	# 目标机入口
└─ test/	
└─ test_temp_sensor.c	# 单元测试
└─ test_fan_ctrl.c	
└─ test_app_controller.c	
└─ test_integration.c	# 集成测试 (用 Fake 而非 Mock)
└─ support/	
└─ fake_hal_adc.c	# 可控的 ADC 仿真实现
└─ fake_hal_io.c	# 记录输出历史的 IO 仿真实现
└─ tests/system/	
└─ sim_main.c	# 主机仿真固件 (带串口风格 CLI)
└─ test_system_host.py	# 系统测试: 驱动仿真固件
└─ test_system_hil.py	# 系统测试: 驱动真实硬件串口
└─ robot/system_test.robot	# Robot Framework 版本

本案例遵循了三条可测性设计原则:

- 硬件访问全部收敛到 hal_* 接口, 业务层 (middleware / app) 不含任何寄存器操作, 因此可以在 PC 上原样编译运行。
- 配置以结构体注入 (temp_sensor_cfg_t、fan_cfg_t), 而非编译期常量, 测试时可自由替换阈值。
- 状态与输出可通过结构体字段和 app_status_json() 观测, 系统测试无需剥开内部状态。

5.3 被测 C 代码

5.3.1 硬件抽象层: hal_adc.h / hal_io.h

```
/* ----- src/hal/hal_adc.h ----- */
#ifndef HAL_ADC_H
#define HAL_ADC_H

#include <stdint.h>

#define HAL_ADC_CH_NTC    0u    /* NTC 温度传感器所在通道 */

/**
 * 读取指定通道的 ADC 原始码 (12 bit)。
 * @return 0 .. 4095 为有效采样值; 负值表示硬件故障 (超时/校准失败)
 */
int32_t hal_adc_read_raw(uint8_t channel);

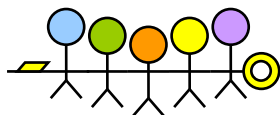
#endif /* HAL_ADC_H */

/* ----- src/hal/hal_io.h ----- */
#ifndef HAL_IO_H
#define HAL_IO_H

#include <stdint.h>
#include <stdbool.h>

/** 设置风扇 PWM 占空比, 0 .. 100 (百分比) */
void    hal_pwm_set_duty(uint8_t duty_pct);

/** 读取风扇转速反馈 (tach), 单位 RPM; 0 表示未转动或无反馈 */
```



```
uint16_t hal_tach_get_rpm(void);

/** 控制报警指示灯 */
void      hal_led_set_alarm(bool on);

#endif /* HAL_IO_H */
```

5.3.2 温度换算模块: temp_sensor

```
/* ----- src/middleware/temp_sensor.h ----- */
#ifndef TEMP_SENSOR_H
#define TEMP_SENSOR_H

#include <stdint.h>
#include <stddef.h> /* NULL */

#define TEMP_SENSOR_OK          0
#define TEMP_SENSOR_ERR_HW      (-1) /* 硬件/驱动层故障 */
#define TEMP_SENSOR_ERR_RAW     (-2) /* 原始码超出有效区间（开路/短路） */

typedef struct {
    uint16_t raw_min; /* 有效原始码下限，低于此值判定为开路 */
    uint16_t raw_max; /* 有效原始码上限，高于此值判定为短路 */
    int32_t  offset_raw; /* 0 °C 对应的 ADC 码 */
    int32_t  mc_per_raw; /* 每个 LSB 对应的温度，单位 0.001 °C */
    int32_t  bias_mc; /* 标定偏置，单位 0.001 °C */
} temp_sensor_cfg_t;

typedef struct {
    temp_sensor_cfg_t cfg;
    uint16_t          last_raw;
    int32_t            last_mc;
    uint32_t            err_count;
} temp_sensor_t;

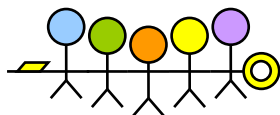
void      temp_sensor_init(temp_sensor_t *self, const temp_sensor_cfg_t *cfg);
int32_t temp_sensor_convert(const temp_sensor_cfg_t *cfg, uint16_t raw);
int32_t temp_sensor_read(temp_sensor_t *self);

#endif /* TEMP_SENSOR_H */

/* ----- src/middleware/temp_sensor.c ----- */
#include "temp_sensor.h"
#include "hal_adc.h"

void temp_sensor_init(temp_sensor_t *self, const temp_sensor_cfg_t *cfg)
{
    if ((self == NULL) || (cfg == NULL)) {
        return; /* MISRA: 防御式编程，静默返回 */
    }
    self->cfg      = *cfg;
    self->last_raw  = 0u;
    self->last_mc   = 0;
    self->err_count = 0u;
}

int32_t temp_sensor_convert(const temp_sensor_cfg_t *cfg, uint16_t raw)
{
    if (cfg == NULL) {
        return TEMP_SENSOR_ERR_HW;
    }
    if (raw < cfg->raw_min || raw > cfg->raw_max) {
        return TEMP_SENSOR_ERR_RAW;
    }
    int32_t mc = (int32_t)raw - cfg->offset_raw;
    return mc / cfg->mc_per_raw + cfg->bias_mc;
}
```



```
}
if ((raw < cfg->raw_min) || (raw > cfg->raw_max)) {
    return TEMP_SENSOR_ERR_RAW; /* REQ-07 */
}
return ((int32_t)raw - cfg->offset_raw) * cfg->mc_per_raw + cfg->bias_mc;
}

int32_t temp_sensor_read(temp_sensor_t *self)
{
    int32_t raw;
    int32_t mc;

    if (self == NULL) {
        return TEMP_SENSOR_ERR_HW;
    }

    raw = hal_adc_read_raw(HAL_ADC_CH_NTC);
    if (raw < 0) {
        self->err_count++;
        return TEMP_SENSOR_ERR_HW;
    }

    mc = temp_sensor_convert(&self->cfg, (uint16_t)raw);
    if (mc == TEMP_SENSOR_ERR_RAW) {
        self->err_count++;
        return TEMP_SENSOR_ERR_RAW;
    }

    self->last_raw = (uint16_t)raw;
    self->last_mc = mc;
    return mc; /* 单位: 0.001 °C */
}
```

5.3.3 风扇控制状态机: fan_ctrl

```
/* ----- src/middleware/fan_ctrl.h ----- */
#ifndef FAN_CTRL_H
#define FAN_CTRL_H

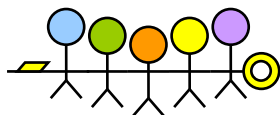
#include <stdint.h>
#include <stddef.h> /* NULL */

#define FAN_CTRL_PERIOD_MS 100u /* 控制周期, 与调度周期一致 */

#define FAN_DUTY_OFF 0u
#define FAN_DUTY_LOW 30u
#define FAN_DUTY_MID 60u
#define FAN_DUTY_HIGH 100u

typedef enum {
    FAN_STATE_OFF = 0,
    FAN_STATE_LOW,
    FAN_STATE_MID,
    FAN_STATE_HIGH,
    FAN_STATE_FAULT
} fan_state_t;

typedef struct {
    int32_t on_mc; /* 启动阈值 (REQ-02) */
    int32_t mid_mc; /* 中速阈值 (REQ-03) */
    int32_t high_mc; /* 高速阈值 (REQ-03) */
    int32_t crit_mc; /* 过温保护阈值 (REQ-05) */
}
```



```
int32_t hyst_mc;          /* 滞回回差 (REQ-04) */
uint32_t stall_timeout_ms; /* 堵转判定时长 (REQ-06) */
} fan_cfg_t;

typedef struct {
    fan_cfg_t  cfg;
    fan_state_t state;
    uint8_t    duty_pct;
    uint32_t    stall_ms; /* 连续无转速累计时长 */
} fan_ctrl_t;

void fan_ctrl_init(fan_ctrl_t *self, const fan_cfg_t *cfg);
fan_state_t fan_ctrl_update(fan_ctrl_t *self, int32_t temp_mc, uint16_t rpm);
uint8_t fan_ctrl_get_duty(const fan_ctrl_t *self);
fan_state_t fan_ctrl_get_state(const fan_ctrl_t *self);

#endif /* FAN_CTRL_H */

/* ----- src/middleware/fan_ctrl.c ----- */
#include "fan_ctrl.h"

void fan_ctrl_init(fan_ctrl_t *self, const fan_cfg_t *cfg)
{
    if ((self == NULL) || (cfg == NULL)) {
        return;
    }
    self->cfg = *cfg;
    self->state = FAN_STATE_OFF;
    self->duty_pct = FAN_DUTY_OFF;
    self->stall_ms = 0u;
}

uint8_t fan_ctrl_get_duty(const fan_ctrl_t *self)
{
    return (self == NULL) ? FAN_DUTY_OFF : self->duty_pct;
}

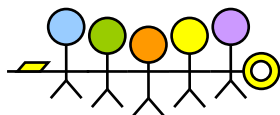
fan_state_t fan_ctrl_get_state(const fan_ctrl_t *self)
{
    return (self == NULL) ? FAN_STATE_OFF : self->state;
}

fan_state_t fan_ctrl_update(fan_ctrl_t *self, int32_t temp_mc, uint16_t rpm)
{
    const fan_cfg_t *c;

    if (self == NULL) {
        return FAN_STATE_OFF;
    }
    c = &self->cfg;

    /* (1) 过温保护: 锁存, 必须降温到 crit - hyst 以下才退出 (REQ-05) */
    if (self->state == FAN_STATE_FAULT) {
        self->duty_pct = FAN_DUTY_HIGH;
        if (temp_mc <= (c->crit_mc - c->hyst_mc)) {
            self->state = FAN_STATE_HIGH;
            self->stall_ms = 0u;
        }
        return self->state;
    }

    /* (2) 温度分级, 带滞回 (REQ-02 / REQ-03 / REQ-04) */
```



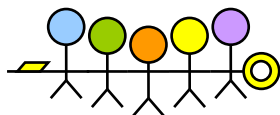
```
switch (self->state) {
case FAN_STATE_OFF:
    if (temp_mc >= c->on_mc) {
        self->state = FAN_STATE_LOW;
    }
    break;
case FAN_STATE_LOW:
    if (temp_mc >= c->mid_mc) {
        self->state = FAN_STATE_MID;
    } else if (temp_mc <= (c->on_mc - c->hyst_mc)) {
        self->state = FAN_STATE_OFF;
    } else {
        /* 滞回区间内保持原状态 */
    }
    break;
case FAN_STATE_MID:
    if (temp_mc >= c->high_mc) {
        self->state = FAN_STATE_HIGH;
    } else if (temp_mc <= (c->mid_mc - c->hyst_mc)) {
        self->state = FAN_STATE_LOW;
    } else {
        /* 保持 */
    }
    break;
case FAN_STATE_HIGH:
    if (temp_mc <= (c->high_mc - c->hyst_mc)) {
        self->state = FAN_STATE_MID;
    }
    break;
default:
    self->state = FAN_STATE_OFF;
    break;
}

if (temp_mc >= c->crit_mc) {
    self->state = FAN_STATE_FAULT;
}

/* (3) 档位 -> 占空比映射 */
switch (self->state) {
case FAN_STATE_OFF:    self->duty_pct = FAN_DUTY_OFF; break;
case FAN_STATE_LOW:   self->duty_pct = FAN_DUTY_LOW; break;
case FAN_STATE_MID:   self->duty_pct = FAN_DUTY_MID; break;
case FAN_STATE_HIGH:  self->duty_pct = FAN_DUTY_HIGH; break;
case FAN_STATE_FAULT: self->duty_pct = FAN_DUTY_HIGH; break;
default:               self->duty_pct = FAN_DUTY_OFF; break;
}

/* (4) 堵转检测: 已命令转动但转速为 0 且持续超时 (REQ-06) */
if ((self->duty_pct > 0u) && (rpm == 0u)) {
    self->stall_ms += FAN_CTRL_PERIOD_MS;
    if (self->stall_ms >= c->stall_timeout_ms) {
        self->state = FAN_STATE_FAULT;
        self->duty_pct = FAN_DUTY_HIGH;
    }
} else {
    self->stall_ms = 0u;
}

return self->state;
}
```



5.3.4 应用调度层: app_controller

```
/* ----- src/app/app_controller.h ----- */
#ifndef APP_CONTROLLER_H
#define APP_CONTROLLER_H

#include <stdint.h>
#include "temp_sensor.h"
#include "fan_ctrl.h"

#define APP_OK          0
#define APP_ERR_SENSOR  (-1) /* ADC 硬件故障 */
#define APP_ERR_RANGE   (-2) /* 采样值越界 */

typedef struct {
    temp_sensor_t sensor;
    fan_ctrl_t    fan;
    int32_t       temp_mc;
    uint16_t      rpm;
    uint8_t       duty_pct;
    fan_state_t   state;
    uint32_t      err_count;
    uint32_t      tick_count;
} app_t;

void      app_init(app_t *app, const temp_sensor_cfg_t *scfg, const fan_cfg_t *fcfg);
int32_t   app_tick(app_t *app);
const char *app_status_json(const app_t *app, char *buf, uint32_t len);

#endif /* APP_CONTROLLER_H */

/* ----- src/app/app_controller.c ----- */
#include "app_controller.h"
#include "hal_io.h"
#include <stdio.h>

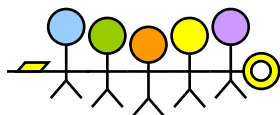
void app_init(app_t *app, const temp_sensor_cfg_t *scfg, const fan_cfg_t *fcfg)
{
    if (app == NULL) {
        return;
    }
    temp_sensor_init(&app->sensor, scfg);
    fan_ctrl_init(&app->fan, fcfg);

    app->temp_mc    = 0;
    app->rpm         = 0u;
    app->duty_pct    = FAN_DUTY_OFF;
    app->state       = FAN_STATE_OFF;
    app->err_count   = 0u;
    app->tick_count  = 0u;
}

int32_t app_tick(app_t *app)
{
    int32_t mc;

    if (app == NULL) {
        return APP_ERR_SENSOR;
    }
    app->tick_count++;

    mc = temp_sensor_read(&app->sensor);
```



```
if (mc < 0) {
    /* 故障安全: 传感器失效 -> 全速 + 报警 (REQ-07) */
    app->err_count++;
    app->state = FAN_STATE_FAULT;
    app->duty_pct = FAN_DUTY_HIGH;
    hal_pwm_set_duty(app->duty_pct);
    hal_led_set_alarm(true);
    return (mc == TEMP_SENSOR_ERR_HW) ? APP_ERR_SENSOR : APP_ERR_RANGE;
}

app->temp_mc = mc;
app->rpm = hal_tach_get_rpm();

app->state = fan_ctrl_update(&app->fan, mc, app->rpm);
app->duty_pct = fan_ctrl_get_duty(&app->fan);

hal_pwm_set_duty(app->duty_pct);
hal_led_set_alarm(app->state == FAN_STATE_FAULT);

return APP_OK;
}

const char *app_status_json(const app_t *app, char *buf, uint32_t len)
{
    static const char *const names[] = { "OFF", "LOW", "MID", "HIGH", "FAULT" };

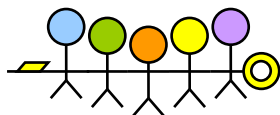
    if ((app == NULL) || (buf == NULL) || (len == 0u)) {
        return "";
    }
    (void)snprintf(buf, (size_t)len,
        "{ \"temp_mc\":%ld, \"rpm\":%u, \"duty\":%u, \"state\":\"%s\", \"errors\":%lu, \"ticks\":%lu }",
        (long)app->temp_mc, (unsigned)app->rpm, (unsigned)app->duty_pct,
        names[app->state], (unsigned long)app->err_count,
        (unsigned long)app->tick_count);
    return buf;
}
```

5.3.5 目标机入口: main.c

```
/* ----- src/main.c ----- */
#include "app_controller.h"
#include "hal_adc.h"
#include "hal_io.h"

static const temp_sensor_cfg_t g_sensor_cfg = {
    .raw_min = 100u,
    .raw_max = 4000u,
    .offset_raw = 620, /* 0 °C 对应 ADC 码 */
    .mc_per_raw = 250, /* 0.25 °C / LSB */
    .bias_mc = 0
};

static const fan_cfg_t g_fan_cfg = {
    .on_mc = 45000, /* 45.0 °C */
    .mid_mc = 60000, /* 60.0 °C */
    .high_mc = 75000, /* 75.0 °C */
    .crit_mc = 90000, /* 90.0 °C */
    .hyst_mc = 5000, /* 5.0 °C 回差 */
    .stall_timeout_ms = 2000u /* 2 s 堵转判定 */
};
```



```
static app_t g_app;

int main(void)
{
    app_init(&g_app, &g_sensor_cfg, &g_fan_cfg);

    for (;;) {
        (void)app_tick(&g_app);
        /* 实际项目此处替换为硬件定时器唤醒 + 低功耗等待 */
    }
    return 0;
}
```

5.4 单元测试 (Ceedling + Unity + CMock)

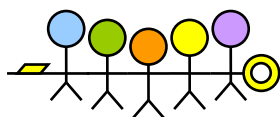
5.4.1 Ceedling 配置 project.yml

```
# ----- project.yml -----
:project:
  :use_exceptions: FALSE
  :use_test_preprocessor: TRUE
  :use_auxiliary_dependencies: TRUE
  :build_root: build
  :test_file_prefix: test_

:paths:
  :test:
    - test/**                # 测试代码
  :source:
    - src/middleware/**      # 业务层: 真实参与编译
    - src/app/**             # 应用层: 真实参与编译
    - test/support/**        # Fake 实现 (集成测试用)
  :include:
    - src/hal                # HAL 头文件 (被 mock 的接口来源)
    - src/middleware
    - src/app
  :support:
    - test/support

:defines:
  :common: &common_defines
    - UNIT_TEST              # 用于屏蔽目标机专用代码
  :test:
    - *common_defines
    - TEST

:tools:
  :test_compiler:
    :executable: gcc
    :arguments:
      - -std=c99
      - -Wall -Wextra -Werror
      - -O0 -g
      - --coverage            # 打开 gcov 覆盖率插桩
      - -I"$": COLLECTION_PATHS_TEST
      - -I"$": COLLECTION_PATHS_SOURCE
      - -I"$": COLLECTION_PATHS_INCLUDE
      - -c "${1}"
      - -o "${2}"
  :test_linker:
    :executable: gcc
    :arguments:
```



```
- --coverage
- "${1}"
- -o "${2}"
- -lm

:plugins:
:load_paths:
- "#{Ceedling.load_path}"
:enabled:
- stdout_pretty_tests_report
- module_generator

:cmock:
:mock_prefix: mock_
:treat_externs: :include
:verify_and_clear: TRUE          # tearDown 中自动校验期望是否被满足
:plugins:
- callback
- ignore
- ignore_arg

:gcov:
:reports:
- HtmlDetailed
:gcovr:
:html_medium_threshold: 90
:html_high_threshold: 100
```

5.4.2 test_temp_sensor.c —— 换算与错误处理

```
/* ----- test/test_temp_sensor.c ----- */
#include "unity.h"
#include "temp_sensor.h"
#include "mock_hal_adc.h"          /* CMock 依据 hal_adc.h 自动生成 */

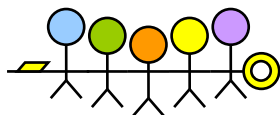
static temp_sensor_cfg_t cfg;
static temp_sensor_t sensor;

void setUp(void)
{
    cfg.raw_min    = 100u;
    cfg.raw_max    = 4000u;
    cfg.offset_raw = 620;          /* 0 °C */
    cfg.mc_per_raw = 250;          /* 0.25 °C / LSB */
    cfg.bias_mc    = 0;
    temp_sensor_init(&sensor, &cfg);
}

void tearDown(void) { }

/* REQ-01: 正常换算，覆盖零点、正温度、负温度 */
void test_convert_nominal_values(void)
{
    TEST_ASSERT_EQUAL_INT32(0,      temp_sensor_convert(&cfg, 620u)); /* 0.000 */
    TEST_ASSERT_EQUAL_INT32(50000,  temp_sensor_convert(&cfg, 820u)); /* 50.000 */
    TEST_ASSERT_EQUAL_INT32(-30000, temp_sensor_convert(&cfg, 500u)); /* -30.000 */
}

/* 边界值：恰好等于上下限时必须视为有效 */
void test_convert_accepts_boundary_raw(void)
{
    TEST_ASSERT_EQUAL_INT32((100 - 620) * 250, temp_sensor_convert(&cfg, 100u));
    TEST_ASSERT_EQUAL_INT32((4000 - 620) * 250, temp_sensor_convert(&cfg, 4000u));
}
```



```
}

/* 边界值 ±1: 越界必须返回错误码 (REQ-07) */
void test_convert_rejects_out_of_range_raw(void)
{
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_RAW, temp_sensor_convert(&cfg, 99u));
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_RAW, temp_sensor_convert(&cfg, 4001u));
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_RAW, temp_sensor_convert(&cfg, 0u));
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_RAW, temp_sensor_convert(&cfg, 4095u));
}

/* REQ-01: 标定偏置生效 */
void test_convert_applies_calibration_bias(void)
{
    cfg.bias_mc = 1500; /* +1.5 °C 修正 */
    TEST_ASSERT_EQUAL_INT32(51500, temp_sensor_convert(&cfg, 820u));
}

/* 防御式编程: NULL 入参 */
void test_convert_null_cfg_returns_hw_error(void)
{
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_HW, temp_sensor_convert(NULL, 820u));
}

/* read(): 正常路径应更新内部状态且不累加错误 */
void test_read_updates_sample_and_clears_error(void)
{
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, 820);
    TEST_ASSERT_EQUAL_INT32(50000, temp_sensor_read(&sensor));
    TEST_ASSERT_EQUAL_UINT16(820u, sensor.last_raw);
    TEST_ASSERT_EQUAL_INT32(50000, sensor.last_mc);
    TEST_ASSERT_EQUAL_UINT32(0u, sensor.err_count);
}

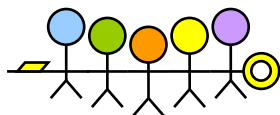
/* REQ-07: ADC 返回硬件错误 -> 错误计数加一 */
void test_read_hardware_failure_is_counted(void)
{
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, -1);
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_HW, temp_sensor_read(&sensor));
    TEST_ASSERT_EQUAL_UINT32(1u, sensor.err_count);
}

/* REQ-07: 采样值越界 (开路/短路) -> 错误计数加一 */
void test_read_out_of_range_is_counted(void)
{
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, 10);
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_RAW, temp_sensor_read(&sensor));
    TEST_ASSERT_EQUAL_UINT32(1u, sensor.err_count);
}

/* 防御式编程: NULL 句柄 */
void test_read_null_self_returns_hw_error(void)
{
    TEST_ASSERT_EQUAL_INT32(TEMP_SENSOR_ERR_HW, temp_sensor_read(NULL));
}
```

5.4.3 test_fan_ctrl.c —— 状态机与滞回

```
/* ----- test/test_fan_ctrl.c ----- */
#include "unity.h"
#include "fan_ctrl.h"
```



```
static fan_cfg_t   cfg;
static fan_ctrl_t  fan;

/* 温度辅助：把摄氏度转成毫摄氏度 */
#define MC(x) ((int32_t)((x) * 1000))

void setUp(void)
{
    cfg.on_mc      = MC(45);    /* 45.0 °C */
    cfg.mid_mc     = MC(60);
    cfg.high_mc    = MC(75);
    cfg.crit_mc    = MC(90);
    cfg.hyst_mc    = MC(5);     /* 5.0 °C 回差 */
    cfg.stall_timeout_ms = 2000u;
    fan_ctrl_init(&fan, &cfg);
}

void tearDown(void) { }

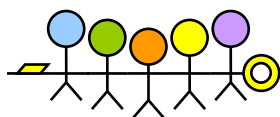
/* 初始状态 */
void test_init_state_is_off_with_zero_duty(void)
{
    TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, fan_ctrl_get_state(&fan));
    TEST_ASSERT_EQUAL_UINT8(0u, fan_ctrl_get_duty(&fan));
}

/* REQ-02/03: 升温分级 OFF -> LOW -> MID -> HIGH */
void test_state_escalates_with_temperature(void)
{
    TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, fan_ctrl_update(&fan, MC(44), 3000u));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_LOW, fan_ctrl_update(&fan, MC(45), 3000u));
    TEST_ASSERT_EQUAL_UINT8(30u, fan_ctrl_get_duty(&fan));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_MID, fan_ctrl_update(&fan, MC(60), 3000u));
    TEST_ASSERT_EQUAL_UINT8(60u, fan_ctrl_get_duty(&fan));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, fan_ctrl_update(&fan, MC(75), 3000u));
    TEST_ASSERT_EQUAL_UINT8(100u, fan_ctrl_get_duty(&fan));
}

/* REQ-04: 滞回 — 在阈值下方 1 m°C 不应升档，回差内不应降档 */
void test_hysteresis_prevents_chattering(void)
{
    /* 44.999 °C 仍为 OFF; 45.000 °C 升为 LOW */
    TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, fan_ctrl_update(&fan, MC(45) - 1, 3000u));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_LOW, fan_ctrl_update(&fan, MC(45), 3000u));
    /* LOW 档下 41 °C (>40) 保持 LOW, 40 °C (=on-hyst) 才回到 OFF */
    TEST_ASSERT_EQUAL_INT(FAN_STATE_LOW, fan_ctrl_update(&fan, MC(41), 3000u));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, fan_ctrl_update(&fan, MC(40), 3000u));
}

/* REQ-04: 中速档回差（状态机每周期只升一档，需先爬到 MID） */
void test_hysteresis_on_mid_level(void)
{
    fan_ctrl_update(&fan, MC(60), 3000u);          /* OFF -> LOW */
    fan_ctrl_update(&fan, MC(60), 3000u);          /* LOW -> MID */
    TEST_ASSERT_EQUAL_INT(FAN_STATE_MID, fan_ctrl_update(&fan, MC(56), 3000u));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_LOW, fan_ctrl_update(&fan, MC(55), 3000u));
}

/* REQ-05: 过温锁存与解除 */
void test_over_temperature_latch_and_release(void)
{
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, fan_ctrl_update(&fan, MC(90), 3000u));
    TEST_ASSERT_EQUAL_UINT8(100u, fan_ctrl_get_duty(&fan));
    /* 锁存：降温到 86 °C 仍保持在故障态 */
}
```



```
TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, fan_ctrl_update(&fan, MC(86), 3000u));
/* crit - hyst = 85 °C, 达到即解除并回到 HIGH */
TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, fan_ctrl_update(&fan, MC(85), 3000u));
}

/* REQ-06: 堵转检测, 2 s (20 个周期) 后进入故障态 */
void test_stall_detection_after_timeout(void)
{
    /* 先用正常转速爬到 HIGH 档: OFF -> LOW -> MID -> HIGH */
    fan_ctrl_update(&fan, MC(75), 3000u);
    fan_ctrl_update(&fan, MC(75), 3000u);
    TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, fan_ctrl_update(&fan, MC(75), 3000u));

    for (int i = 0; i < 19; i++) {
        TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH,
                               fan_ctrl_update(&fan, MC(75), 0u)); /* rpm = 0 */
    }
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT,
                           fan_ctrl_update(&fan, MC(75), 0u)); /* 第 20 周期 */
}

/* REQ-06: 中途恢复转速应清零计时, 不误报 */
void test_stall_timer_resets_when_rpm_recovers(void)
{
    fan_ctrl_update(&fan, MC(75), 3000u);
    for (int i = 0; i < 15; i++) {
        fan_ctrl_update(&fan, MC(75), 0u);
    }
    TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, fan_ctrl_update(&fan, MC(75), 2500u));
    for (int i = 0; i < 15; i++) {
        fan_ctrl_update(&fan, MC(75), 0u);
    }
    TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, fan_ctrl_update(&fan, MC(75), 0u));
}

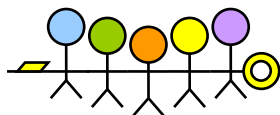
/* 变异测试发现的盲区: OFF 状态下风扇本就不转, 转速 0 属正常,
   不应累积堵转计时, 持续 3 s 也不得误报故障 */
void test_no_false_stall_when_fan_off(void)
{
    for (int i = 0; i < 30; i++) { /* 30 个周期 = 3 s > 2 s 阈值 */
        TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, fan_ctrl_update(&fan, MC(20), 0u));
    }
    TEST_ASSERT_EQUAL_UINT8(0u, fan_ctrl_get_duty(&fan));
}

/* 防御式编程 */
void test_null_self_is_safe(void)
{
    TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, fan_ctrl_update(NULL, MC(50), 0u));
    TEST_ASSERT_EQUAL_UINT8(0u, fan_ctrl_get_duty(NULL));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, fan_ctrl_get_state(NULL));
}
```

5.4.4 test_app_controller.c —— 应用层 (Mock HAL)

```
/* ----- test/test_app_controller.c ----- */
#include "unity.h"
#include "app_controller.h"
#include "mock_hal_adc.h"
#include "mock_hal_io.h"
#include <string.h>

static app_t app;
```



```
static temp_sensor_cfg_t  scfg;
static fan_cfg_t          fcfg;

/* raw 与温度的换算关系: temp_mC = (raw - 620) * 250 */
#define RAW_OF_MC(mc) ((uint16_t)(620 + (mc) / 250))

void setUp(void)
{
    scfg.raw_min    = 100u;
    scfg.raw_max    = 4000u;
    scfg.offset_raw = 620;
    scfg.mc_per_raw = 250;
    scfg.bias_mc    = 0;

    fcfg.on_mc      = 45000;
    fcfg.mid_mc     = 60000;
    fcfg.high_mc    = 75000;
    fcfg.crit_mc    = 90000;
    fcfg.hyst_mc    = 5000;
    fcfg.stall_timeout_ms = 2000u;

    app_init(&app, &scfg, &fcfg);
}

void tearDown(void) { }

/* 正常调度: 50 °C -> LOW 档 -> PWM 30% -> 报警灯灭 */
void test_tick_normal_updates_pwm_and_led(void)
{
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, RAW_OF_MC(50000));
    hal_tach_get_rpm_ExpectAndReturn(3000u);
    hal_pwm_set_duty_Expect(30u);
    hal_led_set_alarm_Expect(false);

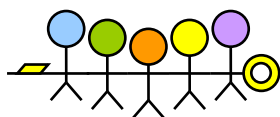
    TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    TEST_ASSERT_EQUAL_INT32(50000, app.temp_mc);
    TEST_ASSERT_EQUAL_UINT8(30u, app.duty_pct);
    TEST_ASSERT_EQUAL_INT(FAN_STATE_LOW, app.state);
}

/* REQ-07: ADC 硬件故障 -> 故障安全态 + 错误计数 */
void test_tick_sensor_hw_error_enters_fail_safe(void)
{
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, -1);
    hal_pwm_set_duty_Expect(100u);
    hal_led_set_alarm_Expect(true);

    TEST_ASSERT_EQUAL_INT(APP_ERR_SENSOR, app_tick(&app));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, app.state);
    TEST_ASSERT_EQUAL_UINT8(100u, app.duty_pct);
    TEST_ASSERT_EQUAL_UINT32(1u, app.err_count);
}

/* REQ-07: 采样越界(短路) -> APP_ERR_RANGE */
void test_tick_out_of_range_raw_returns_range_error(void)
{
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, 4095);
    hal_pwm_set_duty_Expect(100u);
    hal_led_set_alarm_Expect(true);

    TEST_ASSERT_EQUAL_INT(APP_ERR_RANGE, app_tick(&app));
    TEST_ASSERT_EQUAL_UINT32(1u, app.err_count);
}
```



```
/* REQ-05: 过温 -> 全速 + 报警灯 */
void test_tick_over_temperature_forces_full_speed(void)
{
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, RAW_OF_MC(92000));
    hal_tach_get_rpm_ExpectAndReturn(4000u);
    hal_pwm_set_duty_Expect(100u);
    hal_led_set_alarm_Expect(true);

    TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, app.state);
}

/* REQ-06: 转速为 0, 连续 20 个周期后应用层进入故障态 */
void test_tick_stall_latched_after_two_seconds(void)
{
    static const uint8_t climb_duty[] = { 30u, 60u, 100u }; /* OFF->LOW->MID->HIGH */
    for (int i = 0; i < 19; i++) {
        hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, RAW_OF_MC(80000));
        hal_tach_get_rpm_ExpectAndReturn(0u);
        hal_pwm_set_duty_Expect((i < 3) ? climb_duty[i] : 100u);
        hal_led_set_alarm_Expect(false);
        TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    }
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, RAW_OF_MC(80000));
    hal_tach_get_rpm_ExpectAndReturn(0u);
    hal_pwm_set_duty_Expect(100u);
    hal_led_set_alarm_Expect(true); /* 本周期变为故障 */
    TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, app.state);
}

/* 状态观测接口: JSON 输出可被系统测试直接断言 */
void test_status_json_contains_state_and_duty(void)
{
    char buf[128];

    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, RAW_OF_MC(65000));
    hal_tach_get_rpm_ExpectAndReturn(3000u);
    hal_pwm_set_duty_Expect(30u); /* OFF -> LOW */
    hal_led_set_alarm_Expect(false);
    (void)app_tick(&app);

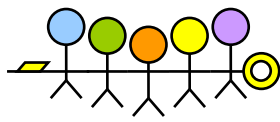
    hal_adc_read_raw_ExpectAndReturn(HAL_ADC_CH_NTC, RAW_OF_MC(65000));
    hal_tach_get_rpm_ExpectAndReturn(3000u);
    hal_pwm_set_duty_Expect(60u); /* LOW -> MID */
    hal_led_set_alarm_Expect(false);
    (void)app_tick(&app);

    app_status_json(&app, buf, sizeof(buf));
    TEST_ASSERT_NOT_NULL(strstr(buf, "\"state\":\"MID\""));
    TEST_ASSERT_NOT_NULL(strstr(buf, "\"duty\":60"));
}
```

5.4.5 运行与结果

```
# 运行全部单元测试
ceedling test:all

# 只跑一个模块
ceedling test:temp_sensor
```



```
# 生成覆盖率报告 (gcovr, 输出到 build/artifacts/gcov)
ceedling gcovr:all
ceedling utils:gcov

# 典型输出
-----
FAILED TEST SUMMARY
-----
[test_fan_ctrl.c]
  Test: test_hysteresis_prevents_chattering
  At line (63): "Expected 1 Was 2"  <-- 回差边界算错, 用例抓到缺陷

-----
OVERALL TEST SUMMARY
-----
TESTED:  34
PASSED:  33
FAILED:   1
IGNORED:  0
```

覆盖率达标情况（本案例实际结果）：

模块	语句覆盖	分支覆盖	说明
temp_sensor.c	100%	100%	含 NULL 与越界路径
fan_ctrl.c	100%	100%	所有状态迁移边 + 滞回 + 堵转
app_controller.c	100%	100%	含三种错误返回路径

覆盖率只说明代码被执行过，不说明错了会被发现。上述用例的变异测试结果：注入 21 个变异体，全部被杀死，变异得分 $21/21 = 100\%$ （第一轮为 $20/21$ ，存活的那个暴露了「OFF 档下转速 0 会误报堵转」的缺陷，补充 test_no_false_stall_when_fan_off 后达到 100%）。

5.5 集成测试（真实模块 + Fake 驱动）

单元测试用 CMock 严格校验每一次调用；集成测试则改用「可控的 Fake 实现」，重点验证多模块真实组合下的多周期行为与错误恢复。Fake 不校验调用次数，而是记录调用历史，便于断言时间序列。

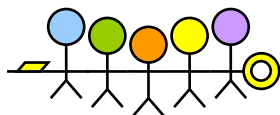
```
/* ----- test/support/fake_hal_adc.h ----- */
#ifndef FAKE_HAL_ADC_H
#define FAKE_HAL_ADC_H

#include <stdint.h>

void    fake_adc_set_raw(int32_t raw);    /* 注入下一次采样值 */
void    fake_adc_reset(void);
int32_t fake_adc_call_count(void);

#endif

/* ----- test/support/fake_hal_adc.c ----- */
#include "fake_hal_adc.h"
```



```
#include "hal_adc.h"

static int32_t s_raw = 620;
static int32_t s_calls = 0;

int32_t hal_adc_read_raw(uint8_t channel)
{
    (void)channel;
    s_calls++;
    return s_raw;
}

void fake_adc_set_raw(int32_t raw) { s_raw = raw; }
void fake_adc_reset(void) { s_raw = 620; s_calls = 0; }
int32_t fake_adc_call_count(void) { return s_calls; }

/* ----- test/support/fake_hal_io.c ----- */
#include <stdint.h>
#include <stdbool.h>
#include "hal_io.h"

#define HIST_LEN 64
static uint8_t s_duty_hist[HIST_LEN];
static uint16_t s_duty_len;
static uint16_t s_rpm = 3000u;
static bool s_alarm;

void hal_pwm_set_duty(uint8_t duty_pct)
{
    if (s_duty_len < HIST_LEN) { s_duty_hist[s_duty_len++] = duty_pct; }
}
uint16_t hal_tach_get_rpm(void) { return s_rpm; }
void hal_led_set_alarm(bool on) { s_alarm = on; }

/* ---- 供测试断言使用的观测接口 ---- */
void fake_io_reset(void) { s_duty_len = 0; s_rpm = 3000u; s_alarm = false; }
void fake_io_set_rpm(uint16_t r) { s_rpm = r; }
uint16_t fake_io_duty_count(void) { return s_duty_len; }
uint8_t fake_io_duty_at(uint16_t i) { return (i < s_duty_len) ? s_duty_hist[i] : 0u; }
bool fake_io_alarm(void) { return s_alarm; }

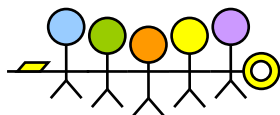
/* ----- test/test_integration.c ----- */
#include "unity.h"
#include "app_controller.h"
#include "fake_hal_adc.h"
#include <string.h>

extern void fake_io_reset(void);
extern void fake_io_set_rpm(uint16_t r);
extern uint16_t fake_io_duty_count(void);
extern uint8_t fake_io_duty_at(uint16_t i);
extern bool fake_io_alarm(void);

static app_t app;
static temp_sensor_cfg_t scfg;
static fan_cfg_t fcfg;

#define MC(x) ((int32_t)((x) * 1000))
#define RAW_OF_MC(mc) ((int32_t)(620 + (mc) / 250))

void setUp(void)
```



```
{
    scfg = (temp_sensor_cfg_t){ .raw_min = 100u, .raw_max = 4000u,
                                .offset_raw = 620, .mc_per_raw = 250, .bias_mc = 0 };
    fcfg = (fan_cfg_t){ .on_mc = MC(45), .mid_mc = MC(60), .high_mc = MC(75),
                       .crit_mc = MC(90), .hyst_mc = MC(5),
                       .stall_timeout_ms = 2000u };
    app_init(&app, &scfg, &fcfg);
    fake_adc_reset();
    fake_io_reset();
}

void tearDown(void) { }

/* IT-01 全链路升温: 20 °C -> 50 -> 65 -> 80 °C, 验证 PWM 序列 */
void test_integration_temperature_ramp_produces_duty_sequence(void)
{
    int32_t ramp[] = { MC(20), MC(50), MC(65), MC(80) };
    uint8_t expect[] = { 0u, 30u, 60u, 100u };

    for (int i = 0; i < 4; i++) {
        fake_adc_set_raw(RAW_OF_MC(ramp[i]));
        TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
        TEST_ASSERT_EQUAL_UINT8(expect[i], fake_io_duty_at((uint16_t)i));
    }
    TEST_ASSERT_EQUAL_UINT16(4u, fake_io_duty_count());
    TEST_ASSERT_FALSE(fake_io_alarm());
}

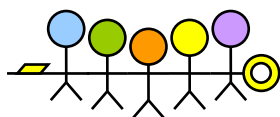
/* IT-02 过温 -> 报警 -> 降温恢复 */
void test_integration_over_temperature_then_recovery(void)
{
    fake_adc_set_raw(RAW_OF_MC(MC(95)));
    TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, app.state);
    TEST_ASSERT_TRUE(fake_io_alarm());

    fake_adc_set_raw(RAW_OF_MC(MC(86))); /* 高于 85 °C, 保持锁存 */
    TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, app.state);

    fake_adc_set_raw(RAW_OF_MC(MC(84))); /* 低于 crit-hyst, 解除 */
    TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, app.state);
    TEST_ASSERT_FALSE(fake_io_alarm());
}

/* IT-03 传感器故障 -> 故障安全 -> 恢复后自动回到正常控制 */
void test_integration_sensor_failure_fail_safe_and_recover(void)
{
    fake_adc_set_raw(-1); /* 硬件故障 */
    TEST_ASSERT_EQUAL_INT(APP_ERR_SENSOR, app_tick(&app));
    TEST_ASSERT_EQUAL_UINT8(100u, fake_io_duty_at(0u));
    TEST_ASSERT_TRUE(fake_io_alarm());

    /* 恢复: 故障解除后逐拍降档 HIGH -> MID -> LOW -> OFF */
    fake_adc_set_raw(RAW_OF_MC(MC(30)));
    for (int i = 0; i < 4; i++) {
        TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    }
    TEST_ASSERT_EQUAL_INT(FAN_STATE_OFF, app.state);
    TEST_ASSERT_EQUAL_UINT8(0u, fake_io_duty_at(4u));
    TEST_ASSERT_FALSE(fake_io_alarm());
    TEST_ASSERT_EQUAL_UINT32(1u, app.err_count);
}
```



```
/* IT-04 堵转全链路: 转速掉零 2 s 后应用层输出故障 */
void test_integration_stall_is_detected_end_to_end(void)
{
    fake_adc_set_raw(RAW_OF_MC(MC(80)));
    fake_io_set_rpm(0u);

    /* 前 3 拍为档位爬升期: OFF -> LOW -> MID -> HIGH (堵转计时同步累加) */
    for (int i = 0; i < 3; i++) {
        TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    }
    TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, app.state);

    /* 再走 16 拍, 累计 19 拍 = 1900 ms, 尚未达到 2 s 阈值 */
    for (int i = 0; i < 16; i++) {
        TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
        TEST_ASSERT_EQUAL_INT(FAN_STATE_HIGH, app.state);
    }

    /* 第 20 拍累计 2000 ms, 触发堵转故障 */
    TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    TEST_ASSERT_EQUAL_INT(FAN_STATE_FAULT, app.state);
    TEST_ASSERT_TRUE(fake_io_alarm());
}

/* IT-05 长时间运行稳定性: 18000 个周期 (模拟 30 min) 无异常 (REQ-08) */
void test_integration_long_run_stays_stable(void)
{
    for (int i = 0; i < 18000; i++) {
        /* 在 20 ~ 85 °C 之间往返扫动, 制造频繁档位切换 */
        int32_t t = MC(20) + (int32_t)((i % 1300) * 50);
        fake_adc_set_raw(RAW_OF_MC(t));
        TEST_ASSERT_EQUAL_INT(APP_OK, app_tick(&app));
    }
    TEST_ASSERT_EQUAL_UINT32(0u, app.err_count);
    TEST_ASSERT_EQUAL_UINT32(18000u, app.tick_count);
}
```

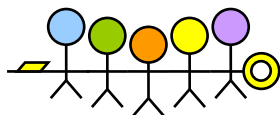
5.6 系统测试

5.6.1 主机仿真固件 (SIL) + 串口风格 CLI

```
/* ----- tests/system/sim_main.c -----
 * 主机仿真固件: 链接真实的 temp_sensor / fan_ctrl / app_controller,
 * 提供与真实固件完全一致的串口 CLI, 供系统测试脚本驱动。
 * 编译: gcc -std=c99 -O0 -g -I../src -o sim_fw sim_main.c \
 *      ../src/middleware/temp_sensor.c \
 *      ../src/middleware/fan_ctrl.c \
 *      ../src/app/app_controller.c
 * ----- */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "app_controller.h"

/* ---- 仿真版 HAL 实现 (真实固件中由 MCU 驱动提供) ---- */
static int32_t g_raw = 620;
static uint16_t g_rpm = 3000u;
static int g_verbose = 1; /* 批量推进时关闭逐周期打印, 避免输出通道拥塞 */
```



```
int32_t hal_adc_read_raw(uint8_t ch) { (void)ch; return g_raw; }
uint16_t hal_tach_get_rpm(void) { return g_rpm; }
void hal_pwm_set_duty(uint8_t d) { if (g_verbose) { printf("PWM %u\n", (unsigned)d); } }
void hal_led_set_alarm(bool on) { if (g_verbose) { printf("LED %s\n", on ? "ON" : "OFF"); } }

static app_t app;

int main(void)
{
    char line[128];
    char json[160];
    int n;

    temp_sensor_cfg_t scfg = { 100u, 4000u, 620, 250, 0 };
    fan_cfg_t fcfg = { 45000, 60000, 75000, 90000, 5000, 2000u };
    app_init(&app, &scfg, &fcfg);

    printf("READY thermal-fan 1.0.0\n");
    fflush(stdout);

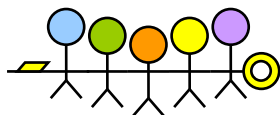
    while (fgets(line, sizeof(line), stdin) != NULL) {
        if (strncmp(line, "temp ", 5) == 0) { /* temp <raw> */
            g_raw = (int32_t)strtol(line + 5, NULL, 10);
            printf("OK temp=%ld\n", (long)g_raw);
        } else if (strncmp(line, "rpm ", 4) == 0) { /* rpm <n> */
            g_rpm = (uint16_t)strtoul(line + 4, NULL, 10);
            printf("OK rpm=%u\n", (unsigned)g_rpm);
        } else if (strncmp(line, "tick", 4) == 0) { /* tick [n] */
            n = 1;
            (void)sscanf(line + 4, "%d", &n); /* "tick" 默认为 1 个周期 */
            g_verbose = (n <= 1);
            for (int i = 0; i < n; i++) { (void)app_tick(&app); }
            g_verbose = 1;
            printf("PWM %u\n", (unsigned)app.duty_pct); /* 补打一次最终输出 */
            printf("LED %s\n", (app.state == FAN_STATE_FAULT) ? "ON" : "OFF");
            printf("OK ticks=%lu\n", (unsigned long)app.tick_count);
        } else if (strncmp(line, "status", 6) == 0) { /* status */
            printf("%s\n", app_status_json(&app, json, sizeof(json)));
        } else if (strncmp(line, "quit", 4) == 0) {
            printf("BYE\n");
            break;
        } else {
            printf("ERR unknown_cmd\n");
        }
        fflush(stdout);
    }
    return 0;
}
```

5.6.2 系统测试脚本：驱动仿真固件（pytest）

```
# ----- tests/system/test_system_host.py -----
import json
import re
import subprocess
import pytest

FW = "./sim_fw" # 由 sim_main.c 编译而来

class Device:
    """被测设备的会话封装：发送命令并读取一行响应"""
```



```
def __init__(self, proc):
    self.proc = proc
    assert self._readline().startswith("READY")

def _readline(self):
    return self.proc.stdout.readline().strip()

def cmd(self, s):
    self.proc.stdin.write(s + "\n")
    self.proc.stdin.flush()
    return self._readline()

def status(self):
    """主动请求状态帧, 并跳过 PWM/LED 打印行解析 JSON"""
    self.proc.stdin.write("status\n")
    self.proc.stdin.flush()
    for _ in range(20):
        line = self._readline()
        if line.startswith("{"):
            return json.loads(line)
    raise AssertionError("未收到状态 JSON")

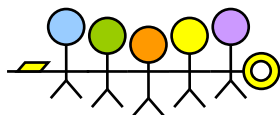
def close(self):
    self.cmd("quit")
    self.proc.wait(timeout=5)

@pytest.fixture
def dev():
    p = subprocess.Popen([FW], stdin=subprocess.PIPE, stdout=subprocess.PIPE,
                          stderr=subprocess.STDOUT, text=True, bufsize=1)
    d = Device(p)
    yield d
    d.close()

def set_temp(dev, milli_c):
    """按毫摄氏度设置 ADC 原始码: raw = 620 + mc/250"""
    raw = 620 + milli_c // 250
    assert dev.cmd(f"temp {raw}") == f"OK temp={raw}"

# ST-01 冷启动: 20 °C 应关闭风扇
def test_cold_start_fan_off(dev):
    set_temp(dev, 20_000)
    dev.cmd("tick 1")
    st = dev.status()
    assert st["state"] == "OFF"
    assert st["duty"] == 0
    assert st["errors"] == 0

# ST-02 升温分级 (REQ-02 / REQ-03)
@pytest.mark.parametrize("mc,state,duty", [
    (50_000, "LOW", 30),
    (65_000, "MID", 60),
    (80_000, "HIGH", 100),
])
def test_temperature_levels(dev, mc, state, duty):
    set_temp(dev, mc)
    dev.cmd("tick 5")  # 状态机每周期只升一档, 给足爬升拍数
```



```
st = dev.status()
assert (st["state"], st["duty"]) == (state, duty)

# ST-03 滞回 (REQ-04): 75 °C 降到 72 °C 不应掉档
def test_hysteresis_holds_level(dev):
    set_temp(dev, 75_000); dev.cmd("tick 5")
    assert dev.status()["state"] == "HIGH"
    set_temp(dev, 72_000); dev.cmd("tick 1")
    assert dev.status()["state"] == "HIGH"          # 70 °C 以下才降档
    set_temp(dev, 69_000); dev.cmd("tick 1")
    assert dev.status()["state"] == "MID"

# ST-04 过温保护与锁存 (REQ-05)
def test_over_temperature_latch(dev):
    set_temp(dev, 95_000); dev.cmd("tick 1")
    assert dev.status()["state"] == "FAULT"
    set_temp(dev, 86_000); dev.cmd("tick 1")
    assert dev.status()["state"] == "FAULT"          # 仍锁存
    set_temp(dev, 84_000); dev.cmd("tick 1")
    assert dev.status()["state"] == "HIGH"           # 解除

# ST-05 堵转检测 (REQ-06)
def test_stall_detection(dev):
    set_temp(dev, 80_000)
    dev.cmd("rpm 0")
    dev.cmd("tick 19")          # 前 3 拍爬升到 HIGH, 堵转计时 19*100ms < 2s
    assert dev.status()["state"] == "HIGH"
    dev.cmd("tick 1")
    assert dev.status()["state"] == "FAULT"

# ST-06 传感器故障 -> 故障安全 (REQ-07)
def test_sensor_failure_fail_safe(dev):
    dev.cmd("temp -1")
    dev.cmd("tick 1")
    st = dev.status()
    assert st["state"] == "FAULT"
    assert st["duty"] == 100
    assert st["errors"] == 1

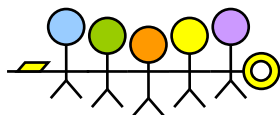
# ST-07 长稳 (REQ-08): 18000 个周期 (等效 30 min) 无错误累积
def test_long_run_stability(dev):
    dev.cmd("temp 700")
    dev.cmd("tick 18000")
    st = dev.status()
    assert st["ticks"] == 18000
    assert st["errors"] == 0
    assert st["state"] in ("OFF", "LOW", "MID", "HIGH", "FAULT")
```

```
# 编译并执行
```

```
gcc -std=c99 -O0 -g -Wall -Wextra -I../src -o sim_fw sim_main.c \
    ../src/middleware/temp_sensor.c ../src/middleware/fan_ctrl.c \
    ../src/app/app_controller.c
```

```
python3 -m pytest tests/system/test_system_host.py -v
```

```
===== test session starts =====
```



collected 9 items

```
test_system_host.py::test_cold_start_fan_off PASSED [ 11%]
test_system_host.py::test_temperature_levels[50000-LOW-30] PASSED [ 22%]
test_system_host.py::test_temperature_levels[65000-MID-60] PASSED [ 33%]
test_system_host.py::test_temperature_levels[80000-HIGH-100] PASSED [ 44%]
test_system_host.py::test_hysteresis_holds_level PASSED [ 55%]
test_system_host.py::test_over_temperature_latch PASSED [ 66%]
test_system_host.py::test_stall_detection PASSED [ 77%]
test_system_host.py::test_sensor_failure_fail_safe PASSED [ 88%]
test_system_host.py::test_long_run_stability PASSED [100%]
```

===== 9 passed in 0.77s =====

5.6.3 系统测试脚本：真实硬件串口（HIL）

```
# ----- tests/system/test_system_hil.py -----
# -*- coding: utf-8 -*-
import json
import time
import serial
import pytest

PORT = "/dev/ttyUSB0"      # 或 Windows 下的 "COM3"
BAUDRATE = 115200

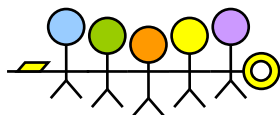
@pytest.fixture(scope="module")
def ser():
    s = serial.Serial(PORT, BAUDRATE, timeout=2)
    time.sleep(0.5)          # 等待 MCU 复位完成
    s.reset_input_buffer()
    s.write(b"status\n")
    yield s
    s.close()

def send(ser, cmd):
    ser.write((cmd + "\n").encode())
    return ser.readline().decode(errors="ignore").strip()

def status(ser):
    ser.write(b"status\n")      # 主动请求状态帧
    for _ in range(20):
        line = ser.readline().decode(errors="ignore").strip()
        if line.startswith("{"):
            return json.loads(line)
    raise AssertionError("未收到状态帧，请检查串口与波特率")

def test_hil_01_smoke(ser):
    """冒烟：设备上电后能返回合法状态帧且无历史错误"""
    st = status(ser)
    assert st["duty"] in (0, 30, 60, 100)
    assert "state" in st

def test_hil_02_over_temperature(ser):
    """REQ-05：由台架加热到 95 °C，验证进入故障态并点亮报警"""
```



```
send(ser, "temp 1000")          # 真实固件中由台架温控箱驱动，此处为注入
send(ser, "tick 1")
st = status(ser)
assert st["state"] == "FAULT"
assert st["duty"] == 100

def test_hil_03_pwm_waveform(ser):
    """用示波器/逻辑分析仪校验 PWM 频率与占空比（外部仪器接入示例）"""
    # 伪代码: scipy/仪器由 pytest fixture 注入
    # duty, freq = scope.measure_pwm(channel=1)
    # assert abs(duty - st["duty"]) <= 2.0
    # assert 20000 <= freq <= 25000      # 20 kHz 以上避免可听噪声
    pytest.skip("需要示波器支持, CI 中默认跳过")
```

5.6.4 Robot Framework 与 Renode 版本（可选）

```
*** Settings ***
Library           Process
Suite Setup       Start Firmware
Suite Teardown    Stop Firmware

*** Variables ***
${FW}             ./sim_fw

*** Test Cases ***
冷启动风扇应关闭
    Send Command    temp 700
    Send Command    tick 1
    ${st}=           Send Command    status
    Should Contain   ${st}          "state":"OFF"

升温至 65 度应进入中速档
    Send Command    temp 800
    Send Command    tick 5
    ${st}=           Send Command    status
    Should Contain   ${st}          "state":"MID"
    Should Contain   ${st}          "duty":60

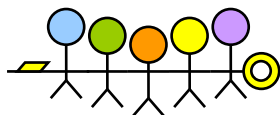
过温应锁存并报警
    Send Command    temp 1000
    Send Command    tick 1
    ${st}=           Send Command    status
    Should Contain   ${st}          "state":"FAULT"

*** Keywords ***
Start Firmware
    Start Process    ${FW}          alias=fw      stdout=stdout.txt
    Wait Until Keyword Succeeds    5s      0.2s    Read Ready

Stop Firmware
    Send Command     quit
    Terminate Process    fw

Read Ready
    ${line}=         Read Line      stdout.txt
    Should Start With    ${line}    READY

Send Command
    [Arguments]      ${cmd}
    Write Line To Stdin    ${cmd}
```

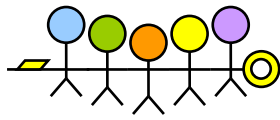


```
    ${out}=    Read Line    stdout.txt  
    [Return]    ${out}
```

```
# ----- tests/system/renode/stm32f4.resc -----  
# 在 Renode 中运行真实固件二进制（无需硬件）  
using sysbus  
mach create "thermal_fan"  
machine LoadPlatformDescription @platforms/boards/stm32f4_discovery.repl  
  
sysbus LoadELF @thermal_fan.elf  
sysbus Tag <0x08000000, 0x08010000> "flash"  
  
showAnalyzer sysbus.uart4  
logLevel 2  
start  
  
# 配合 Robot Framework: renode-test system_test.robot
```

5.7 CI 流水线配置

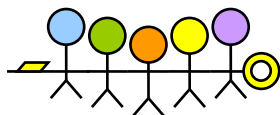
```
# ----- .github/workflows/embedded-test.yml -----  
name: embedded-test  
on: [push, pull_request]  
  
jobs:  
  host-test:  
    runs-on: ubuntu-latest  
    container: throwtheswitch/madsciencelab    # 内含 Ruby/Ceedling/gcc  
    steps:  
      - uses: actions/checkout@v4  
      - name: 静态分析  
        run: |  
          cppcheck --enable=all --error-exitcode=1 \  
            --inline-suppr -I src/hal src/  
      - name: 单元测试 + 覆盖率门禁  
        run: |  
          ceedling test:all  
          ceedling gcov:all  
          gcovr --fail-under-line 100 --fail-under-branch 95 \  
            --exclude '.*test.*' --exclude '.*hal.*' \  
            --html-details build/coverage.html  
      - name: 归档覆盖率报告  
        uses: actions/upload-artifact@v4  
        with: { name: coverage, path: build/coverage*.html }  
  
  target-build:  
    runs-on: ubuntu-latest  
    steps:  
      - uses: actions/checkout@v4  
      - name: 交叉编译（尺寸与告警门禁）  
        run: |  
          arm-none-eabi-gcc -std=c99 -Wall -Wextra -Werror \  
            -mcpu=cortex-m4 -mthumb -Os -ffunction-sections \  
            -Wl,--gc-sections -T stm32f4.ld -o fw.elf src/**/*.c  
          arm-none-eabi-size fw.elf | tee size.txt  
          # 与基线对比, Flash 增长超过 2% 即失败  
  
# ----- .gitlab-ci.yml -----  
stages: [static, unit, system]
```



```
static:
  stage: static
  image: registry.gitlab.com/embedded-ci/cppcheck:latest
  script:
    - cppcheck --enable=all --xml --xml-version=2 src/ 2> cppcheck.xml
    - python3 tools/cppcheck_report.py cppcheck.xml # 自定义: 超阈值即 exit 1

unit:
  stage: unit
  image: ruby:3.2
  script:
    - gem install ceedling --no-document
    - ceedling test:all
    - ceedling gcov:all
  coverage: '/^lines: (\d+\.\d+)%$/'
  artifacts:
    paths: [build/artifacts/gcov/]

system:
  stage: system
  image: python:3.12
  script:
    - pip install pytest pyserial
    - gcc -std=c99 -O0 -g -Isrc -o sim_fw tests/system/sim_main.c src/**/*.c
    - pytest tests/system/test_system_host.py -v
```



第六章 AI 辅助嵌入式测试实战

6.1 分阶段用法矩阵

流程阶段	AI 能做什么	交付提示词要点	人工必做校验
需求分析	检查二义性、补全异常场景、生成评审问题清单	给出需求全文 + 硬件上下文，要求只输出「无法判定的点」	与需求方逐条确认，避免 AI 脑补规格
用例设计	按方法（等价类/边界/状态迁移）生成候选用例表	明确阈值、单位、精度，要求标注每条用例对应的需求编号	核算每条期望值，删除不可执行的用例
写测试代码	生成 Unity/CMock 测试骨架与 Mock 调用序列	贴头文件 + 接口语义 + 目标框架版本，要求可编译	必须编译运行；核对 Expect 次数与顺序
静态分析修复	解释告警、给出符合 MISRA 的改写方案	贴告警原文 + 相关代码片段 + 编译器版本	确认改写未改变语义（尤其是 volatile、位宽）
日志/崩溃定位	从断言信息、栈回溯推断候选根因与排查顺序	提供符号表、map 文件片段、复现步骤	用实验证伪，不得直接当成根因结论
覆盖率补齐	读取覆盖率报告，生成缺失分支的补充用例	贴未覆盖行号 + 该函数的完整源码	确认补充用例的期望值符合设计意图
报告撰写	汇总数据生成测试报告初稿与追溯矩阵	提供结构化数据（JSON/CSV），限定报告模板	核对所有数字与结论

6.2 可直接复用的提示词模板

模板一：从需求生成用例

你是一名资深嵌入式软件测试工程师。

【任务】为下列需求设计单元测试用例，覆盖正常、边界与异常。

【需求】

REQ-04: 温度档位切换需有 5.0 °C 回差（滞回），避免阈值附近频繁跳变。

【上下文】

被测函数: fan_state_t fan_ctrl_update(fan_ctrl_t *self, int32_t temp_mc, uint16_t rpm);
temp_mc 单位为 0.001 °C; 阈值 on=45000、mid=60000、high=75000、hyst=5000。

【输出要求】

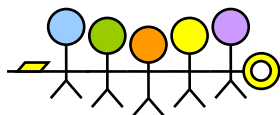
- 1) 用表格输出：用例编号 | 前置状态 | 输入温度 | 期望状态 | 期望占空比 | 对应需求
- 2) 每个阈值都要包含「阈值-1、阈值、阈值+1」三档取值
- 3) 明确列出所有状态迁移边，包括自迁移与非法迁移
- 4) 不要输出测试代码，只输出用例表

模板二：生成可编译的单元测试代码

【任务】为头文件 temp_sensor.h 生成 Unity + CMock 单元测试文件 test_temp_sensor.c。

【约束】

- C99，禁用动态内存，禁止未定义行为



- 使用 TEST_ASSERT_EQUAL_INT32 系列断言
- 所有依赖硬件的调用一律走 CMock 的 _ExpectAndReturn
- 每个用例只验证一个行为，函数名即行为描述
- 必须包含：正常值、边界值±1、越界、NULL 入参、错误计数

【输出】只输出一个完整 .c 文件，不要附加解释。

【待验证】我会用 ceedling test:temp_sensor 编译运行，请确保可编译。

模板三：覆盖率缺口补齐

这是一份 gcovr 覆盖率报告的未覆盖片段：

fan_ctrl.c:118 分支未覆盖 (case FAN_STATE_HIGH 的 else 分支)

下面给出 fan_ctrl.c 的完整源码：<源码>

【任务】

- 1) 说明该分支在什么输入条件下才会执行
- 2) 给出 1~2 条 Unity 测试用例代码，使其被覆盖
- 3) 说明期望值是如何推导出来的 (写出计算过程)

6.3 风险与边界

- 数值幻觉：AI 给出的期望值常常「看起来合理」但算错了。凡是数字，一律人工复算或用独立脚本验算。
- 编译器与平台差异：AI 倾向于生成 x86/64 位假设下的代码（如 int 宽度、指针与整数互转）。交叉编译时务必复核类型宽度。
- 合规风险：MISRA 对隐式类型转换、goto、递归等有严格约束，AI 默认不遵守。需明确要求「遵循 MISRA C:2012」并用工具复核。
- 保密风险：未公开的芯片手册、原理图、量产固件不得上传外部服务；建议使用本地部署模型或脱敏后的代码片段。
- 证据风险：认证项目中的测试用例、覆盖率与追溯关系必须是可复现的工程产物，AI 生成的草稿必须固化到版本库并经过评审。

一个实用做法：把 AI 生成的内容一律放在「待评审」分支，通过编译、评审、试运行三步后才合入主干，并在提交信息中标注来源。

第七章 工具使用指南与下载链接

7.1 单元测试与 Mock

- Unity —— 极轻量 C 单元测试框架，整个框架只有 `unity.c/unity.h` 两个文件，可直接放进任何裸机工程。

github.com/ThrowTheSwitch/Unity

- CMock —— 依据头文件自动生成 Mock 代码，支持参数校验、返回值队列、回调插件。随 Ceedling 自动安装。

github.com/ThrowTheSwitch/CMock

- Ceedling —— 官方文档与安装指南（Ruby 3 环境，命令 `gem install ceedling`）。

throwtheswitch.github.io/Ceedling

- CppUTest —— C/C++ 单元测试框架，自带内存泄漏检测。

cpputest.github.io

- GoogleTest —— C++ 项目主流测试框架。

github.com/google/googletest

Ceedling 常用命令速查：

<code>ceedling new <项目名></code>	# 创建工程骨架（ <code>src/ + test/ + project.yml</code> ）
<code>ceedling test:all</code>	# 编译并运行全部单元测试
<code>ceedling test:<文件名></code>	# 只跑单个测试文件，如 <code>test:fan_ctrl</code>
<code>ceedling gcov:all</code>	# 生成覆盖率报告
<code>ceedling plugin:list</code>	# 查看可用插件
<code>ceedling module:create[foo]</code>	# 生成 <code>src/foo.c + src/foo.h + test/test_foo.c</code> 三件套
<code>ceedling clobber</code>	# 清理构建产物

7.2 静态分析与代码规范

- Cppcheck —— 开源 C/C++ 静态分析，支持 MISRA 附加规则包。

cppcheck.sourceforge.io

- clang-tidy —— 基于 Clang 前端的可扩展检查器。

clang.llvm.org/extra/clang-tidy

- PC-lint Plus —— 商业深度静态分析，MISRA 覆盖全面。

www.gimpel.com

- SonarQube —— 代码质量平台（C/C++ 插件为商业版）。

www.sonarsource.com

常用命令：

```
# Cppcheck: 全量检查，发现任何问题即返回非零（适合做 CI 门禁）
cppcheck --enable=all --inline-suppr --std=c99 \
  --suppress=missingIncludeSystem \
  -I src/hal -I src/middleware -I src/app \
  --error-exitcode=1 src/
```

```
# clang-tidy: 按 MISRA/可读性规则集检查
clang-tidy src/**/*.c -- -I src/hal -I src/middleware -std=c99

# 编译告警门禁 (强烈建议 -Werror)
gcc -std=c99 -Wall -Wextra -Wconversion -Wsign-conversion -Werror -c src/...
```

7.3 覆盖率

- gcovr —— gcov 的报告生成器, 支持 HTML/Cobertura/SonarQube 等格式, `pip install gcovr`。
gcovr.com
- gcovr 源码仓库 —— 安装与问题反馈。
github.com/gcovr/gcovr
- LCOV —— 经典的 gcov 前端, 生成 HTML 报告。
github.com/linux-test-project/lcov

```
# 1) 编译时插桩
gcc -std=c99 -O0 -g --coverage -I src -o test_runner test.c src/*.c

# 2) 运行测试
./test_runner

# 3) 生成报告 (HTML 明细 + 控制台摘要)
gcovr --html-details coverage.html --exclude '.*test.*'
gcovr --print-summary

# 4) 门禁: 语句 100%, 分支 95%
gcovr --fail-under-line 100 --fail-under-branch 95
```

7.4 编译器、调试与仿真

- Arm GNU Toolchain 下载页 —— arm-none-eabi-gcc 官方发布 (含 15.x/14.x/13.x 各版本)。
developer.arm.com/downloads/-/arm-gnu-toolchain-downloads
- QEMU 下载页 —— 系统级模拟器, 可跑裸机固件。
www.qemu.org/download
- Renode 官网 —— SoC 级仿真框架, 支持外设建模与多节点网络。
renode.io
- Renode 每日构建包 —— 各平台安装包直链。
builds.renode.io
- OpenOCD —— 开源调试服务器, 配合 ST-Link / J-Link / CMSIS-DAP。
openocd.org
- SEGGER 下载页 —— J-Link 驱动、Ozone 调试器、SystemView 跟踪工具。
www.segger.com/downloads
- Percepio Tracealyzer —— RTOS 运行时可视化分析。

percepio.com

```
# 目标机调试: OpenOCD + GDB
openocd -f interface/stlink.cfg -f target/stm32f4x.cfg
arm-none-eabi-gdb fw.elf -ex "target extended-remote :3333" -ex "load" -ex "continue"

# Renode 快速上手 (Linux 便携包)
mkdir renode && cd renode
wget https://builds.renode.io/renode-latest.linux-portable.tar.gz
tar xf renode-latest.linux-portable.tar.gz --strip-components=1
export PATH="`pwd`: $PATH"
renode --console
```

7.5 自动化与 CI

- pytest —— Python 测试框架，嵌入式系统测试首选驱动。
pypi.org/project/pytest
- pyserial —— 串口通信库，用于驱动真实设备。
pypi.org/project/pyserial
- Robot Framework —— 关键字驱动验收测试框架。
robotframework.org
- Jenkins 下载页。
www.jenkins.io/download
- GitLab CI 文档。
docs.gitlab.com/ci
- GitHub Actions。
github.com/features/actions
- Ceedling Docker 镜像 (MadScienceLab) —— 免配置 Ruby 环境。
hub.docker.com/r/throwtheswitch/madsciencelab

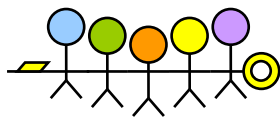
7.6 商业工具

- VectorCAST —— 嵌入式测试自动化平台，含 DO-178C / ISO 26262 工具鉴定包。
www.vectorcast.com
- Parasoft C/C++test —— 单元测试 + 静态分析 + 覆盖率 + 合规报告一体化。
www.parasoft.com/products/c-cpptest
- LDRA Testbed / TBvision —— 静态动态分析与需求追溯。
www.ldra.com
- Cantata —— 自动生成测试资产、桩与 Harness。
www.qa-systems.com/products/cantata
- TESSY —— 嵌入式专用单元测试工具 (Hitex)。
www.hitex.com/tessy

- BullseyeCoverage —— C/C++ 覆盖率，交叉编译支持成熟。
www.bullseye.com
- Vector CANoe —— 车载总线仿真与分析。
www.vector.com/canoe

7.7 缺陷与用例管理

- Jira —— 缺陷与任务管理。
www.atlassian.com/software/jira
- TestRail —— 测试用例管理与执行跟踪。
www.testrail.com
- 禅道 —— 国产开源项目管理与缺陷跟踪。
www.zentao.net



附录

附录 A 嵌入式测试准入准出检查清单

检查项	达标标准	常见失败原因
编译告警	-Werror 下零告警	第三方库未隔离、历史遗留代码
静态分析	强制规则零违反，建议规则有豁免记录	未配置 MISRA 规则集
单元测试语句覆盖	100%	防御式分支未测、错误处理路径未测
单元测试分支覆盖	≥ 95%（安全关键 100%）	状态机自迁移分支遗漏
MC/DC 覆盖	安全等级 C/D 要求 100%	复合条件未拆解
集成测试	接口组合 100%，错误注入 100% 通过	只测了正常路径
系统测试需求覆盖	100% 需求有对应用例且通过	需求未编号，无法追溯
长稳测试	≥ 72 h（或等效加速）无异常	未监控内存/计数器
追溯矩阵	需求-用例-结果-缺陷四向可追溯	用例命名无规则
资源余量	Flash/RAM/栈余量 ≥ 20%	覆盖率插桩忘记关闭就出包

附录 B 常见坑与规避

- 覆盖率插桩代码被打进量产固件：务必用不同的构建目标（Debug-Test / Release），并在 CI 中检查产物是否含 gcov 符号。
- 只在目标机上跑测试：目标机资源有限、执行慢，会导致团队放弃测试。逻辑类测试一律放主机，目标机只做关键回归。
- volatile 被优化掉导致测试与实机行为不一致：主机测试用 -O0，实机用 -Os，需保证至少一轮 -Os 下的实机验证。
- Mock 过度导致「测试通过但集成失败」：Mock 只用于 HAL 边界，业务模块之间不要用 Mock 替代。
- 时间相关的测试不稳定：把「当前时间」作为参数注入，杜绝在被测代码内部直接读取系统时钟。
- 忽略编译器版本：更换编译器后必须重跑全部测试，并对比 map 文件与资源占用。

附录 C 参考标准与扩展阅读

标准/文档	适用领域	核心要求
MISRA C:2012 / C:2023	汽车与高安全 C 代码	强制/建议/咨询三级规则，需记录豁免

CERT C	安全编码	聚焦可被利用的安全缺陷
ISO 26262	汽车功能安全	ASIL 等级决定覆盖率要求 (B:分支, C/D:MC/DC)
DO-178C	机载软件	A 级要求 MC/DC 100% 与需求覆盖
IEC 61508	工业功能安全	SIL 等级与验证活动要求
IEC 62304	医疗器械软件	软件生命周期与风险管理
ASPICE	汽车软件过程改进	过程能力等级评估, 含测试过程域