

---

## N32WB452系列蓝牙组件参考指南

---

### 简介

此文档的目的在于让使用者能够快速熟悉 N32WB452 系列蓝牙组件的使用，以减少开发前期的准备时间，降低开发难度。

# 目录

<b>1. 简介</b> .....	<b>1</b>
1.1. 概述 .....	1
1.2. 适用性 .....	1
<b>2. SDK 应用架构</b> .....	<b>2</b>
<b>3. 蓝牙组件资源需求</b> .....	<b>3</b>
3.1. 蓝牙组件占用资源 .....	3
3.2. 工程架构 .....	3
<b>4. 蓝牙组件参数及接口说明</b> .....	<b>4</b>
4.1. 蓝牙初始化函数 .....	4
4.2. 回调函数 .....	4
4.3. 初始化 BT_WARE_INIT() .....	4
4.4. 中断处理 BT_HANDLER() .....	5
4.5. 主线程 BT_RUN_THREAD() .....	5
4.6. 接收数据 BT_RCV_DATA() .....	5
4.7. 发送数据 BT_SND_DATA() .....	6
4.8. 断开连接 BT_DISCONNECT() .....	6
4.9. 状态监控 BLE_STATUS_MONITOR () .....	6
4.10. 等待蓝牙状态反馈 BLE_MONITOR_WAIT () .....	6
4.11. 状态监控回调函数 BLE_MONITOR_CALLBACK () .....	7
4.12. 蓝牙协议栈初始化 BT_INIT() .....	7
4.13. 蓝牙内核繁忙状态检测 IS_BT_BUSY() .....	7
4.14. 获取当前蓝牙 API 版本 BT_GET_VERSION ( ) .....	7
<b>5. 使用指南</b> .....	<b>9</b>
5.1. 操作流程 .....	9
5.2. 应用范例 .....	10
5.3. BLE 低功耗配置 .....	12
5.3.1. 蓝牙广播参数设置 .....	12
5.3.2. 蓝牙低功耗配置 .....	13
5.3.3. 低功耗唤醒 .....	14
<b>6. 版本历史</b> .....	<b>16</b>
<b>7. 声明</b> .....	<b>17</b>

## 1. 简介

### 1.1. 概述

欢迎使用国民技术 N32WB452 系列蓝牙组件参考手册，文档主要介绍 N32WB452 系列 MCU 蓝牙通讯的接口说明及使用指南。

N32WB452 系列芯片支持蓝牙 BLE5.0 规范，支持 Profile 配置，客户可以根据自己的需求通过接口参数修改 Profile 项。

开发者可配合片上相关资源及 SDK 套件开发出自己的蓝牙产品。

本文档将详细描述蓝牙通讯 API 的资源需求、各 API 函数功能说明及参考范例。

### 1.2. 适用性

- N32WB452 系列中蓝牙 BLE5.0 运行于片内独立的高性能 32 位处理器上；片内应用 ARM Cortex-M4 内核，通过提供的 API 接口对蓝牙 BLE5.0 内核进行控制、及数据收发；因此本文档中的蓝牙组件仅适用于 N32WB452 系列芯片。
- 本文档提供的蓝牙组件屏蔽底层硬件以及复杂协议栈等处理，抽象出简单、易用的用户层接口。为 MCU 平台的开发者提供高集成、方便、蓝牙通讯功能需求。
- SDK 暂只支持 KEIL5 平台，其他编译平台正在完善中...

## 2. SDK 应用架构

API 支持系统或无系统的蓝牙产品。

当有蓝牙相关事件发生时，通过用户层回调函数通知应用层。

图 1 蓝牙组件的应用分层

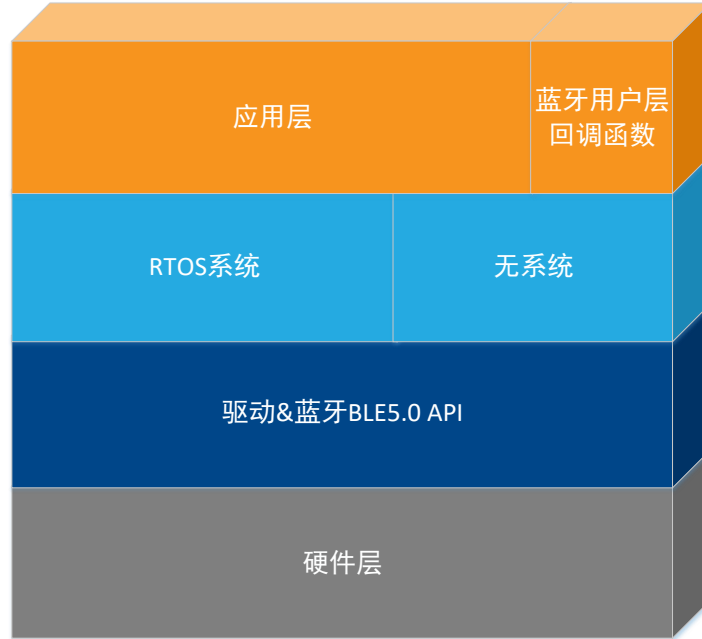
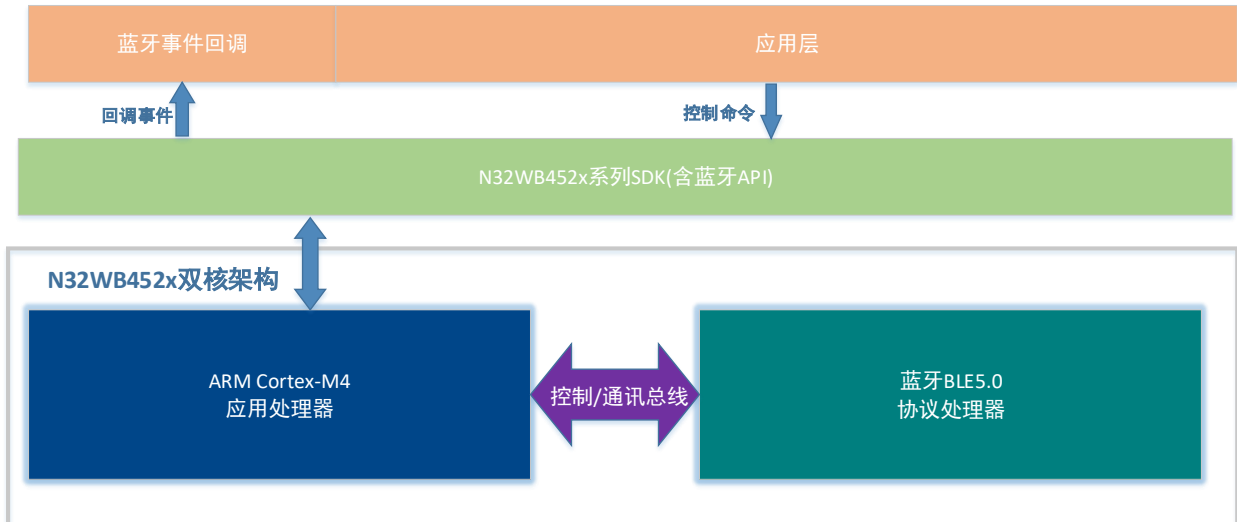


图 2 蓝牙控制及回调事件框



### 3. 蓝牙组件资源需求

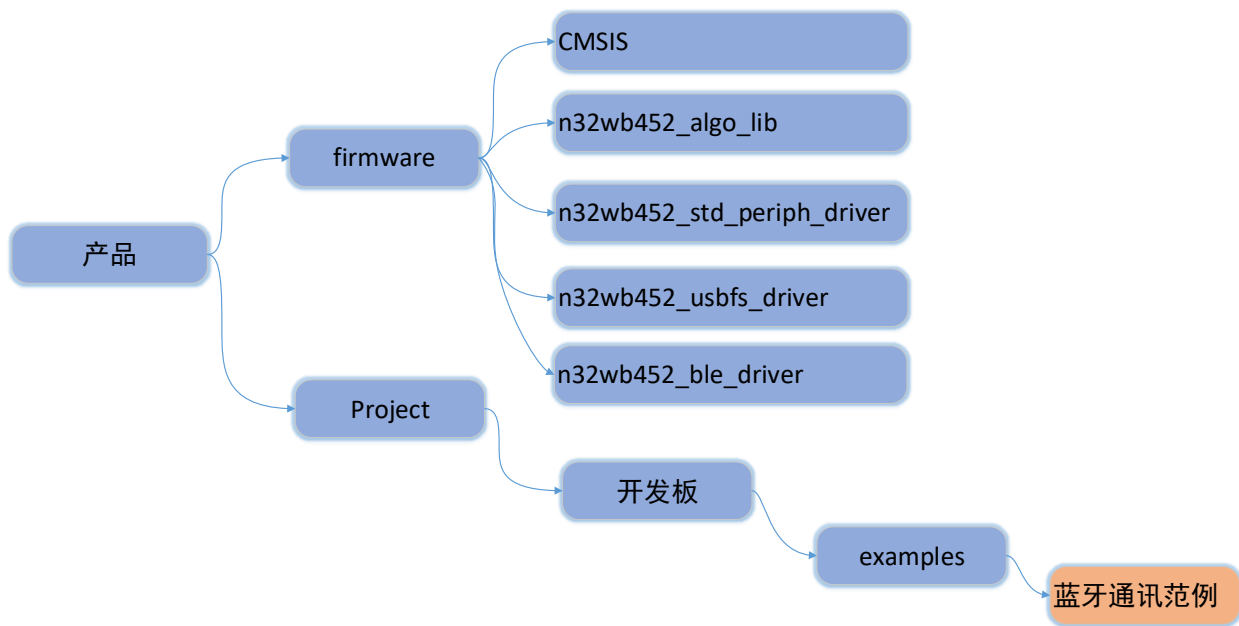
SDK 的使用步骤大致分为：初始化、运行蓝牙主线程、回调(蓝牙事件触发时)。

#### 3.1. 蓝牙组件占用资源

No.	资源	说明	备注
1	FLASH 70KB	约 70KBytes FLASH 空间。	-
2	SRAM	约 6KB	-
3	Stack	约 5KB	-
4	Heap	约 2KB	-
5	外部中断 EXTI6	BLE 状态监控中断	
6	外部中断 EXTI14	BLE 通用中断	
7	定时器 TIM3	BLE 内核超时事件	

#### 3.2. 工程架构

图 3 代码工程目录结构



## 4. 蓝牙组件参数及接口说明

### 4.1. 蓝牙初始化函数

初始化函数 `bt_attr_parambt_init;` //用于定义蓝牙配置信息，如广播名称、地址、UUID、特征子等。详细说明如下：

变量	说明
<code>ctrl_param</code>	控制标志。
<code>device_name[32]</code>	蓝牙广播的名称。 *如为 0 则使用默认蓝牙名称
<code>scan_rsp_data[31]</code>	master 扫描时 slave 的应答数据。 *有效长度不超过 30 字符。
<code>scan_rsp_data_len</code>	应答数据长度
<code>device_addr[20]</code>	蓝牙设备地址 *格式如：“FE:11:22:FA:33:44”。如为 0 则自动配为默认地址
<code>service[1]</code>	蓝牙的服务定义，包括特征子，权限等。

### 4.2. 回调函数

回调函数由用户层实现，主要用于蓝牙事件时，通知上层进行处理。如连接、断开、数据接收等。

**注意：在回调函数执行时间应尽可能短，如果时间过长可能会影响收发效率！**

回调函数类型定义：

```
typedef void (*bt_event_callback_handler_t)( bt_event_enum event,
                                             const uint8_t * data,
                                             uint32_t size,
                                             uint32_t character_uuid);
```

参数说明：

[IN] <code>bt_event_enum</code>	<code>event</code>	蓝牙组件通知事件。
[IN] <code>const uint8_t</code>	<code>*data</code>	数据地址(有数据事件通知时)。
[IN] <code>uint32_t</code>	<code>size</code>	数据大小(单位字节数)
[IN] <code>uint32_t</code>	<code>character_uuid</code>	特征子 id，不关注。

返回值：

无

### 4.3. 初始化 `bt_ware_init()`

定义：`int32_t bt_ware_init(bt_attr_param *pinit, bt_event_callback_handler_t callback);`

蓝牙组件初始化。

参数说明:

[IN] bt_attr_param *	pinit	蓝牙初始化参数
[IN] bt_event_callback_handler_tpcallback		用户回调接口。

返回值:

BT_RET_SUCCESS	初始化成功
其他	初始化失败

#### 4.4. 中断处理 bt\_handler()

定义: void bt\_handler(void)

蓝牙组件中断处理。

参数说明:

void:无。

返回值:

void:无。

#### 4.5. 主线程 bt\_run\_thread()

定义: void bt\_run\_thread(void);

蓝牙组件运行的主线程,用于蓝牙接收、发送、广播、通知等事件处理。建议主线程运行时间间隔应小于1ms,如果需要发送或接收大数据量时,可以集中时间调度。

参数说明:

void:无。

返回值:

void:无。

#### 4.6. 接收数据 bt\_rcv\_data()

定义: uint32\_t bt\_rcv\_data(uint8\_t \*data, uint32\_t size, uint32\_t character);

根据 BT\_EVENT\_RCV\_DATA 事件,读取相应服务特征字的数据。

参数说明:

[OUT] uint8_t	*data	数据地址
[IN] uint32_t	size	数据大小
[IN] uint32_t	character	发生数据的特征字,不关注

返回值:

返回实际读取大小(单位:字节数)

## 4.7. 发送数据 bt\_snd\_data()

uint32\_t bt\_snd\_data(const uint8\_t \*data, uint32\_t size, uint32\_t character);

发送数据

参数说明:

[IN] uint8_t	*data	发送的数据地址
[IN] uint32_t	size	发送数据大小
[IN] uint32_t	character	发送的特征字, 不关注

返回值:

返回值 0。

## 4.8. 断开连接 bt\_disconnect()

Void bt\_disconnect(void)

Slave 主动断开蓝牙连接

参数说明:

void:无。

返回值:

void 无。

## 4.9. 状态监控 ble\_status\_monitor ()

void ble\_status\_monitor (void);

蓝牙工作状态实时监控, 必要时自动对蓝牙进行复位操作。

参数说明:

void:无。

返回值:

void 无。

## 4.10. 等待蓝牙状态反馈 ble\_monitor\_wait ()

uint8\_t ble\_monitor\_wait (uint32\_t timeout);

调用状态监控函数 ble\_status\_monitor 后, 需要调用此函数等待蓝牙反馈实时状态。

参数说明:

timeout:等待超时时间(32bit 无符号整数), 通过 while 循环简单计时。

返回值(8bit 无符号整数):



0: 收到状态反馈，蓝牙工作正常。

1: 未收到状态反馈，等待超时。

#### 4.11. 状态监控回调函数 ble\_monitor\_callback ()

void ble\_monitor\_callback (void);

蓝牙状态监控回调函数

参数说明:

void:无。

返回值:

void 无。

#### 4.12. 蓝牙协议栈初始化 BT\_init()

void BT\_init(void)

初始化蓝牙协议栈。

参数说明:

void:无。

返回值:

void:无。

#### 4.13. 蓝牙内核繁忙状态检测 is\_bt\_busy()

bool is\_bt\_busy(void)

检测蓝牙内核繁忙状态。当内核繁忙时，不允许进入低功耗状态。

参数说明:

void:无。

返回值:

1: 繁忙

0: 空闲

#### 4.14. 获取当前蓝牙 API 版本 BT\_get\_version ()

void BT\_get\_version(uint8\_t \* version)

获取当前蓝牙 API 版本信息。

参数说明:

version: 版本信息字符串缓存，最大 10byte。

返回值:

void:无

## 5. 使用指南

### 5.1. 操作流程

#### ➤ 系统时钟：

请确保系统时钟 `SYCLK_FREQ` 为 8 的整数倍，建议取值为 144MHz、96MHz、48MHz。

#### ➤ 定义回调函数：

回调类型为：`bt_event_callback_handler_t`

#### ➤ 配置参数及初始化：

填充 `bt_ware_init` 参数内容，并调用初始化接口。

*注：判断接口返回值，如 `BT_RET_SUCCESS` 成功后继续，否则检查错误类型*

#### ➤ 定义相关中断：

用于蓝牙组件中断处理 `bt_handler()`。

#### ➤ 运行主线程：

运行 `bt_run_thread()` 主线程。

*注：根据用户的实际环境，如在不系统/系统环境下，确保蓝牙组件主线程的运行周期为 1ms。如其他线程调度关系或频繁擦写 `FLASH`，导致蓝牙主线程得不到执行权限，则会导致那一时刻的数据接收异常或出错。*

#### ➤ 状态监控：

蓝牙状态监控函数用于监控蓝牙控制器及内核状态，用于状态监控与控制、功耗自动管理，要求用户在程序中周期性调用，建议的调用周期为 1 秒钟。

*注意：一旦启用了状态监控，则必须在外部中断函数 `EXTI9_5_IRQHandler()` 中调用状态监控回调函数 `ble_monitor_callback()`。*

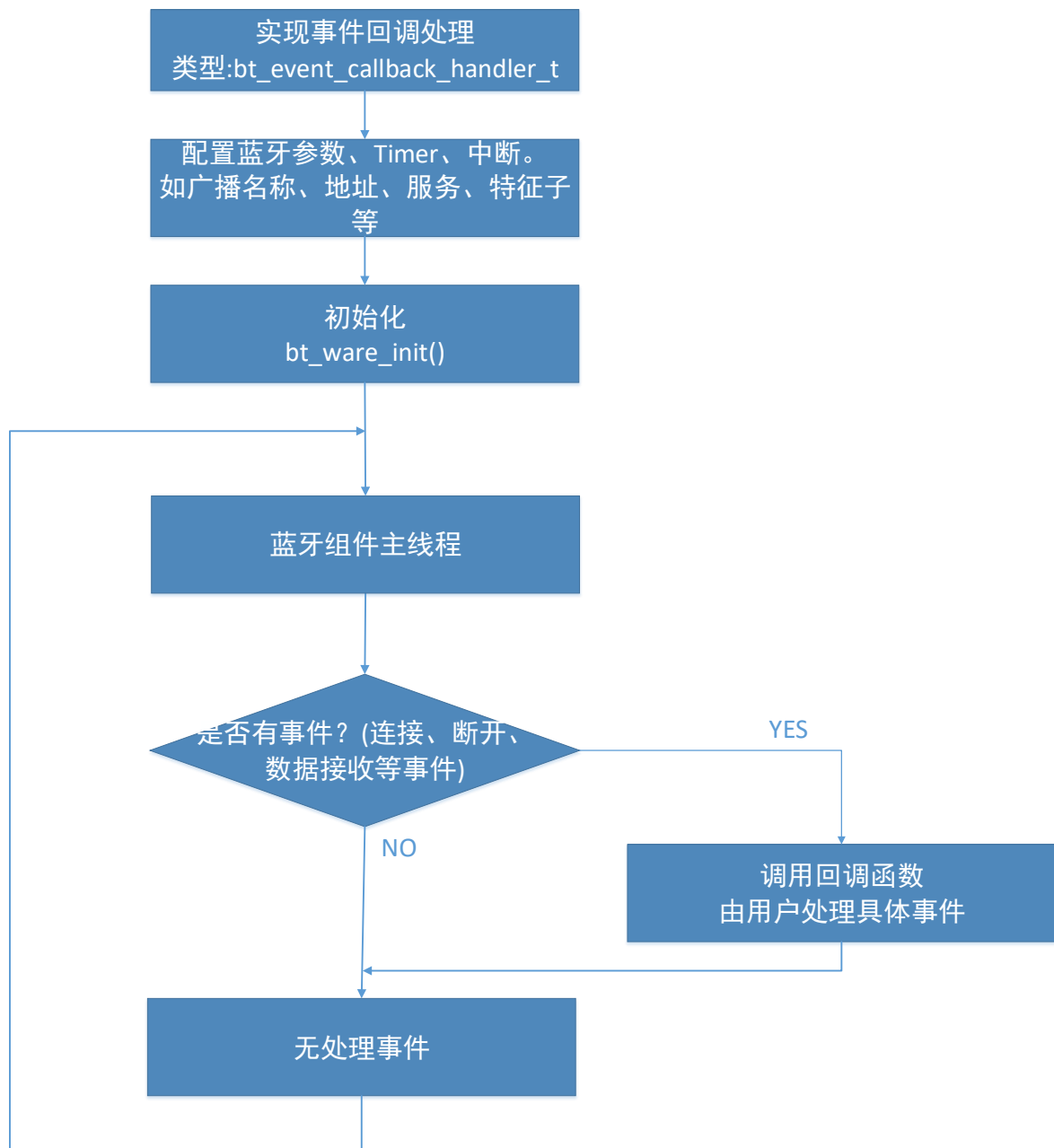
#### ➤ 低功耗机制：

MCU 上电启动 10 秒后会尝试进入 `STOP2` 模式，此时如果蓝牙在非连接状态，系统会进入 `STOP2` 模式，节省功耗。当有蓝牙事件或 MCU 其他外部中断时系统会被唤醒，唤醒之后要重新执行初始化流程。

在被唤醒 5 秒后，系统会再次尝试进入低功耗模式。

若在蓝牙非连接状态下，系统需要长时间处理其他应用任务不能进入低功耗模式时，则请执行完所需操作后再进入低功耗模式。

图 4 蓝牙主组件处理流程



## 5.2. 应用范例

```
uint8_t snd_buf[] = {"0-1234567890."};
bt_attr_parambt_init = {0};
```

```
/**
 * @brief 蓝牙中断处理.
 * @param count specifies the delay time length.
 */
void EXTI15_10_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_LINE14) != RESET)
    {
```

```
        bt_handler();
        EXTI_ClrITPendBit(EXTI_LINE14);
    }
}

Void bt_event_callback_func(bt_event_enum event, uint8_t * data, uint32_t size, uint32_t character_uuid)
{
    uint16_t length;

    switch (event)
    {
    case BT_EVENT_VERSION:
        printf("cb version:\r\n");
        break;

    case BT_EVENT_CONNECTED://连接事件
        printf("cb connect:\r\n");
        break;

    case BT_EVENT_DISCONNECTD: //断开连接事件
        printf("cb disconnect:\r\n");
        break;

    case BT_EVENT_RCV_DATA://数据接收事件
        length = bt_rcv_data(bt_data_buf, sizeof(bt_data_buf), character_uuid);
        if (length)
        {
            printf("rcv data [%d] ok,\r\n", length);
            //bt_snd_data(bt_data_buf, length, USER_IDX_WRITE_NOTIFY_CHAR);
        }
        else
        {
            printf("rcv data err,\r\n");
        }
        break;

    default:
        break;
    }
}

int main(void)
{
    const char *name = "WB452_BLE";
```

```
const char *addr = "10:20:30:a0:b0:c2";
//const char *response = "bt salve say hello";
int32_t ret;

/* System Clocks Configuration */
RCC_Configuration();

/* USART ForPrintf */
USART_Config(256000);
printf("system start...\r\n");

memcpy(bt_init.device_name, name, MIN(strlen(name), sizeof(bt_init.device_name)));
memcpy(bt_init.device_addr, addr, MIN(strlen(addr), sizeof(bt_init.device_addr)));

bt_init.service[0].svc_uuid          = SERVICE_UUID;
bt_init.service[0].character[0].uuid = USER_IDX_WRITE_NOTIFY_CHAR;
bt_init.service[0].character[0].permission = BT_WRITE_PERM | BT_NTF_PERM;

ret = bt_ware_init(&bt_init, (bt_event_callback_handler_t)bt_event_callback_func);
if (ret != BT_RET_SUCCESS)
{
    printf("btinit failed.\r\n");
}
else
{
    printf("btinit ok.\r\n");
}

smartlock_touch_tim6_init();//定时周期 5ms

while(1)
{
    bt_run_thread();
}
}
```

## 5.3. BLE 低功耗配置

### 5.3.1. 蓝牙广播参数设置

在 BLE 开发过程，可以根据应用场景和对功耗的要求，配置蓝牙广播时间间隔；

```
app_env.adv_para.adv_int_min = 0x320; //广播间隔时间最小值：0.5S = 0x320*0.625 ms
```

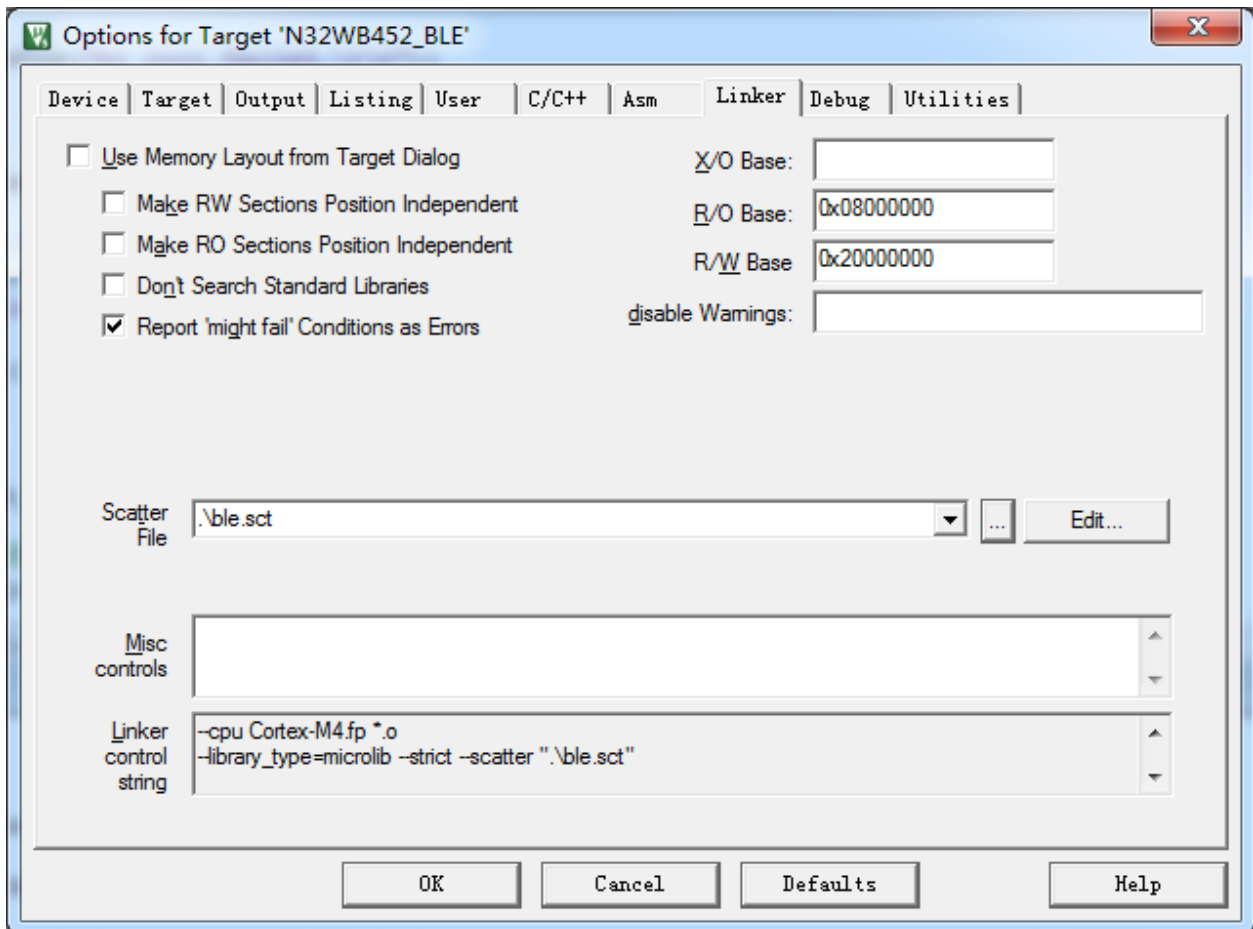
```
app_env.adv_para.adv_int_max = 0x640; //广播间隔时间最大值：0.5S = 0x320*0.625 ms
```

### 5.3.2. 蓝牙低功耗配置

MCU 在进入低功耗 STOP2 模式时，需要把蓝牙相关资源放到 0x20000~0x24000 地址段的 SRAM 中，以便从蓝牙唤醒时，快速获取蓝牙协议栈资源，如下图所示：

```
RW_IRAM2 0x20020000 0x00004000 {; powerdown variables
n32wb452_ble_api.o (+RW +ZI)
user_task.o ;(+RW +ZI)
main.o (+RW +ZI)
n32wb452_data_fifo.o (+RW +ZI)
app_sec.o (+RW +ZI)
app_task.o (+RW +ZI)
interface.o (+RW +ZI)
gapc.o (+RW +ZI)
gattc.o (+RW +ZI)
gapc_task.o (+RW +ZI)
gattc_task.o (+RW +ZI)
gattm_task.o (+RW +ZI)
l2cc_task.o (+RW +ZI)
gapm_task.o (+RW +ZI)
l2cc.o (+RW +ZI)
app.o (+RW +ZI)
app_user.o (+RW +ZI)
prf.o (+RW +ZI)|
ke_task.o (+RW +ZI)
eif_spi.o (+RW +ZI)
ke.o (+RW +ZI)
ke_event.o (+RW +ZI)
gattm.o (+RW +ZI)
rwip.o (+RW +ZI)
gapm.o (+RW +ZI)
h4t1.o (+RW +ZI)
hci.o (+RW +ZI)
hci_t1.o (+RW +ZI)
l2cm.o (+RW +ZI)
stdout.o (+RW +ZI)
rand.o (+RW +ZI)
startup_n32wb452.o (+RW +ZI)
}
```

在工程配置中，加入加载文件：



### 5.3.3. 低功耗唤醒

蓝牙从低功耗唤醒时，最先进入 IRQ 中断服务函数 EXT\_I15\_10\_IRQHandler(), 蓝牙唤醒后，需要马上配置蓝牙驱动相关资源，Cortex-M4 内核才能与蓝牙协处理器正常通信。

```

/**
 * @brief Inserts a delay time.
 * @param count specifies the delay time length.
 */
void EXT_I15_10_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_LINE14) != RESET)
    {
        //When the MCU is in STOP2 mode, it is woken up by Bluetooth, and the peripherals used need to be reconfigured
        if (flag_bt_enter_sleep == 1)
        {
            flag_bt_enter_sleep = 0;
            mainboard_exit_stop2();
        }
        //log_debug("exti 14 irq %d.\r\n", flag_bt_enter_sleep);
        bt_handler();
        /* Clears the SEL Button EXTI line pending bits. */
        EXTI_ClrITPendBit(EXTI_LINE14);
    }
}

```



```
//After exiting from STOP2 mode, all MCU resources used need to be reconfigured
void mainboard_exit_stop2(void)
{
    SystemInit();
    SetSysClock_HSE_PLL(RCC_PLL_MUL_18);
    RCC_Configuration();
    log_init();
    log_debug("system restart...\r\n");
    led_gpio_init();
    LED3_ON();
    ble_hardware_reinit();
    TIM6_init();
    flag_bt_enter_sleep = 0;
    system_10s_flag = 0;
}
```

## 6. 版本历史

版本	日期	备注
V1.0	2020-05-29	创建文档
V1.1.5	2021-3-23	删除心跳守护机制及相关 API，增加蓝牙状态监控机制及状态监控、回调函数说明
V1.2	2022-3-29	删除未使用的计时函数，增加等待蓝牙状态反馈函数说明
V1.3	2023-6-20	蓝牙组件资源需求增加 EXTI6/14 与定时器 TIM3

## 7. 声明

国民技术股份有限公司（下称“国民技术”）对此文档拥有专属产权。依据中华人民共和国的法律、条约以及世界其他法域相适用的管辖，此文档及其中描述的国民技术产品（下称“产品”）为公司所有。

国民技术在此并未授予专利权、著作权、商标权或其他任何知识产权许可。所提到或引用的第三方名称或品牌（如有）仅用作区别之目的。

国民技术保留随时变更、订正、增强、修改和改良此文档的权利，恕不另行通知。请使用者在下单购买前联系国民技术获取此文档的最新版本。

国民技术竭力提供准确可信的资讯，但即便如此，并不推定国民技术对此文档准确性和可靠性承担责任。

使用此文档信息以及生成产品时，使用者应当进行合理的设计、编程并测试其功能性和安全性，国民技术不对任何因使用此文档或本产品而产生的任何直接、间接、意外、特殊、惩罚性或衍生性损害结果承担责任。

国民技术对于产品在系统或设备中的应用效果没有任何故意或保证，如有任何应用在其发生操作不当或故障情况下，有可能致使人员伤亡、人身伤害或严重财产损失，则此类应用被视为“不安全使用”。

不安全使用包括但不限于：外科手术设备、原子能控制仪器、飞机或宇宙飞船仪器、所有类型的安全装置以及其他旨在支持或维持生命的应用。

所有不安全使用的风险应由使用人承担，同时使用人应使国民技术免于因为这类不安全使用而导致被诉、支付费用、发生损害或承担责任时的赔偿。

对于此文档和产品的任何明示、默示之保证，包括但不限于适销性、特定用途适用性和不侵权的保证，国民技术可在法律允许范围内进行免责。

未经明确许可，任何人不得以任何理由对此文档的全部或部分进行使用、复制、修改、抄录和传播。