ZS110A-方案配置指南

发布 ***1.2.0***

2019 年 **04** 月 **11** 日

目录

1 文档介绍 **1**
- 1.1 文档目的 . . . 1
- 1.2 术语说明 . . . 1
- 1.3 参考文档 . . . 1
- 1.4 版本历史 . . . 2

2 配置方式说明 **3**
- 2.1 静态配置方式 . . . 3
- 2.2 动态配置方式 . . . 5

3 SDK 相关配置 **6**
- 3.1 系统相关功能配置 . . . 6
- 3.2 蓝牙相关功能配置 . . . 10
- 3.3 板级硬件相关配置 . . . 13
- 3.4 应用层相关功能配置 . . . 17

CHAPTER 1

文档介绍

1.1 文档目的

介绍本项目中的配置方式和常用的配置项。

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）

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
2019-03-02	1.1	增减部分应用配置	ZS110A 项目组
2019-04-10	1.2	红外自学习配置说明	ZS110A 项目组

CHAPTER 2

配置方式说明

ZS110A 方案的可配置项目提供两种配置方式。

- 静态配置方式：采用宏定义的方式来配置，在编译期间就固定下来，运行期间无法修改。
- 动态配置方式：采用 nvram 的方式来配置，配置项存放在 NOR 上，运行期间可以动态修改。

2.1 静态配置方式

宏定义分布在 3 个头文件中（以 ble 遥控器为例）：

- rom_zephyr 的配置文件：\include\generated\autoconf.h
- 板级相关的配置文件:\samples\voice_rcu\src\peripheral_rmc\include\rmc_atb110*_*.h
- 应用配置主文件：

  \samples\voice_rcu\src\peripheral_rmc\include\autoconf_rmc.h

  应用配置文件会 include 前两个配置文件：先 include autoconf.h，然后 include rmc_atb110*_*.h，并修改配置项。

```
/* common config */
#include "autoconf.h"

/* application config */

/* msg type */
enum {
```

(continues on next page)

(续上页)

```
        MSG_NULL,

        MSG_KEY_INPUT,
        MSG_LOW_POWER,

        MSG_BLE_STATE,
        MSG_AUDIO_INPUT,
        MSG_APP_TIMER,
        MSG_OTA_EVENT,
};

/* BT */
#undef CONFIG_BT_DEVICE_NAME
#undef CONFIG_BT_MAX_CONN
#undef CONFIG_BT_MAX_PAIRED
#undef CONFIG_BT_RX_BUF_COUNT
#undef CONFIG_BT_L2CAP_TX_BUF_COUNT

#define CONFIG_BT_DEVICE_NAME "BLE_RMC"
#define CONFIG_BT_DEVICE_NAME_1 "BLE_RMC_TEST_1"
#define CONFIG_BT_DEVICE_NAME_2 "BLE_RMC_TEST_2"
#define CONFIG_BT_DEVICE_NAME_3 "BLE_RMC_TEST_3"
#define CONFIG_BT_DEVICE_NAME_4 "BLE_RMC_TEST_4"

#define CONFIG_BT_MAX_CONN 1
#define CONFIG_BT_MAX_PAIRED 1
#define CONFIG_BT_RX_BUF_COUNT 3
#define CONFIG_BT_L2CAP_TX_BUF_COUNT 4

/* STACK */
#undef CONFIG_IDLE_STACK_SIZE
#undef CONFIG_BT_RX_STACK_SIZE

#define CONFIG_IDLE_STACK_SIZE (256+256)
#define CONFIG_BT_RX_STACK_SIZE (1280 - 300)

/* choose wp profile or hid profile for voice */
#define CONFIG_USE_HID_PROFILE 1

/* set it if use undirect adv for reconncting, IOS must set
 ↪it, optional for android */
//#define CONFIG_USE_UNDIRECT_ADV_FOR_RECONN 1

/* choose appropriate encode algorithm */
#define USE_AL_ENCODE_3_4_1

//#include "rmc_atb1103_yt.h"
#include "rmc_atb1103_yt_v2.h"
//#include "rmc_atb1103_yt_v21.h"
```

(continues on next page)

(续上页)

```
//#include "rmc_atb110x_dvb_v10.h"
```

2.2 动态配置方式

动态配置项存储在\samples\voice_rcu\nvram.prop，用于存放系统以及应用相关的配置

配置格式：

一般以 Key=value 的格式配置，key 和 value 的值都是以字符串方式存储和读取，不支持其他类型，所以如果是整形和其他的类型，都需要用户自己转换。

```
BT_ADDR=11:22:33:44:0b:10
```

通过 nvram.prop 配置的配置项目，在打包固件的时候会生成 nvram.bin，并在量产过程烧写到 nor 上的 nvram 分区。

CHAPTER 3

SDK 相关配置

本 SDK 支持多个方面的配置, 主要包括以下几个方面

- 系统相关功能配置
- 蓝牙相关功能配置
- 板级硬件相关配置
- 应用层相关功能配置

3.1 系统相关功能配置

1. shell 配置：

   通过静态配置的方式，修改文件为:

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

   使能 shell 配置，用户通过命令行方式对用户程序进行交互和 debug.

```
/* shell */
#define CONFIG_KERNEL_SHELL 1

#define CONFIG_CONSOLE_SHELL 1
#define CONFIG_CONSOLE_SHELL_MAX_CMD_QUEUED 3
#define CONFIG_CONSOLE_SHELL_STACKSIZE 1024
#define CONFIG_UART_CONSOLE_INIT_PRIORITY 60
```

   注解:

   - 在代码空间紧张的时候，建议关掉此功能

- 应用程序处于休眠状态，串口通常无法操作，使用 shell 功能时，建议将休眠功能关掉

2. 应用程序打印 log 级别控制配置：

   通过静态配置的方式，修改文件为:

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* printk & sys_log */
#define CONFIG_PRINTK 1
#define CONFIG_SYS_LOG 1
#define CONFIG_SYS_LOG_SHOW_TAGS 1
#define CONFIG_SYS_LOG_DEFAULT_LEVEL SYS_LOG_LEVEL_INFO
#define CONFIG_SYS_LOG_OVERRIDE_LEVEL 0
#define CONFIG_UART_CONSOLE_ON_DEV_NAME "UART_0"
```

   其中，

   - CONFIG_PRINTK 是否关掉整个系统的打印功能，包含 printk 以及 sys_log
   - CONFIG_SYS_LOG 是否关掉 sys_log 功能
   - CONFIG_SYS_LOG_DEFAULT_LEVEL sys_log 的默认打印级别配置，默认打印出 INFO 级别以上的打印
   - CONFIG_UART_CONSOLE_ON_DEV_NAME 配置打印的串口，默认使用 UART_0 输出打印

3. 栈大小配置

   栈大小通过静态配置的方式，修改文件为:

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

   在系统中存在多个线程，需要配置多个线程的栈。如果用户需要修改默认的栈大小，可以通过重定义宏的方式修改

```
/* workqueue */
#define CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE 1024

/* stack */
#define CONFIG_MAIN_STACK_SIZE 1024
#define CONFIG_IDLE_STACK_SIZE 256
#define CONFIG_ISR_STACK_SIZE 640
```

   注解: 栈的配置修改要特别慎重，避免出现栈溢出，或者是栈浪费的情况。

4. Kernel init 的初始化优先级配置：

   通过静态配置的方式，修改文件为:

\samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* init priority level */
#define CONFIG_KERNEL_INIT_PRIORITY_OBJECTS 30
#define CONFIG_KERNEL_INIT_PRIORITY_DEFAULT 40
#define CONFIG_KERNEL_INIT_PRIORITY_DEVICE 50
```

5. 异步消息配置

   通过静态配置的方式，修改文件为:

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* async msg */
#define CONFIG_NUM_PIPE_ASYNC_MSGS 10
#define CONFIG_NUM_MBOX_ASYNC_MSGS 10
```

6. nvram 分区起始地址和大小配置：

   系统 nvram 分区通过静态配置的方式，修改文件为:

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
#define CONFIG_NVRAM_CONFIG 1
#define CONFIG_NVRAM_ACTS_DRV_NAME "NVRAM"
#define CONFIG_NVRAM_STORAGE_DEV_NAME CONFIG_XSPI_NOR_ACTS_
↪DEV_NAME
#define CONFIG_NVRAM_CONFIG_INIT_PRIORITY 48
#define CONFIG_NVRAM_FACTORY_REGION_BASE_ADDR 0x70000
#define CONFIG_NVRAM_FACTORY_REGION_SIZE 0x2000
#define CONFIG_NVRAM_WRITE_REGION_BASE_ADDR 0x72000
#define CONFIG_NVRAM_WRITE_REGION_SIZE 0x4000
```

7. 板级无关的外设配置：

   通过静态配置的方式，修改文件为:

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* dma */
#define CONFIG_DMA_ACTS_DEVICE_INIT_PRIORITY 40

#define CONFIG_DMA_0_NAME "DMA_0"
#define CONFIG_DMA_0_IRQ_PRI 3

/* rtc */
#define CONFIG_RTC_0_NAME "RTC_0"
#define CONFIG_RTC_0_IRQ_PRI 0

/* pwm */
#define CONFIG_PWM_ACTS_DEV_NAME "PWM_0"

/* spi */
```

(continues on next page)

(续上页)

```
#define CONFIG_SPI_INIT_PRIORITY 70

#define CONFIG_SPI_1_NAME "SPI_1"
#define CONFIG_SPI_1_IRQ_PRI 0
#define CONFIG_SPI_1_DEFAULT_CFG 0x80
#define CONFIG_SPI_1_DEFAULT_BAUD_RATE 500000

#define CONFIG_SPI_2_NAME "SPI_2"
#define CONFIG_SPI_2_IRQ_PRI 0
#define CONFIG_SPI_2_DEFAULT_CFG 0x80
#define CONFIG_SPI_2_DEFAULT_BAUD_RATE 500000

/* spinor */
#define CONFIG_XSPI_NOR_ACTS_DEV_NAME "xspi_nor"
#define CONFIG_XSPI_NOR_ACTS_DEV_INIT_PRIORITY 45

/* adc */
#define CONFIG_ADC_INIT_PRIORITY 80
#define CONFIG_ADC_0_NAME "ADC_0"
#define CONFIG_ADC_0_IRQ_PRI 2

/* i2c */
#define CONFIG_I2C_INIT_PRIORITY 60

#define CONFIG_I2C_0_NAME "I2C_0"
#define CONFIG_I2C_0_DEFAULT_CFG 0x0
#define CONFIG_I2C_0_IRQ_PRI 0

#define CONFIG_I2C_1_NAME "I2C_1"
#define CONFIG_I2C_1_DEFAULT_CFG 0x0
#define CONFIG_I2C_1_IRQ_PRI 0

/* gpio */
#define CONFIG_GPIO_ACTS_DRV_NAME "GPIO"
#define CONFIG_GPIO_ACTS_INIT_PRIORITY 20

/* input */
#define CONFIG_SYS_LOG_INPUT_DEV_LEVEL 0

/* inpu-matrix key */
#define CONFIG_INPUT_DEV_ACTS_MARTRIX_KEYPAD_NAME "MXKEYPAD"
#define CONFIG_INPUT_DEV_ACTS_MARTRIX_KEYPAD_NAME_IRQ_PRI 0

/* input-adckey */
#define CONFIG_INPUT_DEV_ACTS_ADCKEY_NAME "ADCKEY"

/* input-ir */
#define CONFIG_IRC_ACTS_DEV_NAME "IRC"
```

(continues on next page)

(续上页)

```
/* audio_in */
#define CONFIG_AUDIO_IN_ACTS_NAME "AUDIOIN"

/* watchdog */
#define CONFIG_WDG_ACTS_DEV_NAME "WATCHDOG"
```

8. 系统休眠配置

   通过静态配置的方式，修改文件为:

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* deepsleep */
#define CONFIG_DEEPSLEEP 1
#define CONFIG_DEEPSLEEP_TICK_THRESH 10
#define CONFIG_DEEPSLEEP_SWITCH_32M 1
```

   其中，

   - CONFIG_DEEPSLEEP 是否允许 MCU 进入深入休眠状态
   - CONFIG_DEEPSLEEP_TICK_THRESH 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
   - CONFIG_DEEPSLEEP_SWITCH_32M MCU 进入深入休眠状态是否关掉 32M，切换到 3M 时钟

3.2 蓝牙相关功能配置

1. 蓝牙地址配置：

   设备的蓝牙地址是每个设备唯一的标志，需要通过 ATT 工具来烧写。在没有烧写地址的时候，系统使用默认地址。

   烧写对应的配置项是定义在 nvram 区的 BT_ADDR

```
BT_ADDR=11:22:33:44:0b:10
```

2. 蓝牙设备名称配置：

   蓝牙设备名称采用静态配置的方式，修改文件：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* gap name */
#define CONFIG_BT_DEVICE_NAME "Zephyr"
#define CONFIG_BT_DEVICE_APPEARANCE 0
```

   其中，蓝牙名称配置通常在 2 个地方使用：

- 蓝牙广播数据包，对端设备在扫描广播包即可得到名称。
- GAP service 中蓝牙名称，对端设备在连接，发现，访问 GAP 属性服务才能得到名称。

注解:

- 在使用过程中，注意保持一致。
- 在 IOS 设备中，首次扫描阶段显示的名称是广播包的名称，连接后，将会使用 GAP 属性的名称。

---

3. 蓝牙协议栈的连接及配对个数的配置：

   采用静态配置的方式，修改文件：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* host stack CONN Configuration*/
#define CONFIG_BT_MAX_CONN 4
#define CONFIG_BT_MAX_PAIRED 2
```

   其中，

   - CONFIG_BT_MAX_CONN 允许支持的最大连接设备的个数
   - CONFIG_BT_MAX_PAIRED 允许支持的最大配对设备的个数，小于或等于最大连接个数

注解: ATB1103 及 ATB1109 支持的最大连接个数小于或等于 4，主要受控制器的资源限制，如果配置大于 4 个，将会异常

---

4. 蓝牙协议栈所需的 buffer 配置

   采用静态配置的方式，修改文件：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* host stack buffer config */
#define CONFIG_BT_HCI_CMD_COUNT 2
#define CONFIG_BT_RX_BUF_COUNT 4
#define CONFIG_BT_RX_BUF_LEN 76
#define CONFIG_BT_L2CAP_TX_BUF_COUNT 3
#define CONFIG_BT_L2CAP_TX_MTU 65
#define CONFIG_BT_CONN_TX_MAX 7
```

   其中，

   - CONFIG_BT_HCI_CMD_COUNT 用于发送 HCI 命令 buffer 的个数
   - CONFIG_BT_RX_BUF_COUNT 用于接收 HCI 命令/事件/数据/buffer 的个数

- CONFIG_BT_RX_BUF_LEN 用于接收 HCI 命令/事件/数据/buffer 的大小
- CONFIG_BT_L2CAP_TX_BUF_COUNT 用于发送 HCI ACL 数据包 buffer 的个数
- CONFIG_BT_L2CAP_TX_MTU 用于发送 HCI ACL 数据包 buffer 的大小
- CONFIG_BT_CONN_TX_MAX 最大支持能够并发发送 ACL 数据包的个数, 能够并发 TX buffer 个数受限制与 BLE 控制器 TX BUFFER 的个数，当前控制器 TX buffer 的个数默认配置为 4，即最终协议栈并发 TX buffer 个数取 2 者之中的最小值。控制器 ACL buffer 的配置, 修改文件: \subsys\bluetooth\host\bt_host_config_init.c

```
/* Configure ACL buffers for ctrl*/
hci_vs_cfg_acl_bufs(4, 4, 128);
```

5. 蓝牙协议栈大小配置

   采用静态配置的方式，修改文件：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* bt thread stack size*/
#define CONFIG_BT_HCI_TX_STACK_SIZE 256
#define CONFIG_BT_RX_STACK_SIZE 1024
```

   其中，

   - CONFIG_BT_HCI_TX_STACK_SIZE 用于配置蓝牙 TX 线程栈的大小
   - CONFIG_BT_RX_STACK_SIZE 用于配置蓝牙 RX 线程栈的大小

   注解: 栈的配置修改要特别慎重，避免出现栈溢出，或者是栈浪费的情况。

6. 蓝牙协议栈 ATT 层配置

   采用静态配置的方式，修改文件：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* att */
#define CONFIG_BT_ATT_PREPARE_COUNT 0
#define CONFIG_BT_ATT_TX_MAX 2
```

   其中，

- CONFIG_BT_ATT_PREPARE_COUNT 如果设置大于 0，表示 ATT 层支持并发 perpare 写的个数，默认不支持，以节省空间
- CONFIG_BT_ATT_TX_MAX 配置 ATT 层能够并发发送 TX request 类型命令的个数

7. 蓝牙私有地址配置

采用静态配置的方式，修改文件：

\samples\voice_rcu\src\peripheral_rmc\include\autoconf.h

```
/* private addr */
#define CONFIG_BT_PRIVACY 0
#define CONFIG_BT_RPA_TIMEOUT 900
```

其中，

- CONFIG_BT_PRIVACY 是否支持私有地址
- CONFIG_BT_RPA_TIMEOUT 定时更换私有地址的时间，默认 900 秒，15 分钟

注解: 协议栈的私有特性，默认配置已经支持。

3.3 板级硬件相关配置

板级硬件相关配置主要是板子对应的 GPIO 的配置，都采用静态配置的方式，通过修改：

\samples\voice_rcu\src\peripheral_rmc\include\rmc_atb110*_*.h

重新编译即生效

1. gpio/mfp 初始值配置:

```
#define BOARD_PIN_CONFIG        \
        {4, 4 | GPIO_CTL_SMIT | GPIO_CTL_PADDRV_LEVEL(3)},\
        {0, 13 },\
        {1, 13 },\
        {2, 13 },\
        {3, 13 },\
        {8, 13 },\
        {9, 13 }, \
        {18,2},

#define CONFIG_IRC_TX_PIN 5
#define CONFIG_IRC_RX_PIN 5
```

其中，

- BOARD_PIN_CONFIG gpio 的 pin 通常用于固定功能，在系统初始化时，进行配置，如 gpio4 用于 UART_1 打印，gpio0 gpio1 gpio2 gpio3 gpio8 gpio9 用于矩阵按键，gpio18 用于 pwm
- CONFIG_IRC_TX_PIN/CONFIG_IRC_RX_PIN 一般是驱动动态复用的 pin，当驱动使用时，用于驱动功能，当驱动不使用时，恢复到初始化功能

注解: 一般不建议动态复用，除非对场景的使用比较熟悉，否则会造成一定问题，如 GPIO4 用于 TX 的情况，动态复用成红外，打印时，打印输出会影响红外外围电路，并功耗增加。

---

2. uart 配置:

```
#define CONFIG_UART_1 1
```

其中，

- CONFIG_UART_1 使能 UART_1 的驱动，详细见 \drivers\uart\uart_acts.c

注解: 如果使能 UART_0 驱动，需要 #define CONFIG_UART_0 1，详细见\drivers\uart\uart_acts.c

---

3. led 相关 gpio/pwm 配置:

```
#define CONFIG_USE_PWM_LED  1

/* all managed led */
#define LED_PIN_CONFIG \
                        {18, 3, 0}, \

/* app led pin */
#define LED_LPOWER_PIN  18
#define LED_PAIR_PIN    18
#define LED_BTN_PIN     18
#define LED_IR_BTN_PIN  18
```

其中，

- CONFIG_USE_PWM_LED 使用 PWM 模式代码，详细见 \ext\lib\actions\hal\led_hal.c
- LED_PIN_CONFIG 配置 LED 的 GPIO PIN、PWM channel 以及 LED 极性
- LED_LPOWER_PIN 配置应用层低电灯使用的 GPIO PIN
- LED_PAIR_PIN 配置应用层配对灯使用的 GPIO PIN
- LED_BTN_PIN 配置应用层按键灯使用的 GPIO PIN

- LED_IR_BTN_PIN 配置应用层红外键灯使用的 GPIO PIN

此外，如果要使能 GPIO 方式可以这样修改：

```
#define CONFIG_USE_GPIO_LED 1

                /* all managed led */
#define LED_PIN_CONFIG \
                        {18, NONE_PWM_CHAN, 0}, \

/* app led pin */
#define LED_LPOWER_PIN  18
#define LED_PAIR_PIN    18
#define LED_BTN_PIN     18
#define LED_IR_BTN_PIN  18
```

4. 矩阵按键相关配置：

key 的映射关系通过直接修改头文件的方式配置，修改文件：

\samples\voice_rcu\src\peripheral_rmc\include\rmc_atb110*_*.h

修改在编译期间生效，修改后直接编译即生效。

按键经过 2 级映射：

- key 寄存器值映射到 key 值

  在配置中是以 {key_info0_value，key_info1_value，key_info2_value，key_value} 的关系来映射的。下表一共定义了 14 个键。

```
/* key reg val mapping to key val */
#define MXKEYPAD_MASK 0x0f
#define MXKEYPAD_MAPS \
 {0x00000102, 0x00000000, 0x00000000, KEY_1}, \
 {0x00010004, 0x00000000, 0x00000000, KEY_2}, \
 {0x00000000, 0x00000000, 0x00000001, KEY_3}, \
 {0x00000000, 0x00000000, 0x00000800, KEY_4}, \
 {0x00000000, 0x00000000, 0x00000100, KEY_5}, \
 {0x04080000, 0x00000000, 0x00000000, KEY_6}, \
 {0x02000800, 0x00000000, 0x00000000, KEY_7}, \
 {0x01000008, 0x00000000, 0x00000000, KEY_8}, \
 {0x00000000, 0x00000000, 0x00000200, KEY_9}, \
 {0x00020400, 0x00000000, 0x00000000, KEY_10}, \
 {0x00000000, 0x00000000, 0x00000004, KEY_11}, \
 {0x00000000, 0x00000000, 0x00000400, KEY_12}, \
 {0x00000000, 0x00000000, 0x00000002, KEY_13}, \
 {0x01020408, 0x00000000, 0x00000000, KEY_14}, \
 {0x01000008, 0x00000000, 0x00000900, KEY_15}, \
 {0x00000000, 0x00000000, 0x00000009, KEY_16}, \
 {0x0509000c, 0x00000000, 0x00000000, KEY_17}, \
 {0x0300090a, 0x00000000, 0x00000000, KEY_18}, \
 {0x01000008, 0x00000000, 0x00000002, KEY_19}, \
```

- key 值映射成标准功能 key 和红外键码

```
/* key val mapping to standard key */
#define KEY_MAPS \
 {KEY_1, REMOTE_KEY_RIGHT, IR_KB_RIGHT}, \
 {KEY_2, REMOTE_KEY_OK, IR_KB_ENTER}, \
 {KEY_3, REMOTE_KEY_DOWN, IR_KB_DOWN}, \
 {KEY_4, REMOTE_KEY_UP, IR_KB_UP}, \
 {KEY_5, REMOTE_KEY_MUTE, IR_KB_MUTE}, \
 {KEY_6, REMOTE_KEY_LEFT, IR_KB_LEFT}, \
 {KEY_7, REMOTE_KEY_POWER, IR_KB_POWER}, \
 {KEY_8, REMOTE_KEY_BACK, IR_KB_ACBACK}, \
 {KEY_9, REMOTE_KEY_VOL_DEC, IR_KB_VOL_DEC}, \
 {KEY_10, REMOTE_KEY_HOME, IR_KB_ACHOME}, \
 {KEY_11, REMOTE_KEY_VOICE_COMMAND, KEY_RESERVED}, \
 {KEY_12, REMOTE_KEY_VOL_INC, IR_KB_VOL_INC}, \
 {KEY_13, REMOTE_KEY_MENU, IR_KB_MENU}, \
 {KEY_14, REMOTE_COMB_KEY_OK_BACK, KEY_RESERVED}, \
 {KEY_15, REMOTE_COMB_KEY_TEST_ONE, KEY_RESERVED}, \
 {KEY_16, REMOTE_COMB_KEY_TEST_TWO, KEY_RESERVED}, \
 {KEY_17, REMOTE_COMB_KEY_TEST_THREE, KEY_RESERVED}, \
 {KEY_18, REMOTE_COMB_KEY_TEST_FOUR, KEY_RESERVED}, \
 {KEY_19, REMOTE_COMB_KEY_HCI_MODE, KEY_RESERVED}, \
```

5. 红外自学习功能代码使能配置：

   板级硬件是否增加红外自学习功能的配置，采用静态配置的方式，通过修改：

   \samples\voice_rcu\src\peripheral_rmc\include\rmc_atb110*_*.h

   重新编译即生效

```
#define CONFIG_IRC_SW_RX
#define CONFIG_IRC_TX_PIN 4
#define CONFIG_IRC_RX_PIN 5
```

   其中，

   - CONFIG_IRC_SW_RX 用于板级是否增加红外自学习功能
   - CONFIG_IRC_TX_PIN 用于配置红外发送的 pin
   - CONFIG_IRC_RX_PIN 用于配置红外接收的 pin

   注解: 如果版型不需要红外自学习功能，请不要使能 CONFIG_IRC_SW_RX，以免浪费 CODE 空间和 RAM 空间

6. 红外自学习按键相关配置：

   自学习 key 的映射关系通过直接修改头文件的方式配置，修改文件：

   \samples\voice_rcu\src\peripheral_rmc\include\rmc_atb110*_*.h

修改在编译期间生效，修改后直接编译即生效。

红外自学习按键经过 3 级映射：

- key 寄存器值映射到 key 值

  在配置中是以 {key_info0_value，key_info1_value，key_info2_value，key_value} 的关系来映射的，具体说明见矩阵按键相关配置

- key 值映射成标准自学习功能 key

```
/* key val mapping to standard learn key */
#define LEARN_KEY_MAPS \
        {KEY_2,          TV_KEY_POWER}, \
        {KEY_7,          TV_KEY_VOL_INC}, \
        {KEY_8,          TV_KEY_VOL_DEC}, \
```

- 标准自学习功能 key 映射成 NVRAM 存储空间中的键值

```
/* learn key mapping to nv key */
#define NV_KEY_MAPS \
        {TV_KEY_POWER,          "tv_power"}, \
        {TV_KEY_VOL_INC,        "tv_vol_inc"}, \
        {TV_KEY_VOL_DEC,        "tv_vol_dec"}, \
```

注解: 学习区按键和非学习区按键配置要分开配置，学习区按键的具体码值是放在 NVRAM 储存空间中，没有学习之前，是没有值的。

3.4 应用层相关功能配置

应用层相关功能配置，主要达到以下几个目的：

- 根据应用需求，重新配置系统及蓝牙相关配置，以满足应用需求
- 通过一些配置，让应用能够定制出不同个性化需求

应用层配置通常都采用静态配置的方式，主要通过修改：

\samples\voice_rcu\src\peripheral_rmc\include\autoconf_rmc.h

也有少量应用层配置，需要修改：

\samples\voice_rcu\src\peripheral_rmc\include\rmc_atb110*_*.h

1. 重新配置的系统及蓝牙相关配置

   应用层配置通常都采用静态配置的方式，主要通过修改：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf_rmc.h

配置过程是：先 #undef CONFIG_* ，然后 #define CONFIG_* ，达到修改目的.

```
/* BT */
#undef CONFIG_BT_DEVICE_NAME
#undef CONFIG_BT_MAX_CONN
#undef CONFIG_BT_MAX_PAIRED
#undef CONFIG_BT_RX_BUF_COUNT
#undef CONFIG_BT_L2CAP_TX_BUF_COUNT

#define CONFIG_BT_DEVICE_NAME "BLE_RMC"
#define CONFIG_BT_DEVICE_NAME_1 "BLE_RMC_TEST_1"
#define CONFIG_BT_DEVICE_NAME_2 "BLE_RMC_TEST_2"
#define CONFIG_BT_DEVICE_NAME_3 "BLE_RMC_TEST_3"
#define CONFIG_BT_DEVICE_NAME_4 "BLE_RMC_TEST_4"

#define CONFIG_BT_MAX_CONN 1
#define CONFIG_BT_MAX_PAIRED 1
#define CONFIG_BT_RX_BUF_COUNT 3
#define CONFIG_BT_L2CAP_TX_BUF_COUNT 4

/* STACK */
#undef CONFIG_IDLE_STACK_SIZE
#undef CONFIG_BT_RX_STACK_SIZE

#define CONFIG_IDLE_STACK_SIZE (256+256)
#define CONFIG_BT_RX_STACK_SIZE (1280 - 300)
```

其中，

- CONFIG_BT_DEVICE_NAME 当前应用为遥控器，所以将蓝牙名称从 Zephyr 修改为 BLE_RMC
- CONFIG_BT_MAX_CONN 遥控器应用只需要支持一个连接
- CONFIG_BT_MAX_PAIRED 遥控器应用只需要支持与一个对等设备配对，多个设备与遥控器配对，将自动清除原来的配对信息
- CONFIG_BT_RX_BUF_COUNT 减少 RX buffer 个数，可以节省资源
- CONFIG_BT_L2CAP_TX_BUF_COUNT 主要为了适配语音数据包的个数，减少 BLE 语音发送的等待时间
- CONFIG_IDLE_STACK_SIZE 适量增加 IDLE 栈以满足复杂应用需求
- CONFIG_BT_RX_STACK_SIZE 适量增加 RX 栈以满足复杂应用需求

此外，还可以在应用的板级硬件配置中继续重新定义，以满足不同板级需求

如 rmc_atb1103_yt_v2.h 板级硬件配置中蓝牙名称的修改，以区分不

同的硬件板子

```
#undef CONFIG_BT_DEVICE_NAME
#define CONFIG_BT_DEVICE_NAME "BLE_YT_RMC"
```

2. 应用消息类型配置

   应用框架的消息机制，没有定义具体的消息处理，需要用户根据应用需求定义自己的消息类型

   采用静态配置的方式，主要通过修改：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf_rmc.h

```
/* msg type */
enum {
        MSG_NULL,

        MSG_KEY_INPUT,
        MSG_LOW_POWER,

        MSG_BLE_STATE,
        MSG_AUDIO_INPUT,
        MSG_APP_TIMER,
        MSG_OTA_EVENT,
};
```

3. 应用回连方式的选择配置

   在 ble 回连中，有 2 个方式：

   - 使用非定向广播用于回连，回连速度会略慢
   - 使用定向广播用于回连，回连速度快

   采用静态配置的方式，主要通过修改：

   \samples\voice_rcu\src\peripheral_rmc\include\autoconf_rmc.h

```
/* set it if use undirect adv for reconncting, IOS must set␣
↪it, optional for android */
//#define CONFIG_USE_UNDIRECT_ADV_FOR_RECONN 1
```

   - 非定向广播包用于回连，在 android 还是 IOS 设备兼容性较好
   - IOS 规格已经明确，IOS 设备不接收定向广播用于回连

4. 语音编码方式的选择配置

   遥控器应用中，设计 3 大类编码方式：

   - 压缩比为 16:1 编码算法，如 USE_AL_ENCODE_1
   - 压缩比为 8:1 编码算法，如 USE_AL_ENCODE_2_8_1
   - 压缩比为 4:1 编码算法，如 USE_AL_ENCODE_3_4_1 USE_AL_ENCODE_3_4_1_2

采用静态配置的方式，主要通过修改：

\samples\voice_rcu\src\peripheral_rmc\include\autoconf_rmc.h

```
/* choose appropriate encode algorithm */
#define USE_AL_ENCODE_1
```

- 如果配置编码算法为压缩比 8:1 及以上算法，ADC 自适应调整 16K—16bit 采样
- 默认 16：1 算法，ADC 16K_16bit 采样。
- 如果对接的 android APK，APK 端算法自适应解码

5. DIS Service 的配置

在 HID profile 的 central 设备中，通常会读取 DIS Service 信息来标识不同的 HID 设备，所以可以根据不同的板级配置不同的内容，以满足个性化需求。

采用静态配置的方式，主要通过修改：

\samples\voice_rcu\src\peripheral_rmc\include\rmc_atb110*_*.h

```
/* Manufacturer Name */
#define CONFIG_DIS_MANUFACTURER_NAME        "PT corp."

/* Model Number String */
#define CONFIG_DIS_MODEL                    "ATB110x"

/* PNP ID */
#define CONFIG_DIS_PNP_COMPANY_ID_TYPE     0x02
#define CONFIG_DIS_PNP_VENDOR_ID           0x54, 0x2B
#define CONFIG_DIS_PNP_PRODUCT_ID          0x00, 0x16
#define CONFIG_DIS_PNP_PRODUCT_VERSION     0x00, 0x00
```

List of Figures

List of Tables

1.1 术语说明 1
1.2 版本历史 2