旷明 XOS 驱动接口使用说明

部门	
文档编号	
版 本 号	V1.0.3
作　　者	

更新记录

日期	更新人	版本	备注
2024/08/08	QUA	V1.0.0	初稿
2024/11/20	QUA	V1.0.1	增加mass_storage
2024/12/20	QUA	V1.0.2	优化hotplug模块
2024/12/29	QUA	V1.0.3	添加motor控制

前言

概述

本文主要描述了驱动接口开发组件 ，仅供参考

产品版本

芯片名称	内核版本
mc331x	linux-4.9.138
qm10xd	linux-5.10.y
qm10xh	linux-5.10.y
qm10xv	linux-4.9

读者对象

本文档（本指南）主要适用于以下工程师： 技术支持工程师

软件开发工程师

目录

1. 系统概述

Sysutils 是基于 sysfs 封装的一套用户态接口，包括外设接口和系统功能接口，方便应用层对外 设和系统的控制，简化了应用开发难度，方便客户基于这些硬件接口进行应用开发。

2. GPIO

2.1 概述

提供 gpio 基本的用户态接口。

2.2 API 参考

2.2.1 qm_gpio_export

【描述】 将指定的 GPIO 管脚导出到用户空间，使其可以通过文件系统进行访问 【语法】 int qm_gpio_export(uint32_t gpio);

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要具有合适的权限来导出 GPIO 管脚

【注意】需要查看文件系统中是否有/sys/class/gpio 节点，如果没有该节点，就需要在编译内核时 勾选 Device Drivers-> GPIO Support ->/sys/class/gpio/ … (sysfs interface)，对应的 CONFIG 名字为

GPIO_SYSFS; GPIO 管脚编号需要在不被占用的状态下; GPIO 管脚编号必须未被导出过，否则导 出操作会失败

【举例】 无

【相关主题】 qm_gpio_unexport

2.2.2 qm_gpio_unexport

【描述】 取消导出指定的 GPIO 管脚，将其从用户空间中移除 【语法】 int qm_gpio_unexport(uint32__t__gpio);

【参数】

参数名称	描述	输入/输出

uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要在此之前已经通过 qm_gpio_export 成功导出 GPIO 管脚 【注意】 取消导出后， GPIO 将无法再通过用户空间访问

【举例】 无

【相关主题】 qm_gpio_export

2.2.3 qm_gpio_set_direction

【描述】 设置指定 GPIO 管脚的方向为输入或输出

【语法】 int qm_gpio_set_direction(uint32_t gpio, bool input); 【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入
bool input	true:in(输入) false:out(输出)	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要确保 GPIO 管脚已导出

【注意】 在设置为输出之前，必须确保没有其他冲突或锁定该 GPIO 的进程 【举例】 无

【相关主题】 qm_gpio_get_direction

2.2.4 qm_gpio_get_direction

【描述】获取 gpio 引脚的方向信息

【语法】 int qm_gpio_get_direction(uint32_t gpio)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0或1	0:out(输出) 1:in(输入)
负	失败

【需求】 GPIO 管脚必须已成功导出

【注意】 返回值为 0 表示该 GPIO 设置为输出，为 1 表示输入 【举例】 无

【相关主题】 qm_gpio_set_direction

2.2.5 qm_gpio_export_direction

【描述】 gpio 初始化，导出 gpio 的同时指定 gpio 的方向

【语法】 int qm_gpio_export_direction(uint32_t gpio, bool input) 【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入
bool input	true:in(输入) false:out(输出)	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 必须具有权限来导出并设置 GPIO 的方向

【注意】 此操作会同时导出 GPIO 和设置其方向，避免分别调用 qm_gpio_export() 和 qm_gpio_set_direction()

【举例】 无

【相关主题】 qm_gpio_export; qm_gpio_set_direction

2.2.6 qm_gpio_set_value

【描述】设置 GPIO 管脚的值

【语法】 int qm_gpio_set_value(uint32_t gpio, int value) 【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入
int value	0 表示低电平，非零表示高电平	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 GPIO 必须已导出并设置为输出方向

【注意】 如果 GPIO 设置为输入，不能设置其电平值，操作会失败 【举例】 无

【相关主题】 qm_gpio_get_value

2.2.7 qm_gpio_get_value

【描述】获取 GPIO 管脚的值

【语法】 int qm_gpio_get_value(uint32_t gpio) 【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0	低电平
1	高电平
非0	失败

【需求】 GPIO 必须已导出并设置为有效方向

【注意】 对于输入 GPIO，可以直接读取电平值;对于输出 GPIO，获取的是当前输出的电平状态 【举例】 无

【相关主题】 qm_gpio_set_value

3. ADC

3.1 概述

提供基本的 ADC 用户态接口

3.2 API 参考

3.2.1 qm_adc_get_devnum

【描述】 通过设备名获取指定 ADC 设备的设备编号 【语法】 int qm_adc_get_devnum(const char *name) 【参数】

参数名称	描述	输入/输出
const char *name	表示 ADC 设备名称的字符串	输入

【返回值】

返回值	描述
非负整数	设备编号
负数	失败

【需求】 提供有效的 ADC 设备名称; 确保设备已被正确识别和注册

【注意】需要保证在 saradc 初始化之后; 设备名称应与系统中的实际名称匹配 【举例】 无

【相关主题】 qm_adc_get_value

3.2.2 qm_adc_get_value

【描述】 获取指定 ADC 设备通道的当前值

【语法】 int qm_adc_get_value(uint32_t dev_num, uint32_t chn_num) 【参数】

参数名称	描述	输入/输出
uint32_t dev_num	设备编号	输入
uint32_t chn_num	该设备所使用的 IIO 通道	输入

【返回值】

返回值	描述

非负整数	转换值
负数	失败

【需求】 dev_num 必须是有效的 ADC 设备编号;chn_num 必须是有效的通道编号

【注意】 必须确保 ADC 设备已初始化，并且目标通道处于可用状态; 如果通道编号超出设备支持 的范围，函数可能会返回错误

【举例】 无

【相关主题】 qm_adc_get_devnum

4. EVENT

4.1 概述

按照内核 input event 标准方式，监听按键等事件

4.2 API 参考

4.2.1 qm_event_register

【描述】注册一个设备路径到事件系统，开始监听指定设备的事件 【语法】 int qm_event_register(char *dev_path)

【参数】

参数名称	描述	输入/输出
char *dev_path	设备的路径，表示你希望监听事件的 设备文件	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 设备路径必须有效且设备已连接

【注意】 在注册设备之前，请确保设备已存在并处于可用状态; 注册多个设备时，每个设备都需 要单独调用此函数

【举例】 无

【相关主题】 qm_event_unregister

4.2.2 qm_event_unregister

【描述】 取消注册一个设备路径，停止监听该设备的事件

【语法】 int qm_event_unregister(char *dev_path) 【参数】

参数名称	描述	输入/输出
char *dev_path	设备的路径，表示你希望停止监听的 设备文件	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 设备路径必须已通过 qm_event_register() 成功注册 【注意】 确保设备已经注册，否则调用此函数将无效

【举例】 无

【相关主题】 qm_event_register

4.2.3 qm_event_listen_start

【描述】 启动事件监听系统，并注册一个事件处理函数。此函数将启动事件循环，监听注册设备 的事件并触发处理函数

【语法】 int qm_event_listen_start(event_handler_t handler) 【参数】

参数名称	描述	输入/输出
event_handler_t handler	事件处理函数的指针。当监听到设备事件 时，将调用此函数来处理事件	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要先注册设备后再调用此函数; 必须提供有效的事件处理函数

【注意】不支持重复 qm_event_listen_start;qm_event_listen_start 后不支持再注册监听，但是可以注 销监听后调用 qm_event_listen_start ，触发事件回调函数接收数据; 调用此函数前，需确保所有目标 设备已通过 qm_event_register() 注册

【举例】 无

【相关主题】 qm_event_listen_stop

4.2.4 qm_event_listen_stop

【描述】 停止事件监听系统，结束事件循环 【语法】 int qm_event_listen_stop(void)

【参数】 无 【返回值】

返回值	描述
0	成功
非0	失败

【需求】 调用前必须已启动事件监听系统

【注意】 在停止事件监听之前，所有相关的事件处理和资源管理应当完成；停止事件监听后，不 会再接收任何事件，需重新启动才能恢复监听

【举例】 无

【相关主题】 qm_event_listen_start

5. PWM

5.1 概述

提供基本的 pwm 用户态接口。

5.2 API 参考

5.2.1 qm_pwm_export

【描述】将指定的 PWM （脉宽调制）设备导出，使其可以被用户空间访问和控制 【语法】 int qm_pwm_export(uint32_t pwm)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm编号	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 设备编号必须有效且设备存在

【注意】需要查看文件系统中是否有/sys/class/pwm 节点，如果没有该节点，就需要在编译内核时 勾选 Device Drivers-> Pulse-Width Modulation (PWM) Support，对应的 CONFIG 名字为 PWM;

只有成功导出设备后，才能使用其他 PWM 控制 API 【举例】 无

【相关主题】 qm_pwm_unexport

5.2.2 qm_pwm_unexport

【描述】 取消导出指定的 PWM 设备，停止用户空间对其控制 【语法】 int qm_pwm_unexport(uint32_t pwm)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm编号	输人

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 设备编号必须已通过 qm_pwm_export() 成功导出 【注意】 取消导出后，用户空间将无法再访问该 PWM 设备 【举例】 无

【相关主题】 qm_pwm_export

5.2.3 qm_pwm_set_period

【描述】 设置指定 PWM 设备的周期时间，以纳秒为单位

【语法】 int qm_pwm_set_period(uint32_t pwm,uint32_t period) 【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm编号	输人
uint32_t period	周期时长（纳秒）	输入

【返回值】

返回值	描述

0	成功
非0	失败

【需求】pwm 设备必须已导出并处于可操作状态 【注意】周期时长不超过 10 的 9 次方

【举例】 无

【相关主题】 qm_pwm_get_period

5.2.4 qm_pwm_get_period

【描述】 获取指定 PWM 设备的当前周期时间 【语法】 int qm_pwm_get_period(uint32_t pwm); 【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm编号	输入

【返回值】

返回值	描述
非负整数	周期时长（纳秒）
负数	失败

【需求】pwm 设备必须已导出

【注意】确保设备在获取时处于工作状态 【举例】 无

【相关主题】 qm_pwm_set_period

5.2.5 qm_pwm_set_duty

【描述】 设置指定 PWM 设备的占空比

【语法】 int qm_pwm_set_duty(uint32_t pwm,uint32_t duty) 【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm编号	输入
uint32_t duty	占空比时长（纳秒）	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】pwm 设备必须已导出;duty 时间不能超过当前设置的周期时间 【注意】占空比时长不超过 10 的 9 次方，且不超过当前设置的周期时长 【举例】 无

【相关主题】 qm_pwm_get_duty

5.2.6 qm_pwm_get_duty

【描述】 获取指定 PWM 设备的当前占空比时间 【语法】 int qm_pwm_get_duty(uint32_t pwm)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm编号	输人

【返回值】

返回值	描述
非负整数	占空比时长（纳秒）
负数	失败

【需求】pwm 设备必须已导出并处于工作状态

【注意】 确保设备正常运行，才能成功获取占空比 【举例】 无

【相关主题】 qm_pwm_set_duty

5.2.7 qm_pwm_set_polarity

【描述】 设置指定 PWM 设备的极性

【语法】 int qm_pwm_set_polarity(uint32_t pwm, enum pwm_polarity polarity)
【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm引脚编号	输人
enum pwm_polarity polarity	极性值	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】pwm 设备必须已导出

【注意】 需要确保设置的极性与设备的实际应用相匹配 【举例】 无

【相关主题】 qm_pwm_get_period

5.2.8 qm_pwm_get_polarity

【描述】 获取指定 PWM 设备的当前极性

【语法】 int qm_pwm_get_polarity(uint32_t pwm); 【参数】

参数名称	描述	输入/输出
uint32_t pwm	PWM 设备编号	输人

【返回值】

返回值	描述
0或1	0：正极性， 1：负极性
负	失败

【需求】pwm 设备必须已导出并运行

【注意】 确保设备运行正常才能成功获取极性 【举例】 无

【相关主题】 qm_pwm_set_polarity

5.2.9 qm_pwm_set_enable

【描述】 启用或禁用指定的 PWM 设备

【语法】 int qm_pwm_set_enable(uint32_t pwm, bool enabled) 【参数】

参数名称	描述	输入/输出
uint32_t pwm	PWM 设备编号	输人

bool enabled	是否启用 PWM 输出， true 为启用， false 为禁用	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】pwm 设备必须已导出

【注意】 确保设置后，设备的启用状态符合应用需求 【举例】 无

【相关主题】 qm_pwm_get_enable

5.2.10 qm_pwm_get_enable

【描述】 获取指定 PWM 设备的启用状态

【语法】 int qm_pwm_get_enable(uint32_t pwm) 【参数】

参数名称	描述	输入/输出
uint32_t pwm	PWM 设备编号	输人

【返回值】

返回值	描述
0或1	0 ： 禁用， 1 ： 启用
负	失败

【需求】pwm 设备必须已导出

【注意】 确保设备处于工作状态才能成功获取启用状态 【举例】 无

【相关主题】 qm_pwm_set_enable

5.2.11 qm_pwm_init

【描述】 初始化指定的 PWM 设备，包括周期、占空比和极性的设置

【语法】 int qm_pwm_init(uint32_t pwm, uint32_t period, uint32_t duty, enum pwm_polarity polarity) 【参数】

参数名称	描述	输入/输出

uint32_t pwm	PWM 设备编号	输人
uint32_t period	周期时长（纳秒）	输入
uint32_t duty	占空比时长（纳秒）	输入
enum pwm_polarity polarity	极性设置	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】pwm 设备必须已导出

【注意】 初始化完成后，设备将按照设置的参数开始工作 【举例】 无

【相关主题】 qm_pwm_deinit

5.2.12 qm_pwm_deinit

【描述】 反初始化指定的 PWM 设备，停止其工作并释放相关资源 【语法】 int qm_pwm_deinit(uint32_t pwm)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	PWM 设备编号	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】pwm 设备必须已导出并工作

【注意】 反初始化后，设备将停止工作，所有相关设置将被清除 【举例】 无

【相关主题】 qm_pwm_init

1) 数据类型

5.3.1 enum pwm_polarity 【说明】 pwm 极性

【定义】

```
enum pwm_polarity {
PWM_POLARITY_NORMAL,
PWM_POLARITY_INVERSED, };
```

【成员】

成员名称	描述
PWM_POLARITY_NORMAL	正极性
PWM_POLARITY_INVERSED	负极性

6. TIME

6.1 概述

提供硬件时间和系统时间的用户态接口

6.2 API 参考

6.2.1 qm_system_get_time

【描述】 获取 rtc 时间，并将结果存储到提供的 struct tm 结构体中 【语法】 int qm_system_get_time(struct tm *time);

【参数】

参数名称	描述	输入/输出
struct tm *time	指向存储当前系统时间的 tm 结构体指针	输出

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要确保传入的 struct tm 结构体指针有效

【注意】需要确保/dev/rtc 节点开启，如果没有开启，就需要在编译内核时勾选

Device Drivers-> Real Time Clock ，对应的 CONFIG 名字为 RTC_CLASS。

【举例】 无

【相关主题】 qm_system_set_time

6.2.2 qm_system_set_time

【描述】 设置系统时间，将提供的 tm 结构体中的时间值应用于系统中 【语法】 int qm_system_set_time(struct tm *time)

【参数】

参数名称	描述	输入/输出
struct tm *time	指向要设置的 tm 结构体指针	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要确保 struct tm 结构体中的时间有效且格式正确

【注意】需要确保/dev/rtc 节点开启，如果没有开启，就需要在编译内核时勾选 Device Drivers-> Real Time Clock ，对应的 CONFIG 名字为 RTC_CLASS。需要输入有效的时间

【举例】 无

【相关主题】 qm_system_get_time

6.2.3 qm_system_set_alarm

【描述】 获取当前系统设定的报警时间，并将结果存储在 struct tm 结构体中 【语法】 qm_system_set_alarm(struct tm *time);

【参数】

参数名称	描述	输入/输出
struct tm time	指向存储报警时间的 tm 结构体指针	输入

【返回值】

返回值	描述
0	成功

非0	失败

【需求】 需要确保 struct tm 结构体指针有效

【注意】 alarm 中断的触发时间只能是 24 小时内的一个时刻，所以只有时、分、秒的部分是有效的 ,参数 time 的年、月、日部分会被忽略。

【举例】 无

【相关主题】 qm_system_get_alarm

6.2.4 qm_system_get_alarm

【描述】 设置系统报警时间，将提供的 tm 结构体中的时间值设定为报警时间 【语法】 int qm_system_get_alarm(struct tm *time);

【参数】

参数名称	描述	输入/输出
struct tm *time	指向要设置的报警时间的 tm 结构体指针	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要确保 struct tm 结构体中提供的时间值有效

【注意】需要在 qm_system_set_alarm 成功后调用 qm_system_get_alarm 【举例】 无

【相关主题】 qm_system_set_alarm

6.2.5 qm_system_enable_alarm

【描述】 启用系统报警功能，使系统报警机制开始工作 【语法】 int qm_system_enable_alarm(void);

【参数】 无 【返回值】

返回值	描述
0	成功
非0	失败

【需求】 需要在设置报警时间后启用报警功能

【注意】需要在 qm_system_set_alarm 成功后调用 qm_system_enable_alarm ，不支持 enable 后重新设 置 alarm 的触发时间

【举例】 无

【相关主题】 qm_system_disable_alarm

6.2.6 qm_system_disable_alarm

【描述】 禁用系统报警功能，停止任何已设定的报警 【语法】 int qm_system_disable_alarm(void);

【参数】 无 【返回值】

返回值	描述
0	成功
非0	失败

【需求】 在不需要报警功能时，可以通过此 API 禁用报警

【注意】 禁用报警后，已设定的报警时间将失效，直到重新启用报警 【举例】 无

【相关主题】 qm_system_enable_alarm

6.2.7 qm_system_wait_alarm

【描述】 阻塞进程，直到设定的报警时间到达或达到指定的超时时间 【语法】 int qm_system_wait_alarm(uint32_t wait_seconds);

【参数】

参数名称	描述	输入/输出
uint32_t wait_seconds	超时时间，单位为毫秒。如果为 0	输入

,表示无限等待

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 必须设置报警时间并启用报警功能，才能有效等待报警 【注意】阻塞当前线程， 被 qm_system_enable_alarm 调用;

【举例】 无

【相关主题】 qm_system_enable_alarm

7. LED

7.1 概述

目前 led 只提供调节亮度和启停闪烁的功能

7.2 API 参考

7.2.1 qm_led_set_mode

【描述】设置指定 LED 设备的工作模式，包括是否闪烁和亮度设置

【语法】 int qm_led_set_mode(char *dev_path,boolblink, uint32_t brightness) 【参数】

参数名称	描述	输入/输出
char *dev_path	LED 设备的设备路径，指向设备 文件的字符串	输入
boolblink	设置 LED 是否闪烁， true 为闪烁 ， false 为常亮	输入
uint32_t brightness	LED 亮度值，范围通常为 0（关闭)到最大亮度值	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 LED 设备路径 (dev_path) 必须有效且指向正确的设备；亮度值 (brightness) 应在设备允许 的范围内

【注意】需要保证/sys/class/leds 设备节点存在，如果不存在，就需要在编译内核时勾选 Device Drivers-> LED Support ->LED Class Support ,对应的 CONFIG 名字为 LEDS_CLASS; 当设备不支持 闪烁功能时，设置 blink 为 true 可能无效

【举例】 无

【相关主题】无

8. WATCHDOG

8.1 概述

提供看门狗的用户态接口。

8.2 API 参考

8.2.1 qm_watchdog_start

【描述】 启动看门狗定时器，并设置超时时间 【语法】 int qm_watchdog_start(int timeval)

【参数】

参数名称	描述	输入/输出
int timeval	超时时间，单位为秒。当超时未被刷新时，系统将重启	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 看门狗必须在设备支持的情况下使用

【注意】 在超时时间内，需定期刷新看门狗，以防止系统重启 【举例】 无

【相关主题】 qm_watchdog_stop

8.2.2 qm_watchdog_refresh

【描述】 喂狗，刷新看门狗定时器，防止系统重启 【语法】 int qm_watchdog_refresh(void);

【参数】 无 【返回值】

返回值	描述
0	成功
非0	失败

【需求】 看门狗必须已启动才能刷新

【注意】 刷新应在超时时间内定期调用，以保持系统稳定 【举例】 无

【相关主题】 qm_watchdog_start

8.2.3 qm_watchdog_stop

【描述】 停止正在运行的看门狗定时器 【语法】 int qm_watchdog_stop(void);

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 看门狗必须已启动才能停止

【注意】 停止看门狗后，系统不再受到监控 【举例】 无

【相关主题】 qm_watchdog_start

9. SYSTEM

9.1 概述

提供系统操作的用户态接口。包括设备重启，休眠，关机和 chipid 以及 SN 号的获取。

9.2 API 参考

9.2.1 qm_chip_id_get

【描述】 获取芯片的唯一标识符，并将其存储在提供的字符数组中 【语法】 int qm_chip__id__get(char *chipid);

【参数】

参数名称	描述	输入/输出
char *chipid	指向字符数组的指针，用于存储 芯片 ID	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 chipid 指针必须指向足够大的字符数组以存储芯片 ID 【注意】 确保在调用此函数前， chipid 数组已正确分配内存

【举例】 无

【相关主题】 qm_serial__id__get

9.2.2 qm_vendor_write

【描述】向 vendor 写数据

【语法】 int qm_vendor_write(int vendor_id, const char *data, int size); 【参数】

参数名称	描述	输入/输出
intvendor_id	编号	输入
const char *data	待写入数据指针	输入
int size	待写入数据大小	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 确保 data 指向有效的数据并且 size 正确 【注意】

必须保证/dev/vendor_storage 存在 size 的大小建议在 1024 字节内

【举例】 无

【相关主题】 qm_vendor_read

9.2.3 qm_vendor_read

【描述】读取 vendor 的数据并存储在提供的缓冲区中。

【语法】 int qm_vendor_write(int vendor_id, const char *data, int size); 【参数】

参数名称	描述	输入/输出
intvendor_id	编号	输入
char *data	指向缓冲区的指针，用于存储读取的数据	输出
int size	读取数据大小	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 data 缓冲区必须足够大以存储读取的数据。

【注意】必须保证/dev/vendor_storage 存在 【举例】 无

【相关主题】 qm_vendor_write

9.2.4 qm_system_reboot 【描述】系统重启

【语法】 int qm_system_reboot(void); 【参数】 无

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 必须具有适当的权限才能重启系统

【注意】在调用此函数之前请确保已保存所有文件 【举例】 无

【相关主题】无

9.2.5 qm_system_shutdown 【描述】 关闭系统

【语法】 int qm_system_shutdown(void); 【参数】 无

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 必须具有适当的权限才能关闭系统

【注意】在调用此函数之前请确保已保存所有文件 【举例】 无

【相关主题】无

9.2.6 qm_system_suspend 【描述】系统休眠

【语法】 int qm_system_suspend(SUSPEND_TYPE type); 【参数】

参数名称	描述	输入/输出
SUSPEND_TYPE type	休眠类型，枚举值	输入

【返回值】

返回值	描述
0	成功
非0	失败

【需求】 休眠操作需要适当的权限

【注意】 FREEZE 与 MEM 的休眠方式需要查看系统是否支持 【举例】 无

【相关主题】无

1) 数据类型

9.3.1 SUSPEND_TYPE 【说明】 休眠类型

【定义】

```
typedef enum {
SUSPEND_FREEZE = 0,
SUSPEND_MEM,
} SUSPEND_TYPE;
```

【成员】

成员名称	描述
SUSPEND_FREEZE	FREEZE休眠方式
SUSPEND_MEM	MEM休眠方式

10. BACKLIGHT

10.1 概述

提供 gpio 基本的用户态接口,包括获取亮度，设置亮度等

10.2 API 参考

10.2.1 qm_backlight_getbri 【描述】 获取背光亮度

【语法】 int qm_backlight_getbri();

【参数】 无 【返回值】

返回值	描述
0-100	获取成功，即背光亮度值
-1	失败

【需求】头文件： qm_backlight.h 【注意】 用户可见亮度值为 0-100 【举例】 qm_backlight_test

【相关主题】 无

10.2.2 qm_backlight_setbri 【描述】 设置背光亮度

【语法】 int qm_backlight_setbri(uint32_t brightness); 【参数】

参数名称	描述	输入/输出
brightness	亮度值	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】头文件： qm_backlight.h

【注意】 用户可修改亮度值为 0-100 【举例】 qm_backlight_test

【相关主题】 无

11. HOTPLUG

11.1 概述

提供 hotplug 基本的用户态接口,包括热插拔开始 ，热插拔结束等

11.2 API 参考

11.2.1 qm_hotPlug_start

【描述】 启动热插拔监控系统，监听设备插入和移除事件。当事件发生时，会调用指定的回调函 数处理具体的逻辑。

【语法】 int qm_hotPlug_start(HotPlugCallback callback) 【参数】

参数名称	描述	输入/输出
HotPlugCallback callback	回调函数指针。当热插拔事件 发生时，调用此回调函数处理 事件	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】 确保提供一个有效的回调函数 callback

【注意】 回调函数应尽量短小高效，避免长时间阻塞热插拔线程; 多次调用 HotPlug_Start 前，应 先调用 HotPlug_Stop 停止当前运行的监控线程

【举例】 无

【相关主题】 qm_hotPlug_stop

11.2.2 qm_hotPlug_stop

【描述】 停止热插拔监控系统，终止设备插入和移除事件的监听。 【语法】 int qm_hotPlug_stop()

【参数】 无 【返回值】

返回值	描述

0	成功
非零	失败

【需求】 调用此函数来清理资源并停止监控

【注意】 调用此函数会阻塞当前线程，直到热插拔监控线程完全退出 【举例】 无

【相关主题】 qm_hotPlug_start

11.2.3 qm_hotPlugThread

【描述】 这是热插拔监控的线程函数，负责实际监听设备的插入和移除事件。它通常由 HotPlug_Start() 启动，运行在独立的线程中

【语法】 void *qm_hotPlugThread(void *arg); 【参数】

参数名称	描述	输入/输出
void *arg	传递给线程的参数，通常是指 向 HotPlug 结构体的指针	输入

【返回值】 线程返回值，通常为空

【需求】 该线程应持续监听系统的设备变化事件，并调用回调函数进行处理 【注意】 该函数通常由 HotPlug_Start() 调用并运行，不应直接从外部调用

【举例】 无

【相关主题】 qm_hotPlug_start; qm_hotPlug_stop

11.2.4 qm_extract_device_path

【描述】 从系统热插拔事件的原始缓冲区中提取设备路径。设备路径可以用来标识设备的挂载点 或设备文件名

【语法】 const char * qm_extract_device_path (const char *buffer); 【参数】

参数名称	描述	输入/输出
const char *buffer	包含设备事件信息的原始缓冲 区字符串	输入

【返回值】

返回值	描述

const char*	返回提取到的设备路径字符串。如果解析 失败，返回 NULL

【需求】buffer 必须是有效的事件缓冲区

【注意】 该函数不负责缓冲区的分配和释放，调用者应确保 buffer 在整个解析过程中有效;回的字 符串是指向 buffer 的部分内容，因此无需额外释放

【举例】

const char *eventBuffer = "ACTION=add DEVPATH=/dev/sda1 ..."; const char *devicePath = qm_extract_device_path(eventBuffer);

if (devicePath) {

printf("Device path: %s\n",devicePath); }

【相关主题】 qm_hotPlug_start; qm_hotPlugThread

11.2.5 qm_onHotPlugEvent

【描述】 处理热插拔事件，根据设备的添加或移除执行相应操作，包括记录文件状态更新和用户 界面图标更新

【语法】 void qm_onHotPlugEvent (HOTPLUG_ACTION action, const char *device)； 【参数】

参数名称	描述	输入/输出
HOTPLUG_ACTION action	表示热插拔动作的类型，可以 是添加设备或移除设备	输入
const char *device	表示被插拔的设备的名称或标 识符	输入

【返回值】 无

【需求】 在设备被添加或移除时，记录相关信息并更新用户界面状态，确保系统能够正确响应设 备变化

【注意】 确保在调用 RecordFile_NotifyAction 和 RecordFile_GetCapacity 时，recordFile 已正确初 始化；在检查容量时，应确保 m_nUsedCapacity 和 m_nTotalCapacity 的计算正确

【举例】 无

【相关主题】 qm_hotPlug_mount

11.2.6 qm_hotPlug_mount

【描述】 启动热插拔设备的监测，提示用户输入挂载目录名称，创建挂载点，并注册热插拔事件 的回调函数。程序在等待用户输入命令时保持运行，直到用户输入“quit”命令

【语法】 int qm_hotPlug_mount(RecordFile *RecordFile); 【参数】

参数名称	描述	输入/输出
RecordFile *RecordFile	指向记录文件的结构体指针，用于管理设备的挂载和卸载信 息	输入

【返回值】 无

【需求】 用户输入挂载目录名称，程序必须验证输入是否合法；创建指定的挂载点，并启动热插 拔检测，处理设备的添加和移除事件；允许用户通过输入“quit”命令退出程序，清理资源并卸载 设备

【注意】 挂载目录名称不能为空；必须在调用 create_mount_point 和 qm_hotPlug_start 函数前，确 保 RecordFile 结构体已正确初始化；在程序结束时，必须正确停止热插拔检测并卸载设备，清理 挂载点。

【举例】 无

【相关主题】 qm_hotPlug_start

11.2.7 qm_auto_mount

【描述】 自动检测以 sd 开头的设备，创建挂载点，并将第一个检测到的设备挂载到指定的挂载 点

【语法】 int qm_auto_mount(); 【参数】 无

【返回值】 无

【需求】 遍历 /dev 目录，查找以 sd 开头的设备；创建挂载点 /mnt/udisk，如果已存在则不报错； 将找到的设备挂载到指定的挂载点，并提供相应的成功或失败信息

【注意】 仅处理设备名长度为 3 的设备，例如/dev/sda ；确保具有挂载设备所需的权限；使用 system 函数执行挂载命令可能会有安全隐患，需谨慎处理

【举例】 无

【相关主题】 无

11.2.8 qm_usbHotplug

【描述】 用于注册一个 USB 热插拔处理回调函数，并启动 USB 热插拔检测功能。当检测到 USB 设备的插入或移除事件时，系统会调用用户提供的回调函数 callback，以便用户可以自定义事件 处理逻辑

【语法】 int qm_usbHotplug(HotPlugCallback callback); 【参数】

参数名称	描述	输入/输出
HotPlugCallback callback	用户定义的回调函数，用于处 理 USB 设备热插拔事件	输入

【返回值】

返回值	描述
0	成功， USB 热插拔监听已启动
负值	失败

【需求】 必须在目标设备支持 USB 热插拔功能的前提下使用此函数；调用者需要提供一个有效 的回调函数，确保能够正确处理 USB 插入和移除事件

【注意】 调用此函数前，请确保已正确初始化 USB 子系统，避免因系统未初始化而导致无法正 常调用

【举例】 无

【相关主题】 无

12. MASS_STORAGE

12.1 概述

提供 USB 大容量存储模式基本的用户态接口,包括开启 ，关闭大容量存储模式等

12.2 API 参考

12.2.1 qm_mass_storage_enable

【描述】 该函数用于配置并启用 USB Mass Storage 模式。它通过挂载 configfs，设置 USB 描述符 ,创建配置文件和目录，并将指定的存储文件路径挂载到 USB Gadget 中，从而使设备以 USB 存 储设备的形式对外可见

【语法】 int qm_mass_storage_enable(const char *file_path); 【参数】

参数名称	描述	输入/输出
const char *file_path	被挂载到 USB Mass Storage 的 文件路径	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】 设备必须 Device 模式； 内核必须启用 USB Gadget 框架和 Mass Storage 功能；configfs 文 件系统必须被正确挂载到 /sys/kernel/config

【注意】 配置完成后，主机端会识别设备为一个 USB 存储设备。根据主机系统的不同，可能需 要一些时间完成挂载；调用此函数前可能需要调用 qm_mass_storage_disable 以确保清理旧的 USB Gadget 配置；确保传入的 file_path 是有效的文件或设备路径， 并且主机系统具有对该文件的读写 权限；

【举例】 无

【相关主题】 qm_mass_storage_disable

12.2.2 qm_mass_storage_disable

【描述】 关闭 USB Mass Storage 功能模式。该函数会清理配置、删除相关目录和链接，从而完全 关闭 Mass Storage 的 USB Gadget 设置

【语法】 void qm_mass_storage_disable(void);

【参数】 无

【返回值】 无

【需求】 内核中 USB Gadget 支持和 ConfigFS 配置； 需要对 /sys/kernel/config/usb_gadget 目录具有 写权限

【注意】 该函数假设 USB Mass Storage 已正确配置并启用，若目录或链接不存在，可能导致删除 操作失败；关闭后，如果需要重新启用 Mass Storage ，需要重新调用相关的创建和启用函数。

【举例】 无

【相关主题】 qm_mass_storage_enable

12.2.3 qm_check_mass_storage_enabled

【描述】 该函数用于检查 USB 大容量存储功能是否已启用，并且检查是否连接了 TF 卡分区以及

是否已通过 PC 进行 USB 枚举。 通过 USB 插入 PC 的情况下 ，让用户判断是否已经插入 PC 【语法】 int qm_check_mass_storage_enabled();

【参数】 无 【返回值】

返回值	描述
0	失败
-1	失败，异常错误
1	成功， USB 已连接并被 PC 枚举

【需求】 系统必须支持 USB Gadget 驱动，并且配置了大容量存储功能；设备已连接 TF 卡，并且 已挂载在 /dev/mmcblk0p1

【注意】 该函数检查的是 USB Gadget 配置和 TF 卡分区挂载情况，依赖于 /sys/class/udc/ 目录和 设备路径 /dev/mmcblk0p1； 在未正确配置或挂载时，函数将返回 0 或 -1 ；如果 /sys/class/udc/ 中 的 UDC 文件为空或不存在，则表明 USB Gadget 未启用或未连接到 PC

【举例】 无

【相关主题】 qm_mass_storage_enable

STM motor

1) 概述

提供 STM motor 控制接口

2) API 参考

13.1 int qm_motor_run_control(int idx, int direction, int step, int speed);

/**

* @brief 控制电机的运行，设置电机的索引、方向、步数和速度。

*

* 该函数通过一系列内部调用完成对电机的具体操作，并最终返回一个状态码。

*

* @param idx 电机的索引号，用于标识不同的电机。

* @param direction 电机的旋转方向，0 表示正转，非 0 表示反转。

* @param step 64 step 一圈，电机的步数。

* @param speed 1,1.5,表示相对关系

*

* @return int 函数执行后的返回状态码，通常用于表示操作是否成功。

*/

13.2 int qm_motor_manualStop(int idx);

停止对应马达运行 for QM10XV

0: for stop Horizon STM motor

1: for stop Vertial STM motor