软件开发说明

HL9-SDK 模块

（V1.4）

南京芮捷电子科技有限公司

地址：南京市浦口高新区星火路 20 号

电话：156 5102 8736

邮箱：sales@rejeee.com

网址：www.rejeee.com

 2019-07-25

修订历史

日期	版本	描述	作者/修改者	审核
2019-02-20	V1.0	文档创建，初稿	Felix	
2019-07-25	V1.1	完善说明，更新版本	Felix	
	V1.2	增加烧录仿真说明	Felix	
2019-08-06	V1.3	增加代码解析	Felix	
2019-10-18	V1.4	增加 RTC 和 I2C 操作说明	Felix	

目　　录

导　言

1.1. 概述

HL9-SDK 开发包是在 HL9 软硬件基础上，进行优化完善，集成了 HL9 模组的所有功能，并对硬件进行了扩展，提供了更多的外部接口，满足不同的用户使用和开发需求。对成本控制严苛的用户而言，更便于其简单、快速的进行 LoRa 通信技术开发与评估。

1.2. 读者对象

本文档适用于：

▲ 研发工程师

▲ 技术支持工程师

▲ 客户

如果您是第一次本产品，建议您从第一章开始，阅读本文档全部内容，以便更好的了解产品功能，熟悉使用方式，防止造成操作不当等人为原因带来的不必要损失。

2. 硬件介绍

参考对应 M-HL9 相关数据手册。

3. 系统说明

3.1. 关于 HL9 版本

HL9 标准版本由于不用考虑二次开发需求，因此将剩余的空间做了一个缓冲区，用于接收大数据量情况，内部分配了 5 个包，每包 228 个字节缓冲池。串口按照字节流间隔延时或长度进行分隔。一次性突发大数据会自动分成多个大数据包发送出去。

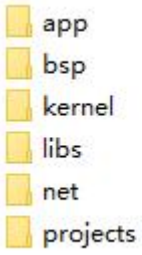
如果串口推送数据大于无线发送能力，则会导致串口数据无法及时取出而与后续数据组合在一起发送。

HL9-SDK 版本去除了缓冲池功能，空出的 RAM 和 Flash 资源用于开发者构建自己的业务逻辑。

HL9-SDK-Lite 版本基于更高开发资源需求，只保留了主任务操作和收发示例操作。

4. 源码说明

4.1. 目录结构



app: 公用程序文件

bsp：主板驱动支持文件

kernel：内核相关文件

libs：通用库文件

net：网络 MAC 协议相关

projects：工程相关文件

4.2. 开发环境

最新版本 SDK 支持 IAR 和 Keil 两种编译环境。使用之前请阅读 MCU 对应的开发软件包，主要先为 IDE 配置 MCU 支持软件包。

IAR 的 IDE 支持包目录结构如下图所示，使用时将其对应目录下的 HDSC 文件夹下的文件一一对应拷贝到 IAR 的安装路径（例如： ~\IAR Systems\Embedded Workbench7.5\arm\config）下，即可使用。

Keil 对应软件包为： HDSC.HC32L13X.1.0.0.pack ， 也可以通过 http://www.keil.com/dd2/pack/看是否有相关 MCU 的最新版本支持包。

SDK 的 IAR 的开发环境用 7.7 构建，更高版本请自行移植。相关软件请自行在官网：https://www.iar.com/iar-embedded-workbench 下载和安装。

Keil 采用 5.25 构建，软件请自行在官网：http://www.keil.com/下载和安装。

仿真器可采用 J-Link 仿真或 Keil，IAR 支持的相关仿真器，采用 SWD 接口。

4.3. 调式说明

本评估板使用 MCU 仿真的话，需要注意两点，相关说明参考《HC32L13 / HC32F03 系列的 MCU 开发工具用户手册》，如下所示截图。

1. MCU 深度休眠时无法使用 SWD 调式，需要复位芯片以恢复 SWD 调式口功能进行程序仿真。
2. MCU 启动支持 BOOT 选择开关，对应的端口是 PD03，PD03 低电平则为运行模式（可仿真调试），高电平为 ISP 烧录模式。

8.1 SWD 端口作为 GPIO 功能程序调试

在应用程序中如果需要将 SWD 端口配置为 IO 使用，程序将无法进行调试。

如果程序中需要使用该功能，建议在调试开发阶段，在程序一开始添加几秒钟的延时程序，或者添加外部 IO 控制程序等方法来决定是否执行该段程序，以便在二次调试开发时 SWD 功能能够正常使用。

8.2 低功耗模式程序调试

在应用程序中，如果使用的芯片具备低功耗模式并需要进入低功耗模式，此时因为 SWD 功能关闭，程序将无法使用调试功能。

如果程序中需要使用该功能，建议在调试开发阶段，在程序一开始添加几秒钟的延时程序，或者添加外部 IO 控制程序等方法来决定是否执行该段程序，或者增加外部唤醒机制，以便在二次调试开发时 SWD 功能能够正常使用。

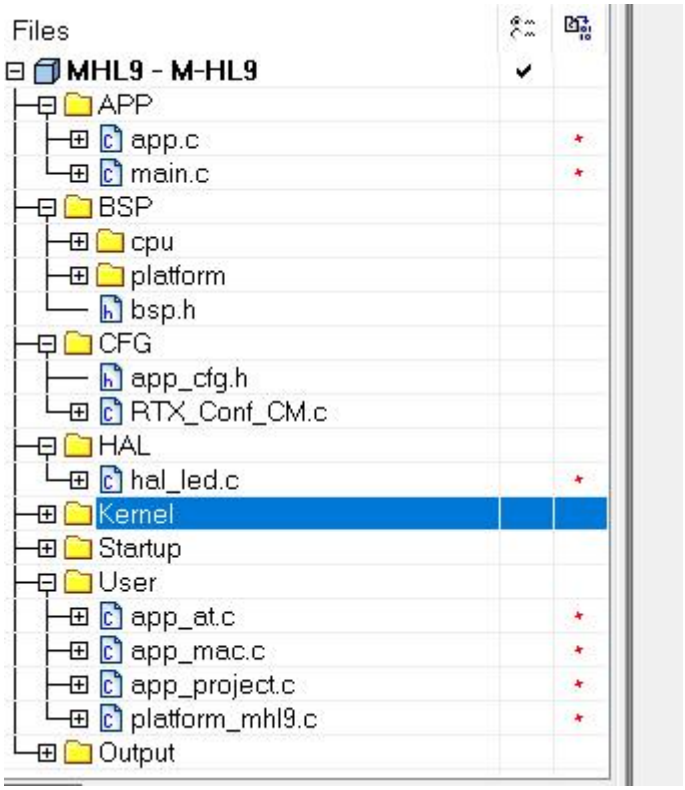
4.4. 工程说明

例程经过很好的代码封装，模块化耦合度低，main 文件为主程序入口。

采用 RTX 系统多任务处理，除主任务外，分别 AT Task（app_at.c）和 Mac Task(app_mac.c)分别处理 AT 指令和无线收发。

User 为 HL9 用户二次开发和可修改代码目录。

上述代码结构目录如下所示。



4.5. 代码解读

4.5.1. 代码架构

基于不同 MCU 和不同项目定制化需求，整合多种 RTOS 和不同硬件设计方案演化出来的目录结构，尽量将公用的接口函数和头文件整合合并，使其在代码复用性和移植性方面更加便捷快速。

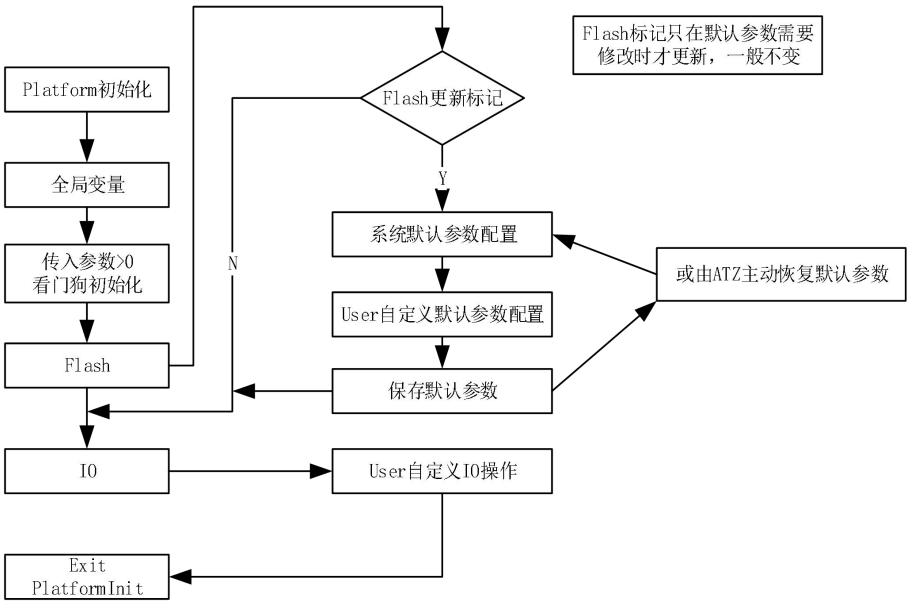
bsp 为芯片驱动代码，platform 为基于 MCU 衍生出来的不同主板定义，projects 为实际工程文件，代表具体的硬件设计方案，其中 HAL 为定制化的硬件操作接口。

4.5.2. Platform 初始化

目前 SDK 最新代码输入参数为看门狗超时时间，单位秒。如果输入为 0，则不初始化看门狗，注意 bsp.h 的喂狗宏定义也需禁用（以防止不必要的调用）。

```
#if DISABLE_WDOG
#define APP_FeedDog()
#else
#define APP_FeedDog()                BSP_WatchdogFeed()
#endif
```

在 PlatformInit 初始化过程中，会回调用户配置和外部 IO 配置，分别对应 DevCfg_UserDefault 和 UserExternalGPIO 两个函数。以便用户可以自定义默认参数、外围 AT 和 WKUP 的 IO 使用，即用户可以配置其它 IO 作为 AT 和 WKUP，也可以去除 AT 和 WKUP 功能而改做他用，相应的需要修改对应的 WKUP 和 AT 判断操作。



4.5.3. LPTIMER 使用注意

BSP_LPowerInit 为初始化 LPTIMER 配置。默认定时低功耗采用 LPTIMER，如果需要 LPTIMER 用作他用，可以不初始化或重新初始化成其它功能，相应的 PlatformSleep 和 PlatformSleepMs 即不要使用，因为这两个休眠操作是通过 LPTIMER 来计时的。

4.5.4. 串口使用

默认 UserDebugInit 为串口初始化操作，其中的 IO 都是可以由用户自定义，但必须使用 UART 标准功能口，即对应的串口编号要与芯片串口定义相符，如下注释对应芯片编号，枚举为重命名定义。默认使用 LPUART0（PB00 和 PB11）。

```
typedef enum {
    BSP_LPUART0 = 0,     /**<    LPUART0          */
    BSP_LPUART1,         /**<    LPUART1          */
#ifndef USE_NO_UART
    BSP_UART0,           /**<    UART0            */
    BSP_UART1,           /**<    UART1            */
#endif
    BSP_UART_INVALID     /**<    Invalid UART     */
} BSP_UART_NUM;
```

如使用 PD00 和 PD01，对应则为 UART1，即初始化时需要使用 BSP_UART1 作为编号。

5	5	4	4	PD00	I2C0_SDA UART1_TXD
6	6	5	5	PD01	I2C0_SCL TIM4_CHB UART1_RXD

SDK 已经封装好了串口接收缓存 FIFO 操作，可以实时接收输入存入 FIFO 中，在初始化之前，需要先声明 FIFO 的缓存空间和信号控制变量，如下所示。用户可以根据实际业务数据量自定义需要的串口缓存区大小，以便合理使用空间。

sDebugBuffer 为 FIFO 循环缓存区，timeout 为字节分包超时时间，单位毫秒，用户可以自定义超时时间以满足自身业务需求。

```
static uint8_t sDebugBuffer[DBG_UART_SIZE + 1] = {0};
static Ringfifo sDebugFIFO = {0};
struct sp_uart_t gDebugUart = {
    .rx_sem = {0},
    .rx_fifo = &sDebugFIFO,
    .timeout = DBG_UART_TIMEOUT,
    .idx = DBG_UART_IDX
};
```

```
/* Note: you can set timeout you need */
gDebugUart.timeout = BSP_UartSplitTime(baudrateType);
gDebugUart.idx = uart.idx;

if(false == reinit){
    /* first init */
    success = BSP_OS_SemCreate(&gDebugUart.rx_sem, 0, DEBUG_SEM_NAME);
    if (false == success) {
        /* NOTE: Must make device reset */
        return false;
    }
    success = DevUART_Init(&uart, &gDebugUart, sDebugBuffer, sizeof(sDebugBuffer));
} else {
    success = DevUART_ReInit(&uart);
}
```

另外，串口结构体支持自定义上下拉操作，以便用户根据实际外接 MCU 和模组需求来配置。

如串口无输入需求，串口 FIFO 和 Callback 都可以不用，可以参考 SDK-Lite 版，直接用 BSP 中 API 初始化串口，节约内存。

```
BSP_UART_TypeDef uart = {
    .cb = DebugCallback,
    .gpio = DBG_GPIO,
    .tx_pin = DBG_TX_PIN,
    .rx_pin = DBG_RX_PIN,
    .af = DBG_AF,
    .pd = GpioPu,
    .idx = DBG_UART_IDX,
    .bdtype = baudrateType,
    .pri = pariType
};
```

串口读取操作参考 AT 任务，DevUART_Read，默认应该使用阻塞式操作，以便正确完整读取一次串口输入。

4.5.5. 任务启动

默认启动 MAC、AT 和主任务。AT 启动后等待输入，MAC 启动后根据全局变量标记判断是否开启接收。Radio 在执行 RX 或 TX 的 Process 时，根据全局参数 gDevFlash.config 执行 Radio 收发操作。因此，在收发之前，如果有需要改变 Radio 参数，请在执行收发动作前修改 gDevFlash.config 相关变量即可。

如果业务代码无需如 AT 或 MAC 任务，收发操作时序是已知，完全可以参考 SDK-Lite 进行处理，Lite 版本只保留了主任务，空出的 RAM 可以用来实现更多的业务功能。

只要数准备好，调用 MacRadio_TxProcess 即为无线发送，根据返回值判断是否成功。在需要接收时调用 MacRadio_RxProcess 即为阻塞时接收，根据返回值判断数据是否有效，如果需要终止接收，可以自定义 Timer 或超时机制来中断接收，中断接收的函数为 MacRadio_AbortRx。

4.6. 二次开发参考

SDK 包中集成了 AT 指令集，AT 模式软硬件切换方式、LoRa 无线自动收发操作，休眠唤醒、低功耗串口等，提供 RTOS 系统接口方便进行任务管理，用户可以根据需要增删功能。下面简要介绍一下相关简单操作。

4.6.1. 工程文件入口

app_project.c 文件为工程文件入口，相关函数：AppTaskCreate 用于初始化硬件平台和任务构建。为了满足不同用户需求，将 LPTimer 使用开放出来，默认

LPTimer 用于低功耗休眠。

低功耗休眠的初始化操作为：BSP_LPowerInit，配合 PlatformSleep 和 PlatformSleepMs，分别对应按秒计数休眠和毫秒级别休眠。如果不用 SDK 的休眠动作，可以不用 BSP_LPowerInit 初始化操作，可以将 LPTimer 用于自己业务需求。

4.6.2. 创建任务

为了节约资源，SDK 任务定义中只定义了 SDK 使用到的任务数量，如果需要自行增加任务，需要在 RTX 的主配置文件 RTX_Conf_CM.C 中修改任务数量限制。如下所示，OS_TASKCNT 最大可运行任务数，OS_STKSIZE 任务堆栈大小，OS_MAINSTKSIZE 主任务堆栈大小。

```
//
// <h>Thread Configuration
// ======================
//
//   <o>Number of concurrent running threads <0-250>
//   <i> Defines max. number of threads that will run at the same time.
//   <i> Default: 6
#ifndef OS_TASKCNT
 #define OS_TASKCNT     3
#endif

//   <o>Default Thread stack size [bytes] <64-4096:8><#/4>
//   <i> Defines default stack size for threads with osThreadDef stacksz = 0
//   <i> Default: 200
#ifndef OS_STKSIZE
 #define OS_STKSIZE     200
#endif

//   <o>Main Thread stack size [bytes] <64-4096:8><#/4>
//   <i> Defines stack size for main thread.
//   <i> Default: 200
#ifndef OS_MAINSTKSIZE
 #define OS_MAINSTKSIZE 100
#endif
```

参考 app_mac.c 文件，创建任务主要有三段：

声明任务处理函数

static void MacTaskHandler(void const *p_arg);

定义任务结构体

osThreadDef(MacTaskHandler, osPriorityNormal, 1, 0);

#define APP_MAC_NAME osThread(MacTaskHandler)

创建任务

BSP_OS_TaskCreate(&gParam.macid, APP_MAC_NAME, NULL);

此处使用 SDK 封装好的函数执行，也可以自己按照 RTX 标准方式创建。

4.6.3. 串口 FIFO 实现

SDK 中主要关键外设代码在文件 platform_mhl9.c 中。主要提供给二次开发者进行外设自定义操作，默认其中主要实现串口、中断、AT 或 WakeUp 相关 IO 的操作，开发者可以根据自己业务需要修改或删除相关外设功能。

这个文件主要实现了串口收发的初始化和回调，实现了串口 FIFO 功能，配合 AT 任务进行串口数据读取。

UART 串口相关代码：

UserDebugInit 串口初始化动作，最新 SDK 为满足客户需要，开放串口分包延时和串口上下拉配置，默认分包延时为 5ms。

FIFO 定义

```
static uint8_t sDebugBuffer[DBG_UART_SIZE] = {0};
static Ringfifo sDebugFIFO = {0};
struct sp_uart_t gDebugUart = {
    .rx_sem = {0},
    .rx_fifo = &sDebugFIFO,
    .timeout = DBG_UART_TIMEOUT,
    .idx= DBG_UART_IDX
};
```

UART GPIO 和中断定义

```
BSP_UART_TypeDef uart = {
    .cb = DebugCallback,
    .gpio = DBG_GPIO,
    .tx_pin = DBG_TX_PIN,
    .rx_pin = DBG_RX_PIN,
    .af = DBG_AF,
    .pd = GpioPu,
    .idx = DBG_UART_IDX,
    .bdtype = baudrateType,
    .pri = pariType
};
```

UART 分包超时时间 timeout 设置，单位 ms

```
/* Note: you can set timeout you need */
gDebugUart.timeout = BSP_UartSplitTime(baudrateType);
gDebugUart.num = uart.num;
```

注意：

PA 端口用于 SPI 操作 Radio 相关动作，SDK 内部自动使能 PA 和中断功能，

不能在外部被禁用或禁止 PA 中断功能，如果禁用需及时恢复，否则射频收发无法正常工作。

4.6.4. 无线收发操作

通过 Mac_Init 初始化射频参数和无线状态控制。只要 PlatformInit 和 Mac_Init 初始化成功后，就可以选择如下函数实现无线收发功能，更简单示例可参考 SDK-Lite 版本代码。

MAC 收发数据，为了节省空间，共用了一个结构体变量 MacParam 进行中转，也可以定义两个对象分别负载发送和接收数据。具体根据需要设计。正常因为模组收发不是同时（半双工）操作的，所以共用对象只要不复写对应内容没有什么问题（即发送过程中不要修改该对象内容，完成后才能操作）。相关数据输出可参考示例函数 RadioPrintRecv。

主要函数如下

MacRadio_RxProcess 为持续接收函数

MacRadio_CadProcess 为侦听接收函数

MacRadio_TxProcess 为发送函数

4.6.5. ADC 功能

最新版本 SDK 开放 ADC 参考源选择，用户使能 ADC 前先配置参考源，如下所示采集 MCU 内部 VCC 电压或示例 PA02。

BSP_ADC_Enable 开始按照参数使能 ADC，中间可采集多次或多个 ADC 口，BSP_ADC_Disable 失能 ADC，关闭 ADC 功能。

```
void Dev_GetVol(void)
{
    BSP_ADC_TypeDef adcCfg;
    uint32_t adc = 0;

    /* Internal 2.5V voltage reference */
    adcCfg.ref = RefVolSelInBgr2p5;

    BSP_ADC_Enable(&adcCfg);
    adc = BSP_ADC_Sample(0, AdcAVccDiV3Input);
    gParam.dev.vol = (adc*2500*3)/4096;

    /** example */
    /*
    adc = BSP_ADC_Sample(0, AdcExInputCH2);
    adc = (adc*2500)/4096;
    */
    BSP_ADC_Disable(NULL);
}
```

4.6.6. RTC 功能

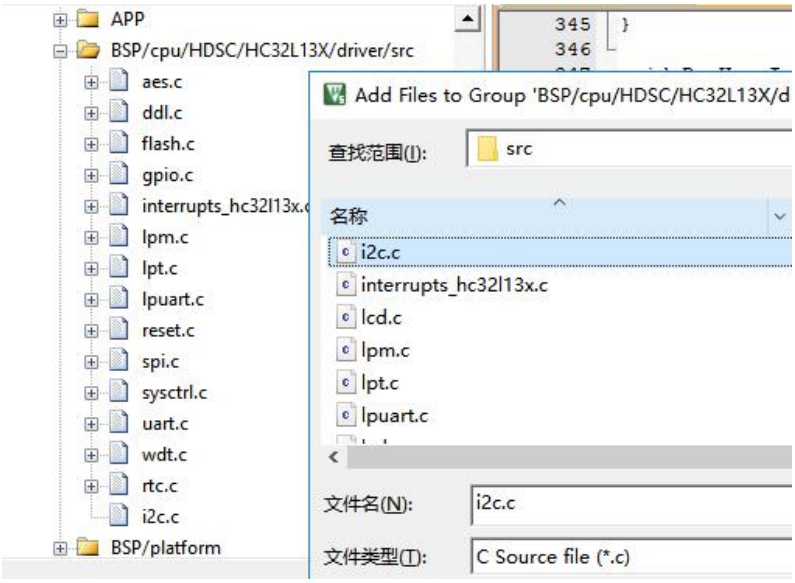
最新版本增加 RTC 驱动，可自由设置 RTC 时钟获取运行时间、日历等。相关示例代码已在 platform_mhl9.c 文件中，可自行决定是否使用或注释。

```
void DevUserInit(void)
{
    BSP_RTC_TypeDef rtcCfg;
    stc_rtc_time_t stcTime;

    rtcCfg.callback = NULL;
    rtcCfg.dateTime = NULL;
    rtcCfg.extl = gParam.dev.extl;
    if(gParam.rst.bits.u8Porl_5V || gParam.rst.bits.u8Por5V){
        DDL_ZERO_STRUCT(stcTime);
        /* Init RTC timer */
        stcTime.u8Year = 0x19;
        stcTime.u8Month = 0x01;
        stcTime.u8Day = 0x01;
        stcTime.u8Hour = 0x00;
        stcTime.u8Minute = 0x00;
        stcTime.u8Second = 0x01;
        stcTime.u8DayOfWeek = Rtc_CalWeek(&stcTime.u8Day);
        rtcCfg.dateTime = &stcTime;
    }
    BSP_RTC_Init(&rtcCfg);
```

4.6.7. I2C 功能

最新版本增加 I2C 驱动，由于 HL9 模组和评估板未板载 I2C 设备，因此默认 I2C 源文件并未引用。如果需要，请自行添加 i2c.c 源文件即可。



配置和开启 I2C 功能非常简单，只要 I2C 相关 BSP 结构体赋值后调用初始化函数即可。示意代码如下所示。

```
/* I2C0 Port Settings */
#define I2C0_SCL_PORT           GpioPortD
#define I2C0_SCL_PIN            GpioPin1
#define I2C0_SDA_PORT           GpioPortD
#define I2C0_SDA_PIN            GpioPin0
#define I2C0_ALTERF             GpioAf1
```

```
BSP_I2C_TypeDef i2c;
bool success = true;

i2c.idx      = 0;
i2c.baudmode     = I2C_BAUD_NORMAL;
i2c.scl_port     = I2C0_SCL_PORT;
i2c.scl_pin      = I2C0_SCL_PIN;
i2c.sda_port     = I2C0_SDA_PORT;
i2c.sda_pin      = I2C0_SDA_PIN;
i2c.af           = I2C0_ALTERF;
BSP_I2C_Init(&i2c);
```

I2C 地址示意，如下所示，芯片 7bit 地址为 0x30，上述 addr 放入参数则为 0x60，BSP 驱动内部自动按读写操作切换 R/W 位。

```
/**
 *   I2C address: [7 bits addrss | 1 bit R/W
 *   this addrss should be left shift 1 bit and append the last bit for R/W
 *   0x60 = (0x30 << 1)
 */
```

I2C 读操作示意

uint8_t buf[len] ={0}; /* 待接收的数据缓存 */
chip_regs 为读操作访问的寄存器

```
/**
* 如大部分传感器都是从某个寄存器读取数据内容。
* 一般 chip_regs 为 1 个字节少数 2 个字节还有部分为直接读则寄存器为 0 字节
*  结构体定义的寄存器值宽度最多不超过 4 字节
*/
```

```
BSP_Slave_Param slave;
slave.addr = addr;     /* 从机地址，8bit 位，包含读写位整个字节 */
slave.idx = 0 或 1（分别代表 I2C0 和 I2C1）;
slave.len = 2; /* 读数据寄存器长度通常 1 或 0 芯片居多，有的地址长的为 2 */
slave.regs[0] = chip_regs >> 8;
slave.regs[1] = chip_regs;
```

该函数满足：直接读（slave.len=0），从指定寄存器读数据（slave.len 和 slave.regs 根据需要填充）

```
return BSP_I2C_Read(&slave, buf, len);
```

I2C 写操作示意

```
uint8_t buf[len] ={0};/* 待写的数据缓存 */
。。。。
/* 数据准备，size 为实际要写的字节数 */
。。。
```

size = 1; /* 一般都是写寄存器，多数为写 1 个字节。部分为写多个字节，具体根据芯片进行赋值 */

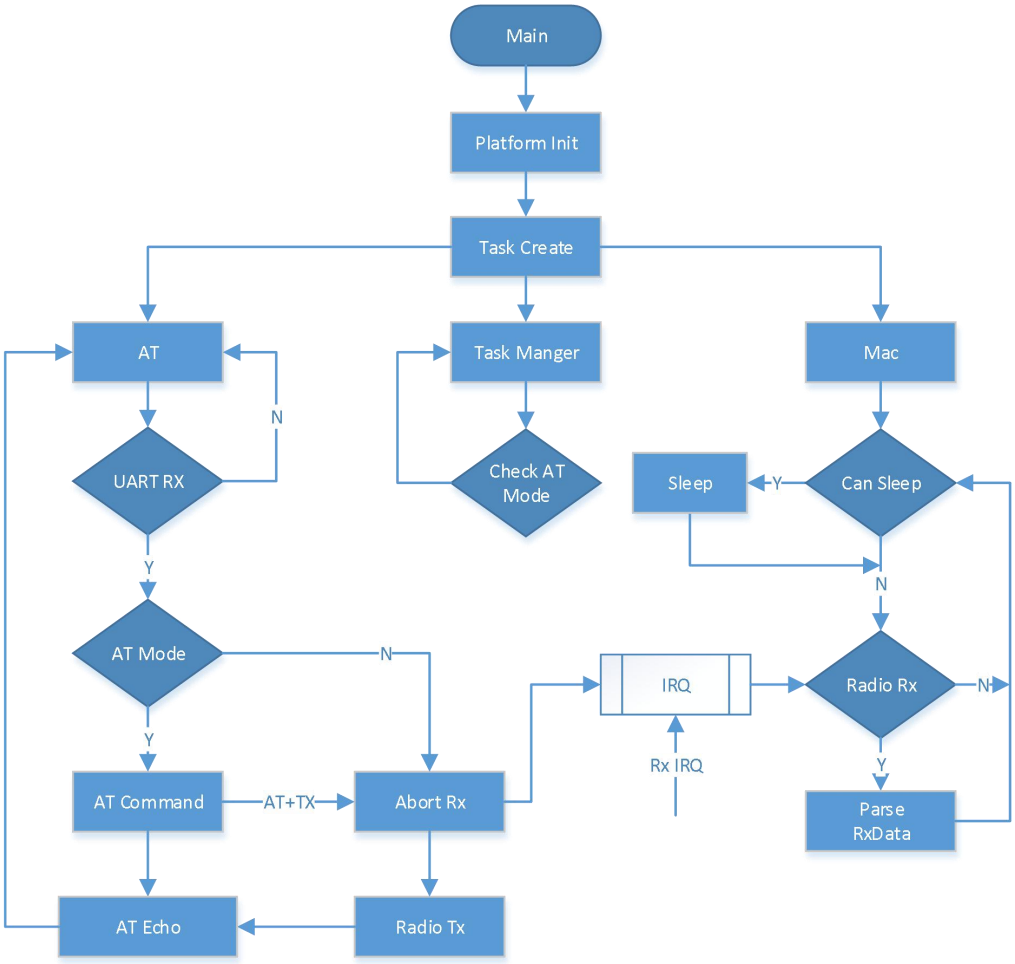
```
BSP_Slave_Param slave;
slave.addr = addr;  /* 从机地址，8bit 位，包含读写位整个字节 */
slave.idx = 0 或 1（分别代表 I2C0 和 I2C1）;
slave.len =  1;
slave.regs[0]  = chip_regs;
return BSP_I2C_Write(&slave, &buf, size);
```

4.6.8. 更多参考

具体函数调用参考：HL9-API.chm

具体 AT 操作，请参考 Rejeee AT 指令手册。

SDK 工程代码参考流程如下所示。

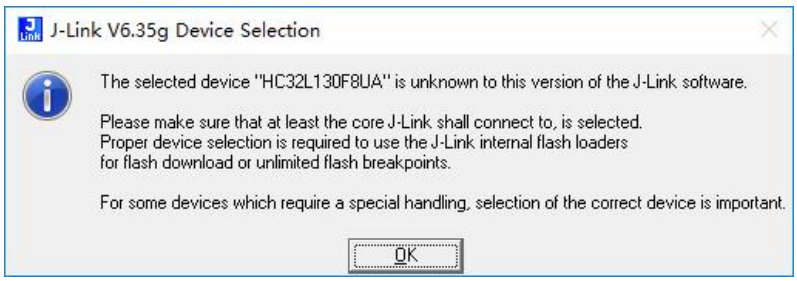


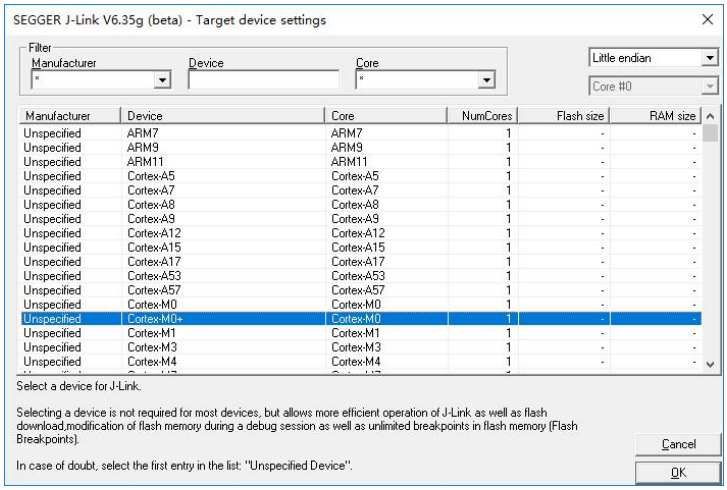
HL9 整体流程框图

5. 烧录仿真说明

5.1. 设备选型

HL9-SDK 源码下载后，工程直接可以编译烧录或仿真。第一次烧录或仿真时，由于本地无相关设备信息，需要手动选择设备类型，如下所示，打开 J-Link 的对话框选择 CortexM0+





5.2. 编译 ROM 注意

SDK 采用固定 Flash 区域存储设备信息和配置信息，HL9 选择最后 1 个扇区作为固定信息存储位置，请不要擦除芯片或复写该扇区内容。倒数第 2 个扇区为配置信息存储位置，AT 命令参数存储在该扇区中。

如删除了相关扇区内容，请联系技术支持或返厂烧录。