ZS110A-SDK 架构介绍

发布 ***1.0.0***

2018 年 **11** 月 **02** 日

目录

CHAPTER 1

文档介绍

1.1 文档目的

ZS110A 是基于炬芯科技 BLE 芯片 ATB110X 开发的 SDK，供用户快速评估学习 BLE 功能开发。

1.2 术语说明

表 1.1: 术语说明

术语	说明
ZEPHYR	为所有资源受限设备，构建了针对低功耗、小型内存微处理器设备而进行优化的物联网嵌入式小型、可扩展的实时操作系统（RTOS）
XIP	eXecute In Place，指不需要将代码加载到 RAM 而是直接在存储介质上运行代码
BROM	BOOT-ROM，实现了引导存储设备上的程序和通过 UART 将程序烧写到存储设备的功能
NVRAM	有 2 层含义：1）指掉电后不会丢失数据的介质；2）在该介质上读写的驱动
TICKLESS_IDLE	在 idle 中关闭 tick 时钟，已达到省电的目的

1.3 参考文档

- http://docs.zephyrproject.org/

1.4 版本历史

表 1.2: 版本历史

日期	版本	注释	作者
2018-08-22	1.0	初始版本	ZS110A 项目组

CHAPTER 2

SDK 概述

2.1 SDK 架构

本 SDK 层级结构如下图所示：

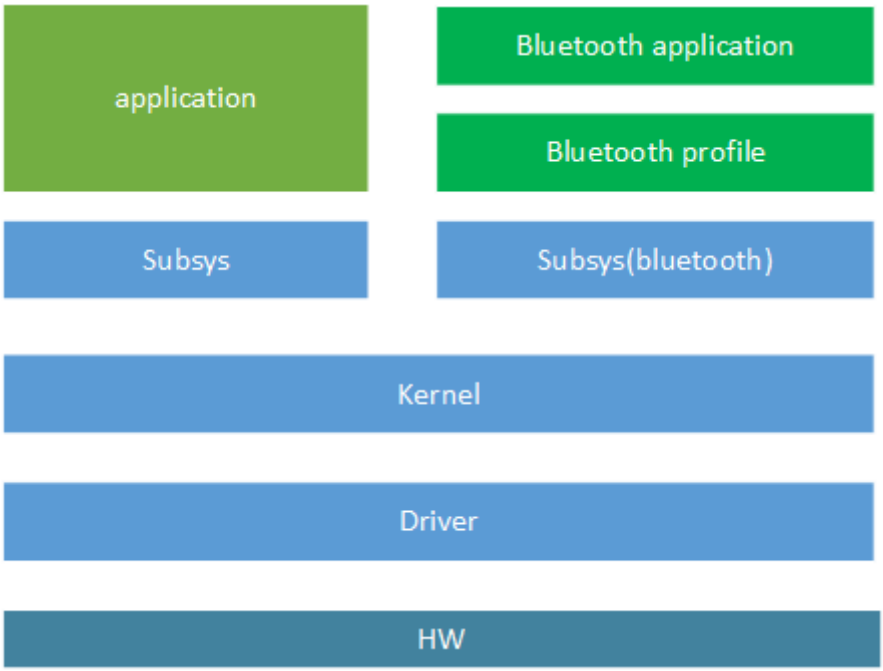


图 2.1: SDK 层次图

SDK 按层次结构从下至上可以分为:

- **驱动层驱动层 (driver)** 实现各个硬件模块的驱动。

- **内核层 (kernel)** 定义了基本的 kernel 服务：任务调度、通信机制、内存管理等。
- **子系统层 (subsys)** 在内核层之上构建了功能子系统，包括 shell/bluetooth 等。
- **应用层 (application)**。 可以基于 bluetooth subsys 开发各种 BLE 应用。

2.2 应用开发模式

为了适应各种复杂度的应用开发，IC 设计时实现了以下几点：

- 设计了大容量 ROM，固化大量 driver/kernel/subsys（bluetooth）代码
- 支持 spinor XIP

根据应用程序的复杂度和是否使用 ROM，应用开发有以下三种模式：

- ROM+RAM: 对于可以使用 ROM，并且比较简单应用。应用的所有代码和数据都从 spinor 中 load 到 ram 运行。

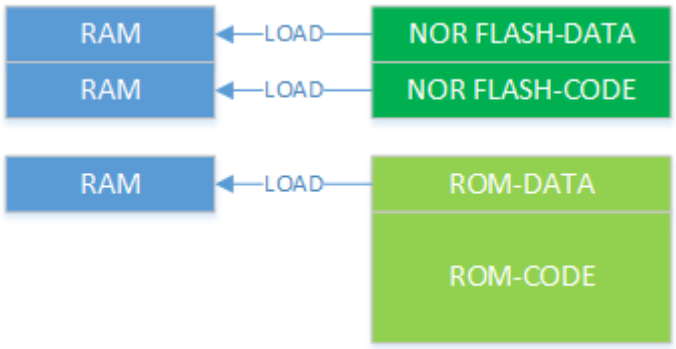


图 2.2: rom+ram+nor 模式

- ROM+XIP+RAM: 对于可以使用 ROM，但逻辑很复杂的应用，RAM 已经不够了，此时需要使用 XIP 机制，代码直接在 spinor 上执行，ram 中只需要存储数据。

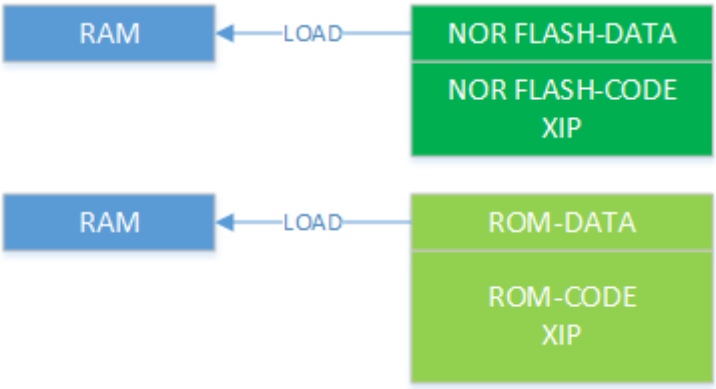


图 2.3: rom+xip+ram 模式

- XIP+RAM: 对于 ROM 完全派不上用场的情况，只能使用 XIP 机制。

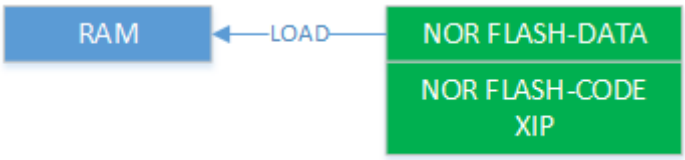


图 2.4: xip+ram 模式

2.3 目录结构

```
├─arch: soc 相关代码
│   ├─cortex_m
│   └─soc
│       └─atb110x
├─boards: 板级相关代码
├─drivers: 驱动源码，包括 rom 化的驱动的初始化代码
├─ext:
│   └─lib
│       └─actions: 炬芯开发的库文件
│           ├─framework：应用框架
│           ├─hal：驱动抽象层
│           ├─include：库相关的头文件
│           ├─libAL：算法库，以 lib 方式提供
│           ├─ota：ota 源码
│           └─startup：基于 keil 开发 zephyr 应用的 startup 公共代码
├─include：  kernel 和驱动相关的头文件
├─kernel：  内核源码，包括 rom 化的内核的初始化代码
├─lib: 只留下必须的 minimal libc 的头文件
├─samples:
│   ├─bluetooth: ble 应用示例，包括 peripheral 示例和 central 示例
│   ├─loader_binaries: loader 示例:loader 根据不同的参数加载对应的文件
│   ├─peripheral: 各种设备驱动的示例
│   ├─uart_product: 用于示范如何将一个普通示例转换成一个可用于 uart 量产的固件
│   └─voice_rcu: 可量产的 ble 遥控器方案
├─scripts：脚本和工具
└─subsys：子系统源码，包括 rom 化的子系统的初始化代码
                ├─bluetooth
                └─shell
```

CHAPTER 3

ROM 说明

ATB110X 内部有 160K ROM，由 BROM 和 zephyr-Rom 组成。

图 3.1: ROM 结构

ROM 空间分布如下：

表 3.1: ROM 空间分布

模块	起始地址	长度	ram 起始地址	长度
BROM	0x0	0x4000	0x2000_9f00	0x100
Zephyr-Rom	0x4000	0x24000	0x2000_0000	0x1000

注解： BROM 占用的 ram 在引导应用之后可以被应用复用。zephyr-rom 的 ram 空间必须保留，除非应用完全不依赖 zephyr。

3.1 BROM

BROM 实现了引导和烧写固件两个功能：

- 引导存储在 spinor 上的固件。
- 通过 UART0(GPIO2/GPIO3) 接口与小机通信，将固件烧写到 spinor。

BROM 运行可以分为以下几个阶段：

- spi0_launcher
- spicache_launcher
- uart_adfu

3.1.1 spi0_launcher

spi0_launcher 支持的规格如下：

- spinor 接在 gpio14/15/16/17 上的，支持 3wire 模式。
- 可以从 spinor 的 offset=0x0 和 offset=0x1000 两个地址引导。
- 支持 spinor 上的数据经过 randomizer 处理。

3.1.2 spicache_launcher

spicache_launcher 支持的规格如下：

- spinor 接在 gpio14/15/16/17 上的。
- 可以从 spinor 的 offset=0x0 和 offset=0x1000 两个地址引导。
- 支持 spinor 上的数据经过 randomizer 处理。

3.1.3 uart_adfu

spinor 引导失败后自动进入 uart_adfu 模式。uart_adfu 在和 pc 量产工具握手成功后进行固件量产。

3.2 Zephyr-ROM

zephyr-ROM 不会自动运行，通过 map 方式导出接口给应用 keil 工程。keil 工程通过接口启动 zephyr kernel，并使用 zephyr 提供的服务。

CHAPTER 4

分区说明

以 ATB1103 参考方案为例，ATB1103 内封的 512KBytes spinor 被划分成以下几个分区

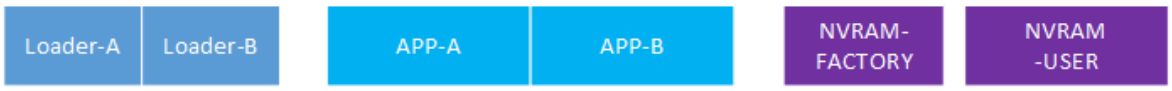


图 4.1: 分区划分

分区地址和长度定义如下：

表 4.1: 分区定义

partition	offset	Length
Loader-A	0x0	0x1000
Loader-B	0x1000	0x1000
App-A	0x1_0000	0x2_0000
App-B	0x3_0000	0x2_0000
NVRAM-Factory	0x7_0000	0x2000
NVRAM-User	0x7_2000	0x4000

注解: Loader 分区的地址必须是 0x0 或 0x1000。其他分区可以根据实际情况动态调整。

Loader-A/Loader-B:

- 引导分区，用于引导 App。

- 固件量产时只会烧写 Loader-A 分区。OTA 升级时会切换 A/B 分区。具体细节参考 OTA 实现。

App-A/App-B:

- 存储应用的分区。
- 固件量产时只会烧写 App-A 分区。OTA 升级时会切换 A/B 分区。具体细节参考 OTA 实现。

NVRAM-Factory：

- 该分区存储出厂设置。
- 建议：不直接修改出厂配置，而是在 User 分区创建同名配置项覆盖。

NVRAM-User：

- 该分区存储运行时动态修改的配置项。
- 建议：优先通过宏定义的方式实现配置项，尽量减少 NVRAM-User 的使用。

CHAPTER 5

启动流程

BROM 支持从 0x0 和 0x1000 两个地址引导 spinor 上的 App。

如果应用程序本身比较复杂（例如：需要支持 ota 功能），App 实现时可以再增加一级引导。

以参考方案为例：该应用拆分成了 Loader 和 App 两个程序，Loader 负责引导 App。

此时整个启动流程可以分为 2 个阶段：

- BROM 引导 Loader
- Loader 引导 App

5.1 BROM 引导 Loader

基础知识：

- spi0 上连接 spinor 可能有 2 种连接方式：正常的 4 线连接方式和的 3 线连接方式 (miso/mosi 连在一起)。
- spinor 上存储的数据可能是原始数据，也可能是经过 randomizer 后的加密数据。
- spinor 上会有 2 个 loader 分区，分别位于地址 0x0 和地址 0x1000。

Loader 镜像文件的结构如下：

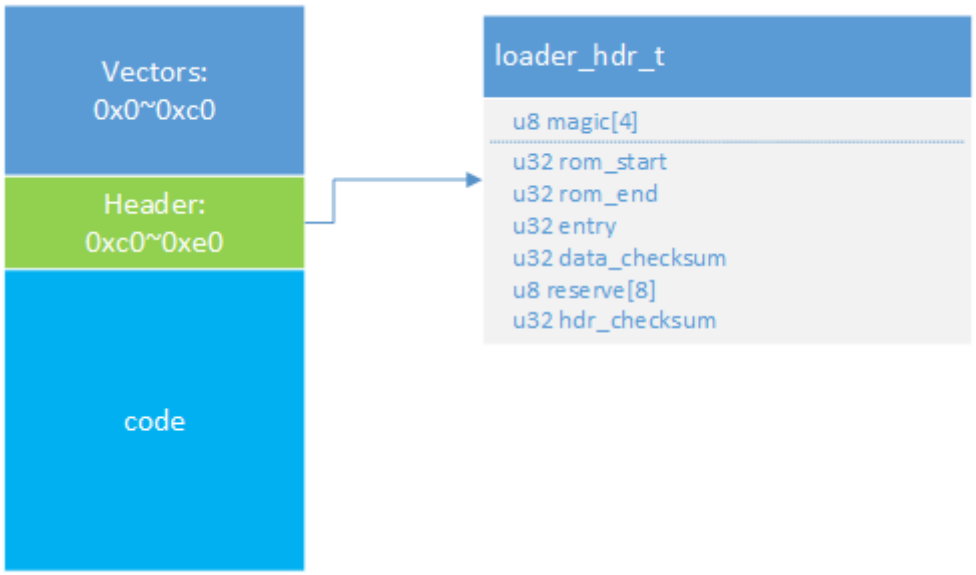


图 5.1: 镜像文件数据结构

说明：

- magic：固定为’A’‘B’‘L’‘E’。
- rom_start: load 到 ram 的起始地址。
- rom_end: load 到 ram 的结束地址。
- entry: 入口函数地址。
- data_checksum：除 hdr 外其他数据的校验和。
- hdr_checksum：hdr 的校验和。

brom 引导 loader 的流程如下：

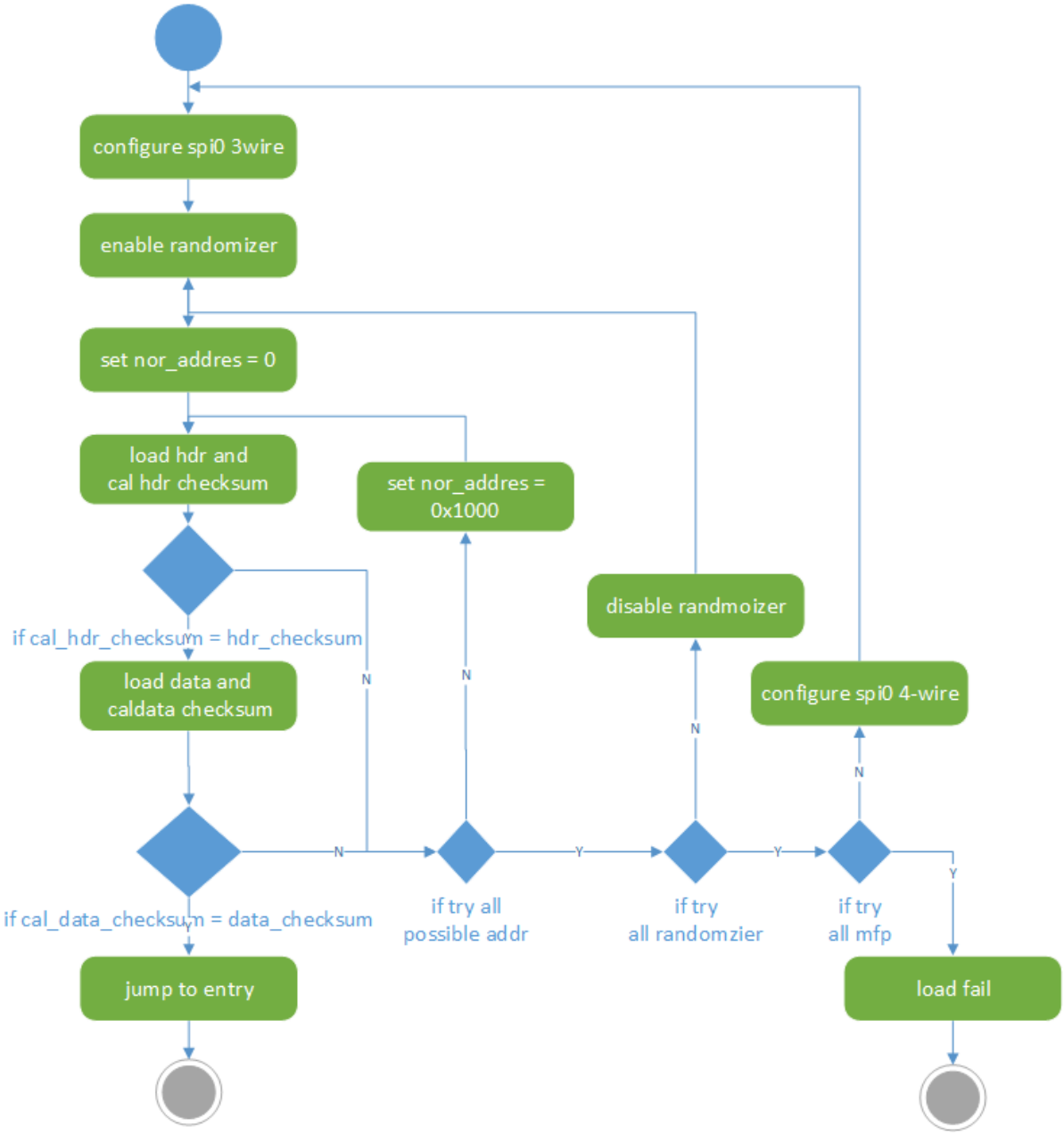


图 5.2: brom 引导 loader 流程

5.2 Loader 引导 App

基础知识：

- spinor 上可能会有 2 个 app 分区，分区的地址需要从分区表中获取。
- app 镜像文件的结构和 loader 相同。

Loader 引导 App 的流程如下：

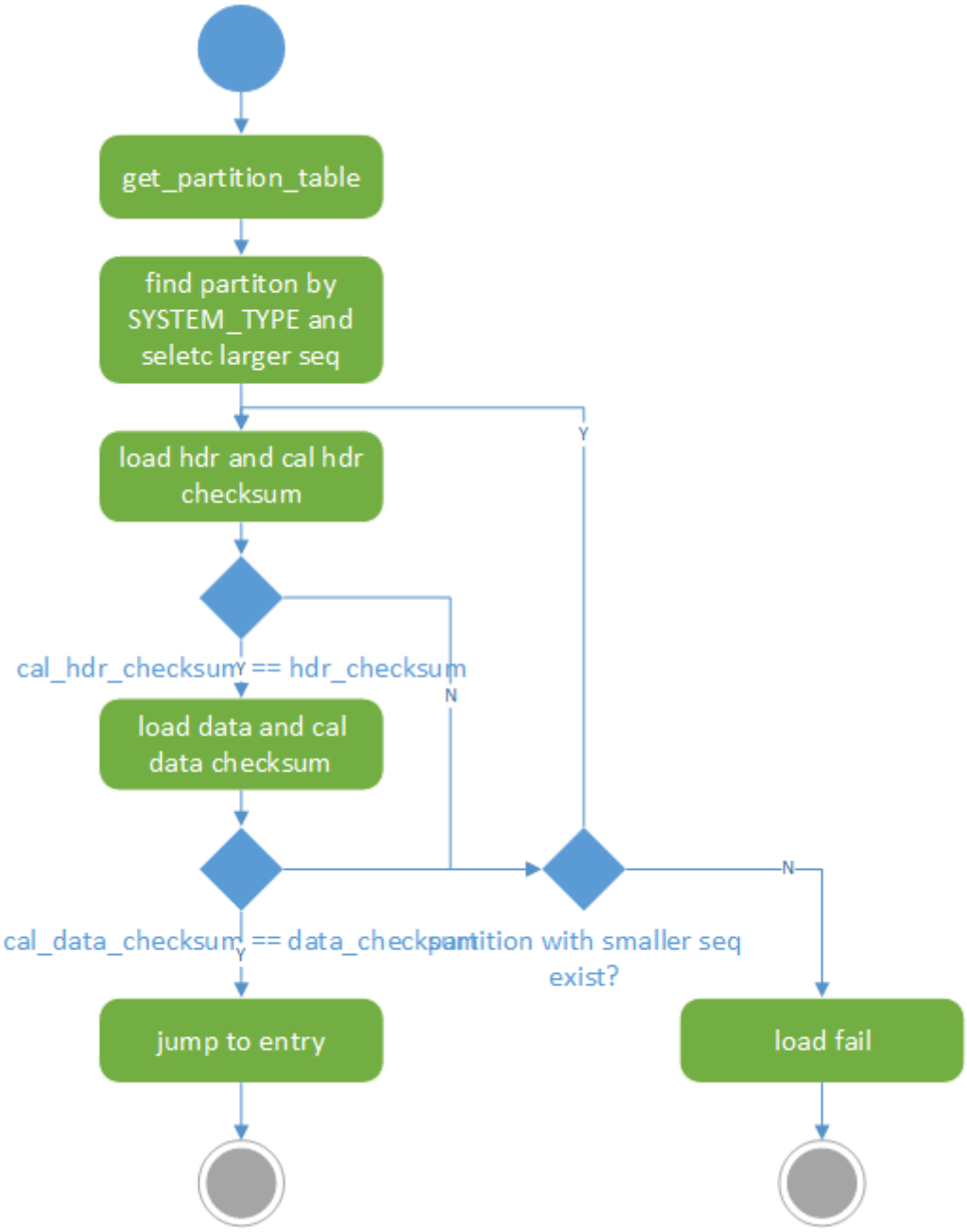


图 5.3: Loader 引导 App 流程

CHAPTER 6

zephyr 应用开发

6.1 App 启动 zephyr kernel

app 启动 zephyr kernel 的流程如下：

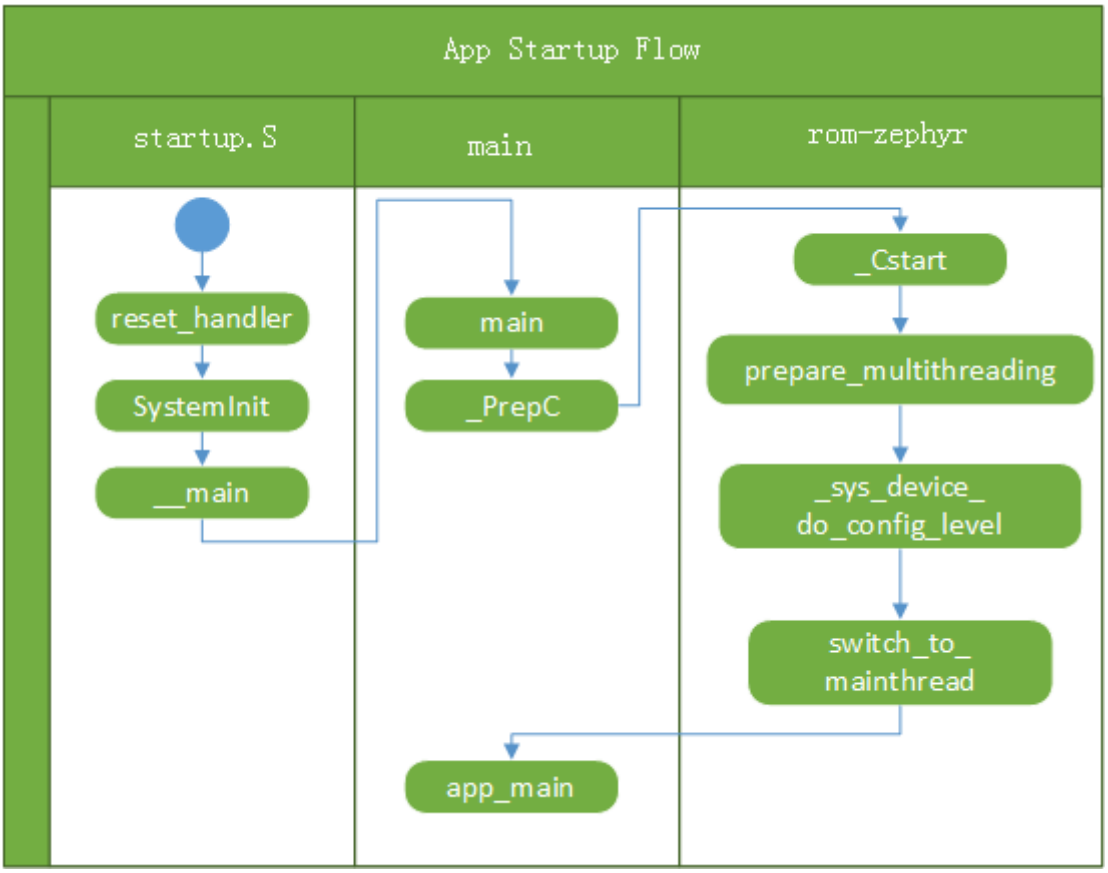


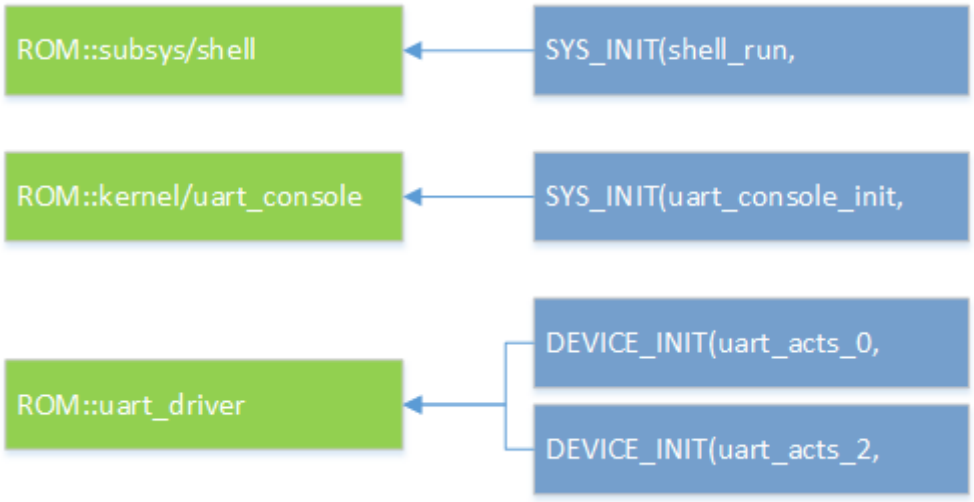
图 6.1: app 启动 zephyr 流程

说明：

- keil 工程标准引导流程:reset_handler->System_Init->__main->main
- main: 调用 Rom 导出的接口 _PrepC。_PrepC 将完成建立完整的 zephyr 环境，包括：
  - 准备 c 运行环境，初始化 zephyr 的 rw/zero
  - 准备好多任务运行环境
  - 依次调用 keil 工程中定义的 device/subsystem 的 Init 函数
  - 切换到 main 线程的入口函数 app_main
- app_main: app_main 是 zephyr main 线程的入口函数。此时 zephyr 运行环境已经初始化完成。接下来就可以调用 zephyr 的接口开发应用了。

6.2 驱动/内核模块/subsys 的初始化

在本 SDK 中，zephyr 主要模块（驱动/内核模块/subsys）都已固化在 ROM 中，应用根据需求有选择性的使用。如下：



图中对 driver/kernel/susbsys 都有例子说明。需要说明的是 zephyr 的 driver 支持多个设备实例，例如图中的一个 uart 驱动支持 uart0/uart2 两个设备。

6.3 功耗管理

休眠唤醒流程如下：

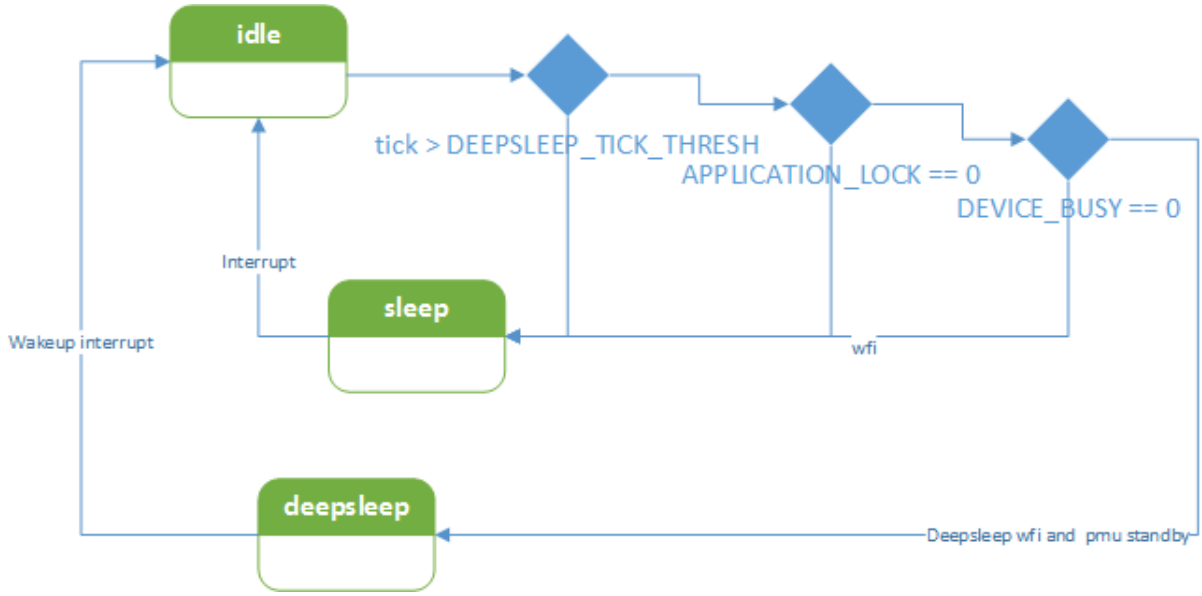


图 6.2: 休眠唤醒流程

deepsleep 模式是功耗最小模式，此时 cpu 进入 deepsleep 模式，pmu 进入 standby 模式。

6.3.1 tickless_idle 机制

tickless idle：如果超过 CONFIG_DEEPSLEEP_TICK_THRESH 个 tick 时间内都没有 expired 的 timer，则关闭 tick timer 以节省功耗。

```
#define CONFIG_DEEPSLEEP_TICK_THRESH 10
```

6.3.2 device_busy 机制

device_busy：指某些驱动通过异步方式（dma 中断、ir 中断、llcc 中断等）实现，在操作完成前系统不能进入 deepsleep。此类驱动在操作开始前 set_device_busy，操作完成后再 clear_device_busy。

```
int irc_acts_output_key(struct device *dev, struct␣
↪input_value *val)
{
        device_busy_set(dev);
}

void irc_acts_isr(struct device *dev)
{
        device_busy_clear(dev);
}
```

6.3.3 application_wake_lock 机制

application_wake_lock 机制：在某些应用场景下，为了性能考虑，禁止进入 deepsleep 模式。此时通过 app_get_wake_lock/app_release_wake_lock 接口在应用层禁止或允许进入 deepsleep。

```
void start_audio_capture(void)
{
        app_get_wake_lock();
        audioin_start_record(audioin);
}

void stop_audio_capture(void)
{
        audioin_stop_record(audioin);
        app_release_wake_lock();
}
```

List of Figures

List of Tables