

AC784xx SDK Customer Development Guide

文档版本: 1.0

发布日期: 2026-04-29

© 2013 - 2026 杰发科技 本文档包含杰发科技的专有信息。未经授权，严禁复制或披露本文档包含的任何信息。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。

修订信息

版本	发布日期	内容
1.0	2026-04-29	整合所有SDK模块用户开发指南，内容覆盖AC784XX全系

版权声明

本文档包含杰发科技的机密信息。禁止未经授权使用或披露本手册包含的信息。

对因未经杰发科技授权而全部或部分披露此文档内容而给杰发科技带来的任何损失或损害，杰发科技将追究责任。

杰发科技保留对此处任何信息进行更改的权利，此处的信息如有变更，恕不另行通知。

杰发科技对使用或依赖此处包含的信息不承担任何责任。

本文档的所有信息均“按原样”提供，不提供任何形式的明示、暗示、法定或其他形式的保证。

杰发科技明确拒绝对适销性，非侵权性和针对特定用途的适用性方面的所有暗示保证。

杰发科技对本手册可能使用、包含或提供的任何第三方软件不提供任何担保，并且用户同意仅向该等第三方寻求与此相关的任何担保索赔。

杰发科技对于根据用户规格或为符合特定标准或公开论坛而产生的任何交付物，也不承担任何责任。

目录

1. 项目整体信息
2. 软件系统架构
3. 子系统介绍
4. 系统配置
5. 系统启动与电源管理
6. 系统烧写与升级
7. 外设适配与调试
8. 系统调试与性能分析
9. 定制应用
10. 用户开发接口
11. 用户开发示例
12. 参考文档

1. 项目整体信息

本文主要面向使用 AC784xx 系列产品进行产品开发的工程技术人员，提供软件开发所需的接口定义、配置方法，帮助用户快速构建稳定可靠的汽车电子应用。

SDK 软件包是一套完整的软件驱动库集合，旨在简化基于 AC784xx 平台的应用程序开发过程。该软件包提供了标准化的硬件抽象层接口，使开发人员能够专注于应用层逻辑的实现，而无需深入了解底层硬件细节。

与 SDK 对应的还有 MCAL 软件包，为需要符合 AUTOSAR 标准的项目提供底层驱动支持，不在本文介绍范围内。

1.1 产品系列概览

AC784xx 系列四款产品共享同一份软件包，软件架构、API 接口、驱动逻辑完全一致，差异通过 `Features.h` 宏或 EB Tresos 配置自动适配，客户应用代码无需为不同型号单独维护。

产品型号	典型应用场景	最高主频	CAN	UART	PWM	SENT	HSM/MRAM	低功耗模式
AC7840	通用车规 MCU	120 MHz	4	4	6	—	—	RUN/ VLPR/ STOP
AC7840E	低成本 / 传感器	120 MHz	2	3	4	2	—	RUN (无 VLPR/ STOP)
AC7842	扩展型号	120 MHz	—	—	—	—	—	—
AC7843	高性能域控	180 MHz	6	8	8	4	<input checked="" type="checkbox"/>	RUN/ VLPR/ STOP

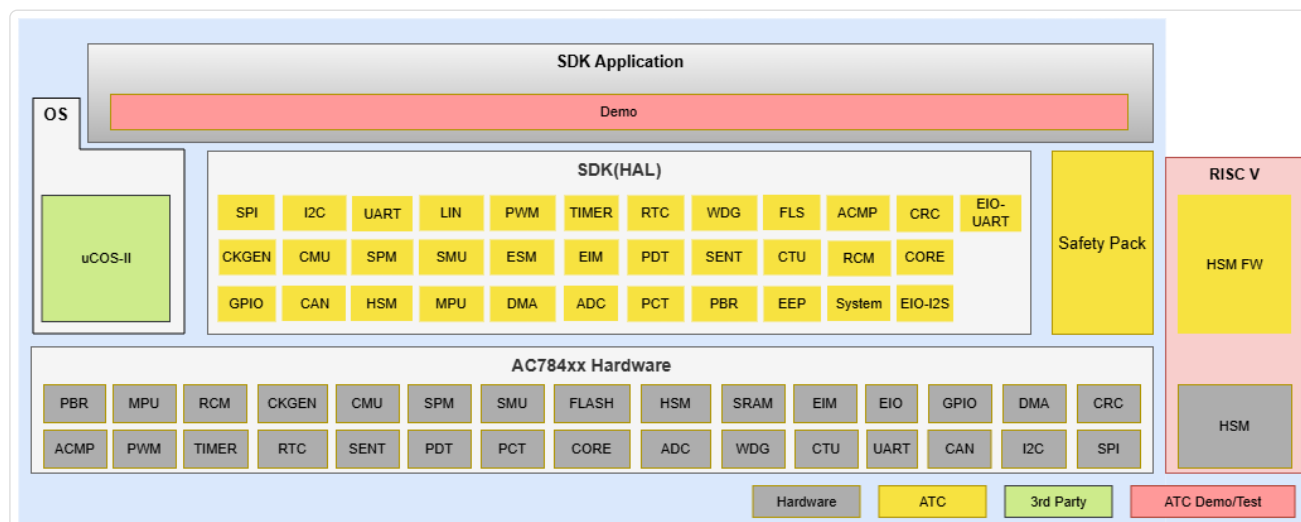
说明:

表中"—"表示规格待补充或与同系列一致；具体章节中将对有差异的参数逐一标注。

2. 软件系统架构

AC784xx软件系统采用分层架构设计，将硬件相关的底层操作与应用层逻辑分离，提高软件的可维护性、可移植性和可重用性。整体架构遵循模块化设计原则，各层之间通过明确定义的接口进行通信，确保系统的稳定性和可靠性。

下图展示了AC784xx软件系统的整体架构：



系统架构分为三层：

1. Application，应用层是用户开发的应用程序所在的位置，直接面向具体的业务功能和产品需求。该层基于SDK提供的标准化接口进行开发，无需关注底层硬件细节。
2. SDK（HAL），该层为我司提供的产品功能软件实现，包括硬件寄存器操作及部分业务逻辑处理。与该层平级还有两个组件：OS & Safety Pack，其中：
 - OS 选用uCOS-II，重点是辅助SDK 软件包的软件功能测试，我司未对uCOS-II进行验证，不保证其质量。
 - Safety Pack 为芯片功能安全机制的软件实现，与SDK层和MCAL软件包保持松耦合设计，可以根据具体项目的安全需求选择性地集成和使用。
3. Hardware，硬件层指AC784xx芯片本身及其集成的各种硬件模块，包括处理器内核、存储器、通信接口、模拟外设等。

3. 子系统介绍

3.1 Communication

3.1.1 LIN

AC784xx系列MCU中LIN模块基于UART硬件实现，通过HAL层封装LIN协议栈，支持情况存在差异，详细对比如下：

特性	AC7840	AC7840E	AC7842	AC7843
LIN 通道数	4	3	4	8
主节点模式	✓	✓	✓	✓
从节点模式	✓	✓	✓	✓
自动波特率检测	✓	✓	✓	✓
增强型校验和	✓	✓	✓	✓
经典型校验和	✓	✓	✓	✓
睡眠/唤醒管理	✓	✓	✓	✓
超时检测	✓	✓	✓	✓

说明:

- **LIN 通道数来源**：各型号 `AC78xxx_Features.h` 中的 `UART_INSTANCE_MAX` 定义（LIN 基于 UART 实现）
- **AC7840E 特殊性**：虽然 UART 实例较少，但保留了完整的 LIN 协议功能
- **AC7843 扩展**：提供 8 个 LIN 通道，适用于复杂车身网络拓扑
- 所有支持 LIN 的型号协议版本、校验算法、状态机完全一致

3.1.1.1 支持的功能

1. 基础通信功能

- **主/从节点模式**：支持配置为主节点（Master）或从节点（Slave）
- **帧传输与接收**：支持 LIN 2.x 标准帧格式（Break + Sync + PID + Data + Checksum）
- **校验和模式**：支持增强型校验和（Enhanced）和经典型校验和（Classic）
- **PID 校验处理**：支持计算和验证 PID 校验位

- **自动波特率检测**：从节点可根据主节点 Sync 字节自动调整波特率

2. 网络管理

- **睡眠模式**：降低功耗，保持唤醒检测能力
- **唤醒信号**：发送和检测 LIN 总线唤醒信号
- **空闲状态**：从睡眠模式退出，准备正常通信

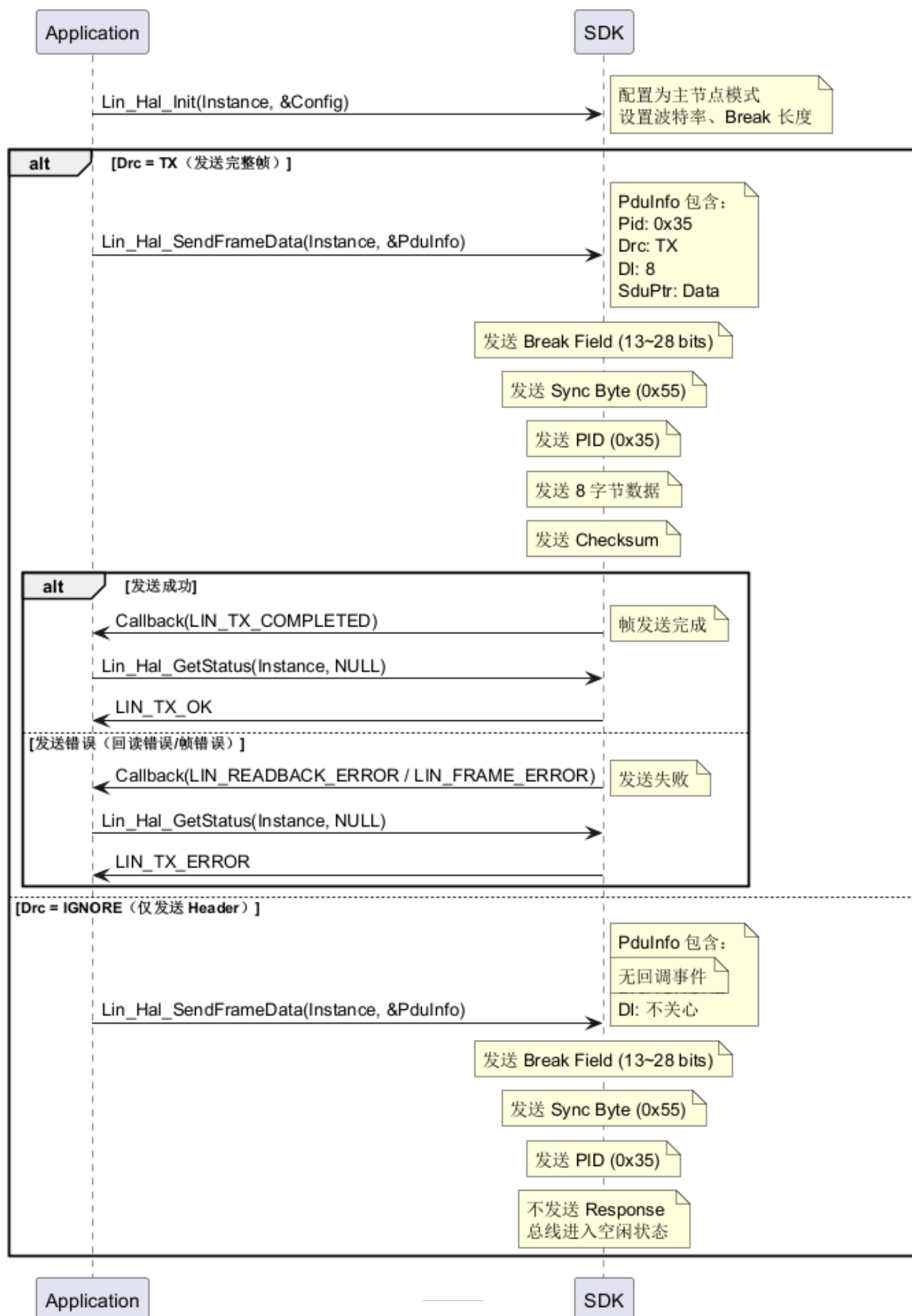
3. 诊断与保护

- **超时检测**：可配置超时计数器，防止总线挂死
- **错误检测**：支持同步错误、PID 错误、校验和错误、帧错误、回读错误等多种错误检测
- **传输中止**：可随时中止当前传输，恢复总线空闲

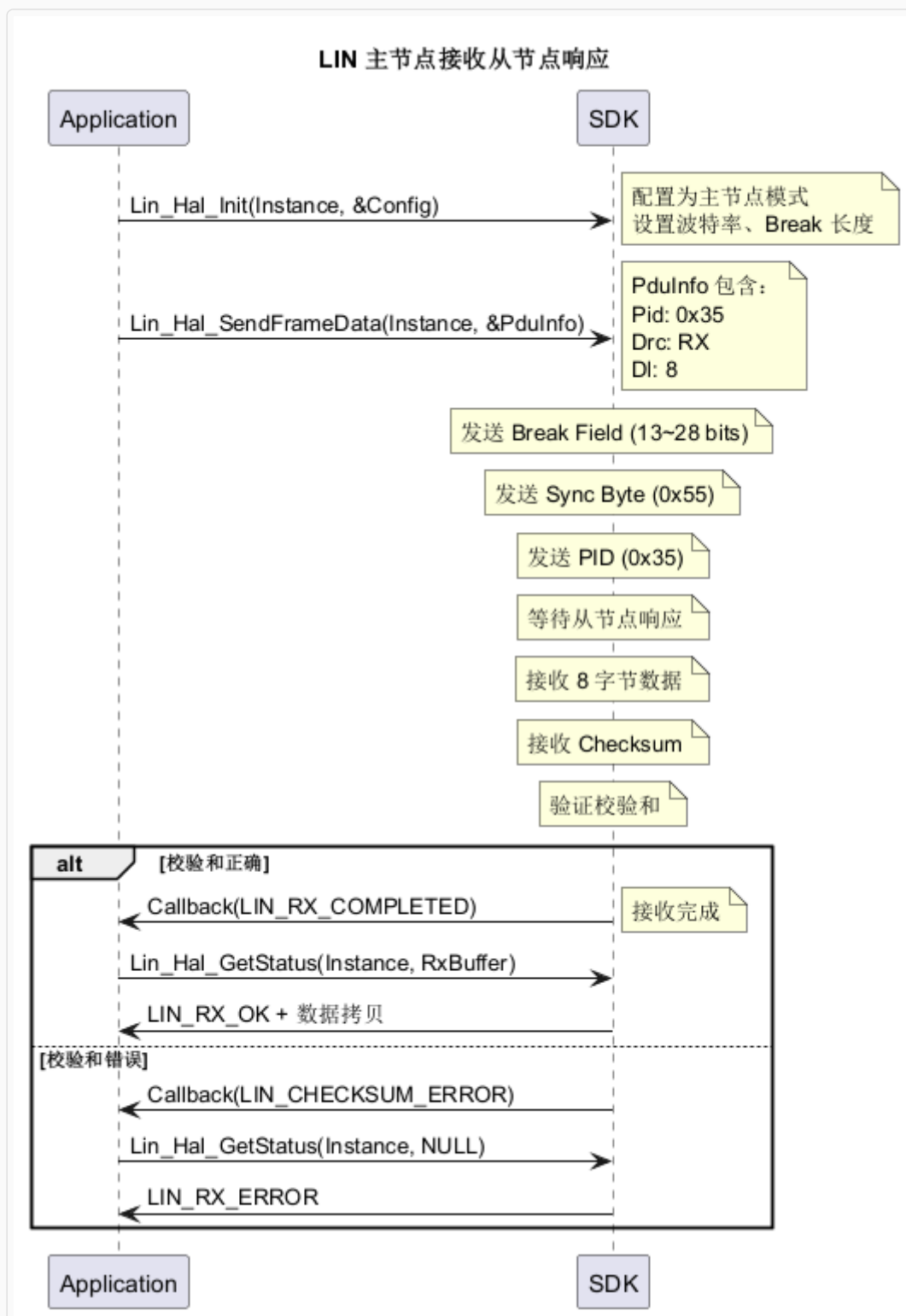
典型通信流程时序图

1. 主节点发送帧 (Header + Response)

LIN 主节点发送完整帧

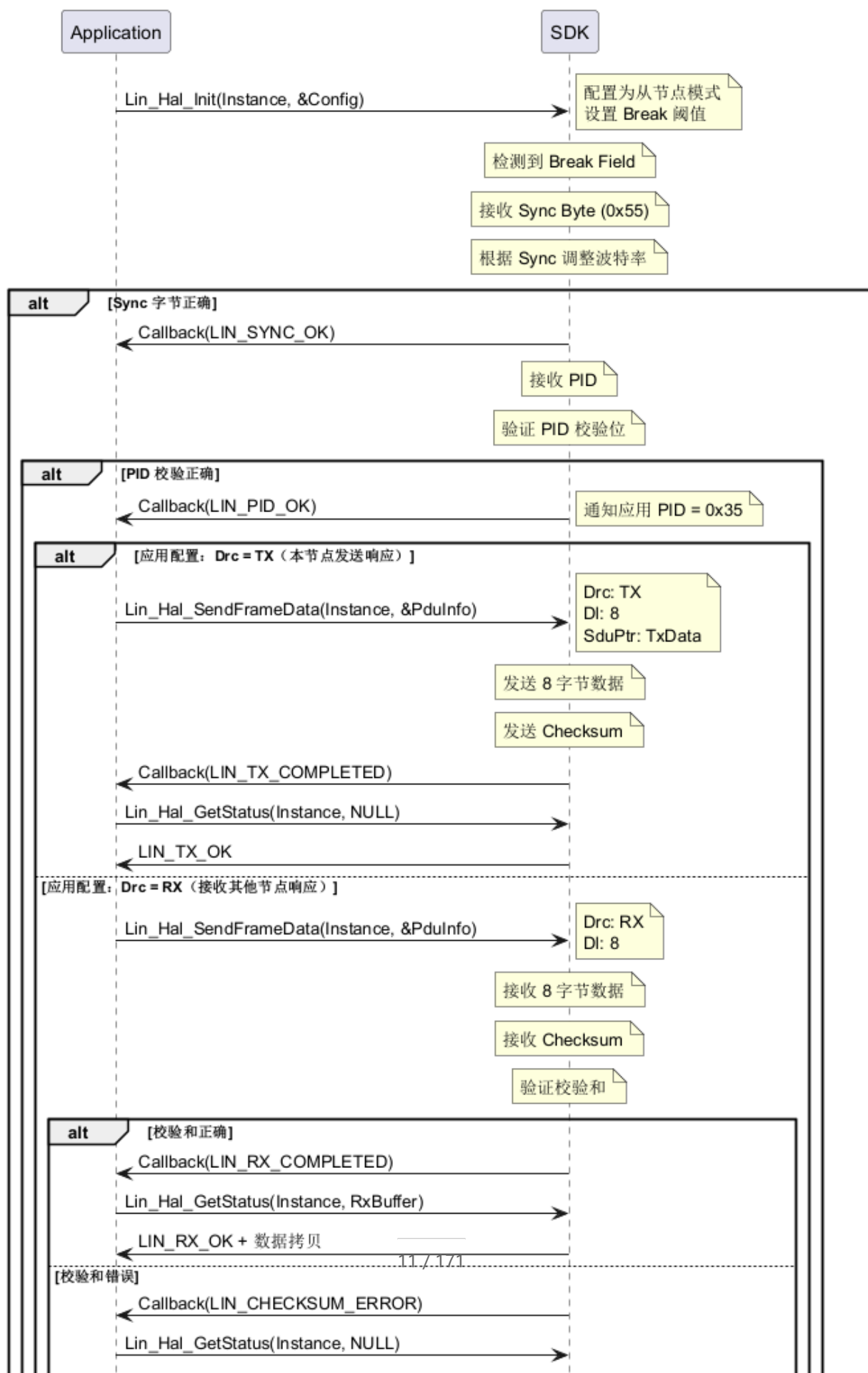


2. 主节点接收帧 (Header + Response)

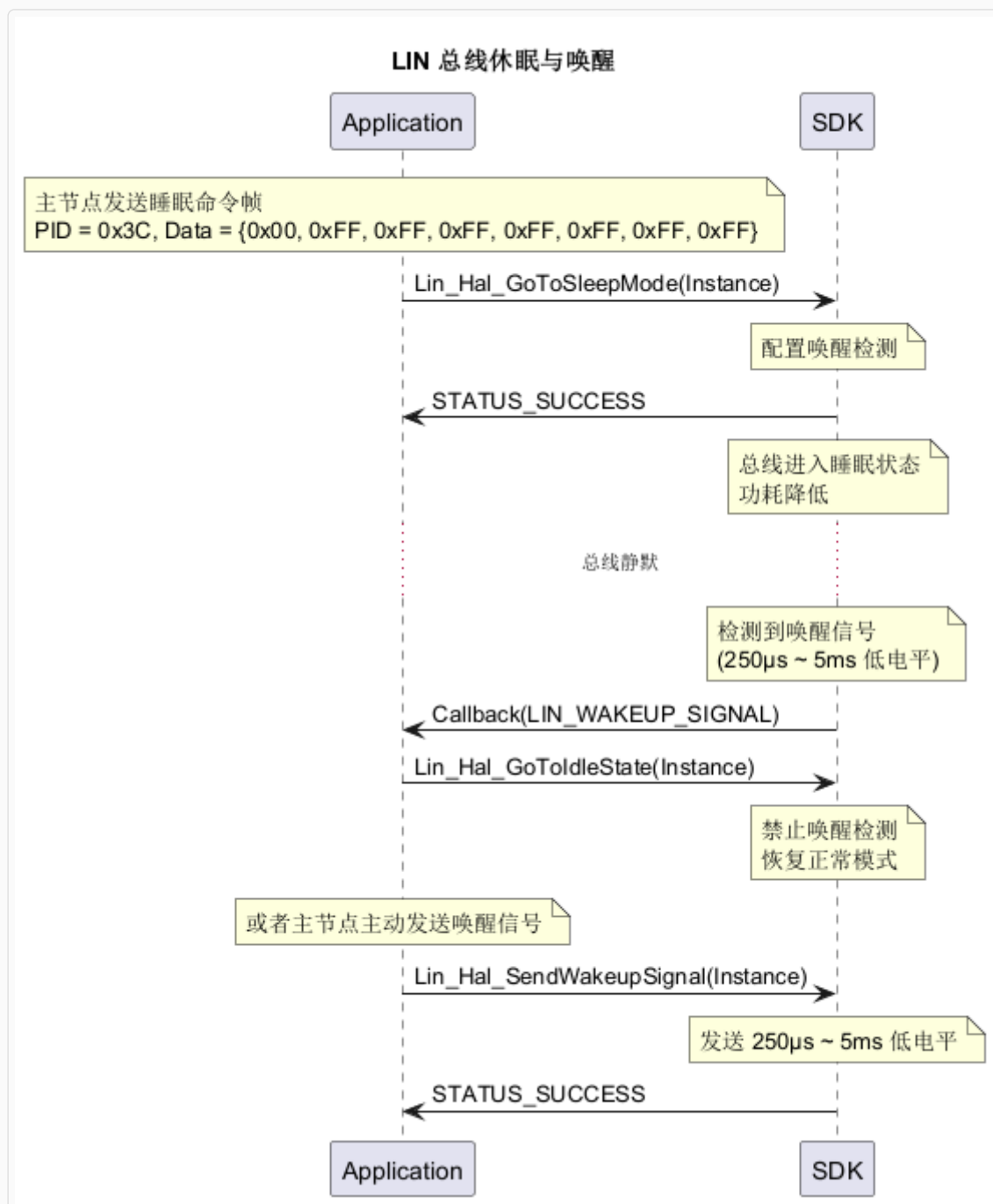


3. 从节点响应帧 (三种响应类型)

LIN 从节点响应处理 (TX/RX/IGNORE)



4. 睡眠与唤醒流程



3.1.1.2 与标准或规范的差异

AC784xx LIN HAL 层实现遵循 **LIN 2.x 规范** (ISO 17987 系列标准)，与标准协议无差异。

3.1.1.3 静态配置项说明

LIN 模块在编译阶段需要配置以下宏定义，位于 `device/config/Conf_AC784xx.h` 文件中。

通道编译控制

配置宏	说明	取值	适用产品
<code>CONFIG_LIN0_ENABLE</code>	LIN 通道 0 编译开关	0: 禁止编译 1: 使能编译	所有型号
<code>CONFIG_LIN1_ENABLE</code>	LIN 通道 1 编译开关	0: 禁止编译 1: 使能编译	所有型号
<code>CONFIG_LIN2_ENABLE</code>	LIN 通道 2 编译开关	0: 禁止编译 1: 使能编译	AC7840 / AC7842 / AC7843
<code>CONFIG_LIN3_ENABLE</code>	LIN 通道 3 编译开关	0: 禁止编译 1: 使能编译	AC7840 / AC7842 / AC7843
<code>CONFIG_LIN4_ENABLE</code>	LIN 通道 4 编译开关	0: 禁止编译 1: 使能编译	AC7843 专用
<code>CONFIG_LIN5_ENABLE</code>	LIN 通道 5 编译开关	0: 禁止编译 1: 使能编译	AC7843 专用
<code>CONFIG_LIN6_ENABLE</code>	LIN 通道 6 编译开关	0: 禁止编译 1: 使能编译	AC7843 专用
<code>CONFIG_LIN7_ENABLE</code>	LIN 通道 7 编译开关	0: 禁止编译 1: 使能编译	AC7843 专用

注意事项

- 禁用通道的状态结构体指针为 `NULL_PTR`，调用对应通道的 API 会触发断言失败
- LIN 与 UART 互斥**：LIN 基于 UART 硬件实现，如某通道配置为 LIN 使用，建议将对应的 UART 通道编译开关（`CONFIG_UARTx_ENABLE`）设为 0，避免同一硬件通道被重复配置
- 根据实际使用的通道数合理配置，可有效减少 RAM 占用

3.1.1.4 软件集成依赖

依赖模块

LIN HAL 层功能依赖以下模块正确配置：

依赖模块	依赖说明	初始化顺序要求
CKGEN	LIN 需要时钟源才能工作 波特率精度依赖时钟频率稳定性	必须先初始化后才能使用 LIN
PORT	LIN 需配置 UART_TX / UART_RX 引脚 为 UART 功能 外部还需连接 LIN 收发器	必须先配置后才能正常通信
RCM	用于 UART 模块复位	系统启动时自动初始化

3.1.1.5 软件限制/开发须知

重要限制

- 硬件实现方式** - LIN 基于 UART 硬件实现，每个 LIN 通道占用一个 UART 实例 - 使用 LIN 功能时，对应的 UART 通道不可再用于普通串口通信
- 超时管理要求** - 如使用超时功能，必须周期性调用 `Lin_Hal_TimeoutService`（推荐 1ms 周期） - 超时计数器在每次 `Lin_Hal_TimeoutService` 调用时递减 - 超时发生时触发 `LIN_TIMEOUT_ERROR` 事件并中止传输 - 超时值建议设置为：帧时间 × 1.5 ~ 2.0（如 19.2 Kbit/s 下 8 字节帧约 5ms，可设置 10ms 超时）

3.1.2 UART

特性	AC7840	AC7840E	AC7842	AC7843
UART 通道数	4	3	4	8
DMA 传输支持	✓	✓	✓	✓
中断传输支持	✓	✓	✓	✓
硬件流控 (RTS/CTS)	✓	✓	✓	✓
RS485 模式	✓	✓	✓	✓
IrDA 模式	✓	✓	✓	✓
单线半双工模式	✓	✓	✓	✓
地址过滤/唤醒	✓	✓	✓	✓

说明:

- UART 通道数来源：各型号 `AC78xxx_Features.h` 中的 `UART_INSTANCE_MAX` 定义

- **AC7840E 限制**: 仅提供 3 个 UART 通道 (UART0~UART2)
- **AC7843 扩展**: 提供 8 个 UART 通道 (UART0~UART7), 适用于多路通信需求
- 所有型号的 UART 功能特性 (帧格式、流控、RS485、IrDA) 完全一致

3.1.2.1 支持的功能

AC784xx UART HAL 层提供以下功能:

1. 基础通信功能

• 初始化与反初始化

- 配置波特率、数据位、停止位、校验位
- 选择传输方式 (中断/DMA)
- 注册发送/接收回调函数

• 数据收发

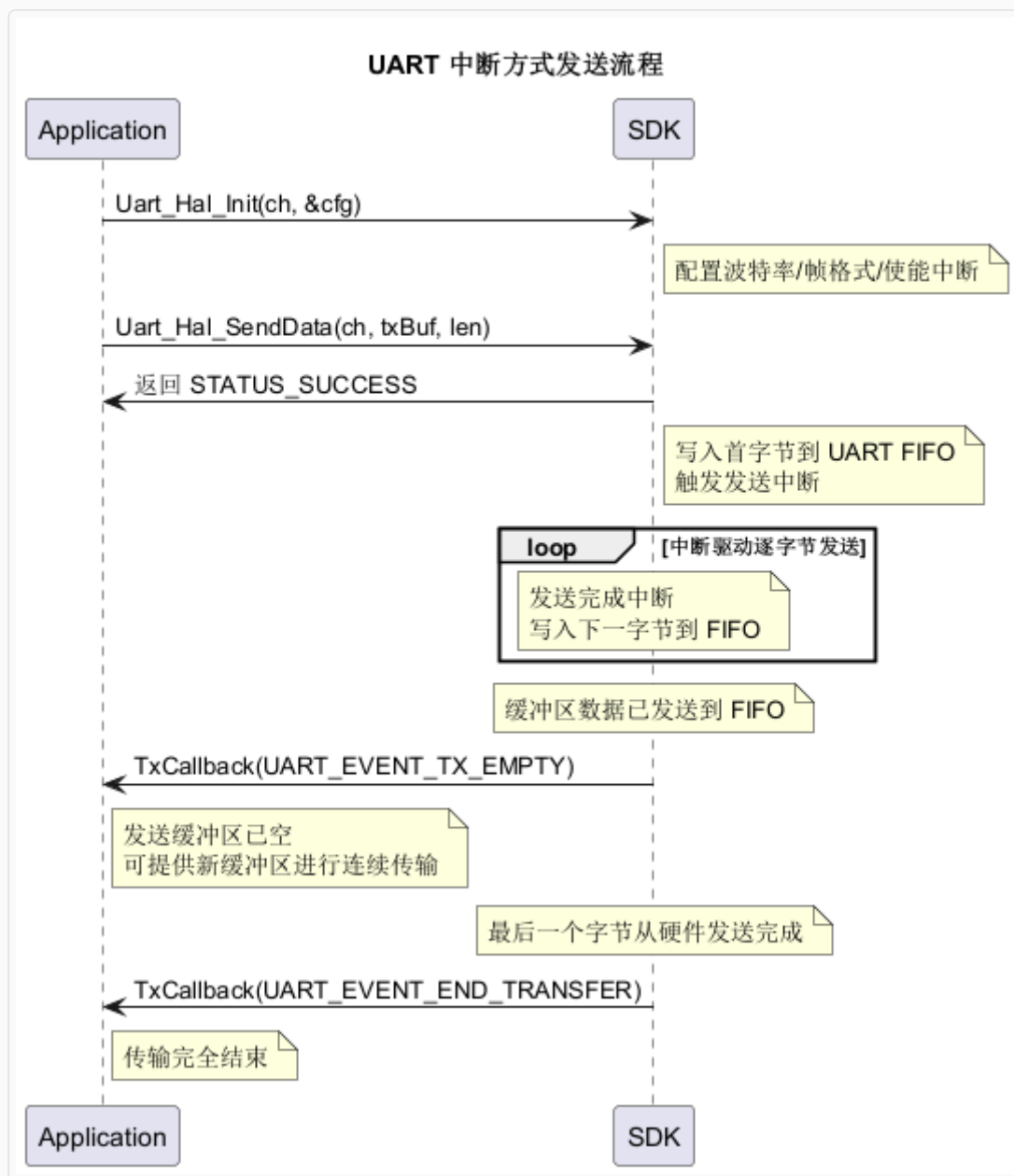
- 非阻塞发送/接收 (中断或 DMA 方式)
- 轮询发送/接收 (不使用中断)
- 阻塞发送/接收 (带超时控制)

• 传输状态监控

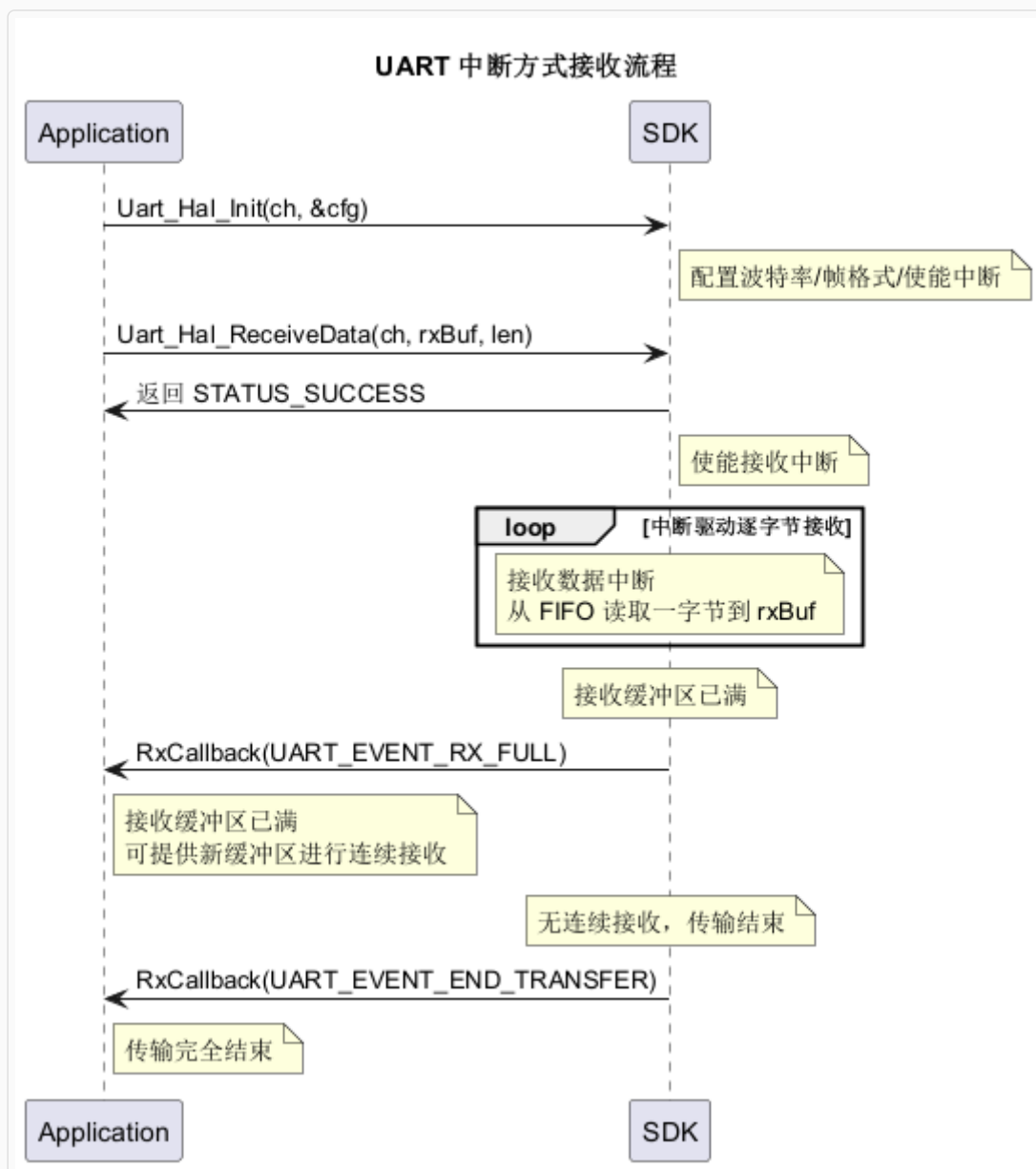
- 查询发送/接收状态
- 获取剩余字节数

典型通信流程时序图

1. 中断方式发送流程

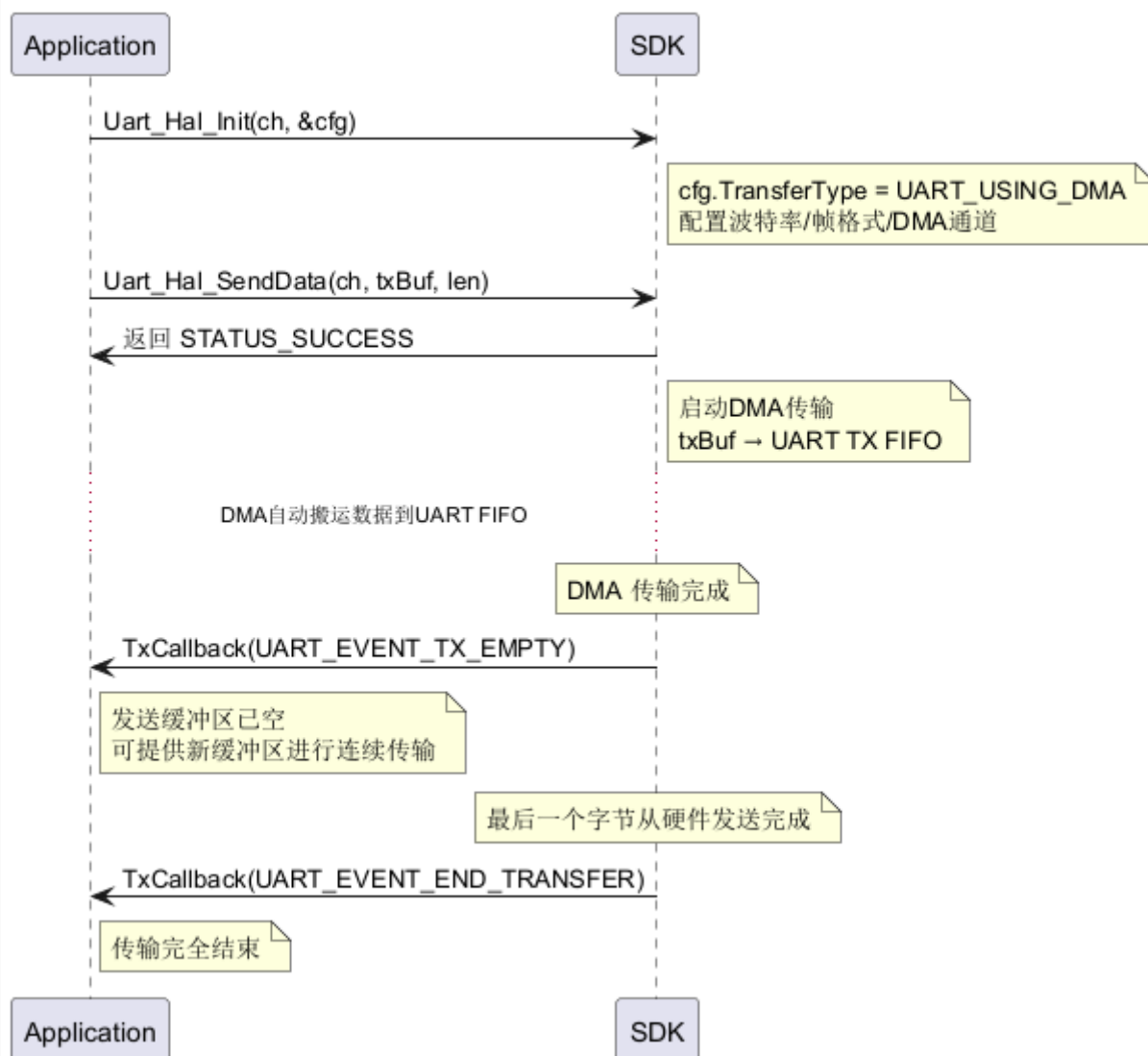


2. 中断方式接收流程



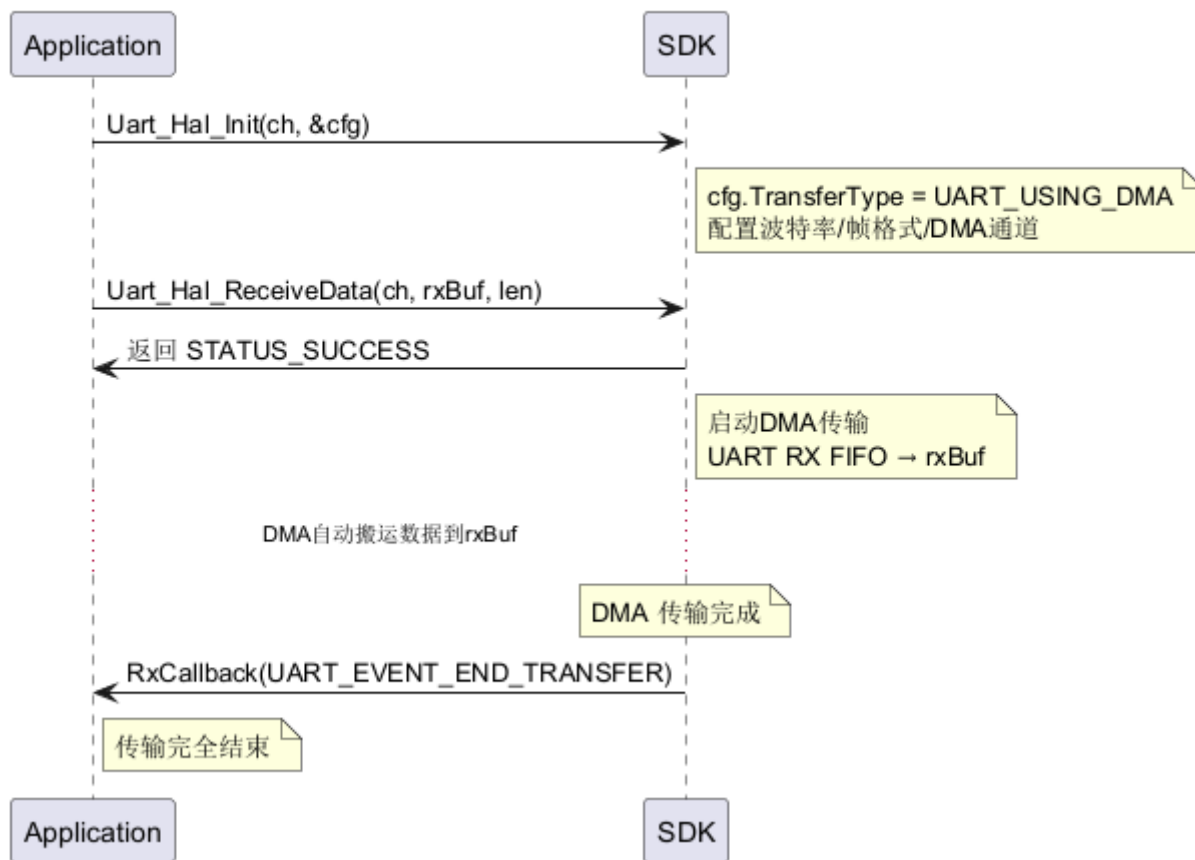
3. DMA 方式发送流程

UART DMA 方式发送流程

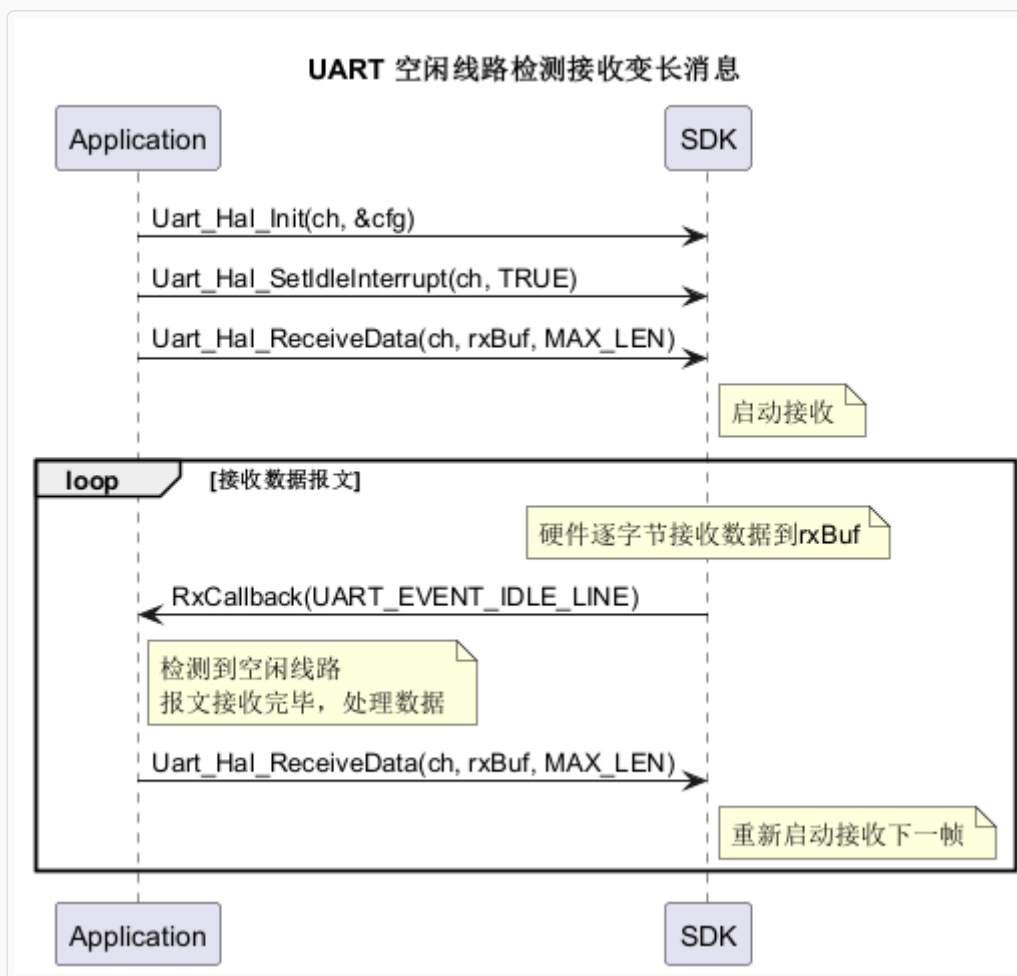


4. DMA 方式接收流程

UART DMA 方式接收流程



5. 空闲线路检测接收变长消息



3.1.2.2 与标准或规范的差异

AC784xx UART HAL 层实现遵循标准异步串行通信协议，与通用 UART 规范无差异。

3.1.2.3 静态配置项说明

UART HAL 层通过编译时宏定义进行静态配置，这些宏定义在 `device/config/Conf_AC784xx.h` 文件中设置。

通道使能配置

宏定义： `CONFIG_UARTx_ENABLE` ($x = 0 \sim 7$)

功能说明： 控制各 UART 通道是否参与编译，未使能的通道不会占用 RAM 和 ROM 资源。

配置值

- **1**：使能该通道，生成对应的状态结构体和驱动代码

- `0`：禁止该通道，节省内存资源

注意事项

- 禁用通道的状态结构体指针为 `NULL_PTR`，调用对应通道的 API 会触发断言失败
- 如果某通道配置为 LIN 使用，建议将对应的 UART 通道设为 0（参见"UART 与 LIN 互斥限制"）
- 根据实际使用的通道数合理配置，可有效减少 RAM 占用

3.1.2.4 软件集成依赖

依赖模块

UART HAL 层功能依赖以下模块正确配置：

依赖模块	依赖说明	初始化顺序要求
CKGEN	UART 需要时钟源才能工作 波特率计算依赖外设时钟频率	必须先初始化后才能使用 UART
PORT	UART TX/RX/RTS/CTS 引脚需配置为 UART 功能	必须先配置后才能正常通信
DMA	使用 DMA 传输方式时需要	如使用 DMA，需先初始化 DMA 模块

3.1.2.5 软件限制/开发须知

重要限制

- UART 与 LIN 互斥限制** - UART 和 LIN 共享同一硬件通道，**同一通道只能配置为 UART 或 LIN 之一** - 如果某通道已配置为 LIN 使用，则该通道无法用于 UART 通信 - 反之，如果某通道已配置为 UART 使用，则该通道无法用于 LIN 通信 - 配置时必须明确通道用途，避免重复配置导致功能冲突 - 建议在系统设计阶段规划好各通道的用途分配
- DMA 传输长度限制** - 使用 DMA 模式时,单次传输长度不得超过 **32768 字节**(32 KB) - 代码中定义为 `UART_USE_DMA_TRANSMIT_LEN_MAX`,超出限制会触发断言 - 大数据量传输需分片:在 `TxCallback` 的 `UART_EVENT_TX_EMPTY` 事件中继续发送下一片 - 中断模式无此限制,但传输效率较低,建议大数据量优先使用 DMA
- 9 位数据模式对齐要求** - 配置为 9 位数据位(`UART_9_BITS_PER_CHAR`)时,传输/接收长度必须是 **偶数** - 每个 9 位字符占用 2 字节存储空间(低字节存数据低 8 位,高字节存第 9 位) - 传入奇数长度会触发断言失败: `DEVICE_ASSERT((BitCountPerChar !=`

`UART_9_BITS_PER_CHAR) || (0U == (TxSize & 1U)))` - 示例:发送 100 字节数据时,实际发送 50 个 9 位字符

4. DMA 发送完成检测机制差异 - AC7840 特殊处理: DMA 传输完成后立即触发

`UART_EVENT_END_TRANSFER` 回调 - 其他型号: DMA 传输完成后先触发

`UART_EVENT_TX_EMPTY` 回调, 然后使能 TX COMPLETE 中断等待最后一个字节从硬件发送完成, 最后触发 `UART_EVENT_END_TRANSFER` 回调 - 影响: AC7840 上 DMA 发送仅有一次回调, 其他型号有两次回调 (TX_EMPTY + END_TRANSFER) - 建议: 应用层应统一以 `UART_EVENT_END_TRANSFER` 作为传输完全结束的判断标志

5. 轮询发送接口返回时机说明 - `Uart_Hal_SendDataPolling` 返回时, 数据已写入 UART FIFO, 但最后字节可能仍在硬件发送中 - 如需确保发送数据完全从 TX 引脚输出, 应使用中断/DMA 模式并等待 `UART_EVENT_END_TRANSFER` 回调

3.1.3 EIO_UART

本章介绍该项目在 AC7840 / AC7840E / AC7842 / AC7843 产品上, EIO UART 模块软件支持的功能特性、接口定义和配置方法, 旨在指导用户正确集成和使用 EIO UART 驱动。

特性	AC7840	AC7840E	AC7842	AC7843
EIO 模块数量	1	1	1	1
Shifter 资源总数	4	4	4	4
Timer 资源总数	4	4	4	4
最大 UART 通道数	2 个全双工	2 个全双工	2 个全双工	2 个全双工
DMA 传输支持	✓	✓	✓	✓
中断传输支持	✓	✓	✓	✓
轮询模式	✓	✓	✓	✓

资源分配说明

- 每个 EIO UART 通道的每个方向 (TX 或 RX) 需要占用 **1 个 Shifter** 和 **1 个 Timer**
- 全双工 UART 通道 (TX + RX) 需要 **2 个 Shifter** 和 **2 个 Timer**
- AC784xx 系列所有型号的 EIO 模块均包含 **4 个 Shifter** 和 **4 个 Timer**
- 因此, **最多支持 2 个全双工 UART**
- EIO 资源与其他 EIO 协议 (I2C、I2S) 共享, 需合理规划分配

3.1.3.1 支持的功能

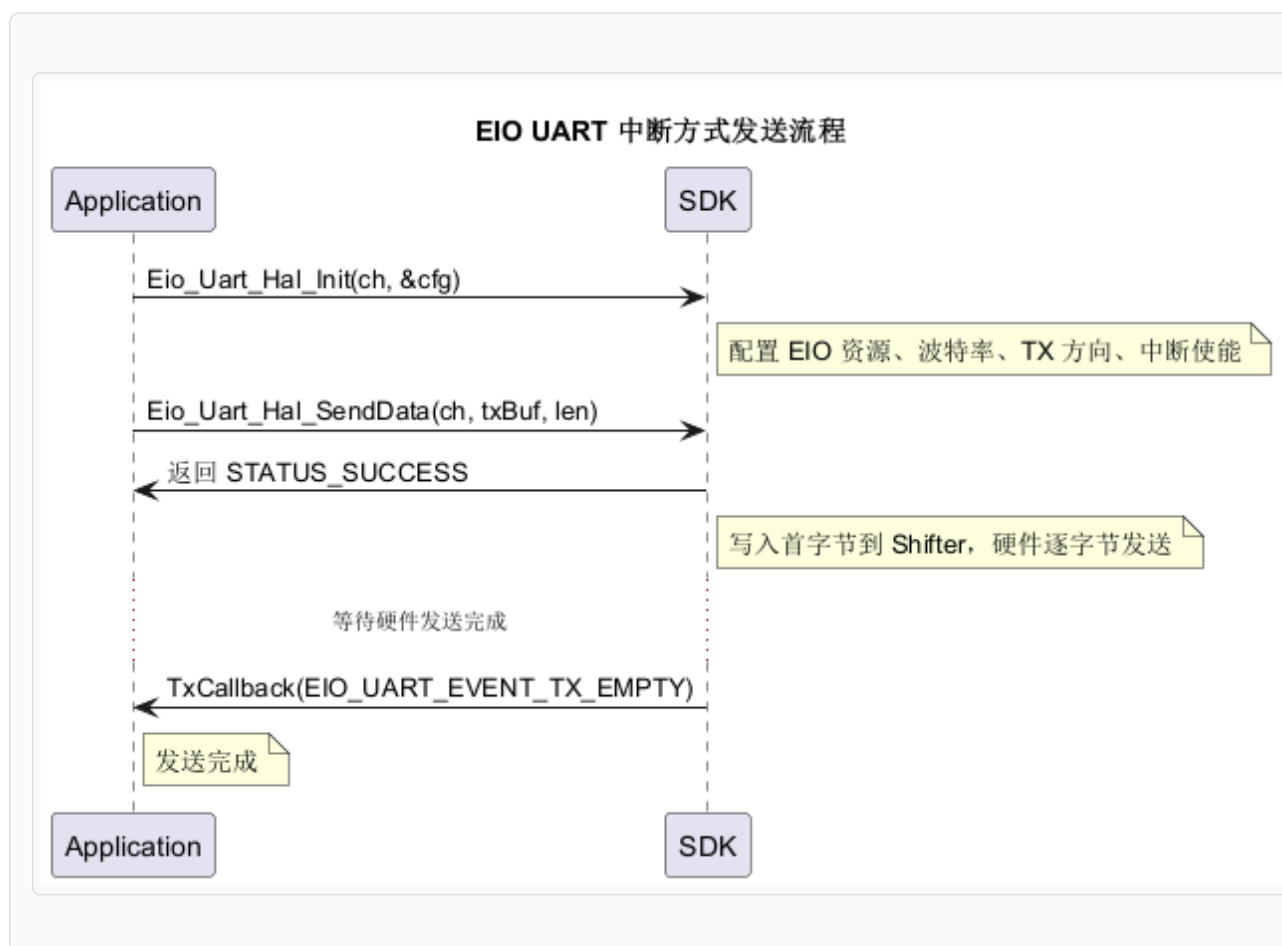
AC784xx EIO UART HAL 层提供以下功能：

1. 基础通信功能

- **初始化与反初始化** 配置 EIO UART 通道参数并启动驱动，或反初始化释放资源
- **数据收发** 支持非阻塞发送/接收，可查询传输状态和中止传输
- **波特率管理** 支持动态修改和查询波特率
- **缓冲区管理** 支持动态设置发送/接收缓冲区

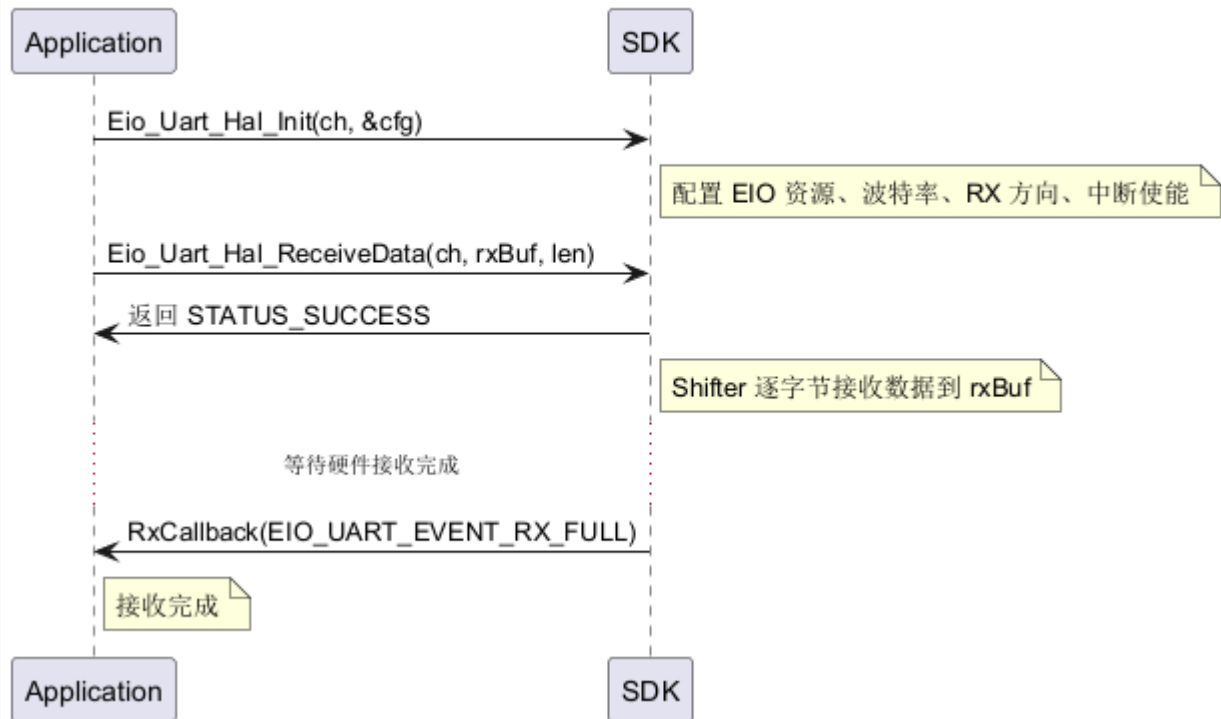
典型通信流程时序图

1. 中断方式发送流程



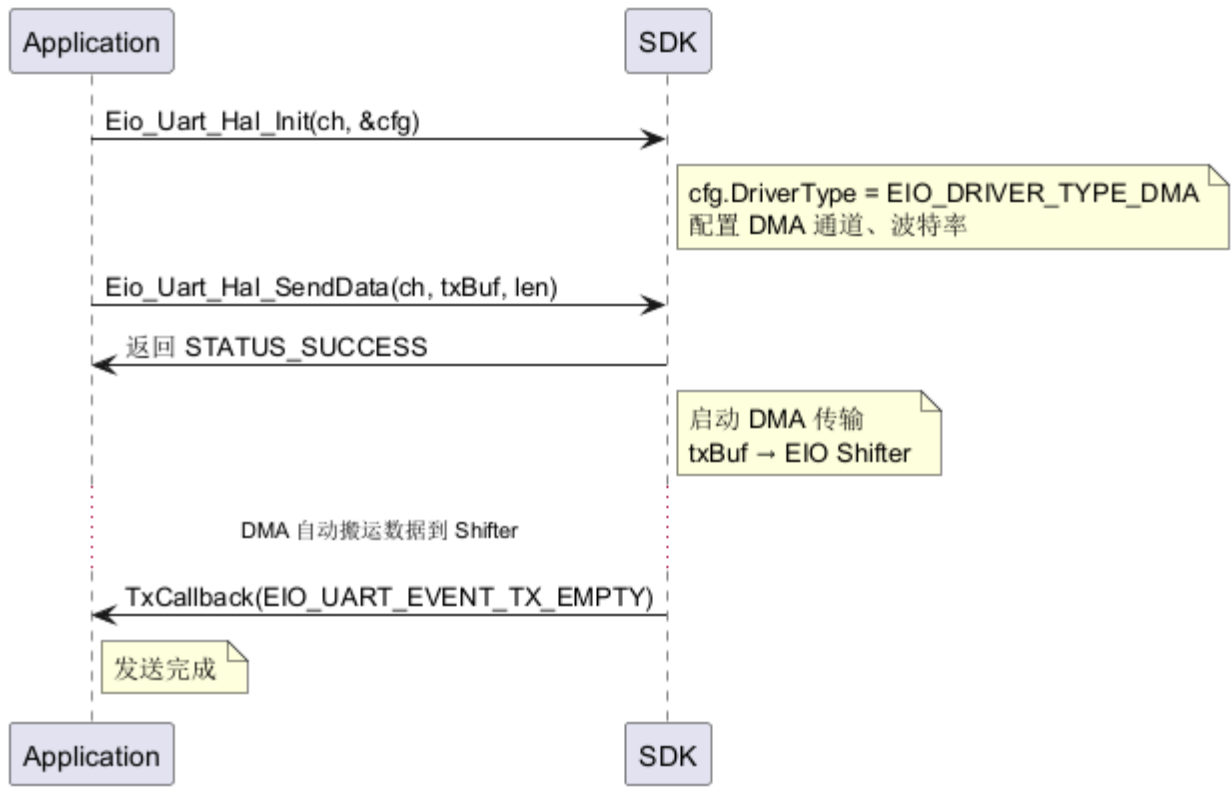
2. 中断方式接收流程

EIO UART 中断方式接收流程

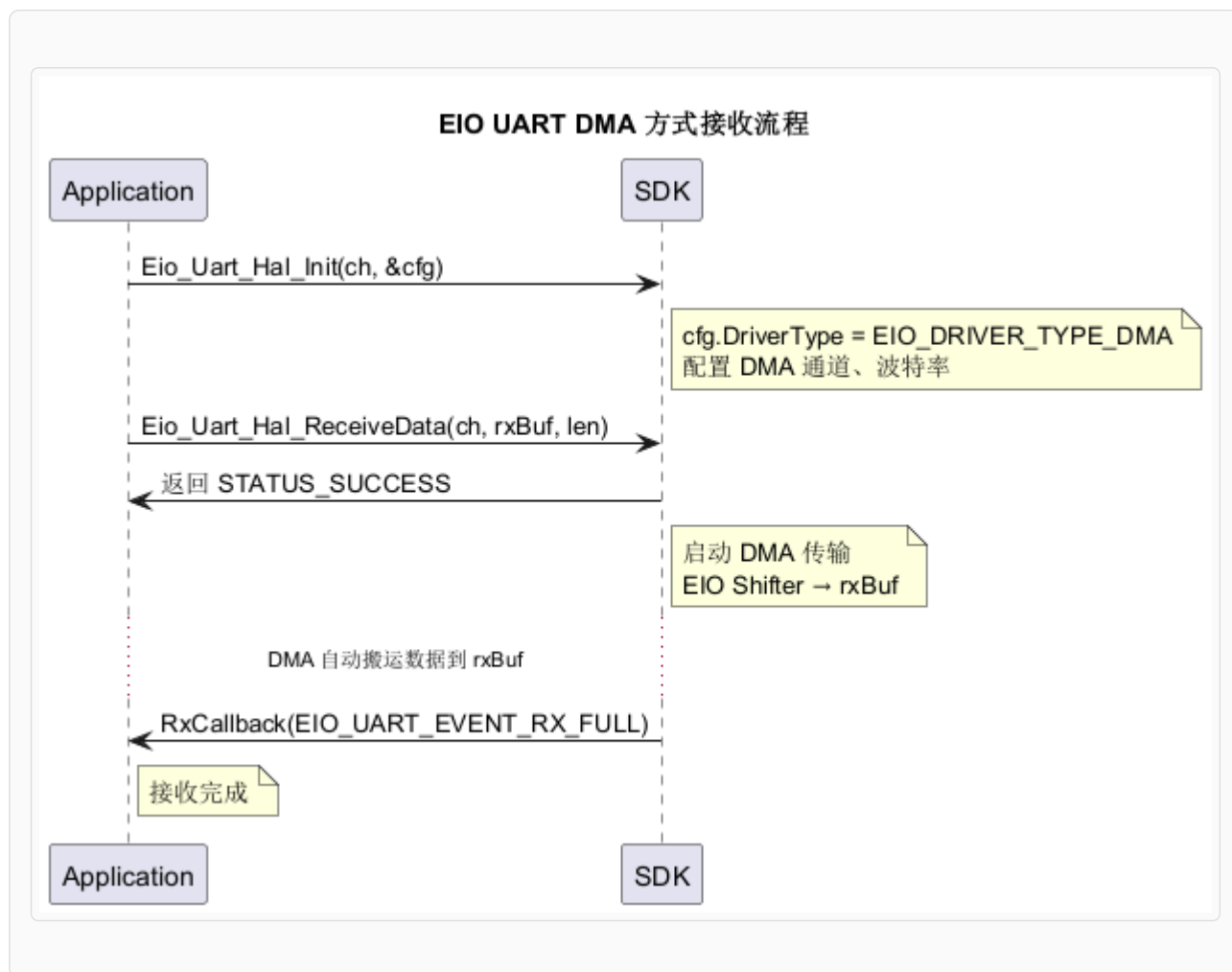


3. DMA 方式发送流程

EIO UART DMA 方式发送流程



4. DMA 方式接收流程



3.1.3.2 与标准或规范的差异

EIO UART 实现遵循标准异步串行通信协议（起始位、数据位、停止位），与通用 UART 帧格式无明显差异。

与传统硬件 UART 的差异

功能特性	传统 UART	EIO UART
分数波特率	支持	不支持（仅整数分频）
硬件流控（RTS/CTS）	支持	不支持
RS485 模式	支持	不支持
IrDA 模式	支持	不支持
自动校验检测	支持	不支持
空闲线路检测	支持	不支持
地址过滤/唤醒	支持	不支持

功能特性	传统 UART	EIO UART
DMA 传输	支持	支持
中断传输	支持	支持
轮询传输	支持	支持

说明:

- EIO UART 是纯净的异步串行数据传输实现，专注于基础收发功能
- 如果应用需要硬件流控、RS485、IrDA 等扩展特性，请使用传统 UART

3.1.3.3 静态配置项说明

通道使能配置

宏定义： `CONFIG_EUART0_ENABLE` / `CONFIG_EUART1_ENABLE`

定义位置： `device/config/Conf_AC784xx.h`

功能说明：控制各 EIO UART 通道是否参与编译，未使能的通道不会占用 RAM 和 ROM 资源。

配置值

```

#define CONFIG_EUART0_ENABLE (1) /* 1: 使能通道 0, 0: 禁用通道 0 */
#define CONFIG_EUART1_ENABLE (1) /* 1: 使能通道 1, 0: 禁用通道 1 */

```

配置说明

- `1`：使能该通道，生成对应的状态结构体和驱动代码
- `0`：禁用该通道，节省内存资源

注意事项

- 禁用通道的状态结构体指针为 `NULL_PTR`，调用对应通道的 API 会触发断言失败
- 根据实际使用的通道数合理配置，可有效减少 RAM 占用
- 最多可使能 2 个通道（通道 0 和通道 1）

3.1.3.4 软件集成依赖

依赖模块

EIO UART HAL 层功能依赖以下模块正确配置：

依赖模块	依赖说明	初始化顺序要求
CKGEN	EIO 模块需要外设时钟才能工作 波特率计算依赖 EIO 时钟频率	必须先初始化时钟后才能使用 EIO UART
PORT	EIO TX/RX 引脚需配置为 EIO 功能	必须先配置引脚复用后才能正常通信
DMA	使用 DMA 传输方式时需要	如使用 DMA，需先初始化 DMA 模块

3.1.3.5 软件限制/开发须知

重要限制

1. **Shifter/Timer 资源分配限制** - AC784xx 系列所有型号的 EIO 模块均包含 **4 个 Shifter** 和 **4 个 Timer** - 每个 EIO UART 方向 (TX 或 RX) **占用 1 个 Shifter 和 1 个 Timer** - **全双工通道 (TX + RX) 需要 2 个 Shifter/Timer** - **最多支持 2 个全双工 UART** - Shifter/Timer 索引范围：**0 ~ 3** - **不同通道的资源索引不得冲突**（如通道 0 使用 Shifter 0/1，通道 1 使用 Shifter 2/3） - 如果 EIO 同时用于其他协议（I2C、I2S），需合理规划 shifter 和 timer 资源分配

3.1.4 SENT

本章介绍该项目在 AC7840E / AC7842 / AC7843 产品上，SENT 模块软件支持的功能特性、接口定义和配置方法，旨在指导用户正确集成和使用 SENT 驱动。支持的产品有：AC7840E、AC7842、AC7843。AC7840 不包含 SENT 硬件模块，本章内容不适用。

特性	AC7840	AC7840E	AC7842	AC7843
SENT 通道数	—（不支持）	2	2	4
快通道数据接收	—	✓	✓	✓
慢通道串行消息	—	✓	✓	✓
SPC（短PWM码）发送	—	✓	✓	✓
CRC 校验	—	✓	✓	✓
暂停脉冲支持	—	✓	✓	✓

特性	AC7840	AC7840E	AC7842	AC7843
时间戳功能	—	✓	✓	✓
FIFO 深度	—	4	4	4

说明:

- **SENT 通道数来源:** 各型号 `AC78xxx_Features.h` 中的 `SENT_INSTANCE_MAX` 定义
- **AC7840 不支持:** AC7840 不包含 SENT 硬件模块, 无法使用 SENT 功能
- **AC7840E/AC7842:** 提供 2 个 SENT 通道 (SENT_CHANNEL_0、SENT_CHANNEL_1)
- **AC7843 扩展:** 提供 4 个 SENT 通道 (SENT_CHANNEL_0~SENT_CHANNEL_3) , 适用于多传感器采集场景
- 所有支持 SENT 的型号功能特性 (协议版本、CRC、时间戳) 完全一致

3.1.4.1 支持的功能

AC784xx SENT HAL 层提供以下功能:

1. 基础通信功能

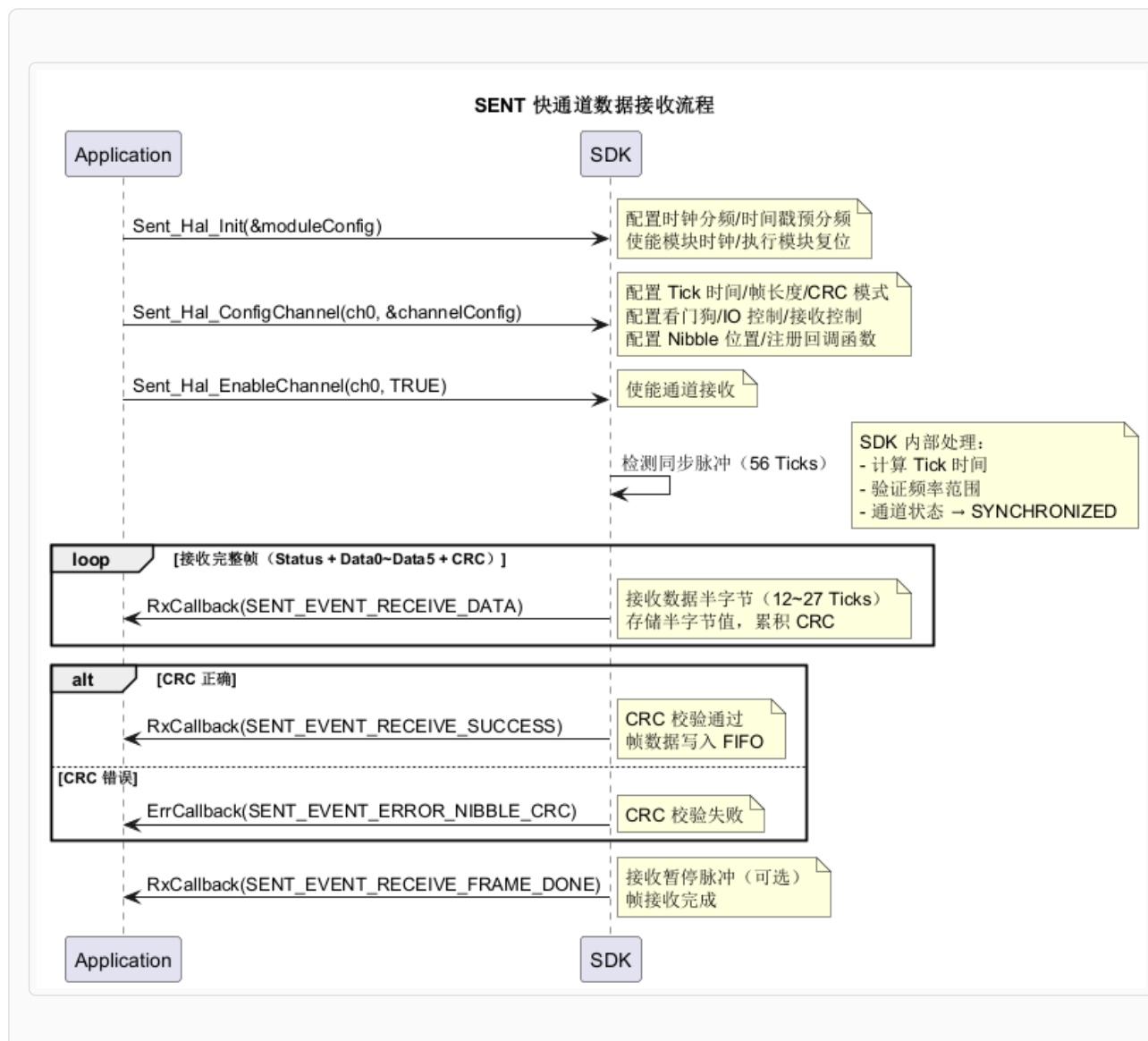
- **模块初始化与反初始化:** 初始化/反初始化 SENT 模块, 配置时钟分频和时间戳参数
- **通道配置与控制**
 - 配置 Tick 时间单位、帧长度、CRC 模式
 - 配置输入源选择、毛刺滤波、信号反转
 - 使能/禁止通道接收
 - 查询通道状态 (停止/初始化/运行/同步)
- **快通道数据接收**
 - 接收状态半字节和数据半字节 (1~8 个)
 - CRC 校验 (4位或6位)
 - 时间戳记录
- **慢通道串行消息接收**
 - 支持短串行消息 (Short Serial Message, 16 帧)
 - 支持增强串行消息 (Enhanced Serial Message, 18 帧)
 - 串行消息 CRC 校验

◦ 消息 ID 和数据解析

• **SPC 发送**：配置和触发短 PWM 码 (Short PWM Code) 发送

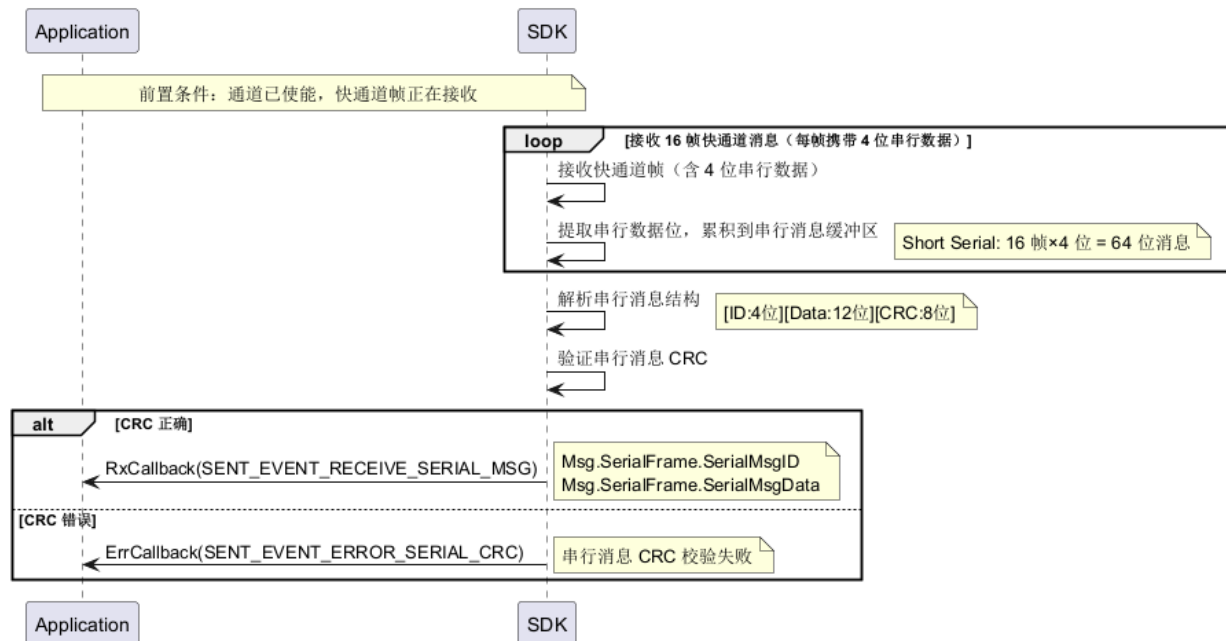
典型通信流程时序图

1. SENT 快通道接收 (成功路径)



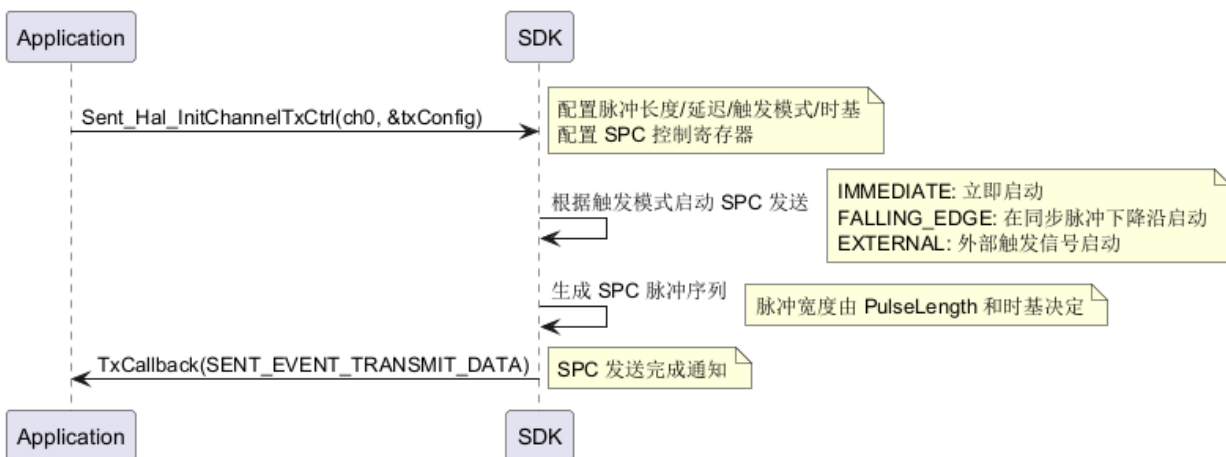
2. SENT 慢通道串行消息接收

SENT 慢通道串行消息接收流程 (Short Serial)



3. SPC (短 PWM 码) 发送流程

SENT SPC 发送流程



3.1.4.2 与标准或规范的差异

AC784xx SENT HAL 层实现遵循 SAE J2716 SENT 规范, 与标准协议无差异。

3.1.4.3 静态配置项说明

SENT 模块在编译阶段需要配置以下宏定义，位于 `device/config/Conf_AC784xx.h` 文件中：

通道编译控制

配置宏	说明	取值	适用产品
<code>CONFIG_SENT_CHANNEL0_ENABLE</code>	通道 0 编译开关	0: 禁止编译 1: 使能编译	AC7840E / AC7842 / AC7843
<code>CONFIG_SENT_CHANNEL1_ENABLE</code>	通道 1 编译开关	0: 禁止编译 1: 使能编译	AC7840E / AC7842 / AC7843
<code>CONFIG_SENT_CHANNEL2_ENABLE</code>	通道 2 编译开关	0: 禁止编译 1: 使能编译	AC7843 专用
<code>CONFIG_SENT_CHANNEL3_ENABLE</code>	通道 3 编译开关	0: 禁止编译 1: 使能编译	AC7843 专用

注意事项

- 禁用通道的状态结构体指针为 `NULL_PTR`，调用对应通道的 API 会触发断言失败
- AC7840E / AC7842 仅支持通道 0 和通道 1，通道 2 和 3 的宏必须设为 0
- AC7843 支持通道 0 ~ 3
- 根据实际使用的通道数合理配置，可有效减少 ROM 和 RAM 占用

3.1.4.4 软件集成依赖

依赖模块

SENT HAL 层功能依赖以下模块正确配置：

依赖模块	依赖说明	初始化顺序要求
CKGEN	SENT 需要时钟源才能工作 Tick 时间精度依赖时钟频率稳定性	必须先初始化后才能使用 SENT
PORT	SENT 输入引脚需配置为 SENTx_RX 功能 SPC 输出引脚需配置为 SENTx_TX 功能	必须先配置后才能正常通信
RCM	用于 SENT 模块复位	系统启动时自动初始化

3.1.4.5 软件限制/开发须知

重要限制

- 1. 时钟精度要求** - SENT 协议对 Tick 时间精度有严格要求 (通常 $\pm 20\% \sim \pm 25\%$) - SENT 模块时钟必须足够稳定, 避免频率漂移导致 `SENT_EVENT_ERROR_FREQUENCY_DRIFT` - 建议使用高精度外部晶振作为时钟源
- 2. Tick 时间配置限制** - 典型 Tick 时间: $3\mu s$ (SAE J2716 标准值) - 配置范围: 受 SENT 模块时钟和分频器精度限制 - 实际 Tick 时间 = $(\text{FractionDivider} + 1) / \text{SENT 模块时钟}$, 可能与期望值略有偏差 - 必须确保实际 Tick 时间在传感器容差范围内
- 3. 帧长度配置约束** - `RxControl.FrameLength` 定义数据半字节数量 (不含 Status 和 CRC) - 取值范围: $1 \sim 255$ - 必须与传感器配置一致, 否则触发 `SENT_EVENT_ERROR_NIBBLE_NUMBER`
- 4. CRC 模式匹配** - SENT 支持多种 CRC 模式 (4位/6位、Legacy/Enhanced、是否含 Status 等) - 配置必须与传感器 CRC 模式严格匹配, 否则所有帧都会触发 `SENT_EVENT_ERROR_NIBBLE_CRC` - 常见配置:
 - SAE J2716 标准: 4 位 CRC, Status 不参与 CRC
 - 增强 CRC: 6 位 CRC, Status 参与 CRC
- 5. 串行消息接收限制** - 短串行消息: 需接收 16 帧快通道消息才能完成一条串行消息 - 增强串行消息: 需接收 18 帧快通道消息 - 串行消息接收时间 = 快通道帧周期 $\times 16$ (或 18) - 示例: 快通道帧周期 1ms, 短串行消息接收时间约 16ms

3.1.5 SPI

本文档为 AC784xx 系列 SDK 软件包中 **SPI (串行外围接口) 模块** 的客户开发指南。面向使用 AC784xx 进行产品开发的工程技术人员, 介绍 SPI 模块的功能特性、配置方法和应用指南。

本文档适用于 AC784xx 系列全产品 (AC7840、AC7840E、AC7842、AC7843)。SPI HAL 模块支持轮询、中断和 DMA 三种传输模式。

一致性说明: 各型号 SPI HAL API 接口与逻辑完全一致, 差异通过 `Features.h` 宏自动适配, 客户应用代码无需改动。

3.1.5.1 支持的功能

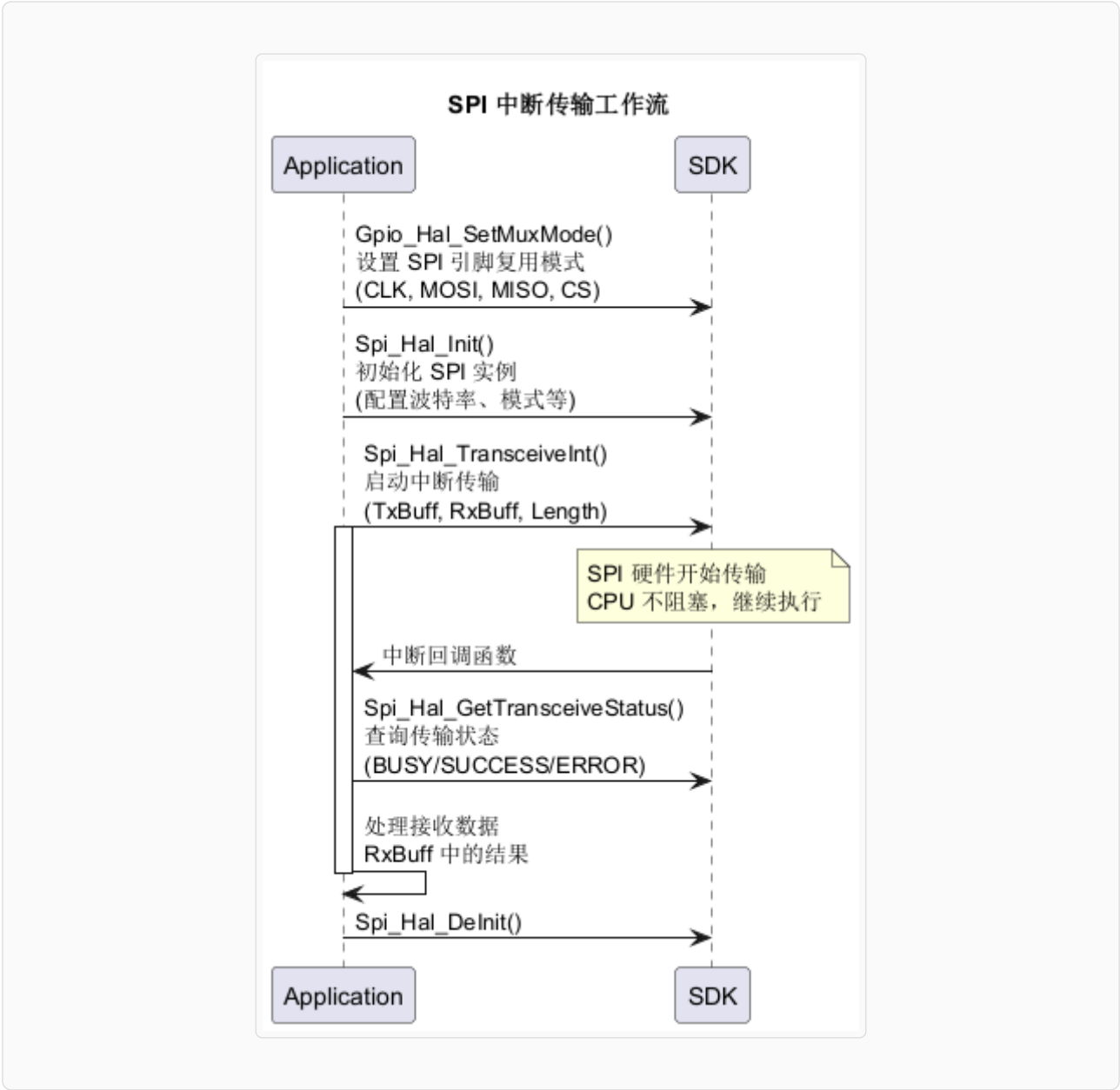
SPI 模块是一个全双工同步串行接口, 支持主从模式和三种传输机制, 每个时钟单位可传输 1/2/4-bit 数据, 主要特性如下:

工作模式与传输机制

模式	API 函数	功能说明
轮询传输 (Poll)	<code>Spi_Hal_TransceivePoll()</code>	阻塞式传输，直到数据发送接收完成或超时
中断传输 (Interrupt)	<code>Spi_Hal_TransceiveInt()</code>	中断驱动传输，通过回调函数通知完成
DMA 传输	<code>Spi_Hal_TransceiveDma()</code>	高速数据流传输，由 DMA 自动搬运

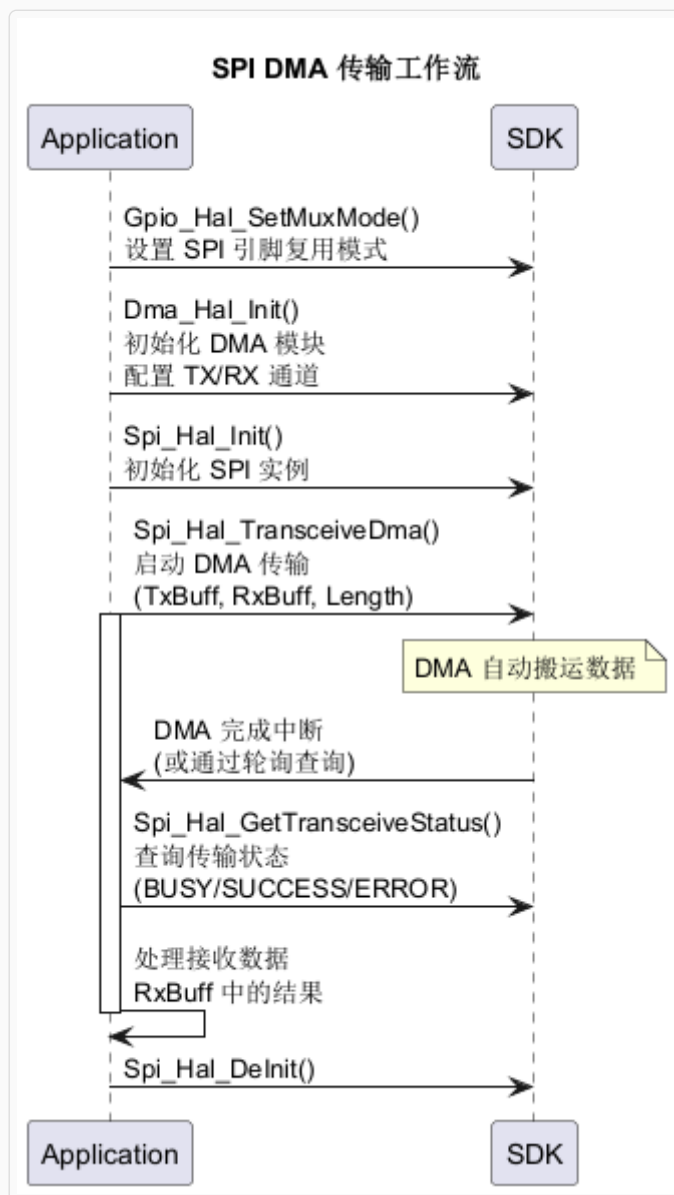
SPI 中断传输 workflow

下图展示使用中断模式进行 SPI 通信的典型工作流程：



SPI DMA 传输 workflow

下图展示使用 DMA 模式进行 SPI 通信的典型 workflow:



3.1.5.2 与标准或规范的差异

无差异。 SPI HAL 模块的基本功能（全双工同步通信、主从模式、CPOL/CPHA 模式配置）与 SPI 标准规范完全一致。HAL 层接口直接对应硬件寄存器操作，客户可根据硬件设备规格灵活选择合适的传输模式。

3.1.5.3 静态配置项说明

SPI 模块的 HAL 配置通过结构体 `Spi_HalConfigType` 进行，定义在 `Spi_Hal_Types.h` 中。SPI 实例可通过 `Conf_AC784xx.h` 中的 `CONFIG_SPIx_ENABLE` 宏使能或禁用。

3.1.5.4 软件集成依赖

依赖模块

功能	依赖模块	说明
基础初始化	MCU (CKGEN)	SPI 时钟源来自 CKGEN，需先初始化时钟树
引脚配置	GPIO	SPI 引脚 (CLK、MOSI、MISO、CS) 需配置为 SPI 复用模式
DMA 传输	DMA	使用 DMA 模式需先初始化 DMA 模块并分配专用 TX/RX 通道

初始化前置条件

1. MCU 时钟需先初始化（设置 SPI 时钟源和分频）
2. 目标 SPI 引脚需通过 GPIO 配置为 SPI 功能复用
3. 若使用硬件 CS，CS 引脚需配置为 SPI 功能；若使用 GPIO 片选，需配置为 GPIO 输出
4. 若使用 DMA 模式，需先初始化 DMA 模块和分配 TX/RX 通道

3.1.5.5 软件限制/开发须知

关键限制

1. **数据宽度限制** - 帧长 (FrmSize) 必须在 4~32 位范围内 - 超出范围会导致硬件行为未定义 - 若需要非标准宽度，可通过软件组装多帧实现
2. **从机接收** - 从机模式下，`Spi_Hal_SlaveGetReceiveLen()` 获取已接收数据长度 - `Spi_Hal_SlaveReceiveNolimitLen()` 支持动态接收，但缓冲不能超过 `MaxreceiveByteCount` - 从机必须提前分配接收缓冲，否则溢出
3. **CPOL/CPHA 时序** - 必须与通信对端（从设备或主设备）的时序配置匹配 - 不匹配会导致数据错误或无响应 - 常见配置：Mode 0 (CPOL=0, CPHA=0)、Mode 3 (CPOL=1, CPHA=1)

SPI 与系统集成规范

- 1. 初始化顺序** - 先初始化 MCU 时钟 (CKGEN) 设置 SPI 时钟频率 - 配置相关 GPIO 引脚为 SPI 功能复用 - 调用 `Spi_Hal_Init()` 初始化 SPI 实例 - 若使用 DMA, 需先初始化 DMA 并分配通道 - 若使用中断, 需配置中断优先级
- 2. 断电与低功耗** - 进入低功耗模式前调用 `Spi_Hal_DeInit()` 关闭 SPI 时钟 - 确保无进行中的传输, 已调用 `Spi_Hal_AbortTransceive()` 中止所有操作 - 唤醒后重新调用 `Spi_Hal_Init()` 恢复 SPI
- 3. 中断优先级** - SPI 中断优先级应低于系统关键中断 (看门狗、系统异常) - 建议优先级为中等 (不宜过高以避免实时性干扰, 不宜过低以保证及时响应) - 在系统中断向量表中明确分配, 避免与其他外设冲突
- 4. DMA 集成 (如使用)** - 先初始化 DMA 模块, 分配专用 TX 和 RX 通道 - 在 `Spi_HalConfigType` 中填写 `TxDmaChannel` 和 `RxDmaChannel` - DMA 传输完成后触发 SPI 完成中断或回调, 应用在回调中查询结果

3.1.6 CAN

CAN (Controller Area Network, 控制器局域网) 模块是AC784xx系列MCU的核心通信外设之一, 提供标准的CAN协议通信功能, 支持CAN 2.0和CAN FD两种工作模式, 广泛应用于车身网络 (如动力总成CAN、车身CAN) 和实时通信系统中。

产品型号差异概览:

特性	AC7840	AC7840E	AC7842	AC7843
CAN 控制器实例数	4	2	6	6
CAN FD 支持	支持	支持	CAN0~CAN3 支持	支持
接收过滤器个数	60	60	CAN0~CAN3: 60, CAN4~CAN5:31	128(标准)+64(扩展)
发送缓冲区个数(深度)	1 个主发送缓冲区 (深度 1) + 1 个次发送缓冲区 (深度 6)	1 个主发送缓冲区 (深度 1) + 1 个次发送缓冲区 (深度 6)	1 个主发送缓冲区 (深度 1) + 1 个次发送缓冲区 (CAN0~CAN3: 6, CAN4~CAN5:20)	1发送FIFO (32)
接收缓冲区个数(深度)	1接收FIFO (13)	1接收FIFO (13)	1接收FIFO (CAN0~CAN3: 13, CAN4~CAN5:21)	1 个专用接收缓冲区 (64) + 2 个接收 FIFO (各 64)

3.1.6.1 支持的功能

AC784xx CAN HAL 层提供以下功能：

API 名称	功能说明	适用产品
<code>Can_Hal_Init</code>	初始化 CAN 控制器及基础配置，包括波特率、工作模式、邮箱分配、中断配置	全系列
<code>Can_Hal_Deinit</code>	反初始化 CAN 控制器，释放资源并恢复默认状态	全系列
<code>Can_Hal_SetControllerState</code>	设置 CAN 控制器状态（停止、运行、待机）	全系列
<code>Can_Hal_GetControllerState</code>	获取 CAN 控制器当前工作状态	全系列
<code>Can_Hal_WriteTxBuffer</code>	发送 CAN 报文，写入邮箱并触发发送	全系列
<code>Can_Hal_WriteTxBufferMsgSetOnly</code>	发送 CAN 报文，仅写入邮箱，不触发发送	AC7840X/AC7842X/AC7840E
<code>Can_Hal_ReadRxBuffer</code>	从接收邮箱或 FIFO 读取已接收的 CAN 报文	全系列
<code>Can_Hal_GetTxStatus</code>	查询发送邮箱状态（已发送/未发送） [AC7843X 使用 HwBufIdx 标识]	全系列
<code>Can_Hal_GetRxStatus</code>	查询接收邮箱或 FIFO 状态（有接收/空）	全系列
<code>Can_Hal_GetTxErrorCount</code>	获取发送错误计数	全系列
<code>Can_Hal_GetRxErrorCount</code>	获取接收错误计数	全系列
<code>Can_Hal_GetErrorState</code>	获取控制器当前错误状态（主动/被动/离线）	全系列
<code>Can_Hal_GetErrorsInfo</code>	获取详细的错误掩码信息（位错/格式错/填充错等）	全系列
<code>Can_Hal_ConfigExtendMode</code>	配置控制器扩展模式（正常/监听/内部环回/外部环回）	全系列
<code>Can_Hal_ConfigTimeStamp</code>	配置时间戳功能，用于报文发送或接收时间记录	全系列
<code>Can_Hal_GetTxTimeStamp</code>	获取发送邮箱的时间戳值 [AC7840X/AC7842X/AC7840E 专属]	AC7840X/AC7842X/AC7840E
<code>Can_Hal_GetBase</code>	获取 CAN 控制器硬件基地址，用于配合寄存器层接口使用	全系列
<code>Can_Hal_AbortTransmit</code>	中止正在进行的发送操作	全系列

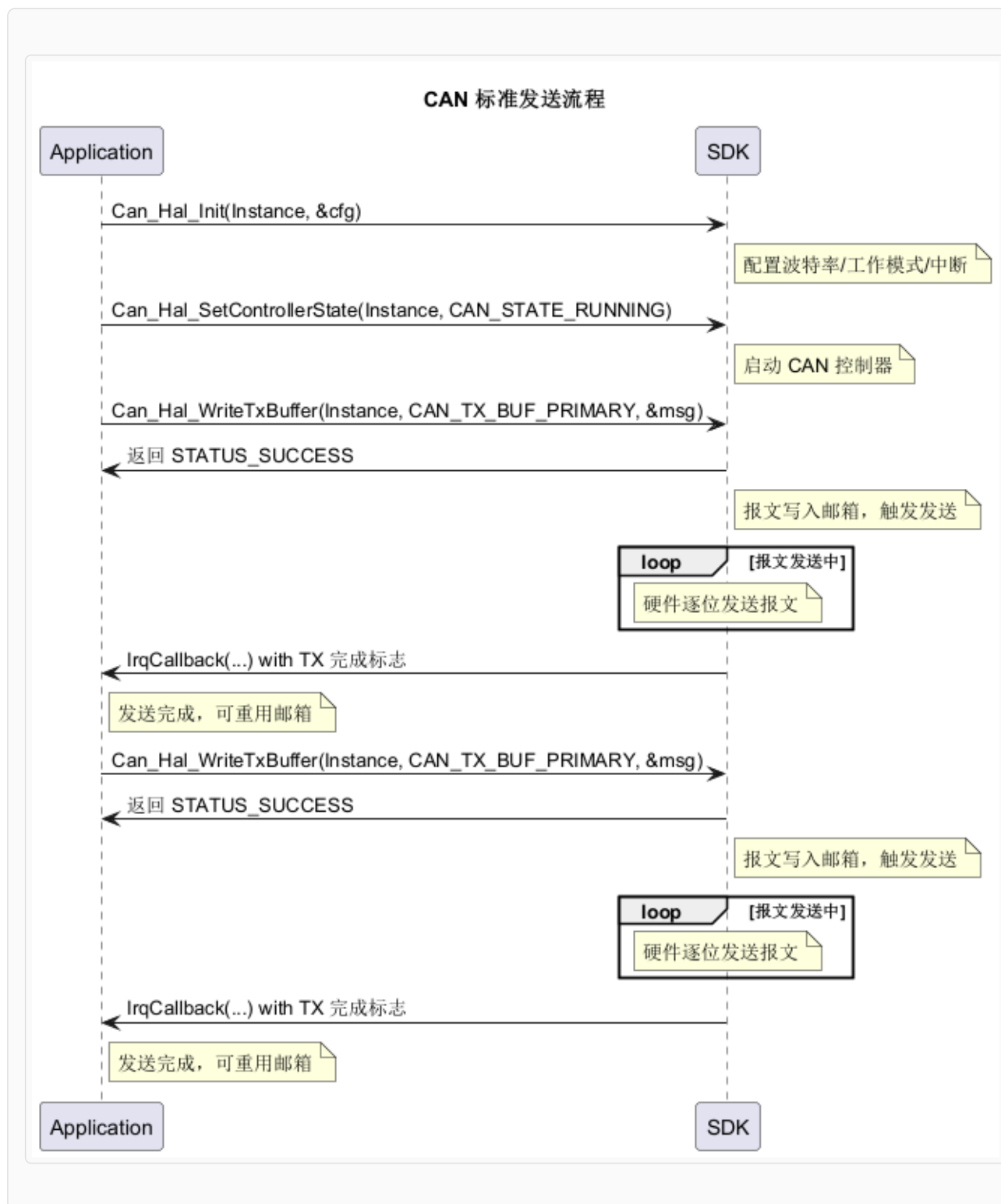
API 名称	功能说明	适用产品
Can_Hal_ReadTxEvent	读取 TX Event FIFO 元素	AC7843
Can_Hal_SetMsgInfo	写入邮箱报文信息，无任何状态检查	全系列
Can_Hal_StartTransmit	启动发送动作	AC7840X/AC7842X/AC7840E
Can_Hal_GetMsgInfo	从接收缓冲提取报文信息	全系列
Can_Hal_StartNextDma	启动下一次 DMA 传输（用于 FIFO DMA 模式）	全系列
Can_Hal_SetTxSecAmount	配置次发缓冲区发送模式	AC7840X/AC7842X/AC7840E

基础通信功能

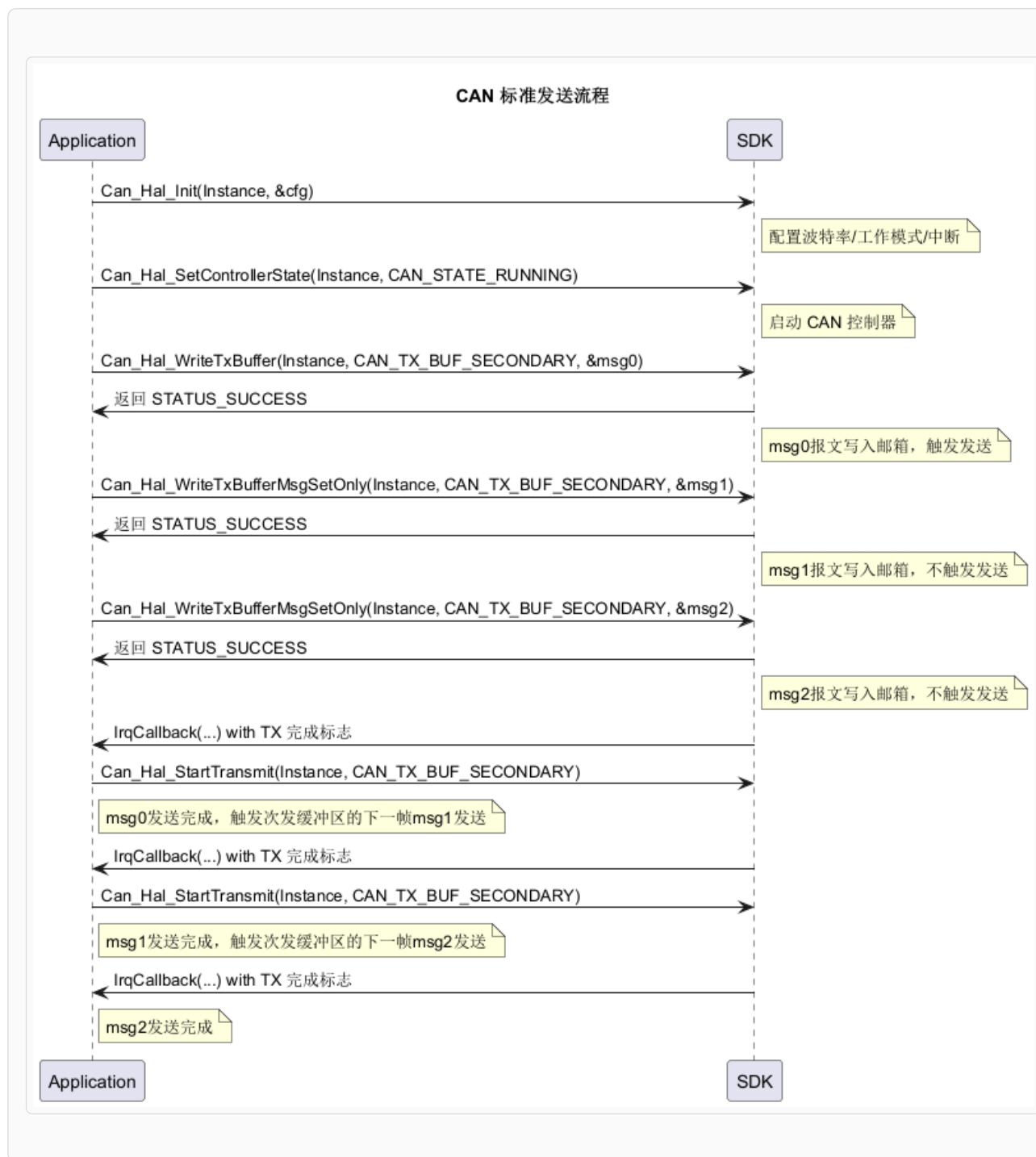
- 初始化与反初始化
 - 配置波特率（仲裁域和数据域）、工作模式（CAN 2.0/FD）
 - 分配邮箱和接收 FIFO 资源
 - 注册中断回调函数
 - 过滤器配置
 - 其它基础控制器参数配置
- 报文收发
 - 发送 CAN 报文（标准帧/扩展帧，CAN 2.0/FD）
 - 接收 CAN 报文（邮箱/FIFO 模式）
 - 查询发送/接收状态
 - 中止正在进行的发送操作
- 错误处理与诊断
 - 获取控制器错误状态（主动/被动/离线）
 - 统计收发错误计数
 - 获取详细的错误掩码信息
 - 支持时间戳记录用于故障追溯

典型通信流程时序图

1. 主发送缓冲区发送流程（AC7840X/AC7842X/AC7840E）



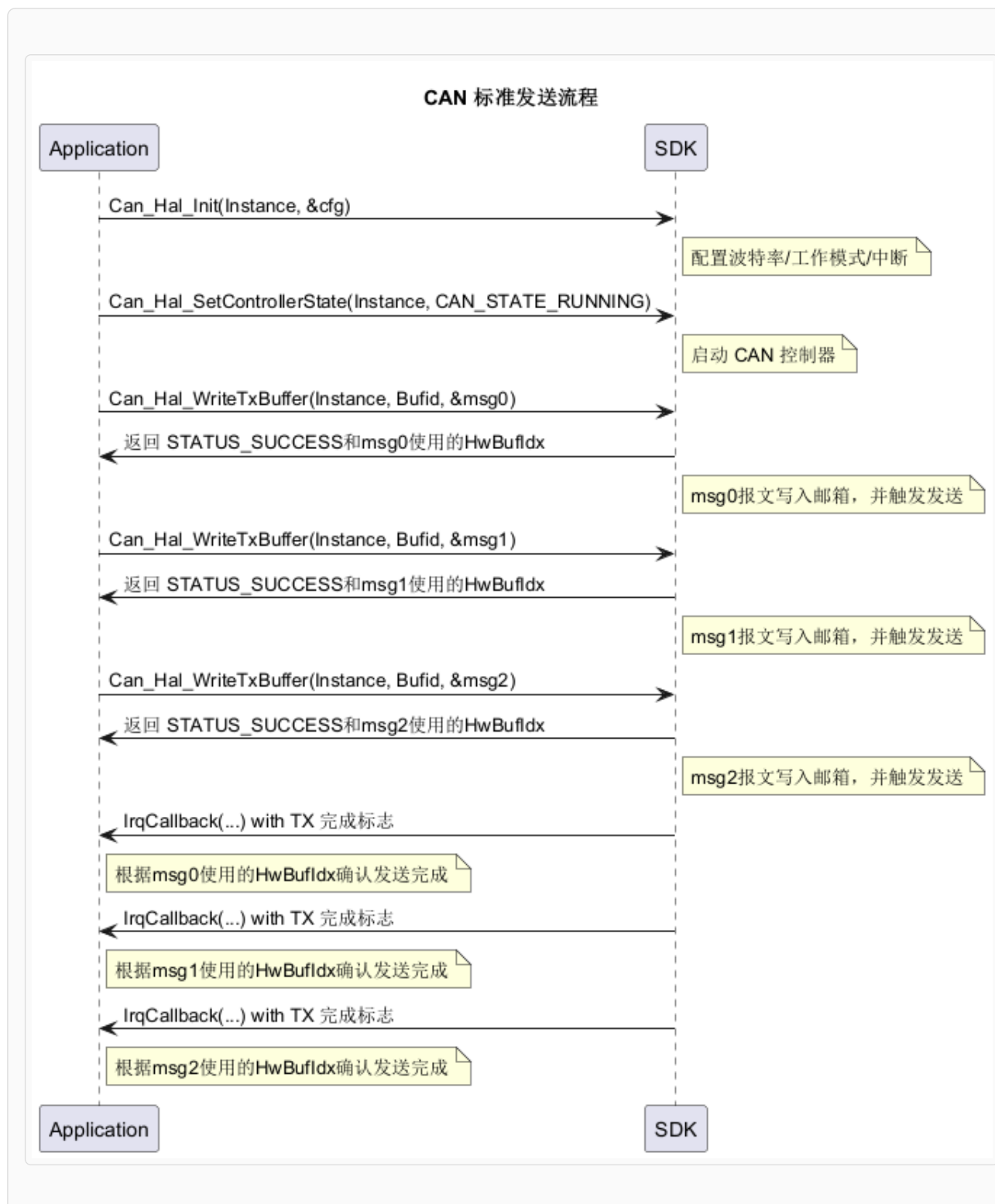
2. 次发送缓冲区_需要识别每一帧发送完成的状态(TSONE模式)_发送流程 (AC7840X/AC7842X/AC7840E)



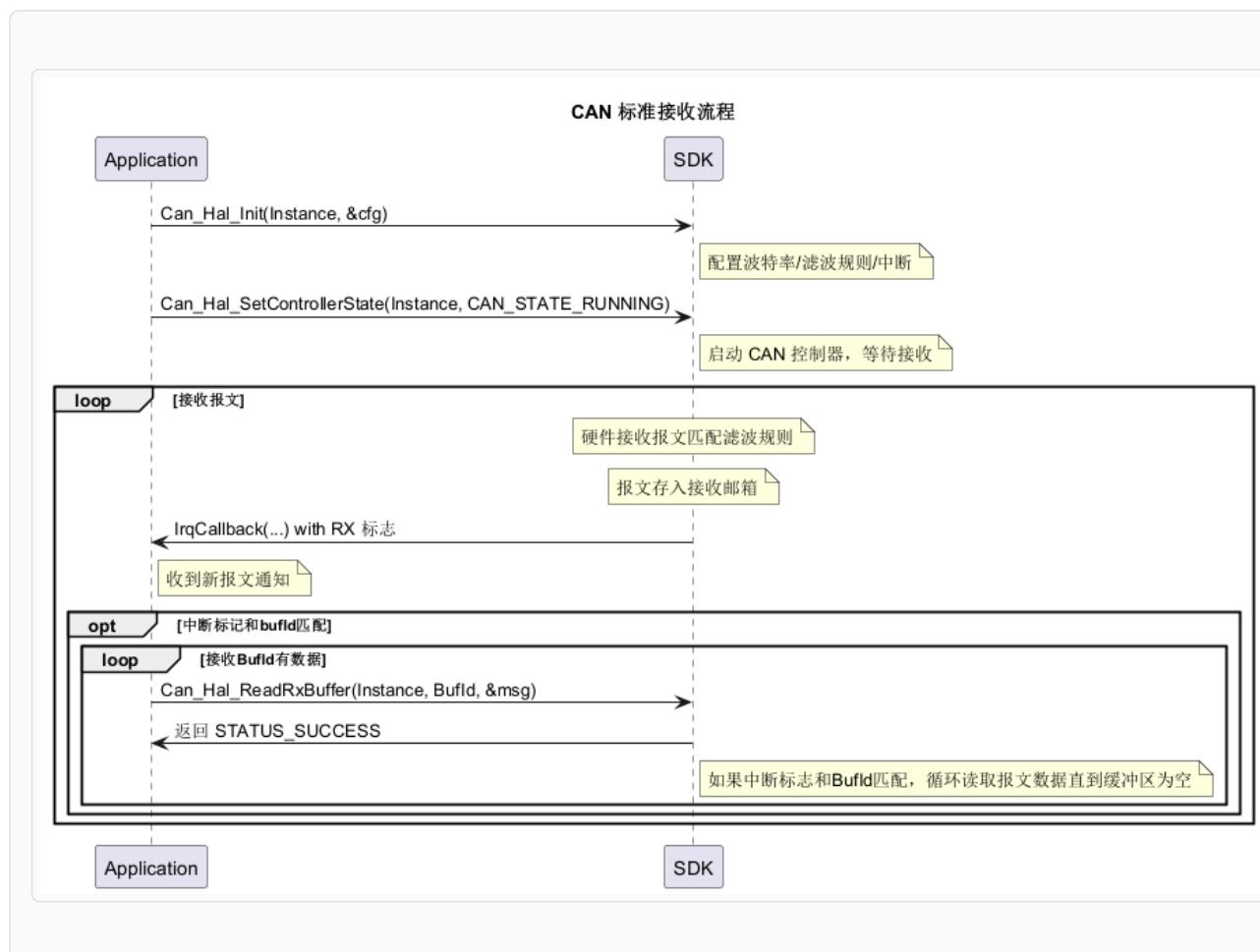
3. 次发送缓冲区_不需要识别每一帧的发送完成状态 (TSALL模式)_发送流程 (AC7840X/AC7842X/AC7840E)



4. 发送流程 (AC7843)



5. 接收流程



3.1.6.2 与标准或规范的差异

AC784xx CAN HAL 层实现遵循标准 CAN 协议，与通用 CAN 规范无差异。

3.1.6.3 静态配置项说明

CAN HAL 层通过编译时宏定义进行静态配置，这些宏定义在 `device/config/Conf_AC784xx.h` 文件中设置

通道使能配置

宏定义： `CONFIG_CANx_ENABLED` (x = 0~5)

功能说明： 控制各 CAN 通道是否参与编译，未使能的通道不会占用 RAM 和 ROM 资源。

配置值： - `1`：使能该通道，生成对应的状态结构体和驱动代码 - `0`：禁止该通道，节省内存资源

注意事项：

- 禁用通道的状态结构体指针为 `NULL_PTR`，调用对应通道的 API 会触发断言失败

- 根据实际使用的通道数合理配置，可有效减少 RAM 占用

3.1.6.4 软件集成依赖

1. 初始化依赖顺序

CAN 模块初始化前，需确保以下依赖模块已就绪：

依赖模块	依赖说明	初始化顺序要求
CKGEN	CAN 需要时钟源才能工作 波特率计算依赖外设时钟频率	必须先初始化后才能使用 CAN
PORT	CAN TX/RX 引脚需配置为 CAN功能	必须先配置后才能正常通信
DMA	使用 DMA 传输方式时需要	如使用 DMA，需先初始化 DMA 模块

3.1.6.5 软件限制/开发须知

重要限制

1. 消息RAM配置约束（7843）

- 接收FIFO0、接收FIFO1、发送FIFO、发送专用缓冲区、发送事件FIFO、扩展过滤器、标准过滤器共享消息RAM的存储空间
- CAN0~CAN3 消息 RAM（MRAM0）共 8K 字节，CAN4~CAN5 消息RAM（MRAM1）共 4K 字节
- 可在初始化接口参数中对上述配置项使用的MRAM空间合理分配，不能超过MRAM总限制，否则会触发断言失败

3.1.7 I2C

I2C（Inter-Integrated Circuit）模块是AC784xx系列MCU的核心通信外设之一，提供标准的I2C协议通信功能，支持主机和从机两种工作模式，广泛应用于传感器、EEPROM上位寄存器算法、RTC等外设通信场景。

3.1.7.1 产品型号差异概览

参考对应型号的技术文档，查看对应型号的产品特性。

3.1.7.2 支持的功能

I2C 模块提供的主要功能接口

I2C 模块提供以下主要功能接口：

API 名称	功能说明	适用产品
I2c_Hal_Init	初始化 I2C 控制器及基础配置，包括波特率、工作模式（主/从）、地址匹配、中断配置	全系列
I2c_Hal_DeInit	反初始化 I2C 控制器，释放资源并恢复默认状态	全系列
I2c_Hal_MasterGetBaudRate	获取 I2C 主机当前配置的波特率	全系列
I2c_Hal_MasterSetBaudRate	设置 I2C 主机波特率，支持运行时动态调整	全系列
I2c_Hal_SyncTransceive	主机执行同步的收发操作（阻塞式）	全系列
I2c_Hal_AsyncTransceive	主机执行异步的收发操作（非阻塞式）	全系列
I2c_Hal_AbortTransceive	中止正在进行的收发操作	全系列
I2c_Hal_GetStatus	获取 I2C 通道当前状态（空闲、忙碌、错误等）	全系列
I2c_Hal_GetBase	获取 I2C 控制器硬件基地址，用于高级应用	全系列
I2C_Hal_SlaveSetTxBuffer	从机模式：配置发送缓冲区及数据长度	全系列
I2C_Hal_SlaveSetRxBuffer	从机模式：配置接收缓冲区及缓冲长度	全系列
I2C_Hal_SlaveGetRxSize	从机模式：获取实际接收的数据长度	全系列

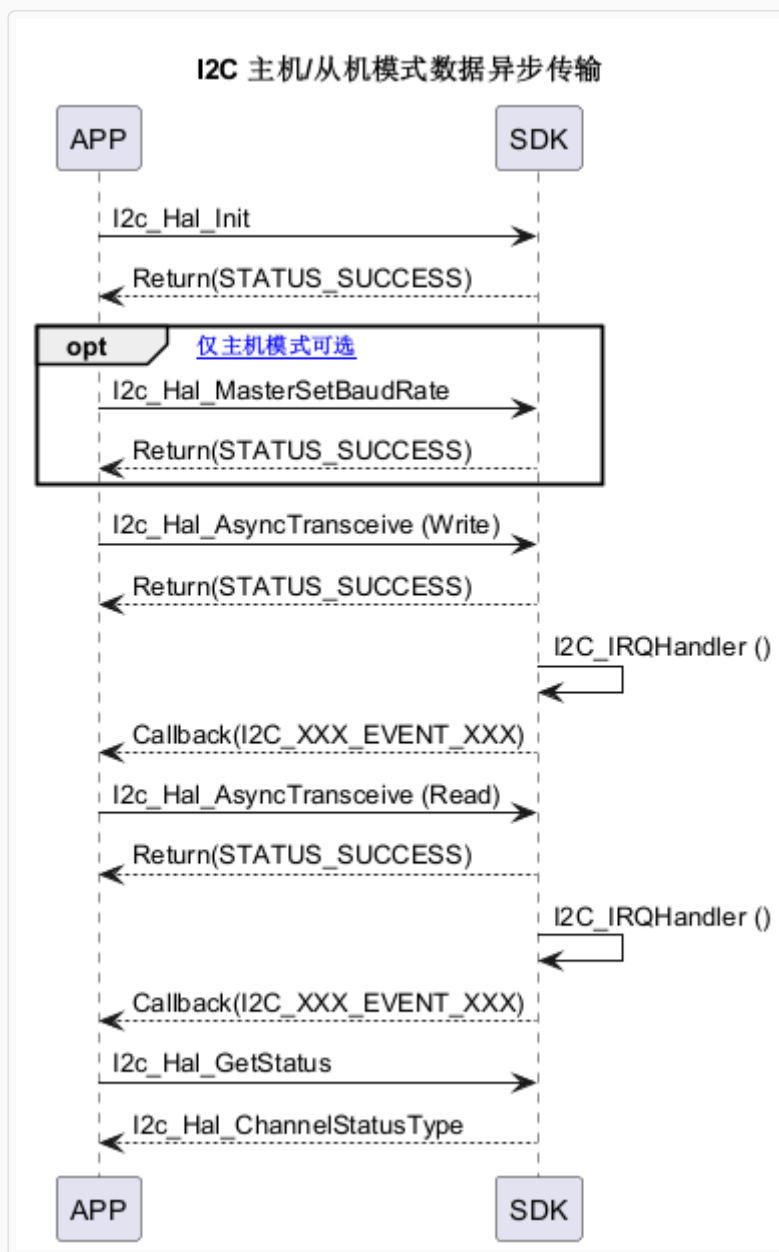
基础通信功能

- 初始化与反初始化
 - 配置波特率、工作模式（主/从）
 - 配置地址匹配和中断回调
 - DMA 通道配置（可选）
 - 其它基础控制器参数配置
- 报文收发
 - 主机接收和发送数据
 - 从机接收和发送数据

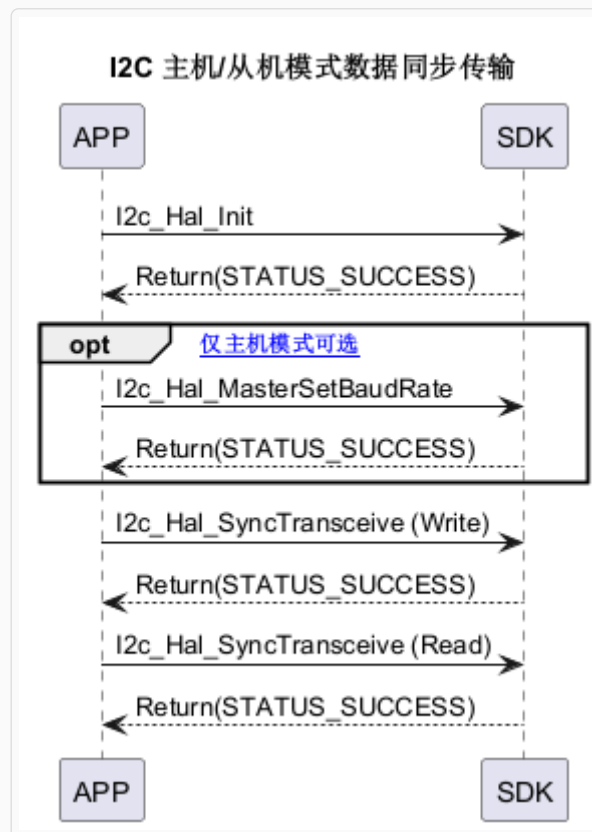
- 查询传输状态
- 中止正在进行的传输操作
- **通道状态获取**
 - 获取通道当前状态（空闲、忙碌、完成、错误等）

典型通信流程时序图

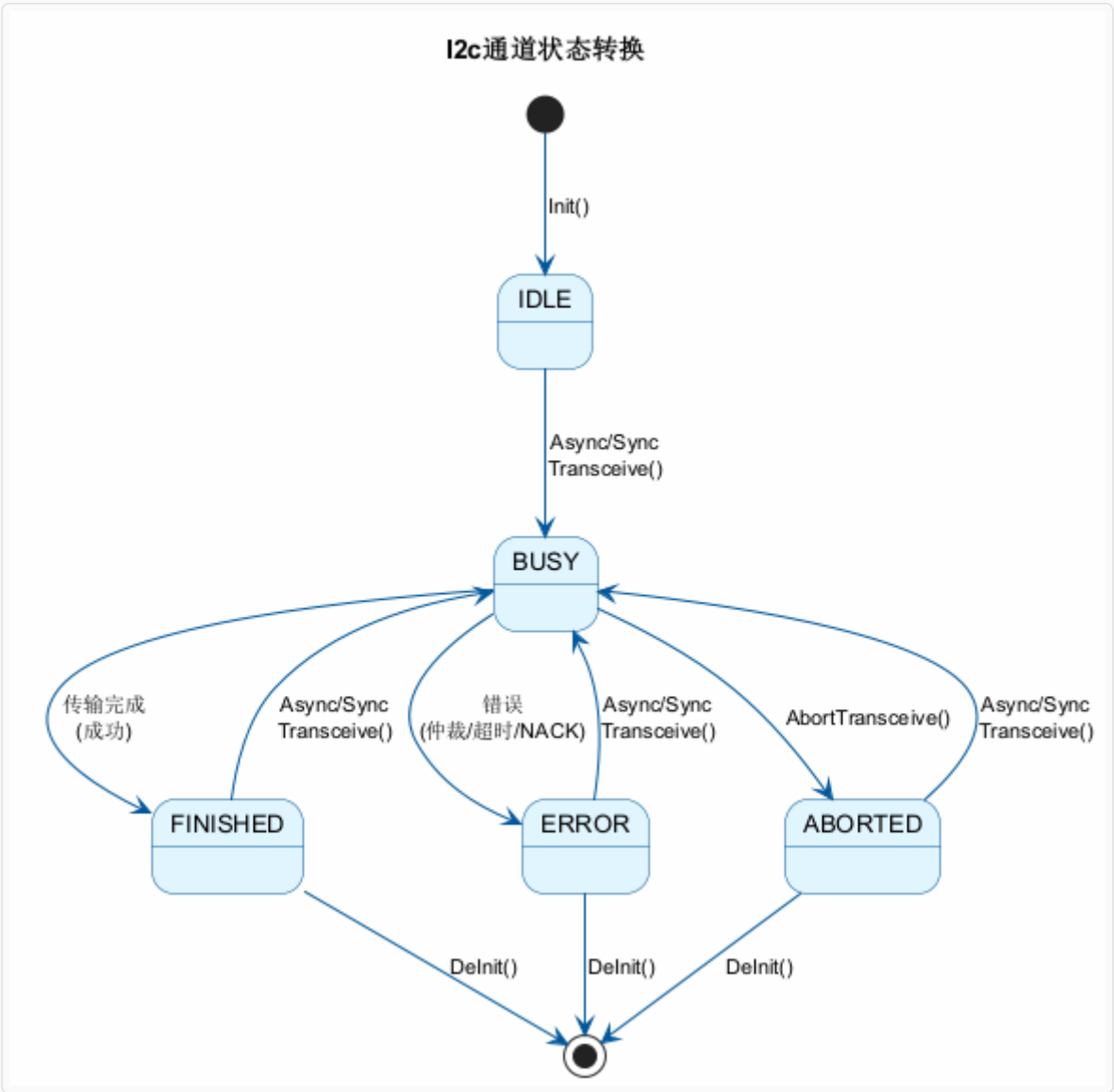
1. 主机/从机模式数据异步传输



2. 主机/从机模式数据同步传输



3. I2c 通道状态转换图



状态定义表:

状态	枚举值	说明
IDLE	I2C_CHANNEL_IDLE	通道初始化完成，空闲可用
BUSY	I2C_CHANNEL_BUSY_TRANSMIT	数据传输进行中（发送或接收）
FINISHED	I2C_CHANNEL_FINISHED	传输成功完成
ERROR	I2C_CHANNEL_ERROR_PRESENT	传输错误（仲裁丢失/超时/NACK/缓冲溢出）
ABORTED	I2C_CHANNEL_ABORTED_SUCCESS	传输被主动中止

内部状态映射:

• BUSY_TRANSMIT 映射:

- Master: I2C_MASTER_CHANNEL_BUSY_SEND / I2C_MASTER_CHANNEL_BUSY_RECEIVE
- Slave: I2C_SLAVE_CHANNEL_BUSY_SEND / I2C_SLAVE_CHANNEL_BUSY_RECEIVE / I2C_SLAVE_CHANNEL_BUSY_TRANSMIT

• ERROR_PRESENT 映射:

- Master: I2C_MASTER_CHANNEL_DMA_ERROR / I2C_MASTER_CHANNEL_ARBITRATION_LOST / I2C_MASTER_CHANNEL_TIMEOUT / I2C_MASTER_CHANNEL_RECEIVE_NACK
- Slave: I2C_SLAVE_CHANNEL_DMA_ERROR / I2C_SLAVE_CHANNEL_TX_EMPTY / I2C_SLAVE_CHANNEL_TX_UNDERRUN / I2C_SLAVE_CHANNEL_RX_OVERRUN

转换关键规则:

1. Init() 后进入 IDLE 状态
2. 调用 AsyncTransceive/SyncTransceive 进入 BUSY 状态
3. FINISHED/ERROR/ABORTED 都可直接调用 AsyncTransceive/SyncTransceive 重新启动 (无需 Delnit)
4. 任何状态调用 Delnit() 释放资源

3.1.7.3 与标准或规范的差异

I2C 标准对标:

SDK 接口完全遵循 I2C 总线标准, 核心通信协议一致

3.1.7.4 静态配置项说明

I2C HAL 层通过编译时宏定义进行静态配置, 这些宏定义在 device/config/Conf_AC784xx.h 文件中设置

通道使能配置

宏定义: CONFIG_I2Cx_ENABLED (x = 0~2)

功能说明: 控制各 I2C 通道是否参与编译, 未使能的通道不会占用 RAM 和 ROM 资源。

配置值: - 1: 使能该通道, 生成对应的状态结构体和驱动代码 - 0: 禁止该通道, 节省内存资源

注意事项: - 禁用通道的状态结构体指针为 NULL_PTR, 调用对应通道的 API 会触发断言失败 - 根据实际使用的通道数合理配置, 可有效减少 RAM 占用

3.1.7.5 软件集成依赖

初始化依赖顺序

I2C 模块初始化前，需确保以下依赖模块已就绪：

依赖模块	依赖说明	初始化顺序要求
CKGEN (时钟生成)	I2C 需要时钟源才能工作 波特率计算依赖外设时钟频率	必须先初始化后才能使用 I2C
PORT (引脚复用)	I2C 引脚需配置为 I2C 功能	必须先配置后才能正常通信
DMA	使用 DMA 传输方式时需要	如使用 DMA，需先初始化 DMA 模块

3.1.7.6 软件限制/开发须知

传输模式的选择

• 同步模式 (`I2c_Hal_SyncTransceive`):

- 阻塞式调用，等待传输完成才返回；
- 简单易用，适合初始化、配置等低频操作；
- 不能在中断上下文调用（会导致死锁）；
- 超时时间由 `I2C_HW_DEADLINE_TIMEOUT` 宏定义指定。

• 异步模式 (`I2c_Hal_AsyncTransceive`):

- 非阻塞式调用，立即返回；
- 传输完成或出错时通过回调函数通知；
- 支持在任何上下文调用（包括中断）；
- 适合高频数据传输、实时性要求高的场景；
- 需要合理处理回调函数中的事件。

3.1.8 EIO-I2C

待后续版本支持

3.2 IO

3.2.1 PWM

本文档为 AC784xx 系列 SDK 软件包中 **PWM（脉宽调制）模块** 的客户开发指南。面向使用 AC784xx 进行产品开发的工程技术人员，介绍 PWM 模块的功能特性、配置方法和应用指南。

本文档适用于 AC784xx 系列全产品（AC7840、AC7840E、AC7842、AC7843）。

一致性说明：各型号 PWM HAL API 接口与逻辑完全一致，差异通过 `Features.h` 宏自动适配，客户应用代码无需改动。

3.2.1.1 支持的功能

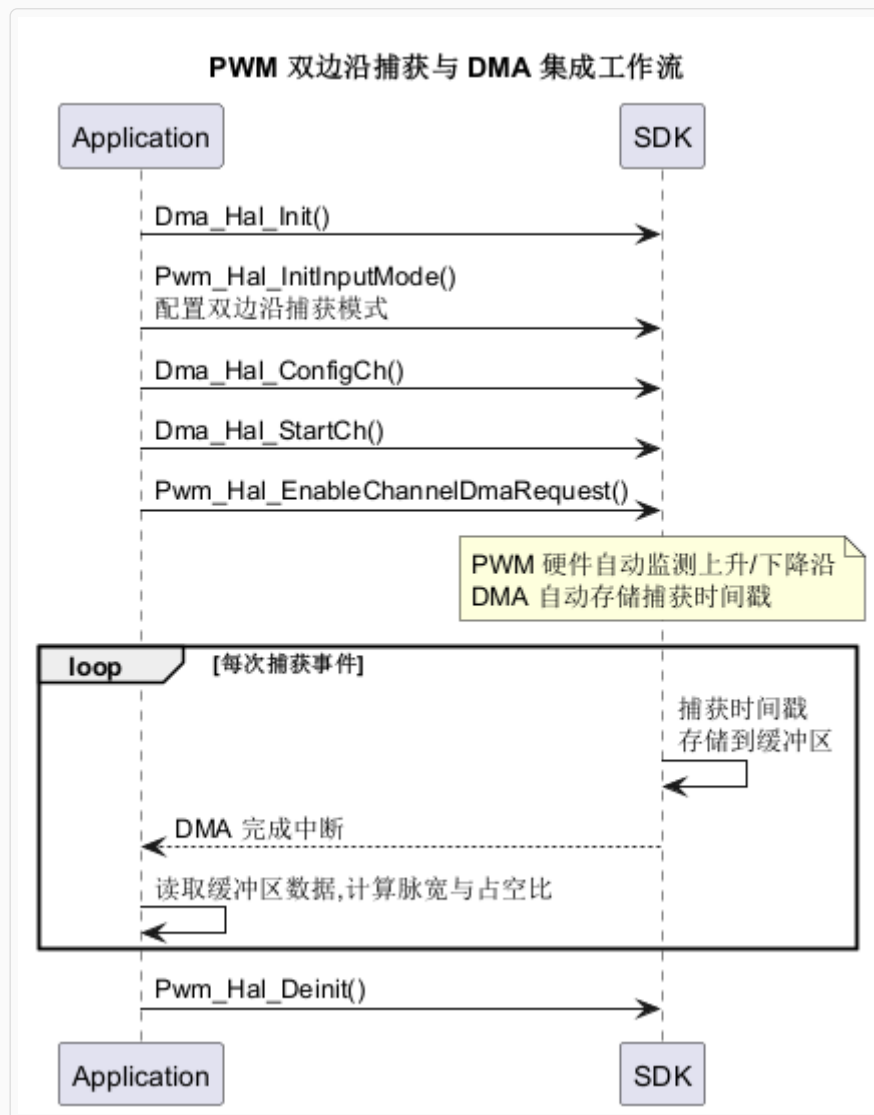
PWM 模块是一个多功能的 16 位定时器，支持以下工作模式和功能。

核心工作模式

模式	功能说明	应用场景
边沿对齐 PWM (EPWM)	生成边沿对齐的 PWM 信号，支持可编程频率与占空比	标准 PWM 生成、LED 调光、风扇控制
中心对齐 PWM (CPWM)	生成中心对齐的 PWM 信号，对称输出，用于电机控制	无刷直流电机、步进电机驱动
组合/互补模式	两个相邻通道配合生成互补或对称 PWM，支持死区插入	推挽输出驱动、电源开关拓扑
输入捕获模式	捕获脉冲宽度或周期，支持上升沿/下降沿/双边沿触发	脉冲频率测量、占空比检测
双边沿捕获	同一通道上升沿与下降沿分别捕获，用于精确脉宽测量	高精度脉冲宽度测量
正交解码模式	ABZ 编码解码，用于旋转传感器和电机速度测量	旋转编码器解码、速度反馈

DMA 双边沿捕获集成

下图展示使用 DMA 实现 PWM 双边沿捕获的典型工作流程：



3.2.1.2 与标准或规范的差异

无差异。 PWM 模块的基本功能（PWM 生成、输入捕获、正交解码）与硬件规格书及行业通用设计一致。HAL 层接口直接对应硬件寄存器操作，客户可根据硬件规格灵活配置。

3.2.1.3 静态配置项说明

PWM 模块的配置根据工作模式分为以下类别，均定义于 `Pwm_Hal_Types.h` 中：

输出模式配置

使用结构体 `Pwm_Hal_OutputCfg` 配置全局参数（时钟源、预分频、计数模式、计数初值与终值）。通过 `Pwm_Hal_OutputChnConfig` 数组为各通道设置工作模式、输出电平、比较值和中断使能。

输入捕获配置

使用结构体 `Pwm_Hal_InputCfg` 配置时钟和计数参数。通过 `Pwm_Hal_InputChnConfig` 数组为各通道设置捕获边沿、输入滤波器、捕获中断和计数器复位等参数。

正交解码配置

使用结构体 `Pwm_Hal_QuadDecoderCfg` 配置编码器类型、各相输入极性、输入滤波和计数参数。

关键配置约束

- 计数初值与终值需满足：`MaxCount > MinCount`
- 正交解码固定引脚分配：A 相→CH0，B 相→CH1，Z 相→CH2（其他通道禁用）
- HALL模式固定引脚分配：HALL_A→CH0，HALL_B→CH1，HALL_C→CH2（其他通道禁用）
- 输入滤波支持范围：仅 CH0-CH3 支持硬件滤波

3.2.1.4 软件集成依赖

模块依赖关系

功能	依赖模块	说明
基础初始化	MCU (CKGEN)	PWM 时钟源来自 CKGEN，需先初始化时钟树
引脚输出/输入	GPIO	PWM 输出/输入引脚通过 GPIO 复用连接，需配置复用模式
高速数据传输	DMA	使用 DMA 传输 PWM 捕获数据或在 PWM 事件触发 DMA 操作
多通道协同	CTU (可选)	多个 PWM 通道同步或由 ADC 采样结果触发 PWM 时使用

初始化前置条件

1. 系统时钟需先初始化（MCU/CKGEN 模块）
2. 若使用外部时钟源，需确保外部时钟频率不超过系统时钟的 1/4
3. 若需 GPIO 复用，需先确认目标引脚的复用配置
4. 若使用 DMA，需先初始化 DMA 模块并分配专用通道
5. 若使用中断驱动，需在 NVIC 配置中断优先级

3.2.1.5 软件限制/开发须知

关键限制

1. **死区插入的使用前提** - 死区功能仅在**组合模式且互补功能启用**时有效
2. **写保护功能** - 启用写保护后（ `FDSR[WPEN] = 1` ），关键寄存器位无法修改 - 需通过 `FUNCSEL[WPDIS] = 1` 临时关闭写保护才能改动配置 - 适用于电机驱动等对误配置敏感的应用

常见陷阱与应对

问题现象	原因	解决方案
捕获值错误	计数器溢出未处理	处理溢出中断
死区失效	未启用互补模式或死区值为 0	确保组合模式 + 互补使能 + 死区值 > 0

PWM 与系统集成规范

1. **初始化顺序** - 先初始化 MCU/时钟模块设置系统时钟频率 - 配置 GPIO 复用模式 - 再初始化 PWM 模块（ `Pwm_Hal_InitOutputMode()` 等） - 使用前验证硬件输出
2. **断电与低功耗** - 进入低功耗模式前调用 `Pwm_Hal_DeInit(instance)` 关闭 PWM 时钟 - 确保所有输出已安全释放 - 唤醒后重新初始化
3. **中断优先级** - PWM 中断优先级应低于实时关键中断（如看门狗、系统异常） - 建议在系统中断向量表中明确分配 - 避免与高优先级外设中断冲突
4. **DMA 集成（如使用）** - 先初始化 DMA 模块与 PWM 模块 - 通过 DMA 触发寄存器链接 PWM 事件（如捕获完成） - DMA 传输完成后触发中断通知应用

3.2.2 ADC

AC784xx 系列 MCU 中 ADC 模块提供片上模数转换功能，支持多通道输入、多种分辨率、软件/硬件触发及 DMA 传输。

3.2.2.1 支持的功能

AC784xx ADC HAL 层提供以下功能:

1. 基础转换功能

- **模块初始化与反初始化:** 通过 `Adc_Hal_Init` 完成中断回调注册; `Adc_Hal_ConfigConverter` 配置时钟、分辨率、参考电压等全局参数, 内部自动使能外设时钟
- **软件触发转换:** `Adc_Hal_SwTriggerRegularConvert` / `Adc_Hal_SwTriggerInjectConvert` 分别触发常规组/注入组转换
- **转换结果读取:** `Adc_Hal_GetSeqResult` 按序列位置读取转换结果 (12/10/8 位)

2. 输入通道

- **外部通道:** AC7840/AC7843 支持 `ADC_CH_0 ~ ADC_CH_31` (32 路), AC7842 支持 `ADC_CH_0 ~ ADC_CH_23` + 校准/内部通道
- **内部通道:** 内置 Bandgap (`ADC_CH_BANDGAP`)、温度传感器 (`ADC_CH_TSENSOR`)、电源监测 (`ADC_CH_SUPPLY`) 等, 各型号通道映射有差异
- **内部 AMUX 通道:** AC7842/AC7843/AC7840E 通过 `Adc_InternalChannelSrcType` 选择 AMUX 源, 可测量 VDD、VDDA、LDO、BG、TSENSOR 等内部信号

3. 转换分辨率与时序

- **分辨率:** 支持 12 位 (`ADC_RESOLUTION_12BIT`)、10 位 (`ADC_RESOLUTION_10BIT`)、8 位 (`ADC_RESOLUTION_8BIT`) 三档
- **采样时间:** 8 档可选 (AC7840E: 7~215 时钟周期; AC7840/AC7842/AC7843: 5~185 时钟周期)
- **时钟分频:** 1~16 分频 (`Adc_PrescaleType`), 灵活适配外设时钟频率
- **结果对齐:** 支持右对齐 (`ADC_ALIGN_RIGHT`) 和左对齐 (`ADC_ALIGN_LEFT`)

4. 转换组与序列管理

- **常规组 (Regular Group):** 最多 24 路序列 (`ADC_RSEQ_0 ~ ADC_RSEQ_23`), 支持扫描模式、连续模式和间断模式
- **注入组 (Inject Group):** 最多 4 路序列 (`ADC_ISEQ_0 ~ ADC_ISEQ_3`), 可设置自动跟随常规组触发 (`InjectAutoModeEn`)
- **间断模式:** 常规组支持分批转换 (`RegularDiscontinuousModeEn`, 可配置每批转换通道数); 注入组同样支持间断模式

5. 触发方式

- **软件触发:** `ADC_TRIGG_SRC_SW`, 直接调用 API 启动转换

- **硬件触发：** `ADC_TRIGG_SRC_HW`，通过 CTU（交叉触发单元）接收来自 Timer/PWM 的定期触发脉冲，实现无 CPU 干预的同步采样

6. DMA 传输

- **DMA 结果搬运：** `DmaEnable = TRUE` 时，转换结果由 DMA 自动搬运至用户指定目标地址（`DmaDstAddr`），无需 CPU 逐一读取
- **DMA 完成回调：**通过 `DmaCallback` 注册 DMA 传输完成通知，参数 `DmaArgs` 传递用户上下文

7. 硬件平均（抗噪）

- **多次采样平均：** `Adc_Average_ConfigType` 使能后，硬件自动对 4 / 8 / 16 / 32 次采样求平均值，输出平均结果，有效抑制噪声

8. 模拟监视器（AMO）

- **电压窗口监测：** `Adc_Hal_ConfigAmo` 设置上下阈值（`AmoUpThreshold` / `AmoLowThreshold`），转换结果超出范围时自动产生中断
- **触发模式：**支持电平触发（`ADC_AMO_TRIGGER_LEVEL`）和边沿触发（`ADC_AMO_TRIGGER_EDGE`）两种
- **监测范围：**可监测全部常规通道（`AmoRegularEn`）、全部注入通道（`AmoInjectEn`）或单一指定通道（`AmoSingleModeEn` + `AmoSingleChannel`）
- **阈值偏移：**上下阈值各支持独立偏移量（`AmoUpOffset` / `AmoLowOffset`），用于设置迟滞区间

9. Interleave 模式（除 AC7840E 外均支持）

- **双 ADC 交错采样：**AC7840/AC7842/AC7843 均支持 ADC0/ADC1 交错工作（`Adc_InterleaveType`），等效采样率翻倍，可选两组引脚路由（`ADC_INTERLEAVE_0` / `ADC_INTERLEAVE_1`）；AC7843 额外提供两个专用交织通道（`ADC_CH_INTERLEAVE38` / `ADC_CH_INTERLEAVE39`）

10. 中断与回调

- **EOC/IEOC 中断：**常规组转换结束（`ADC_EVENT_EOC`）和注入组转换结束（`ADC_EVENT_IEOC`）均可触发中断
- **AMO 告警中断：**`ADC_EVENT_AMO`（电平模式告警）、`ADC_EVENT_AAMO`（边沿模式异常）、`ADC_EVENT_NAMO`（边沿模式正常）
- **统一回调：**注册 `Adc_CallbackType` 回调，中断参数 `Adc_InterruptInfoType` 携带实例号、事件掩码和序列索引

产品差异：Interleave 模式（`ADC_INTERLEAVE_0/1`）AC7840/AC7842/AC7843 均支持，AC7840E 仅有 1 个 ADC 实例，不支持交织；AC7843 额外提供 `ADC_CH_INTERLEAVE38/39`

专用交织通道；内部 AMUX 通道枚举（`Adc_InternalChannelSrcType`）各型号定义不同，不可互换；AC7840E 采样时间档位与其他型号不同。

3.2.2.2 与标准或规范的差异

软件实现与芯片定义一致。

3.2.2.3 静态配置项说明

无特别的配置。

3.2.2.4 软件集成依赖

依赖模块

依赖模块	依赖说明	初始化顺序要求
CKGEN	ADC 总线时钟使能 (<code>Adc_Hal_Init</code> 内部自动调用)	系统时钟须先完成初始化
RCM	ADC 软复位（ <code>Adc_Hal_Init</code> 内部执行）	系统启动时自动初始化
DMA	DMA 通道配置与结果搬运 (<code>DmaEnable = TRUE</code> 时使用)	须先于 <code>Adc_Hal_Init()</code> 初始化 DMA 模块
CTU	路由 Timer/PWM 触发信号至 ADC (场景一)	须在 <code>Adc_Hal_ConfigGroup()</code> 之后配置 CTU 路由
Core	内核操作（关/开中断）	系统启动时自动初始化

推荐初始化调用顺序：

```

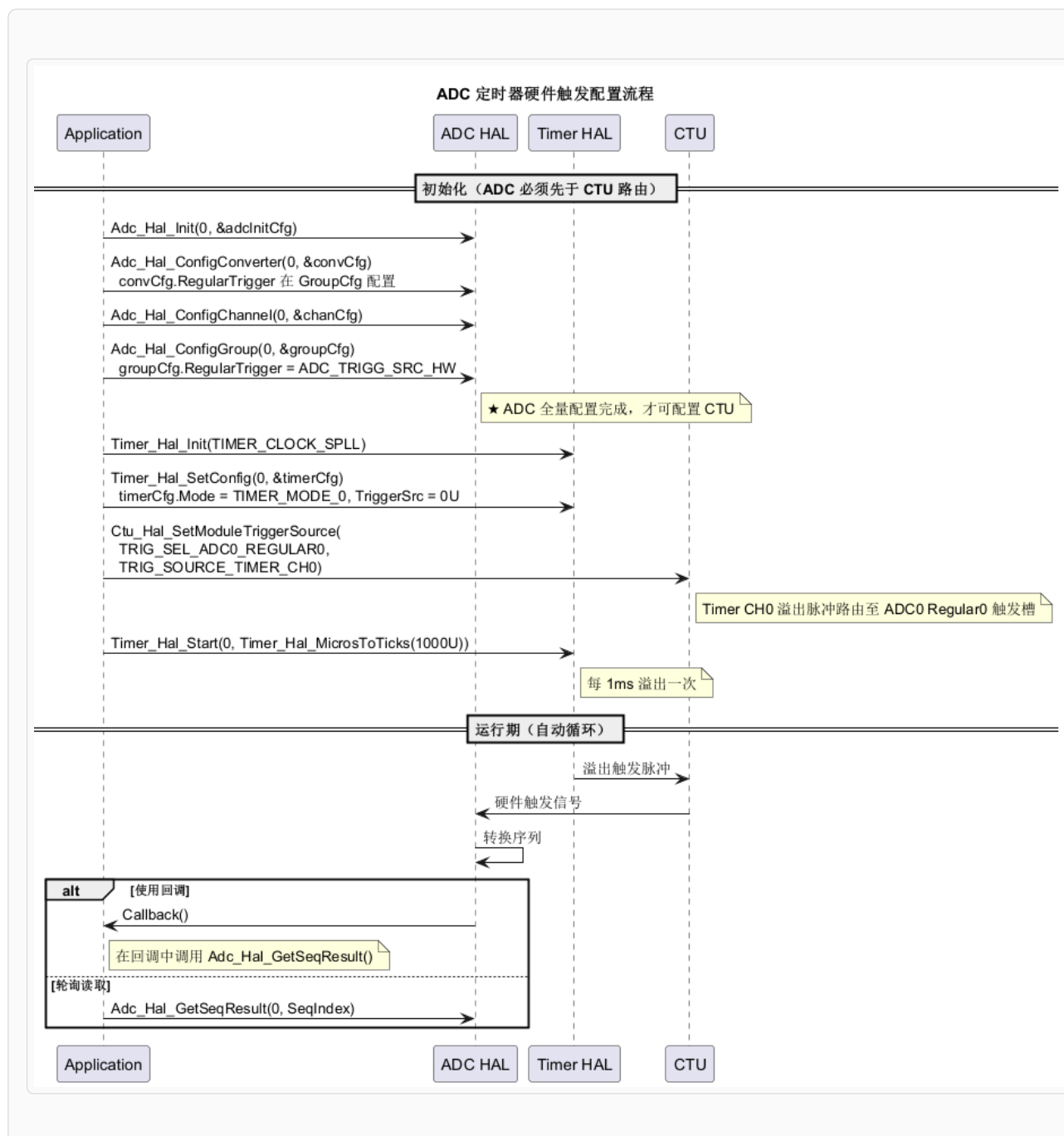
Adc_Hal_Init(Instance, &InitConfig)
↓
Adc_Hal_ConfigConverter(Instance, &ConvConfig)
↓
Adc_Hal_ConfigChannel(Instance, &ChanConfig)      /* 可多次调用，配置每个序列位 */
↓
Adc_Hal_ConfigGroup(Instance, &GroupConfig)
↓
[可选] Adc_Hal_ConfigAmo(Instance, &AmoConfig)
↓
Adc_Hal_SwTriggerRegularConvert(Instance)          /* 软件触发，或等待硬件触发 */
↓
Adc_Hal_GetSeqResult(Instance, SeqIndex)           /* 轮询读取，或在回调中读取 */

```

CTU 联动触发初始化顺序约束：配置硬件触发（`ADC_TRIGG_SRC_HW`）通过 CTU 时，**CTU 必须在 ADC 完成配置后再使能触发输出**，避免 ADC 未就绪时产生转换异常。

场景一：外部定时器硬件触发 ADC

信号路径：Timer 通道 → CTU 路由 → ADC Regular/Injection 序列硬件触发。Timer 每次溢出产生一个触发脉冲，经 CTU 路由后自动启动 ADC 转换，无需软件干预。



触发信号映射关系

CTU 目标 (Ctu_TargetModuleType)	说明
TRIG_SEL_ADC0_REGULAR0 ~ TRIG_SEL_ADC0_REGULAR3	ADC0 Regular 序列触发槽 0~3
TRIG_SEL_ADC0_INJECTION0 ~ TRIG_SEL_ADC0_INJECTION3	ADC0 Injection 序列触发槽 0~3
TRIG_SEL_ADC1_REGULAR0 ~ TRIG_SEL_ADC1_INJECTION3	ADC1 对应槽 (仅 AC7840/AC7842/AC7843)

CTU 触发源 (Ctu_TriggerSourceType)	说明
TRIG_SOURCE_TIMER_CH0 ~ TRIG_SOURCE_TIMER_CH3	Timer 通道 0~3 溢出触发

AC7840E 说明：AC7840E 仅有 ADC0，只能使用
TRIG_SEL_ADC0_REGULAR0 ~ ADC0_INJECTION3 。

完整配置步骤

步骤 1 — ADC 全量配置 (先于 CTU 路由)

```
/* 1.1 初始化 ADC */
Adc_Hal_Init(0U, &adcInitCfg);

/* 1.2 配置 Converter : 使用硬件触发模式 */
Adc_ConverterConfigType convCfg;
Adc_Hal_InitConverterStruct(&convCfg);
/* 根据需要设置时钟分频、分辨率等字段 */
Adc_Hal_ConfigConverter(0U, &convCfg);

/* 1.3 配置采样通道（可多次调用） */
Adc_ChannelConfigType chanCfg;
Adc_Hal_InitChanStruct(&chanCfg);
chanCfg.Channel = ADC_CH0; /* 目标物理通道 */
chanCfg.SeqIndex = ADC_RSEQ_0; /* Regular 序列位 0 */
Adc_Hal_ConfigChannel(0U, &chanCfg);

/* 1.4 配置 Group : 选择硬件触发 */
Adc_GroupConfigType groupCfg;
Adc_Hal_InitGroupStruct(&groupCfg);
groupCfg.RegularTrigger = ADC_TRIGG_SRC_HW; /* 硬件触发 */
groupCfg.RegularSequenceLength = 1U;
Adc_Hal_ConfigGroup(0U, &groupCfg);
/* ★ ADC 配置完成后，才可配置 CTU 触发路由 */
```

步骤 2 — 初始化 Timer

```
Timer_Hal_Init(TIMER_CLOCK_SPLL); /* 选择 SPLL 作为 Timer 时钟源 */
```

步骤 3 — 配置 Timer 通道（触发输出模式）

```
Timer_Channel_ConfigType timerCfg;
timerCfg.Mode = TIMER_MODE_0; /* 32-bit 周期计数模式 */
timerCfg.Config = 0U; /* 不开 IRQ, 仅产生溢出触发脉冲 */
timerCfg.TriggerSrc = 0U; /* Timer 本身为触发源，不接收外部触发 */
Timer_Hal_SetConfig(0U, &timerCfg); /* 通道 0 */
```

步骤 4 — CTU 路由：将 Timer CH0 溢出触发接入 ADC0 Regular0（必须在 `Adc_Hal_ConfigGroup` 之后）

```
/* 将 Timer 通道 0 溢出脉冲路由至 ADC0 Regular 序列触发槽 0 */
Ctu_Hal_SetModuleTriggerSource(TRIG_SEL_ADC0_REGULAR0, TRIG_SOURCE_TIMER_CH0);
```

如需触发 ADC1 Regular 序列，改用 `TRIG_SEL_ADC1_REGULAR0`；如需触发 Injection 序列，改用 `TRIG_SEL_ADC0_INJECTION0`。

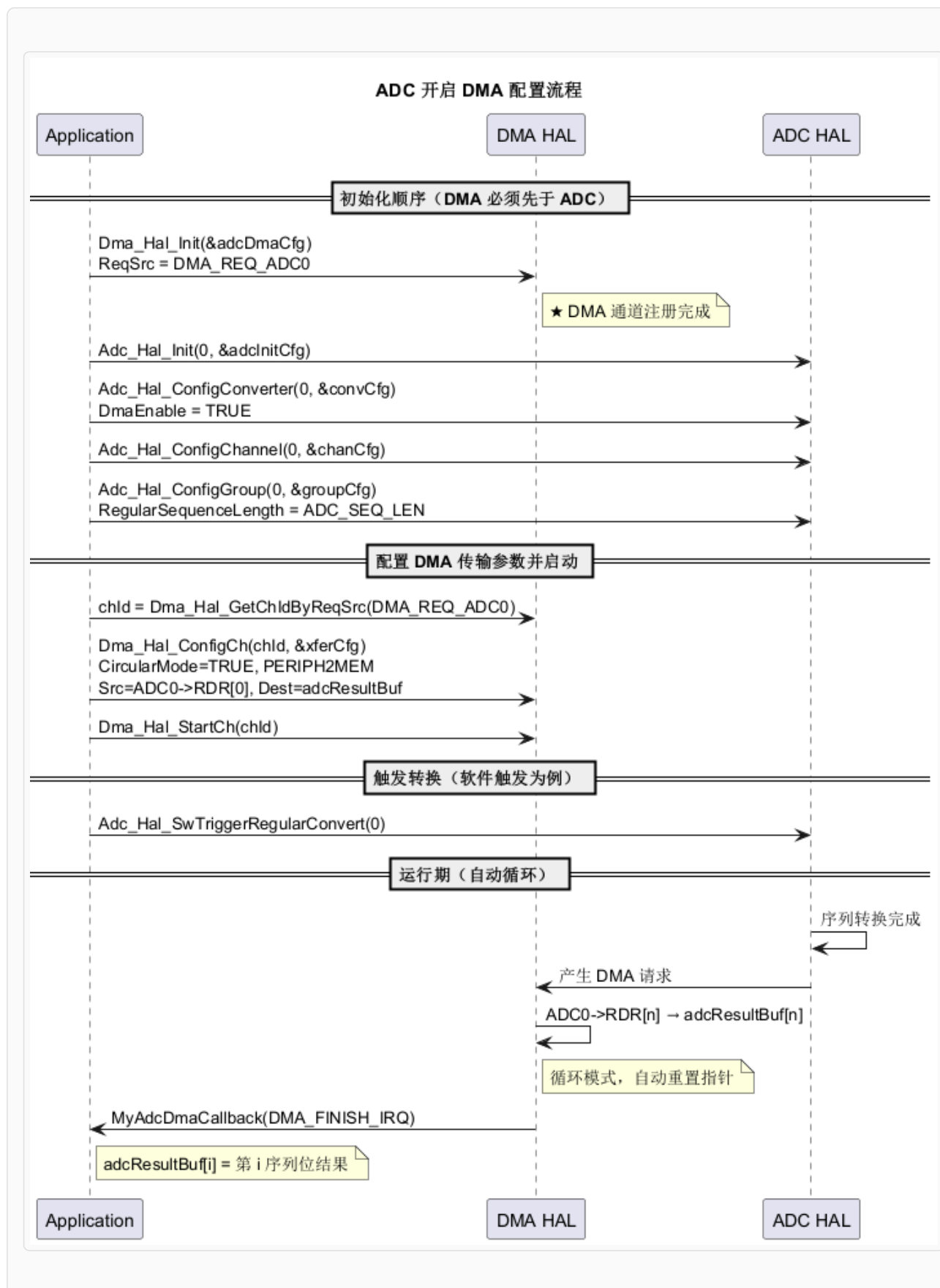
步骤 5 — 启动 Timer (ADC 即开始周期性自动转换)

```
/* 每 1000 μs (1 ms) 触发一次 ADC 转换 */
Timer_Hal_Start(0U, Timer_Hal_MicrosToTicks(1000U));
```

Timer 启动后，每次溢出都会经 CTU 自动触发 ADC 转换；结果可在 `Adc_GroupConfigType.Callback` 回调中读取，或通过 `Adc_Hal_GetSeqResult()` 轮询获取。

场景二：开启 DMA 搬运 ADC 转换结果

DMA 路径：ADC `RDR[n]` 寄存器（外设）→ DMA 引擎 → 应用结果缓冲区（内存）。每完成一次序列转换，ADC 产生 DMA 请求，DMA 自动将全部序列结果搬入用户缓冲区，无需 CPU 逐字读取。



DMA 请求源对应关系

型号	ADC0 DMA 请求源	ADC1 DMA 请求源
AC7840 / AC7842 / AC7840E	<code>DMA_REQ_ADC0</code> (值 53U)	<code>DMA_REQ_ADC1</code> (值 54U, AC7840E 不支持)
AC7843	<code>DMA_REQ_ADC0</code> (值 113U)	<code>DMA_REQ_ADC1</code> (值 114U)

AC7840E 说明： 仅有 ADC0 实例，只能使用 `DMA_REQ_ADC0`。

完整配置步骤

步骤 1 — 初始化 DMA 模块（必须先于 `Adc_Hal_Init`）

```

/* 定义 DMA 通道配置：硬件通道 0，绑定 ADC0 DMA 请求源 */
static const Dma_ChannelConfigType adcDmaChanCfg = {
    .HwChannelId      = 0U,                /* 物理 DMA 通道号，根据系统分配 */
    .VirtualChannelId = 0U,
    .Priority          = DMA_PRIORITY_LOW,
    .ReqSrc            = DMA_REQ_ADC0,     /* 绑定 ADC0 DMA 请求 */
};

static const Dma_ConfigType adcDmaCfg = {
    .ChannelCnt = 1U,
    .ChannelCfg = &adcDmaChanCfg,
};

Dma_Hal_Init(&adcDmaCfg); /* ★ 必须在 Adc_Hal_Init 之前调用 */

```

步骤 2 — ADC 初始化

```

Adc_Hal_Init(0U, &adcInitCfg);

```

步骤 3 — 配置 Converter：使能 DMA

```

Adc_ConverterConfigType convCfg;
Adc_Hal_InitConverterStruct(&convCfg);
convCfg.DmaEnable = TRUE; /* ★ 开启 DMA 传输 */
/* 根据需要设置时钟分频、分辨率等其他字段 */
Adc_Hal_ConfigConverter(0U, &convCfg);

```

步骤 4 — 配置采样通道

```

Adc_ChannelConfigType chanCfg;
Adc_Hal_InitChanStruct(&chanCfg);
chanCfg.Channel = ADC_CH0;
chanCfg.SeqIndex = ADC_RSEQ_0;
Adc_Hal_ConfigChannel(0U, &chanCfg);

```

步骤 5 — 配置 Group

```

Adc_GroupConfigType groupCfg;
Adc_Hal_InitGroupStruct(&groupCfg);
groupCfg.RegularSequenceLength = ADC_SEQ_LEN;
Adc_Hal_ConfigGroup(0U, &groupCfg);

```

步骤 6 — 配置 DMA 传输参数并启动 DMA 通道

```

static uint16_t adcResultBuf[ADC_SEQ_LEN]; /* 结果缓冲区, 大小 = 序列长度 */

/* 获取已绑定 ADC0 请求源的 DMA 通道 ID */
uint8 chId = Dma_Hal_GetChIdByReqSrc(DMA_REQ_ADC0);

Dma_TransferConfigType xferCfg;
xferCfg.TriggerMode = FALSE;
xferCfg.CircularMode = TRUE; /* 循环模式: 每次转换完成自动重置 */
xferCfg.SrcUnit = DMA_TRANSFER_UNIT_2B; /* ADC 结果寄存器 16-bit */
xferCfg.DestUnit = DMA_TRANSFER_UNIT_2B; /* 目标缓冲区 16-bit */
xferCfg.Type = DMA_TRANSFER_PERIPH2MEM; /* 外设 → 内存 */
xferCfg.SrcOffset = 4U; /* RDR 寄存器间步长 4 字节 */
xferCfg.DestOffset = 2U; /* 结果缓冲区步长 2 字节 */
xferCfg.SrcStartAddr = (uint32)&ADC0->RDR[0]; /* ADC0 结果寄存器起始地址 */
xferCfg.DestStartAddr = (uint32)&adcResultBuf[0]; /* ★ DMA 目标地址 */
xferCfg.Length = (uint16)(ADC_SEQ_LEN * 2U); /* 总字节数 = 序列长度 × 2 */
xferCfg.SrcEndAddr = xferCfg.SrcStartAddr + (xferCfg.Length / 2U) * xferCfg.SrcOffset;
xferCfg.DestEndAddr = xferCfg.DestStartAddr + xferCfg.Length;
xferCfg.Callback = MyAdcDmaCallback; /* 可选, NULL_PTR 则不使用中断 */
xferCfg.UserArgs = NULL_PTR;
xferCfg.IrqSrc = (NULL_PTR != xferCfg.Callback) ?
    ((uint8)DMA_FINISH_IRQ | (uint8)DMA_ERROR_IRQ) :
    (uint8)DMA_IRQ_NONE;

Dma_Hal_ConfigCh(chId, &xferCfg);
Dma_Hal_StartCh(chId);

```

步骤 7 — 触发 ADC 转换

```
/* 软件触发：ADC 开始转换，完成后 DMA 自动搬运结果到 adcResultBuf */
Adc_Hal_SwTriggerRegularConvert(0U);
```

```
/* 如使用硬件触发（定时器/PWM），结合场景一配置，无需此步 */
```

转换结束后，`MyAdcDmaCallback` 被调用（如已设置），`adcResultBuf[i]` 中即为第 `i` 个序列位的转换结果。

3.2.2.5 软件限制/开发须知

重要限制

1. **AC7840E 仅有 1 个 ADC 实例** `Instance` 参数只能为 `0`，传入 `1` 将访问无效硬件地址，行为未定义。
2. **采样时间枚举值跨型号不兼容** `Adc_SamplingTimeType` 在 AC7840E 和非 AC7840E 型号下枚举名称与数值均不同，跨型号移植时必须重新确认 `Spt` 字段赋值。
3. **ADC 时钟频率约束** ADC 工作时钟（总线时钟 / `ClockDivide`）的有效范围需满足芯片数据手册要求（具体最大频率限制见数据手册 [TBD]）。驱动内部对时钟频率执行校验，超出范围将不写入寄存器。
4. **ADC 参考电压范围要求** `VoltageRef` 选择 `ADC_VOLTAGEREF_VREF` 时，外部 VREF 管脚电压须满足芯片数据手册规定范围（具体电压限制 [TBD]），低于最小值时转换精度不保证。
5. **连续模式与非连续模式互斥** `ContinuousModeEn` 与 `RegularDiscontinuousModeEn` 不可同时为 `TRUE`，否则硬件行为未定义。
6. **注入非连续模式与注入自动模式互斥** `InjectDiscontinuousModeEn` 与 `InjectAutoModeEn` 不可同时为 `TRUE`。
7. **AMO 阈值约束** `AmoUpThreshold` 必须严格大于 `AmoLowThreshold`，否则 AMO 行为未定义。
8. **Adc_Hal_SelfTest 会复位 ADC 寄存器** 执行完成后 ADC 总线时钟关闭，所有寄存器恢复默认值。调用方须在 `SelfTest` 后重新执行完整初始化序列，方可正常使用 ADC。
9. **DMA 资源独占** 启用 DMA（`DmaEnable = TRUE`）时，驱动占用与该 ADC 实例绑定的 DMA 通道，应用层不可同时使用该通道。`Adc_Hal_Deinit` 时自动释放。

10. **AC7840E 不支持交织模式** `ADC_INTERLEAVE_0` / `ADC_INTERLEAVE_1` 在 AC7840E 上无效；AC7840/AC7842/AC7843 均支持交织模式。在 AC7840E 上须将 `Interleave` 字段设为 `ADC_INTERLEAVE_DISABLE`，否则行为未定义。
11. **AC7840E 不支持分辨率配置** `Adc_Reg_SetResolution` 在 AC7840E 编译时不包含（`#ifndef AC7840E` 保护），`Adc_ConverterConfigType.Resolution` 字段在 AC7840E 上将被忽略，ADC 固定以 12-bit 工作。跨型号移植时须删除分辨率配置逻辑。
12. **内部通道 AMUX 配置差异** 测量内部通道时，驱动内部通过条件编译自动适配各型号的 AMUX 路由。应用层须选择目标型号对应的 `Adc_InternalChannelSrcType` 枚举值，不同型号的枚举定义不可互换。

3.2.3 ACMP

ACMP（Analog Comparator，模拟比较器）模块提供片上模拟比较功能，支持 8 路外部输入通道、内置 DAC 参考、数字滤波、迟滞控制、轮询模式及 Hall 输出。

3.2.3.1 支持的功能

AC784xx ACMP HAL 层提供以下功能：

1. 基础比较功能

- **模块初始化与反初始化：**配置比较器参数、MUX、DAC 及轮询模式；`Acmp_Hal_Init` 内部自动使能外设时钟并执行复位，无需手动调用 CKGEN/RCM 接口
- **比较器使能控制：**通过 `Acmp_Hal_Enable` 动态上/断电，支持运行时切换
- **输出读取：**`Acmp_Hal_GetOutputData` 读取当前比较结果（0/1），正输入 > 负输入时输出 1
- **默认配置获取：**`Acmp_Hal_GetDefaultConfig` 填充安全默认值，作为自定义配置的起始点

2. 输入通道选择（MUX）

- **8 路外部通道：**`ACMP_EXTERNAL_CH0` ~ `ACMP_EXTERNAL_CH7`，正/负输入端均可独立配置
- **内置 DAC 通道：**`ACMP_DAC_OUTPUT` 可接入正输入或负输入端，用作内部参考电压

3. 内置 DAC 参考

- **参考源选择：**Bandgap（~1.2V）或 VDD，对应 `Acmp_VoltageReferenceType`
- **分辨率：**8 位（0~255），输出电压 = $V_{ref} \times Voltage / 256$
- **DAC 输出到引脚：**AC7842/AC7843 支持将 DAC 输出路由至外部引脚（`OutToPin`）

4. 中断与触发

- **边沿触发模式：**支持下降沿（`ACMP_FALLING_EDGE`）、上升沿（`ACMP_RISING_EDGE`）、双边沿（`ACMP_BOTH_EDGES`）三种中断触发方式

- **回调机制：**用户注册 `Acmp_CallbackType` 回调函数，中断发生时由驱动调用，参数携带实例号和状态标志
- **标志管理：**`Acmp_Hal_GetOutputFlags` 读取事件标志，`Acmp_Hal_ClearOutputFlags` 清除标志

5. 数字滤波

- **采样滤波：**可配置滤波采样计数（`FilterSampleCount`）与滤波时钟分频（`ACMP_FLT_DIVIDE_1 ~ ACMP_FLT_DIVIDE_8`）
- **低通滤波器：**内置模拟低通滤波器，带宽可选 500KHz / 1MHz / 2MHz / 旁路（`Acmp_LowPassFilterType`）
- **输出选择：**可选滤波后 COUT 或原始 COUTA 输出（`Acmp_OutputSelectType`）

6. 迟滞控制

- **迟滞方向：**支持仅下降沿迟滞（`ACMP_HYS_FALLING_EDGE`）或双边沿迟滞（`ACMP_HYS_BOTH_EDGE`）
- **迟滞电压等级：**0mV / 10mV / 20mV / 40mV（`Acmp_HysteresisType`），防止阈值附近信号振荡

7. 轮询模式（多通道自动扫描）

- **正/负输入轮询：**支持对正输入端（`ACMP_POSITIVE_POLLING`）或负输入端（`ACMP_NEGATIVE_POLLING`）多路通道自动轮询
- **轮询序列位图：**`PollingSequence`（bit0~bit7 分别对应 CH0~CH7），灵活选择参与轮询的通道
- **轮询时钟分频：**4 档可选（/256、/101、/71、/51），控制轮询速率
- **结果读取：**`Acmp_Hal_GetPollingData` 返回 16 位数据，每位对应一路通道的比较结果

8. Hall 输出（BLDC 电机换相）

- **Hall A/B/C 映射：**将轮询结果中指定通道的比较输出映射到 Hall A/B/C 信号（`HallAOutputCh` / `HallBOutputCh` / `HallCOutputCh`）
- **硬件换相辅助：**配合 MCU 内部电机控制外设（如 PWM/FTM），无需软件轮询即可实现换相位置检测

9. 其他特性

- **结果反相：**`InverterEnable` 对输出逻辑取反
- **窗口模式：**`WindowModeEnable` 通过外部窗口信号限制有效比较区间
- **LSI 时钟源：**AC7842/AC7843 支持使用内部低速时钟（LSI）作为比较器工作时钟（`UsingLSIEnable`）；AC7840E 不支持此选项

产品差异： `OutToPin` / `BufferEnable` (DAC 输出到引脚/缓冲) 仅 AC7842X/AC7843X 支持；`UsingLSIEnable` 仅 AC7842/AC7843 支持；AC7840E 不含上述选项。

3.2.3.2 与标准或规范的差异

软件实现与芯片定义一致。

3.2.3.3 静态配置项说明

无特别配置。

3.2.3.4 软件集成依赖

依赖模块

依赖模块	依赖说明	初始化顺序要求
CKGEN	ACMP 外设时钟使能 (<code>Acmp_Hal_Init</code> 内部自动调用)	系统时钟须先完成初始化
RCM	ACMP 软复位 (<code>Acmp_Hal_Init</code> / <code>Acmp_Hal_Reset</code> 内部执行)	系统启动时自动初始化
Core	内核操作 (关/开中断)	系统启动时自动初始化

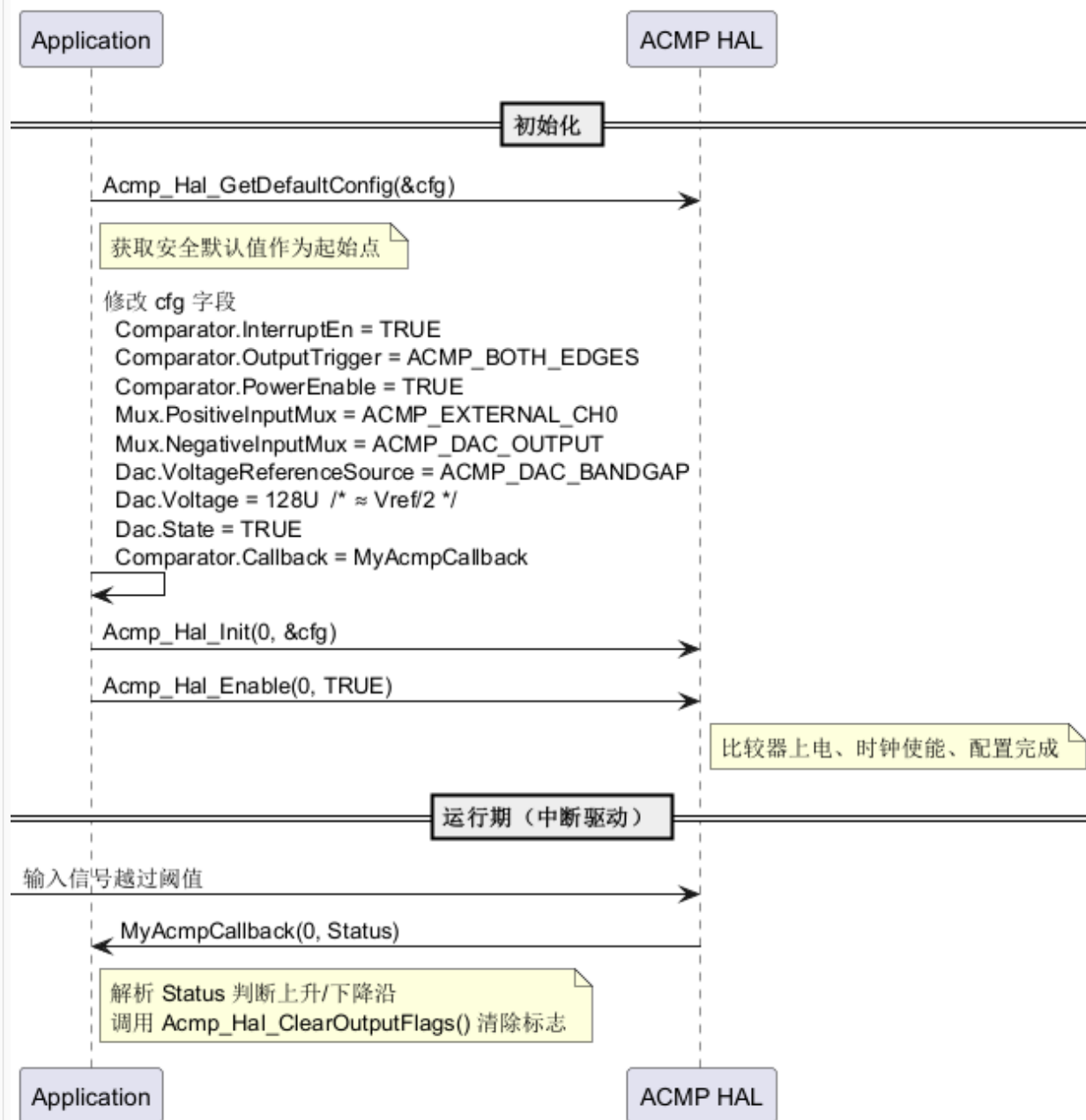
推荐初始化调用顺序：

```
Acmp_Hal_GetDefaultConfig(&Config)
↓
修改 Config 中需要自定义的字段
↓
Acmp_Hal_Init(Instance, &Config)      /* 初始化，内部自动使能时钟、复位、配置所有子模块 */
↓
Acmp_Hal_Enable(Instance, TRUE)        /* 使能比较器 */
↓
Acmp_Hal_GetOutputData(Instance)      /* 普通模式读取结果 */
或
Acmp_Hal_GetPollingData(Instance)     /* 轮询模式读取结果 */
```

场景一：外部信号过阈值检测（内置 DAC 参考 + 中断）

信号路径：外部模拟信号接入 ACMP 正输入端（CH0），内置 DAC 输出作为负输入端参考电压。当输入信号越过 DAC 设定阈值时，触发中断回调。典型用途：过压/欠压检测、模拟信号窗口保护。

ACMP 电压阈值检测配置流程



完整配置步骤

步骤 1 — 获取默认配置

```

Acmp_ModuleType acmpCfg;
Acmp_Hal_GetDefaultConfig(&acmpCfg);
    
```

步骤 2 — 配置比较器参数

```
/* 使能中断，双边沿触发（信号上升超阈值 / 下降低于阈值均触发） */
acmpCfg.Comparator.InterruptEn      = (boolean)TRUE;
acmpCfg.Comparator.OutputTrigger     = ACMP_BOTH_EDGES;
acmpCfg.Comparator.PowerEnable       = (boolean)TRUE;

/* 可选：启用数字滤波，滤除毛刺干扰 */
acmpCfg.Comparator.FilterEnable      = (boolean)TRUE;
acmpCfg.Comparator.ClockDivide       = ACMP_FLT_DIVIDE_4;
acmpCfg.Comparator.FilterSampleCount = 4U;

/* 选择输出滤波后的结果（降噪） */
acmpCfg.Comparator.OutputSelect      = ACMP_COUT;

/* 迟滞控制：双边沿 20mV 迟滞，防止阈值附近振荡 */
acmpCfg.Comparator.HysteresisMode    = ACMP_HYS_BOTH_EDGE;
acmpCfg.Comparator.HysteresisLevel   = ACMP_LEVEL_HYS_20MV;

/* 注册中断回调 */
acmpCfg.Comparator.Callback          = MyAcmpCallback;
```

步骤 3 — 配置输入 MUX：正输入接外部 CH0，负输入接内置 DAC

```
acmpCfg.Mux.PositiveInputMux = ACMP_EXTERNAL_CH0; /* 待监测的模拟信号 */
acmpCfg.Mux.NegativeInputMux = ACMP_DAC_OUTPUT; /* 内置 DAC 作为参考电平 */
```

步骤 4 — 配置内置 DAC（参考阈值电压）

```
/* DAC 输出 = Vref × Voltage / 256
 * 以 Bandgap (~1.2V) 为参考：Voltage = 128 → ~0.6V
 * 以 VDD (3.3V) 为参考：Voltage = 128 → ~1.65V */
acmpCfg.Dac.VoltageReferenceSource = ACMP_DAC_VDD;
acmpCfg.Dac.Voltage                = 128U; /* 阈值 ≈ VDD/2 ≈ 1.65V */
acmpCfg.Dac.State                  = (boolean)TRUE;
```

步骤 5 — 初始化并使能

```
Acmp_Hal_Init(0U, &acmpCfg);
Acmp_Hal_Enable(0U, (boolean)TRUE);
```

步骤 6 — 中断回调实现


```
void MyAcmpCallback(uint8 Instance, uint32 Status)
{
    uint8 output;

    /* 清除输出标志（须在读取前/后清除，防止重复触发） */
    Acmp_Hal_ClearOutputFlags(Instance);

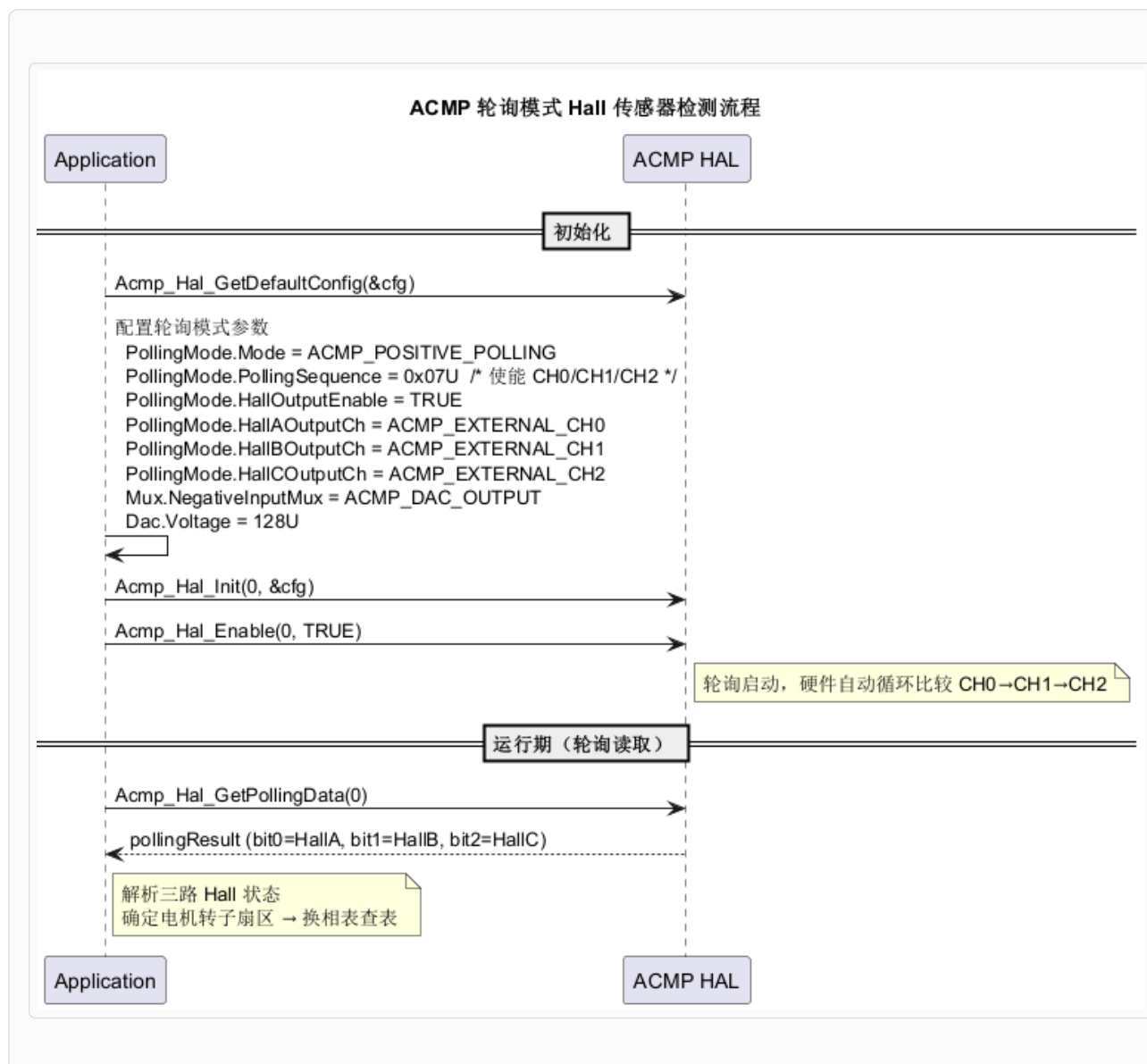
    /* 读取当前比较器输出电平 */
    output = Acmp_Hal_GetOutputData(Instance);

    if (output != 0U)
    {
        /* 输入信号高于 DAC 参考阈值：过压事件 */
    }
    else
    {
        /* 输入信号低于 DAC 参考阈值：欠压事件 */
    }
}
```

DAC 阈值计算： $V_{threshold} = V_{ref} \times Voltage / 256$ 。以 ACMP_DAC_VDD (3.3V) 为参考， $Voltage = 100$ 时阈值约为 1.29V； $Voltage = 200$ 时阈值约为 2.58V。

场景二：多路轮询模式（BLDC Hall 传感器检测）

信号路径：3 路 Hall 传感器信号分别接入 ACMP 正输入通道 CH0/CH1/CH2，内置 DAC 作为固定参考，ACMP 轮询模式自动逐路比较，输出 Hall A/B/C 逻辑电平。典型用途：无刷直流电机换相位置检测。



完整配置步骤

步骤 1 — 获取默认配置并配置 DAC 参考（中点电压）

```

Acmp_ModuleType hallCfg;
Acmp_Hal_GetDefaultConfig(&hallCfg);

/* DAC 参考: VDD/2 作为 Hall 信号的逻辑判决阈值 */
hallCfg.Dac.VoltageReferenceSource = ACMP_DAC_VDD;
hallCfg.Dac.Voltage                = 128U; /* ≈ VDD/2 */
hallCfg.Dac.State                  = (boolean)TRUE;
hallCfg.Comparator.PowerEnable     = (boolean)TRUE;
    
```

步骤 2 — 配置输入 MUX（负输入接 DAC，正输入由轮询模式控制）

```
/* 轮询模式下，正输入端由硬件自动切换，只需固定负输入端 */
hallCfg.Mux.PositiveInputMux = ACMP_EXTERNAL_CH0; /* 初始值（轮询模式下被覆盖）*/
hallCfg.Mux.NegativeInputMux = ACMP_DAC_OUTPUT;
```

步骤 3 — 配置轮询模式

```
/* 正输入轮询：按序对 CH0/CH1/CH2 逐通道与 DAC 比较 */
hallCfg.PollingMode.Mode = ACMP_POSITIVE_POLLING;

/* 轮询序列位图：Bit0=CH0, Bit1=CH1, Bit2=CH2, 三路均参与 */
hallCfg.PollingMode.PollingSequence = 0x07U;

/* 轮询时钟分频，决定轮询速率（系统时钟 / 51） */
hallCfg.PollingMode.PollingClockDivide = ACMP_CLK_DIVIDE_51;

/* 使能 Hall 输出，将轮询结果映射到 Hall A/B/C */
hallCfg.PollingMode.HallOutputEnable = (boolean)TRUE;
hallCfg.PollingMode.HallAOutputCh = ACMP_EXTERNAL_CH0; /* Hall A 对应 CH0 */
hallCfg.PollingMode.HallBOutputCh = ACMP_EXTERNAL_CH1; /* Hall B 对应 CH1 */
hallCfg.PollingMode.HallCOutputCh = ACMP_EXTERNAL_CH2; /* Hall C 对应 CH2 */
```

步骤 4 — 初始化并使能

```
Acmp_Hal_Init(0U, &hallCfg);
Acmp_Hal_Enable(0U, (boolean)TRUE);
```

步骤 5 — 主循环中读取 Hall 状态并换相

```
/* Hall 扇区换相表（6步换相）：Hall[A/B/C] → 换相步序 */
static const uint8 HallSectorTable[8U] = {
    0U, /* 000 - 无效 */
    5U, /* 001 - 扇区 5 */
    3U, /* 010 - 扇区 3 */
    4U, /* 011 - 扇区 4 */
    1U, /* 100 - 扇区 1 */
    6U, /* 101 - 扇区 6 */
    2U, /* 110 - 扇区 2 */
    0U, /* 111 - 无效 */
};

void Motor_UpdateCommutation(void)
{
    uint16 pollingData;
    uint8 hallIndex;
    uint8 sector;

    /* 读取三路轮询比较结果（Bit0=HallA, Bit1=HallB, Bit2=HallC） */
    pollingData = Acmp_Hal_GetPollingData(0U);
    hallIndex = (uint8)(pollingData & 0x07U);

    /* 查表得当前扇区，驱动换相 */
    sector = HallSectorTable[hallIndex];
    if (sector != 0U)
    {
        Motor_SetCommutation(sector);
    }
}
```

轮询速率估算： $T_{poll} = (1/F_{sys}) \times PollingDivide \times \text{通道数}$ 。以系统时钟 80MHz、`ACMP_CLK_DIVIDE_51` (/51)、3 路通道为例，单次完整轮询周期约为 $1/80\text{MHz} \times 51 \times 3 \approx 1.9\mu\text{s}$ 。

3.2.3.5 软件限制/开发须知

重要限制

- AC7840E 不支持 LSI 时钟源** `Acmp_ComparatorType.UsingLSIEnable` 在 AC7840E 编译时不包含（`#if !defined(AC7840E)` 保护）。AC7840E 上初始化 `Acmp_ModuleType` 时不得访问此字段。

2. **DAC OutToPin / BufferEnable 仅 AC7842/AC7843 支持** `Acmp_DacType.OutToPin` 和 `BufferEnable` 字段在 AC7840/AC7840E 上不包含。跨型号移植时须删除相关初始化代码。
3. **AC7843 有 2 个 ACMP 实例** AC7843 上 `Instance` 延伸至 1 (ACMP1)。其余型号上 `Instance` 只能为 0，传入 1 将访问无效硬件地址。
4. **Acmp_Hal_Init 内部执行硬件复位** `Acmp_Hal_Init` 内部调用 `Acmp_Hal_Reset`，将所有寄存器恢复默认值后再写入配置。如需保留部分寄存器状态，应先调用 `Acmp_Hal_GetConfigAll` 读取当前配置。
5. **PowerEnable 必须置 TRUE** `Acmp_ComparatorType.PowerEnable` 为 `FALSE` 时比较器模拟电源断开，调用 `Acmp_Hal_GetOutputData` 返回的结枚不可靠。
6. **轮询模式与普通比较模式切换** 切换轮询模式需要重新调用 `Acmp_Hal_Init`，不得在运行中直接修改 `PollingMode.Mode` 寄存器位。
7. **正/负输入端不得同时内接 DAC 输出** 若 `PositiveInputMux` 和 `NegativeInputMux` 同时设为 `ACMP_DAC_OUTPUT`，比较器输出始终为固定值，行为无意义。

3.2.4 GPIO

3.2.4.1 支持的功能

- 引脚复用 (Mux) 设置: `Gpio_Hal_SetMuxMode`
- 引脚方向与输出控制: `Gpio_Hal_SetPinDirection` / `Gpio_Hal_WritePin` / `Gpio_Hal_WritePins` / `Gpio_Hal_TogglePin`
- 引脚输入读取: `Gpio_Hal_ReadPin` / `Gpio_Hal_ReadPins`
- 上拉/下拉配置: `Gpio_Hal_SetPullSel`
- 中断配置与回调: `Gpio_Hal_SetPinIntSel` / `Gpio_Hal_InstallCallback` / `Gpio_Hal_ClearIntStatus`
- 数字滤波与高阻态控制: `Gpio_Hal_EnableDigitalFilter` / `Gpio_Hal_ConfigDigitalFilter` / `Gpio_Hal_SetHighZ`

3.2.4.2 与标准或规范的差异

GPIO HAL 为设备寄存器的直接抽象层，实现与硬件寄存器映射一致的基础功能。功能点（方向、拉电阻、中断、复用）与通用 MCU GPIO 行为一致，无协议层面差异。

3.2.4.3 静态配置项说明

- 无

3.2.4.4 软件集成依赖

- 无

3.2.4.5 软件限制/开发须知

- **复用冲突规避**：当引脚被复用为其他外设功能（如 SPI_SCK）时，禁止再通过 GPIO API 修改其电平或方向，否则可能导致通信异常。
- **引脚锁定 (PinLock)**：若配置了 `PinLock = TRUE`，则初始化后该引脚的配置寄存器 (PCR) 将被锁定，直到下次复位，运行时无法再通过 `SetMuxMode` 等 API 修改。
- **中断共享**：同一端口（如 PORTA）的所有引脚共用一个中断向量，在回调函数中需通过 `Status` 位判断具体触发引脚。
- **高阻态注意事项**：`Gpio_Hal_SetHighZ` 会切断引脚的驱动与输入路径，常用于低功耗或特定总线隔离场景。

3.3 System

3.3.1 CRC

3.3.1.1 支持的功能

- 硬件 CRC 引擎初始化/反初始化：`Crc_Hal_Init` / `Crc_Hal_Deinit`
- 硬件加速 CRC 计算：`Crc_Hal_CalculateCRC`（按字/半字/字节处理）
- DMA 加速 CRC：`Crc_Hal_DmaCalculateCRC`（启动 DMA 计算并异步获取结果）
- 读取结果与配置：`Crc_Hal_GetCRCResult` / `Crc_Hal_SetSeed` / `Crc_Hal_GetConfig`
- 纯软件 CRC 函数（兼容不同协议）：`Crc_Hal_CalculateCRC8`，`Crc_Hal_CalculateCRC16`，`Crc_Hal_CalculateCRC32`，`Crc_Hal_CalculateCRC64`，及 CRC8H2F/CRC16ARC/CRC32P4 等

3.3.1.2 与标准或规范的差异

- 支持多种纯软件 CRC 实现，包括 CRC8H2F、CRC8、CRC16ARC、CRC32P4 等。

3.3.1.3 静态配置项说明

无

3.3.1.4 软件集成依赖

依赖模块	说明
CKGEN	硬件 CRC 引擎时钟需启用
DMA	使用 <code>Crc_Hal_DmaCalculateCRC</code> 时需确保 DMA 模块已就绪并分配了可用通道

3.3.1.5 软件限制/开发须知

- **硬件重入性限制：**硬件 CRC 引擎为单实例外设，非重入。若多任务同时调用硬件计算，需在应用层使用互斥锁（Mutex）或信号量保护。
- **DMA 完成检测：**调用 `Crc_Hal_DmaCalculateCRC` 后，必须在读取结果前确认 DMA 传输已完成（通过 DMA 回调或查询状态位）。
- **分段计算：**对于超大数据块，可多次调用 `Crc_Hal_CalculateCRC`，硬件会自动保留中间结果，或通过 `Crc_Hal_SetSeed` 手动注入上次计算的余数。
- **软件实现宏开关：**纯软件 CRC 函数通常需要定义 `SW_CRC_SUPPORT` 宏后方可链接使用。

3.3.2 PCT

3.3.2.1 支持的功能

- **两种工作模式：**定时器（Timer）模式与脉冲计数器（Pulse Counter）模式，通过 `Pct_ModeType` 选择
- **初始化/反初始化：**`Pct_Hal_Init` / `Pct_Hal_Deinit`，需在操作 PCT 前首先调用
- **配置与运行控制：**`Pct_Hal_SetConfig`（运行时更新配置）、`Pct_Hal_Start` / `Pct_Hal_Stop`
- **比较值读取/设置：**支持以计数刻度（Tick）或微秒（μs）为单位
 - `Pct_Hal_SetCompareValueByCount` / `Pct_Hal_GetCompareValueByCount`
 - `Pct_Hal_SetCompareValueByUs` / `Pct_Hal_GetCompareValueByUs`
- **中断与回调：**`Pct_Hal_SetInterrupt`（使能/禁止中断）、`Pct_Hal_InstallCallback`（注册回调）、`ISR(PCT_IRQHandler)`
- **比较标志查询/清除：**`Pct_Hal_GetCompareFlag` / `Pct_Hal_ClearCompareFlag`
- **运行时计数读取：**`Pct_Hal_GetCurrentValue`，获取当前计数器值
- **状态查询：**`Pct_Hal_GetStatus`，返回模块当前状态（UNINITED / IDLE / RUNNING）

集成场景示例：Pulse Counter 模式用于统计外部脉冲数量，Timer 模式用于定时触发事件。

3.3.2.2 与标准或规范的差异

- AC7840E 无 PCT 外设
- PCT 模块仅在 AC7840 / AC7842 / AC7843 上可用，AC7840E 芯片不支持该外设

3.3.2.3 静态配置项说明

无

3.3.2.4 软件集成依赖

依赖模块	说明
CKGEN	PCT 时钟源（如 HSIDIV2 / PCC）必须通过 <code>Ckgen_Hal_Init</code> 启用并稳定
CTU	若选择 <code>PCT_PINSELECT_TRGMUX</code> ，需通过 CTU 配置触发路由
GPIO	若需要接入外部时钟输入，需配置对应 GPIO 复用为 PCT 功能

3.3.2.5 软件限制/开发须知

- **Prescaler 与 FreeRun 互斥：**FreeRun 模式启用时不能同时启用预分频（`BypassPrescaler` 除外），否则 `Pct_Hal_SetConfig` 返回 `STATUS_ERROR`
- **脉冲计数滤波：**在滤波模式下（`PreSc1 >= PCT_PRESCALE_4_GLITCHFILTER_2`），滤波宽度会引入额外延迟，高频脉冲可能丢失
- **BUSCLK 与 FUNCTION_CLK 比率：**`Pct_CVALSelectType` 需根据实际时钟比率选择，以避免比较值读取时出现一致性问题
- **比较值单位转换精度：**`Pct_Hal_SetCompareValueByUs` 的转换精度依赖当前时钟频率配置，时钟切换后需重新计算

3.3.3 PDT

3.3.3.1 支持的功能

- **初始化/反初始化：**`Pdt_Hal_Init` / `Pdt_Hal_Deinit`

• 定时器配置与运行控制：

- `Pdt_Hal_ConfigTimer`：配置 timer 参数（连续/单次模式、时钟预分频、触发源、模数值、中断延迟等）
- `Pdt_Hal_Start` / `Pdt_Hal_Stop`：启动/停止定时器
- `Pdt_Hal_LoadValuesCmd`：将配置值从阴影寄存器加载到工作寄存器（需配合 `Pdt_Hal_ConfigTimer` 使用）
- `Pdt_Hal_SoftTriggerCmd`：软件触发定时器

• 触发延迟通道：8 个独立延迟通道（`PDT_DLY_0` ~ `PDT_DLY_7`），支持：

- `Pdt_Hal_ConfigTriggerDelay`：配置延迟参数和使能遮罩
- `Pdt_Hal_SetTriggerDelayValue`：设置指定通道的延迟值
- 触发延迟支持硬件/软件两种触发源

• 脉冲输出：

- `Pdt_Hal_ConfigPulseOut`：配置脉冲输出参数（高/低电平持续时间）
- `Pdt_Hal_SetCmpPulseOutEnable` / `Pdt_Hal_SetCmpPulseOutDelayForHigh` / `Pdt_Hal_SetCmpPulseOutDelayForLow`

• 中断与回调：`Pdt_Hal_EnableInterrupt` / `Pdt_Hal_InstallCallback` / `ISR(PDT0_IRQHandler)` 等

• 状态与标志：`Pdt_Hal_GetStatus` / `Pdt_Hal_ClearTimerIntFlag`

• 模数值与中断延迟值动态更新：`Pdt_Hal_SetTimerModulusValue` / `Pdt_Hal_SetValueForTimerInterrupt`

集成场景示例：PDT 可用于发动机喷油时序控制（触发延迟）、PWM 脉冲生成（脉冲输出）等高精度时序控制场景。

3.3.3.2 与标准或规范的差异

- AC7840E 无 PDT 外设
- PDT 模块仅在 AC7840 / AC7842 / AC7843 上可用，AC7840E 芯片不支持该外设

3.3.3.3 静态配置项说明

无

3.3.3.4 软件集成依赖

依赖模块	说明
CKGEN	PDT 时钟来源需配置并稳定，通过 <code>Ckgen_Hal_GetFreq</code> 获取频率用于时间计算
CTU	PDT 可连接多种硬件触发源（外部触发/CTU/Timer 等），触发路由需在 CTU 中配置

3.3.3.5 软件限制/开发须知

- **Load 模式选择对时序的影响：** `LoadValueMode` 选择立即加载会在下次写 `LDOK` 位后立即更新计数器；选择计数到或触发加载时，数值更新存在一个触发周期的延迟，变更前需保证模块处于 IDLE 状态
- **脉冲输出分辨率限制：** 脉冲输出的高/低电平持续时间（`Poly1` / `Poly2`）分辨率为一个 PDT 时钟周期，受预分频/预乘法配置影响，超出范围返回错误或行为未定义
- **连续模式与非连续模式切换：** 在定时器运行状态下（`Pdt_Hal_GetStatus` 返回 RUNNING）不允许切换连续/单次模式，需先 Stop 后重新 Config

3.3.4 TIMER

特性	AC7840	AC7840E	AC7842	AC7843
Timer 通道数	4	4	4	4
时钟源 (SPLL/VHSI/HSI/HSE)	✓	✓	✓	✓
32 位周期定时器 (模式 0)	✓	✓	✓	✓
双 16 位周期定时器 (模式 1)	✓	✓	✓	✓
32 位脉冲累加器 (模式 2)	✓	✓	✓	✓
32 位脉冲捕获器 (模式 3)	✓	✓	✓	✓
中断 + 回调支持	✓	✓	✓	✓
调试模式运行控制	✓	✓	✓	✓
单次触发模式	✓	✓	✓	✓

3.3.4.1 支持的功能

AC784xx Timer HAL 层提供以下功能:

1. 基础定时器功能

• 初始化与反初始化

- 指定时钟源 (`TIMER_CLOCK_SPLL` / `TIMER_CLOCK_VHSI` / `TIMER_CLOCK_HSI` / `TIMER_CLOCK_HSE`) 初始化 Timer 模块
- 通过 `Timer_Hal_DeInit()` 释放 Timer 模块 (所有通道需已停止)

• *通道配置

- 通过 `Timer_Hal_SetConfig(Channel, ConfigPtr)` 配置通道工作模式与选项:
- **Mode**: 选择 `TIMER_MODE_0` (32 位周期)、`TIMER_MODE_1` (双 16 位周期)、`TIMER_MODE_2` (脉冲累加)、`TIMER_MODE_3` (脉冲捕获)
- **Config**: 按位组合 `TIMER_ONESHOT_EN` (单次触发)、`TIMER_DBG_EN` (调试模式保持运行)、`TIMER_CHN_EN` (通道链接)、`TIMER_TROT`、`TIMER_TSOT`
- **TriggerSrc**: 指定触发源 (用于脉冲累加/捕获模式)
- **注意**: 默认配置下已定义 `WDG_SDK_NON_EXTENDED_API`, `Timer_Hal_SetConfig` / `Timer_Hal_GetConfig` / `Timer_Hal_GetRemainingValue` / `Timer_Hal_ResetValue` 不参与编译

• 启动与停止

- `Timer_Hal_Start(Channel, Timeout)`: 启动指定通道, 设置超时计数值
- `Timer_Hal_Stop(Channel)`: 停止指定通道

• 状态查询

- `Timer_Hal_GetCurrentValue(Channel)`: 获取当前已计数 Tick 值
- `Timer_Hal_MicrosToTicks(Micros)`: 将微秒换算为基于当前时钟源的 Tick 值

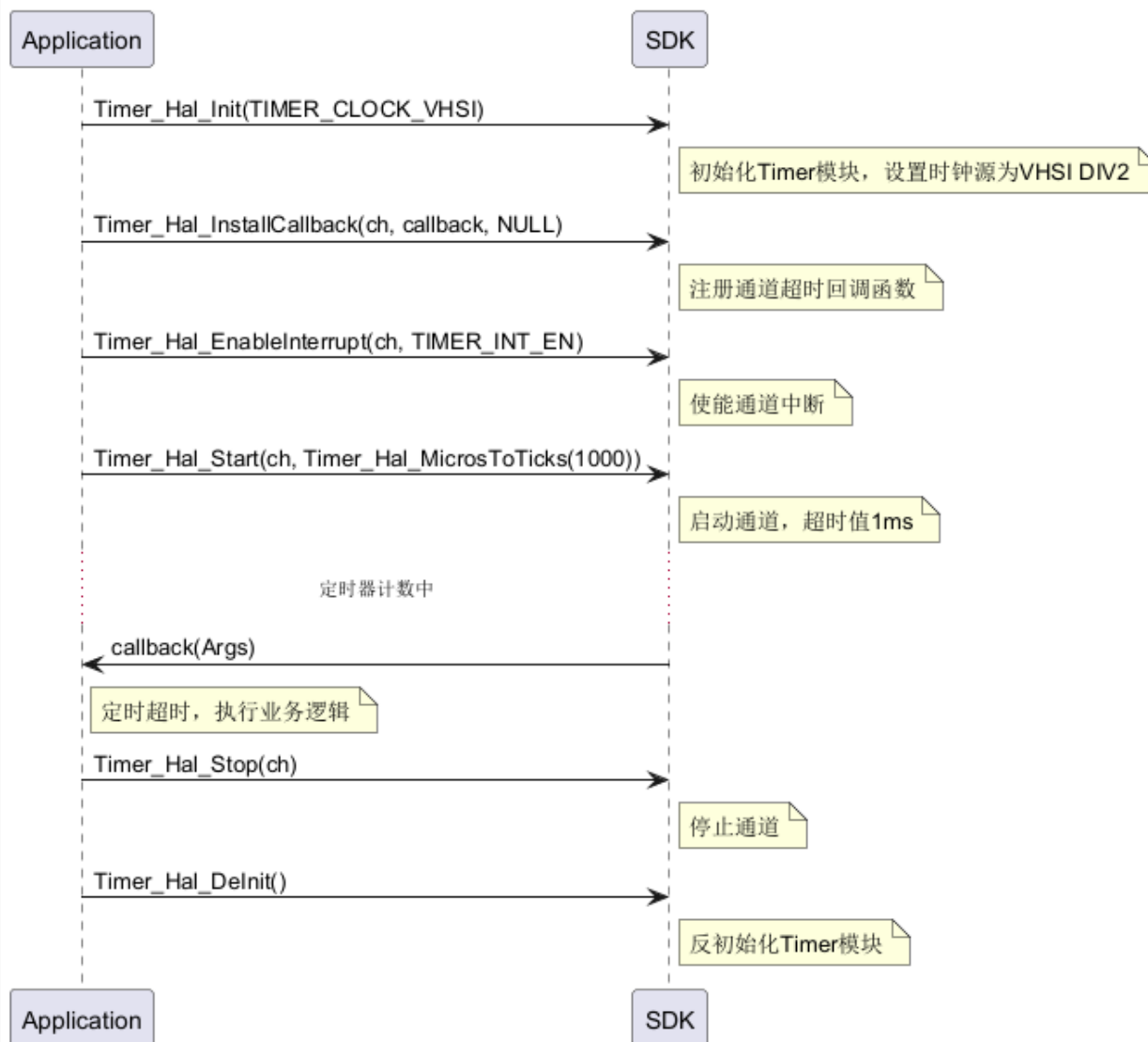
• 中断与回调

- `Timer_Hal_EnableInterrupt(Channel, InterruptBits)`: 使能/禁用通道中断
- `Timer_Hal_InstallCallback(Channel, Func, Args)`: 注册通道超时回调函数

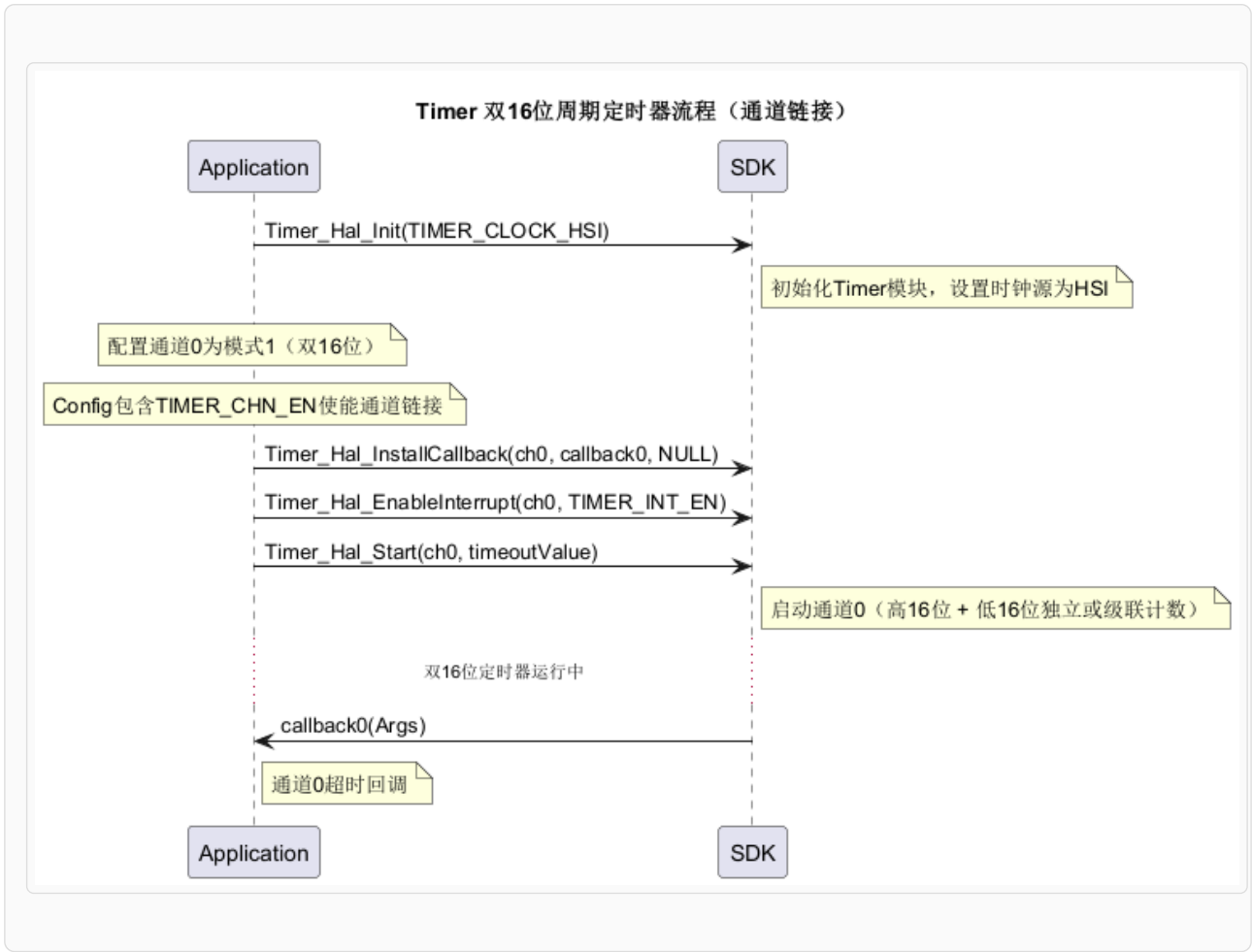
典型使用流程时序图

1. 32 位周期定时器流程 (VHSI 时钟源)

Timer 32位周期定时器流程（VHSI时钟源）



2. 双 16 位周期定时器流程（通道链接，HSI 时钟源）



3.3.4.2 与标准或规范的差异

软件实现与芯片规格一致。

3.3.4.3 静态配置项说明

无特别配置

3.3.4.4 软件集成依赖

依赖模块

Timer HAL 层功能依赖以下模块正确配置：

依赖模块	依赖说明	初始化顺序要求
CKGEN	Timer 时钟源（SPLL / VHSI / HSI / HSE）须由 CKGEN 模块提供 时钟频率直接影响超时 Tick 值的实际 时间计算	必须先初始化后才能使用 Timer

3.3.4.5 软件限制/开发须知

重要限制

- 调试模式行为** - 配置 `TIMER_DBG_EN` 后，调试器暂停期间定时器**仍继续计数** - 调试时可能出现非预期超时中断，建议调试阶段不配置 `TIMER_DBG_EN`
- DeInit 前置条件** - `Timer_Hal_DeInit()` 要求**所有通道均已停止**，否则返回 `STATUS_TIMER_WRONG_STATE` - 请确认在调用 `Timer_Hal_DeInit()` 前，已对所有已启动的通道调用 `Timer_Hal_Stop()`
- 通道链接 (TIMER_CHN_EN) 使用注意** - 通道链接功能用于实现双 16 位级联计数（模式 1），启用时相邻通道配置需协调一致 - 错误的链接配置可能导致计数行为异常
- Tick 换算精度** - `Timer_Hal_MicrosToTicks()` 基于当前时钟源频率进行整数换算，存在截断误差 - 对超时精度要求较高的场景，建议选用精度更高的时钟源（如 HSE）

3.3.5 RTC

特性	AC7840	AC7840E	AC7842	AC7843
RTC 实例数	1	1	1	1
时钟源 HSE / VHSE / LSI32K / CLKIN	✓	✓	✓	✓
时钟源 STB_LSI32K / STB_CLKIN (SPM)	✓	X	X	X
定时器溢出中断	✓	✓	✓	✓
闹钟中断	✓	✓	✓	✓
预分频器溢出中断	✓	✓	✓	✓
RTC 时钟输出	✓	✓	✓	✓
暂停 (Pause) 功能	✓	✓	✓	✓

3.3.5.1 支持的功能

AC784xx RTC HAL 层提供以下功能：

1. 基础 RTC 功能

• 初始化与反初始化

- 通过 `Rtc_Hal_Init(Clk)` 指定时钟源初始化 RTC 模块

- 通过 `Rtc_Hal_DeInit()` 释放 RTC 模块（需已停止）

• 启动、停止与暂停

- `Rtc_Hal_Start(Unit)`：启动 RTC，`Unit` 为预分频值，决定实时寄存器的计数单位
- `Rtc_Hal_Stop()`：停止 RTC 计数
- `Rtc_Hal_Pause()`：暂停 RTC 计数（RTC 已启动时可调用）

• 计数值读写

- `Rtc_Hal_GetCurrentValue()`：读取当前实时计数寄存器值
- `Rtc_Hal_SetCurrentValue(Value)`：写入初始计数值（RTC 未启动时调用）
- `Rtc_Hal_GetPrescalerValue()`：读取预分频寄存器当前值
- `Rtc_Hal_GetWorkFreq()`：查询当前工作时钟频率

• 闹钟功能

- `Rtc_Hal_SetAlarm(Value)`：设置闹钟比较值，RTC 计数达到该值时触发闹钟中断

• 时钟输出配置

- `Rtc_Hal_SetConfig(OutputCfg)`：配置 RTC 时钟输出模式（`RTC_OUTPUT_DISABLED` / `RTC_OUTPUT_CLOCK` / `RTC_OUTPUT_PRESCALER`）

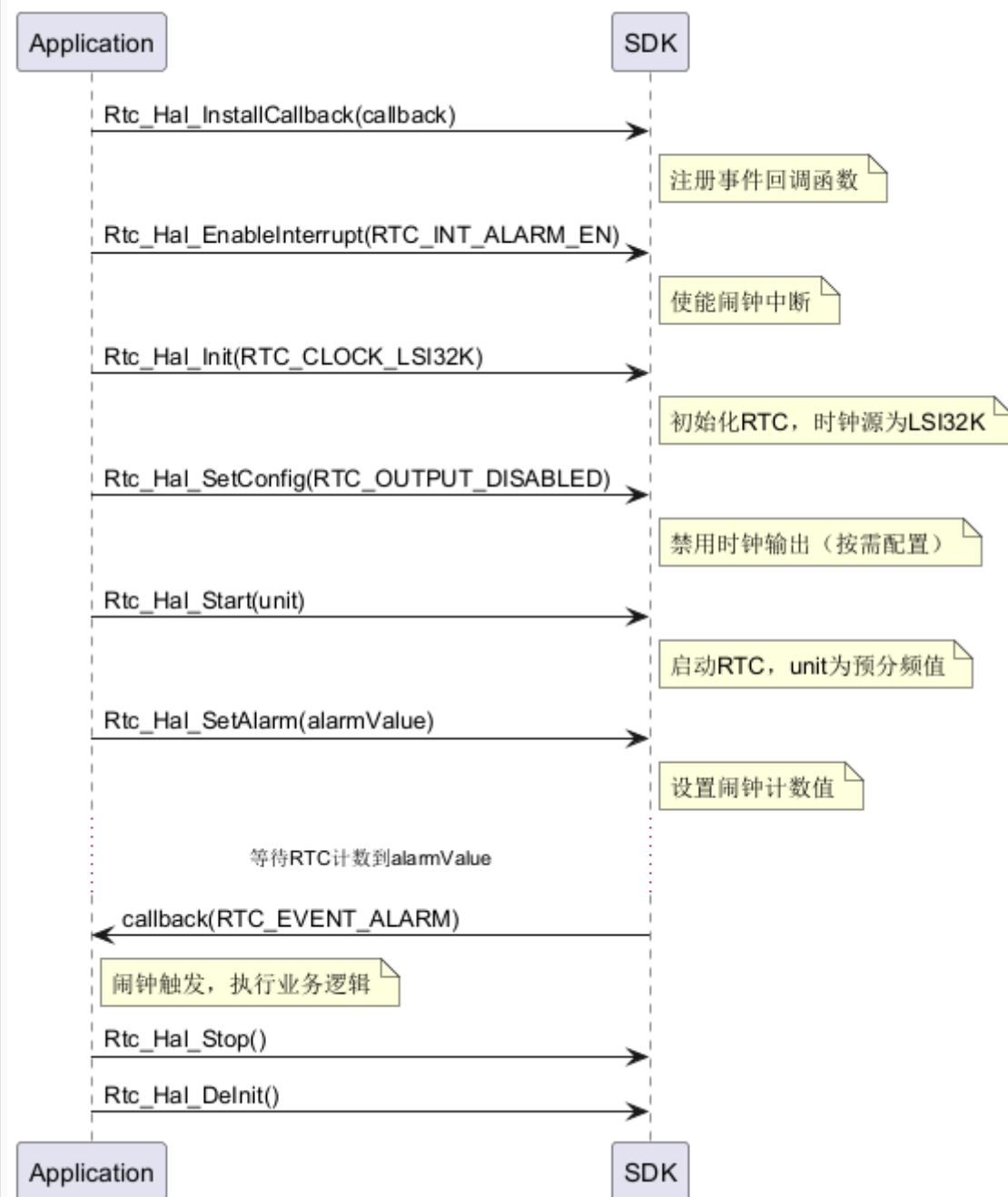
• 中断与回调

- `Rtc_Hal_InstallCallback(Func)`：注册事件回调函数，回调参数 `Event` 为以下事件的按位组合：
 - `RTC_EVENT_TIMER`：定时器溢出
 - `RTC_EVENT_ALARM`：闹钟触发
 - `RTC_EVENT_PRESCALER`：预分频器溢出
- `Rtc_Hal_EnableInterrupt(InterruptBits)`：使能/禁用中断，`InterruptBits` 为 `RTC_INT_TIMER_EN` / `RTC_INT_ALARM_EN` / `RTC_INT_PRESCALER_EN` 的按位组合

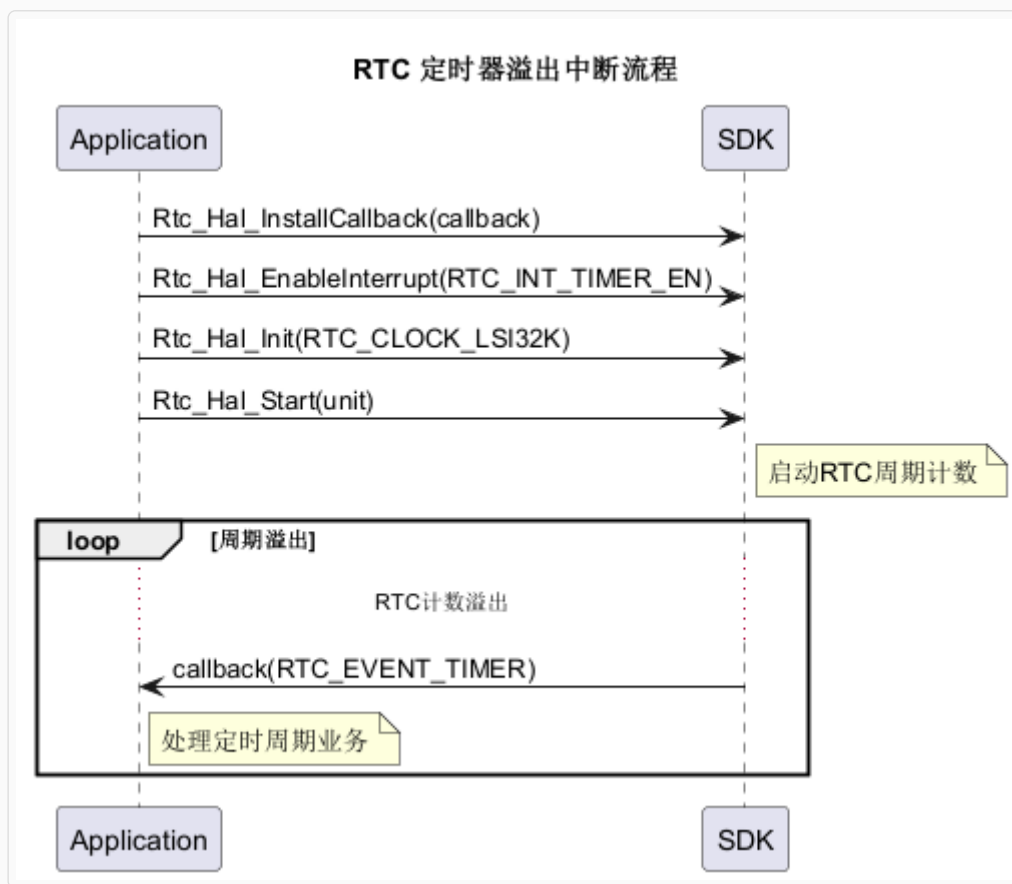
典型使用流程时序图

1. 闹钟功能流程（LSI32K 时钟源）

RTC 闹钟功能流程 (LSI32K时钟源)



2. 定时器溢出周期中断流程



3.3.5.2 与标准或规范的差异

软件实现与芯片定义一致。

3.3.5.3 静态配置项说明

当前 RTC HAL 层无独立的编译时裁剪宏，时钟源选项通过 `Rtc_ClockSourceType` 枚举值在运行时配置。

- AC7840x 特有时钟源（`RTC_CLOCK_STB_LSI32K` / `RTC_CLOCK_STB_CLKIN`）仅在宏 `AC7840X` 已定义时参与编译
- 其余型号编译时不包含这两个枚举值，使用时须通过 `#if defined(AC7840X)` 条件编译保护

3.3.5.4 软件集成依赖

依赖模块

RTC HAL 层功能依赖以下模块正确配置：

依赖模块	依赖说明	初始化顺序要求
CKGEN	RTC 时钟源（HSE / VHSI / LSI32K / CLKIN）须由 CKGEN 模块提供 时钟源频率直接影响计数单位与闹钟精度	必须先初始化后才能使用 RTC

3.3.5.5 软件限制/开发须知

重要限制

- AC7840x 专属时钟源限制** - `RTC_CLOCK_STB_LSI32K` 和 `RTC_CLOCK_STB_CLKIN` 仅 AC7840x 系列支持 - 在其他型号上使用将导致编译错误，须用 `#if defined(AC7840X)` 进行条件编译保护
- SetCurrentValue 调用时机约束** - `Rtc_Hal_SetCurrentValue()` 要求 **RTC 未启动**时调用，否则返回错误 - 需要重置计数值时，须先调用 `Rtc_Hal_Stop()`，再设置计数值，最后重新 `Rtc_Hal_Start()`
- DeInit 前置条件** - `Rtc_Hal_DeInit()` 要求 **RTC 已初始化且未处于运行状态**，否则返回 `STATUS_ERROR` - 请确认在调用 `Rtc_Hal_DeInit()` 前已调用 `Rtc_Hal_Stop()`
- 闹钟值与计数单位关系** - `Rtc_Hal_SetAlarm(Value)` 中的 `Value` 以实时寄存器计数单位为基准，该单位由 `Rtc_Hal_Start(Unit)` 中的 `Unit`（预分频值）决定 - 设置闹钟前需根据时钟频率和预分频值计算正确的比较值：
`alarmValue = 目标时间(s) × (时钟频率 / Unit)` - 可通过 `Rtc_Hal_GetWorkFreq()` 获取当前时钟频率辅助计算

3.3.6 MCU

MCU 子系统负责芯片时钟配置（CKGEN）、时钟监控（CMU）、复位管理（RCM）和电源模式管理（SPM）。HAL 层入口头文件：

- 时钟源初始化：`Ckgen_Hal.h`
- 时钟监控：`Cmu_Hal.h` / `Cmu_Hal_Types.h`
- 复位管理：`Rcm_Hal.h`
- 电源模式管理：`Spm_Hal.h`

各型号在时钟树结构和 CMU 实例数上存在差异，详见下表：

特性	AC7840	AC7840E	AC7842	AC7843
CMU 实例数	3	4	3	4
AC7840E 专属 SPLL1	—	✓	—	—
AC7843 专属 ADC SPLL 分频	—	—	—	✓
SPM 待机唤醒源字段	StandbyWakeupSource (单字段)	StandbyWakeupSource0/1 (双字段)	StandbyWakeupSource (单字段)	StandbyWakeupSource (单字段)
SPM StandbyAction 字段	—	—	✓	✓

3.3.6.1 支持的功能

MCU HAL 层提供以下功能，分四个子模块描述：

子模块一：CKGEN（时钟配置）

API	说明
Ckgen_Hal_InitClk	初始化时钟源（HSI/VHSI/HSE/PLL）
Ckgen_Hal_DistributeClk	等待时钟稳定并分发系统/模块时钟
Ckgen_Hal_SetSysClk	设置系统时钟（RUN/VLPR 模式）
Ckgen_Hal_SetPeriphClkDiv	设置外设时钟分频
Ckgen_Hal_SetPeriphClkMux	设置外设时钟源选择（Mux）
Ckgen_Hal_GetClkMux	获取指定时钟当前时钟源
Ckgen_Hal_EnablePeriphClk	使能/禁用外设总线时钟
Ckgen_Hal_GetFreq	获取指定时钟 ID 的频率
Ckgen_Hal_GetClkStatus	获取时钟状态（开关/稳定状态）

子模块二：CMU（时钟监控）

API	说明
Cmu_Hal_Init	初始化指定 CMU 模块，设置监控参数

API	说明
Cmu_Hal_Deinit	反初始化 CMU 模块
Cmu_Hal_Enable	使能/禁用 CMU 模块
Cmu_Hal_ClearErrStatus	清除 CMU 错误状态标志
Cmu_Hal_GetErrInfo	获取 CMU 错误信息

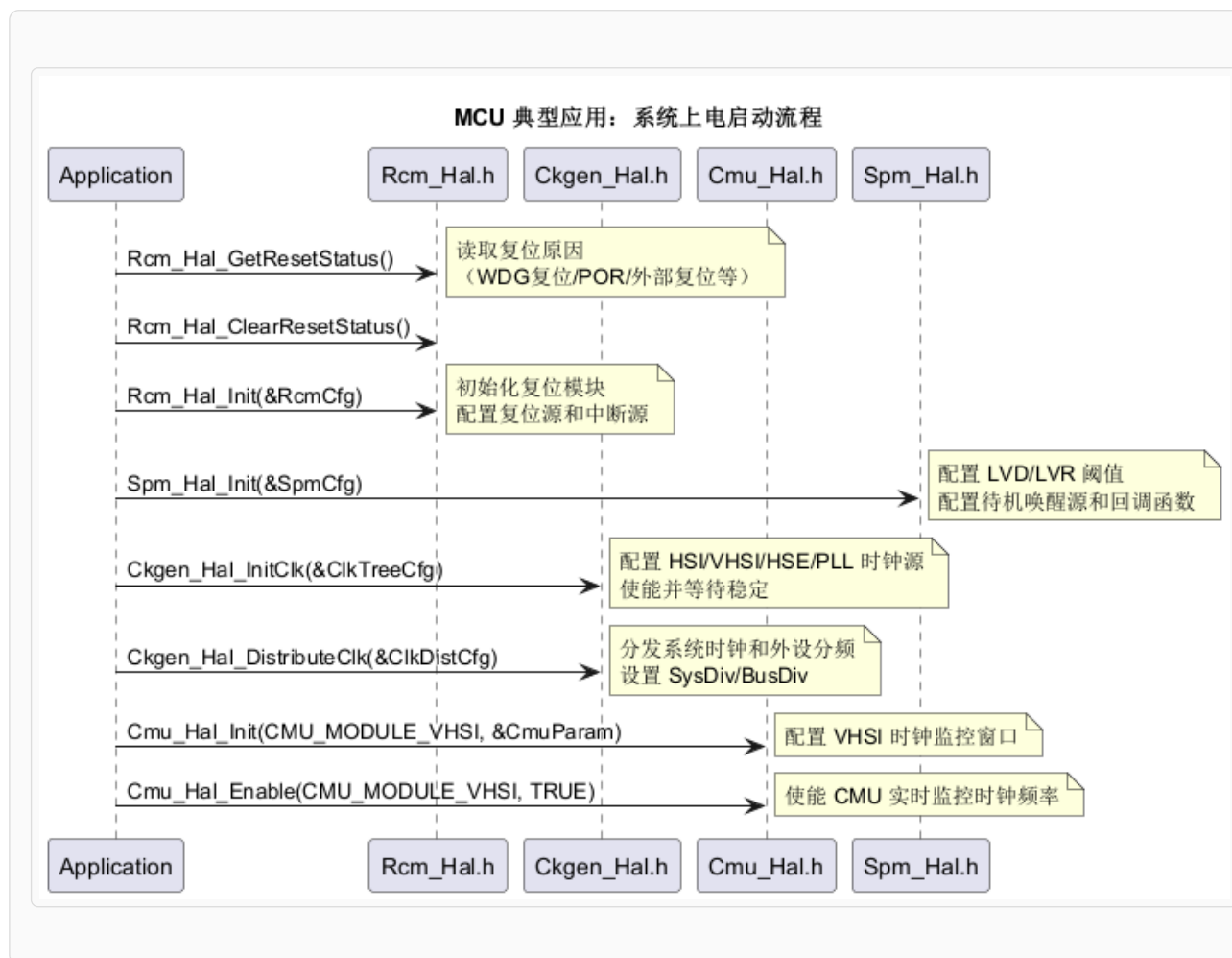
子模块三：RCM（复位管理）

API	说明
Rcm_Hal_Init	初始化复位模块，配置中断源和复位源
Rcm_Hal_GetResetStatus	获取复位状态（区分 POR/WDG/SW/外部复位等）
Rcm_Hal_ClearResetStatus	清除所有复位状态标志
Rcm_Hal_SetResetState	Assert/Deassert 外设复位

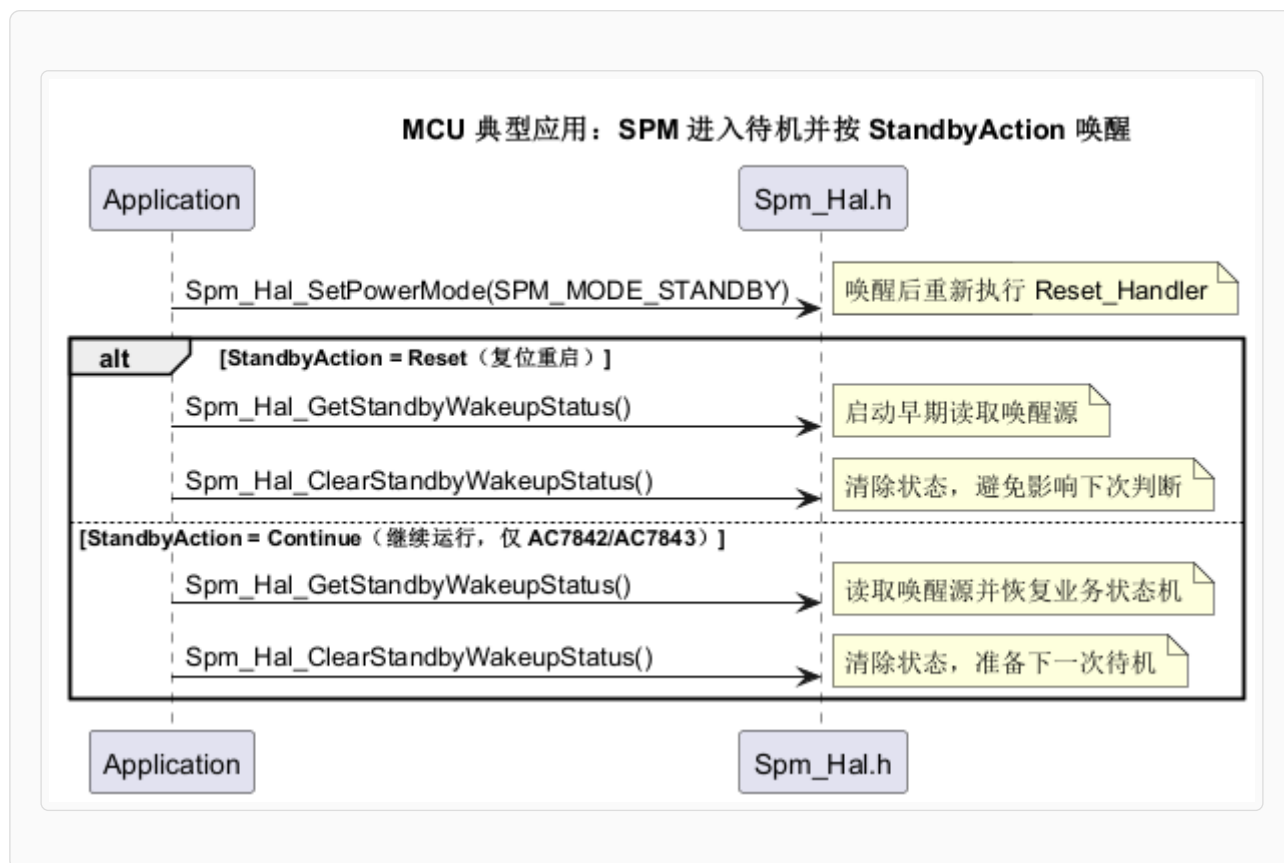
子模块四：SPM（电源模式管理）

API	说明	适用型号
Spm_Hal_Init	初始化 SPM，配置 LVD/LVR 阈值、待机唤醒源及中断回调	全系列
Spm_Hal_GetCurrentPowerMode	读取当前运行模式（RUN / VLPR / STOP1 / STOP2 / VLPS / STANDBY）	全系列
Spm_Hal_SetPowerMode	切换目标运行模式	全系列
Spm_Hal_GetStandbyWakeupStatus	读期待机唤醒状态位（返回 32-bit，各 bit 对应唤醒引脚）	全系列
Spm_Hal_ClearStandbyWakeupStatus	清除待机唤醒状态	全系列

典型应用场景：系统上电启动流程



典型应用场景：进入/退出低功耗模式



继续运行分支建议采用统一恢复流程：待机唤醒后先完成 `SystemInit()`，再根据编译器启动模型进入对应启动入口（如 `main()` / `__iar_program_start()` / `__main()`），并补做唤醒源状态清理与业务上下文恢复，确保系统状态一致。

3.3.6.2 与标准或规范的差异

- 软件实现与芯片定义功能一致

3.3.6.3 静态配置项说明

无特别配置

3.3.6.4 软件集成依赖

- 无

3.3.6.5 软件限制/开发须知

1. 在 `Ckgen_Hal_DistributeClk` 之前调用 `Ckgen_Hal_GetClkStatus` 获取SPLL稳定。
2. **SPM_IRQ_CONTROL_INTERNAL 宏**： `Spm_Hal.h` 中 `SPM_IRQ_CONTROL_INTERNAL`、`LVD_IRQ_CONTROL_INTERNAL`、`STB_IRQ_CONTROL_INTERNAL` 默认均为 `STD_ON`，表示中断由 SDK 内部管理；如需外部管理中斷，須修改对应宏为 `STD_OFF` 并自行注册中断服务函数。

3.3.7 MCL

MCL（微控制器库）子系统提供 DMA、CTU、MPU和 PBR功能。HAL 层入口头文件：

- DMA: `Dma_Hal.h`
- CTU: `Ctu_Hal.h`
- MPU: `Mpu_Hal.h`
- PBR: `Pbr_Hal.h`

各型号差异概览：

特性	AC7840	AC7840E	AC7842	AC7843
DMA 硬件通道数	16	16	16	16
DMA 虚拟通道数	20	20	20	20
DMA Daisy 模式	—	✓	—	—
DMA 半完成中断	—	✓	✓	✓
CTU 实例数	1	1	1	1
MPU 实例数	1	1	1	2
MPU PID 功能	✓	—	✓	✓

3.3.7.1 支持的功能

MCL HAL 层提供以下功能，分四个子模块描述：

子模块一：DMA（直接内存访问）

API	说明
<code>Dma_Hal_Init</code>	初始化 DMA 模块，配置通道与请求源映射
<code>Dma_Hal_Deinit</code>	反初始化 DMA 模块
<code>Dma_Hal_ConfigCh</code>	配置 DMA 通道传输参数（地址、长度、模式等）
<code>Dma_Hal_StartCh</code>	启动 DMA 通道传输
<code>Dma_Hal_StopCh</code>	停止 DMA 通道传输
<code>Dma_Hal_GetChStatus</code>	获取通道传输状态（IDLE/BUSY/ABORT/CONFIG）
<code>Dma_Hal_GetTransBytes</code>	获取通道已传输字节数
<code>Dma_Hal_UpdateChAddr</code>	动态更新通道源/目的地址

API	说明
Dma_Hal_EnableChIrq	使能/禁用通道中断
Dma_Hal_ChReset	复位 DMA 通道
Dma_Hal_GetChIdByReqSrc	通过请求源 ID 获取通道 ID

子模块二：CTU（连接单元）

API	说明
Ctu_Hal_Init	初始化 CTU，批量配置触发源映射
Ctu_Hal_DeInit	反初始化 CTU
Ctu_Hal_SetModuleTriggerSource	设置指定目标模块的触发源
Ctu_Hal_GetModuleTriggerSource	获取指定目标模块当前触发源
Ctu_Hal_EnableSWTrigger	使能/禁用软件触发源
Ctu_Hal_EnableFunc	使能/禁用 CTU 功能

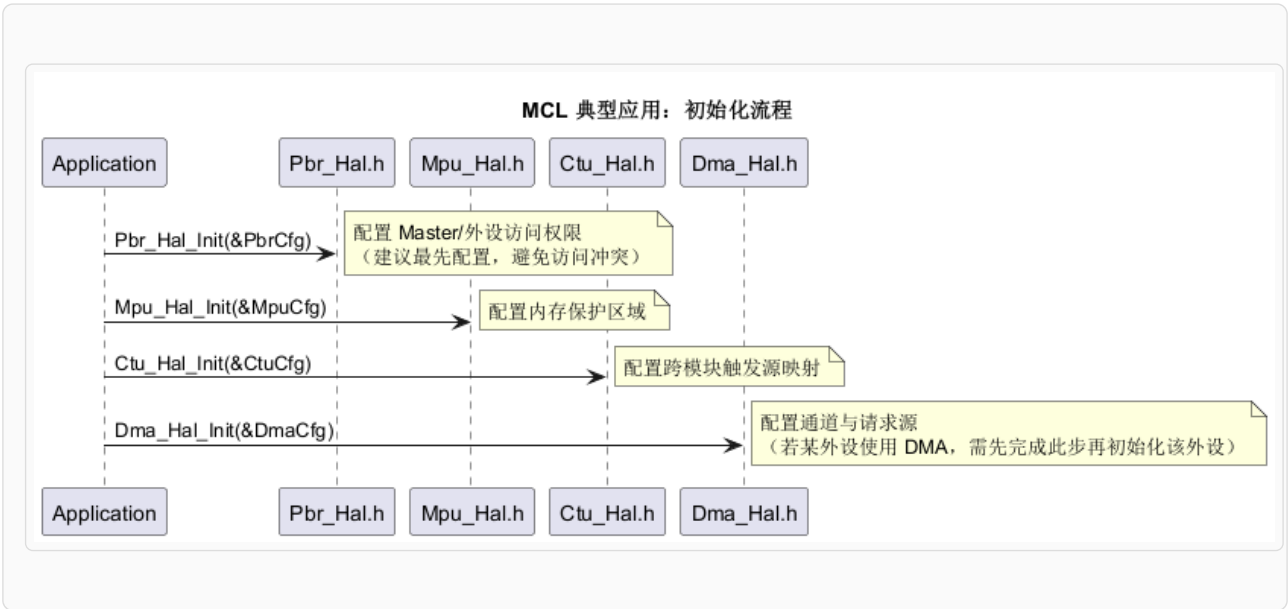
子模块三：MPU（内存保护单元）

API	说明
Mpu_Hal_Init	初始化 MPU，批量配置保护区域
Mpu_Hal_Deinit	反初始化 MPU
Mpu_Hal_SetRegionAddr	设置指定区域的起止地址
Mpu_Hal_SetRegionAttrs	设置指定区域的访问权限
Mpu_Hal_EnableRegion	使能/禁用指定区域保护
Mpu_Hal_GetRegionInfo	获取指定区域的当前配置
Mpu_Hal_SetMasterPid	设置 Master PID [AC7840/AC7842/AC7843 专属]
Mpu_Hal_GetSlaveErrorInfo	获取 Slave 访问错误信息
Mpu_Hal_ClearSlaveError	清除 Slave 访问错误状态

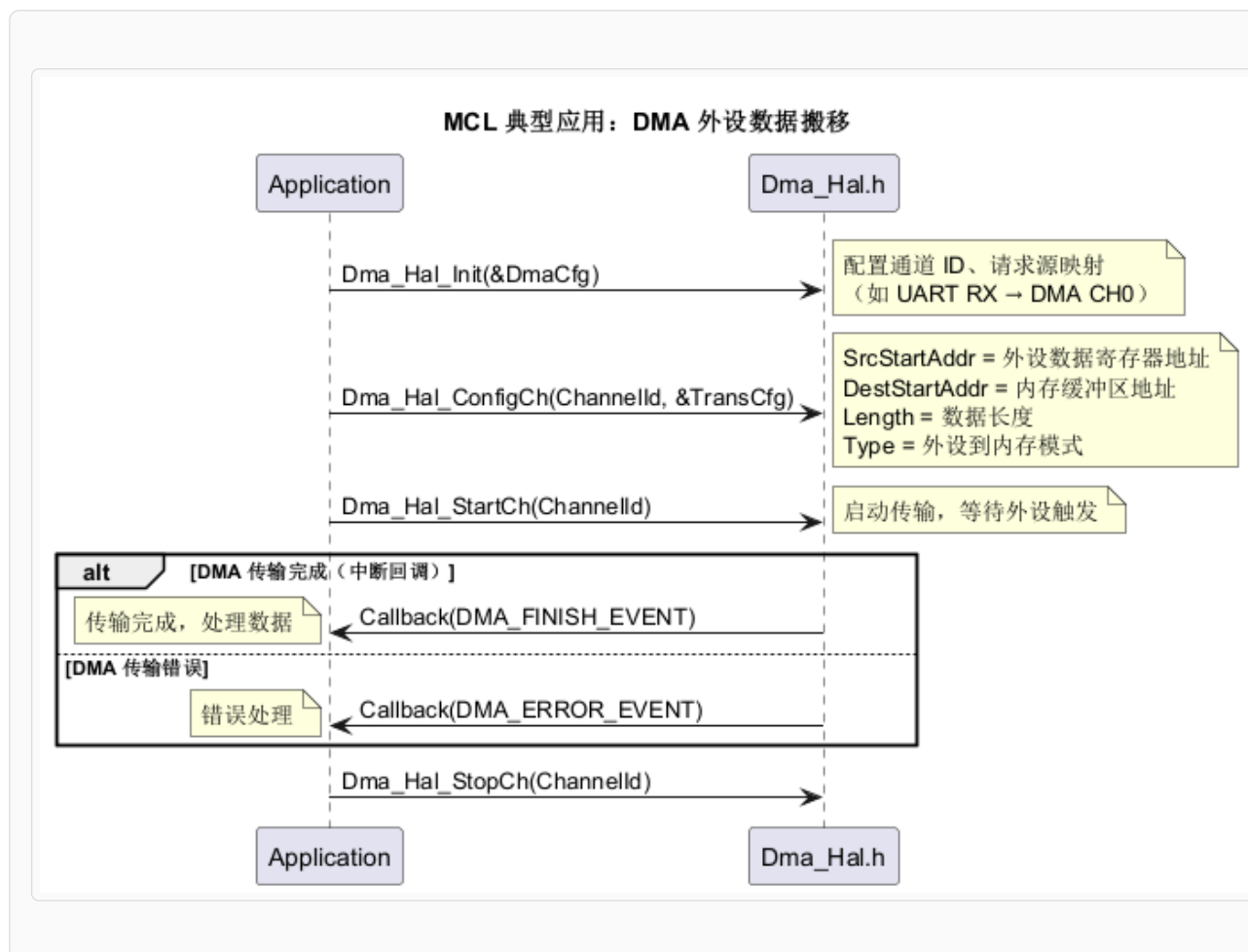
子模块四：PBR

API	说明
Pbr_Hal_Init	初始化 PBR，配置 Master/Peripheral 访问权限

典型应用场景：MCL 初始化流程



典型应用场景：DMA 数据搬移（外设→内存）



3.3.7.2 与标准或规范的差异

软件实现与芯片定义一致。

3.3.7.3 静态配置项说明

无特别配置

3.3.7.4 软件集成依赖

- 无

3.3.7.5 软件限制/开发须知

1. DMA 须在使用 DMA 的外设（如 UART、SPI）初始化之前完成配置。
2. CTU 的触发源配置需在涉及联动触发的外设初始化前完成。
3. MPU 配置需在涉及 MPU 保护的区域初始化前完成。
4. CTU 锁定注意： `Ctu_CfgType.EnLock` 设置为 `TRUE` 时，CTU 配置被锁定，无法再通过 `Ctu_Hal_SetModuleTriggerSource` 修改触发源，需复位后才能重新配置。

5. EEP_MPU_PROTECTION 打开宏，默认会占用一个 MPU 区域，用户需根据实际需求调整或禁用。
6. 7843 DMA 使用cache，需要注意一致性问题，确保DMA传输的内存区域被正确配置为 Non-cacheable 或在传输前后进行 Cache 维护操作。
7. DMA 错误处理，如果是调试阶段，建议在回调函数中打印错误信息，方便定位问题。
8. 7840 DMA使用SRAM必须都在SRAML区使用，7840E、7842、7843 DMA使用SRAM没有限制但是都推荐SRAML区使用，代码段放在SRAMU区。
9. DMA 配置的地址以及地址偏移都必须是传输数据宽带（1Byte/2Byte/4Byte）的倍数，否则会导致传输错误。

3.3.8 OSIF

3.3.8.1 支持的功能

- **基础 OS 抽象接口：**初始化/反初始化 `OsIf_Init` / `OsIf_Deinit`
- **计数器与延时：**
 - `OsIf_GetCounter`：获取系统启动以来的 SysTick 计数值
 - `OsIf_GetElapsed`：获取相对于引用值的已用 SysTick 数，并更新引用值
 - `OsIf_MicrosToTicks`：将微秒值转换为 SysTick 节拍数
 - `OsIf_UDelay`：微秒级延时（忙等待）
- **核心信息获取：** `OsIf_GetCoreId`，返回当前核心 ID（默认返回 0）
- **临界区保护：** `OsIf_Irq.h` 中提供 `EnterCritical` / `ExitCritical` 用于中断嵌套控制
- **平台支持：**通过 `OS_PLATFORM` 宏选择 `OSIF_BAREMETAL`（裸机）或 `OSIF_OS`（RTOS）实现分支

3.3.8.2 与标准或规范的差异

OsIf 为HAL对操作系统或裸机环境的抽象，分为 `OSIF_BAREMETAL` 与 `OSIF_OS` 两类实现：

- **裸机实现：**依赖 ARM Cortex-M SysTick 定时器，提供最基础的计时与延时能力
- **RTOS 实现：**由操作系统平台提供等效接口实现，SysTick 通常由 OS 管理

接口为项目的HAL层代码移植提供统一入口，调用方无需关心底层是裸机还是 RTOS。

3.3.8.3 静态配置项说明

无

3.3.8.4 软件集成依赖

依赖模块	说明
CKGEN	用于获取核心频率 (Ckgen_Hal_GetFreq) , OsIf_Init 会读取并配置 SysTick 计数周期

3.3.8.5 软件限制/开发须知

- **SysTick 独占**: 裸机实现下 SysTick 被 OSIF 独占使用, 若 RTOS 也使用 SysTick, 则必须选择 OSIF_OS 模式并由 RTOS 提供 OsIf_* 实现
- **OsIf_UDelay 为忙等待**: OsIf_UDelay 在延时期间持续占用 CPU, 期间中断屏蔽会导致外部事件响应延迟, 请勿在严格实时路径使用长延时
- **精度限制**: 系统时钟较低或中断屏蔽期间, OsIf_UDelay 和 OsIf_GetElapsed 的精度会下降

3.3.9 WDG

特性	AC7840	AC7840E	AC7842	AC7843
WDG 实例数	1	1	1	1
多时钟源支持 (LSI/ HSI/HSE/Bus)	✓	✓	✓	✓
窗口模式	✓	✓	✓	✓
调试模式控制	✓	✓	✓	✓
低功耗模式控制	✓	✓	✓	✓
超时中断 / 复位两种处理方式	✓	✓	✓	✓
快速测试模式	✓	✓	✓	✓

3.3.9.1 支持的功能

AC784xx WDG HAL 层提供以下功能:

1. 基础看门狗功能

- **初始化与反初始化**
 - 配置时钟源 (LSI / HSI / HSE / Bus Clock)

- 配置超时值 (`TimeoutValue` , 单位: ms)
- 配置窗口值 (`WindowValue` , 单位: ms, 窗口模式启用时有效)
- 通过 `Config` 字段按位组合功能选项:
 - `WDG_CONFIG_PRESCALER_EN` : 使能分频器
 - `WDG_CONFIG_DBG_EN` : 调试模式下保持看门狗运行
 - `WDG_CONFIG_LP_EN` : 低功耗模式下保持看门狗运行
 - `WDG_CONFIG_WIN_EN` : 使能窗口模式
 - `WDG_CONFIG_FAST_TEST` : 使能快速测试模式

• 喂狗操作

- 调用 `Wdg_Hal_Feed()` 重置计数器, 防止超时触发复位

• 时钟频率查询

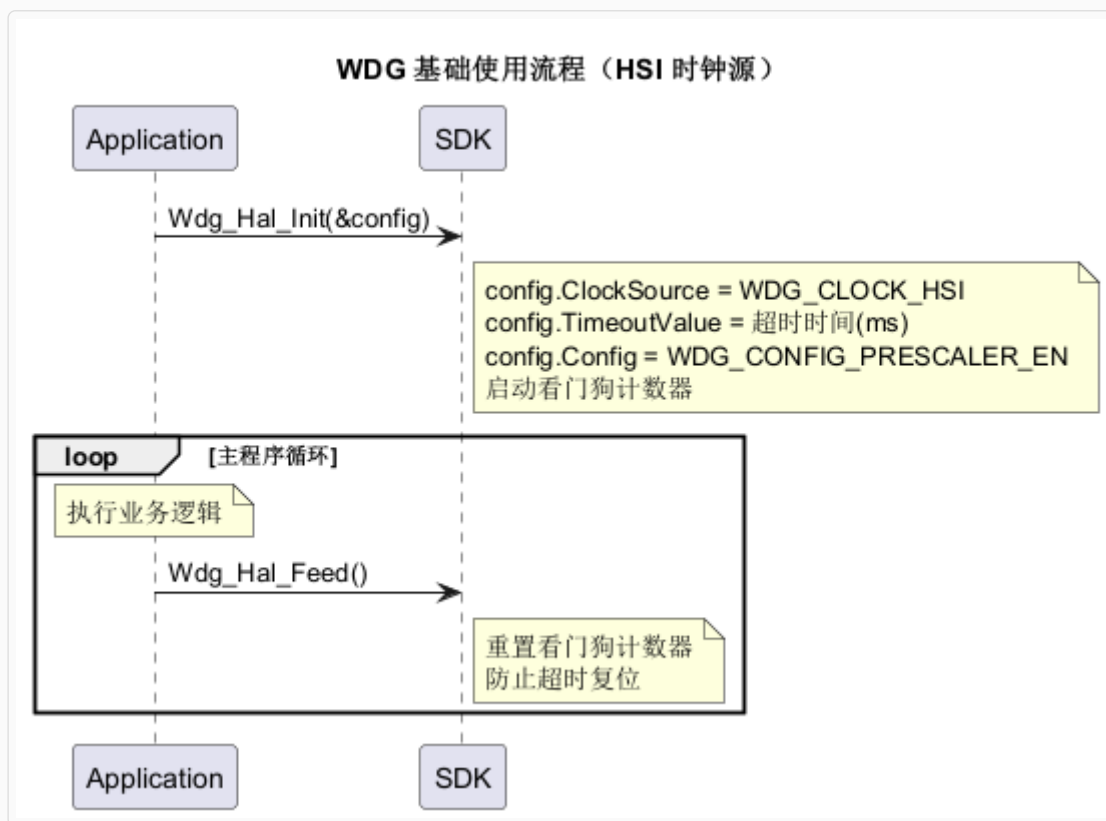
- 调用 `Wdg_Hal_GetFrequency(ClockSource)` 查询指定时钟源频率 (单位: KHz)

• 中断回调

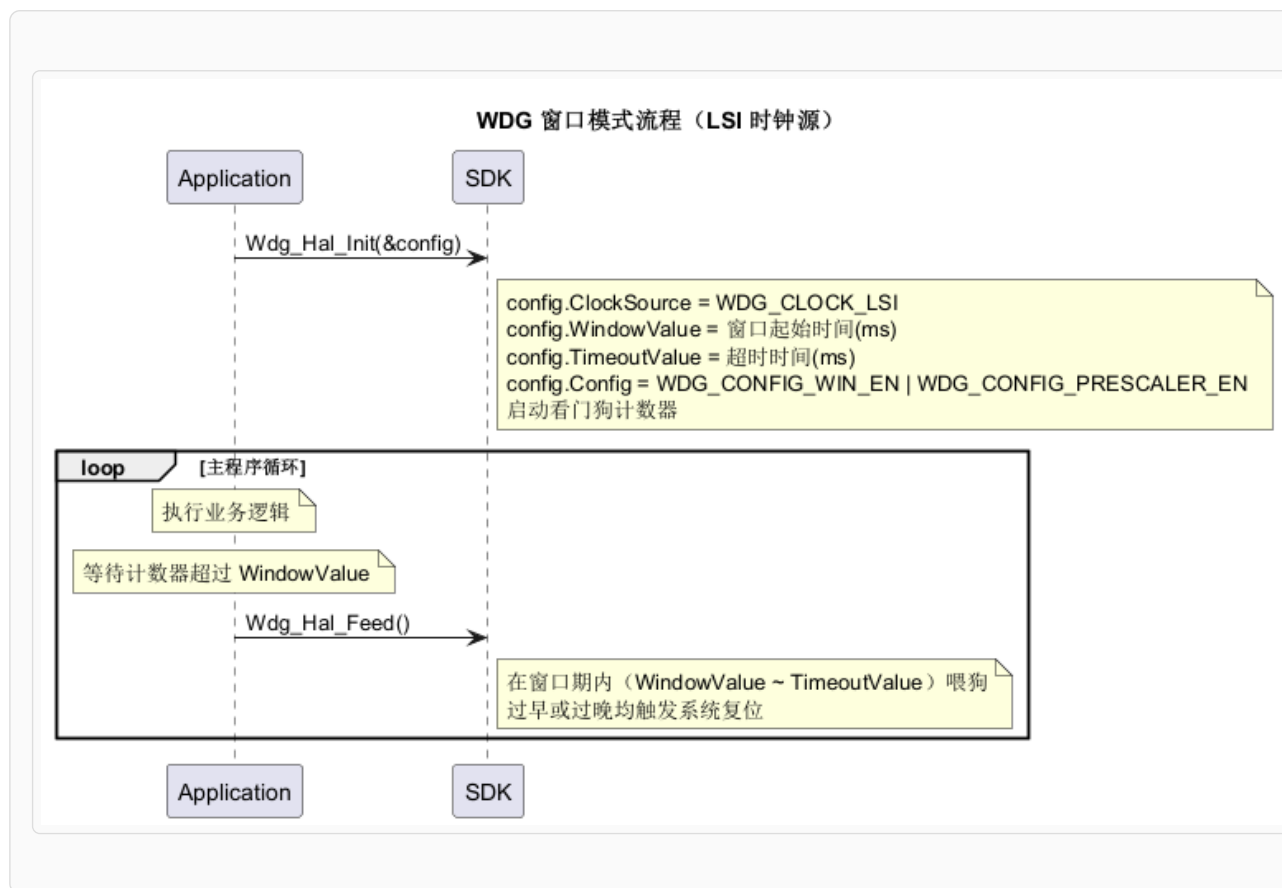
- 通过 `Wdg_Hal_InstallCallback()` 注册超时前中断回调
- 通过 `Wdg_Hal_EnableInterrupt(TRUE)` 使能看门狗中断

典型使用流程时序图

1. 基础看门狗流程 (HSI 时钟源)



2. 窗口模式看门狗流程（LSI 时钟源）



3.3.9.2 与标准或规范的差异

WDG 的软件实现与芯片手册功能一致。

3.3.9.3 静态配置项说明

无特别配置

3.3.9.4 软件集成依赖

依赖模块

WDG HAL 层功能依赖以下模块正确配置：

依赖模块	依赖说明	初始化顺序要求
CKGEN	WDG 时钟源 (LSI / HSI / HSE / Bus Clock) 须由 CKGEN 模块提供 时钟频率直接影响超时值与窗口值的实际时间计算	必须先初始化后才能使用 WDG

3.3.9.5 软件限制/开发须知

重要限制

- 窗口模式喂狗时机约束** - 启用窗口模式（`WDG_CONFIG_WIN_EN`）后，喂狗必须在规定的窗口内执行 - **过早喂狗（早于 `WindowValue`）或过晚喂狗（超过 `TimeoutValue`）** 均会触发系统复位 - 建议在系统设计阶段根据主循环周期合理配置 `WindowValue` 与 `TimeoutValue`，并留有余量
- 调试模式行为** - 使用 `WDG_CONFIG_DBG_EN` 时，调试器暂停期间看门狗计数器**仍继续运行** - 调试时若未及时喂狗，将触发系统复位，影响调试体验 - 调试期间建议设置较大的超时值，或不配置 `WDG_CONFIG_DBG_EN`
- 低功耗模式行为** - 使用 `WDG_CONFIG_LP_EN` 时，低功耗模式下看门狗计数器**仍继续运行** - 进入低功耗模式前须确认喂狗策略：主循环暂停期间仍需在超时前完成喂狗，或通过唤醒中断触发喂狗
- 时钟源选择对超时精度的影响** - **LSI（内部低速时钟）**：适用于低功耗场景，时钟精度较低，超时时间存在偏差 - **HSI（内部高速时钟）**：时钟精度较高，适用于对超时精度有要求的场景 - **HSE（外部高速时钟）**：需外部晶振，精度最高 - **Bus Clock**：随总线频率变化，配置超时值时须以当前总线频率为基准 - 可调用 `Wdg_Hal_GetFrequency(ClockSource)` 查询当前时钟源频率（单位：KHz），用于验证超时值计算
- 反初始化（DeInit）的使用限制** - `Wdg_Hal_DeInit()` 可禁用看门狗；禁用后如需重新使能，须重新调用 `Wdg_Hal_Init()` - 在安全关键系统中，禁用看门狗须谨慎评估功能安全风险

3.4 Security

Security 子系统包含各型号的硬件安全加速模块。AC784xx 系列四款产品的安全模块各有不同：AC7840 和 AC7842 内置 **CSE**（符合 SHE 规范），AC7840E 内置 **HSM-Lite**（轻量增强安全模块），AC7843 内置 **HSM**（增强硬件安全模块）。同一产品上只存在一种安全模块，各模块不可混用。

产品型号	安全模块	规范基础	HAL 接口前缀	头文件
AC7840	CSE	AUTOSAR SHE V1.1	<code>CSE_Hal_</code>	<code>hal/include/Cse_Hal.h</code>
AC7840E	HSM-Lite	轻量 HSM（直接内嵌主核）	<code>HSM_Hal_</code>	<code>hal/include/Hsm_Hal.h</code>

产品型号	安全模块	规范基础	HAL 接口前缀	头文件
AC7842	CSE	AUTOSAR SHE V1.1	CSE_Hal_	hal/include/ Cse_Hal.h
AC7843	HSM	Evita Full	HSM_Hal_	hal/include/ Hsm_Hal.h

3.4.1 HSM

本章介绍 AC784xx 系列各型号安全模块（CSE / HSM-Lite / HSM）的软件接口与集成方法，以四型号横向对比为基础，帮助用户在不同产品间做出正确的技术选型和迁移决策。

3.4.1.1 支持的功能

各型号安全能力总览

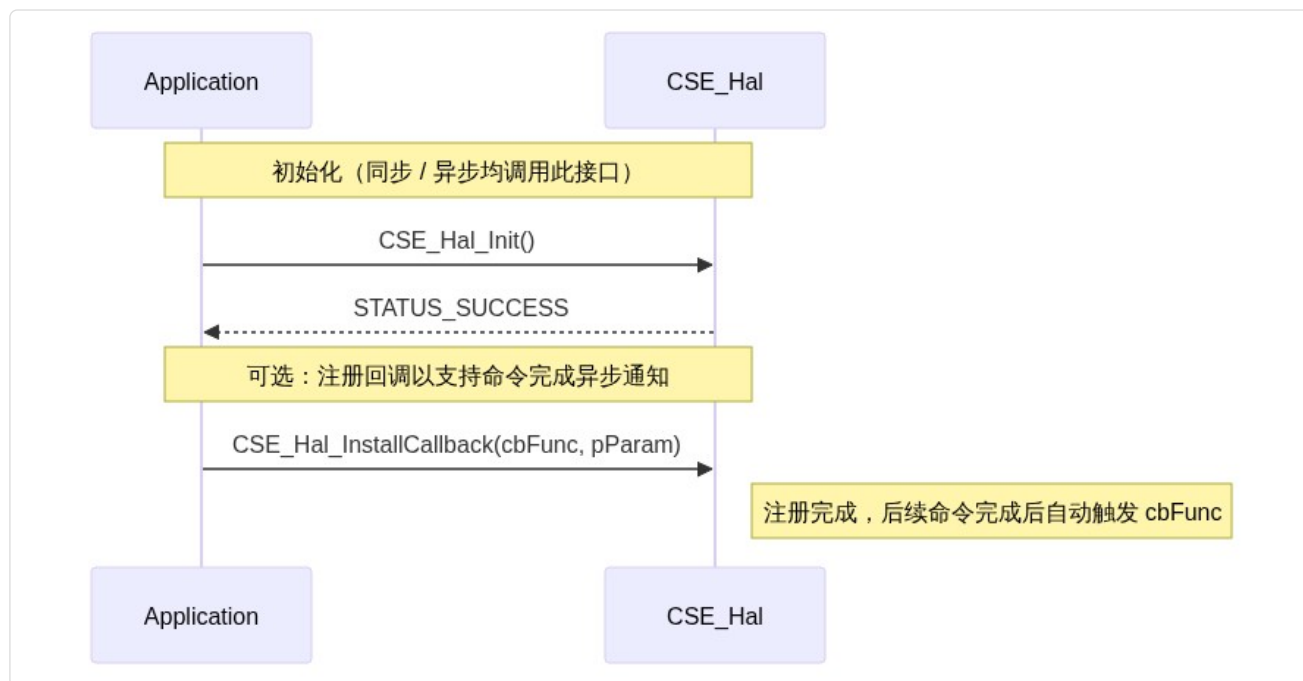
功能维度	AC7840	AC7840E	AC7842	AC7843
安全模块	CSE（SHE 规范）	HSM-Lite（内嵌主核）	CSE（SHE 规范）	HSM（增强版，独立安全核）
对称算法	AES-128 ECB/CBC	AES-128/AES-256（可选）、SM4；ECB/CBC/CFB8/CFB128/OFB/CTR	AES-128 ECB/CBC	AES-128/256、SM4；ECB/CBC/CFB/OFB/CTR/GCM/CCM
消息认证（MAC）	AES-128-CMAC	AES-CMAC、SM4-CMAC（生成/验证）	AES-128-CMAC	AES CMAC、SM4 CMAC、HMAC（参考hash算法支持类型）
哈希算法	不支持	SHA256（软件实现，需开启HSM_SUPPORT_SW_RSA，受限）	不支持	SM3、SHA224/256/384/512/SHA3_224/SHA3_256/SHA3_384/SHA3_512
非对称算法	不支持	RSA-2048（软件实现，需开启HSM_SUPPORT_SW_RSA，受限）	不支持	RSA-1024/2048、ECC（SECP256R1/384R1）、SM2、DH
国密算法	不支持	SM4（对称加解密及CMAC）；无 SM2 / SM3	不支持	SM2 / SM3 / SM4（完整国密套件）
随机数生成	☑ PRNG（128 位）	☑ TRNG	☑ PRNG（128 位）	

功能维度	AC7840	AC7840E	AC7842	AC7843
				<input checked="" type="checkbox"/> TRNG/PRNG; AES-CTR DRBG / SM4-CTR DRBG
密钥管理方式	SHE M1~M5 协议	Flash 密钥槽明文写入; RAM 密钥临时存储	SHE M1~M5 协议	Flash/RAM slot; 支持明文/加密/派生/DH 协商导入
密钥槽数量	SECRET_KEY + MASTER_ECU + BOOT 相关 + KEY_1~17 + RAM_KEY	Flash: 对称×10 (HSM_FLASH_KEY_SYM_0~9) + RSA×1; RAM: 对称×1 (HSM_RAM_KEY_SYM_0)	同 AC7840	Flash: 对称×10、 ECC×10、SM2×10、 RSA×4; RAM: 对称×10 + 非对称各若干; 密钥存储空间有限, 密钥槽数量均为最大数量,具体数量根据密钥槽类型不同时, 密钥槽数量不同
OTP 管理	不支持	<input checked="" type="checkbox"/> (10 个 OTP 密钥槽)	不支持	<input checked="" type="checkbox"/> (17 个 OTP 密钥槽)
安全启动	SHE 安全启动 (CMAC 验签)	<input checked="" type="checkbox"/> (AES-CMAC / SM4-CMAC 验签)	SHE 安全启动 (CMAC 验签)	可配算法 (RSA-2048 / SM2 / SM4 CMAC/ AES-CMAC)
安全升级/镜像验证	不支持	<input checked="" type="checkbox"/> (HSM_Hal_FirmVerifyStart / HSM_Hal_FirmVerifyUpdate)	不支持	<input checked="" type="checkbox"/> (START/ UPDATE/FINISH 分段流式处理)
生命周期管理	不支持	不支持	不支持	<input checked="" type="checkbox"/> 7 阶段 (unnormal→test→ develop→manufacture→user→debug→ destroy)
调试认证	不支持	<input checked="" type="checkbox"/> (Challenge- Response, 基于 OTP 认证密钥)	不支持	<input checked="" type="checkbox"/> (SM2、 ECDSA(SHA256))
多任务保护	应用层自行实现互斥	应用层自行实现互斥	应用层自行实现互斥	HSM_Hal_Lock / HSM_Hal_Unlock

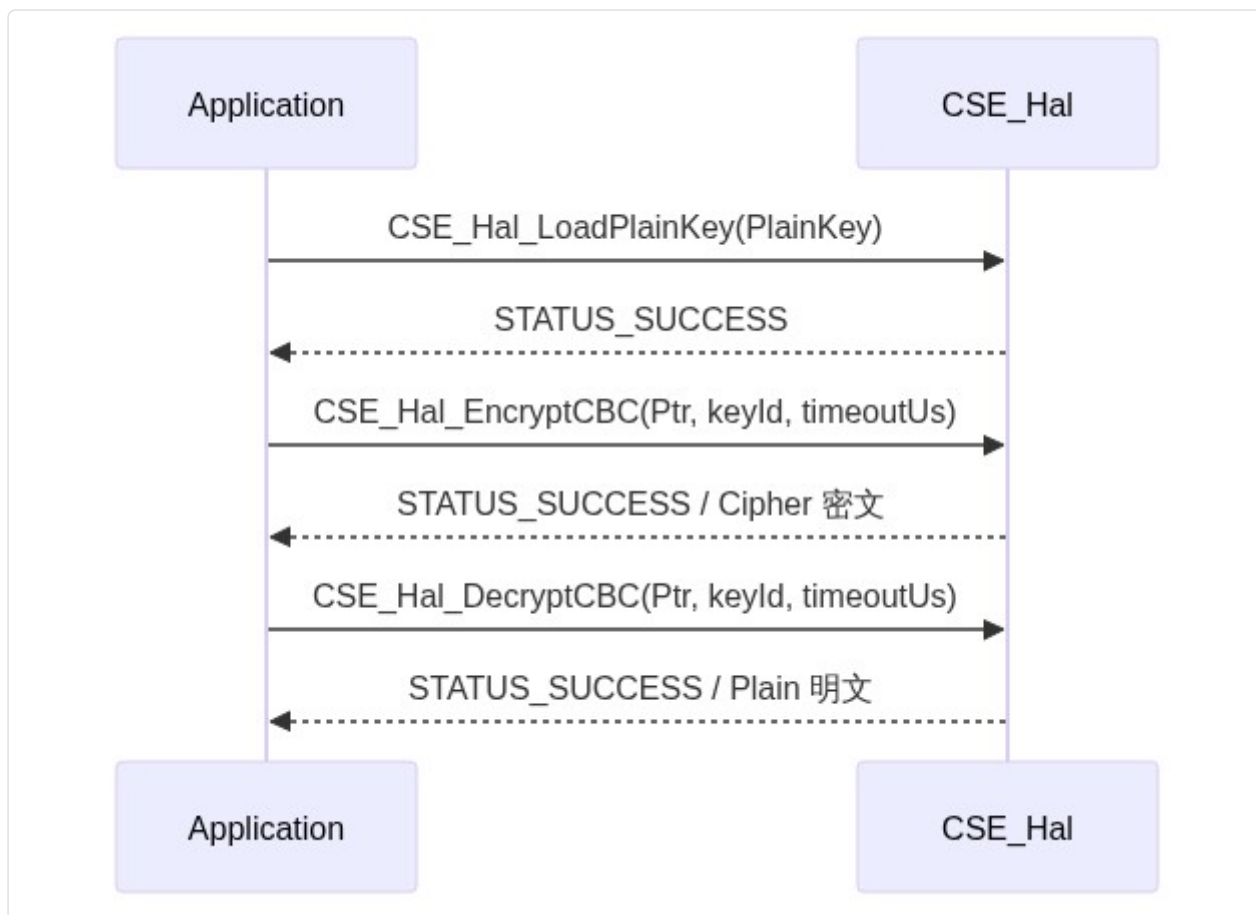
AC7840 / AC7842 CSE 应用场景时序图

AC7840 与 AC7842 共享相同的 CSE HAL 接口，通过编译宏 `AC7840X` / `AC7842X` 自动适配。以下时序图展示了主要功能的典型调用流程，时序图仅包含 Application 与 HAL 层。

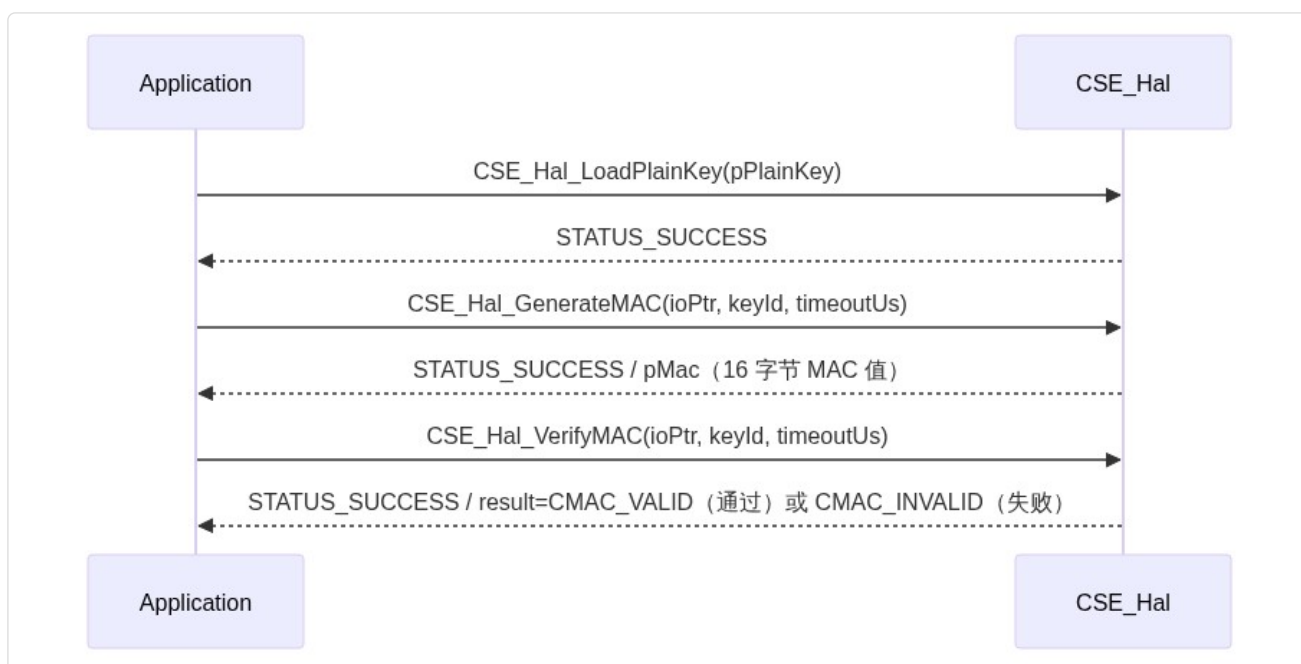
场景 1：初始化 (Init)



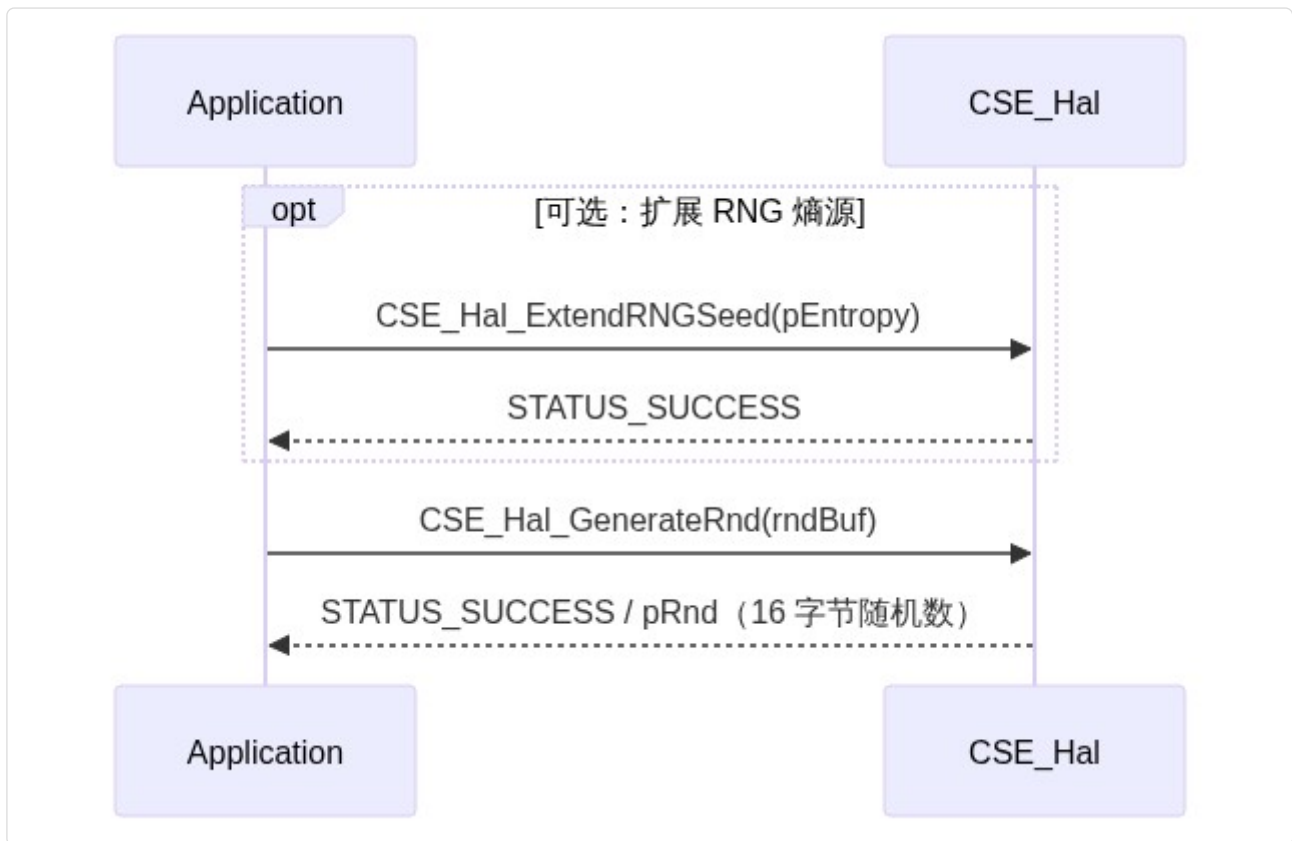
场景 2：AES-128 CBC 加解密



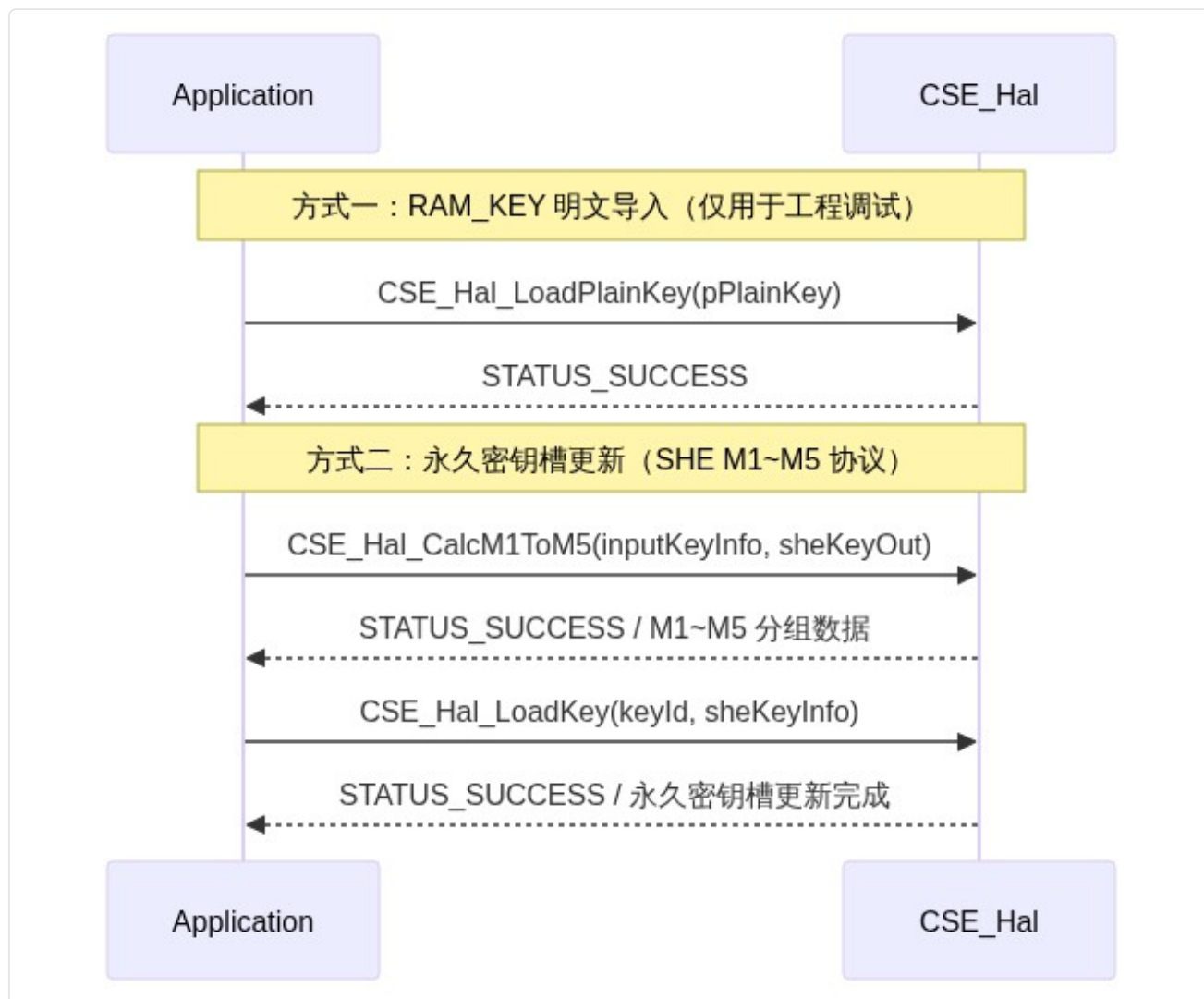
场景 3: AES-128-CMAC 生成与验证



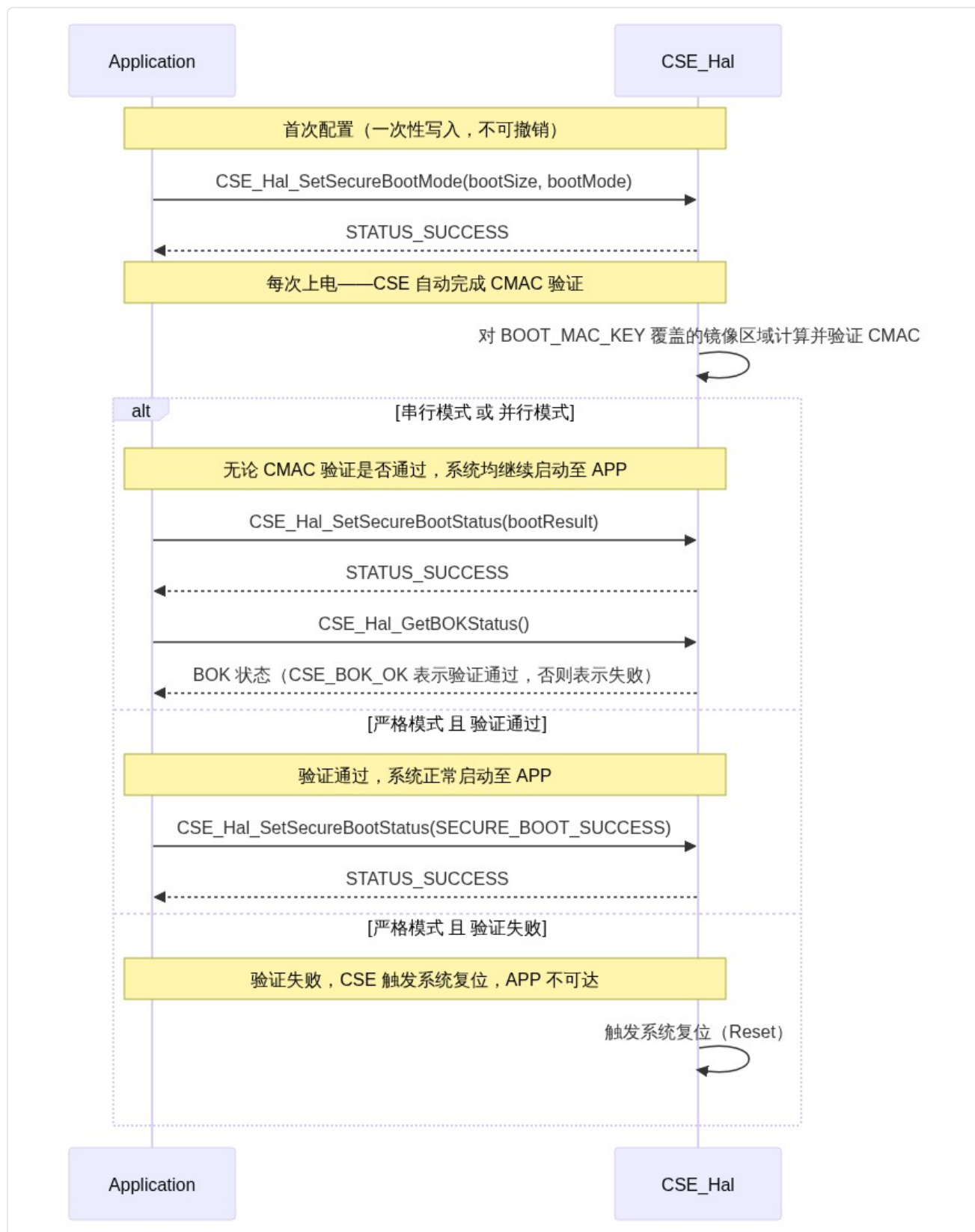
场景 4: 随机数生成 (RNG)



场景 5：密钥管理（明文导入 / M1~M5 协议更新）



场景 6：安全启动（Secure Boot）



各启动模式行为说明：

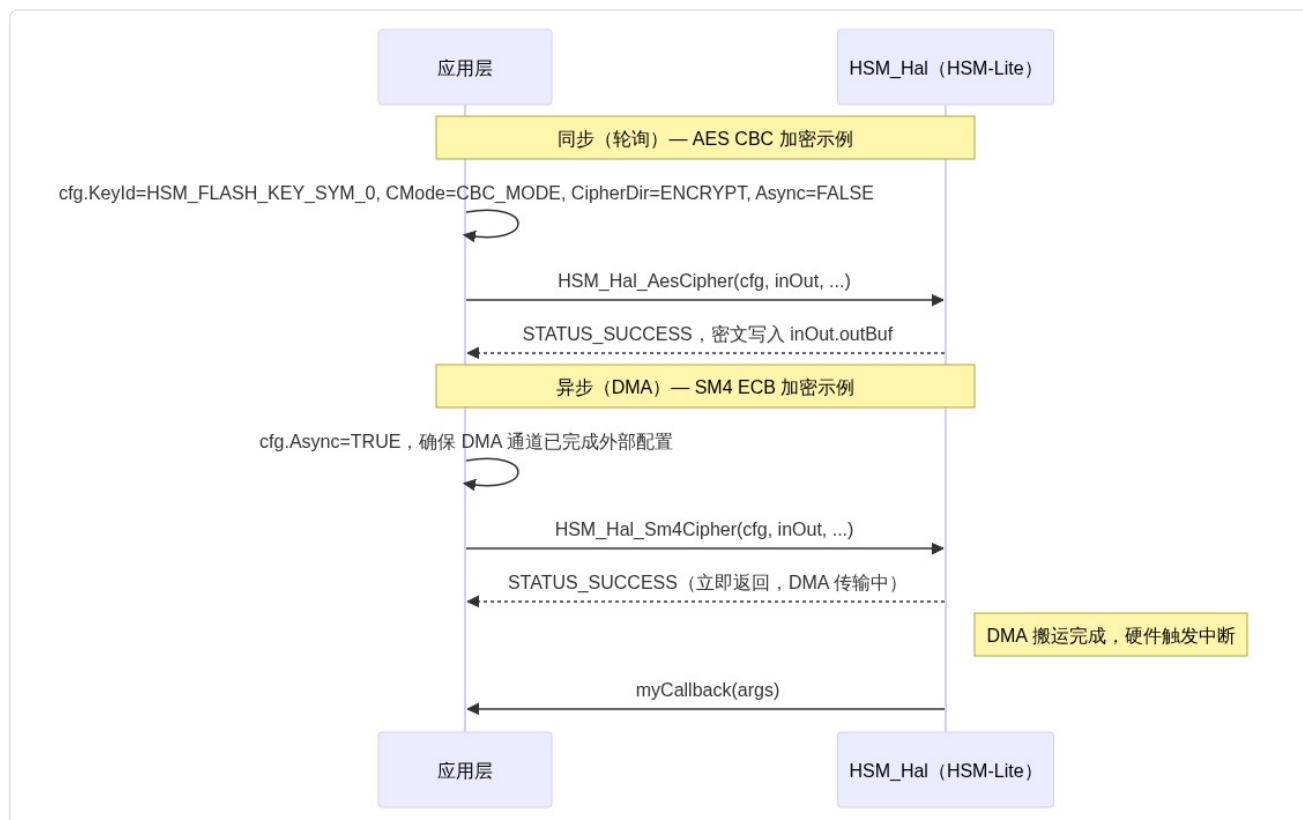
启动模式	CMAC 验证失败时的行为	如何判断验证结果
串行模式 (Sequential)	系统仍然启动至 APP，不会复位	在 APP 中调用 <code>CSE_Hal_GetBOKStatus()</code> 读取 BOK 状态位
并行模式 (Parallel)	系统仍然启动至 APP，不会复位	在 APP 中调用 <code>CSE_Hal_GetBOKStatus()</code> 读取 BOK 状态位
严格模式 (Strict)	CSE 触发系统复位，系统无法启动至 APP	验证失败时 APP 不可达，无法通过接口查询

注意： - 串行/并行模式下，即使安全启动验证失败，代码仍会执行至 APP。安全敏感逻辑须在 APP 入口处主动调用 `CSE_Hal_GetBOKStatus()` 检查结果，并在返回值非 `CSE_BOK_OK` 时执行相应的故障处理。 - 严格模式下，验证失败会立即复位，APP 层无法感知失败事件，适用于对安全要求最高的场景，但需确保 `BOOT_MAC_KEY` 与镜像始终保持一致，否则系统将无法正常启动。

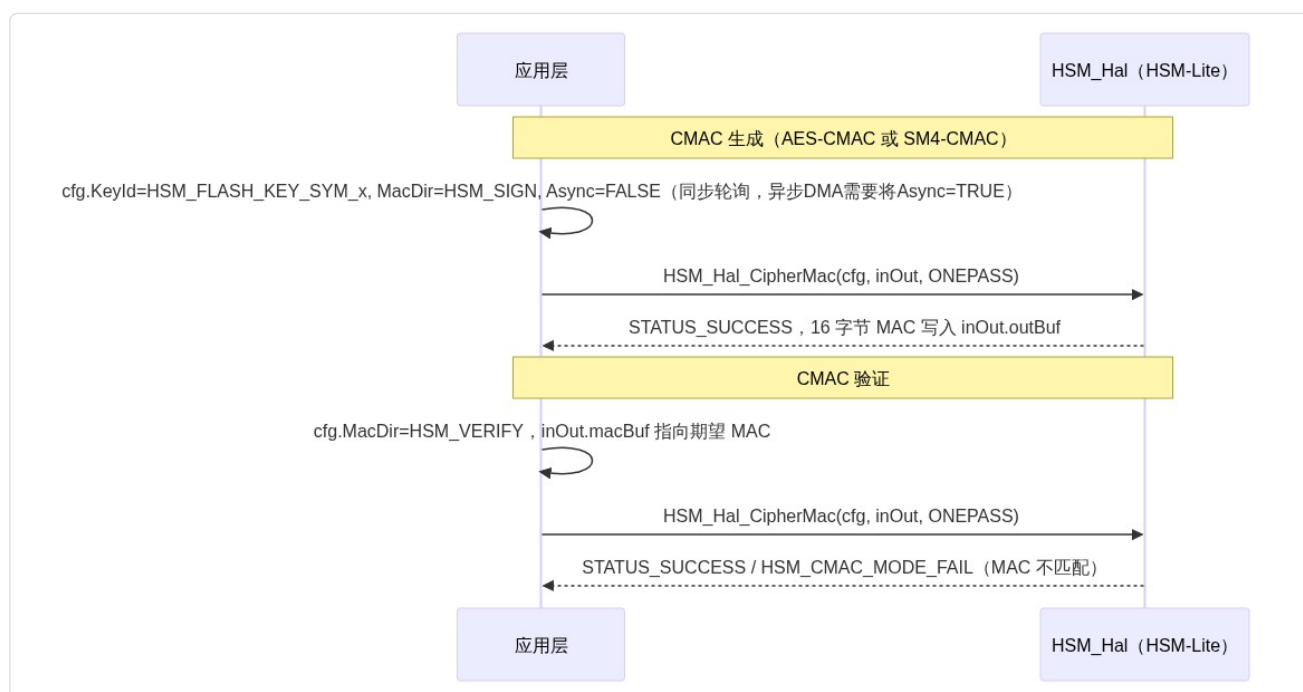
AC7840E HSM-Lite 接口应用场景时序图

AC7840E HSM-Lite 与 AC7843 HSM 共用 `HSM_Hal_` 接口前缀和 `hal/include/Hsm_Hal.h` 头文件，但 API 集合不同（无 Hash / ECC / OTP Write 等接口），且密钥 ID 不同：Flash 对称密钥 ID 为 `HSM_FLASH_KEY_SYM_0` (12) ~ `HSM_FLASH_KEY_SYM_9` (21)，RAM 密钥 `HSM_RAM_KEY_SYM_0` (22)。AC7840E 与 AC7843 **不可混用**。

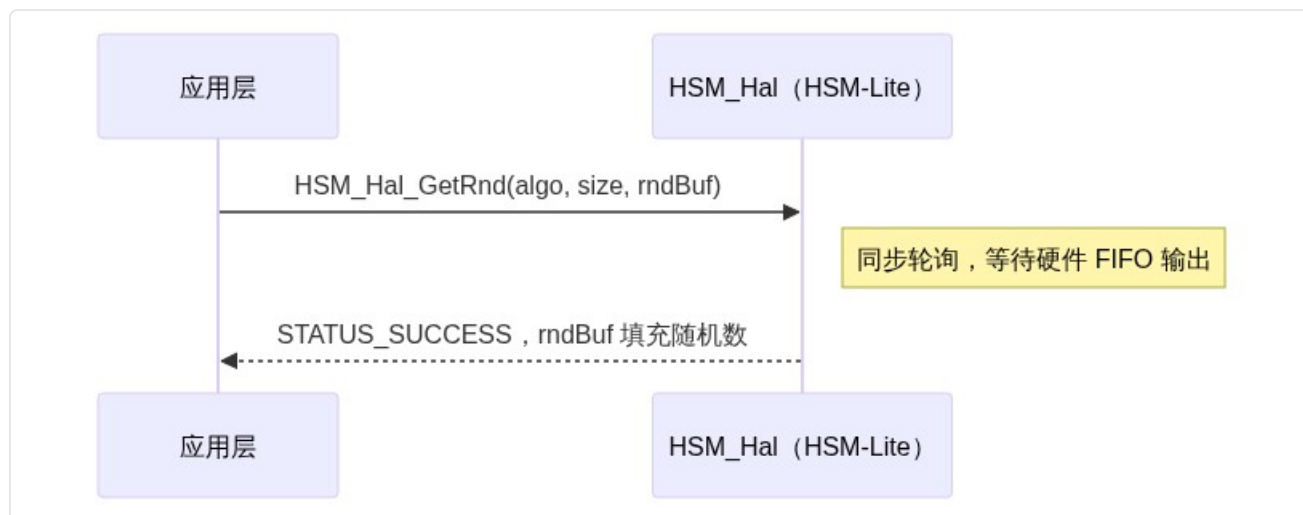
场景 1：对称加解密（AES / SM4，同步 / 异步）



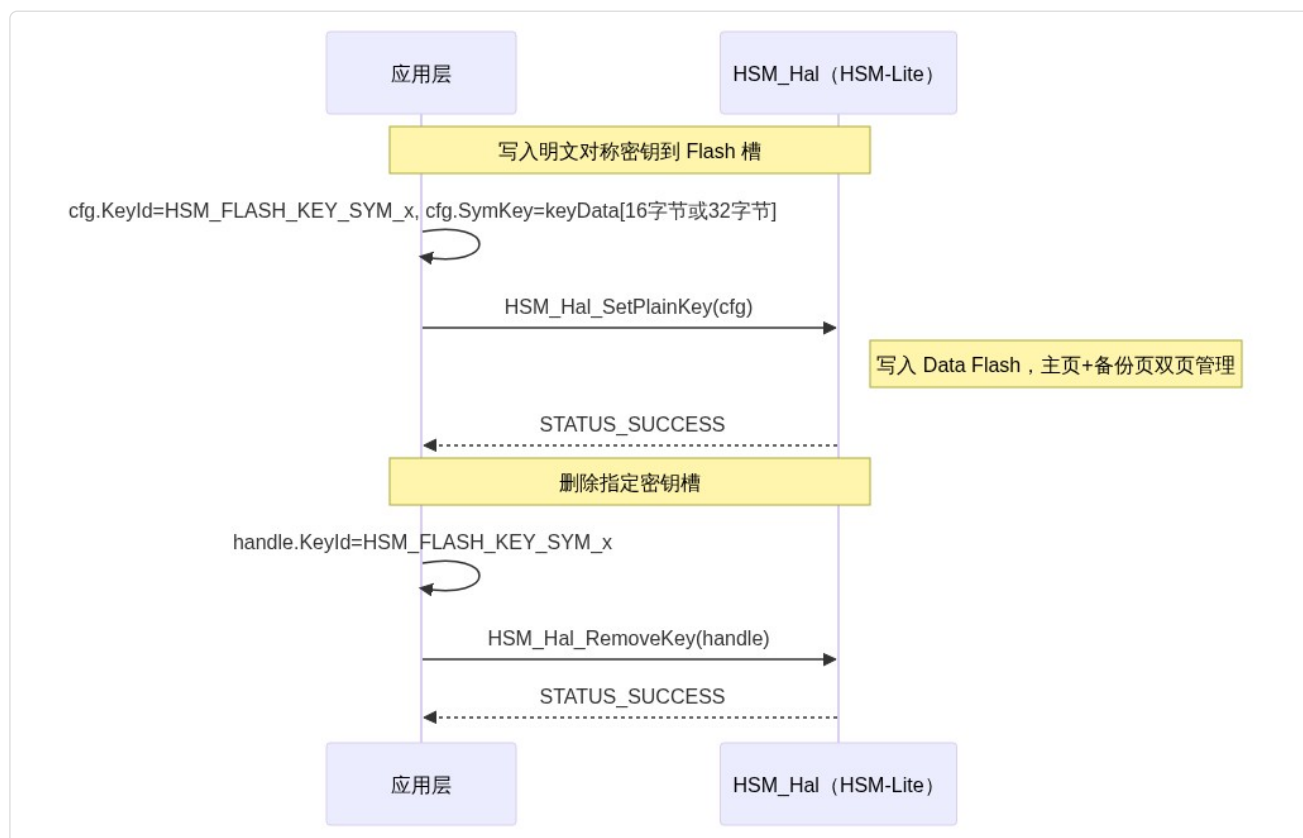
场景 2: CMAC 生成与验证 (同步 / 异步)



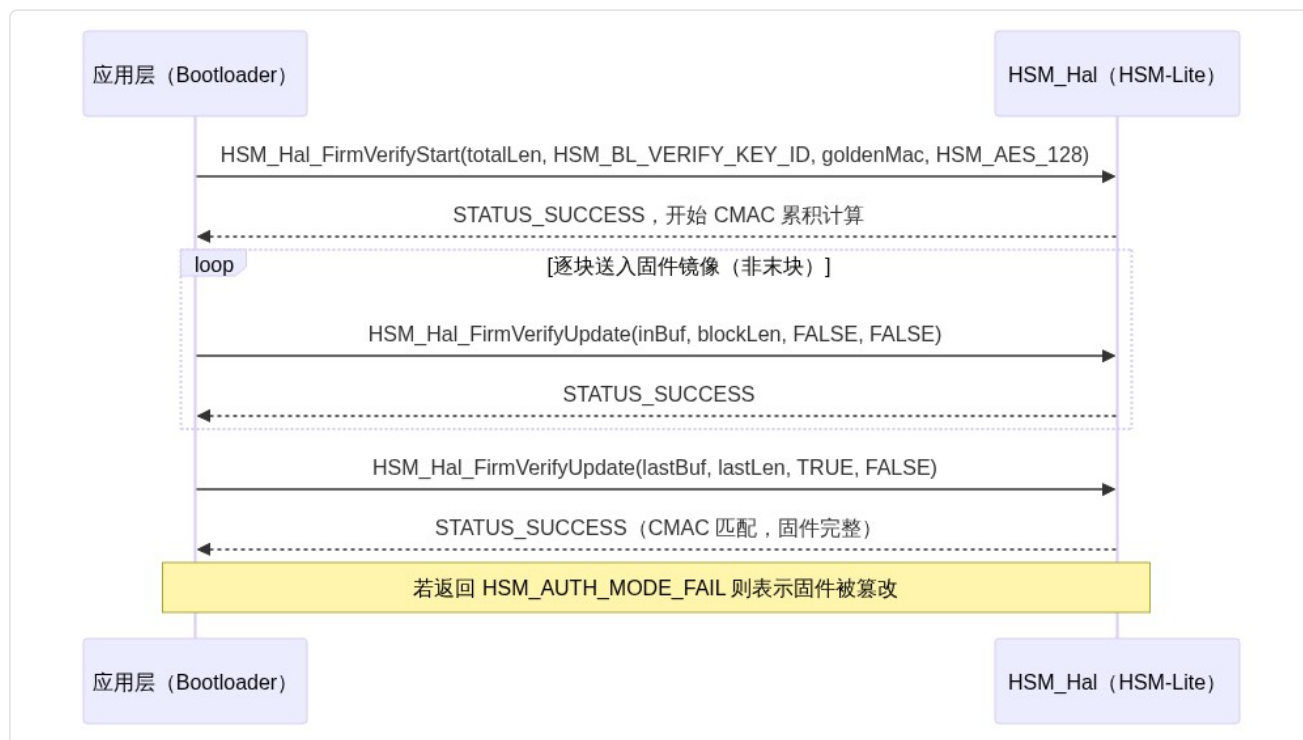
场景 3: 随机数生成



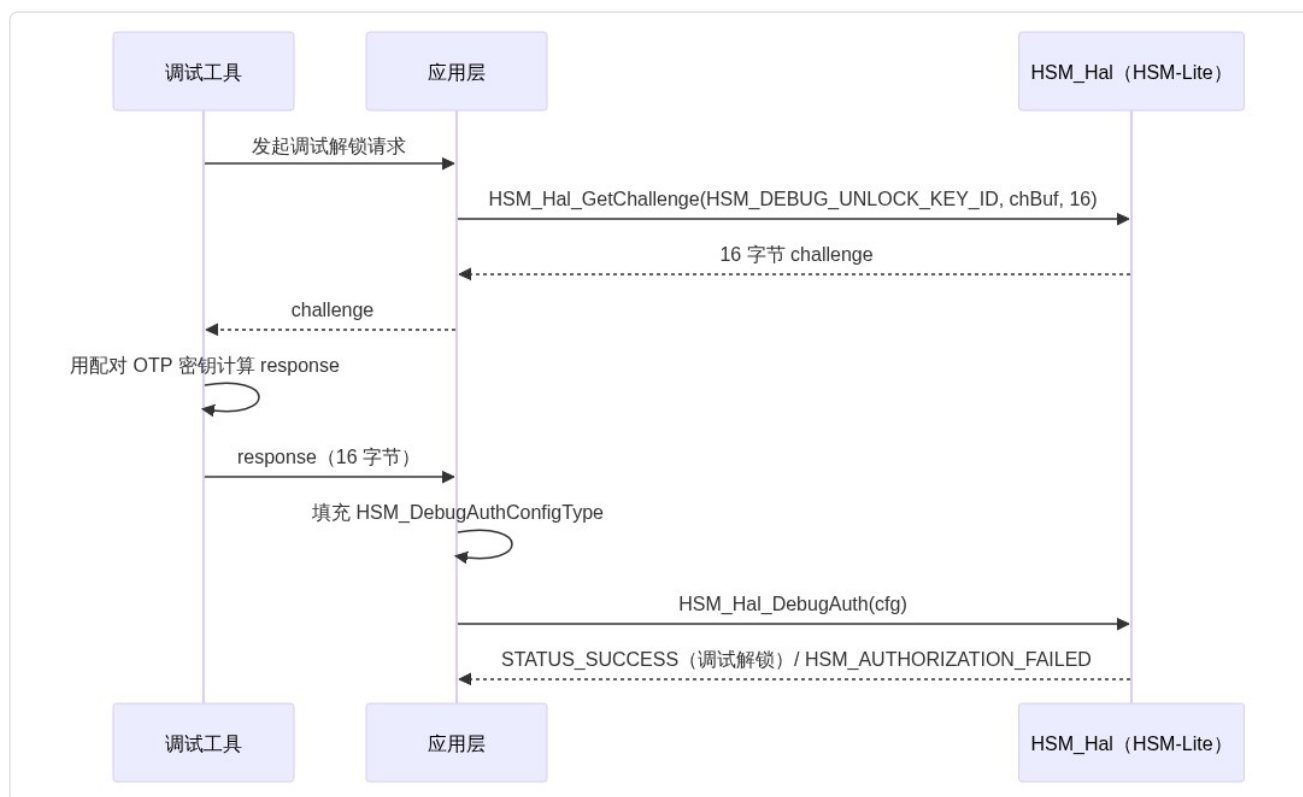
场景 4: Flash 密钥写入与删除



场景 5: 固件完整性验证 (安全启动)



场景 6: 调试认证 (Challenge-Response)

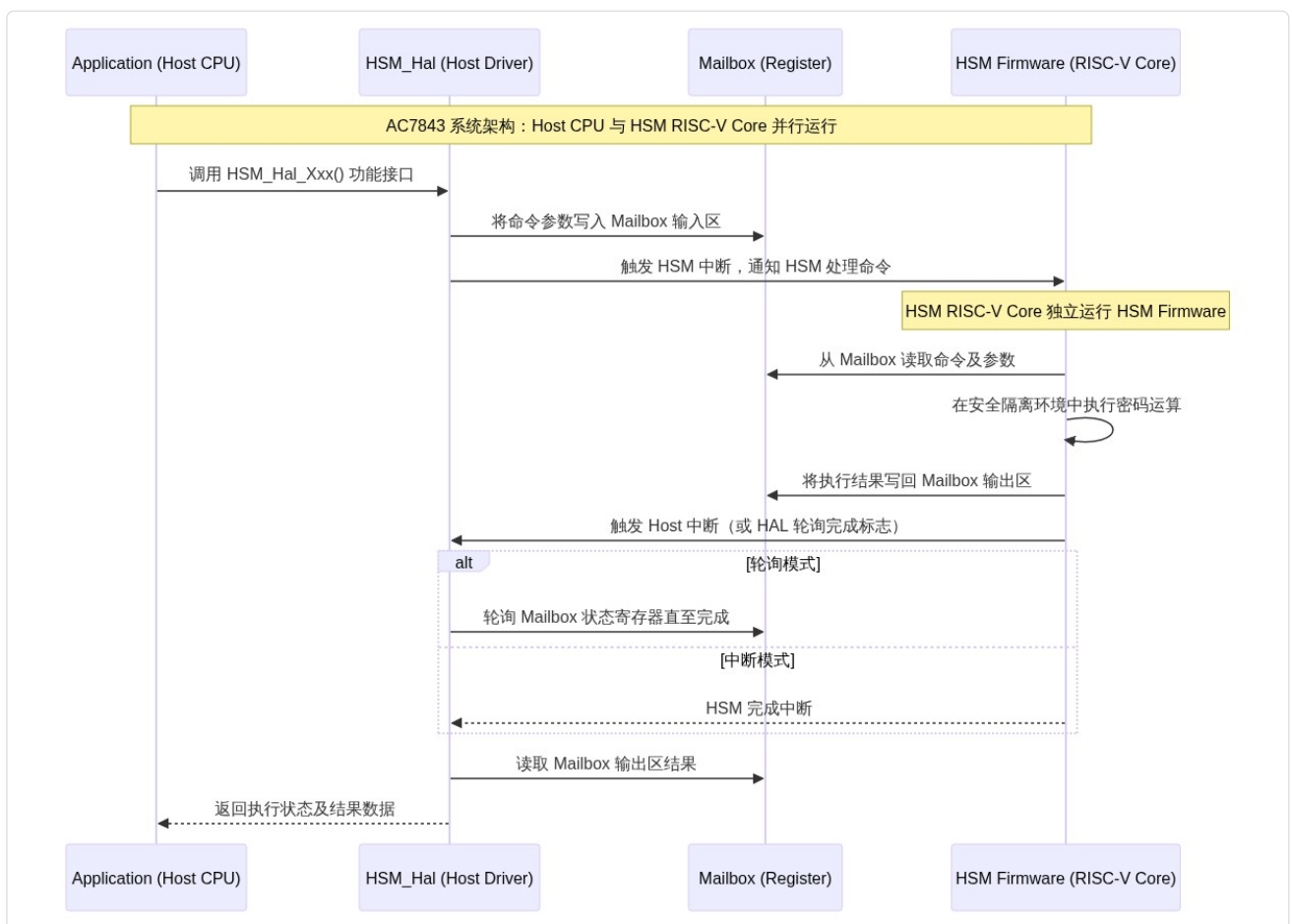


AC7843 HSM 应用场景时序图

AC7843 HSM 运行于独立安全核，Application 通过 Mailbox 机制调用 **HSM_Hal_** 接口。以下时序图展示了主要功能的典型调用流程，时序图仅包含 Application 与 HAL 层。

系统架构概览：HSM 独立核通信机制

AC7843 内部集成独立的 RISC-V 安全核，运行专属的 HSM Firmware（由我司提供，随 SDK 交付）。Host CPU 侧的应用程序通过 **HSM_Hal_** 驱动接口与 HSM 核通信，底层通信通道为共享内存 Mailbox。下图展示一次完整的 HSM 服务请求的系统级交互路径。

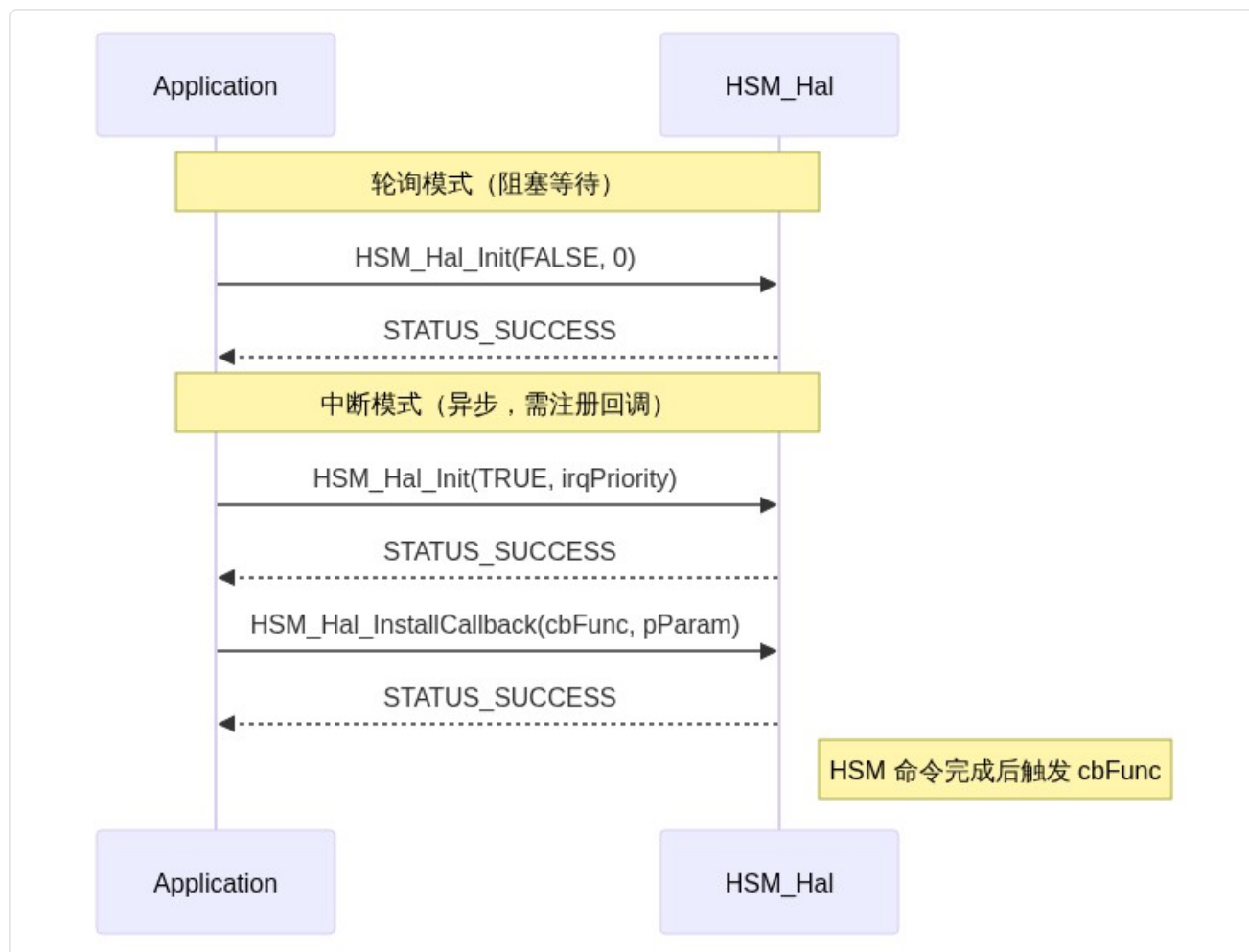


说明：

- **HSM Firmware** 固化于 HSM RISC-V Core 的专用 Flash 区域，Host 侧无需（也无法）直接访问其内部逻辑。
- **Mailbox** 为双向共享内存通道，Host 与 HSM Core 各有独立的写入区域，防止竞争冲突。
- **轮询模式**下，**HSM_Hal_Init(FALSE, 0)** 初始化，HAL 调用后阻塞等待直至命令完成。
- **中断模式**下，**HSM_Hal_Init(TRUE, irqPriority)** 初始化，命令由 HSM 异步执行，完成后触发 Host 中断并回调已注册的 **cbFunc**。
- 以下各应用场

景时序图均在此架构基础上展开，图中 **HSM_Hal** 代表 Host 侧驱动（含 Mailbox 通信），不展示 HSM Firmware 内部细节。

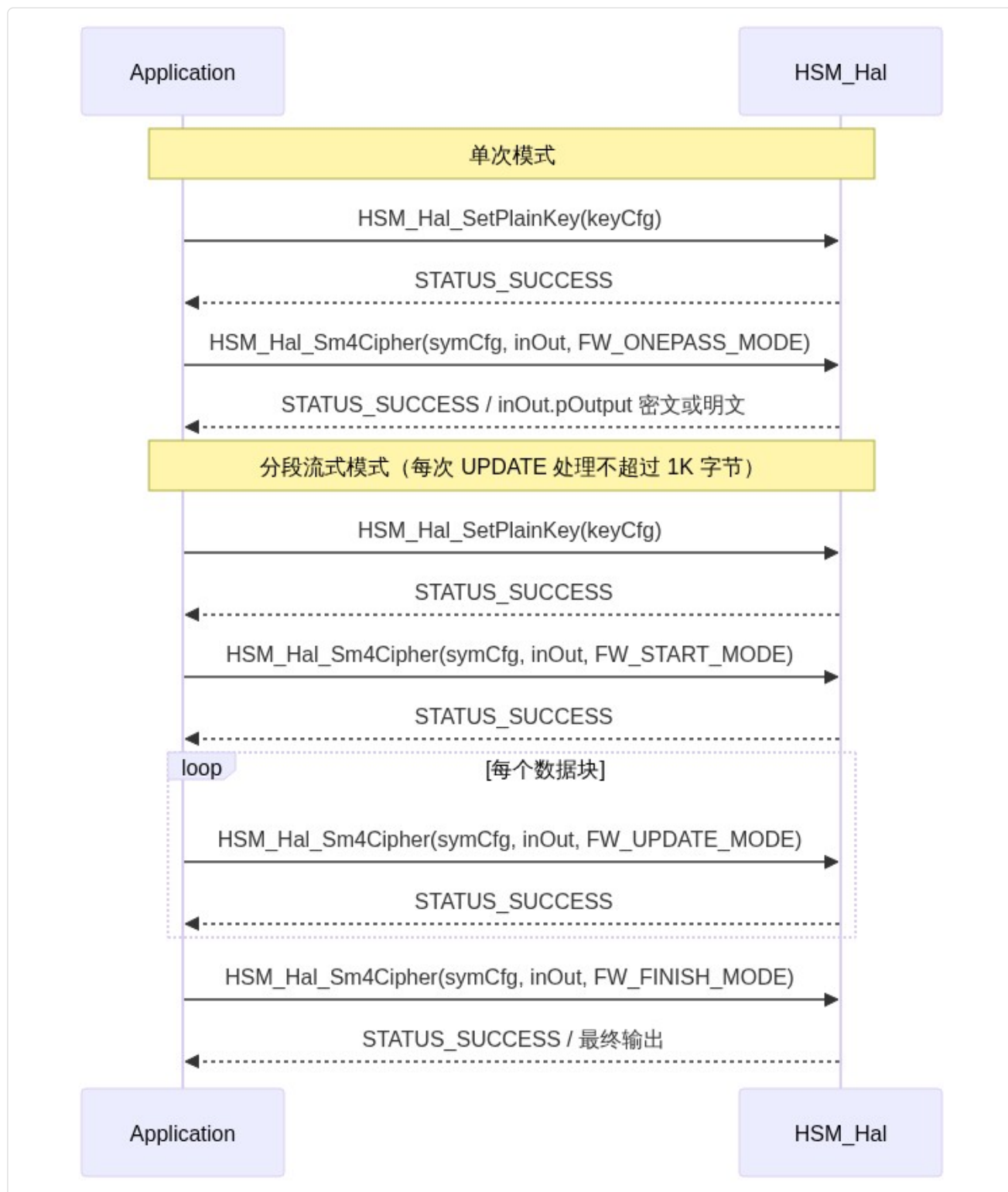
场景 1：初始化 (Init)



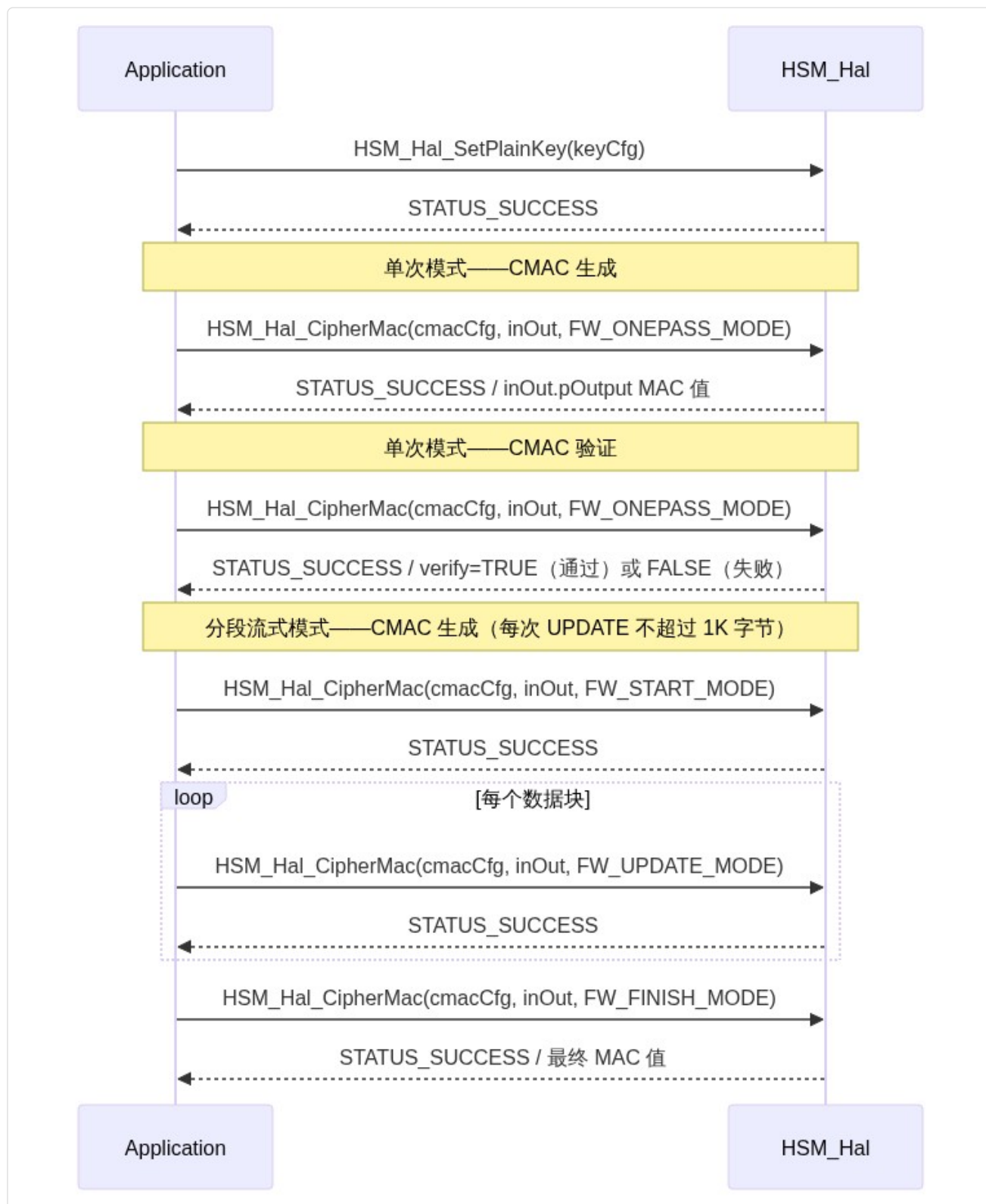
场景 2：AES 加解密



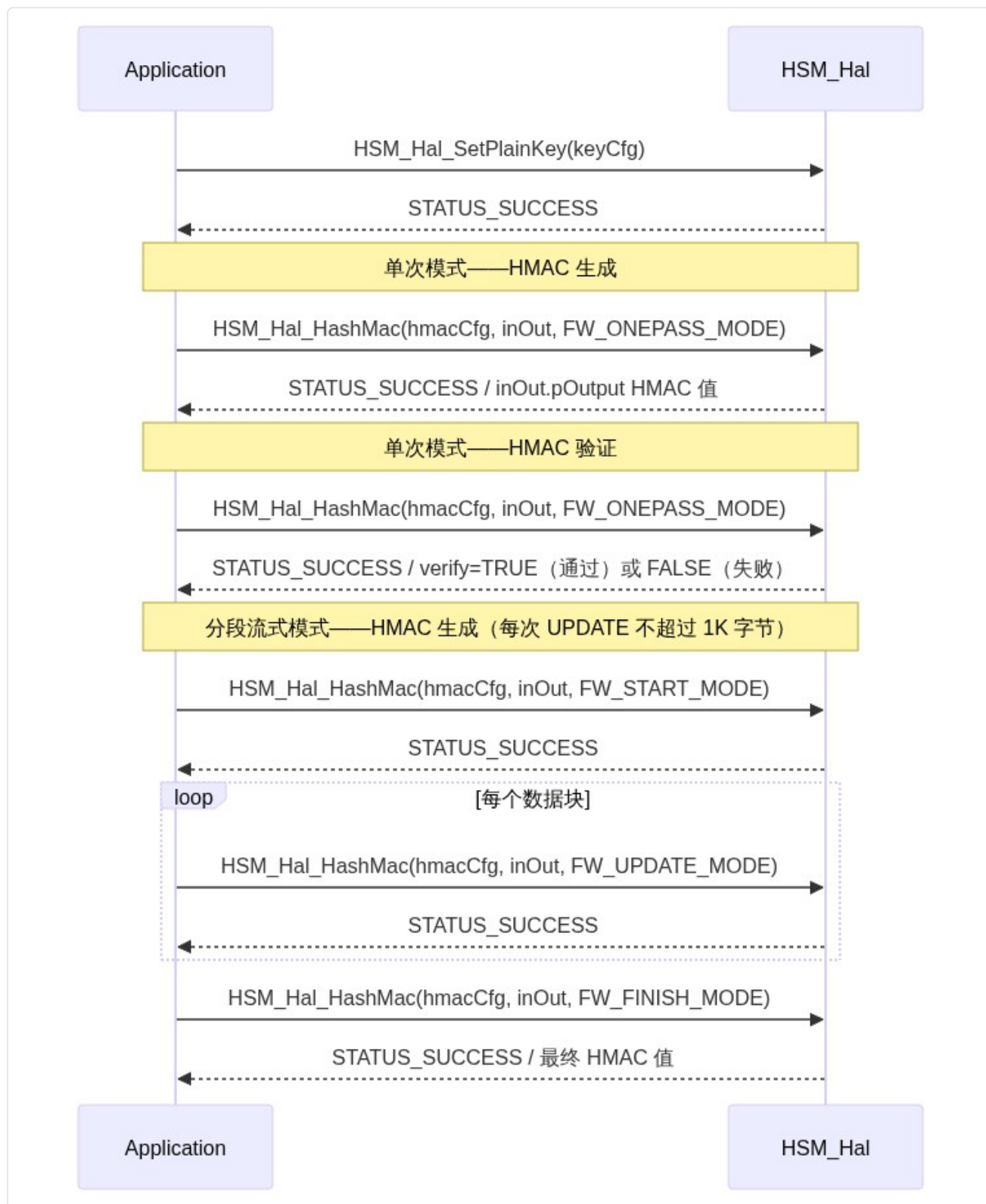
场景 3: SM4 加解密



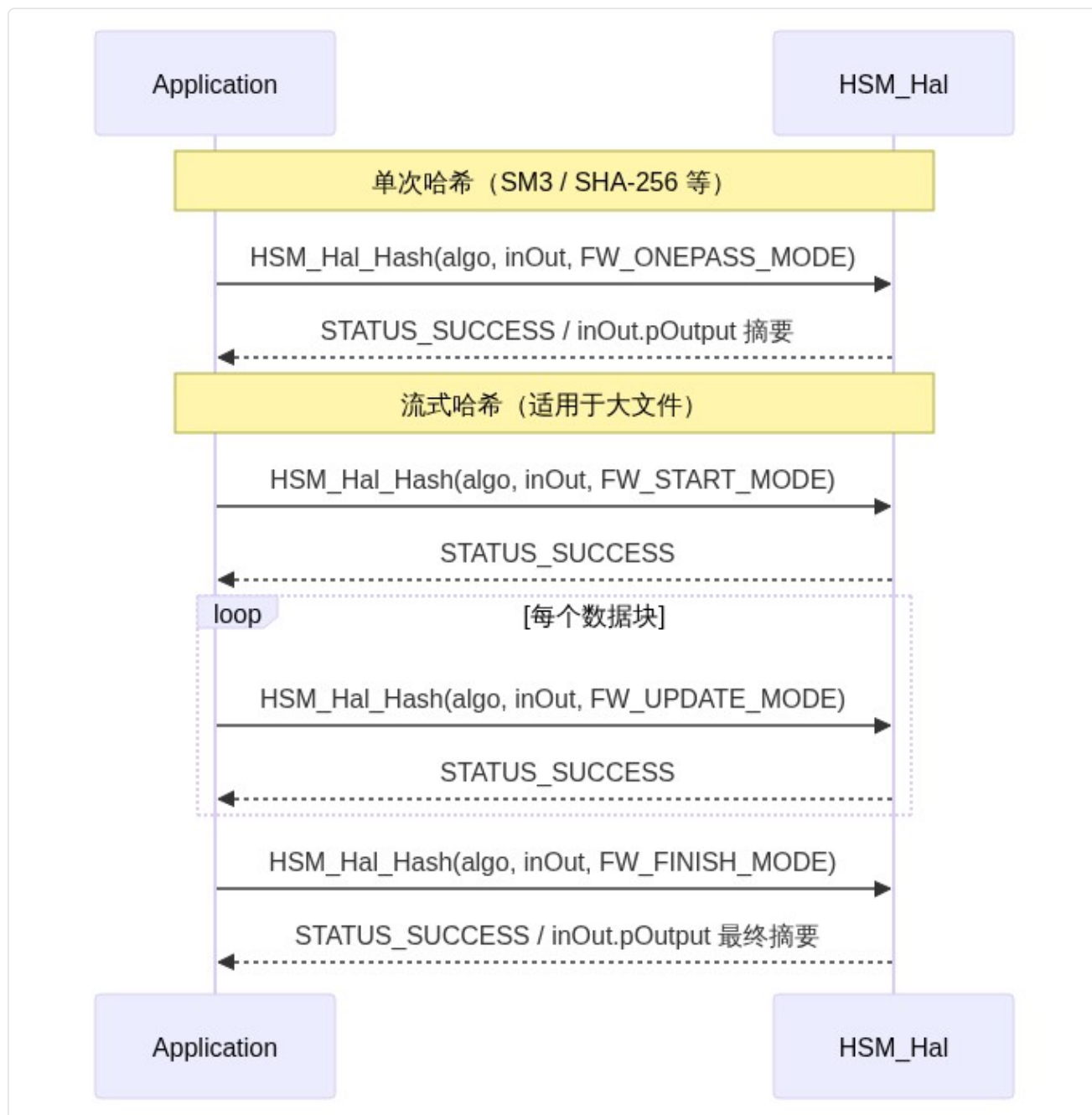
场景 4: CMAC 消息认证码



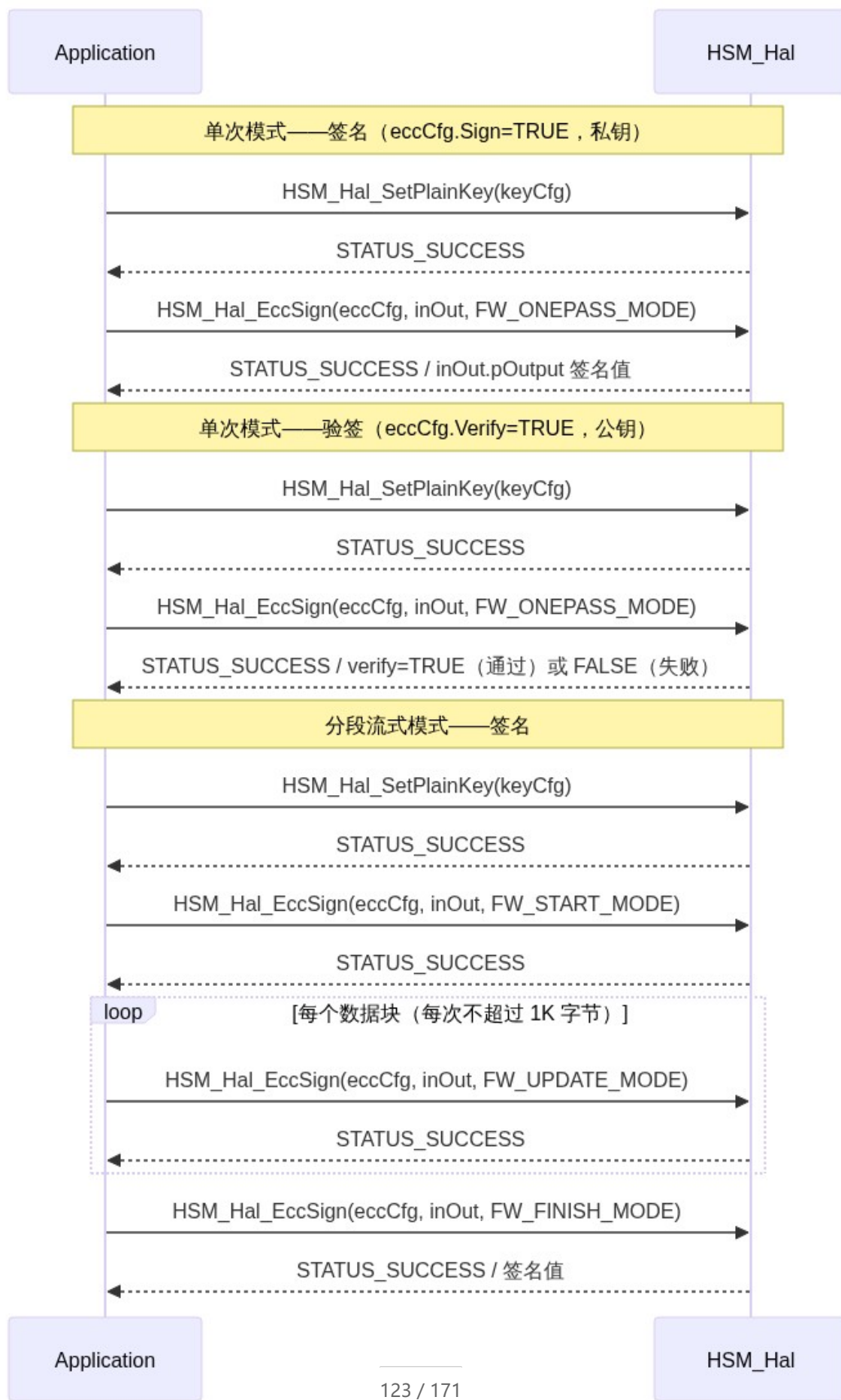
场景 5: HMAC 消息认证码



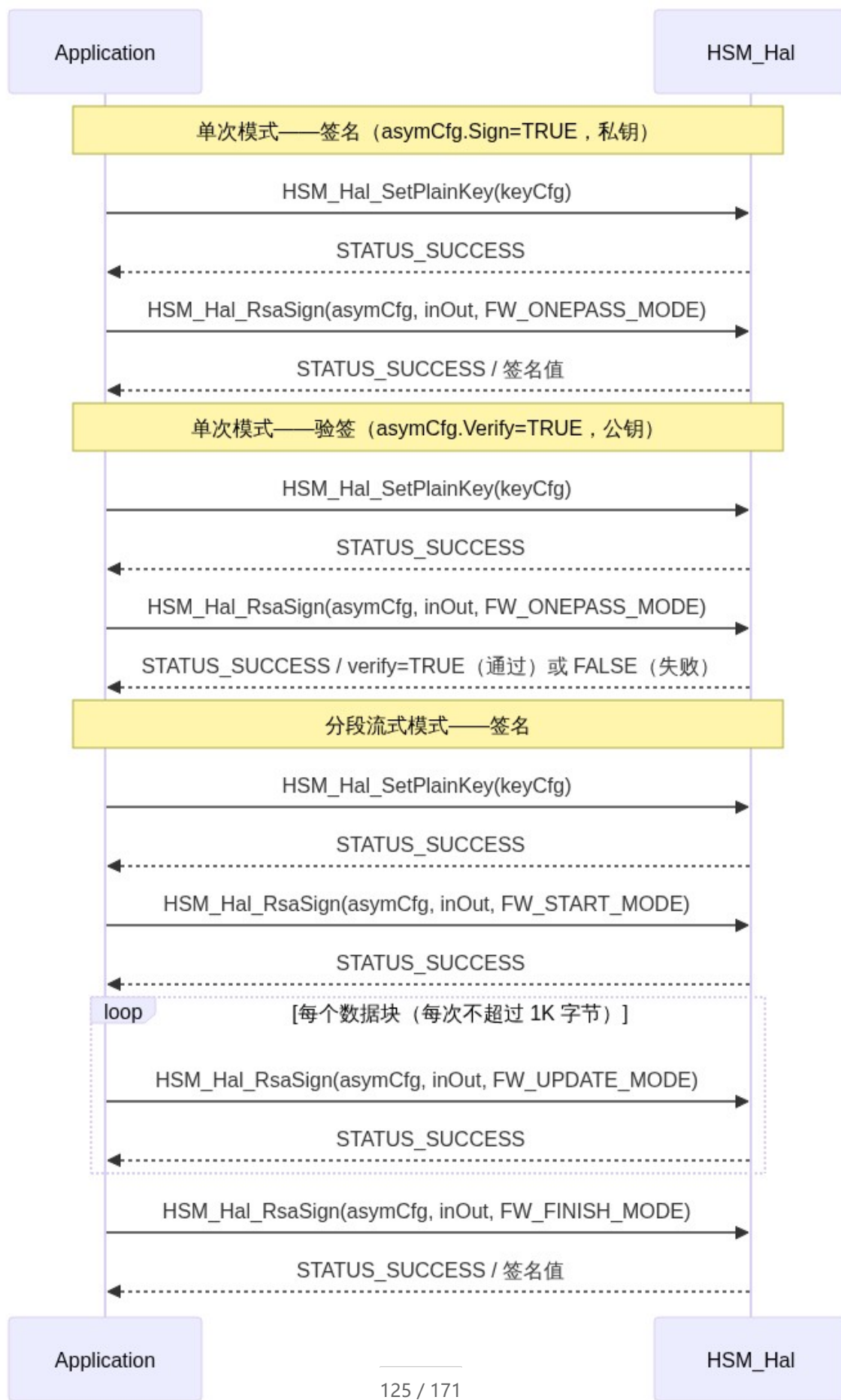
场景 6: 哈希 (Hash)



场景 7: ECDSA 签名与验签

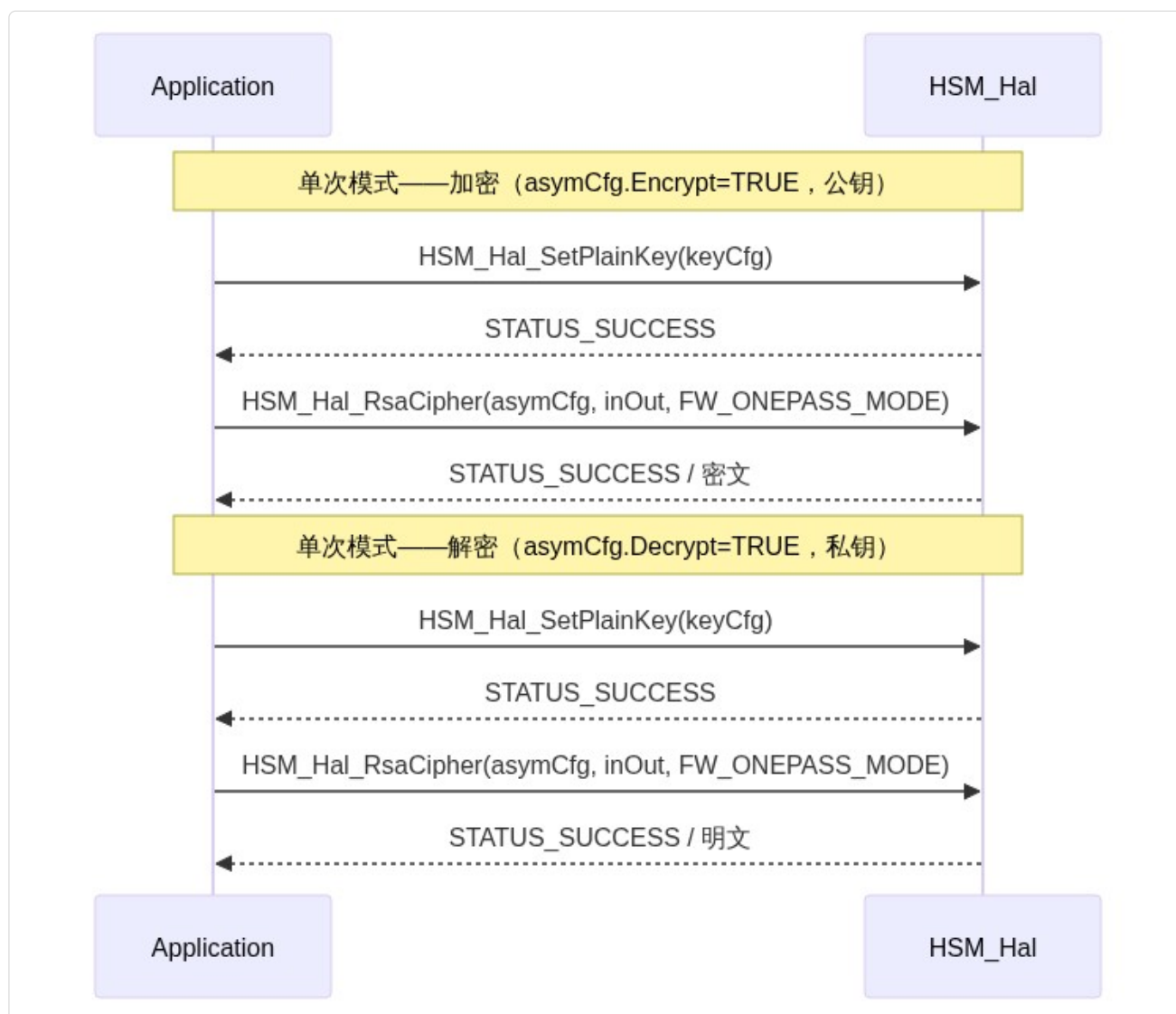


场景 8: RSA 签名与验签



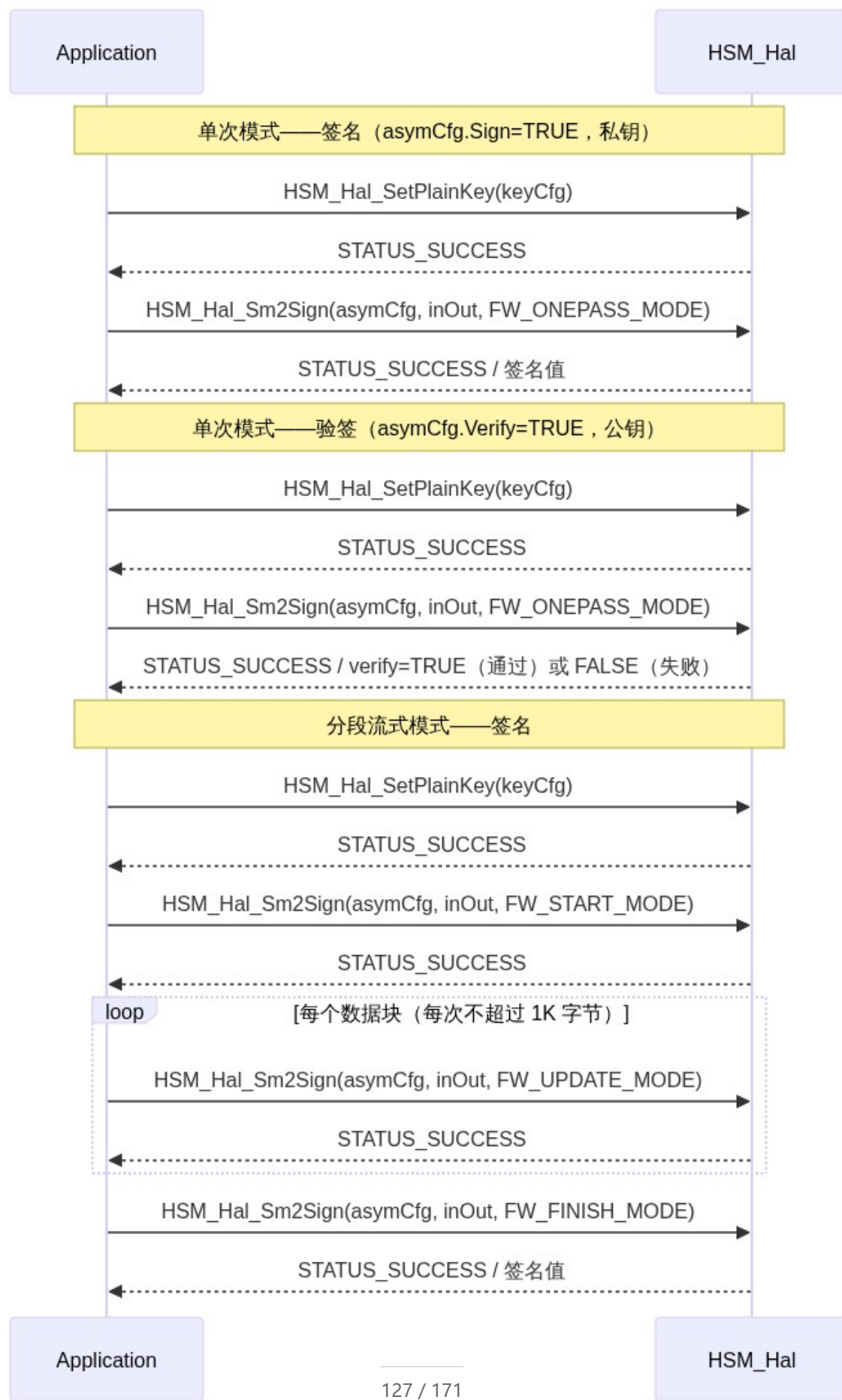
说明： `HSM_AsymCfgType` 中 `Sign=TRUE` 表示签名（使用私钥），`Verify=TRUE` 表示验签（使用公钥）；`AsymAlgo` 字段区分 RSA 1024/2048；`SaltLen` 字段仅 PSS padding 时使用。

场景 9：RSA 加解密



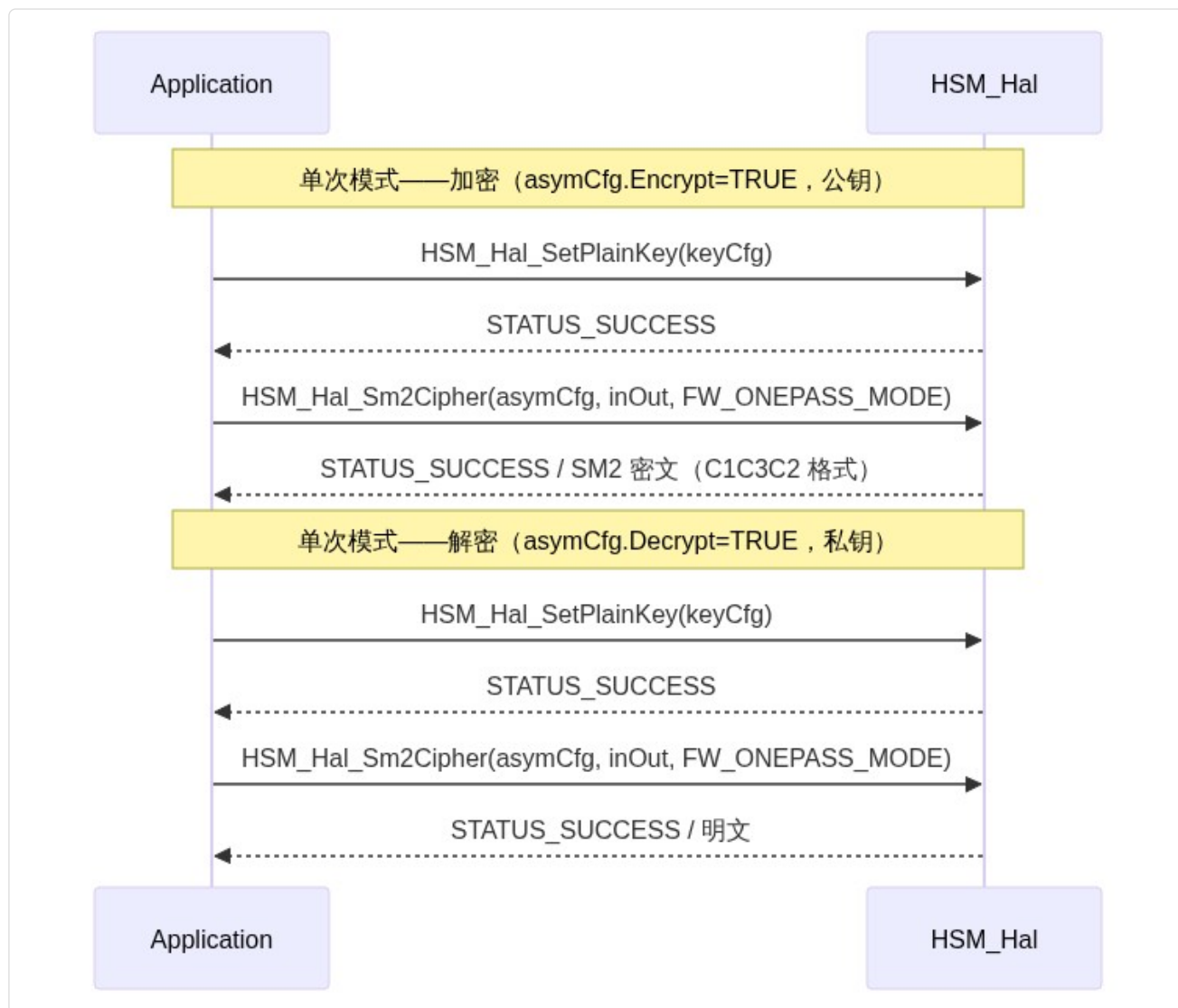
说明： RSA 加解密仅支持单次模式，不支持分段流式。`asymCfg.Encrypt=TRUE` 选择加密方向，`asymCfg.Decrypt=TRUE` 选择解密方向；padding 模式通过 `AsymAlgo` 的 OAEP / NO_PADDING 配置区分。

场景 10：SM2 签名与验签



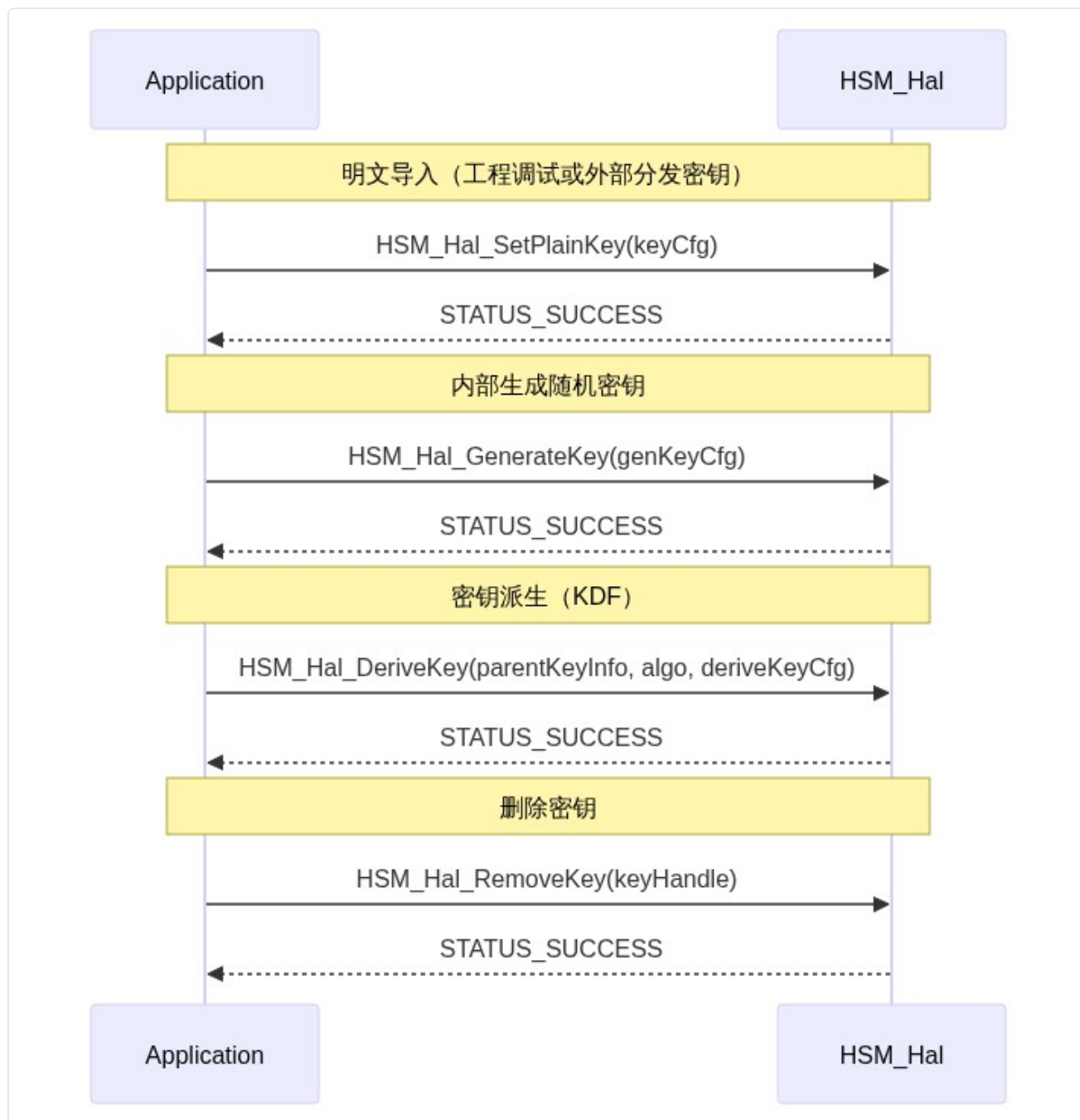
说明： SM2 签名场景需在 `asymCfg` 中设置 `Sm2UserIdBuf` 和 `Sm2UserIdLen` 字段（用户标识符，长度不超过 64 字节）；分段流式模式下，内部自动完成 SM3 哈希的分段计算，应用层接口保持一致。

场景 11：SM2 加解密

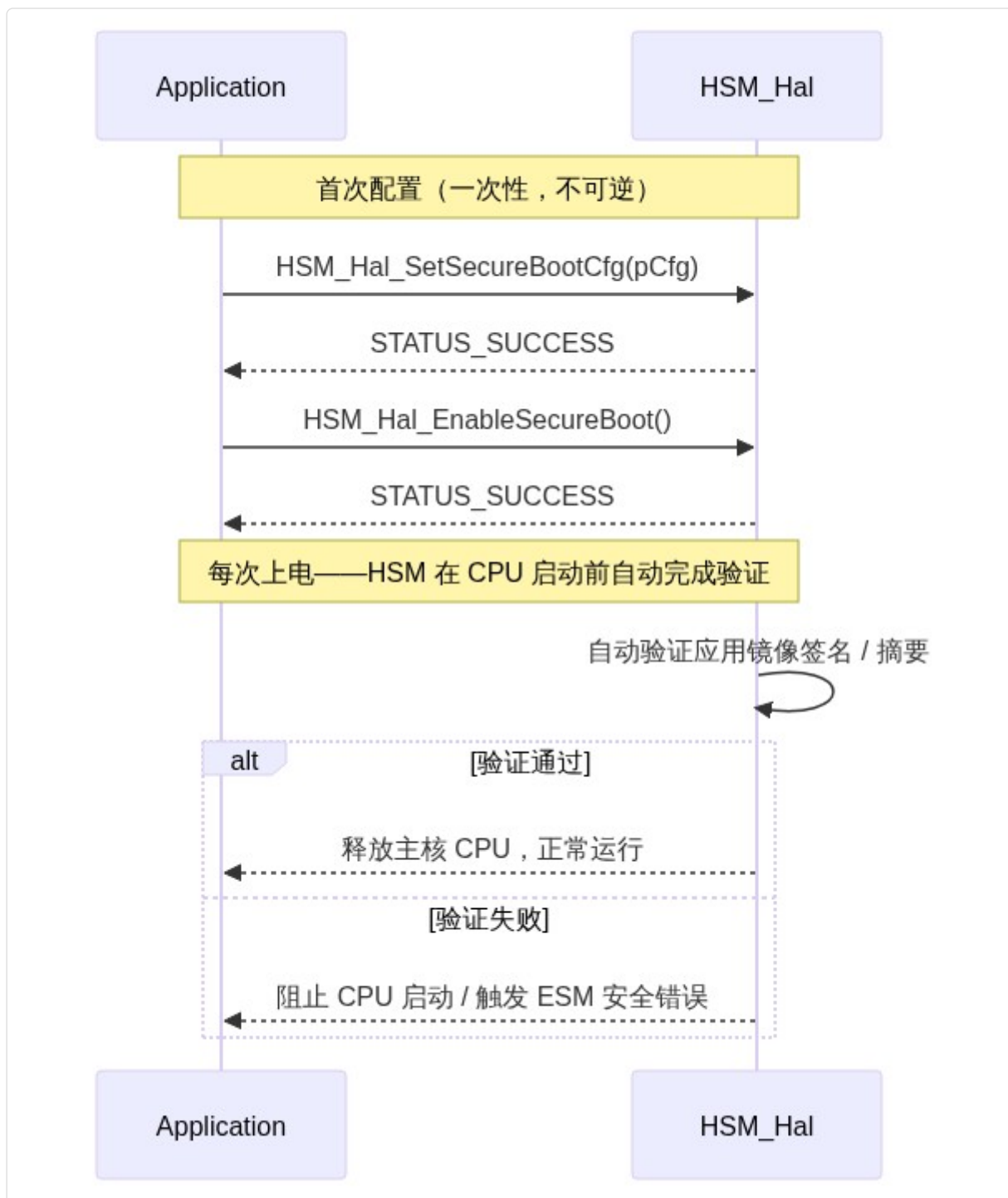


说明： SM2 加解密仅支持单次模式。输出密文采用 C1C3C2 格式（GM/T 0009 标准），若需对接 C1C2C3 格式须在应用层做格式转换。

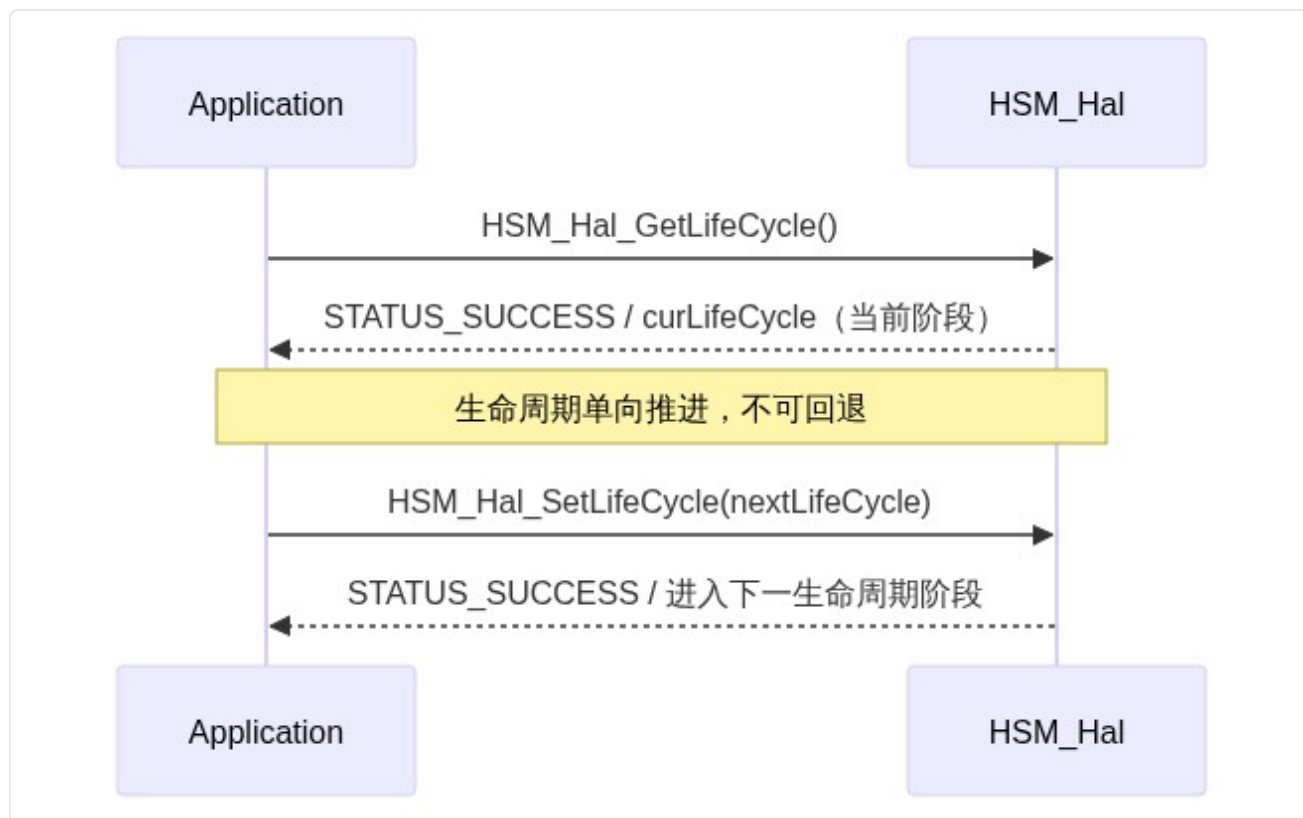
场景 12：密钥管理



场景 13: 安全启动 (配置与上电自动验证)



场景 14：生命周期管理



场景 15: 调试认证 (Challenge-Response)



3.4.1.2 与标准或规范的差异

产品	基准规范	主要差异
AC7840 / AC7842	SHE 规范	

产品	基准规范	主要差异
		在标准 KEY_1~KEY_10 之外扩展了 KEY_11~KEY_17 (厂商扩展)；其余与规范完全一致
AC7840E	Evita-Lite	-
AC7843	Evita Full 安全架构；GM/T 0002 / 0003 / 0004 (国密系列)	SM2 密文固定为 C1C3C2 格式，如需对接 C1C2C3 须应用层格式转换

3.4.1.3 静态配置项说明

编译宏配置

宏名称	说明	适用产品
AC7840X	指定目标为 AC7840X 系列，HAL 使用 AC7840X PRAM 布局	AC7840
AC7842X	指定目标为 AC7842X 系列，HAL 使用 AC7842X PRAM 布局	AC7842
CSE_SDK_NON_EXTENDED_API	定义后，Deinit / VerifyMACAddrMode / ExtendRNGSeed / GetAsyncCmdStatus / GetStatus / GetBOKStatus 等扩展 API 不可用 (精简版场景)	AC7840 / AC7842
AC7840E	选择 AC7840E 芯片配置，激活 HSM-Lite 驱动路径 (device/config/ Conf_AC784xx.h 通过 #if defined(AC7840E) 条件编译)	AC7840E
HSM_SUPPORT_SW_RSA	启用软件 RSA 支持 (默认未定义；启用后增加约 10~20 KB 代码空间，运算在主核执行)	AC7840E
HSM_HAL_SUPPORT_OTP_INTERFACE	启用 OTP 操作接口 (HSM_Hal_OtpWrite 、 HSM_Hal_SetLifeCycle 等；默认未定义，生产环境按需开启)	AC7843

3.4.1.4 软件集成依赖

初始化依赖顺序对比

步骤	AC7840 / AC7842 (CSE)	AC7840E	AC7843 (HSM)
1	系统时钟 (CKGEN) 初始化	系统时钟 (CKGEN) 初始化	系统时钟 (CKGEN) 初始化
2	CSE_Hal_Init()	-	HSM_Hal_Init(HsmIrqEn, HsmIrqPri)
3	(可选) CSE_Hal_InstallCallback ack 注册异步回调	(可选, 中断模式) HSM_Hal_InstallCallback ack 注册 DMA 完成回调	(可选) HSM_Hal_InstallCallback ack 注册中断回调
4	CSE_Hal_LoadPlainKey y / CSE_Hal_LoadKey 装载密钥	HSM_Hal_SetPlainKey 将明文密钥写入 Flash 槽 (HSM_FLASH_KEY_SYM_0~9 或 RAM 槽	HSM_Hal_SetPlainKey / HSM_Hal_GenerateKey 确保密钥槽就绪

调用模型差异

产品	同步/异步模型
AC7840 / AC7842	TimeoutUs != 0 为同步模式; TimeoutUs == 0 为异步模式 (需轮询 CSE_Hal_GetAsyncCmdStatus 或 依赖回调)
AC7840E	HSM_SymCfgType.Async = FALSE 为同步轮询模式; Async = TRUE 为异步 DMA 模式 (需配合 HSM_Hal_InstallCallback 回调, 操作完成后触发)
AC7843	默认同步轮询模式; 通过 HSM_Hal_Init(TRUE, priority) + HSM_Hal_InstallCallback 切换为中断驱动模式

3.4.1.5 软件限制/开发须知

通用注意事项 (适用所有型号)

- 1. **安全模块不可跨型号混用**: CSE (AC7840/AC7842) 和 HSM (AC7843) 使用完全不同的 API, 同一应用代码不可同时调用两套接口。从 AC7840/AC7842 迁移到 AC7843 时需全量替换安全接口。
- 2. **RAM 密钥掉电丢失**: 所有型号的 RAM 类密钥 (CSE CSE_RAM_KEY 槽、HSM HSM_RAM_KEY_* 槽) 在掉电或复位后丢失, 每次上电需重新装载。

AC7840 / AC7842 CSE 特有限制

3. **AES-128 输入对齐要求**: ECB/CBC 加解密的输入长度必须为 **16 字节整数倍**, 非整块数据须应用层自行填充。
4. **密钥更新计数器单调递增**: `Cse_InputKeyInfoType.Count` 必须严格大于当前值, 否则 CSE 拒绝更新并返回 `CSE_KEY_UPDATE_ERROR`。
5. **写保护密钥不可更改**: 设置了 `KEY_ATTR_WRITE_PROTECT` 的密钥写入后永久不可修改, 只能通过 `CSE_Hal_ResetKey` (擦除全部密钥) 后重建, 操作不可逆。
6. **并发调用不安全**: CSE 为单命令串行处理模型, 多任务环境须应用层自行实现互斥, 防止并发调用 `CSE_Hal_*` 接口。

AC7843 HSM 特有限制

7. **初始化顺序强制要求**: 必须先完成 CKGEN 时钟初始化, 再调用 `HSM_Hal_Init`; 在 `HSM_Hal_Init` 返回 `STATUS_SUCCESS` 之前, 不得调用任何其他 `HSM_Hal_*` 接口, 否则行为未定义。
8. **分段模式每段不超过 1024 字节**: START/UPDATE/FINISH 分段流中, 每次 UPDATE 处理不超过 1024 字节, 且必须为 **16 字节整数倍**; ONEPASS 模式输入同样须 16 字节整数倍。
9. **OTP 写操作不可逆**: `HSM_Hal_OtpWrite`、`HSM_Hal_SetOtpExternalKey`、`HSM_Hal_EnableSecureBoot`、`HSM_Hal_SetLifeCycle` 等接口修改 OTP, 一旦写入无法撤销, 量产前必须在开发/测试环境充分验证。
10. **生命周期单向推进**: 生命周期只能向前推进 (如 Development → Manufacture → User), 不可回退; 误操作将导致相应功能永久锁定。
11. **多任务互斥访问**: RTOS 多任务环境中, 调用前执行 `HSM_Hal_Lock()`, 操作完成后立即调用 `HSM_Hal_Unlock()`, 避免 Mailbox 竞争导致数据错误或超时。
12. **SM2 密文固定为 C1C3C2**: `HSM_Hal_Sm2Cipher` 的加密输出和解密输入均为 C1C3C2 格式; 对接使用 C1C2C3 格式的外部设备时, 应用层须在调用前后手动进行格式转换。

AC7840E HSM-Lite 特有限制

13. **输入长度对齐要求**: 加解密输入必须为 **16 字节整数倍**; 非整块数据须应用层自行填充 (ECB/CBC/CFB8/CFB128/OFB/CTR 均适用)。
14. **RAM 密钥仅限寄存器级**: `HSM_RAM_KEY_SYM_0` 直接写入硬件寄存器, 掉电或复位后立即丢失, 每次上电需重新调用 `HSM_Hal_SetPlainKey` (`KeyId = HSM_RAM_KEY_SYM_0`) 装载, 且 RAM 密钥仅生效一次, 在加解密或CMAC操作后, 如需再次使用RAM密钥, 需要再次进行装载。

15. **Flash 密钥页整理消耗擦写寿命**: Flash 密钥区采用主页 + 备份页双页管理, 密钥槽耗尽时 SDK 自动触发页整理 (半页擦写); 频繁更新密钥时需评估 Data Flash 擦写寿命上限。
16. **RSA 为纯软件实现**: 需定义 `HSM_SUPPORT_SW_RSA`; RSA-2048 运算全在主核执行, 能耗较长, 不适合时延敏感场景。
17. **异步 DMA 通道独占**: 异步加解密、CMAC 占用 `DMA_REQ_HSM_TX` 和 `DMA_REQ_HSM_RX` 两个 DMA 请求通道, 异步操作期间不可被其他外设共用; 多外设 DMA 场景需应用层协调通道分配。
18. **并发调用不安全**: HSM-Lite 直接操作硬件寄存器, 无内置互斥保护; RTOS 多任务环境须应用层加锁 (如互斥量), 防止并发调用 `HSM_Hal_*` 导致寄存器状态混乱。

关键差异说明:

1. **安全模块互斥**: CSE (AC7840/AC7842)、HSM-Lite (AC7840E) 与 HSM (AC7843) 三种安全模块 API 互不兼容。CSE → HSM 需全量替换接口; HSM-Lite → HSM 虽同为 `HSM_Hal_` 前缀但 API 集合不同 (AC7840E 无 `HSM_Hal_EccSign`、`HSM_Hal_OtpWrite` 等), 且密钥 ID 定义不同, 迁移时需逐接口核对。
2. **分段处理模型**: CSE 仅支持单次调用; HSM 引入 ONEPASS 和 START / UPDATE / FINISH 三段式模型, 支持分段计算的模式。
3. **密钥认证机制**: CSE 通过 M1~M5 协议认证密钥更新; HSM 通过 `AuthValue` 对密钥槽设置访问认证, 概念和接口不同。
4. **国密算法**: AC7843 HSM 完整支持 SM2/SM3/SM4 国密套件; AC7840E 支持 SM4 对称加解密及 SM4-CMAC, 不支持 SM2/SM3; AC7840/AC7842 CSE 均不支持任何国密算法。

3.4.1.6 算法性能参考

以下数据供开发者评估各算法调用时延参考。所有数据均选取数据量较大的典型测试场景展示。

对称算法 (AC7840/AC7842 CSE-MCA)

测试环境: AC7840/AC7842 | CSE-MCA 硬件加速 | Core 180 MHz | RAM 密钥

算法	操作	模式	数据量 (字节)	耗时 (μs)	吞吐量 (MB/s)
AES-128	加密	ECB	16	746	0.02
AES-128	解密	ECB	16	284	0.05
AES-128	加密	CBC	16	290	0.05
AES-128	解密	CBC	16	265	0.05
AES-128 CMAC	生成	—	16	8367	0.0

算法	操作	模式	数据量 (字节)	耗时 (μs)	吞吐量 (MB/s)
AES-128 CMAC	校验	—	16	59	0.26
AES-128	加密	ECB	1024	193	5.06
AES-128	解密	ECB	1024	200	4.88
AES-128	加密	CBC	1024	201	4.86
AES-128	解密	CBC	1024	207	4.72
AES-128 CMAC	生成	—	1024	131	7.45
AES-128 CMAC	校验	—	1024	142	6.88

说明:

- 以上数据来源于 MCA(42x) 硬件 AES 加速器实测 (AC7842) , 使用 RAM 密钥。
- Flash 密钥与 RAM 密钥性能基本相同, 差异 < 5%。
- AC7840/AC7842 CSE-MCA 不支持 SM4 算法。

对称算法 (AC7840E HSM)

测试环境: AC7840E | SDK V5.1.0 | Core 120 MHz | Flash 密钥

算法	操作	模式	数据量 (字节)	性能 (MB/s)
AES-128	加密	ECB	1024	6.69
AES-128	解密	ECB	1024	6.83
AES-128	加密	CBC	1024	6.78
AES-128	解密	CBC	1024	6.73
AES-128 CMAC	生成	—	1024	12.21
AES-128 CMAC	校验	—	1024	11.91
SM4	加密	ECB	1024	6.51
SM4	解密	ECB	1024	6.51
SM4	加密	CBC	1024	6.55
SM4	解密	CBC	1024	6.51
SM4 CMAC	生成	—	1024	12.36
SM4 CMAC	校验	—	1024	12.06

说明:

- 以上数据来源于 HSM_HAL 接口实测，均使用 FLASH 密钥。RAM 密钥在加解密场景下性能相近，但 CMAC 场景下因密钥加载开销时延显著增大，生产项目中建议提前将工作密钥写入 RAM 密钥槽。

非对称算法 (AC7840E HSM)

测试环境: AC7840E | SDK V5.1.0 | Core 120 MHz | Flash 密钥

算法	操作	填充/哈希	数据量 (字节)	性能 (ms/次)
RSA-2048	签名 (PSS)	SHA256	1024	958.64
RSA-2048	验签 (PSS)	SHA256	1024	58.70
RSA-2048	加密 (OAEP)	SHA256	256	57.77
RSA-2048	解密 (OAEP)	SHA256	256	960.80

说明:

- RAM Key 与 Flash Key 性能基本相同 (差异 < 2%) 。
- ECDSA/SM2 的签名与验签时延受数据长度影响较小，主要耗时在椭圆曲线模运算。

对称算法 (AC7843 HSM)

测试环境: AC7843 | HSM FW V2.0.1 | SDK V5.7.0 | Core 180 MHz | ONE PASS 模式 | Flash 密钥

算法	操作	模式	数据量 (字节)	耗时 (μs)	吞吐量 (MB/s)
AES-128	加密	ECB	16	680	0.02
AES-128	解密	ECB	16	268	0.06
AES-128	加密	CBC	16	256	0.06
AES-128	解密	CBC	16	280	0.05
AES-128 CMAC	生成	—	16	787	0.02
AES-128 CMAC	校验	—	16	301	0.05
AES-128	加密	ECB	1024	258	1.14
AES-128	解密	ECB	1024	345	2.83

算法	操作	模式	数据量 (字节)	耗时 (μs)	吞吐量 (MB/s)
AES-128	加密	CBC	1024	332	2.94
AES-128	解密	CBC	1024	368	2.65
AES-128 CMAC	生成	—	1024	850	1.15
AES-128 CMAC	校验	—	1024	788	1.24

非对称算法 (AC7843 HSM)

测试环境： AC7843 | HSM FW V2.0.1 | SDK V5.7.0 | Core 180 MHz | ONE PASS 模式 | Flash 密钥

算法	操作	填充/哈希	数据量 (字节)	耗时 (us/次)	吞吐量 (MB/s)
ECDSA-256	签名	SHA256	16	27053	0.00
ECDSA-256	验签	SHA256	16	51596	0.00
ECDSA-256	签名	SHA256	1024	26313	0.04
ECDSA-256	验签	SHA256	1024	51241	0.02
RSA-2048	签名 (PSS)	SHA256	16	322407	0.00
RSA-2048	验签 (PSS)	SHA256	16	4373	0.00
RSA-2048	签名 (PSS)	SHA256	1024	333559	0.00
RSA-2048	验签 (PSS)	SHA256	1024	15512	0.06
SM2	签名	SM3	16	27053	0.00
SM2	验签	SM3	16	51596	0.00
SM2	签名	SM3	1024	27053	0.02
SM2	验签	SM3	1024	51596	0.02

3.5 Memory

Memory 子系统提供对片上 Flash 存储器的读、写、擦除和校验能力，包含 FLS (Flash) 和 EEP (模拟EEPROM) 两个模块。

3.5.1 EEP

EEP (Emulated EEPROM, EEPROM 仿真) 模块使用芯片内部 **DFlash + SRAM** 模拟 EEPROM，为应用层提供字节粒度、可按逻辑地址随机访问的持久化数据存储能力，无需外置 EEPROM 器件。

其核心机制如下：

- **SRAM 作为读写缓存：**所有读操作直接访问 SRAM，速度快且无 Flash 寿命消耗
- **DFlash 用于持久化：**每次写入同步将数据（含地址索引）追加写入当前 Flash Group；断电后下次初始化时加载至 SRAM
- **Group 轮转保护寿命：**当前 Group 写满后自动迁移至下一 Group 并擦除旧 Group，延长 Flash 使用寿命
- **校验保护数据完整性：**初始化时校验数据有效性，防止脏数据写入 SRAM

3.5.1.1 支持的功能

Eep HAL 层提供以下功能：

功能分类	说明
初始化	参数校验，从 DFlash 回放有效数据到 SRAM，建立 Group 状态
数据读取	直接从 SRAM 读取，支持任意起始地址和长度（字节粒度）
数据写入	更新 SRAM 后追加写入 DFlash，自动触发 Group 切换
全量擦除	擦除全部 EEP DFlash 区域并重新初始化 SRAM
状态查询	查询驱动当前运行状态（未初始化 / 空闲 / 忙 / 超时）
Group 管理	查询各 Block 剩余可写空间；支持强制切换 Group
版本查询	获取驱动主版本 / 次版本 / 修订号

API 完整说明请参考代码包 Doxygen 文档。

驱动状态枚举

```

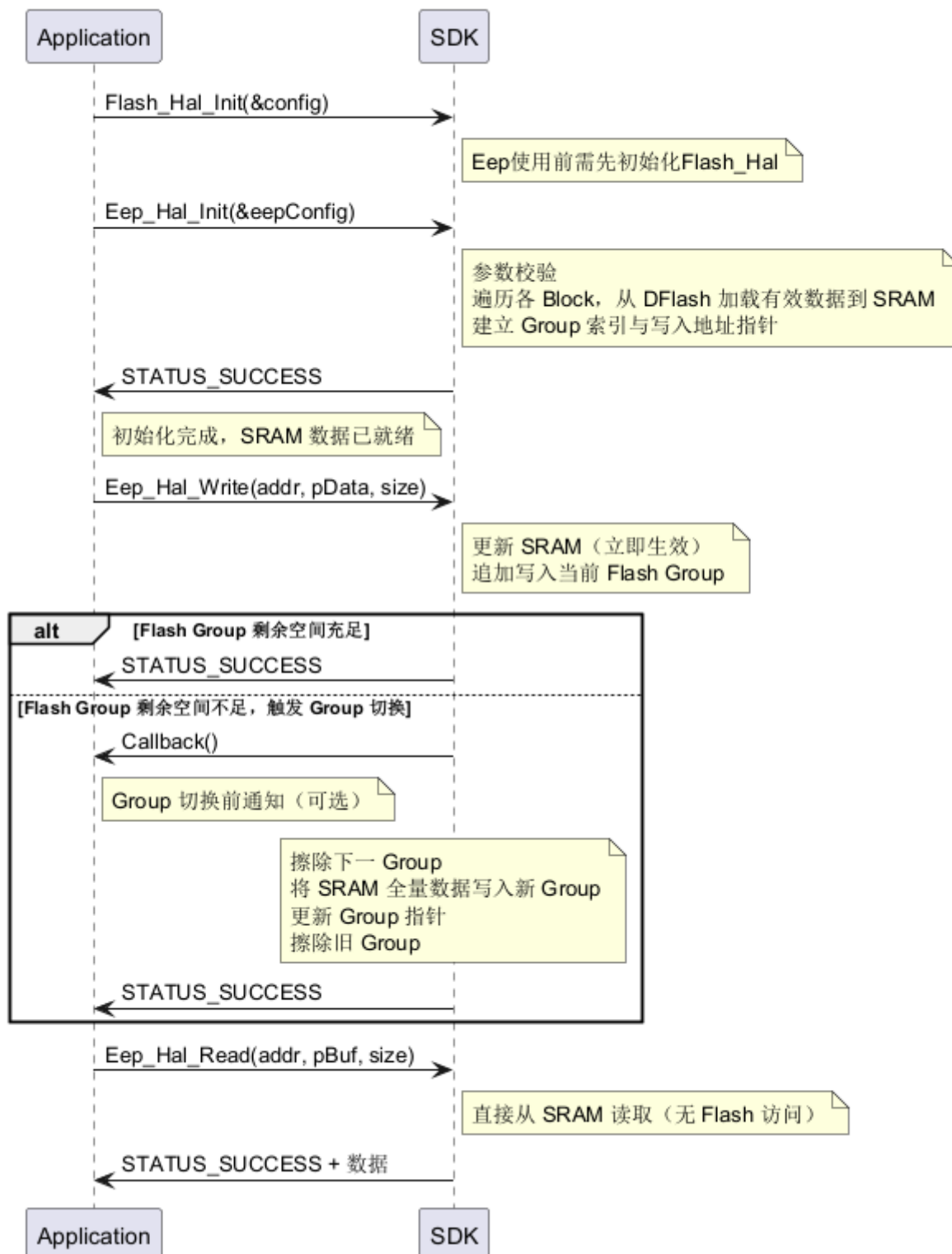
typedef enum {
    EEP_HAL_UNINIT      = 0x00U, /* 未初始化 */
    EEP_HAL_INITIALIZED = 0x01U, /* 已初始化（首次初始化完成后） */
    EEP_HAL_IDLE        = 0x02U, /* 空闲（写/擦除操作完成后） */
    EEP_HAL_TIMEOUT     = 0x03U, /* 超时（操作失败） */
    EEP_HAL_BUSY        = 0x04U, /* 忙（写/擦除操作进行中） */
} Eep_Hal_StatusType;

```

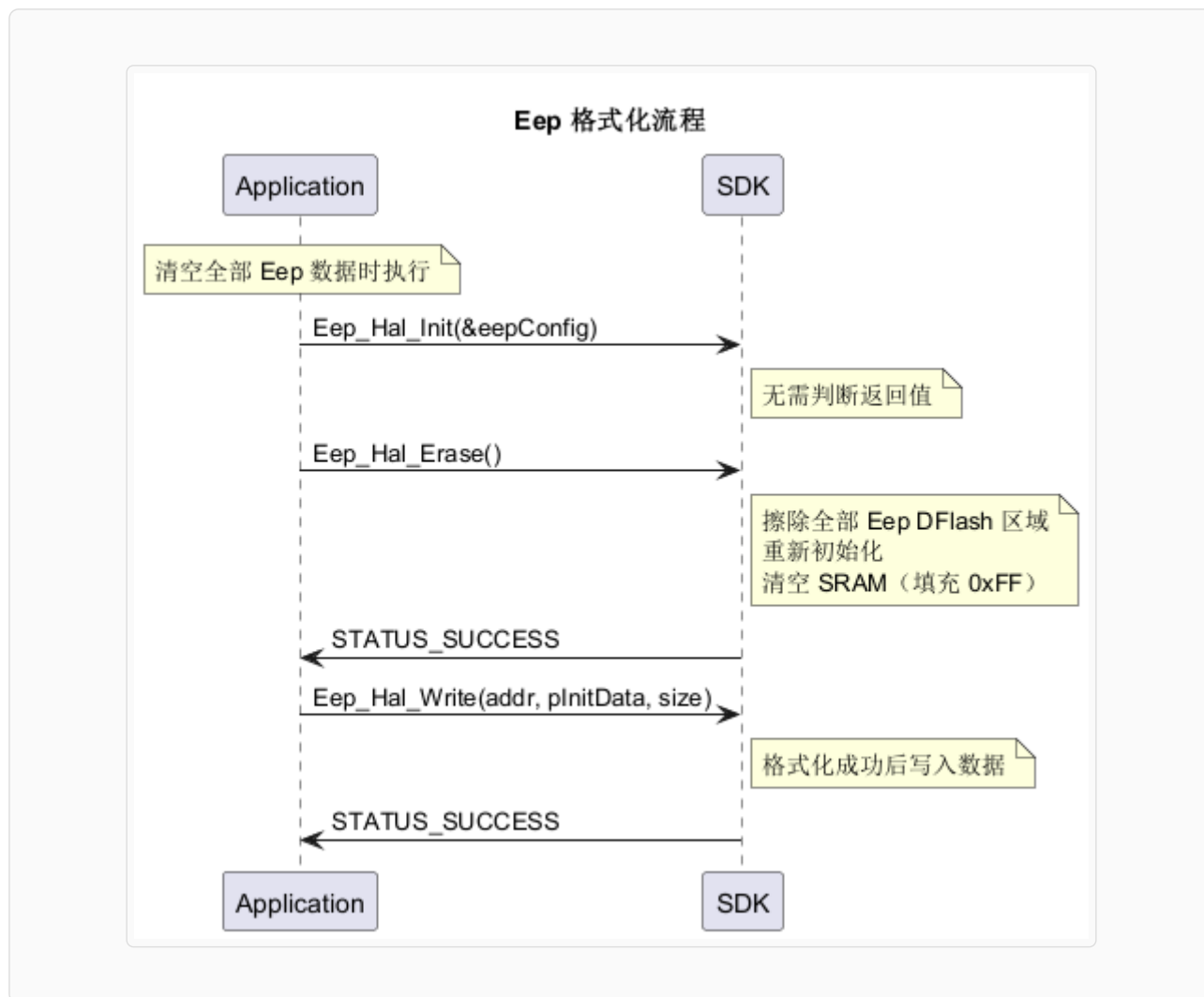
典型使用流程时序图

初始化与读写流程

Eep 初始化与读写典型流程



格式化流程



3.5.1.2 与标准或规范的差异

无，Eep模块为芯片私有模拟EEPROM实现。

3.5.1.3 静态配置项说明

配置结构体

```
typedef struct {
    uint16 EepSize;          /* EEP 总大小（字节）*/
    uint16 EepBlockSize;    /* 每个 Block 的 SRAM 大小（字节）*/
    uint16 EepGroupNum;     /* 每个 Block 对应的 Flash Group 数量 */
    uint32 EepRamBaseAddr;   /* EEP 使用的 SRAM 起始地址 */
    uint32 EepFlashBaseAddr; /* EEP 使用的 DFlash 起始地址 */
    Eep_CallbackType Callback; /* Group 切换前回调，可为 NULL_PTR */
} Eep_Hal_ConfigType;
```

字段说明与约束

字段	说明	取值范围	注意事项
EepSize	模拟 EEP 大小（单位 byte），决定模拟 EEP 占用 Flash 和 SRAM 的空间大小	512, 1024, 2048, 4096	由用户根据需求选择
EepBlockSize	EEP 按照 EepBlockSize 空间分组管理	固定设置 512	如 EepSize = 4K，EEP 内部划分 8 个组管理数据
EepGroupNum	每个分组使用的 DFLASH page 个数	3 ~ 4	如配置 EepSize = 4K，EepGroupNum = 3，则一共使用 3×8 = 24 个 dflash page (AC7843X/AC7840E DFlash需乘以2)
EepRamBaseAddr	模拟 EEP 占用 SRAM 空间起始地址	有效 RAM 地址	需要 4 字节对齐，从此地址之后 EepSize 的 SRAM 空间分配给模拟 EEP 使用
EepFlashBaseAddr	模拟 EEP 占用 Flash 空间起始地址	有效 DFlash 地址	必须为 Flash 页的页起点，从此地址开始，后 Flash Size 的 Flash 空间分配给模拟 EEP 使用
Callback	回调函数	函数指针或 NULL	Group 切换开始前调用，可用于应用层记录日志或通知；执行时间不应过长，避免影响写入

各产品 RAM / DFlash 地址约束

- AC7840X/AC7842X 如CSE启用 FlexRam区域不可用
- AC7840X/AC7842X 用户实际可用DFlash容量视CSE分区配置而定
- AC7843X/AC7840E 用户实际可用DFlash容量视HSM配置而定

空间使用说明

EEP 同时占用 SRAM 和 DFlash 两种存储资源：

EepSize	SRAM 占用	DFlash 占用 (EepGroupNum = 4为例)
512 B	512 B	8 KB (4 × 2 KB)
1024 B	1 KB	16 KB (4 × 4 KB)
2048 B	2 KB	32 KB (4 × 8 KB)
4096 B	4 KB	64 KB (4 × 16 KB)

说明： - SRAM 占用 = EepSize，初始化后常驻 SRAM

编译控制宏

宏名称	定义位置	适用产品	默认状态	说明
CONFIG_EEP_DRV_ENABLE	device/config/Conf_AC784xx.h	AC7840 专属	关闭	使能时切换至 eep_drv.c 实现，切换到 AC7840 旧版本数据格式；eep_drv.c 默认不集成到工程
CONFIG_EEP_SAVE_PHY_ADDR	device/config/Conf_AC784xx.h	AC7840 专属	关闭	使能时 EEP 在 Flash 中直接存储物理 RAM 地址而非逻辑偏移，用于切换到 AC7840 CMSIS 版本 EEP 数据格式
EEP_MPU_PROTECTION	编译器宏 / Conf_AC784xx.h	全系列	关闭	使能时 EEP 在写入/擦除期间通过 MPU 将 SRAM 区域设为只读，防止并发访问破坏缓存数据

CONFIG_EEP_DRV_ENABLE 详细说明：

- **默认状态：** 关闭，工程使用 Eep_Hal.c 标准实现
- **使能方式：** 在 Conf_AC784xx.h 中定义 CONFIG_EEP_DRV_ENABLE，并将 eep_drv.c 加入工程编译
- **使能效果：** Eep_Hal.h 转为包含 eep_drv.h，驱动实现由 Eep_Hal.c 切换至 eep_drv.c；接口名称保持不变，内部数据格式兼容 AC7840 旧版本 EEP

- **使用场景：**项目中存在 AC7840 旧版本 EEP 驱动写入的遗留 Flash 数据，需在不格式化数据的前提下切换至当前 SDK 时使用
- **注意事项：** `eep_drv.c` 默认不集成，须手动添加到工程；仅对 AC7840 有效，AC7840E / AC7842 / AC7843 不支持此宏；`CONFIG_EEP_DRV_ENABLE` 与 `CONFIG_EEP_SAVE_PHY_ADDR` 应根据实际遗留数据格式**二选一**使用，不能同时使能

`CONFIG_EEP_SAVE_PHY_ADDR` 详细说明：

- **默认状态：**关闭，EEP 以逻辑偏移地址存储数据
- **使能方式：**在 `Conf_AC784xx.h` 中定义 `CONFIG_EEP_SAVE_PHY_ADDR` 为 `1U`
- **使能效果：**Flash 中记录的地址字段由逻辑偏移（相对于 `EepRamBaseAddr`）改为物理 SRAM 地址；
- **使用场景：**项目已有使用 AC7840 CMSIS EEP 驱动写入的 Flash 数据，切换至当前 SDK EEP 驱动时需保持数据兼容，应使能此宏
- **注意事项：**该宏使能与否决定 Flash 数据的地址编码格式；**使能状态变更后原有 Flash 数据将无法正确解析，须在变更前执行 `Eep_Hal_Erase` 清空数据**；仅对 AC7840 有效，其他型号忽略此宏

`EEP_MPU_PROTECTION` 详细说明：

- **默认状态：**关闭
- **使能方式：**需用户在编译器宏或 `Conf_AC784xx.h` 中手动定义以启用
- **使能效果：**EEP 在执行初始化、写入、擦除等操作前进入临界区并将 SRAM EEP 区域的 MPU 权限降为只读（`MPU_ATTR_R`），操作完成后恢复读写权限（`MPU_ATTR_RW`），防止其他任务或中断在操作期间意外修改 SRAM 缓存数据
- **MPU 资源占用：**使能后固定占用 `MPU_ID_0` 的 `MPU_REGION_ID_4` 区域，须确保该 MPU 区域未被其他模块占用，另外需要 Eep Sram 缓存区和 MPU 保护区域绑定
- **注意事项：**不使能时 SRAM EEP 区域无保护，应用层注意避免踩内存到 SRAM EEP 区域

3.5.1.4 软件集成依赖

依赖模块	依赖说明	初始化顺序要求
CKGEN（时钟生成）	DFlash 读写操作依赖时钟配置正确	必须在 <code>Eep_Hal_Init</code> 之前完成时钟初始化
DFlash（Flash HAL）	EEP 内部通过 <code>Fls_Hal</code> 进行 DFlash 读/写/擦除操作	DFlash 模块须可正常工作
MPU（内存保护单元）		内部初始化客户无需关心

依赖模块	依赖说明	初始化顺序要求
	EEP 在初始化或写入擦除完成后配置 MPU 将 SRAM 区域降为只读，防止其他程序误破坏缓存数据	

3.5.1.5 软件限制/开发须知

1. EEP Flash 区域禁止设置写保护

EEP 使用的 DFlash 地址范围内禁止配置写保护（ `Flash_Hal_WriteProtectSet` ）。若该区域存在写保护，写入和擦除操作将返回错误。

2. 接口不支持重入，禁止并发调用

`Eep_Hal_Write` 和 `Eep_Hal_Erase` 不支持重入。禁止在中断中调用写入/擦除接口，也禁止在多任务环境中不加保护地并发调用。应用层须在调用前确保独占访问，或通过互斥锁保护。

3. 每次上电或复位后必须重新调用初始化

`Eep_Hal_Init` 负责将 DFlash 中的有效数据回放至 SRAM，若跳过初始化直接调用读写接口，将触发异常或返回 `STATUS_ERROR` 。

4. 首次使用建议先执行格式化

芯片DFlash 首次使用时，Flash 内容可能为随机值，Eep 可能初始化失败或读取到非预期数据。**建议在产品首次使用Eep时先调用 `Eep_Hal_Erase` 进行格式化**，再写入初始数据。

3.5.2 FLASH

3.5.2.1 支持的功能

AC784xx FLS HAL 层提供以下功能：

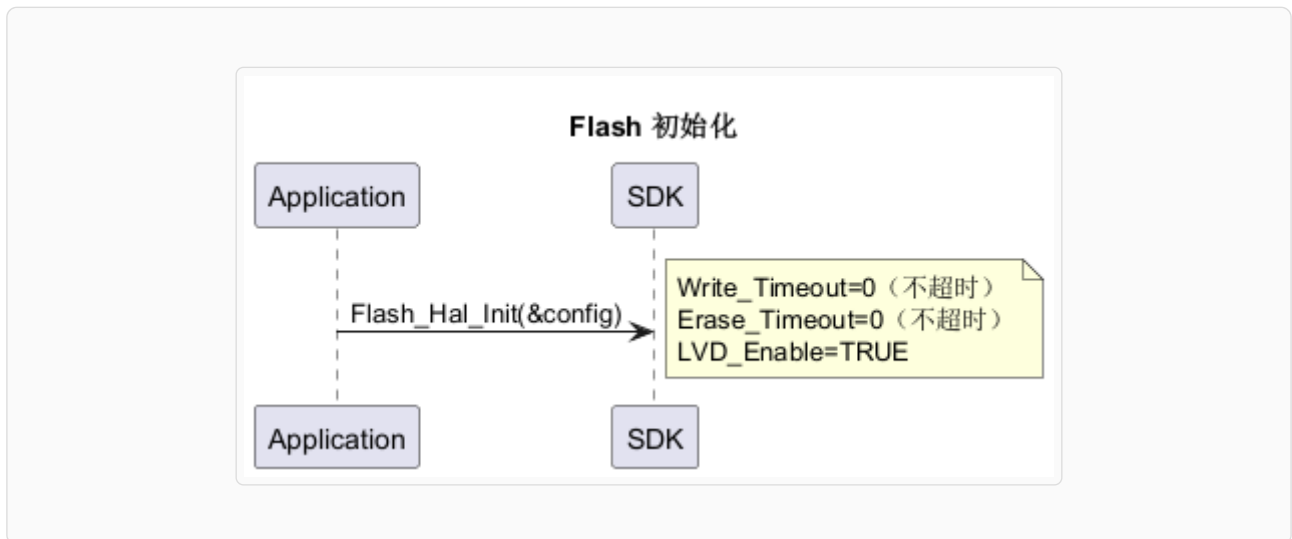
- **读取 / 编程 / 擦除 / 验证**：覆盖 PFlash、DFlash、Info 等区域；
- **读取 / 编程**：覆盖Flash OTP区域（AC7840E Hal层不支持编程）
- **读保护** 配置读保护，读保护生效后无法通过JTAG/SWD访问整个MCU空间，可通过backdoor key解除（AC7840E不支持读保护）
- **写保护** 不同区域写保护配置，支持寄存器写保护（立即生效，断电丢失），写保护信息区写保护（复位生效，断电保存）
- **CSE 分区** 划分 DFlash 末尾区域作为 CSE 密钥存储空间 (AC7843X / AC7840E 不支持)
- **Flash Swap** (AC7840X 不支持)

- **FlexRAM** 支持配置FlexRAM作为普通SRAM使用或者设置为CSE访问使用 (AC7843X / AC7840E不支持)

API 完整说明请参考代码包 Doxygen 文档。

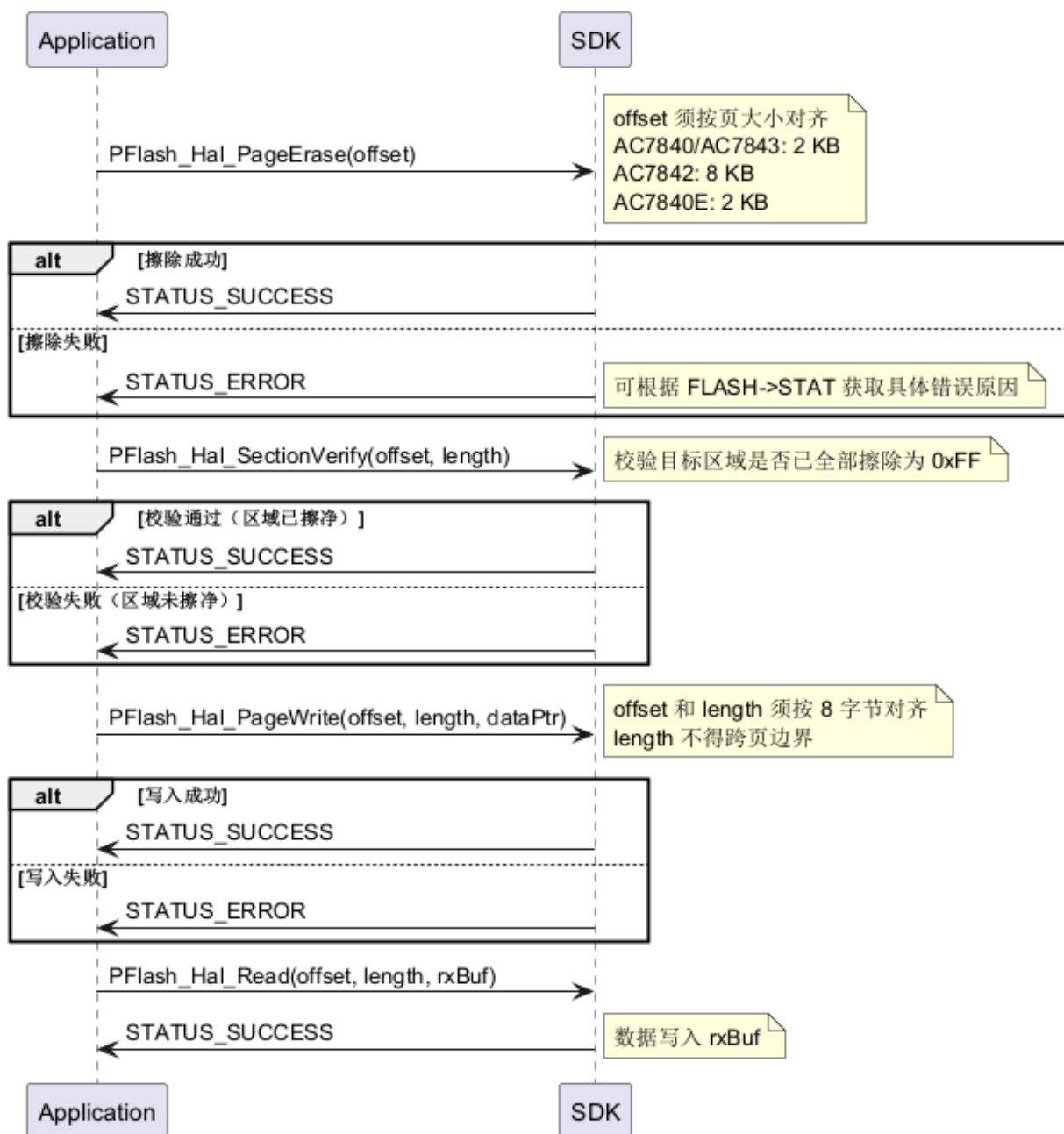
典型操作时序

初始化时序

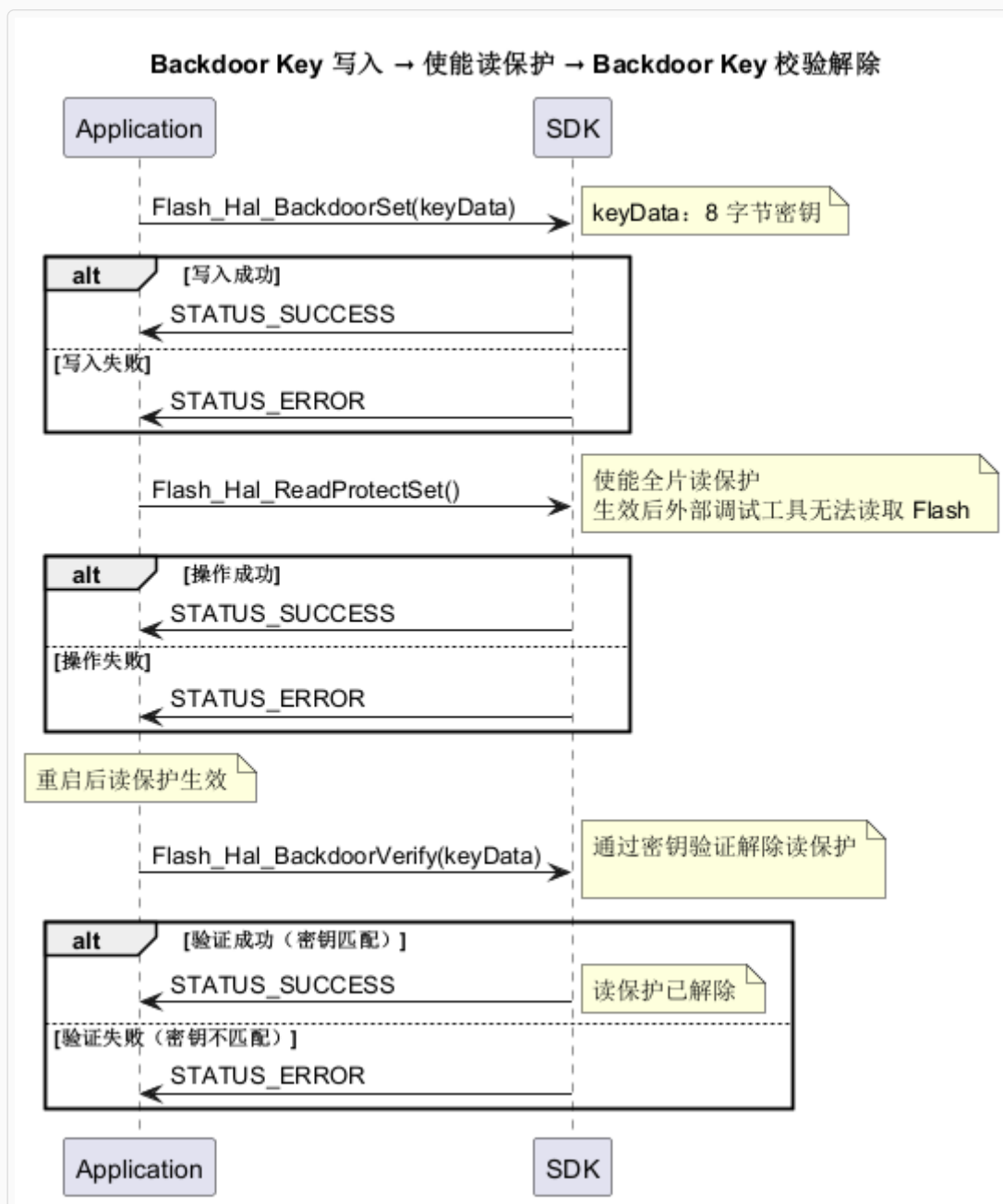


读取 / 写入 / 擦除 / 校验时序

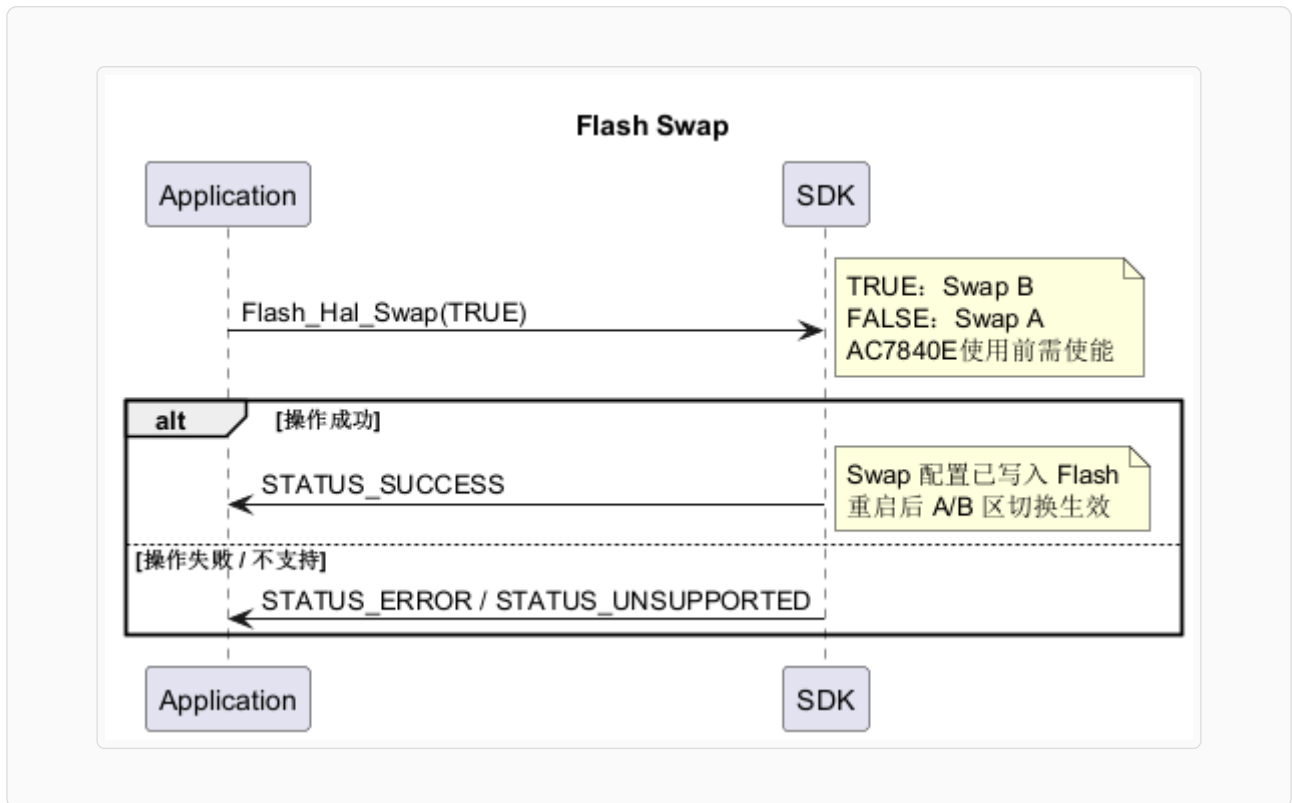
PFlash 擦除 → 校验 → 写入 → 读取



读保护设置和解除时序

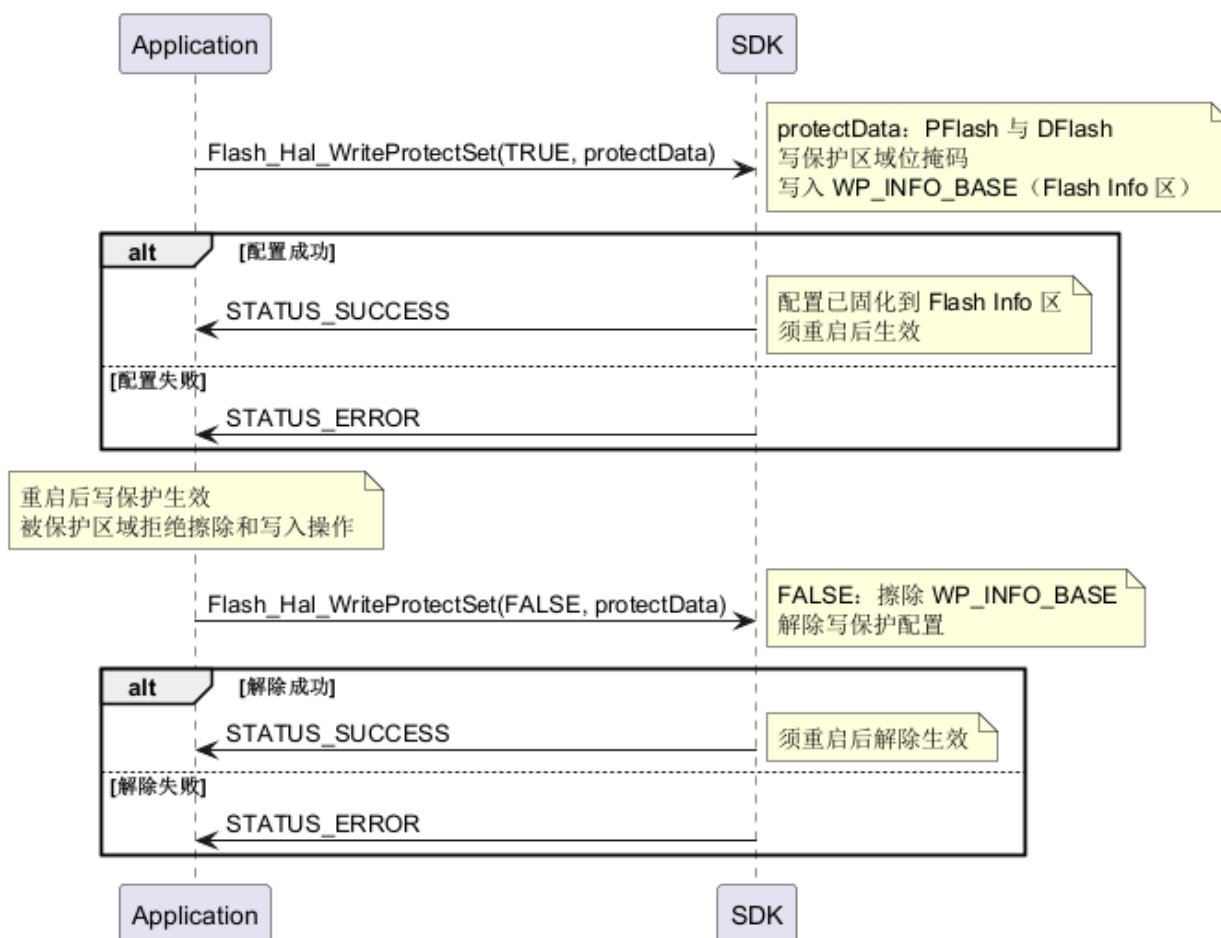


Flash Swap 配置时序

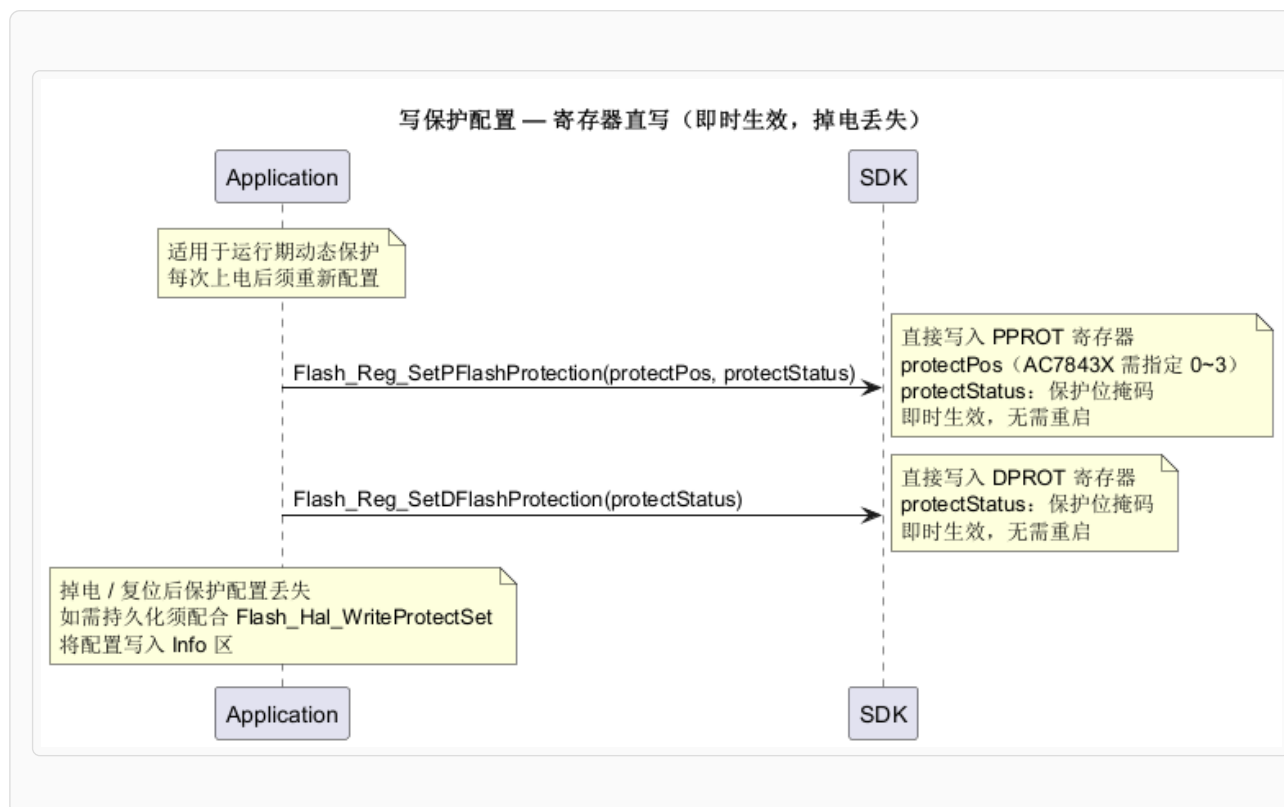


写保护配置时序 (Info 区持久化, 重启后生效)

写保护配置 — Info 区持久化（重启后生效）



写保护配置时序（寄存器直写，掉电丢失）



3.5.2.2 与标准或规范的差异

- Fls 模块中断功能未实现
- Fls 划分LayOut功能未实现（AC7840E only）
- Fls 多页擦除接口未实现（AC7840E only）

3.5.2.3 静态配置项说明

初始化配置结构体

FLS 模块通过 `Flash_Hal_Init` 传入 `Flash_Config` 结构体完成静态配置。结构体定义位于 `mcu/hal/include/Fls_Hal.h`：

```

typedef struct
{
    uint32 Write_Timeout; /* 写操作超时计数; 0 表示不启用超时, 等待直到操作完成 */
    uint32 Erase_Timeout; /* 擦除操作超时计数; 0 表示不启用超时, 等待直到操作完成 */
    boolean LVD_Enable; /* 低压检测使能; TRUE : 检测到低压时中止操作; FALSE : 不检测 */
} Flash_Config;
    
```

字段	类型	默认值	说明
<code>Write_Timeout</code>	<code>uint32</code>	<code>0U</code>	写超时计数, 0 为不超时

字段	类型	默认值	说明
Erase_Timeout	uint32	0U	擦除超时计数，0 为不超时
LVD_Enable	boolean	TRUE	低压检测使能，建议保持 TRUE

CSE 分区配置结构体

CSE 分区通过 `Flash_Hal_SetPartition` 配置，结构体定义于 `mcu/hal/include/Fls_Hal.h`：

```
typedef enum
{
    FLASH_CSESIZE_0BYTES,    /* 不设置 CSE 分区 */
    FLASH_CSESIZE_128BYTES,  /* CSE 分区大小 128 字节 */
    FLASH_CSESIZE_256BYTES,  /* CSE 分区大小 256 字节 */
    FLASH_CSESIZE_512BYTES   /* CSE 分区大小 512 字节 */
} Flash_CseSize;

typedef struct
{
    Flash_CseSize KeySize; /* CSE 密钥空间大小配置 */
    boolean Sfe;           /* Secure Flash Enable : TRUE 使能安全 Flash */
} Flash_CseType;
```

编译控制宏

宏定义	位置	说明
PFLASH_ENABLE	Fls_Hal.h	P-Flash API 编译开关，默认 STD_ON
DFLASH_ENABLE	Fls_Hal.h	D-Flash API 编译开关，默认 STD_ON

3.5.2.4 软件集成依赖

FLS HAL层依赖以下模块正确配置。

依赖项	说明	初始化顺序要求
CKGEN(时钟生成)	依赖时钟配置	时钟初始化后才可正常工作

3.5.2.5 软件限制/开发须知

1. 写操作地址和长度必须按 8 字节对齐

所有写入 API (`PFlash_Hal_PageWrite` 、 `PFlash_Hal_SectionWrite` 、 `DFlash_Hal_PageWrite` 、 `DFlash_Hal_SectionWrite`) 的 `Offset` 和 `Length` 均须为 8 字节 (`PFLASH_WRITE_UNIT_SIZE` / `DFLASH_WRITE_UNIT_SIZE`) 的整数倍。

2. 擦除操作地址必须按页大小对齐

`PFlash_Hal_PageErase` 和 `DFlash_Hal_PageErase` 的 `Offset` 须按对应芯片的页大小对齐：

芯片	PFlash 页大小	DFlash 页大小
AC7840X	2 KB	2 KB
AC7842X	8 KB	2 KB
AC7843X	2 KB	1 KB
AC7840E	2 KB	1 KB

3. DFlash 容量限制

AC7840X/AC7842X 用户实际可用DFlash容量视CSE分区配置而定; AC7843X/AC7840E 用户实际可用DFlash容量视HSM配置而定

4. 避免并发写入/擦除

Flash 控制器不支持重入，应用层须确保同一时刻只有一个Flash操作进行，禁止并发调用写入擦除校验等接口。

5. Section编程限制

Section编程能够按照8字节对齐方式把FlexRAM中的数据编程到指定Flash(P-Flash或者D-Flash)地址空间，需要注意的是：该编程方式不支持跨页编程，如果编程范围超过flash物理页地址范围，则只会编程本页，其余未编程的数据不会继续编程，并且需要FlexRAM功能为普通SRAM (AC7843X / AC7840E不支持)

4. 系统配置

本章按工程可落地的配置入口组织为三部分：工程编译配置、工作动态参数配置、默认 section 配置。

4.1 工程编译配置

AC784xx SDK 支持 ARMCC、GCC、IAR、GHS 四种编译器。本节逐一列出各编译器的编译器选项、汇编器选项和链接器选项配置，以及最小系统集成所需文件清单。

4.1.1 支持的编译器

编译器	配套IDE	IDE版本
ARMCC	Keil MDK	V5.23 ~ V5.41 (V5.30除外)
GCC	Eclipse	无限制
IAR	IAR	V9.70.1
GHS	MULTI IDE	8.1.6d 以上

GCC 默认环境：`arm-gnu-toolchain-12.3.rel1`、`cmake-3.15.0`、`mingw x86_64-12.2.0`。

4.1.2 ARMCC 编译配置

表 4.1 ARMCC 编译器选项配置

选项配置	描述
<code>--c99</code>	编译器按照 ISO/IEC 9899:1999 标准编译 C
<code>--cpu Cortex-M4.fp</code>	选择目标处理器，ARM Cortex-M4 带 FPU
<code>--apcs=interwork</code>	生成可以在任何 CPU 模式（ARM 或 Thumb）下调用的代码
<code>--split_sections</code>	为每个源文件的函数创建一个 section，方便在链接时去掉 .o 文件中不用的函数
<code>-D</code>	定义一个预处理器符号，并且可以选择将其设置为一个值
<code>-O0</code>	最小优化级别
<code>__MICROLIB</code>	目标选项中已启用使用 MicroLIB
<code>__UVISION_VERSION</code>	µVision 的主要版本和次要版本
<code>-g</code>	生成调试信息
<code>--wchar32</code>	定义 wchar_t 为 unsigned int
	头文件的路径

选项配置	描述
<code>-I path</code>	
<code>-DDEBUT_CMD_INTERRUPT</code>	使用中断接收命令
<code>-DATC_STP_SUPPORT</code>	支持 ATC SAFETY PACK
<code>-DATC_DEVICE_ASSERT</code>	支持 ATC 断言判断
<code>-DOS_PLATFORM=1</code>	支持在 OS 中使用 OSIF 接口
<code>-DAC7840X</code>	编译适配 AC7840x 的代码
<code>-DAC7840E</code>	编译适配 AC7840E 的代码
<code>-DAC7842X</code>	编译适配 AC7842x 的代码
<code>-DAC7843X</code>	编译适配 AC7843x 的代码
<code>-DEEP_MPU_PROTECTION</code>	支持 EEP 内存添加 MPU 保护

表 4.2 ARMCC 汇编器选项配置

选项配置	描述
<code>--cpu Cortex-M4.fp</code>	选择目标处理器，ARM Cortex-M4
<code>--apcs=interwork</code>	生成可以在任何 CPU 模式（ARM 或 Thumb）下调用的代码
<code>-g</code>	生成调试信息
<code>__UVISION_VERSION</code>	µVision 的主要版本和次要版本

表 4.3 ARMCC 链接器选项配置

选项配置	描述
<code>--cpu Cortex-M4.fp</code>	选择目标处理器，ARM Cortex-M4
<code>--split_sections</code>	为每个函数单独生成一个节
<code>--strict</code>	检查源文件是否符合 ANSI C
<code>--library_type=microlib</code>	目标选项中已启用使用 MicroLIB
<code>--scatter ""</code>	指定镜像描述文件
<code>--summary_stderr</code>	信息输出到标准错误流
<code>--info summarysizes</code>	显示代码和数据大小摘要

选项配置	描述
<code>-map</code>	生成映射文件
<code>-ref</code>	生成交叉引用列表
<code>-callgraph</code>	生成调用图
<code>-symbols</code>	输出包含符号信息
<code>-info sizes</code>	输出程序中各段的大小
<code>-info totals</code>	输出程序总大小
<code>-info unused</code>	输出未使用的代码和数据信息
<code>-info veneers</code>	给出链接产生 ARM/Thumb 的信息

4.1.3 GCC 编译配置

表 4.4 GCC 编译器选项配置

选项配置	描述
<code>-c</code>	为每个源文件生成一个目标文件（称为 input-file.o）
<code>--std=c99</code>	使用 c99 的代码标准
<code>-O1</code>	优化代码大小
<code>-gdwarf-2</code>	生成 gdwarf-2 格式的调试信息
<code>-mcpu=cortex-m4</code>	选择目标处理器：ARM Cortex M4
<code>-mfpu=fpv4-sp-d16</code>	生成支持 FPUv4-SP-D16 浮点处理器的代码
<code>-march=armv7e-m</code>	选择 armv7e-m 的架构
<code>-mthumb</code>	选择生成在 Thumb 状态下执行的代码
<code>-mthumb-interwork</code>	生成可以在 Thumb 和 ARM 指令集之间进行交叉调用的代码
<code>-mlittle-endian</code>	为在小端模式下运行的处理器生成代码
<code>-mfloat-abi=hard</code>	使用硬件浮点指令
<code>-Wall</code>	启用所有关于某些用户认为有问题的构造的警告
<code>-Werror</code>	所有警告都转成 error

选项配置	描述
<code>-D</code>	定义一个预处理器符号，并且可以选择将其设置为一个值
<code>-DGCC</code>	定义 GCC 预处理器符号
<code>-DDEBUT_CMD_INTERRUPT</code>	使用中断接收命令
<code>-DATC_STP_SUPPORT</code>	支持 ATC SAFETY PACK
<code>-DATC_DEVICE_ASSERT</code>	支持 ATC 断言判断
<code>-DOS_PLATFORM=1</code>	支持在 OS 中使用 OSIF 接口
<code>-DAC7840X</code>	编译适配 AC7840x 的代码
<code>-DAC7840E</code>	编译适配 AC7840E 的代码
<code>-DAC7842X</code>	编译适配 AC7842x 的代码
<code>-DAC7843X</code>	编译适配 AC7843x 的代码
<code>-DEEP_MPU_PROTECTION</code>	支持 EEP 内存添加 MPU 保护

表 4.5 GCC 编译器选项配置

选项配置	描述
<code>-c</code>	为每个源文件生成一个目标文件（称为 input-file.o）
<code>--std=c99</code>	使用 c99 的代码标准
<code>-O1</code>	优化代码大小
<code>-gdwarf-2</code>	生成 gdwarf-2 格式的调试信息
<code>-mcpu=cortex-m4</code>	选择目标处理器：ARM Cortex M4
<code>-mfpu=fpv4-sp-d16</code>	生成支持 Fpv4-SP-D16 浮点处理器的代码
<code>-march=armv7e-m</code>	选择 armv7e-m 的架构
<code>-mthumb</code>	选择生成在 Thumb 状态下执行的代码
<code>-mthumb-interwork</code>	生成可以在 Thumb 和 ARM 指令集之间进行交叉调用的代码
<code>-mlittle-endian</code>	为在小端模式下运行的处理器生成代码
<code>-mfloat-abi=hard</code>	使用硬件浮点指令
<code>-Wall</code>	启用所有关于某些用户认为有问题的构造的警告
<code>-Werror</code>	所有警告都转成 error

表 4.6 GCC 链接器选项配置

选项配置	描述
-Map=filename	将链接映射打印到文件 mapfile
-T scriptfile	使用 scriptfile 作为链接器脚本
-Wl,uXXX	强制链接某函数
-static	告诉链接器链接静态而不是动态链接
-specs=nosys.specs	告诉链接器将此二进制文件链接到 "nosys" 库

4.1.4 IAR 编译配置

表 4.7 IAR 编译器选项配置

选项配置	描述
--cpu=Cortex-M4	选择目标处理器：Arm Cortex M4
--endian=little	指定核心的字节顺序：小端
-c	为每个源文件生成一个目标文件（称为 input-file.o）
--no_clustering	禁用静态集群优化
--debug	使编译器包含目标模块中的信息
--no_cse	减少重复计算优化
-D	定义一个预处理器符号，并且可以选择将其设置为一个值
-D IAR	定义 IAR 预处理器符号
-D DEBUT_CMD_INTERRUPT	使用中断接收命令
-D ATC_STP_SUPPORT	支持 ATC SAFETY PACK
-D __DLIB_FILE_DESCRIPTOR	使用 IAR 自带的重定向函数
--no_unroll	禁止循环展开
--no_inline	禁止内联函数
--no_code_motion	禁用代码移动优化
--no_tbaa	禁用类型基础的别名分析
--no_scheduling	禁用指令调度优化

选项配置	描述
<code>--fpu=VFPv4_sp</code>	设置浮点单元 (FPU) 为 VFPv4 单精度
<code>--dlib_config</code>	支持 IAR 的标准库
<code>--warnings_are_errors</code>	警告视为错误
<code>-e</code>	启用语言扩展
<code>-DATC_DEVICE_ASSERT</code>	支持 ATC 断言判断
<code>-DOS_PLATFORM=1</code>	支持在 OS 中使用 OSIF 接口
<code>-DAC7840X</code>	编译适配 AC7840x 的代码
<code>-DAC7840E</code>	编译适配 AC7840E 的代码
<code>-DAC7842X</code>	编译适配 AC7842x 的代码
<code>-DAC7843X</code>	编译适配 AC7843x 的代码
<code>-DEEP_MPU_PROTECTION</code>	支持 EEP 内存添加 MPU 保护

表 4.8 IAR 汇编器选项配置

选项配置	描述
<code>--cpu=Cortex-M4</code>	选择目标处理器: Arm Cortex M4
<code>-g</code>	禁用自动搜索系统包含文件

表 4.9 IAR 链接器选项配置

选项配置	描述
<code>--cpu=Cortex-M4</code>	选择目标处理器: Arm Cortex M4
<code>--map filename</code>	生成 map 文件
<code>--no_out_extension</code>	禁止输出文件添加默认扩展名
<code>--entry __iar_program_start</code>	将符号 <code>__iar_program_start</code> 视为根符号和应用程序的开始
<code>--semihosting</code>	链接静态库
<code>--vfe</code>	启用向量浮点扩展
<code>--text_out</code>	指定代码输出的路径
<code>--config</code>	指定链接器要使用的配置文件

选项配置	描述
<code>--fpu=VFPv4_sp</code>	设置浮点单元（FPU）为 VFPv4 单精度

4.1.5 GHS 编译配置

表 4.10 GHS 目标选项配置

选项配置	描述
<code>-cpu=cortexm4f</code>	ARM Cortex-M4F 内核
<code>-G</code>	调试版本
<code>-bsp generic</code>	使用 generic bsp
<code>-littleendian</code>	小端字节序
<code>-fhard</code>	支持硬件浮点计算
<code>--gnu_asm</code>	兼容 GNU 扩展的汇编语法
<code>-thumb</code>	生成 16 位的 thumb 代码
<code>-thumb_lib</code>	运行时链接 thumb 代码的库

表 4.11 GHS 工程选项配置

选项配置	描述
<code>-object_dir=directory</code>	目标文件输出目录
<code>-Idirectory</code>	头文件路径
<code>-Llibrary_directory</code>	库文件目录
<code>-llibrary</code>	链接库文件
<code>-o filename</code>	目标文件或可执行文件输出名称
<code>-align8</code>	数据按照 8 字节对齐

表 4.12 GHS 优化选项配置

选项配置	描述
<code>-Odebug</code>	启用调试优化

选项配置	描述
<code>-Onoinline</code>	不支持模块间内联
<code>-Onovector</code>	不启用向量优化
<code>-Onolink</code>	不启用链接优化
<code>-Onoipa</code>	不启用过程间优化

表 4.13 GHS 编译器选项配置

选项配置	描述
<code>-c99</code>	模式编译规范采用 ISO C99
<code>-pack=none</code>	结构体类型按照最大对齐方式
<code>-align8</code>	数据对象按照 8 字节对齐
<code>--gnu_asm</code>	启用 GNU 扩展的 asm 语法
<code>-DATC_DEVICE_ASSERT</code>	支持 ATC 断言判断
<code>-DDEBUG_CMD_INTERRUPT</code>	使用中断接收命令
<code>-DATC_STP_SUPPORT</code>	支持 ATC SAFETY PACK
<code>-DOS_PLATFORM=1</code>	支持在 OS 中使用 OSIF 接口
<code>-DAC7840X</code>	编译适配 AC7840x 的代码
<code>-DAC7842X</code>	编译适配 AC7842x 的代码
<code>-DAC7843X</code>	编译适配 AC7843x 的代码
<code>-DAC7840E</code>	编译适配 AC7840E 的代码
<code>-DEEP_MPU_PROTECTION</code>	支持 EEP 内存添加 MPU 保护

表 4.14 GHS 汇编器选项配置

选项配置	描述
<code>-no_list</code>	不产生文件列表

表 4.15 GHS 链接器选项配置

选项配置	描述
<code>-hex=name</code>	生成格式为 Hex 的文件
<code>-nostrip</code>	不去除符号表以及调试信息
<code>-I</code>	链接非标准链接指令文件

4.1.6 最小系统集成与编译

SDK 最小系统所需静态文件如下：

模块	静态文件位置	描述
BASE	源文件： <code>mcu\base\src</code> 头文件： <code>mcu\base\include</code>	编译和平台基础实现相关
Device	源文件： <code>mcu\device\src</code> 头文件： <code>mcu\device\include</code>	平台驱动代码
HAL	源文件： <code>mcu\hal\src\mcu</code> 源文件： <code>mcu\hal\src\gpio</code> 头文件： <code>mcu\hal\include</code>	系统 Core、时钟、GPIO相关的组件实现

以 KEIL 集成最小系统工程为例，步骤如下：

1. 在 KEIL 中新建工程，添加平台文件：`startup_ac784xx.s`、`System_AC784xx.c`、`Debugout_AC784xx.c`、`main.c`。
2. 在 KEIL 工程中添加静态文件：Device 组件、HAL 组件、BASE 组件。
3. KEIL 工程编译无报错后，便搭建好 AC784xx SDK 的最小系统开发环境，接下来可根据需求新增模块开发与测试。

4.2 工作动态参数配置

本节描述运行期会变化、需由软件读取或更新的动态参数。此类参数不建议写死在应用逻辑中，应通过 HAL/API 实时获取。

4.2.1 系统时钟动态参数

参数	来源	说明
<code>SystemCoreClock</code>	<code>System_AC784xx.h</code> 全局变量	当前内核时钟频率缓存值
<code>SystemCoreClockUpdate()</code>	<code>System_AC784xx.c</code>	调用 <code>Ckgen_Hal_GetFreq(CKGEN_CORE_CLK, ...)</code> 刷新 <code>SystemCoreClock</code>

建议：在调用时钟切换接口后，立即调用 `SystemCoreClockUpdate()`，避免延时和通信波特率计算使用旧值。

4.3 默认 section 配置

默认 section 配置由启动文件与链接脚本共同决定，影响向量表位置、代码段/数据段布局、堆栈大小和 RAM 初始化行为。

4.3.1 启动文件默认 section（ARMC 示例）

以 `device/src/armc/startup_ac7840x.s` 为例，默认定义如下：

section/区域	定义方式	说明
<code>STACK</code>	<code>AREA STACK, NOINIT, READWRITE</code>	默认栈区， <code>Stack_Size = 0x00002000</code>
<code>HEAP</code>	<code>AREA HEAP, NOINIT, READWRITE</code>	默认堆区， <code>Heap_Size = 0x00000000</code>
<code>RESET</code>	<code>AREA RESET, DATA, READONLY</code>	中断向量表与 <code>Reset_Handler</code> 入口
<code> .text </code>	<code>AREA .text , CODE, READONLY</code>	代码区

`Reset_Handler` 默认调用顺序（该文件可见）为：先 `SystemInitRam`，再 `SystemInit`，最后跳转 `__main`。

4.3.2 链接脚本默认 section（GCC 示例）

以 `test/QualificationTest/Template/gccProject/AC7840x_flash_gcc.ld` 为例：

section	主要符号/用途
<code>.text</code>	向量表与代码只读区
<code>.data</code>	已初始化数据，配套符号 <code>__DATA_ROM/ __DATA_RAM/ __DATA_END</code>
<code>.code</code>	RAM 中执行代码段，配套符号 <code>__CODE_ROM/ __CODE_RAM/ __CODE_END</code>
<code>.bss</code>	未初始化数据，配套符号 <code>__BSS_START/ __BSS_END</code>
<code>.heap</code>	堆区， <code>HEAP_SIZE</code> 默认 0
<code>.stack</code>	栈区， <code>STACK_SIZE</code> 默认 <code>0x0000800</code>
<code>.dma</code>	DMA 或 RAM 代码相关区

4.3.3 与系统初始化的关联

`SystemInitRam()` 通过链接符号完成 RAM 初始化。根据 `System_AC784xx.c` 的条件编译实现，不同工具链对 `SystemCopyTable` / `SystemZeroTable` 的处理方式不同：GCC、GHS 路径下，这两个表会直接关联链接脚本导出的地址符号，表项的有效范围由链接脚本中的 section 边界决定；ARMCC、IAR 路径下，代码中这两个表的占位项会被初始化为 `0U`，工程集成时通常配合 `SystemCopyTable`、`SystemZeroTable` 等链接配置将其 size 视为 0，避免与工具链默认初始化流程重复。

关键行为包括：

1. 基于 `SystemCopyTable` 将 `__DATA_ROM -> __DATA_RAM`、`__CODE_ROM -> __CODE_RAM` 拷贝到 RAM；
2. 基于 `SystemZeroTable` 对 `__BSS_START ~ __BSS_END` 清零；
3. 在特定模式下保留 retention SRAM，不做全量清零。

说明：GCC、GHS 工程应重点核对链接脚本中 `__DATA_ROM/ __DATA_RAM/ __DATA_END`、`__CODE_ROM/ __CODE_RAM/ __CODE_END`、`__BSS_START/ __BSS_END` 等符号及其 size 是否正确；ARMCC、IAR 工程除核对 scatter/linker 配置外，还应确认 `SystemCopyTable` / `SystemZeroTable` 的 size 被正确处理为 0，避免在 `SystemInitRam()` 中发生重复拷贝或重复清零。若项目使用非模板链接配置，请以工程实际 `.ld/.icf/.lsl/.sct` 文件为准，对齐 section 划分与符号名。

5. 系统启动与电源管理

本章描述 AC784xx 系列 MCU 的系统启动流程与电源管理机制，包括启动文件行为、系统初始化函数职责、SRAM 初始化策略、电源模式切换与待机唤醒配置等内容。

5.1 系统启动流程

AC784xx 的启动代码由汇编启动文件（`startup_ac784xx.s`）与 C 运行时初始化函数共同构成，支持 ARMCC、GCC、GHS、IAR 四种编译器工具链，启动文件分别位于 `device/src/armcc/`、`device/src/gcc/`、`device/src/ghs/`、`device/src/iar/` 目录下。

5.1.1 启动流程概述

芯片上电或复位后，硬件自动将程序计数器设置到中断向量表中 `Reset_Handler` 的地址，随后依次执行以下步骤：

步骤	函数/操作	说明
1	<code>Reset_Handler</code> (汇编)	设置主栈指针 (MSP)，加载 <code>.data</code> 段和 <code>.bss</code> 段的起止地址
2	<code>SystemInitRam()</code>	SRAM ECC 初始化；将 ROM 中的初始化数据拷贝到 RAM；BSS 段清零；向量表复制（如启用 <code>VECT_TAB_SRAM</code> ）
3	<code>SystemInit()</code>	关闭看门狗（可配置）；使能 FPU；配置 SRAM ECC；配置 ICache/DCache（AC7843 专属）；重定向中断向量表（VTOR）
4	<code>main()</code>	进入用户应用层入口函数

5.1.2 启动相关 API

API	函数 ID	说明
<code>SystemInit()</code>	DES_BOOT_API_000	系统初始化：关闭 WDG、使能 FPU、配置 Cache、重定向 VTOR
<code>SystemCoreClockUpdate()</code>	DES_BOOT_API_001	从 CKGEN HAL 读取当前 CORE_CLK 频率，更新全局变量 <code>SystemCoreClock</code>

API	函数 ID	说明
<code>SystemInitRam()</code>	DES_BOOT_API_002	SRAM ECC 初始化 + ROM→RAM 数据拷贝 + BSS 清零
<code>System_Memset()</code>	DES_BOOT_API_014	内存填充工具函数
<code>System_Memcpy()</code>	DES_BOOT_API_013	内存拷贝工具函数
<code>System_FlsDeviceTryLock()</code>	DES_BOOT_API_010	DFlash 设备锁定（防多模块并发访问）
<code>System_FlsDeviceUnlock()</code>	DES_BOOT_API_011	DFlash 设备解锁
<code>System_FlsDeviceFreeLock()</code>	DES_BOOT_API_012	DFlash 设备强制解锁

5.2 SystemInit 行为说明

`SystemInit()` 是芯片上电后在进入 `main()` 之前由启动文件自动调用的系统初始化函数，完成以下工作：

5.2.1 看门狗处理

`SystemInit()` 中包含关闭看门狗的逻辑，通过写入解锁序列并清除使能位来禁用 WDG。该行为受编译宏 `WDG_DISABLE` 控制：

宏	默认值 (<code>System_AC784xx.c</code>)	说明
<code>WDG_DISABLE</code>	<code>1</code> （宏为 1 时在 <code>SystemInit</code> 中关闭 WDG）	设为 <code>0</code> 则保留 WDG 默认状态，由应用层自行管理

注意：如果项目需要 WDG 保持使能以满足功能安全要求，必须将 `WDG_DISABLE` 设置为 `0`，并在应用层完成 WDG 初始化配置后立即喂狗，避免启动期间复位。

5.2.2 SRAM ECC 初始化配置

SRAM ECC 相关初始化受以下宏控制：

宏	默认值	说明
<code>SRAM_ECC_READ_ENABLE</code>	<code>1</code>	使能 SRAM L 和 SRAM U 的 ECC 读校验；上电时须先写全零完成 ECC 种子填充

宏	默认值	说明
SRAM_ECC_ERR_RST_ENABLE	0	使能 ECC 双 bit 错误自动复位；默认关闭，由应用层处理 ECC 故障
NOT_INIT_SRAM_AFTER_RESET	1	在非 POR 复位后跳过 SRAM 重新初始化（通过标志位记录，避免不必要的清零开销）

5.2.3 Cache 配置

AC7843 具有 ICache 和 DCache，`SystemInit()` 中提供对应的使能/禁用控制宏：

宏	默认值	说明
ICACHE_DISABLE	0（使能）	设为 1 则在 <code>SystemInit</code> 中禁用 ICache
ICACHE_LINE_16BYTE	1	ICache 行大小使用 16 字节（默认 32 字节），根据代码密度选择
DCACHE_DISABLE	1（禁用）	默认禁用 DCache；使能前须确保所有 DMA 缓冲区已正确配置 Cache 属性

AC7843 专属：上述 Cache 宏仅对 AC7843 生效，AC7840 / AC7840E / AC7842 无 Cache 硬件，相关宏不产生任何效果。

5.2.4 中断向量表重定向（VTOR）

`SystemInit()` 中通过写入 SCB->VTOR 完成中断向量表的重定向：

- 默认情况下（未定义 `VECT_TAB_SRAM`）：向量表指向 Flash 中的 `__Vectors` 符号地址；
- 若定义 `VECT_TAB_SRAM`：向量表被重定向到 SRAM 中的固定地址（`VECTOR_TABLE_SRAM_ADDR`）。

各型号的 SRAM 向量表地址：需要根据实际使用修改 `device/src/armc/system_ac784xx.c` 文件中的 `VECTOR_TABLE_SRAM_ADDR` 宏定义。

5.3 SRAM 初始化策略

`SystemInitRam()` 在不同复位场景下的 SRAM 清零策略存在差异，具体如下：

5.3.1 SRAM 分区

AC784xx 片内 SRAM 分为两个区域：

区域	基地址标识	说明
SRAM_L	SRAM_L_BASE	低地址 SRAM 段
SRAM_U	SRAM_U_BASE	高地址 SRAM 段

其中低地址区末端保留一段 **Retention SRAM** (`SRAM_RETENTION_BASE` ~ `SRAM_RETENTION_BASE + SRAM_RETENTION_SIZE`) ，在待机 (STANDBY) 模式唤醒后该区域内容不清零，用于保存必要的跨唤醒状态数据。

5.3.2 初始化策略对照表

复位场景	SRAM 初始化行为
POR / LVR (首次上电或欠压复位)	全部 SRAM (SRAM_L + SRAM_U) 清零，ECC 种子全部重写
STANDBY 模式唤醒	跳过 Retention SRAM 区域，仅清零非 Retention 区域 (保留用户跨唤醒数据)
其他软件/硬件复位	依据 <code>NOT_INIT_SRAM</code> 宏决定是否跳过清零 (AC7843 默认跳过，由 BootROM 处理)

AC7843 专属：AC7843 由 BootROM 负责 SRAM 初始化，`SystemInitRam()` 中 AC7843 路径下 `NOT_INIT_SRAM = 1`，用户层无需重复清零。

5.3.3 FlexRAM (AC7840 / AC7842)

上述三款型号含有 FlexRAM (可作为 EEPROM 仿真用途)。当 FlexRAM 未被 CSE 模块占用时，`SystemInitRam()` 会对其进行清零初始化。

5.3.4 缩短 SRAM 初始化时间 (加快启动)

`SystemInitAllRam()` 在 POR/LVR 场景下会对全片 SRAM (`SRAM_L_BASE` ~ `SRAM_U_END` ，跳过栈区) 进行 8 字节对齐清零以完成 ECC 种子填充，耗时与 SRAM 总量正相关。客户可根据项目需求选择以下三种方式缩短初始化时间：

方式一：利用跨复位跳过重复初始化 (推荐)

`NOT_INIT_SRAM_AFTER_RESET` 宏 (`System_AC784xx.c` 中默认为 `1U`) 会在首次 POR 初始化完成后向 `CKGEN->RCM_EN` 写入 `CKGEN_RCM_EN_NOT_INIT_SRAM_Msk` 标志, 此后除 POR / LVR / 待机唤醒外的复位 (软件复位、外部复位等) 均会跳过 SRAM 清零, 启动速度显著提升。该标志由硬件在 POR 时自动清除, 不影响上电首次初始化的安全性。

适用型号: AC7840E、AC7842、AC7843 (AC7840 无此硬件标志位, 不适用)。

方式二: 完全跳过 SDK SRAM 初始化

将 `NOT_INIT_SRAM` 宏在工程中定义为 `1U` 可使 `SystemInitAllRam()` 完全不执行清零操作。此方式要求 SRAM ECC 种子已由其他机制保证 (例如 AC7843 的 BootROM 在 SDK 介入前已完成全片初始化)。在其他型号上使用此方式须自行负责 ECC 完整性。

5.4 电源管理

AC784xx 片内包含 SPM 硬件, 用于实现芯片的电源管理功能, SPM 功能的实现请参考勘误手册和 API 文档。

5.5 注意事项

- 启动文件与编译器必须匹配:** `device/src/` 下分为 `armc/`、`gcc/`、`ghs/`、`iar/` 四个子目录, 分别对应不同工具链的启动文件, 禁止跨目录混用。
- SRAM Retention 区域的保留条件:** 只有在进入 STANDBY 模式并通过唤醒源正常唤醒时, Retention SRAM 内容才会被保留; 若发生 POR / LVR 复位, 即使是 STANDBY 唤醒后触发的 POR 也会导致全部 SRAM 清零。
- SystemCoreClockUpdate 需在时钟切换后调用:** `SystemCoreClock` 全局变量不会自动更新, 调用 `Ckgen_Hal_SetSysClk()` 切换系统时钟后必须显式调用 `SystemCoreClockUpdate()` 刷新该变量, 否则依赖 `SystemCoreClock` 的延时计算或波特率计算将出错。
- AC7843 HSM 固件就绪等待:** AC7843 在 `SystemInitRam()` 内部会检查 HSM 固件是否正在更新 (`SystemWaitFwIdle()`), 若 HSM 固件处于更新状态将阻塞等待。确保 HSM 固件在正式产品中处于正常状态, 避免启动死锁。
- DCache 使能须谨慎 (AC7843):** AC7843 的 DCache 默认禁用 (`DCACHE_DISABLE = 1`), 若需使能, 必须确保所有 DMA 传输缓冲区已配置为 Non-cacheable 或在传输前后进行 Cache 维护操作, 否则可能导致数据一致性问题。

6. 系统烧写与升级

本章节只介绍我司开发及测试过程中使用的烧写方式；升级未涉及到。

7. 外设适配与调试

本项目暂未适配过其它外设（如：EWDG、Flash 等）。

8. 系统调试与性能分析

8.1 调试开关（Device Assert）

SDK 中大量参数检查通过 `DEVICE_ASSERT(...)` 实现。若希望在开发调试阶段快速定位非法参数、空指针和状态机错误，需在编译选项中开启 `ATC_DEVICE_ASSERT`。

`Device_Assert.h` 中的行为如下：

- 开启 `ATC_DEVICE_ASSERT`：`DEVICE_ASSERT` 失败后调用 `WaitingForDebug(__FILE__, __LINE__)`
- 未开启：`DEVICE_ASSERT` 退化为空操作（仅保留表达式）

默认 `WaitingForDebug` 在 `device/src/System_AC784xx.c` 中提供弱定义，行为为打印断言文件/行号并进入死循环等待调试器介入。

建议：

1. Debug 版本开启 `ATC_DEVICE_ASSERT`，用于问题快速暴露。
2. Release 版本关闭 `ATC_DEVICE_ASSERT`，避免断言停机影响量产运行。

8.2 Exception 客户对接

异常处理入口位于 `device/src/Exception_AC784xx.c`，默认实现包含：

- `HardFault_Handler()`：HardFault 统一入口。
- `HardFault_Custom_Callback()`：弱符号回调，供客户覆盖。

客户对接建议流程:

1. 在用户代码中重写 `HardFault_Custom_Callback()`，采集现场信息（任务上下文、关键寄存器、错误计数等）。
2. 在回调中写入最小化故障记录（如 RAM 日志或保留区），避免复杂阻塞操作。
3. 根据项目策略决定后续动作：复位、进入安全状态、或保持停机等待诊断。

说明:

- 若使能 `EEP_MPU_PROTECTION`，默认 `HardFault_Handler` 中会尝试读取/清理 MPU 错误信息。
- 未使能该保护时，默认逻辑进入 `for(;;)` 循环。

9. 定制应用

暂无（未来加上bootloader及其它）

10. 用户开发接口

见代码包doc目录下

11. 用户开发示例

见项目单独释放的demo

12. 参考文档

无