TK8610 无线终端芯片

编程指南（SDK）

V1.0

修订记录

修订时间	修订版本	修订描述
2023-03-28	V1.0	初始版本

重要声明

TurMass™ 为上海道生物联技术有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

上海道生物联技术有限公司保留随时变更、订正、增强、修改和改良此文档的权利，本文档内容可能会在未提前知会的情况下不定期进行更新。

除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议都依赖于具体的操作环境，并且不构成任何明示或暗示的担保。

联系方式

地址：上海嘉定皇庆路 333 号上海智能传感器产业园区 4 幢 5 层

邮编：201899

电话：021-61519850

邮箱：info@taolink-tech.com

网址：www.taolink-tech.com

目录

图形目录

表格目录

1 引言

1.1 编写目的

本文通过介绍 TK8610 芯片的 SDK 架构、接口使用规范以及应用场景的实现示例，指导用户实现个性化的功能软件开发。

1.2 适用用户

预期读者：产品应用开发人员、技术支持人员等。

1.3 术语和缩略语

名称	英文全称	中文全称
GPIO	General-Purpose Input/Output	通用型输入输出
I2C	Inter-Integrated Circuit	二线制同步串行总线
UART	Universal Asynchronous Receiver/Transmitter	通用异步收发传输器
SPI	Serial Peripheral Interface	串行外部设备接口
PWM	Pulse Width Modulation	脉冲宽度调制
RTC	Real-Time Clock	实时时钟
WATCHDOG	Watch Dog	看门狗

2 TK8610 芯片简介

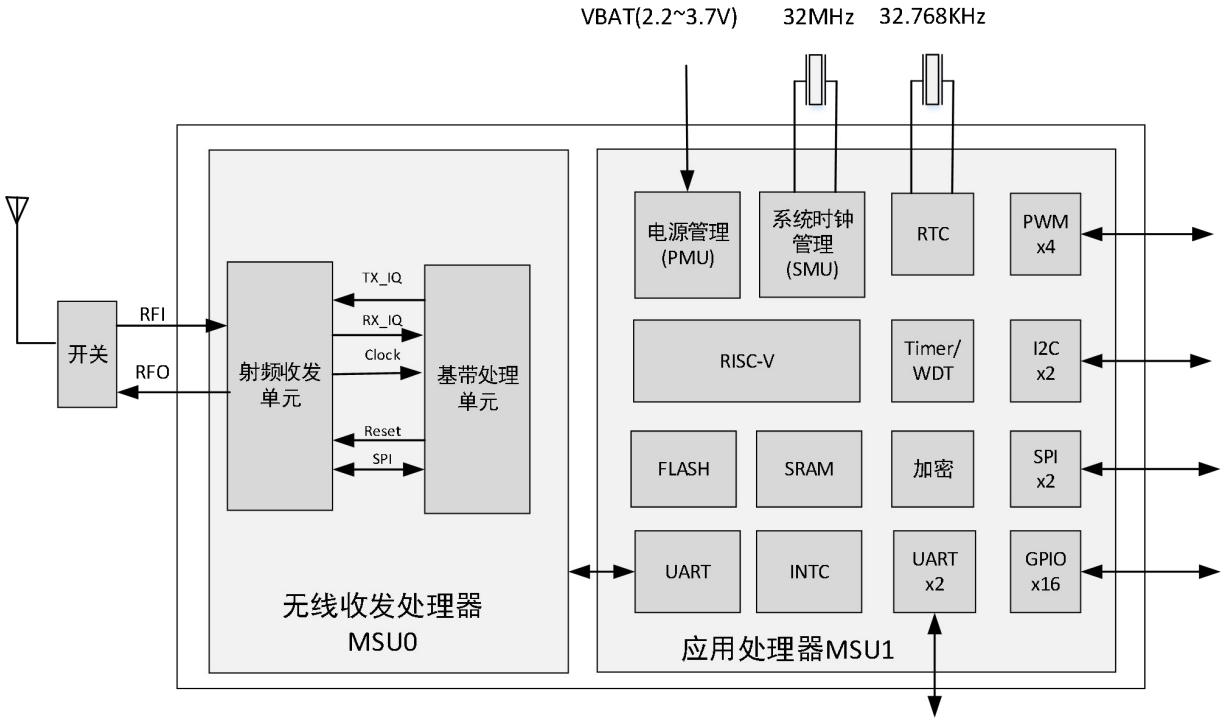

图 2-1 TK8610 功能框图

2.1 工作原理

TK8610 是一款采用 TurMass 技术的窄带无线通信 SoC 芯片，它内部包含两颗 MCU，分别称为 MSU1 和 MSU0。其中 MSU1 可以开放给用户使用，TK8610 的 SDK 运行在 MSU1 上，用户可以通过调用 SDK 所提供的各种接口函数进行应用开发；MSU0 主要负责射频和基带的处理和调度，不对外开放。MSU1 与 MSU0 通过串口以及核间中断进行通讯。

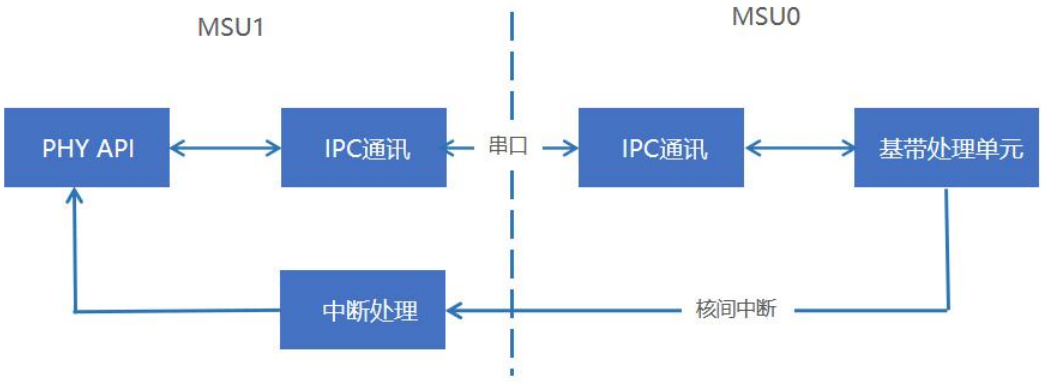

图 2-2 核间通讯原理

用户通过调用 SDK 提供的设置接口，将频点、速率模式、时隙等参数经过核间通信接口（IPC）发送给 MSU0，MSU0 再将相关参数配置到基带处理单元中。基带处理单元工作在一个周期循环的状态机中，状态机的每一个工作状态可以称为时隙（slot），状态切换中间需要有一定的间隔（interval），状态切换过程中通过核间中断通知 MSU1（如图 2-2 中 T0，T1…），应用程序调用 SDK 提供的查询接口，确认中断产生的原因，并进行相应的处理，如数据的接收或发送数据的准备等。

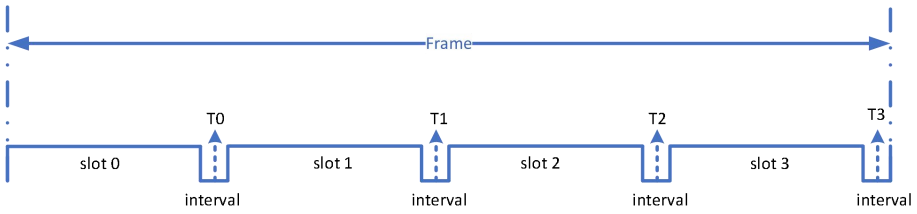

图 2-3 时隙与核间中断

2.2 时隙

TK8610 以时隙作为最小处理单位，按照功能的不同，时隙可以分为四种：信标时隙（BCN slot）、数据时隙（Data slot）、空闲时隙（IDLE slot），数据时隙又可以分为发送数据时隙（TX Data）和接收数据时隙（RX Data）。

序号	时隙类型	说明
1	IDLE slot	空闲时隙，芯片处于空闲状态
2	BCN slot	信标时隙，芯片处于信标的发送或接收处理状态 RX BCN 表示接收信标、TX BCN 表示发射信标
4	RX Data	数据接收时隙，芯片处于数据接收处理状态
5	TX Data	数据接收时隙，芯片处于数据发送处理状态

表 2-1 时隙类型说明

时隙的基本参数主要包括类型、长度、频率、功率、频带宽度等，每一个时隙可以独立配置不同的参数。

	属性	范围
时隙	时隙类型	例如：IDLE、TX BCN、RX BCN、RX DATA、TX DATA
	时间长度	例如：0~33553920us
	频率	例如：470~510MHz
	功率	例如：-32~17dBm
	频带宽度	例如：1KHz、2KHz、4KHz、8KHz、16KHz、32KHz、64KHz、128KHz

表 2-2 时隙参数说明

2.3 帧

1 个或多个紧密相连的时隙组成了帧（Frame），每一帧内的时隙可以相同，也可以不同。下图 2-3 为典型一个由 4 个不同时隙组成的帧结构：

图 2-4 典型 4 时隙的帧结构

3 接口设计

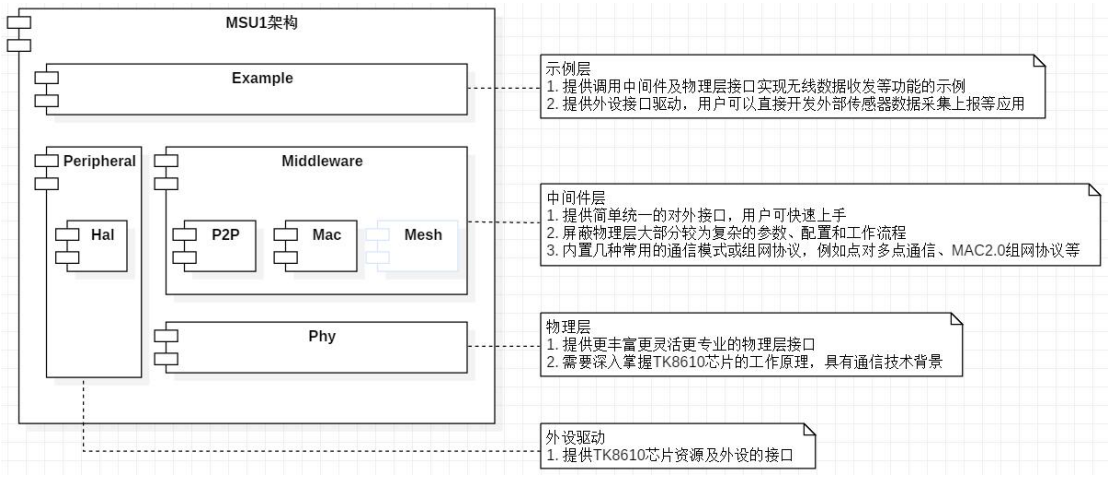

图 3-1 MSU1 软件架构

MSU1 软件架构主要分为三层：物理层、中间件层、示例层，以及必要的外设驱动，用户可以根据具体需求选用中间件层或物理层进行应用开发。

示例层为用户展现各种通信模式下的传感器数据采集上报、休眠唤醒等功能示例，用户可以根据自身需求在示例上进行二次开发。示例分为两大类：一类通过中间件的接口实现，另一类通过物理层接口实现。

中间件层接口适合绝大多数开发人员使用，提供了一套极为简洁的接口，用户仅需在初始化时设置最基本的类型和参数，即可快速实现无线数据收发，而无需关心复杂的控制流程。中间件一共包含三个模块，即 P2P 模块，MAC 模块，MESH 模块。

物理层适合具有特定需求的专业通信开发人员，用户可以通过物理层所提供的更加丰富、灵活、多样化接口，实现各种个性化的功能，可最大限度发挥 TK8610 灵活的特点。

外设驱动提供 TK8610 片上资源及外设的接口，可以根据具体需求调用。

3.1 中间件模块介绍

- P2P 模块是针对 P2P 场景或配合 TKG-300 网关使用。
- MAC 模块是针对大型组网，增加了 MAC 协议，配合 TKG-800 网关使用。
- Mesh 模块是不带网关，具有自组网特性。待发布。

特性	P2P 模块	MAC 模块
配合网关组网	TKG-300 或 P2P	TKG-800
是否带协议	否	是
数据是否加密	否	是
是否有 ACK 确保可达性	否	是
是否支持丢包重传	否	是

是否支持分包重组	否	是

表 3-1 MAC 以及 P2P 模块对比

3.2 中间件接口表

序号	函数名	说明
1	tk86xx_init	中间件初始化，设置模块类型、模块配置、状态回调函数及数据接收的回调函数
2	tk86xx_poll	状态机轮询并处理耗时操作，需在主循环调用执行
3	tk86xx_config_update	配置更新，更新一个或多个配置项
4	tk86xx_config_get	配置查询
5	tk86xx_data_transmit	数据发送
6	tk86xx_control	功能控制

表 3-2 中间件接口表

3.3 外设接口表

3.3.1 Flash 接口表

序号	函数名	说明
1	flash_user_sector_erase	flash 扇区擦除，每次只能擦除 1 个扇区（4Kbyte）
2	flash_user_read	flash 上读数据
3	flash_user_write	flash 上写数据

表 3-3 Flash 接口表

3.3.2 GPIO 接口表

序号	函数名	说明
1	sysctrl_gpiomux_set	GPIO 引脚功能设置
2	gpio_init	GPIO 引脚初始化
3	gpio_dir_get	GPIO 引脚方向查询
4	gpio_dir_set	GPIO 引脚方向设置
5	gpio_pin_read	GPIO 引脚输出电平查询
6	gpio_pin_write	GPIO 引脚输出电平设置
7	gpio_pulldirection_set	GPIO 引脚模式设置，有上拉模式与下拉模式

表 3-4 GPIO 接口表

3.3.3 定时器接口表

序号	函数名	说明
1	timer_init	定时器初始化
2	timer_register_callback	注册定时器回调函数
3	timer_enable	设置定时器使能
4	timer_disable	设置定时器失效

表 3-5 定时器接口表

3.3.4 I2C 接口表

序号	函数名	说明
1	i2c_init	I2C 总线初始化
2	i2c_deinit	I2C 总线复位
3	i2c_start	I2C 总线上产生起始信号
4	i2c_stop	I2C 总线上产生停止信号
5	i2c_read	I2C 总线上读数据
6	i2c_write_onebyte	I2C 总线上写数据

表 3-6 I2C 接口表

3.3.5 UART 接口表

序号	函数名	说明
1	uart_init	UART 总线初始化
2	uart_read	UART 总线上读数据
3	uart_read_int	UART 总线上中断方式读数据
4	uart_write	UART 总线上写数据
5	uart_write_int	UART 总线上中断方式写数据

表 3-7 UART 接口表

3.3.6 SPI 接口表

序号	函数名	说明
1	spi_init	SPI 总线初始化
2	spi_read	SPI 总线上读数据
3	spi_write	SPI 总线上写数据

4	spi_write_read_byte	SPI 总线上读写数据

表 3-8 SPI 接口

3.3.7 PWM 接口表

序号	函数名	说明
1	pwm_init	PWM 初始化
2	pwm_update	PWM 配置更新
3	pwm_enable	设置 PWM 使能
4	pwm_disable	设置 PWM 失效

表 3-9 PWM 接口表

3.3.8 RTC 接口表

序号	函数名	说明
1	rtc_enable	设置 RTC 使能
2	rtc_disable	设置 RTC 失效
3	rtc_current_value_get	RTC 计数器查询
4	rtc_interrupt_enable	设置中断使能
5	rtc_interrupt_disable	设置中断失效
6	rtc_alarm_set	设置 RTC 定时
7	rtc_interrupt_flag_clear	RTC 中断标志位清空

表 3-10 RTC 接口表

3.3.9 WATCHDOG 接口表

序号	函数名	说明
1	watchdog_config	看门狗配置
2	watchdog_register_callback	注册看门狗超时回调函数
3	watchdog_enable	设置看门狗使能
4	watchdog_disable	设置看门狗失效
5	watchdog_feed	喂狗功能

表 3-11 WATCHDOG 接口表

4 接口定义

4.1 中间件接口定义

4.1.1 tk86xx_init

mid_ret_e tk86xx_init(mid_mode_e mode, mid_config_u *config, mid_callback_u *callback);

初始化函数用于设置模块类型、模块配置、状态回调函数及数据接收的回调函数。其中模块配置(config)、回调函数定义都是联合体，根据模块类型值(mode)的不同选取不同模块的配置。注意：callback 不能为空指针。

参数	参数定义
mode	输入参数，中间件模块选择。 typedef enum { P2P_MODE, // P2P 模块 MAC_MODE, // MAC 模块 } mid_mode_e;

config	输入参数，中间件的配置，当赋值 NULL 时配置取默认值。
	typedef union
	{
	p2p_config_t p2p_config; // P2P 模块的配置，详见 4.1.1.1
	mac_config_t mac_config; // MAC 模块的配置，详见 4.1.1.2
	} mid_config_u;

callback	输入参数，中间件的回调定义。
	typedef union
	{
	p2p_callback_t p2p_callback; // P2P 模块的回调，详见 4.1.1.3
	mac_callback_t mac_callback; // MAC 模块的回调，详见 4.1.1.4
	} mid_callback_u;

mid_ret_e	中间件返回值，详见 4.1.1.5

表 4-1 初始化函数

4.1.1.1 p2p_config_t

P2P 模块的配置。

结构体	结构体定义
p2p_config_t	typedef struct

	{
	p2p_work_mode_e work_mode; // 工作模式
	p2p_common_config_t common_config; // 公共配置，任何模式均需配置
	p2p_burst_config_t burst_config; // 突发模式的特定配置
	p2p_slot_config_t slot_config; // 时隙模式的特定配置
	p2p_wakeup_config_t wakeup_config; // 休眠唤醒配置
	} p2p_config_t;

p2p_work_mode_e	typedef enum
	{
	P2P_SLOT_MASTER = 11, // 时隙主机模式
	P2P_SLOT_SLAVE = 12, // 时隙从机模式
	P2P_SLOT_TKG300 = 13, // TKG-300 网关时隙模式
	P2P_BURST_NORMAL = 21, // 突发模式
	P2P_RX_SENSI = 72, // 灵敏度测试模式
	} p2p_work_mode_e;

表 4-2 P2P 模块配置

4.1.1.1.1 p2p_common_config_t

公共配置，任何模式均需配置。

结构体	结构体定义

p2p_common_config_t	typedef struct
	{
	p2p_rate_mode_e rate_mode; // 速率模式
	p2p_bcn_mode_e bcn_mode; // BCN 模式
	uint8_t bcn_id; // BCN 的 ID。取值范围：0~7
	uint32_t bcn_freq; // BCN 频率，470000000~510000000Hz
	uint32_t tx_freq; // 发送频率，470000000~510000000Hz
	uint32_t rx_freq; // 接收频率，470000000~510000000Hz
	tx_power_e tx_power; // 发射功率，功率表索引，0~15
	} p2p_common_config_t;

p2p_rate_mode_e	typedef enum
	{
	P2P_RAET_7 = 7, // 理论速率 202bps
	P2P_RATE_8 = 8, // 理论速率 404bps

	P2P_RATE_9 = 9, // 理论速率 808bps
	P2P_RATE_10 = 10, // 理论速率 1762bps
	P2P_RATE_11 = 11, // 理论速率 3523bps
	P2P_RATE_12 = 12, // 理论速率 7047bps
	P2P_RATE_13 = 13, // 理论速率 2578bps
	P2P_RATE_14 = 14, // 理论速率 5156bps
	P2P_RATE_15 = 15, // 理论速率 10313bps
	P2P_RATE_16 = 16, // 理论速率 20625bps
	P2P_RATE_17 = 17, // 理论速率 41250bps
	P2P_RATE_18 = 18, // 理论速率 82500bps
	} p2p_rate_mode_e;

p2p_bcn_mode_e	typedef enum
	{
	P2P_BCN_AUTO = 0, // 自动配置，按速率映射
	P2P_BCN_11 = 11, // BCN1.0 的模式 1
	P2P_BCN_12 = 12, // BCN1.0 的模式 2
	P2P_BCN_13 = 13, // BCN1.0 的模式 3
	P2P_BCN_14 = 14, // BCN1.0 的模式 4
	P2P_BCN_15 = 15, // BCN1.0 的模式 5
	P2P_BCN_16 = 16, // BCN1.0 的模式 6
	P2P_BCN_17 = 17, // BCN1.0 的模式 7
	P2P_BCN_21 = 21, // BCN2.0 的模式 1
	P2P_BCN_22 = 22, // BCN2.0 的模式 2
	P2P_BCN_23 = 23, // BCN2.0 的模式 3
	P2P_BCN_24 = 24, // BCN2.0 的模式 4
	P2P_BCN_25 = 25, // BCN2.0 的模式 5
	P2P_BCN_26 = 26, // BCN2.0 的模式 6
	} p2p_bcn_mode_e;

tx_power_e	typedef enum { TX_POWER_0, // 发送功率-20dbm TX_POWER_1, // 发送功率-16dbm TX_POWER_2, // 发送功率-10dbm TX_POWER_3, // 发送功率-6dbm TX_POWER_4, // 发送功率-4dbm

	TX_POWER_5, // 发送功率-2dbm TX_POWER_6, // 发送功率 0dbm TX_POWER_7, // 发送功率 2dbm TX_POWER_8, // 发送功率 6dbm TX_POWER_9, // 发送功率 8dbm TX_POWER_10, // 发送功率 10dbm TX_POWER_11, // 发送功率 12dbm TX_POWER_12, // 发送功率 15dbm TX_POWER_13, // 发送功率 17dbm TX_POWER_14, // 发送功率 17dbm TX_POWER_15, // 发送功率 17dbm } tx_power_e;

表 4-3 P2P 公共配置

4.1.1.1.2 p2p_burst_config_t

突发模式的特定配置。

结构体	结构体定义

p2p_burst_config_t	typedef struct
	{
	uint32_t dev_addr; // 本地地址
	p2p_filter_flag_e filter_flag; // 地址过滤
	} p2p_burst_config_t;

p2p_filter_flag_e	typedef enum
	{
	P2P_FILTER_OFF = 0, // 不进行地址过滤
	P2P_FILTER_ON = 1, // 进行地址过滤
	} p2p_filter_flag_e;

表 4-4 突发模式的特定配置

4.1.1.1.3 p2p_slot_config_t

时隙模式的特定配置。

结构体	结构体定义

p2p_slot_config_t	typedef struct
	{
	uint16_t tx_len_max; // 配置最大发送字节数，最大 600 字节
	} p2p_slot_config_t;

表 4-5 时隙模式的特定配置

4.1.1.1.4 p2p_wakeup_config_t

休眠唤醒配置。

结构体	结构体定义

p2p_wakeup_config_t	typedef struct
	{
	p2p_wakeup_src_e wakeup_source; // 唤醒源
	uint16_t wakeup_id; // 唤醒 ID，取值范围：1~720
	uint32_t wakeup_freq; // 唤醒信号载波频率（Hz）
	uint32_t wakeup_timing; // 定时唤醒时间间隔，单位 ms
	p2p_exti_level_e wakeup_level; // 电平唤醒触发条件
	uint32_t wakeup_cad_period; // 唤醒监听周期，单位 ms
	} p2p_wakeup_config_t;

p2p_wakeup_src_e	typedef enum
	{
	P2P_WAKEUP_NONE = 0, // 休眠后无唤醒源
	P2P_WAKEUP_EXTI_B0 = 0X0001, // 外部 IO B0 作为唤醒源
	P2P_WAKEUP_EXTI_B1 = 0X0002, // 外部 IO B1 作为唤醒源
	P2P_WAKEUP_EXTI_B2 = 0X0004, // 外部 IO B2 作为唤醒源
	P2P_WAKEUP_EXTI_B3 = 0X0008, // 外部 IO B3 作为唤醒源
	P2P_WAKEUP_EXTI_B4 = 0X0010, // 外部 IO B4 作为唤醒源
	P2P_WAKEUP_EXTI_B5 = 0X0020, // 外部 IO B5 作为唤醒源
	P2P_WAKEUP_EXTI_B6 = 0X0040, // 外部 IO B6 作为唤醒源
	P2P_WAKEUP_EXTI_B7 = 0X0080, // 外部 IO B7 作为唤醒源
	P2P_WAKEUP_TIMER = 0X0100, // 定时器作为唤醒源
	P2P_WAKEUP_WIRELESS = 0X0200, // 空中载波信号作为唤醒源
	} p2p_wakeup_src_e;

p2p_exti_level_e	typedef enum
	{
	P2P_EXTI_LOW = 0X00, // 低电平触发中断
	P2P_EXTI_HIGH = 0X01, // 高电平触发中断
	} p2p_exti_level_e;

表 4-6 P2P 休眠唤醒配置

4.1.1.2 mac_config_t

MAC 模块的配置。

结构体	结构体定义

mac_config_t	typedef struct
	{
	mac_base_config_t common_config; // 基础配置，均需配置
	mac_auto_uplink_config_t auto_send_config; // 自动发送配置
	mac_wakeup_config_t wakeup_config; // 休眠唤醒配置
	mac_dbg_config_t dbg_config; // 调试配置
	} mac_config_t;

表 4-7 MAC 模块配置

4.1.1.2.1 mac_base_config_t

MAC 基础配置。

结构体	结构体定义

mac_base_config_t	typedef struct
	{
	mac_dr_e dr; // 速率。0-低速率；1-中速率；2-高速率
	class_mode_e class_mode; // 工作模式。0-CLASS A；1-CLASS C
	join_mode_e join_mode; // 入网模式。0-OTAA；1-ABP
	uint8_t retrans; // 丢包重传次数，仅用于带确认的消息
	uint8_t dev_eui[8]; // 设备 ID
	uint8_t dev_addr[4]; // 设备地址，OTAA 入网模式时无需配置
	uint8_t app_key[16]; // AppKey
	uint8_t gw_chan_num; // 搜索网关的信道个数，最多 8 个
	uint32_t gw_chan_list[8]; // 搜索网关的信道列表（Hz）
	uint8_t relay_chan_num; // 搜索中继的信道个数，最多 16 个
	uint32_t relay_chan_list[16]; // 搜索中继信道列表（Hz）
	int32_t freq_offset; //频偏（Hz）。正数向上偏，负数向下偏
	uint8_t freq_hop; // 调频开关。0-关 ； 1-开
	tx_power_e tx_power; // 发射功率，功率表索引，0-15
	uint8_t power_adr; // 功控开关。0-关 ； 1-开
	uint8_t bcn_id; // BCN 的 ID。取值范围：0~7
	slot_mode_e slot_mode; // 时隙模式。0-6 时隙；1-8 时隙；2-10 时隙；3-12 时隙

	uint8_t layer; // 层级
	} mac_base_config_t;

mac_dr_e	typedef enum
	{
	MAC_DR0 = 0, // 低速率
	MAC_DR1, // 中速率
	MAC_DR2, // 高速率
	} mac_dr_e;

class_mode_e	typedef enum
	{
	MAC_CLASS_A = 0, // CLASS A 设备模式
	MAC_CLASS_C, // CLASS C 设备模式
	} class_mode_e;

join_mode_e	typedef enum
	{
	MAC_JOIN_OTAA = 0, // OTAA 入网模式
	MAC_JOIN_ABP, // ABP 入网模式
	} join_mode_e;

slot_mode_e	typedef enum { MAC_SLOT_MODE_6 = 0, // 6 时隙 MAC_SLOT_MODE_8 = 1, // 8 时隙 MAC_SLOT_MODE_10 = 2, // 10 时隙 MAC_SLOT_MODE_12 = 3, // 12 时隙 } slot_mode_e;
tx_power_e	typedef enum { TX_POWER_0, // 发送功率-20dbm TX_POWER_1, // 发送功率-16dbm TX_POWER_2, // 发送功率-10dbm TX_POWER_3, // 发送功率-6dbm TX_POWER_4, // 发送功率-4dbm TX_POWER_5, // 发送功率-2dbm TX_POWER_6, // 发送功率 0dbm TX_POWER_7, // 发送功率 2dbm

	TX_POWER_8, // 发送功率 6dbm TX_POWER_9, // 发送功率 8dbm TX_POWER_10, // 发送功率 10dbm TX_POWER_11, // 发送功率 12dbm TX_POWER_12, // 发送功率 15dbm TX_POWER_13, // 发送功率 17dbm TX_POWER_14, // 发送功率 17dbm TX_POWER_15, // 发送功率 17dbm } tx_power_e;

表 4-8 MAC 基础配置

4.1.1.2.2 mac_auto_uplink_config_t

MAC 自动发送配置。

结构体	结构体定义

mac_auto_uplink_config_t	typedef struct
	{
	uint8_t period; // 发送周期，单位：1 个 TDD 周期
	uint16_t uplink_bytes; // 发送长度，单位：1 个字节
	} mac_auto_uplink_config_t;

表 4-9 MAC 自动发送配置

4.1.1.2.3 mac_wakeup_config_t

MAC 休眠唤醒配置。

结构体	结构体定义

mac_wakeup_config_t	typedef struct
	{
	mac_wakeup_src_e wakeup_source; // 唤醒源 0~8
	mac_exti_level_e wakeup_lever; // 0-低电平 1-高电平
	uint32_t wakeup_period; // 唤醒周期，单位:ms
	} mac_wakeup_config_t;

mac_wakeup_src_e	typedef enum
	{
	MAC_WAKEUP_NONE = 0, // 休眠后无唤醒源
	MAC_WAKEUP_EXTI_B0 = 0X0001, //外部 IO B0 作为唤醒源
	MAC_WAKEUP_EXTI_B1 = 0X0002, //外部 IO B1 作为唤醒源
	MAC_WAKEUP_EXTI_B2 = 0X0004, //外部 IO B2 作为唤醒源

	MAC_WAKEUP_EXTI_B3 = 0X0008, //外部 IO B3 作为唤醒源
	MAC_WAKEUP_EXTI_B4 = 0X0010, //外部 IO B4 作为唤醒源
	MAC_WAKEUP_EXTI_B5 = 0X0020, //外部 IO B5 作为唤醒源
	MAC_WAKEUP_EXTI_B6 = 0X0040, //外部 IO B6 作为唤醒源
	MAC_WAKEUP_EXTI_B7 = 0X0080, //外部 IO B7 作为唤醒源
	MAC_WAKEUP_TIMER = 0X0100, //定时器作为唤醒源
	} mac_wakeup_src_e;

mac_exti_level_e	typedef enum
	{
	MAC_EXTI_LOW = 0X00, // 低电平触发中断
	MAC_EXTI_HIGH = 0X01, // 高电平触发中断
	} mac_exti_level_e;

表 4-10 MAC 休眠唤醒配置

4.1.1.2.4 mac_dbg_config_t

MAC 调试配置。

结构体	结构体定义

mac_dbg_config_t	typedef struct
	{
	dbg_type_e dbg_type; // MAC 调试类型
	int16_t nst_threshold; // 网络搜索阈值
	uint8_t ack_tmr_enable; // 应答超时使能值
	uint8_t rti_enable; // RX-->IDLE 使能值
	uint8_t print_mode; // 调试打印模式
	int16_t ref_rssi; // 功控参考 RSSI
	uint32_t single_tone_freq; // 单 TONE 发送频点
	uint32_t sensitivity_freq; // 接收灵敏度测试频点
	} mac_dbg_config_t;

dbg_type_e	typedef enum
	{
	NST_THRESHOLD_DBG = 0, // 网络搜索阈值配置
	ACK_TMR_ENABLE_DBG, // 应答超时使能配置
	RTI_ENABLE_DBG, // RX-->IDLE 转换使能配置
	PRINT_MODE_DBG, // 调试打印配置
	REF_RSSI_DBG, // 功控参考 RSSI 阈值配置

	} dbg_type_e;

表 4-11 MAC 调试配置

4.1.1.3 p2p_callback_t

P2P 模块回调。

结构体	结构体定义

p2p_callback_t	typedef struct
	{
	p2p_data_receive_callback_t receive_callback; // 数据回调，不能为空指针
	p2p_status_callback_t status_callback; // 状态回调，不能为空指针
	} p2p_callback_t;

p2p_data_receive_callback_t	输入参数，P2P 数据接收的回调函数
	typedef void (*p2p_data_receive_callback_t)(p2p_message_t *message, mid_quality_t *quality);
	其中 message 为接收的消息，quality 为信号质量
	p2p_message_t 的定义详见 4.1.5.1
	typedef struct {
	int16_t rssi; // 信号强度
	int16_t snr; // 信噪比
	} mid_quality_t;

p2p_status_callback_t	输入参数，P2P 状态的回调函数
	typedef void (*p2p_status_callback_t)(mid_status_e status);
	typedef enum
	{
	P2P_SEND_FINISH = 1001, // P2P 发送数据完成
	} mid_status_e; // P2P 状态

表 4-12 P2P 模块回调

4.1.1.4 mac_callback_t

MAC 模块回调。

结构体	结构体定义
mac_callback_t	typedef struct {

	mac_data_receive_callback_t receive_callback; // 数据回调
	mac_status_callback_t status_callback; // 状态回调
	} mac_callback_t;

mac_data_receive_callback_t	输入参数，MAC 数据接收的回调函数。
	typedef void (*mac_data_receive_callback_t)(mac_message_t *message, mid_quality_t *quality);
	其中 message 为接收的消息，quality 为信号质量
	mac_message_t 的定义详见 4.1.5.2
	typedef struct {
	int16_t rssi; // 信号强度
	int16_t snr; // 信噪比
	} mid_quality_t;

mac_status_callback_t	输入参数，MAC 状态的回调函数 typedef void (*mac_status_callback_t)(mid_status_e status, mac_status_param_t *status_param);

mid_status_e	typedef enum
	{
	MAC_SEARCH_SUCCESS = 2001, // 搜网成功
	MAC_JOIN_SUCCESS = 2002, // 入网成功（包括 ABP 及 OTAA）
	MAC_ONLINE = 2003, // 上线，表示可以正常收发数据
	MAC_DATA_PENDING = 2004, // 数据挂起
	MAC_DATA_SEND_SUCCESS = 2005, // 数据发送成功
	MAC_SEARCH_FAILED = 2101, // 搜网失败
	MAC_JOIN_FAILED = 2102, // 入网失败
	MAC_OFFLINE = 2103, // 下线，表示不能收发数据
	MAC_DATA_SEND_TIMEOUT = 2104, // 数据发送超时
	MAC_DATA_SEND_FAILED = 2105, // 数据发送失败
	} mid_status_e; // MAC 状态

mac_status_param_t	typedef struct
	{
	data_confirm_t data_confirm; // 数据发送结果信息
	net_search_t net_search; // 网络搜索结果信息
	} mac_status_param_t;

data_confirm_t	typedef struct {

	uint8_t ack_received; // 接收到 ACK 应答标记
	int8_t tx_power; // 发送时所用功率
	uint8_t nb_retries; // 重发次数
	} data_confirm_t;

net_search_t	typedef struct
	{
	uint8_t channel_type; // 信道类型 0-网关信道，1-中继信道
	uint32_t freq; // 信道频点
	int16_t snr; // 信噪比
	int16_t rssi; // 信号强度
	} app_net_search_status_param_t;

表 4-13 MAC 模块回调

4.1.1.5 mid_ret_e

中间件的返回值。

结构体	结构体定义

mid_ret_e	typedef enum
	{
	MID_SUCCESS = 0, // 接口调用成功
	MID_FAILED = -1, // 接口调用失败
	MID_TIMEOUT = -2, // 接口调用超时
	MID_PARM_ERR = -3, // 接口调用参数错误
	
	MID_PHY_SUCCESS = 0, // 成功
	MID_PHY_FAIL = -50, // 失败
	MID_PHY_PARAM_ERR = -51, // 参数错误
	MID_PHY_PARAM_LEN_ERR = -52, // 参数长度错误
	MID_PHY_TIMEOUT = -53, // 超时
	MID_PHY_PARAM_WAKEUP_MODE_ERR = -54, // 唤醒模式错误
	MID_PHY_PARAM_WAKEUP_IO_LEVEL_ERR = -55, // IO 唤醒电平错误
	MID_PHY_PARAM_WAKEUP_TIMER_TIME_ERR = -56, // 定时唤醒时间错误
	MID_PHY_PARAM_WAKEUP_ID_ERR = -57, // 空中信号唤醒 ID 错误
	MID_PHY_PARAM_WAKEUP_FREQ_ERR = -58, // 空中信号唤醒频率错误

	MID_PHY_PARAM_WAKEUP_CAD_PERIOD_ERR = -59, // 空中信号唤醒周期错误
	MID_PHY_PARAM_WAKEUP_MS_MODE_ERR = -60, // 空中信号唤醒主从模式错误
	
	MID_MAC_SUCCESS = 0,
	MID_MAC_FAIL = -100,
	MID_MAC_BUSY = -101,
	MID_MAC_LEN_ERR = -102,
	MID_MAC_OFF_LINE = -103,
	MID_MAC_PARAM_ERR = -104,
	MID_MAC_NO_NETWORK = -105,
	
	MID_P2P_SUCCESS = 0,
	MID_P2P_FAIL = -150,
	MID_P2P_ERR_INIT = -151, // P2P 初始化失败
	MID_P2P_RF_BUSY = -152, // 射频非空闲状态
	} mid_ret_e; // 中间件接口返回值

表 4-14 中间件的返回值

4.1.2 tk86xx_poll

void tk86xx_poll(void);

状态机轮询函数，处理耗时操作，需要在主循环调用执行。

4.1.3 tk86xx_config_update

mid_ret_e tk86xx_config_update(mid_config_u *config, uint32_t option);

配置更新函数用于更新一个、多个或全部的配置项，option 为多个更新项标志位的“或”运算。例如更新 MAC 模块的速率及调频开关，option = MAC_DR | MAC_FREQ_HOP

参数	参数定义

config	输入参数，中间件的配置。
	typedef union
	{
	p2p_config_t p2p_config; // P2P 模块的配置，详见 4.1.1.1
	mac_config_t mac_config; // MAC 模块的配置，详见 4.1.1.2
	} mid_config_u;

option	输入参数，配置项的集合，是多个更新项标志位的“或”运算。例如更新 MAC

	模块的速率及调频开关，option = MAC_DR | MAC_FREQ_HOP P2P 模块的配置项标志位定义如下： #define P2P_DEV_ADDR 0x00000001 // P2P 模块的设备地址标志位 #define P2P_FILTER_FLAG 0x00000002 // P2P 模块的地址过滤标志位 #define P2P_WORK_MODE 0x00000004 // P2P 模块的工作模式标志位 #define P2P_RATE_MODE 0x00000008 // P2P 模块的速率模式标志位 #define P2P_BCN_MODE 0x00000010 // P2P 模块的 BCN 模式标志位 #define P2P_BCN_ID 0x00000020 // P2P 模块的 BCN 的 ID 标志位 #define P2P_BCN_FREQ 0x00000040 // P2P 模块的 BCN 频率标志位 #define P2P_TX_FREQ 0x00000080 // P2P 模块的发送频率标志位 #define P2P_RX_FREQ 0x00000100 // P2P 模块的接收频率标志位 #define P2P_TX_POWER 0x00000200 // P2P 模块的发射功率标志位 #define P2P_TX_LEN_MAX 0x00000400 // P2P 模块发送最长字节数标志位 #define P2P_WAKEUP_SRC 0x00000800 // P2P 模块的唤醒源标志位 #define P2P_WAKEUP_ID 0x00001000 // P2P 模块的唤醒 ID 标志位 #define P2P_WAKEUP_FREQ 0x00002000 // P2P 模块的唤醒信号载波频率标志位 #define P2P_WAKEUP_TIMING 0x00004000 //P2P 模块的唤醒时间间隔标志位 #define P2P_WAKEUP_LEVEL 0x00008000 //P2P 模块的唤醒电平条件标志位 #define P2P_WAKEUP_CAD_PERIOD 0x00010000 // P2P 模块的唤醒监听周期标志位 MAC 模块的配置项标志位定义如下： #define MAC_DR 0x00000001 // MAC 模块的速率标志位 #define MAC_CLASS_MODE 0x00000002 // MAC 模块的工作模式标志位 #define MAC_JOIN_MODE 0x00000004 // MAC 模块的入网模式标志位 #define MAC_RETRANS 0x00000008 // MAC 模块的重发次数标志位 #define MAC_DEV_EUI 0x00000010 // MAC 模块的设备 ID 标志位 #define MAC_DEV_ADDR 0x00000020 // MAC 模块的设备地址标志位 #define MAC_APP_KEY 0x00000040 // MAC 模块的 AppKey 标志位 #define MAC_GW_CHAN_NUM 0x00000080 // MAC 模块的搜索网关的信道个数标志位

	#define MAC_GW_CHAN_LIST 0x00000100 // MAC 模块的搜索网关的信道列表标志位 #define MAC_RELAY_CHAN_NUM 0x00000200 // MAC 模块的搜索中继的信道个数标志位 #define MAC_RELAY_CHAN_LIST 0x00000400 // MAC模块的搜索中继的信道列表标志位 #define MAC_FREQ_OFFSET 0x00000800 // MAC 模块的频偏标志位 #define MAC_FREQ_HOP 0x00001000 // MAC 模块的调频开关标志位 #define MAC_TX_POWER 0x00002000 // MAC 模块的发射功率标志位 #define MAC_POWER_ADR 0x00004000 // MAC 模块的功控开关标志位 #define MAC_BCN_ID 0x00008000 // MAC 模块的 BCN 的 ID 标志位 #define MAC_SLOT_MODE 0x00010000 // MAC 模块的时隙模式标志位 #define MAC_LAYER 0x00020000 // MAC 模块的层级标志位 #define MAC_WAKEUP_CFG 0x00040000 // MAC 模块的唤醒配置位 #define MAC_AUTOSEND_CFG 0X00080000 // MAC 模块自动发送控制位 #define MAC_DBG_CFG 0X00100000 // MAC 模块调试指令配置控制位 #define MID_ALL_OPTION 0xFFFFFFFF // 所有配置标志位
mid_ret_e	中间件返回值，详见 4.1.1.5

表 4-15 配置更新函数

4.1.4 tk86xx_config_get

mid_ret_e tk86xx_config_get(mid_mode_e *mode, mid_config_u *config, mid_info_u *info);

配置查询函数用于查询模块类型及模块配置。

参数	参数定义
mode	输出参数，中间件的模式。 typedef enum { P2P_MODE, // P2P 模块 MAC_MODE, // MAC 模块 } mid_mode_e;
config	输出参数，中间件的配置。 typedef union {

	p2p_config_t p2p_config; // P2P 模块的配置，详见 4.1.1.1
	mac_config_t mac_config; // MAC 模块的配置，详见 4.1.1.2
	} mid_config_u;

info	输出参数，中间件的只读信息。
	typedef union
	{
	p2p_info_t p2p_info; // P2P 模块的只读信息
	mac_info_t mac_info; // MAC 模块的只读信息
	} mid_info_u;

mid_ret_e	中间件返回值，详见 4.1.1.5

表 4-16 配置查询函数

4.1.4.1 p2p_info_t

P2P 模块只读信息。

结构体	结构体定义

p2p_info_t	typedef struct
	{
	uint8_t version[32]; // 版本号
	} p2p_info_t;

表 4-17 P2P 模块只读信息

4.1.4.2 mac_info_t

MAC 模块只读信息。

结构体	结构体定义

mac_info_t	typedef struct
	{
	nwk_status_e nwk_status; // 网络状态。0-没有网络；1-有网络但未激活；2-已激活。
	uint8_t band_plan[8]; // 带内频率分配信息
	uint8_t version[32]; // 版本号
	} mac_info_t;

nwk_status_e	typedef enum
	{
	MAC_WIRELESS_NONE = 0, // 没有网络
	MAC_WIRELESS_ON, // 有网络但未激活

	MAC_JOIN_ON, // 已激活 } nwk_status_e;

表 4-18 MAC 模块只读信息

4.1.5 tk86xx_data_transmit

mid_ret_e tk86xx_data_transmit(mid_message_u *message);

数据发送函数。

参数	参数定义

message	输入参数，中间件消息。
	typedef union
	{
	p2p_message_t p2p_message; // P2P 模块的消息
	mac_message_t mac_message; // MAC 模块的消息
	} mid_message_u;

mid_ret_e	中间件返回值，详见 4.1.1.5

表 4-19 数据发送

4.1.5.1 p2p_message_t

P2P 模块消息。

结构体	结构体定义

p2p_message_t	typedef struct
	{
	uint8_t *data; // 数据
	uint32_t len; // 数据的长度
	uint32_t dest_addr; // 目的地址，用于突发模式
	} p2p_message_t;

表 4-20 P2P 模块消息

4.1.5.2 mac_message_t

MAC 模块消息。

结构体	结构体定义

mac_message_t	typedef struct
	{
	uint8_t *data; // 数据
	uint32_t len; // 数据的长度
	uint8_t port; // 协议号，用于业务区分。取值范围：1~251

	message_mode_e message_mode; // 消息模式。0-不带确认的消息；1-带确认的消息
	} mac_message_t;

message_mode_e	typedef enum
	{
	MAC_MESSAGE_UNCONFIRM = 0, // 不带确认的消息模式
	MAC_MESSAGE_CONFIRM, // 带确认的消息模式
	} message_mode_e;

表 4-21 MAC 模块消息

4.1.6 tk86xx_control

mid_ret_e tk86xx_control(uint32_t cmd, void *arg_in, void *arg_out);

功能控制函数用于功能控制，cmd 为功能项的标志位,arg_in 为功能项的输入参数，arg_out 为功能项的输出参数。

参数	参数定义
cmd	输入参数，功能项的标志位。 P2P 模块的功能项标志位定义如下： #define P2P_SLEEP 1001 // P2P 模块的休眠控制标志位 #define P2P_LISTEN 1002 // P2P 模块的信道监听标志位 #define P2P_SINGLE_TONE 1003 // P2P 模块的单 TONE 测试标志位 MAC 模块的配置项标志位定义如下： #define MAC_JOIN 2001 // MAC 模块的入网控制标志位 #define MAC_LISTEN 2002 // MAC 模块的信道监听标志位 #define MAC_SINGLE_TONE 2003 // MAC 模块的单 TONE 测试标志位 #define MAC_SENSITIVITY 2004 // MAC 模块的灵敏度测试标志位 #define MAC_NET_SEARCH 2005 // MAC 模块的网络搜索控制标志位 #define MAC_ENTER_SLEEP 2006 // MAC 模块进入休眠控制标志位 #define MAC_PARAM_RESET 2007 // MAC 模块的参数复位标志位 #define MAC_RF_CAL 2008 // MAC 模块的射频校准标志位

arg_in	输入参数，功能项的输入参数。
	P2P 模块的功能项参数定义如下：
	P2P_SLEEP 时，arg_in 为 NULL
	P2P_LISTEN 时，arg_in 为 uint32_t *listen_freq; // 监听频率
	P2P_SINGLE_TONE 时，arg_in 为 uint32_t *single_tone_freq

	MAC 模块的功能项参数定义如下：
	MAC_JOIN 时，arg_in 为 NULL
	MAC_LISTEN 时，arg_in 为 uint32_t *listen_freq; // 监听频率
	MAC_SINGLE_TONE 时，arg_in 为 uint32_t *single_tone_freq
	MAC_SENSITIVITY 时，arg_in 为 NULL
	MAC_NET_SEARCH 时，arg_in 为 NULL
	MAC_ENTER_SLEEP 时，arg_in 为 NULL
	MAC_PARAM_RESET 时，arg_in 为 NULL
	MAC_RF_CAL 时，arg_in 为 uint32_t *cal_freq; // 校准频率

arg_out	输出参数，功能项的输出参数。
	输入参数，功能项的输入参数。
	P2P 模块的功能项参数定义如下：
	P2P_SLEEP 时，arg_out 为 NULL
	P2P_LISTEN 时，arg_out 为 int16_t *listen_rssi; // 监听结果
	P2P_SINGLE_TONE 时，arg_out 为 NULL
	MAC 模块的功能项参数定义如下：
	MAC_JOIN 时，arg_out 为 NULL
	MAC_LISTEN 时，arg_out 为 int16_t *listen_rssi; // 监听结果
	MAC_SINGLE_TONE 时，arg_out 为 NULL
	MAC_SENSITIVITY 时，arg_out 为 NULL
	MAC_NET_SEARCH 时，arg_out 为 NULL
	MAC_ENTER_SLEEP 时，arg_out 为 NULL
	MAC_PARAM_RESET 时，arg_out 为 NULL
	MAC_RF_CAL 时，arg_out 为 NULL

mid_ret_e	中间件返回值，详见 4.1.1.5

表 4-22 功能控制函数

4.2 外设接口定义

4.2.1 RAM 和 FLASH

在 MSU1 中，0x00080000 到 0x00088000 的 32K 空间为数据空间（DRAM），从 0x20000000 到 0x2000ffff 的 64K 空间为程序空间（IRAM），即最大代码容量 64K bytes。

从 0xc1000000 到 0xc30f0fff 为寄存器空间。

RAM 空间分配如下：

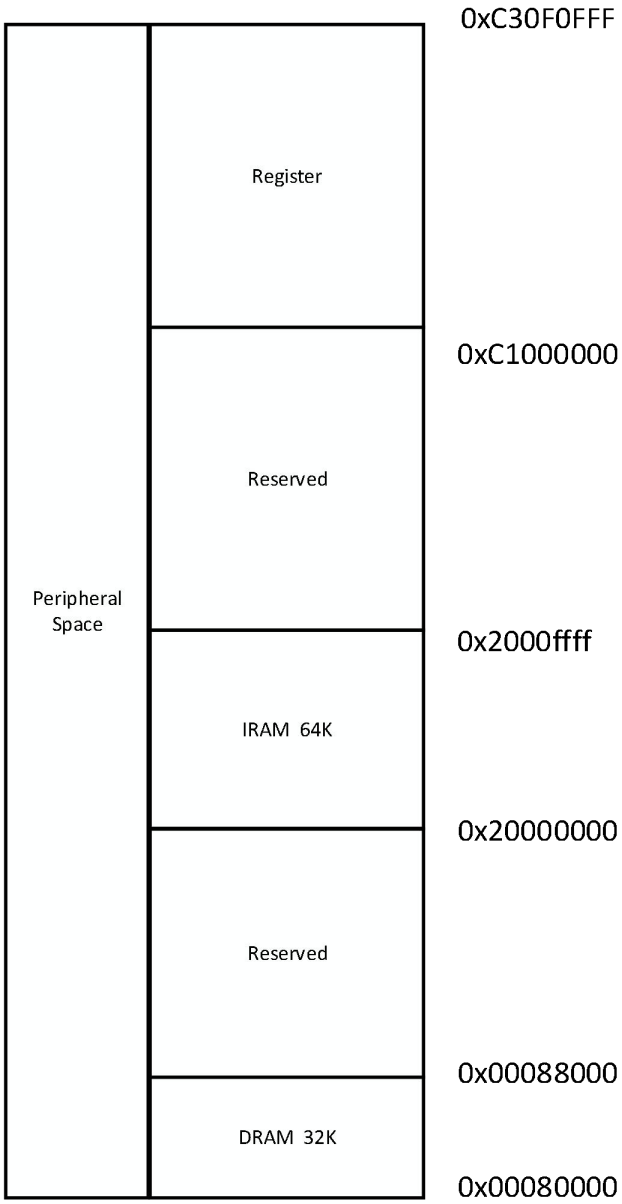

图 4-1 RAM 空间分配

flash 空间从 0x6C000 到 0x78000 空间大小为 48Kbyte，这段空间对用户开放，用户可以存储信息。

4.2.2 Flash 接口定义

用户输入的地址是用户使用空间的偏移地址，偏移地址范围为 0~0xC000。

4.2.2.1 flash_user_sector_erase

int flash_user_sector_erase(uint32_t sector_addr);

flash 扇区擦除，每次只能擦除 1 个扇区（4Kbyte）。

参数	参数定义
sector_addr	输入参数，扇区起始地址

返回值	0 -- 成功； -1 -- 失败

表 4-23 flash 扇区擦除

4.2.2.2 flash_user_read

int flash_user_read(uint32_t addr, uint8_t *buf, uint32_t len);

flash 上读数据。

参数	参数定义
addr	输入参数，起始地址
buf	输出参数，存储读取数据的缓存
len	输入参数，读取数据的长度
返回值	读取数据的长度

表 4-24 flash 上读取数据

4.2.2.3 flash_user_write

int flash_user_write(uint32_t addr, uint8_t *buf, uint32_t len);

flash 上写数据。

参数	参数定义
addr	输入参数，起始地址
buf	输入参数，需要写入的数据
len	输入参数，写入数据的长度
返回值	写入数据的长度

表 4-25 flash 上写数据

4.2.2.4 FLASH 接口使用示例

```
// 向 0x6C000 写入 4 字节数据
uint8_t data[] = {1,2,3,4};
flash_user_sector_erase(0);                    // 先擦除
flash_user_write(0, data, 4);                  // 写入数据

// 从 0x6C000 读出 4 字节数据
uint8_t data[4];
flash_user_read (0, data, 4);                  // 读出数据
```

4.2.3 时钟模块

4.2.3.1 时钟源

TK8610 可以外接一个 32MHz 晶体或者输入一个有源时钟信号，芯片内部具有 2 个 PLL 锁相环，分别用于 MSU0 和 MSU1 子系统。除此之外，系统内部还有低功耗 32KHz OSC 和 RC（精度+-20%），可用于芯片的电源管理和休眠唤醒。

内部 RC32K 和 OSC32K 时钟可用于开关机时序管理和低功耗电源管理，可通过寄存器配置选择哪个时钟作为低功耗电源管理模块的主时钟。

时钟源:

1．外部数字 IO 输入 32MHz;

2．PLL 两个，分别给 MSU0 和 MSU1 用;

3．RC32K 和 OSC32K。

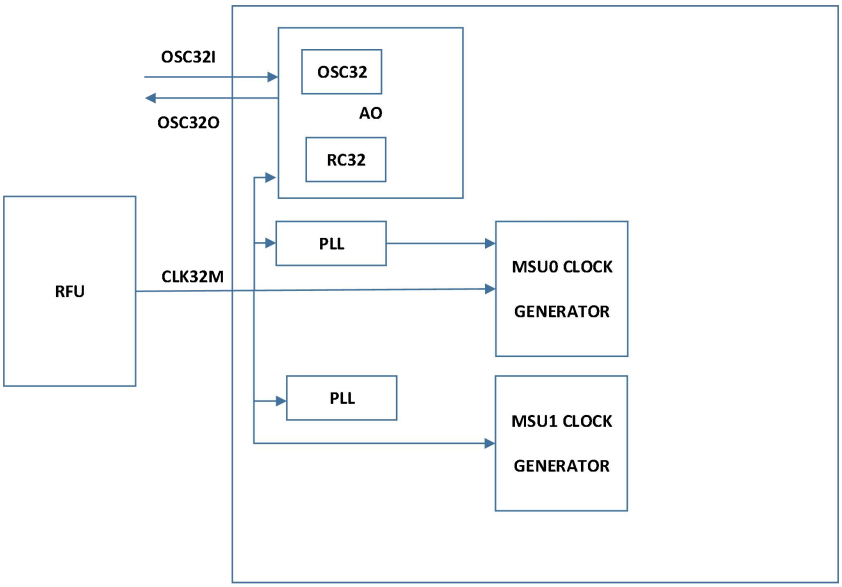

图 4-2 时钟源

4.2.3.2 MSU1 时钟树

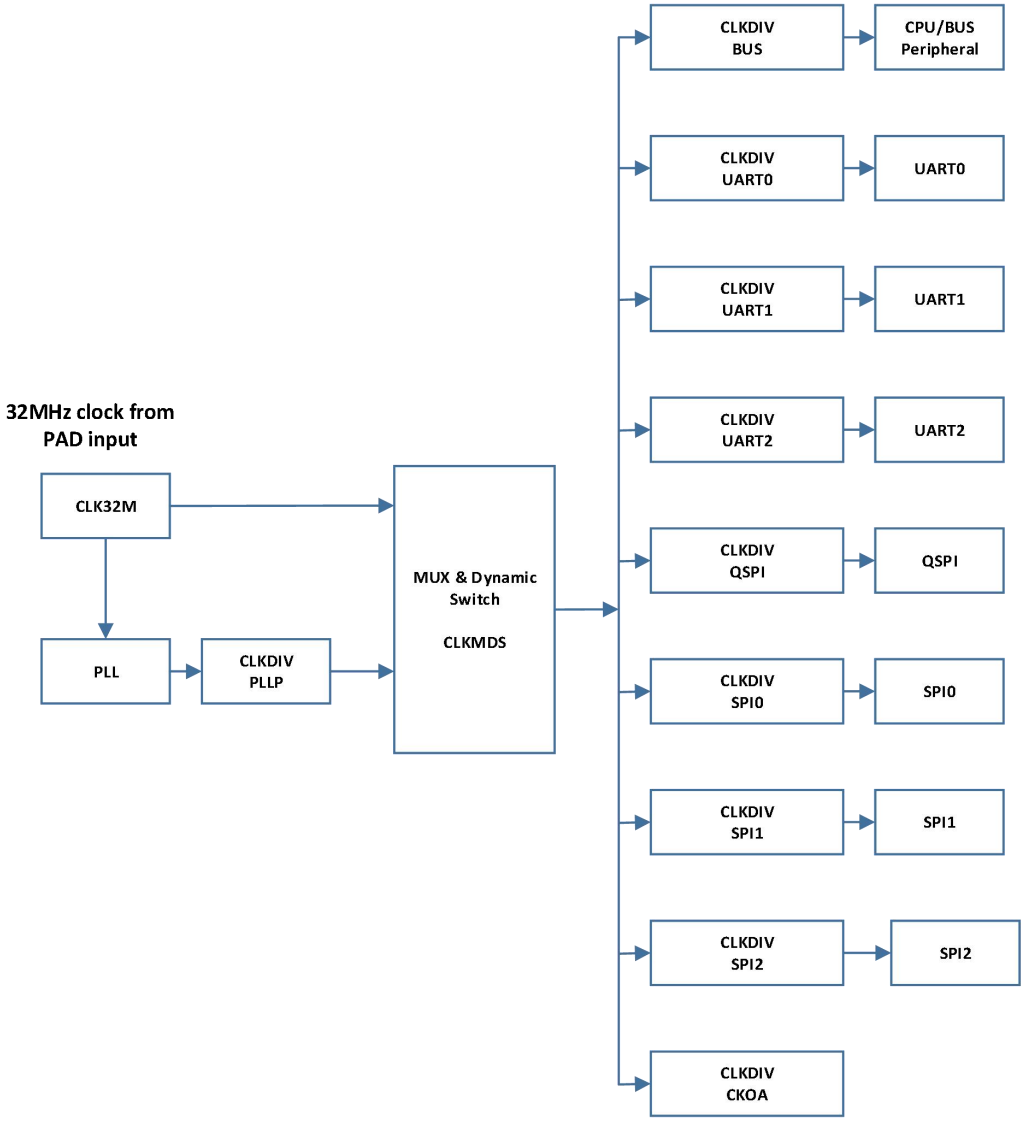

图 4-3 MSU1 时钟树结构图

4.2.4 GPIO 模块

TK8610 共有两组 IO 可用，GPIO_A 和 GPIO_B，每组 8 个，总共 16 个 GPIO。上电、硬复位或者休眠醒来后，GPIO_A0- GPIO_A7 默认是输出高电平，GPIO_B0- GPIO_B7 默认输入上拉。

4.2.4.1 GPIO 定义

GPIO 定义及复用功能如下表，其中仅有 GPIO_B0 ~ GPIO_B7 可作为外部中断源：

Pin Name	Pin Type	Alternate functions1	Alternate functions2	Alternate functions3	Alternate functions 4
GPIO_A0	I/O	UART0_TXD	SPI0_CS	PWM0	

GPIO_A1	I/O	UART0_RXD	SPI0_CLK	PWM1	
GPIO_A2	I/O		SPI0_TXD	PWM2	
GPIO_A3	I/O		SPI0_RXD	PWM3/	
GPIO_A4	I/O	I2C_SDA1		PWM0	
GPIO_A5	I/O	I2C_SCL1		PWM1	
GPIO_A6	I/O	UART1_RXD		PWM2	
GPIO_A7	I/O	UART1_TXD		PWM3	
GPIO_B0	I/O	I2C_SDA0	SPI1_CS	PWM0	EXTINT0
GPIO_B1	I/O	I2C_SCL0	SPI1_CLK	PWM1	EXTINT1
GPIO_B2	I/O		SPI1_TXD	PWM2	EXTINT2
GPIO_B3	I/O		SPI1_RXD	PWM3	EXTINT3
GPIO_B4	I/O	UART0_TXD			EXTINT4
GPIO_B5	I/O	UART0_RXD			EXTINT5
GPIO_B6	I/O				EXTINT6
GPIO_B7	I/O				EXTINT7

表 4-26 GPIO 定义

4.2.4.2 GPIO 接口定义

通用结构体	通用结构体定义
GPIO 索引号	enum gpio_pin { GPIO_PIN_A0 = 0, /*!< PA0 索引为 0 */ GPIO_PIN_A1, /*!< PA1 索引为 1 */ GPIO_PIN_A2, /*!< PA2 索引为 2 */ GPIO_PIN_A3, /*!< PA3 索引为 3 */ GPIO_PIN_A4, /*!< PA4 索引为 4 */ GPIO_PIN_A5, /*!< PA5 索引为 5 */ GPIO_PIN_A6, /*!< PA6 索引为 6 */ GPIO_PIN_A7, /*!< PA7 索引为 7 */ GPIO_PIN_B0, /*!< PB0 索引为 8 */ GPIO_PIN_B1, /*!< PB1 索引为 9 */ GPIO_PIN_B2, /*!< PB2 索引为 10 */ GPIO_PIN_B3, /*!< PB3 索引为 11 */ GPIO_PIN_B4, /*!< PB4 索引为 12 */ GPIO_PIN_B5, /*!< PB5 索引为 13 */ GPIO_PIN_B6, /*!< PB6 索引为 14 */ GPIO_PIN_B7, /*!< PB7 索引为 15 */ GPIO_PIN_MAX /*!< GPIO 索引最大值 */

	};

GPIO 功能号	enum
	{
	IO_GPIO = (0),
	IO_UART0_TXD = (1),
	IO_UART0_RXD = (2),
	IO_UART1_TXD = (3),
	IO_UART1_RXD = (4),
	IO_I2C0_SCL = (5),
	IO_I2C0_SDA = (6),
	IO_I2C1_SCL = (7),
	IO_I2C1_SDA = (8),
	IO_SPI0_CLK = (9),
	IO_SPI0_RXD = (10),
	IO_SPI0_TXD = (11),
	IO_SPI0_CS = (12),
	IO_SPI1_CLK = (13),
	IO_SPI1_RXD = (14),
	IO_SPI1_TXD = (15),
	IO_SPI1_CS = (16),
	IO_PWM0 = (17),
	IO_PWM1 = (18),
	IO_PWM2 = (19),
	IO_PWM3 = (20),
	IO_A03_UART0_SIRIN = (21),
	IO_A47_UART1_CTSIN1 = (21),
	IO_B03_UART0_SIROUT = (21),
	IO_B47_UART1_RTSN1 = (21),
	IO_UART0_TXD_MSU0 = (22),
	IO_UART0_RXD_MSU0 = (23),
	IO_A_CLK_OUT0 = (24),
	IO_B_EXTINT = (24),
	};

表 4-27 GPIO 接口定义

4.2.4.2.1 sysctrl_gpiomux_set

void sysctrl_gpiomux_set(uint8_t gpio, uint8_t func);

GPIO 引脚功能设置。

参数	参数定义
gpio	输入参数，GPIO 索引号，取值见 4.2.4.2
func	输入参数，GPIO 功能号，取值见 4.2.4.2

表 4-28 引脚功能设置

4.2.4.2.2 gpio_init

void gpio_init(enum gpio_pin gpio);

GPIO 引脚初始化

参数	参数定义
gpio	输入参数，GPIO 索引号，取值见 4.2.4.2

表 4-29 GPIO 引脚初始化

4.2.4.2.3 gpio_dir_get

GPIO_PinDirState gpio_dir_get(enum gpio_pin gpio);

GPIO 引脚方向查询。

参数	参数定义
gpio	输入参数，GPIO 索引号，取值见 4.2.4.2

返回值	typedef enum
	{
	GPIO_PIN_IN = 0, // 输入方向
	GPIO_PIN_OUT // 输出方向
	} GPIO_PinDirState;

表 4-30 GPIO 引脚方向查询

4.2.4.2.4 gpio_dir_set

void gpio_dir_set(enum gpio_pin gpio, GPIO_PinDirState dir);

GPIO 引脚方向设置。

参数	参数定义
gpio	输入参数，GPIO 索引号，取值见 4.2.4.2
dir	输入参数，GPIO 的方向 typedef enum {

	GPIO_PIN_IN = 0, // 输入方向
	GPIO_PIN_OUT // 输出方向
	} GPIO_PinDirState;

表 4-31 GPIO 引脚方向设置

4.2.4.2.5 gpio_pin_read

GPIO_PinState gpio_pin_read(enum gpio_pin gpio);

GPIO 引脚输出电平查询。

参数	参数定义
gpio	输入参数，GPIO 索引号，取值见 4.2.4.2

返回值	typedef enum
	{
	GPIO_PIN_RESET = 0, // 输出低电平
	GPIO_PIN_SET // 输出高电平
	} GPIO_PinState;

表 4-32 GPIO 引脚方向电平查询

4.2.4.2.6 gpio_pin_write

void gpio_pin_write(enum gpio_pin gpio, GPIO_PinState state);

GPIO 引脚输出电平设置。

参数	参数定义
gpio	输入参数，GPIO 索引号，取值见 4.2.4.2

state	输入参数，GPIO 输出电平的状态
	typedef enum
	{
	GPIO_PIN_RESET = 0, // 输出低电平
	GPIO_PIN_SET // 输出高电平
	} GPIO_PinState;

表 4-33 GPIO 引脚输出电平设置

4.2.4.2.7 gpio_pulldirection_set

void gpio_pulldirection_set(enum gpio_pin gpio, GPIO_PullDir status);

GPIO 引脚模式设置。

参数	参数定义
gpio	输入参数，GPIO 索引号，取值见 4.2.4.2

status	输入参数，GPIO 引脚模式
	typedef enum
	{
	GPIO_PIN_PULLUP = 0, // 上拉模式
	GPIO_PIN_PULLDOWN // 下拉模式
	} GPIO_PullDir;

表 4-34 引脚模式设置

4.2.4.2.8 GPIO 接口使用示例

```
//将 GPIO_B3 配置为输出状态，并输出高电平。
sysctrl_gpiomux_set(GPIO_B3, IO_GPIO);
gpio_init(GPIO_B3);
gpio_dir_set(GPIO_B3, GPIO_PIN_OUT);
gpio_pin_write(GPIO_B3, GPIO_PIN_SET);

//将 GPIO_A6、GPIO_A7 复用为 UART1。
sysctrl_gpiomux_set(GPIO_A6, IO_UART1_RXD);
sysctrl_gpiomux_set(GPIO_A7, IO_UART1_TXD);
```

4.2.5 定时器接口定义

TK8610 具有 4 个通用 32 位定时器，分别为 TIMER0、TIMER1、TIMER2、TIMER3。定时器向下计数，并支持对输入时钟的 1/16/256 分频。

4.2.5.1 timer_init

void timer_init(TIMER_TypeDef *TIMERx, uint32_t count_us, uint8_t counter_mode);

定时器初始化。

参数	参数定义
TIMERx	输入参数，定时器 #define TIMER0 ((TIMER_TypeDef *) TIMER0_BASE) #define TIMER1 ((TIMER_TypeDef *) TIMER1_BASE) #define TIMER2 ((TIMER_TypeDef *) TIMER2_BASE) #define TIMER3 ((TIMER_TypeDef *) TIMER3_BASE)
count_us	输入参数，计时周期，单位微秒
counter_mode	输入参数，计数模式 enum {

	CNTMODE_WRAPPING, // 周期计数 CNTMODE_ONESHOT, // 单次计数 };

表 4-35 定时器初始化

4.2.5.2 timer_register_callback

void timer_register_callback(TIMER_TypeDef *TIMERx, timer_callback_t callback);

注册定时器回调函数。

参数	参数定义
TIMERx	输入参数，定时器 #define TIMER0 ((TIMER_TypeDef *) TIMER0_BASE) #define TIMER1 ((TIMER_TypeDef *) TIMER1_BASE) #define TIMER2 ((TIMER_TypeDef *) TIMER2_BASE) #define TIMER3 ((TIMER_TypeDef *) TIMER3_BASE)
callback	typedef void (*timer_callback_t)(void);

表 4-36 注册定时器回调函数

4.2.5.3 timer_enable

void timer_enable(TIMER_TypeDef *TIMERx);

设置定时器使能。

参数	参数定义
TIMERx	输入参数，定时器 #define TIMER0 ((TIMER_TypeDef *) TIMER0_BASE) #define TIMER1 ((TIMER_TypeDef *) TIMER1_BASE) #define TIMER2 ((TIMER_TypeDef *) TIMER2_BASE) #define TIMER3 ((TIMER_TypeDef *) TIMER3_BASE)

表 4-37 定时器使能设置

4.2.5.4 timer_disable

void timer_disable(TIMER_TypeDef *TIMERx);

设置定时器失效。

参数	参数定义
TIMERx	输入参数，定时器 #define TIMER0 ((TIMER_TypeDef *) TIMER0_BASE) #define TIMER1 ((TIMER_TypeDef *) TIMER1_BASE) #define TIMER2 ((TIMER_TypeDef *) TIMER2_BASE)

	#define TIMER3 ((TIMER_TypeDef *) TIMER3_BASE)

表 4-38 定时器失效设置

4.2.5.5 定时器接口使用示例

```
//配置定时器 TIMER1
timer_init(TIMER1, 1000000, CNTMODE_WRAPPING);
timer_register_callback(TIMER1, tim_callback);
timer_enable(TIMER1);
//开启定时器中断
eclic_irq_enable(TIMER1_IRQn, 0, 0);
```

4.2.6 I2C 接口定义

TK8610 有 2 个 I2C 模块，分别为 I2C0、I2C1。

4.2.6.1 i2c_init

void i2c_init(I2C_TypeDef *i2c, uint32_t bps);

I2C 总线初始化。

参数	参数定义
i2c	输入参数，I2C 模块号 #define I2C0 ((I2C_TypeDef *) I2C0_BASE) #define I2C1 ((I2C_TypeDef *) I2C1_BASE)
bps	输入参数，波特率

表 4-39 I2C 总线初始化

4.2.6.2 i2c_deinit

void i2c_deinit(I2C_TypeDef *i2c);

I2C 总线复位。

参数	参数定义
i2c	输入参数，I2C 模块号 #define I2C0 ((I2C_TypeDef *) I2C0_BASE) #define I2C1 ((I2C_TypeDef *) I2C1_BASE)

表 4-40 I2C 总线复位

4.2.6.3 i2c_start

ErrorStatus i2c_start(I2C_TypeDef *i2c, uint8_t device, uint8_t rw);

I2C 总线上产生起始信号。

参数	参数定义

i2c	输入参数，I2C 模块号 #define I2C0 ((I2C_TypeDef *) I2C0_BASE) #define I2C1 ((I2C_TypeDef *) I2C1_BASE)
device	输入参数，从机设备地址 #define BH1750_ADDR (0x23<<1)
rw	输入参数，数据的类型 #define I2C_WRITE (0x00) /*!< I2C 写控制 */ #define I2C_READ (0x01) /*!< I2C 读控制 */

表 4-41 I2C 总线产生起始信号

4.2.6.4 i2c_stop

ErrorStatus i2c_stop(I2C_TypeDef *i2c);

I2C 总线上产生停止信号。

参数	参数定义
i2c	输入参数，I2C 模块号 #define I2C0 ((I2C_TypeDef *) I2C0_BASE) #define I2C1 ((I2C_TypeDef *) I2C1_BASE)

表 4-42 I2C 总线产生停止信号

4.2.6.5 i2c_read

ErrorStatus i2c_read(I2C_TypeDef *i2c, uint8_t *buf, uint32_t size);

I2C 总线上读数据。

参数	参数定义
i2c	输入参数，I2C 模块号 #define I2C0 ((I2C_TypeDef *) I2C0_BASE) #define I2C1 ((I2C_TypeDef *) I2C1_BASE)
buf	输出参数，读取的数据存储空间
size	输入参数，读取的数据长度

表 4-43 I2C 总线上读数据

4.2.6.6 i2c_write_onebyte

ErrorStatus i2c_write_onebyte(I2C_TypeDef *i2c, uint8_t byte);

I2C 总线上写数据。

参数	参数定义
i2c	输入参数，I2C 模块号

	#define I2C0 ((I2C_TypeDef *) I2C0_BASE) #define I2C1 ((I2C_TypeDef *) I2C1_BASE)
byte	输入参数，byte 为数据内容

表 4-44 I2C 总线上写数据

4.2.6.7 I2C 接口使用示例

```
//I2C 启动
if (SUCCESS != i2c_start(I2C0, BH1750_ADDR, I2C_WRITE))
    return -1;
//写入一个字节数据
if (SUCCESS != i2c_write_onebyte(I2C0, cmd))
    return -1;
//I2C 停止
if (SUCCESS != i2c_stop(I2C0))
    return -1;
```

4.2.7 UART 接口定义

TK8610 有 2 个 UART 模块，分别为 UART0、UART1。UART1 用于固件升级；故只有 UART0 可用。

4.2.7.1 uart_init

int32_t uart_init(void *uartx, uart_config_t config);

UART 总线初始化。

参数	参数定义
uartx	输入参数，UART 模块号 #define UART0 ((UART_TypeDef *) UART0_BASE) #define UART1 ((UART1_TypeDef *) UART1_BASE) #define UART2 ((UART_TypeDef *) UART2_BASE)
config	输入参数，UART 模块配置 typedef struct { uint32_t baud_rate; /*!< 波特率 */ uint32_t word_length; /*!< 数据位 */ uint32_t stop_bits; /*!< 停止位 */ uint32_t parity; /*!< 奇偶校验位 */ uint32_t hardware_flow_control; /*!< 流控制 */ } uart_config_t;

返回值	-0 -- 成功; -1 -- UART 模块号错误 -2 -- 波特率配置失败 -3 -- 奇偶校验位配置失败 -4 -- 数据位配置失败 -5 -- 停止位配置失败

表 4-45 UART 总线初始化

4.2.7.2 uart_read

int32_t uart_read(void *uartx, uint8_t *buf, uint32_t size);

UART 总线上读取数据。

参数	参数定义
uartx	输入参数，UART 模块号 #define UART0 ((UART_TypeDef *) UART0_BASE) #define UART1 ((UART1_TypeDef *) UART1_BASE) #define UART2 ((UART_TypeDef *) UART2_BASE)
buf	输出参数，存储读取数据的缓存
size	输入参数，读取数据的长度
返回值	-0 -- 成功 -1 -- UART 模块号错误 -2 -- 缓存为空 -3 -- 超时

表 4-46 UART 总线上读取数据

4.2.7.3 uart_read_int

int32_t uart_read_int(void *uartx, uint8_t *buf, uint32_t len, uart_isr_callback callback);

UART 总线上中断方式读数据。

参数	参数定义
uartx	输入参数，UART 模块号 #define UART0 ((UART_TypeDef *) UART0_BASE) #define UART1 ((UART1_TypeDef *) UART1_BASE) #define UART2 ((UART_TypeDef *) UART2_BASE)
buf	输出参数，存储读取数据的缓存
size	输入参数，读取数据的长度
callback	typedef void (*uart_isr_callback)(uint8_t status);

返回值	-0 -- 成功; -1 -- UART 模块号错误; -2 -- 缓存为空

表 4-47 UART 总线上中断方式读数据

4.2.7.4 uart_write

int32_t uart_write(void *uartx, uint8_t *buf, uint32_t size);

UART 总线上写数据。

参数	参数定义
uartx	输入参数，UART 模块号 #define UART0 ((UART_TypeDef *) UART0_BASE) #define UART1 ((UART1_TypeDef *) UART1_BASE) #define UART2 ((UART_TypeDef *) UART2_BASE)
buf	输入参数，需要写入的数据
size	输入参数，写入的数据长度
返回值	0 -- 成功 -1 -- UART 模块号错误 -2 -- 缓存为空 -3 -- 超时

表 4-48 UART 总线上写数据

4.2.7.5 uart_write_int

int32_t uart_write_int(void *uartx, uint8_t *buf, uint32_t len, uart_isr_callback callback);

UART 总线上中断方式写数据。

参数	参数定义
uartx	输入参数，UART 模块号 #define UART0 ((UART_TypeDef *) UART0_BASE) #define UART1 ((UART1_TypeDef *) UART1_BASE) #define UART2 ((UART_TypeDef *) UART2_BASE)
buf	输入参数，需要写入的数据
size	输入参数，写入的数据长度
callback	typedef void (*uart_isr_callback)(uint8_t status);
返回值	0 -- 成功 -1 -- UART 模块号错误 -2 -- 缓存为空

表 4-49 UART 总线上中断方式写数据

4.2.7.6 UART 接口使用示例

```
//初始化 uart 参数结构体
uart_config_t config;
config.baud_rate     = BAUD_RATE_115200;
config.word_length = DATA_BITS_8;
config.stop_bits     = STOP_BITS_1;
config.parity          = PARITY_NONE;
//uart 外设初始化
uart_init(UART1, config);
//uart 发送字符串
uart_write(UART1, "hello TK8610\r\n", sizeof("hello TK8610\r\n") - 1);
//UART 中断接收 30 个字符串，完成后执行 uart1_isr_rx_callback 函数
uart_read_int(UART1, buf, 30, uart1_isr_rx_callback);

//打开 UART 中断
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL0_PRIO3);
eclic_irq_enable(UART1_IRQn, 0, 2);
eclic_global_interrupt_enable();
//UART 中断发送字符串，完成后执行 uart1_isr_tx_callback 函数
uart_write_int(UART1, "tx int send!\n", sizeof("tx int send!\n") - 1, uart1_isr_tx_callback);
```

4.2.8 SPI 接口定义

TK8610 芯片具有 2 个独立的 SPI 接口，分别为 SPI0、SPI1。

4.2.8.1 spi_init

void spi_init(SPI_TypeDef *spi, spi_param_t param);

SPI 总线初始化。

参数	参数定义
spi	输入参数，SPI 模块号 #define SPI0 ((SPI_TypeDef *) SPI0_BASE) #define SPI1 ((SPI_TypeDef *) SPI1_BASE)
param	输入参数，SPI 模块配置 typedef struct { uint32_t device_mode; /*!< 主机或设备模式配置 */

	uint32_t frame_size; /*!< 数据帧格式配置 */ uint32_t clock_polarity; /*!< 时钟极性配置 */ uint32_t clock_phase; /*!< 时钟相位配置 */ uint32_t prescale; /*!< 预分频器配置 */ } spi_param_t;

表 4-50 SPI 总线初始化

4.2.8.2 spi_read

void spi_read(SPI_TypeDef *SPIx, uint8_t *buf, uint32_t size);

SPI 总线上读取数据。

参数	参数定义
spi	输入参数，SPI 模块号 #define SPI0 ((SPI_TypeDef *) SPI0_BASE) #define SPI1 ((SPI_TypeDef *) SPI1_BASE)
buf	输出参数，存储读取数据的缓存
size	输入参数，读取数据的长度

表 4-51 SPI 总线上读取数据

4.2.8.3 spi_write

void spi_write(SPI_TypeDef *SPIx, uint8_t *buf, uint32_t size);

SPI 总线上写数据。

参数	参数定义
spi	输入参数，SPI 模块号 #define SPI0 ((SPI_TypeDef *) SPI0_BASE) #define SPI1 ((SPI_TypeDef *) SPI1_BASE)
buf	输入参数，需要写入的数据
size	输入参数，写入的数据长度

表 4-52 SPI 总线上写数据

4.2.8.4 spi_write_read_byte

uint8_t spi_write_read_byte(SPI_TypeDef *SPIx, uint8_t byte);

SPI 总线上读写数据。

参数	参数定义
spi	输入参数，SPI 模块号 #define SPI0 ((SPI_TypeDef *) SPI0_BASE)

	#define SPI1 ((SPI_TypeDef *) SPI1_BASE)
byte	输入参数，需要写入的数据
返回值	读取的数据

表 4-53 SPI 总线上读写数据

4.2.8.5 SPI 接口使用示例

```
//SPI flash 参数配置
spi_param_t spi_config;
spi_config.device_mode      = SPI_MODE_MASTER;
spi_config.frame_size       = SPI_DATASIZE_8BIT;
spi_config.clock_polarity   = SPI_POLARITY_HIGH;
spi_config.clock_phase      = SPI_PHASE_2EDGE;
spi_config.prescale         = 2;

//SPI 引脚配置
sysctrl_gpiomux_set(GPIO_A2, IO_SPI0_CLK);
sysctrl_gpiomux_set(GPIO_A3, IO_SPI0_TXD);
sysctrl_gpiomux_set(GPIO_A4, IO_SPI0_RXD);
sysctrl_gpiomux_set(GPIO_A5, IO_GPIO);
gpio_dir_set(GPIO_A5, GPIO_PIN_OUT);
gpio_pin_write(GPIO_A5, GPIO_PIN_SET);

spi_flash_select(SPI0, GPIO_A5);
//SPI0 初始化
spi_init(SPI0, spi_config);
```

4.2.9 PWM 接口定义

TK8610 芯片内置 4 个 PWM 接口，分别为 PWM0、PWM1、PWM2、PWM3。

4.2.9.1 pwm_init

void pwm_init(PWM_TypeDef *pwm, uint32_t frequency, uint32_t high_duty);

PWM 初始化。

参数	参数定义
pwm	输入参数，PWM 模块号 #define PWM0 ((PWM_TypeDef *) PWM0_BASE) #define PWM1 ((PWM_TypeDef *) PWM1_BASE)

	#define PWM2 ((PWM_TypeDef *) PWM2_BASE) #define PWM3 ((PWM_TypeDef *) PWM3_BASE)
frequency	输入参数，频率（Hz）
high_duty	输入参数，占空比，取值范围：0~99

表 4-54 PWM 初始化

4.2.9.2 pwm_update

void pwm_update(PWM_TypeDef *pwm, uint32_t frequency, uint32_t high_duty);

PWM 配置更新。

参数	参数定义
pwm	输入参数，PWM 模块号 #define PWM0 ((PWM_TypeDef *) PWM0_BASE) #define PWM1 ((PWM_TypeDef *) PWM1_BASE) #define PWM2 ((PWM_TypeDef *) PWM2_BASE) #define PWM3 ((PWM_TypeDef *) PWM3_BASE)
frequency	输入参数，频率（Hz）
high_duty	输入参数，占空比，取值范围：0~99

表 4-55 PWM 配置更新

4.2.9.3 pwm_enable

void pwm_enable(PWM_TypeDef *pwm);

设置 PWM 使能。

参数	参数定义
pwm	输入参数，PWM 模块号 #define PWM0 ((PWM_TypeDef *) PWM0_BASE) #define PWM1 ((PWM_TypeDef *) PWM1_BASE) #define PWM2 ((PWM_TypeDef *) PWM2_BASE) #define PWM3 ((PWM_TypeDef *) PWM3_BASE)

4.2.9.4 pwm_disable

void pwm_disable(PWM_TypeDef *pwm);

设置 PWM 失效。

参数	参数定义
pwm	输入参数，PWM 模块号

	#define PWM0	((PWM_TypeDef *) PWM0_BASE)
	#define PWM1	((PWM_TypeDef *) PWM1_BASE)
	#define PWM2	((PWM_TypeDef *) PWM2_BASE)
	#define PWM3	((PWM_TypeDef *) PWM3_BASE)

4.2.9.5 PWM 接口使用示例

```
//PWM 配置
pwm_init(PWM0, 1000, 0);
//PWM 使能启动
pwm_enable(PWM0);
```

4.2.10 RTC 接口定义

4.2.10.1 rtc_enable

void rtc_enable(void);

设置 RTC 使能。

4.2.10.2 rtc_disable

void rtc_disable(void);

设置 RTC 失效。

4.2.10.3 rtc_current_value_get

uint32_t rtc_current_value_get(void);

RTC 计数器查询。

参数	参数定义
返回值	计数器的值

表 4-56 RTC 计数器查询

4.2.10.4 rtc_interrupt_enable

void rtc_interrupt_enable(void);

设置中断使能。

4.2.10.5 rtc_interrupt_disable

void rtc_interrupt_disable(void);

设置中断失效。

4.2.10.6 rtc_alarm_set

void rtc_alarm_set(uint32_t alarm, rtc_callback_t callback);

设置 RTC 定时。

参数	参数定义

alarm	输入参数，定时值，单位 1/32768 s
callback	typedef void (*rtc_callback_t)(void);

表 4-57 设置 RTC 定时

4.2.10.7 rtc_interrupt_flag_clear

void rtc_interrupt_flag_clear(void);

RTC 中断标志位清空。

4.2.10.8 RTC 接口使用示例

```
//RTC 外设使能
rtc_enable();
tk_printf("current value: 0x%08x\r\n", rtc_current_value_get());
tk_delay_ms(1000);
tk_printf("current value: 0x%08x\r\n", rtc_current_value_get());
//设置 1s 后的闹钟，1S 后执行 rt_callback 函数
rtc_alarm_set(sysctrl_clk_calculate(RTC), rtc_callback);
rtc_interrupt_enable();
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL0_PRIO3);
eclic_irq_enable(RTC_IRQn, 0, 3);
eclic_global_interrupt_enable();
```

4.2.11 WATCHDOG 接口定义

4.2.11.1 watchdog_config

void watchdog_config(uint32_t count_ms, enum wdt_mode reset_en);

看门狗配置。

参数	参数定义
count_ms	输入参数，看门狗超时时间，单位毫秒

reset_en	输入参数，中断处理模式。注意，休眠后看门狗不工作。
	enum wdt_mode {
	WDT_DISABLE_RESET_MODE = 0, //看门狗中断触发后，不重启模式
	WDT_ENABLE_RESET_MODE = 1，//看门狗中断触发后，重启模式
	};

4.2.11.2 watchdog_register_callback

void watchdog_register_callback(wdt_callback_t callback);

注册看门狗超时回调函数。

参数	参数定义
callback	typedef void (*wdt_callback_t)(void);

4.2.11.3 watchdog_enable

void watchdog_enable(void);

设置看门狗使能。

4.2.11.4 watchdog_disable

void watchdog_disable(void);

设置看门狗失效。

4.2.11.5 watchdog_feed

void watchdog_feed(void);

看门狗喂狗。

4.2.11.6 WATCHDOG 接口使用示例

```
sysctrl_reset_peripheral(WDT);
tk_delay_ms(1);
//设置看门狗 10s 超时，超时后不重启
watchdog_config(10000, WTD_DISABLE_RESET_MODE);
//使能看门狗
watchdog_enable();
```

5 代码目录结构

```
├─── common // 公共头文件
├─── lib // 库文件
├─── middleware // 中间件头文件
├─── peripheral // 外设头文件
├─── phy // 物理层头文件
├─── riscv // 启动头文件
└─── scripts // 空间占用计算文件
```