文件名称：ECR6600_API

文件编号：

版　　本：

共 66 页

拟　制 _____

审　核 _____

会　签 _____

_____

_____

_____

标准化 _____

批　准 _____

修改记录

文件编号	版本号	拟制人/ 修改人	拟制/修改日期	更改理由	主要更改内容 （写要点即可）
	V0.1		2021/1/13		新建
注 1：每次更改归档文件（指归档到事业部或公司档案室的文件）时，需填写此表。 注 2：文件第一次归档时，“更改理由”、“主要更改内容”栏写“无”。					

目 录

1 引言

1.1 编写目的

本文描述 TR6260 解决方案支持的 API，便于用户参考。

1.2 文档约定

无。

2 驱动 API

2.1 GPIO

2.1.1 gpio_set_dir

函数	int32_t gpio_set_dir(uint32_t pin_num,uint32_t pin_dir)
说明	设置 GPIO Pin 脚的工作模式为输入或输出
参数	pin_num：DRV_GPIO_PIN_NAME， GPIO Pin 脚 （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外） pin_dir ：DRV_GPIO_DIR，输入或输出 （DRV_GPIO_DIR_INPUT 或 DRV_GPIO_DIR_OUTPUT）
返回	DRV_ERR_INVALID_PARAM：设置失败，传入参数不合法 DRV_SUCCESS：设置成功
说明	

2.1.2 gpio_get_dir

函数	int32_t gpio_get_dir(uint32_t pin_num, uint32_t * pin_dir)
说明	获取某 GPIO Pin 脚的工作模式
参数	pin_num：DRV_GPIO_PIN_NAME， GPIO Pin 脚 （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外）

	pin_dir ： uint32_t *
返回	DRV_ERR_INVALID_PARAM：获取失败，传入参数不合法 DRV_SUCCESS：获取成功
说明	

2.1.3 gpio_get_write_value

函数	int32_t gpio_get_write_value(uint32_t pin_num, uint32_t * pin_value)
说明	获取 Pin 脚输出时电平值
参数	pin_num：DRV_GPIO_PIN_NAME， GPIO Pin 脚 （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外） pin_value ： uint32_t *
返回	DRV_ERR_INVALID_PARAM：获取失败，传入参数不合法 DRV_SUCCESS：获取成功
说明	

2.1.4 gpio_config

函数	int32_t gpio_config(DRV_GPIO_CONFIG *pGpioConfig)
说明	GPIO 配置
参数	pGpioConfig：DRV_GPIO_CONFIG
返回	DRV_ERR_INVALID_PARAM：配置失败，传入参数不合法 DRV_SUCCESS：配置成功
说明	

2.1.5 gpio_write

函数	int32_t gpio_write(uint32_t pin_num,uint32_t pin_value)
说明	写 GPIO Pin 脚为低电平或高电平
参数	pin_num：DRV_GPIO_PIN_NAME， GPIO Pin 脚 （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外） pin_value ： DRV_GPIO_LEVEL DRV_GPIO_LEVEL_LOW 或 DRV_GPIO_LEVEL_HIGH
返回	DRV_ERR_INVALID_PARAM：写失败，传入参数不合法 DRV_SUCCESS：写成功
说明	

2.1.6 gpio_read

函数	int32_t gpio_read(uint32_t pin_num,uint32_t *pin_value)
说明	获取 GPIO Pin 脚输入为低电平或高电平
参数	pin_num：DRV_GPIO_PIN_NAME， GPIO Pin 脚 （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外） pin_value ： DRV_GPIO_LEVEL DRV_GPIO_LEVEL_LOW 或 DRV_GPIO_LEVEL_HIGH
返回	DRV_ERR_INVALID_PARAM：写失败，传入参数不合法 DRV_SUCCESS：写成功
说明	

2.1.7 gpio_wakeup_config

函数	int32_t gpio_wakeup_config(uint32_t pin_num,uint32_t wakeup_en)

说明	使能/不使能 GPIO Pin 脚
参数	pin_num：DRV_GPIO_PIN_NAME， GPIO Pin 脚 （只支持 DRV_GPIO_13 和 DRV_GPIO_17） wakeup_en：DRV_GPIO_WAKEUP_ENABLE DRV_GPIO_WAKEUP_DIS 或 DRV_GPIO_WAKEUP_EN
返回	DRV_ERR_INVALID_PARAM：设置失败，传入参数不合法 DRV_SUCCESS：设置成功
说明	

2.1.8 gpio_power_area_config

函数	int32_t gpio_power_area_config(uint32_t pin_num,uint32_t pwr)
说明	设置 GPIO Pin 的供电区
参数	pin_num：DRV_GPIO_PIN_NAME， GPIO Pin 脚 （DRV_GPIO_0 到 DRV_GPIO_24 和 DRV_GPIO_MAX） pwr：DRV_GPIO_PWR DRV_GPIO_PWR_PD 或 DRV_GPIO_PWR_AO
返回	DRV_ERR_INVALID_PARAM：设置失败，传入参数不合法 DRV_SUCCESS：设置成功
说明	

2.1.9 gpio16_write

函数	int32_t gpio16_write(uint32_t pin_value)
说明	写 GPIO16 的值
参数	pin_value ： DRV_GPIO_LEVEL DRV_GPIO_LEVEL_LOW 或 DRV_GPIO_LEVEL_HIGH

返回	DRV_SUCCESS：设置成功
说明	

2.1.10 gpio17_write

函数	int32_t gpio17_write(uint32_t pin_value)
说明	写 GPIO17 的值
参数	pin_value ： DRV_GPIO_LEVEL DRV_GPIO_LEVEL_LOW 或 DRV_GPIO_LEVEL_HIGH
返回	DRV_SUCCESS：设置成功
说明	

2.1.11 gpio17_read_config

函数	int32_t gpio17_read_config(uint32_t pull_up_en)
说明	Gpio17 配置为输入引脚
参数	pull_up_en ： DRV_GPIO_PULL_UP_ENABLE DRV_GPIO_PULL_UP_EN 或 DRV_GPIO_PULL_UP_DIS
返回	DRV_ERR_INVALID_PARAM：设置失败，传入参数不合法 DRV_SUCCESS：设置成功
说明	

2.1.12 gpio17_read

函数	int32_t gpio17_read(uint32_t *pin_value)
说明	读取 GPIO17 输入的电平值
参数	pin_value ： uint32_t *
返回	DRV_SUCCESS：成功

说明	

2.1.13 gpio_irq_level

函数	int32_t gpio_irq_level(uint32_t pin_num, uint32_t mode)
说明	设置 GPIO 中断出发模式
参数	pin_num: DRV_GPIO_PIN_NAME （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外） mode: DRV_GPIO_INT_LEVEL
返回	DRV_ERR_INVALID_PARAM：失败，传入参数不合法 DRV_SUCCESS：成功
说明	

2.1.14 gpio_irq_callback_register

函数	int32_t gpio_irq_callback_register(uint32_t pin_num, void(* callback)(void * data), void * data)
说明	注册中断回调函数
参数	pin_num: DRV_GPIO_PIN_NAME （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外） void * data ： void(* callback) data：void *
返回	DRV_ERR_INVALID_PARAM：失败，传入参数不合法 DRV_SUCCESS：成功
说明	

2.1.15 gpio_irq_unmusk

函数	int32_t gpio_irq_unmusk(uint32_t pin_num)
说明	使能 pin 脚中断。
参数	pin_num: DRV_GPIO_PIN_NAME （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外）

返回	DRV_ERR_INVALID_PARAM：失败，传入参数不合法 DRV_SUCCESS：成功
说明	

2.1.16 gpio_irq_musk

函数	int32_t gpio_irq_musk(uint32_t pin_num)
说明	屏蔽 pin 脚中断。
参数	pin_num: DRV_GPIO_PIN_NAME （DRV_GPIO_0 到 DRV_GPIO_24，DRV_GPIO_16 和 DRV_GPIO_17 除外）
返回	DRV_ERR_INVALID_PARAM：失败，传入参数不合法 DRV_SUCCESS：成功
说明	

2.1.17 gpio_isr_init

函数	void gpio_isr_init(void)
说明	GPIO 初始化
参数	void
返回	void
说明	

2.2 I2C

2.2.1 i2c_driver_init

函数	int i2c_driver_init(i2c_config_t *i2c_config)
说明	I2C 驱动初始化

参数	i2c_config：i2c_config_t*
返回	ERROR_I2C_PARAM/1：失败 ERROR_I2C_MEM/3：失败 ERROR_I2C_FAILED/5：失败 I2C_OK/0：成功
说明	

2.2.2 i2c_driver_deinit

函数	int i2c_driver_deinit(void)
说明	I2C 复位
参数	void
返回	I2C_OK/0：成功
说明	

2.2.3 i2c_master_write

函数	int i2c_master_write(uint16_t slave_addr, uint8_t* data, size_t data_len, bool dma_mode, TickType_t ticks_to_wait)
说明	向主机写数据
参数	slave_addr：uint16_t，外设地址 data：uint8_t*，写入的数据 data_len：size_t，数据长度 dma_mode：bool，是否为 DMA 模式 ticks_to_wait：TickType_t
返回	ERROR_I2C_PARAM：失败 ERROR_I2C_BUSY：失败 ERROR_I2C_TIMEOUT：失败 I2C_OK：成功

说明	

2.2.4 i2c_master_read

函数	int i2c_master_read(uint16_t slave_addr, uint8_t* data, size_t data_len, bool dma_mode, TickType_t ticks_to_wait)
说明	从主机读数据
参数	slave_addr：uint16_t，外设地址 data：uint8_t*，读出数据 data_len：size_t，数据长度 dma_mode：bool，是否为 DMA 模式 ticks_to_wait：TickType_t
返回	ERROR_I2C_PARAM：失败 ERROR_I2C_BUSY：失败 ERROR_I2C_TIMEOUT：失败 I2C_OK：成功
说明	

2.2.5 i2c_slave_write

函数	int i2c_slave_write(uint8_t* data, size_t data_len, bool dma_mode, TickType_t ticks_to_wait)
说明	向 slave 设备写数据
参数	data：uint8_t*，写入数据 data_len：size_t，数据长度 dma_mode：bool，是否为 DMA 模式 ticks_to_wait：TickType_t
返回	ERROR_I2C_PARAM：失败 ERROR_I2C_BUSY：失败 ERROR_I2C_TIMEOUT：失败 I2C_OK：成功

说明	

2.2.6 i2c_slave_read

函数	int i2c_slave_read(uint8_t* data, size_t data_len, bool dma_mode, TickType_t ticks_to_wait)
说明	从 slave 设备读数据
参数	data：uint8_t*，读出数据 data_len：size_t，数据长度 dma_mode：bool，是否为 DMA 模式 ticks_to_wait：TickType_t
返回	ERROR_I2C_PARAM：失败 ERROR_I2C_BUSY：失败 ERROR_I2C_TIMEOUT：失败 I2C_OK：成功
说明	

2.2.7 i2c_reset_fifo

函数	int i2c_reset_fifo(void)
说明	重置 I2C 命令队列
参数	void
返回	I2C_OK：成功
说明	

2.2.8 i2c_bus_is_busy

函数	bool i2c_bus_is_busy(void)
说明	查询 I2C 总线是否可用
参数	void
返回	true：可用

	false：不可用
说明	

2.2.9 i2c_register_callback

函数	int i2c_register_callback(i2c_callback_t i2c_callback_func, void* arg)
说明	注册回调函数
参数	i2c_callback_func ： i2c_callback_t arg ： void*
返回	I2C_OK：成功
说明	

2.3 PWM

2.3.1 pwm_init

函数	int32_t pwm_init(uint32_t channel)
说明	pwm 指定通路初始化
参数	Channel（0-5）：PMW_CHANNEL_0 到 PMW_CHANNEL_5
返回	DRV_ERR_INVALID_PARAM：失败 DRV_SUCCESS：成功
说明	

2.3.2 pwm_deinit

函数	int32_t pwm_deinit(uint32_t channel)
说明	释放初始化资源（目前函数未实现）
参数	Channel（0-5）：PMW_CHANNEL_0 到 PMW_CHANNEL_5
返回	DRV_ERR_INVALID_PARAM：失败

	DRV_SUCCESS：成功
说明	

2.3.3 pwm_config

函数	int32_t pwm_config(uint32_t channel, uint32_t freq, uint32_t duty_ratio)
说明	配置 PWM
参数	channel（0-5）：PMW_CHANNEL_0 到 PMW_CHANNEL_5 freq：uint32_t，频率 duty_ratio：uint32_t，占空比
返回	DRV_ERR_INVALID_PARAM：失败 DRV_ERROR：失败 DRV_SUCCESS：成功
说明	

2.3.4 pwm_start

函数	int32_t pwm_start(uint32_t channel)
说明	指定通路 pwm 启动
参数	channel（0-5）：PMW_CHANNEL_0 到 PMW_CHANNEL_5
返回	DRV_ERR_INVALID_PARAM：失败 DRV_ERROR：失败 DRV_SUCCESS：成功
说明	

2.3.5 pwm_stop

函数	int32_t pwm_stop(uint32_t channel)
说明	指定通路 pwm 停止

参数	channel（0-5）：PMW_CHANNEL_0 到 PMW_CHANNEL_5
返回	DRV_ERR_INVALID_PARAM：失败 DRV_ERROR：失败 DRV_SUCCESS：成功
说明	

2.4 UART

2.4.1 uart_get_intType

函数	unsigned char uart_get_intType(unsigned int regBase)
说明	获取指定寄存器的中断类型
参数	regBase：unsigned int，地址
返回	UART_INT_NONE：0 UART_INT_RX_DONE：3 UART_INT_TX_EMPTY：4 UART_INT_TIMEOUT：2 UART_INT_ERROR：1
说明	

2.4.2 uart_set_baudrate

函数	void uart_set_baudrate(unsigned int regBase, unsigned int baud, unsigned int cond)
说明	设置指定寄存器的波特率
参数	regBase：unsigned int baud：unsigned int，波特率 cond：unsigned int， 0：主频 20MHz，1:主频 40MHz，其他：主频 80MHz

返回	void
说明	

2.4.3 uart_set_lineControl

函数	void uart_set_lineControl(unsigned int regBase, unsigned int databits, unsigned int parity, unsigned int stopbits, unsigned int bc)
说明	设置串口通信参数
参数	regBase：unsigned int，地址 databits：unsigned int 数据位/* 0：5bits, 1：6bits, 2：7bits, 3：8bits */ parity ：unsigned int 奇偶校验/* 0：无, 1：奇校验, 2：偶校验*/ stopbits ：unsigned int 停止位/* 0：1bit stopbits, 1：the num of stopbits is based on the databits*/ bc： unsigned int /* break control */
返回	void
说明	

2.4.4 uart_set_fifoControl

函数	void uart_set_fifoControl(unsigned int regBase, unsigned int tFifoRst, unsigned int rFifoRst,unsigned int fifoEn)
说明	设置 FIFO 通信参数
参数	regBase：unsigned int tFifoRst：unsigned int，FIFO 发送复位 rFifoRst：unsigned int，FIFO 接收复位 fifoEn：unsignd int，FIFO 使能
返回	void
说明	

2.4.5 uart_set_intEnable

函数	void uart_set_intEnable(unsigned int regBase, unsigned int tx, unsigned int rx)
说明	使能/不使能 TX/RX

参数	regBase：unsigned int tx：unsigned int，UART_INT_ENABLE/ UART_INT_DISABLE rx：unsigned int，UART_INT_ENABLE/ UART_INT_DISABLE
返回	void
说明	

2.4.6 uart_data_write

函数	void uart_data_write(unsigned int regBase, const unsigned char * buf, unsigned int len)
说明	uart 写数据
参数	regBase：unsigned int，地址 buf：const unsigned char *，数据 len：unsigned int，数据长度
返回	void
说明	

2.4.7 uart_data_tstc

函数	int uart_data_tstc(unsigned int regBase)
说明	查询 uart 是否收到数据
参数	regBase：unsigned int，地址
返回	1：收到数据 0：无数据
说明	

2.4.8 uart_data_getc

函数	unsigned char uart_data_getc(unsigned int regBase)
说明	如 uart 有收到数据，获取收到的数据，

参数	regBase：unsigned int，地址
返回	char：uart 收到的数据
说明	

2.4.9 hal_uart_register_recv_callback

函数	int hal_uart_register_recv_callback(uart_handle_t handle, void (* callback)(void *), void *data)
说明	注册回调函数
参数	handle ：uart_handle_t void *：void (* callback) data ：void *
返回	int：0
说明	

2.4.10 hal_uart_get_recv_len

函数	int hal_uart_get_recv_len(uart_handle_t handle)
说明	获取接受数据的长度
参数	handle ：uart_handle_t
返回	int：接收数据的长度
说明	

2.4.11 hal_uart_write

函数	void hal_uart_write(uart_handle_t handle, const unsigned char * buf, int len)
说明	uart 写数据，调用 2.4.6 uart_data_write 函数
参数	handle ：uart_handle_t buf ：const unsigned char *，数据 len：int，数据长度
返回	void

说明	

2.4.12 hal_uart_read

函数	unsigned int hal_uart_read(uart_handle_t handle, unsigned char * buf, int len)
说明	Uart 读数据
参数	handle ： uart_handle_t buf ： const unsigned char *，数据 len：int，数据长度
返回	读出数据的长度
说明	

2.4.13 hal_uart_close

函数	void hal_uart_close(uart_handle_t handle)
说明	关闭 uart 端口
参数	handle ： uart_handle_t
返回	void
说明	

2.4.14 hal_uart_open

函数	uart_handle_t hal_uart_open(unsigned int id, unsigned int databits, unsigned int baud, unsigned int parity, unsigned int stopbits, unsigned int flow)
说明	打开 uart 端口

参数	id：UART_ID_0，UART_ID_1，UART_ID_2 databits：/* 0--5bits, 1--6bits, 2--7bits, 3--8bits */ baud：波特率 parity：奇偶校验位 stopbits：停止位 flow：预留
返回	uart_handle_t
说明	

2.5 AES

2.5.1 aes_128_encrypt

函数	int32_t aes_128_encrypt(uint8_t *input,uint32_t input_len,const uint8_t *key,const uint8_t key_len,uint8_t *output)
说明	AES128 数据加密
参数	input ：待加密数据 input_len：待加密数据长度 key：加密 key key_len：key 长度 output：加密后数据
返回	输出数据的长度
说明	

2.5.2 aes_128_decrypt

函数	int32_t aes_128_decrypt(uint8_t *input,uint32_t input_len,const uint8_t *key,const uint8_t key_len,uint8_t *output)
说明	AES128 数据解密
参数	input ：待解密数据 input_len：待解密数据长度 key：解密 key key_len：key 长度 output：解密后数据
返回	输出数据的长度

说明	

2.6 ADC

2.6.1 vbat_sensor_get

函数	int16_t vbat_sensor_get(uint32_t volt_div)
说明	获得 vbat 电压
参数	volt_div：DRV_ADC_INPUT_VOL_DIV，支持 4/5/6 分压
返回	Vbat 电压值，单位 mV
说明	

2.6.2 tout_sensor_get

函数	int16_t tout_sensor_get(uint32_t tout_num,uint32_t volt_div,uint32_t volt)
说明	获得 TOUTx 端口的输入电压
参数	tout_num：DRV_ADC_INPUT_SEL，可选 0~3 通道 volt_div：DRV_ADC_INPUT_VOL_DIV，支持 4/5/6 分压 volt：输入的电压值，小于等于 1.2V 请选择 ADC_INPUT_VOLT_MIN， 大于 1.2V 请选择 ADC_INPUT_VOLT_MAX。
返回	Vbat 电压值，单位 mV
说明	此函数值支持单端输入

2.7 reset

函数	void wdt_reset_chip(int type)
说明	重启系统

参数	type 取以下 #define WDT_RESET_CHIP_TYPE_REBOOT 0 #define WDT_RESET_CHIP_TYPE_RESYSTEM 1
返回	void
说明	0：从 flash 0x0 地址重启 1：从内部 ram 64K 地址处重启

2.8 RTC

2.8.1 rtc_read_time

函数	time rtc_read_time(void)
说明	读取 RTC 时间
参数	void
返回	typedef struct { int32_t ms; uint8_t tm_sec; /* 秒 – 取值区间为[0,59] */ uint8_t tm_min; /* 分 - 取值区间为[0,59] */ uint8_t tm_hour; /* 时 - 取值区间为[0,23] */ }time;
说明	

2.8.2 rtc_set_time

函数	void rtc_set_time(struct tm sysTime)
说明	设置 RTC 时间
参数	struct tm { uint8_t tm_sec; /* 秒 – 取值区间为[0,59] */ uint8_t tm_min; /* 分 - 取值区间为[0,59] */ uint8_t tm_hour; /* 时 - 取值区间为[0,23] */

	uint8_t tm_mday; /* 一个月中的日期 - 取值区间为[1,31] */ uint8_t tm_mon; /* 月份（从一月开始，0 代表一月） - 取值区间为[0,11] */ uint16_t tm_year; /* 年份，其值等于实际年份减去 1900 */ uint8_t tm_wday; /* 星期 – 取值区间为[0,6]，其中 0 代表星期天，1 代表星期一，以此类推 */ };
返回	void
说明	

2.8.3 rtc_get_system_time

函数	struct tm rtc_get_system_time(void)
说明	获取系统时间（年月日分秒时）
参数	void
返回	struct tm { uint8_t tm_sec; /* 秒 – 取值区间为[0,59] */ uint8_t tm_min; /* 分 - 取值区间为[0,59] */ uint8_t tm_hour; /* 时 - 取值区间为[0,23] */ uint8_t tm_mday; /* 一个月中的日期 - 取值区间为[1,31] */ uint8_t tm_mon; /* 月份（从一月开始，0 代表一月） - 取值区间为[0,11] */ uint16_t tm_year; /* 年份，其值等于实际年份减去 1900 */ uint8_t tm_wday; /* 星期 – 取值区间为[0,6]，其中 0 代表星期天，1 代表星期一，以此类推 */

	};
说明	

2.9 timer

2.9.1 hal_timer_init

函数	int hal_timer_init(void)
说明	打开 timer 中断，并注册 isr。
参数	void
返回	0：成功 非 0：失败
说明	

2.9.2 hal_timer_config

函数	int hal_timer_config(unsigned int us, unsigned int reload)
说明	配置定时时间和是否重新装载。
参数	us：定时时间，单位 us； reload：是否自动装载 1：自动装载 0：单次定时
返回	0：成功 非 0：失败
说明	

2.9.3 hal_timer_start

函数	int hal_timer_start(void)
说明	开启定时器

参数	void
返回	0：成功 非 0：失败
说明	

2.9.4 hal_timer_stop

函数	int hal_timer_stop(void)
说明	停止定时器
参数	void
返回	0：成功 非 0：失败
说明	

2.9.5 hal_timer_callback_register

函数	void hal_timer_callback_register(void (* user_timer_cb_fun)(void *), void *data)
说明	注册用户回调函数
参数	user_timer_cb_fun：回调函数指针 data：回调函数入参。
返回	void
说明	

2.9.6 hal_timer_callback_unregister

函数	void hal_timer_callback_unregister(void)
说明	注销用户回调函数
参数	void

返回	void
说明	

2.10 flash

2.10.1 hal_spiflash_read

函数	int hal_spiflash_read(unsigned int addr, unsigned char * buf, unsigned int len)
说明	Nor flash 读
参数	addr：需要 4byte 对齐。 buf：数据指针； len：需要读取的数据长度。
返回	0：成功 非 0：失败
说明	

2.10.2 hal_spifiash_write

函数	int hal_spifiash_write(unsigned int addr, unsigned char * buf, unsigned int len)
说明	Nor flash 写
参数	addr：需要 4byte 对齐。 buf：数据指针； len：需要写的数据长度。
返回	0：成功 非 0：失败
说明	

2.10.3 hal_spiflash_erase

函数	int hal_spiflash_erase(unsigned int addr, unsigned int len)
说明	Nor flash 擦除

参数	addr：需要 4byte 对齐。 buf：数据指针； len：需要写的数据长度。
返回	0：成功 非 0：失败
说明	

2.10.4 spiFlash_OTP_Read

函数	int spiFlash_OTP_Read(int addr,int length,unsigned char *pdata)
说明	Nor flash OTP 区域读
参数	addr：需要 4byte 对齐。 length ：需要读取数据长度。 pdata：数据指针；
返回	0：成功 非 0：失败
说明	

2.10.5 hal_spifiash_OTPWrite

函数	int hal_spifiash_OTPWrite(unsigned int addr, unsigned int len, unsigned char * buf)
说明	Nor flash OTP 区写
参数	addr：需要 4byte 对齐。 len：需要写的数据长度。 buf：数据指针。
返回	0：成功 非 0：失败
说明	

2.10.6 spiFlash_OTP_Se

函数	int spiFlash_OTP_Se(unsigned int addr)
说明	Nor flash OTP 区擦除
参数	addr：需要擦除 otp 区域的地址。
返回	0：成功 非 0：失败
说明	一次擦除一个块的数据

2.10.7 spiFlash_OTP_Lock

函数	void spiFlash_OTP_Lock(int LB)
说明	Nor flash OTP 区锁定块
参数	LB：需要锁定的块，不同块在代码中对应不同地址。
返回	无返回值
说明	OTP 区域中块一旦锁定，读操作有效，写和擦除操作无效。

2.11 SPI host

2.11.1 spi_master_init

函数	void spi_master_init(spi_master_dev *spi_master_dev)
说明	SPI host 初始化
参数	typedef struct{ unsigned int inten; unsigned int addr_len; unsigned int data_len; spi_master_clk master_clk; spi_master_cmd cmd_write; spi_master_cmd cmd_read;

	}spi_master_dev;
	
	其中：
	typedef struct {
	unsigned char cmd; //flash SPI Command
	unsigned int transCtrl; //SPI Transfer Control
	}spi_master_cmd;
	
	typedef enum{
	SPI_MASTER_FREQ_40M=0x00,
	SPI_MASTER_FREQ_20M,
	SPI_MASTER_FREQ_13M,
	SPI_MASTER_FREQ_10M,
	SPI_MASTER_FREQ_80M=0xFF
	}spi_master_clk;

返回	void
说明	

2.11.2 spi_master_read

函数	int spi_master_read(int addr,int length,unsigned int *buf)
说明	读取数据
参数	add：slave 地址 length：数据长度 buf：数据指针
返回	0：成功 非 0：失败
说明	

2.11.3 spi_master_write

函数	int spi_master_write(int addr,int length,unsigned int *buf)

说明	发送数据
参数	add：slave 地址 length：数据长度 buf：数据指针
返回	0：成功 非 0：失败
说明	

2.12 watchdog

2.12.1 wdt_init

函数	void wdt_init(void)
说明	watchdog 初始化
参数	无
返回	无
说明	

2.12.2 wdt_config

函数	void wdt_config(int intr_period, int rst_period, int intr_enalbe, int rst_enalbe)
说明	watchdog 参数配置
参数	intr_period：中断周期，取值范围如下 WDT_INTR_PERIOD_0 // 1/512 sec WDT_INTR_PERIOD_1 // 1/128 sec WDT_INTR_PERIOD_2 // 1/32 sec WDT_INTR_PERIOD_3 // 1/16 sec WDT_INTR_PERIOD_4 // 1/8 sec WDT_INTR_PERIOD_5 // 1/4 sec WDT_INTR_PERIOD_6 // 1/2 sec WDT_INTR_PERIOD_7 // 1 sec WDT_INTR_PERIOD_8 // 4 sec

	WDT_INTR_PERIOD_9 // 16 sec WDT_INTR_PERIOD_10 // 64 sec WDT_INTR_PERIOD_11 // 4min 16 sec WDT_INTR_PERIOD_12 // 17min 4sec WDT_INTR_PERIOD_13 // 1hour 8min 16sec WDT_INTR_PERIOD_14 // 4hour 33min 4sec WDT_INTR_PERIOD_15 // 18hour 12min 16sec rst_period: 复位时间，取值范围如下 WDT_RST_TIME_0 // 1/256 sec WDT_RST_TIME_1 // 1/128 sec WDT_RST_TIME_2 // 1/64 sec WDT_RST_TIME_3 // 1/32 sec WDT_RST_TIME_4 // 1/16 sec WDT_RST_TIME_5 // 1/8 sec WDT_RST_TIME_6 // 1/4 sec WDT_RST_TIME_7 // 1/2 sec intr_enalbe：中断使能/去使能 WDT_INTR_ENABLE WDT_INTR_DISABLE rst_enalbe：复位使能/去使能 WDT_RST_ENABLE WDT_RST_DISABLE
返回	无
说明	

2.12.3 wdt_isr_register

函数	void wdt_isr_register(void (*func)(void))
说明	watchdog 中断回调函数注册
参数	func：中断回调函数
返回	无
说明	

2.12.4 wdt_restart

函数	void wdt_restart(void)
说明	watchdog 重新启动。watchdog 计时器到了，使用该函数可以使计时器重新计时，避免 watchdog 复位芯片
参数	无
返回	无
说明	

2.12.5 wdt_start

函数	void wdt_start(int type)
说明	watchdog 启动
参数	type：watchdog 复位芯片的方式 WDT_RESET_CHIP_TYPE_REBOOT WDT_RESET_CHIP_TYPE_RESYSTEM
返回	无
说明	

2.12.6 wdt_reset_chip

函数	void wdt_reset_chip(int type)
说明	watchdog 触发芯片复位
参数	type：watchdog 复位芯片的方式 WDT_RESET_CHIP_TYPE_REBOOT WDT_RESET_CHIP_TYPE_RESYSTEM
返回	无
说明	

3 WIFI API

3.1 wifi_set_opmode

函数	void wifi_set_opmode(wifi_work_mode_e opmode)
功能	设置工作模式
参数	opmode: 工作模式 typedef enum { WIFI_MODE_STA, /**< WiFi station mode */ WIFI_MODE_AP, /**< WiFi soft-AP mode */ WIFI_MODE_AP_STA, /**< WiFi station + soft-AP mode */ WIFI_MODE_MAX } wifi_work_mode_e;
返回	无
说明	

3.2 wifi_get_opmode

函数	wifi_work_mode_e wifi_get_opmode(void)
功能	获取工作模式
参数	无
返回	当前工作模式
说明	

3.3 wifi_system_init

函数	sys_err_t wifi_system_init(void)
功能	wifi 模块初始化
参数	无

返回	typedef int32_t sys_err_t;
	#define SYS_OK 0x0
	#define SYS_ERR 0x1
	// for psm
	#define SYS_PM_NOT_HANDLED 0x10
	#define SYS_DVEICE_NOT_SUPPORT 0x11
	#define SYS_SOC_SLEEP_MODE_ERR 0x12
	#define SYS_SYS_SLEEP_MODE_ERR 0x13
	// for common error
	#define SYS_ERR_NO_MEM 0x101
	#define SYS_ERR_INVALID_ARG 0x102
	#define SYS_ERR_INVALID_STATE 0x103
	#define SYS_ERR_INVALID_SIZE 0x104
	#define SYS_ERR_NOT_FOUND 0x105
	#define SYS_ERR_NOT_SUPPORTED 0x106
	#define SYS_ERR_TIMEOUT 0x107
	#define SYS_ERR_INVALID_RESPONSE 0x108
	#define SYS_ERR_INVALID_CRC 0x109
	#define SYS_ERR_INVALID_VERSION 0x10A
	#define SYS_ERR_INVALID_MAC 0x10B
	// for wifi
	#define SYS_ERR_WIFI_BASE 0x3000/*Starting number of WiFi error codes*/
	#define SYS_ERR_WIFI_MODE 0x3001 /*wifi operate mode error*/
	#define SYS_ERR_WIFI_BUSY 0X3002 /*wifi busy*/

说明	只需初始化一次即可

3.4 wifi_set_status

函数	void wifi_set_status(int vif, wifi_status_e status)
功能	设置网络状态
参数	vif: 网络接口 typedef enum { STATION_IF = 0, SOFTAP_IF, MAX_IF } wifi_interface_e; status: 网络状态 typedef enum {

	// ap status.
	AP_STATUS_STOP,
	AP_STATUS_STARTED,
	// sta status.
	STA_STATUS_STOP,
	STA_STATUS_START,
	STA_STATUS_DISCON,
	STA_STATUS_CONNECTED
	} wifi_status_e;

返回	无
说明	系统会自动设置该状态

3.5 wifi_get_status

函数	wifi_status_e wifi_get_status(int vif)
功能	获取网络状态
参数	vif: 0=sta, 1=ap
返回	当前网络状态
说明	

3.6 wifi_get_ap_status

函数	wifi_status_e wifi_get_ap_status(void)
功能	获取 softap 状态
参数	无
返回	softap 状态
说明	

3.7 wifi_get_sta_status

函数	wifi_status_e wifi_get_sta_status(void)
功能	获取 station 状态
参数	无
返回	station 状态
说明	

3.8 wifi_add_config

函数	int wifi_add_config(int vif)
功能	添加网络接口
参数	vif
返回	0: OK -1: FAIL
说明	

3.9 wifi_remove_config_all

函数	int wifi_remove_config_all(int vif)
功能	删除所有网络接口
参数	vif
返回	0: OK -1: FAIL
说明	

3.10 wifi_config_ssid

函数	int wifi_config_ssid(int vif, char *ssid)

功能	配置 ssid
参数	vif ssid: 字符串，最大 32 字节
返回	0: OK -1: FAIL
说明	

3.11 wifi_config_ap_mode

函数	int wifi_config_ap_mode(int vif)
功能	配置工作模式为 softap
参数	vif
返回	0: OK -1: FAIL
说明	默认为 station 模式

3.12 wifi_config_channel

函数	int wifi_config_channel(int vif, int channel)
功能	配置 softap 信道参数
参数	vif channel: 信道 1~13
返回	0: OK -1: FAIL
说明	station 模式不需要配置

3.13 wifi_rf_set_channel

函数	sys_err_t wifi_rf_set_channel(uint8_t channel)

功能	配置系统工作信道
参数	channel: 信道 1~13
返回	0: OK -1: FAIL
说明	

3.14 wifi_rf_get_channel

函数	uint8_t wifi_rf_get_channel(void)
功能	获取系统工作信道
参数	无
返回	当前信道
说明	

3.15 wifi_config_encrypt

函数	int wifi_config_encrypt(int vif, char *pwd, wifi_auth_mode_e mode)
功能	配置加密模式
参数	vif pwd: 字符串密码，8~63 字节 mode: 加密模式 typedef enum { AUTH_OPEN = 0, /**< authenticate mode : open */ AUTH_WEP, /**< authenticate mode : WEP */ AUTH_WPA_PSK, /**< authenticate mode : WPA_PSK */ AUTH_WPA2_PSK, /**< authenticate mode : WPA2_PSK */ AUTH_WPA_WPA2_PSK, /**< authenticate mode : WPA_WPA2_PSK */ AUTH_MAX } wifi_auth_mode_e;
返回	0: OK -1: FAIL

说明	OPEN 模式 pwd=NULL

3.16 wifi_set_scan_hidden_ssid

函数	sys_err_t wifi_set_scan_hidden_ssid(void)
功能	使能扫描隐藏 SSID 的网络
参数	无
返回	0: OK -1: FAIL
说明	默认不使能

3.17 wifi_config_commit

函数	int wifi_config_commit(int vif)
功能	使能网络
参数	vif
返回	0: OK -1: FAIL
说明	

3.18 wifi_start_softap

函数	sys_err_t wifi_start_softap(wifi_config_u *config)
功能	创建 softap

参数	config
	typedef union {
	wifi_ap_config_t ap; /**< configuration of AP */
	wifi_sta_config_t sta; /**< configuration of STA */
	} wifi_config_u;
	typedef struct {
	char ssid[WIFI_SSID_MAX_LEN]; /**< SSID of target AP*/

	char password[WIFI_PWD_MAX_LEN]; /**< password of target AP*/ } wifi_sta_config_t; typedef struct { char ssid[WIFI_SSID_MAX_LEN]; char password[WIFI_PWD_MAX_LEN]; uint8_t channel; wifi_auth_mode_e authmode; } wifi_ap_config_t;
返回	0: OK -1: FAIL
说明	

3.19 wifi_stop_softap

函数	sys_err_t wifi_stop_softap(void)
功能	删除 softap
参数	无
返回	0: OK -1: FAIL
说明	

3.20 wifi_start_station

函数	sys_err_t wifi_start_station(wifi_config_u *config)
功能	创建 station
参数	config
返回	0: OK -1: FAIL
说明	

3.21 wifi_stop_station

函数	sys_err_t wifi_stop_station(void)

功能	删除 station
参数	无
返回	0: OK -1: FAIL
说明	

3.22 wifi_scan

函数	sys_err_t wifi_scan(int block, wifi_info_t *scan_result)
功能	扫描网络
参数	block: 0=非阻塞，1=阻塞 scan_result: typedef struct { uint8_t ssid[WIFI_SSID_MAX_LEN]; uint8_t pwd[WIFI_PWD_MAX_LEN]; uint8_t ssid_len; uint8_t pwd_len; uint8_t auth; uint8_t cipher; uint8_t channel; uint8_t bssid[6]; int8_t rssi; } wifi_info_t;
返回	0: OK -1: FAIL
说明	非阻塞模式下，调用回调函数 void (*user_scan_callback)(void *parameter) 阻塞模式下，将所有扫描到的网络信息保存到 scan_result，需保证有足够空间

3.23 wifi_get_scan_result

函数	sys_err_t wifi_get_scan_result(unsigned int index, wifi_info_t *info)
功能	获取扫描结果

参数	index: 扫描到的网络序号，从 0 开始 info: 扫描结果结构体指针
返回	0: OK -1: FAIL
说明	

3.24 wifi_get_wifi_info

函数	sys_err_t wifi_get_wifi_info(wifi_info_t *info)
功能	获取当前网络信息
参数	info: 网络信息结构体指针
返回	0: OK -1: FAIL
说明	

3.25 wifi_get_mac_addr

函数	sys_err_t wifi_get_mac_addr(unsigned char *mac)
功能	获取 MAC 地址
参数	mac: mac 指针
返回	0: OK -1: FAIL
说明	mac 长度 6 个字节，中间没有分隔符

3.26 wifi_get_ip_addr

函数	sys_err_t wifi_get_ip_addr(unsigned int *ip)
功能	获取 IP 地址
参数	ip: ip 指针

返回	0: OK -1: FAIL
说明	

3.27 wifi_get_mask_addr

函数	sys_err_t wifi_get_mask_addr(unsigned int *mask)
功能	获取子网掩码
参数	mask: mask 指针
返回	0: OK -1: FAIL
说明	

3.28 wifi_get_gw_addr

函数	sys_err_t wifi_get_gw_addr(unsigned int *gw)
功能	获取 gateway 地址
参数	gw: gw 指针
返回	0: OK -1: FAIL
说明	

3.29 wifi_get_dns_addr

函数	sys_err_t wifi_get_dns_addr(unsigned int index, unsigned int *dns)
功能	获取 dns 地址
参数	index: 0=dns1, 1=dns2 dns: dns 指针
返回	0: OK -1: FAIL

说明	

3.30 wifi_send_raw_pkt

函数	sys_err_t wifi_send_raw_pkt(const uint8_t *frame, const uint16_t len)
功能	发送任意数据
参数	frame: 数据指针 len: 数据长度
返回	0: OK -1: FAIL
说明	

3.31 wifi_sniffer_start

函数	sys_err_t wifi_sniffer_start(wifi_promiscuous_cb_t cb, wifi_sniffer_filter_t *filter)
功能	开启 sniffer 模式
参数	cb: 回调函数 typedef void (*wifi_promiscuous_cb_t)(const uint8_t *, uint16_t); filter: 过滤类型参数 typedef struct { uint8_t extend; uint8_t type; uint16_t subtype; } wifi_sniffer_filter_t;
返回	0: OK -1: FAIL
说明	wifi_sniffer_filter_t 结构体的类型选择见 protocol.h。

3.32 wifi_sniffer_ext_start

函数	sys_err_t wifi_sniffer_ext_start(wifi_promiscuous_ext_cb_t cb, wifi_sniffer_filter_t *filter)

功能	开启 sniffer 模式
参数	cb: 回调函数 typedef void (*wifi_promiscuous_ext_cb_t)(int8_t rssi, const uint8_t *, uint16_t); filter: 过滤类型参数 typedef struct { uint8_t extend; uint8_t type; uint16_t subtype; } wifi_sniffer_filter_t;
返回	0: OK -1: FAIL
说明	1. wifi_sniffer_filter_t 结构体的类型选择见 protocol.h； 2. 可获得 RSSI 地址；

3.33 wifi_sniffer_stop

函数	sys_err_t wifi_sniffer_stop(void)
功能	关闭 sniffer 模式
参数	无
返回	0: OK -1: FAIL
说明	

3.34 wifi_system_init_complete

函数	unsigned char wifi_system_init_complete(void)
功能	标识 WPA 及 WIFI 协议栈是否初始化完成
参数	无

返回	0: 未初始化完成。 1: 初始化完成。
说明	

4 PSM

4.1 TrPsmSetSysDeepSleep

函数	uint32_t TrPsmSetSysDeepSleep(uint32_t deepsleep, uint32_t sleepperiod)
功能	设置系统进入/退出深度睡眠
参数	Deepsleep：TR_PSM_DEEP_SLEEP_MODE Sleepperiod：睡眠时间，单位秒
返回	0: OK 其他: FAIL
说明	

4.2 TrPsmGetSysDeepSleep

函数	TR_PSM_DEEP_SLEEP_MODE TrPsmGetSysDeepSleep(TR_PSM_CONTEXT *ctx)
功能	获取系统睡眠状态
参数	ctx：TR_PSM_CONTEXT
返回	0: OK 其他: FAIL
说明	

5 PIN_MUX

5.1 PIN_FUNC_SET

函数	PIN_FUNC_SET(PIN_NAME,PIN_FUNC) \

	PIN_MUX_SET(PIN_NAME##_REG，PIN_NAME##_BITS, PIN_NAME##_OFFSET, PIN_FUNC)
功能	设置 GPIO 的工作模式，功能引脚，或者 GPIO
参数	PIN_NAME、PIN_FUNC：见 soc_pin_mux.h
返回	0: OK 其他: FAIL
说明	

6 SOCKET

6.1 accept

函数	int accept(int s, struct sockaddr *addr, socklen_t *addrlen)
功能	获取 client 的接收请求
参数	s: socket 描述符 addr: 协议地址 addrlen: 协议地址长度
返回	>0: OK，新建立的 socket 描述符 -1: FAIL
说明	

6.2 bind

函数	int bind(int s, const struct sockaddr *name, socklen_t namelen)
功能	绑定 socket 描述符与协议地址
参数	s: socket 描述符

	name: 协议地址 namelen: 协议地址长度
返回	0: OK 其他: FAIL
说明	

6.3 shutdown

函数	int shutdown(int s, int how)
功能	关闭 socket 连接
参数	s: socket 描述符 how: 0=rx, 1=tx, 2=rx+tx
返回	0: OK 其他: FAIL
说明	

6.4 getpeername

函数	int getpeername(int s, struct sockaddr *name, socklen_t *namelen)
功能	获取与 s 连接的对方协议地址信息
参数	s: socket 描述符 name: 协议地址 namelen: 协议地址长度
返回	0: OK 其他: FAIL
说明	

6.5 getsockname

函数	int getsockname(int s, struct sockaddr *name, socklen_t *namelen)
功能	获取与 s 关联的本地协议地址信息
参数	s: socket 描述符 name: 协议地址 namelen: 协议地址长度
返回	0: OK 其他: FAIL
说明	

6.6 setsockopt

函数	int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen)
功能	设置 socket 选项
参数	s: socket 描述符 level: 选项所在协议层 optname: 选项名 optval: 选项值 optlen: 选项长度
返回	0: OK 其他: FAIL
说明	

6.7 getsockopt

函数	int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen)
功能	获取 socket 选项
参数	s: socket 描述符 level: 选项所在协议层 optname: 选项名 optval: 选项值 optlen: 选项长度
返回	0: OK 其他: FAIL
说明	

6.8 closesocket

函数	int closesocket(int s)
功能	关闭 socket 连接
参数	s: socket 描述符
返回	0: OK 其他: FAIL
说明	

6.9 connect

函数	int connect(int s, const struct sockaddr *name, socklen_t namelen)
功能	client 请求连接

参数	s: socket 描述符 name: 协议地址 namelen: 协议地址长度
返回	0: OK 其他: FAIL
说明	

6.10 listen

函数	int listen(int s, int backlog)
功能	server 监听 socket 端口
参数	s: socket 描述符 backlog: 最大连接数
返回	0: OK 其他: FAIL
说明	

6.11 recv

函数	int recv(int s, void *mem, size_t len, int flags)
功能	接收数据
参数	s: socket 描述符 mem: 接收数据地址 len: 接收数据长度

	flags: 选项，一般为 0
返回	>0: 接收长度 其他: FAIL
说明	

6.12 recvfrom

函数	int recvfrom(int s, void *mem, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)
功能	接收数据
参数	s: socket 描述符 mem: 接收数据地址 len: 接收数据长度 flags: 选项，一般为 0 from: 源端协议地址 fromlen: 源端地址长度
返回	>0: 接收长度 其他: FAIL
说明	

6.13 send

函数	int send(int s, const void *data, size_t size, int flags)
功能	发送数据
参数	s: socket 描述符

	data: 发送数据地址 size: 发送数据长度 flags: 选项，一般为 0
返回	>0: 发送长度 其他: FAIL
说明	

6.14 sendto

函数	int sendto(int s, const void *data, size_t size, int flags, const struct sockaddr *to, socklen_t tolen)
功能	发送数据
参数	s: socket 描述符 data: 发送数据地址 size: 发送数据长度 flags: 选项，一般为 0 to: 目标协议地址 tolen: 目标协议地址长度
返回	>0: 发送长度 其他: FAIL
说明	

6.15 socket

函数	int socket(int domain, int type, int protocol)
功能	创建 socket 描述符
参数	domain: 协议族 type: socket 类型 protocol: 协议类型
返回	其他: socket 描述符 -1: FAIL
说明	

6.16 select

函数	int select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)
功能	select 操作
参数	maxfdp1: 所有文件描述符的范围 readset: 读文件描述符 writeset: 写文件描述符 exceptset: 异常文件描述符 timeout: 超时时间
返回	>0: 可操作的文件描述符 0: 超时返回 <0: FAIL
说明	

7 NV

7.1 easyflash_init

函数	EfErrCode easyflash_init(void)
功能	初始化 NV 系统
参数	void
返回	EF_NO_ERR：初始化成功 EF_ERASE_ERR：擦除 flash 过程出错 EF_READ_ERR：读 flash 出错 EF_WRITE_ERR：写 flash 出错 EF_ENV_NAME_ERR：key 名称出错 EF_ENV_FULL：空间满 EF_ENV_INIT_FAILED：初始化出错
说明	NV 初始化函数是使用 nv 功能必须要先调用且判断返回值的.

7.2 ef_set_env_blob

函数	EfErrCode ef_set_env_blob(const char *key, const void *value_buf, size_t buf_len)
功能	设置 key 值进入 nv 系统
参数	key：key 值指针 value_buf: value 的指针 buf_len:value 的实际长度
返回	EF_NO_ERR：初始化成功 EF_ERASE_ERR：擦除 flash 过程出错

	EF_READ_ERR：读 flash 出错 EF_WRITE_ERR：写 flash 出错 EF_ENV_NAME_ERR：key 名称出错 EF_ENV_FULL：空间满 EF_ENV_INIT_FAILED：初始化出错
说明	

7.3 ef_get_env_blob

函数	size_t ef_get_env_blob(const char *key, void *value_buf, size_t buf_len, size_t *value_len)
功能	获取 flash 中 key 的 value 值
参数	key：key 值指针 value_buf: 输出 value 的 buff 指针 buf_len:需要获取的 value 的长度 value_len：实际获取的到了 value 长度，若是 key 不存在，该指针指向内容不会改变
返回	0: 没有正确获取到 key 的对应 value 值 其他: 获取成功 EF_ENV_NAME_ERR：key 名称出错 EF_ENV_FULL：空间满 EF_ENV_INIT_FAILED：初始化出错
说明	

7.4 ef_print_env

函数	void ef_print_env(void)

功能	获取 NV 系统在 flash 中的使用情况
参数	void
返回	void
说明	打印获取 nv 的 flash 资源的使用情况

7.5 ef_del_env

函数	EfErrCode ef_del_env(const char *key)
功能	删除 key 值
参数	key：key 值指针
返回	EF_NO_ERR：初始化成功 EF_ERASE_ERR：擦除 flash 过程出错 EF_READ_ERR：读 flash 出错 EF_WRITE_ERR：写 flash 出错 EF_ENV_NAME_ERR：key 名称出错 EF_ENV_FULL：空间满 EF_ENV_INIT_FAILED：初始化出错
说明	

7.6 backup_set_env_blob

函数	EfErrCode backup_set_env_blob(const char *key, const void *value_buf, size_t buf_len)
功能	设置 key 值进入 backup nv 系统
参数	key：key 值指针

	value_buf: value 的指针 buf_len:value 的实际长度
返回	EF_NO_ERR：初始化成功 EF_ERASE_ERR：擦除 flash 过程出错 EF_READ_ERR：读 flash 出错 EF_WRITE_ERR：写 flash 出错 EF_ENV_NAME_ERR：key 名称出错 EF_ENV_FULL：空间满 EF_ENV_INIT_FAILED：初始化出错
说明	

7.7 backup_get_env_blob

函数	size_t backup_get_env_blob(const char *key, void *value_buf, size_t buf_len, size_t *value_len)
功能	获取 flash 中 backup 分区 key 的 value 值
参数	key：key 值指针 value_buf: 输出 value 的 buff 指针 buf_len:需要获取的 value 的长度 value_len：实际获取的到了 value 长度，若是 key 不存在，该指针指向内容不 会改变
返回	0: 没有正确获取到 key 的对应 value 值 其他: 获取成功 EF_ENV_NAME_ERR：key 名称出错

	EF_ENV_FULL：空间满 EF_ENV_INIT_FAILED：初始化出错
说明	

7.8 backup_del_env

函数	EfErrCode backup_del_env(const char *key)
功能	删除 backup 分区 key 值
参数	key：key 值指针
返回	EF_NO_ERR：初始化成功 EF_ERASE_ERR：擦除 flash 过程出错 EF_READ_ERR：读 flash 出错 EF_WRITE_ERR：写 flash 出错 EF_ENV_NAME_ERR：key 名称出错 EF_ENV_FULL：空间满 EF_ENV_INIT_FAILED：初始化出错
说明	

7.9 backup_recovery

函数	void backup_recovery(void)
功能	将 backup 分区的所有 nv 项复制到 user nv 中
参数	无

返回	无
说明	

8 SNTP

8.1 sntp_setoperatingmode

函数	void sntp_setoperatingmode(u8_t operating_mode)
功能	设置 SNTP 工作模式
参数	#define SNTP_OPMODE_POLL 0 #define SNTP_OPMODE_LISTENONLY 1 operating_mode: 默认用的是 SNTP_OPMODE_POLL
返回	无
说明	

8.2 sntp_getoperatingmode

函数	u8_t sntp_getoperatingmode(void)
功能	获取 SNTP 工作模式
参数	无
返回	工作模式
说明	

8.3 sntp_setserver

函数	void sntp_setserver(u8_t idx, const ip_addr_t *addr)
功能	设置 SNTP 服务器地址

参数	idx：服务器序号 addr：IP 地址
返回	无
说明	

8.4 sntp_getserver

函数	const ip_addr_t* sntp_getserver(u8_t idx)
功能	获取 SNTP 服务器地址
参数	idx：服务器序号
返回	IP 地址
说明	

8.5 sntp_setservername

函数	void sntp_setservername(u8_t idx, const char *server)
功能	设置 sntp 服务器域名
参数	idx：服务器序号 server：域名
返回	无
说明	

8.6 sntp_getservername

函数	const char *sntp_getservername(u8_t idx)

功能	获取 sntp 服务器域名
参数	idx：服务器序号
返回	域名
说明	

8.7 sntp_getreachability

函数	u8_t sntp_getreachability(u8_t idx)
功能	查询 sntp 结果状态
参数	idx：服务器序号
返回	0: sntp 功能未使能，或正在获取服务器时间（获取失败会一直重试） 1: sntp 时间成功
说明	初始为 0，当 sntp 成功后为 1，保持一个循环周期，直到下次 sntp 开始变为 0

8.8 sntp_enabled

函数	u8_t sntp_enabled(void)
功能	查询 sntp 使能状态
参数	无
返回	0: sntp 未使能 1: sntp 使能
说明	

8.9 sntp_stop

函数	void sntp_stop(void)
功能	关闭 sntp
参数	无
返回	无
说明	

8.10 sntp_init

函数	void sntp_init(void)
功能	初始化 sntp
参数	无
返回	无
说明	

8.11 sntp_start

函数	void sntp_start(void)
功能	使能 sntp
参数	无
返回	无
说明	sntp_start 调用 sntp_init，系统获取到 ip 后会自动调用 sntp_start sntp_start 会获取 flash 中存储的参数，如果获取失败，则使用默认值，timezone

	= 0，sntp_period = 3600000，sntp_server = pool.ntp.org

8.12 sntp_restart

函数	void sntp_restart(void)
功能	立即执行一次 sntp
参数	无
返回	无
说明	必须使能 sntp

8.13 set_sntp_period

函数	int set_sntp_period(unsigned int period)
功能	设置 sntp 周期，并立即调用一次 sntp_restart
参数	period: 单位 ms，Must not be beolw 60 seconds by specification
返回	0: OK -1: FAIL
说明	新的 period 值会保存到 flash

8.14 get_sntp_period

函数	unsigned int get_sntp_period(void)
功能	获取 sntp 周期
参数	无
返回	sntp 周期，单位 ms
说明	

8.15 set_timezone

函数	int set_timezone(char tz)
功能	设置时区，并立即调用一次 sntp_restart
参数	tz: 时区，有符号整数，-12,-11.....-1,0,1.....11,12
返回	0: OK -1: FAIL
说明	新的 timezone 值会保存到 flash

8.16 get_timezone

函数	char get_timezone(void)
功能	获取时区
参数	无
返回	时区，有符号整数，-12,-11.....-1,0,1.....11,12
说明	

8.17 set_servername

函数	int set_servername(unsigned char idx, const char *s)
功能	设置 sntp 服务器
参数	idx：服务器序号 server：服务器名称
返回	0: OK -1: FAIL
说明	新的 server 值会保存到 flash

8.18 get_servername

函数	char *get_servername(unsigned char idx)
功能	获取 sntp 服务器
参数	idx：服务器序号
返回	sntp 服务器名称
说明	

9 OTA

9.1 otaHal_init

函数	char otaHal_init(void)
功能	OTA 底层初始化
参数	无
返回	0：成功 -1：失败
说明	确认并准备待升级 flash 分区

9.2 otaHal_write

函数	char otaHal_write(const unsigned char * data, unsigned short len)
功能	OTA 分区烧写
参数	data：升级包地址 len：升级包长度
返回	0：成功

	1：失败
说明	待升级版本会被拆分成多个升级包，该函数可以把单个升级包烧写到 OTA 分区中，按顺序烧写单个升级包就能够将整个版本烧写完成。

9.3 otaHal_done

函数	char otaHal_done(void)
功能	OTA 升级完成，更新 OTA NV，同时触发重启切换版本
参数	无
返回	0：成功 -1：失败
说明	

10 AMT

10.1 amt_nv_init

函数	void amt_nv_init(void)
功能	初始化 AMT 分区
参数	无
返回	无
说明	

10.2 amt_nv_write

函数	int amt_nv_write(unsigned int addr, unsigned char * buf, unsigned int len)

功能	设置 key 值对应地址所在的数据
参数	addr：key 所对应的 addr 地址 buf: key 对应要写入的 value 的指针 len:value 的实际长度
返回	0：成功 非 0：失败 DRV_ERR_INVALID_PARAM：输入参数无效 DRV_ERR_MEM_ALLOC：申请内存失败
说明	

10.3 amt_nv_read

函数	int amt_nv_read(unsigned int addr, unsigned char * buf, unsigned int len)
功能	获取 flash 中 amt 分区 key 分配地址的 value 值
参数	addr：key 所对应的 addr 地址 buf: key 对应要读出的 value 的指针 len:value 的实际长度
返回	0：成功 非 0：失败 DRV_ERR_INVALID_PARAM：输入参数无效
说明	