

ECx00U&EGx00U 系列

QuecOpen GPIO API 参考手册

LTE Standard 模块系列

版本：1.0

日期：2021-05-26

状态：受控文件



上海移远通信技术股份有限公司始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司

上海市闵行区田林路 1016 号科技绿洲 3 期（B 区）5 号楼 邮编：200233

电话：+86 21 51086236 邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：<http://www.quectel.com/cn/support/sales.htm>。

如需技术支持或反馈我司技术文档中的问题，可随时登陆如下网址：

<http://www.quectel.com/cn/support/technical.htm> 或发送邮件至：support@quectel.com。

前言

上海移远通信技术股份有限公司提供该文档内容用以支持其客户的产品设计。客户须按照文档中提供的规范、参数来设计其产品。因未能遵守有关操作或设计规范而造成的损害，上海移远通信技术股份有限公司不承担任何责任。在未声明前，上海移远通信技术股份有限公司有权对该文档进行更新。

免责声明

上海移远通信技术股份有限公司尽力确保开发中功能的完整性、准确性、及时性或效用，但不排除上述功能错误或遗漏的可能。除非其他有效协议另有规定，否则上海移远通信技术股份有限公司对开发中功能的使用不做任何暗示或明示的保证。在适用法律允许的最大范围内，上海移远通信技术股份有限公司不对任何因使用开发中功能而遭受的损失或损害承担责任，无论此类损失或损害是否可以预见。

保密义务

除非上海移远通信技术股份有限公司特别授权，否则我司所提供文档和信息的接收方须对接收的文档和信息保密，不得将其用于除本项目的实施与开展以外的任何其他目的。未经上海移远通信技术股份有限公司书面同意，不得获取、使用或向第三方泄露我司所提供的文档和信息。对于任何违反保密义务、未经授权使用或以其他非法形式恶意使用所述文档和信息的违法侵权行为，上海移远通信技术股份有限公司有权追究法律责任。

版权申明

本文档版权属于上海移远通信技术股份有限公司，任何人未经我司允许而复制转载该文档将承担法律责任。

版权所有 ©上海移远通信技术股份有限公司 2021，保留一切权利。

Copyright © Quectel Wireless Solutions Co., Ltd. 2021.

文档历史

修订记录

版本	日期	作者	变更表述
-	2020-12-01	JoJo YAN	文档创建
1.0	2021-05-26	JoJo YAN	受控版本

目录

文档历史	2
目录	3
表格索引	5
1 引言	6
1.1. 适用模块	6
2 GPIO 数据结构及 API 介绍	7
2.1. 头文件	7
2.2. 函数概览	7
2.3. GPIO 相关 API	8
2.3.1. ql_gpio_init	8
2.3.1.1. ql_GpioNum	8
2.3.1.2. ql_GpioDir	9
2.3.1.3. ql_PullMode	10
2.3.1.4. ql_LvlMode	10
2.3.1.5. ql_errcode_gpio	11
2.3.2. ql_gpio_deinit	11
2.3.3. ql_gpio_set_level	12
2.3.4. ql_gpio_get_level	12
2.3.5. ql_gpio_set_direction	13
2.3.6. ql_gpio_get_direction	13
2.3.7. ql_gpio_set_pull	14
2.3.8. ql_gpio_get_pull	14
2.3.9. ql_int_register	15
2.3.9.1. ql_TriggerMode	16
2.3.9.2. ql_DebounceMode	16
2.3.9.3. ql_EdgeMode	16
2.3.10. ql_int_unregister	17
2.3.11. ql_int_enable	17
2.3.12. ql_int_disable	18
2.3.13. ql_pin_set_func	18
2.3.14. ql_pin_get_func	19
2.3.15. ql_pin_set_gpio	19
3 GPIO 的配置方法	20
3.1. GPIO 统一配置	20
3.2. GPIO 单独配置	20
4 GPIO 开发示例	22
4.1. APP 侧的 GPIO 开发示例	22
4.1.1. GPIO Demo 说明	22
4.2. GPIO 相关调试	23
4.2.1. 调试准备	23

4.2.2.	GPIO 功能调试	24
4.2.3.	GPIO 中断调试	24
5	附录 参考文档和术语缩写	26

表格索引

表 1: 适用模块 6

表 2: 函数概览 7

表 3: 参考文档 26

表 4: 术语缩写 26

1 引言

移远通信 LTE Standard ECx00U 系列和 EGx00U 模块支持 QuecOpen®方案；QuecOpen 是开源的基于 RTOS 的嵌入式开发平台，可简化 IoT 应用的软件设计和开发过程。有关 QuecOpen®的详细信息，请参考 [文档\[1\]](#)。

本文档主要介绍在 QuecOpen®方案下，移远通信 ECx00U 系列和 EGx00U 模块的 APP 侧 GPIO 的开发流程，包括数据结构、相关 API 和调试方法。

1.1. 适用模块

表 1：适用模块

模块系列	模块
ECx00U	EC200U 系列
	EC600U 系列
EGx00U	EG500U-CN
	EG700U-CN

备注

本文档中的 GPIO 引脚编号为模块内嵌芯片的 GPIO 编号，而非 ECx00U 系列和 EGx00U QuecOpen®模块的引脚编号，对应的 GPIO 关系请参考[文档 \[2\]、\[3\]、\[4\]和\[5\]](#)。

2 GPIO 数据结构及 API 介绍

2.1. 头文件

`ql_gpio.h` 文件即为 GPIO API 的头文件，位于 `components\ql-kernel\inc` 目录下；若无特别说明，本文档所提及的头文件均位于该目录下。

2.2. 函数概览

表 2：函数概览

函数	说明
<code>ql_gpio_init()</code>	对指定的 GPIO 进行初始化配置
<code>ql_gpio_deinit()</code>	注销已经配置的 GPIO
<code>ql_gpio_set_level()</code>	设置指定 GPIO 的引脚电平
<code>ql_gpio_get_level()</code>	获取指定 GPIO 的引脚电平
<code>ql_gpio_set_direction()</code>	设置 GPIO 引脚输入/输出方向
<code>ql_gpio_get_direction()</code>	获取指定 GPIO 的引脚输入/输出方向
<code>ql_gpio_set_pull()</code>	配置指定 GPIO 的上下拉设置
<code>ql_gpio_get_pull()</code>	获取指定 GPIO 的上下拉属性
<code>ql_int_register()</code>	配置指定的 GPIO 的中断触发方式
<code>ql_int_unregister()</code>	注销 GPIO 的中断回调函数
<code>ql_int_enable()</code>	启用所配置的 GPIO 中断
<code>ql_int_disable()</code>	禁用所配置的 GPIO 中断

<code>ql_pin_set_func()</code>	配置引脚的功能
<code>ql_pin_get_func()</code>	获取引脚的功能
<code>ql_pin_set_gpio()</code>	直接配置引脚的 GPIO 功能

2.3. GPIO 相关 API

2.3.1. ql_gpio_init

该函数用于对指定的 GPIO 进行初始化配置，包括引脚输入/输出方向、上下拉、输出引脚默认电平值。在初次设置或重新设置 GPIO 时都可以调用该函数。

- 函数原型

```
ql_errcode_gpio ql_gpio_init(ql_GpioNum gpio_num, ql_GpioDir gpio_dir, ql_PullMode gpio_pull, ql_LvlMode gpio_lvl)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 `ql_GpioNum`。

gpio_dir:

[In] 引脚输入/输出方向设置。请参考第 2.3.1.2 章枚举 `ql_GpioDir`。

gpio_pull:

[In] 引脚上下拉设置。仅对输入引脚有效。请参考第 2.3.1.3 章枚举 `ql_PullMode`。

gpio_lvl:

[In] 引脚电平设置。仅对输出引脚有效。请参考第 2.3.1.4 章枚举 `ql_LvlMode`。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 `ql_errcode_gpio`。

2.3.1.1. ql_GpioNum

GPIO 引脚编号枚举信息定义如下：

```
typedef enum
{
    GPIO_0 = 0,
```

```
GPIO_1,
GPIO_2,
GPIO_3,
GPIO_4,
GPIO_5,
.....
GPIO_28,
GPIO_29,
GPIO_30,
GPIO_31,
GPIO_MAX
}ql_GpioNum;
```

● 参数

参数	描述
GPIO_0	GPIO 引脚编号 0
GPIO_1	GPIO 引脚编号 1
.....
GPIO_31	GPIO 引脚编号 31

备注

GPIO 引脚编号 6 和 16 暂不支持。

2.3.1.2. ql_GpioDir

引脚输入/输出方向枚举信息定义如下：

```
typedef enum
{
    GPIO_INPUT,
    GPIO_OUTPUT
}ql_GpioDir;
```

● 参数

参数	描述
GPIO_INPUT	输入

GPIO_OUTPUT

输出

2.3.1.3. ql_PullMode

引脚上下拉枚举信息定义如下，仅对输入引脚有效：

```
typedef enum
{
    PULL_NONE,
    PULL_DOWN,
    PULL_UP
}ql_PullMode;
```

● 参数

参数	描述
<i>PULL_NONE</i>	悬空
<i>PULL_DOWN</i>	下拉
<i>PULL_UP</i>	上拉

2.3.1.4. ql_LvlMode

引脚电平枚举信息定义如下，仅对输出引脚有效：

```
typedef enum
{
    LVL_LOW,
    LVL_HIGH
}ql_LvlMode;
```

● 参数

参数	描述
<i>LVL_LOW</i>	低电平
<i>LVL_HIGH</i>	高电平

2.3.1.5. ql_errcode_gpio

GPIO 错误码表示函数是否执行成功，若失败则返回错误原因，枚举信息定义如下：

```
typedef enum
{
    QL_GPIO_SUCCESS = QL_SUCCESS,
    QL_GPIO_EXECUTE_ERR    = 1|QL_GPIO_ERRCODE_BASE,
    QL_GPIO_INVALID_PARAM_ERR,
    QL_GPIO_OPEN_ERR,
    QL_GPIO_CONFIG_ERR,
    QL_GPIO_CALLBACK_ERR,
    QL_GPIO_LEVEL_TRIGGER_ERR
    QL_GPIO_MEM_NULL_ERR
}ql_errcode_gpio;
```

● 参数

参数	描述
QL_GPIO_SUCCESS	函数执行成功
QL_GPIO_EXECUTE_ERR	函数执行失败
QL_GPIO_INVALID_PARAM_ERR	函数的参数为错误值
QL_GPIO_OPEN_ERR	GPIO 打开失败
QL_GPIO_CONFIG_ERR	GPIO 参数设置失败
QL_GPIO_CALLBACK_ERR	GPIO 设置中断回调函数失败
QL_GPIO_LEVEL_TRIGGER_ERR	无效的电平触发中断方式
QL_GPIO_MEM_NULL_ERR	指针参数为 NULL

2.3.2. ql_gpio_deinit

该函数用于注销已经配置的 GPIO。

● 函数原型

```
ql_errcode_gpio ql_gpio_deinit(ql_GpioNum gpio_num)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

2.3.3. ql_gpio_set_level

该函数用于设置指定 GPIO 的引脚电平，仅对输出引脚有效。

- 函数原型

```
ql_errcode_gpio ql_gpio_set_level(ql_GpioNum gpio_num, ql_LvlMode gpio_lvl)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

gpio_lvl:

[In] 引脚电平设置。仅对输出引脚有效。请参考第 2.3.1.4 章枚举 *ql_LvlMode*。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

2.3.4. ql_gpio_get_level

该函数用于获取指定 GPIO 的引脚电平。

- 函数原型

```
ql_errcode_gpio ql_gpio_get_level(ql_GpioNum gpio_num, ql_LvlMode *gpio_lvl)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

gpio_level:

[Out] 获取到的引脚电平值。

LVL_LOW 低电平

LVL_HIGH 高电平

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 `ql_errcode_gpio`。

2.3.5. ql_gpio_set_direction

该函数用于设置 GPIO 引脚输入/输出方向。

- 函数原型

```
ql_errcode_gpio ql_gpio_set_direction(ql_GpioNum gpio_num, ql_GpioDir gpio_dir)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 `ql_GpioNum`。

gpio_dir:

[In] 引脚输入/输出方向设置。请参考第 2.3.1.2 章枚举 `ql_GpioDir`。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 `ql_errcode_gpio`。

2.3.6. ql_gpio_get_direction

该函数用于获取指定 GPIO 的引脚输入/输出方向。

- 函数原型

```
ql_errcode_gpio ql_gpio_get_direction(ql_GpioNum gpio_num, ql_GpioDir *gpio_dir)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 `ql_GpioNum`。

gpio_dir:

[Out] 获取到的引脚输入/输出方向。请参考第 2.3.1.2 章枚举 `ql_GpioDir`。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 `ql_errcode_gpio`。

2.3.7. ql_gpio_set_pull

该函数用于配置指定 GPIO 的上下拉设置，仅对输入引脚有效。

- 函数原型

```
ql_errcode_gpio ql_gpio_set_pull(ql_GpioNum gpio_num, ql_PullMode gpio_pull)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

gpio_pull:

[In] 引脚上下拉设置。仅对输入引脚有效。请参考第 2.3.1.3 章枚举 *ql_PullMode*。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

2.3.8. ql_gpio_get_pull

该函数用于获取指定 GPIO 的上下拉属性。

- 函数原型

```
ql_errcode_gpio ql_gpio_get_pull(ql_GpioNum gpio_num, ql_PullMode *gpio_pull)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

gpio_pull:

[Out] 获取到的 GPIO 引脚上下拉属性。

PULL_NONE 悬空

PULL_DOWN 下拉

PULL_UP 上拉

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

2.3.9. ql_int_register

该函数用于配置指定的 GPIO 的中断触发方式，仅对输入引脚有效。在初次设置或重新设置 GPIO 时都可以调用该函数。调用该函数后，需要调用 *ql_int_enable* 才能使能中断回调函数。

● 函数原型

```
ql_errcode_gpio ql_int_register(ql_GpioNum gpio_num,
                                ql_TriggerMode gpio_trigger, ql_DebounceMode gpio_debounce,
                                ql_EdgeMode gpio_edge, ql_PullMode gpio_pull,
                                void *int_cb, void *cb_ctx)
```

● 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

gpio_trigger:

[In] 中断触发模式。请参考第 2.3.9.1 章枚举 *ql_TriggerMode*。

gpio_debounce:

[In] 中断硬件消抖使能。请参考第 2.3.9.2 章枚举 *ql_DebounceMode*。

gpio_edge:

[In] 中断触发方式。请参考第 2.3.9.3 章枚举 *ql_EdgeMode*。

gpio_pull:

[In] 引脚上下拉设置。仅对输入引脚有效。请参考第 2.3.1.3 章枚举 *ql_PullMode*。

int_cb:

[In] 中断回调函数。正常触发中断会调用此函数。

cb_ctx:

[In] 中断回调函数参数。调用中断回调函数时会使用该参数。

● 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

2.3.9.1. ql_TriggerMode

中断触发仅对输入引脚有效，中断触发模式枚举信息定义如下：

```
typedef enum
{
    EDGE_TRIGGER,
    LEVEL_TRIGGER
}ql_TriggerMode;
```

● 参数

参数	描述
<i>EDGE_TRIGGER</i>	边沿中断触发模式
<i>LEVEL_TRIGGER</i>	电平中断触发模式

2.3.9.2. ql_DebounceMode

中断触发仅对输入引脚有效，硬件消抖使能枚举信息定义如下：

```
typedef enum
{
    DEBOUNCE_DIS,
    DEBOUNCE_EN
}ql_DebounceMode;
```

● 参数

参数	描述
<i>DEBOUNCE_DIS</i>	硬件消抖无效
<i>DEBOUNCE_EN</i>	硬件消抖有效

2.3.9.3. ql_EdgeMode

中断触发仅对输入引脚有效，中断触发方式枚举信息定义如下：

```
typedef enum
{
    EDGE_RISING,
    EDGE_FALLING,
```

```
EDGE_BOTH
}ql_EdgeMode;
```

- 参数

参数	描述
<i>EDGE_RISING</i>	边沿触发：上升沿触发中断；电平触发：高电平触发中断
<i>EDGE_FALLING</i>	边沿触发：下降沿触发中断；电平触发：低电平触发中断
<i>EDGE_BOTH</i>	双边沿触发中断

2.3.10. ql_int_unregister

该函数用于注销 GPIO 的中断回调函数。调用该函数后，需要重新注册中断回调函数。

- 函数原型

```
ql_errcode_gpio ql_int_unregister(ql_GpioNum gpio_num)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

2.3.11. ql_int_enable

该函数用于启用所配置的 GPIO 中断。若未使用 *ql_int_unregister* 注销回调函数，则无需重新配置中断，可以再次调用该接口来使能中断，该函数需要跟 *ql_int_register()* 配套使用才能有效中断回调函数。

- 函数原型

```
ql_errcode_gpio ql_int_enable(ql_GpioNum gpio_num)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 *ql_GpioNum*。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 `ql_errcode_gpio`。

2.3.12. ql_int_disable

该函数用于禁用所配置的 GPIO 中断。调用 `ql_int_enable` 即可再次使能 GPIO 中断回调。

- 函数原型

```
ql_errcode_gpio ql_int_disable(ql_GpioNum gpio_num)
```

- 参数

gpio_num:

[In] GPIO 引脚编号。请参考第 2.3.1.1 章枚举 `ql_GpioNum`。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 `ql_errcode_gpio`。

2.3.13. ql_pin_set_func

该函数用于配置引脚的功能。

- 函数原型

```
ql_errcode_gpio ql_pin_set_func(uint8_t pin_num, uint8_t func_sel)
```

- 参数

pin_num:

[In] 引脚编号。

func_sel:

[In] 引脚功能编号。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 `ql_errcode_gpio`。

2.3.14. ql_pin_get_func

该函数用于获取引脚的功能。

- 函数原型

```
ql_errcode_gpio ql_pin_get_func(uint8_t pin_num, uint8_t *func_sel)
```

- 参数

pin_num:

[In] 引脚编号。

func_sel:

[Out] 引脚功能编号。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

2.3.15. ql_pin_set_gpio

该函数用于直接配置引脚的 GPIO 功能。

- 函数原型

```
ql_errcode_gpio ql_pin_set_gpio(uint8_t pin_num)
```

- 参数

pin_num:

[In] 引脚编号。

- 返回值

返回 GPIO 错误码。请参考第 2.3.1.5 章枚举 *ql_errcode_gpio*。

3 GPIO 的配置方法

本章节主要介绍如何使用上述 API 对 GPIO 进行配置。

3.1. GPIO 统一配置

1. 建立 GPIO 的参数配置表，如下图所示。

```
static ql_gpio_cfg_t ql_gpio_cfg[] =
{
    /* gpio_num gpio_dir gpio_pull gpio_lvl */
    { GPIO_0, GPIO_INPUT, PULL_DOWN, 0xff }, /* set input pull-down */
    { GPIO_1, GPIO_OUTPUT, 0xff, LVL_HIGH } /* set output high-level */
};
```

2. 使用类似 `ql_gpio_demo_init()` 功能的函数初始化可以快速配置多引脚的参数，使用方法如下图所示。

```
void ql_gpio_demo_init(void)
{
    uint16_t num;
    for (num = 0; num < sizeof(_ql_gpio_cfg)/sizeof(_ql_gpio_cfg[0]); num++)
    {
        ql_gpio_deinit(_ql_gpio_cfg[num].gpio_num);
        ql_gpio_init(_ql_gpio_cfg[num].gpio_num, _ql_gpio_cfg[num].gpio_dir, _ql_gpio_cfg[num].gpio_pull, _ql_gpio_cfg[num].gpio_lvl);
    }
}
```

3.2. GPIO 单独配置

1. 设置 GPIO 为输入引脚，需要设置 GPIO 引脚编号、引脚输入/输出方向和上下拉，如下图所示。

```
/* set input pull-up */
ql_gpio_set_direction(GPIO_0, GPIO_INPUT);
ql_gpio_set_pull(GPIO_0, PULL_UP);
```

2. 设置 GPIO 为输出引脚，需要设置 GPIO 引脚编号、引脚输入/输出方向和输出电平，如下图所示。

```

/*set output high*/
ql_gpio_set_direction(GPIO_0, GPIO_OUTPUT);
ql_gpio_set_level(GPIO_0, LVL_HIGH);

```

3. 设置 GPIO 为中断触发，需要设置 GPIO 引脚编号、触发模式、消抖使能、触发方式、上下拉和中断回调函数，如下图所示。

```

ql_pin_set_func(QL_TEST1_PIN_GPIO0, QL_TEST1_PIN_GPIO0_FUNC_GPIO); .....//set GPIO0
ql_pin_set_func(QL_TEST1_PIN_GPIO1, QL_TEST1_PIN_GPIO1_FUNC_GPIO); .....//set GPIO1

//ql_gpio_deinit(GPIO_0);|
//ql_gpio_deinit(GPIO_1);

ql_int_register(GPIO_0, EDGE_TRIGGER, DEBOUNCE_EN, EDGE_RISING, PULL_DOWN, _gpioint_callback01, NULL);
ql_int_register(GPIO_1, EDGE_TRIGGER, DEBOUNCE_EN, EDGE_FALLING, PULL_UP, _gpioint_callback02, NULL);

ql_int_enable(GPIO_0);
ql_int_enable(GPIO_1);

```

备注

该配置需调用 `ql_int_enable()` 使能中断。

4 GPIO 开发示例

本章节主要介绍在 APP 侧如何使用上述 API 进行 GPIO 开发以及简单的调试。GPIO 示例和 GPIO 中断示例中默认使用 GPIO0 和 GPIO1 进行测试。

4.1. APP 侧的 GPIO 开发示例

4.1.1. GPIO Demo 说明

ECx00U 系列和 EGx00U QuecOpen SDK 代码中提供了 GPIO 的示例，即 `gpio_demo.c` 文件和 `gpio_int_demo.c` 文件。

- `gpio_demo.c` 文件中主要包含了对 GPIO 的基本设置，包括引脚方向、引脚电平、上下拉等设置。入口函数为 `ql_gpio_app_init()`，如下图所示。

例如，引脚 a 和引脚 b 都可以复用为 GPIO0，可以选择将其中一个配置为 GPIO0，但不能同时将这两个引脚都配置为 GPIO0。该 Demo 以 3 秒一次的频率改变 GPIO 的配置：先将引脚 a 复用为 GPIO 并完成 GPIO 的基本设置，完成后切换至引脚 b，将其复用为 GPIO 并完成配置，后重复执行以上的操作。

```
void ql_gpio_app_init(void)
{
    ...QIosStatus.err = QL_OSI_SUCCESS;
    ...ql_task_t gpio_task = NULL;

    ...err = ql_rtos_task_create(&gpio_task, 1024, APP_PRIORITY_NORMAL, "ql_gpiodemo", ql_gpio_demo_thread, NULL, 1);
    ...if (err != QL_OSI_SUCCESS)
    ...{
    ...    ...QL_GPIOIDEMO_LOG("gpio_demo_task_created_failed");
    ...}
}
```

- `gpio_int_demo.c` 文件主要包含了对 GPIO 中断的基本设置，包括中断回调函数、中断触发方式等。入口函数为 `ql_gpioint_app_init()`，如下图所示。

该 Demo 开始时，使用 GPIO0 和 GPIO1 都可以触发中断，之后每 20 秒只有一个 GPIO 可以触发中断，GPIO0 和 GPIO1 循环有效。

```
void ql_gpioint_app_init(void)
{
    ...QIosStatus.err = QL_OSI_SUCCESS;
    ...ql_task_t gpioint_task = NULL;

    ...err = ql_rtos_task_create(&gpioint_task, 1024, APP_PRIORITY_NORMAL, "ql_gpiointdemo", ql_gpioint_demo_thread, NULL, 1);
    ...if (err != QL_OSI_SUCCESS)
    ...{
    ...    ...QL_GPIOINTDEMO_LOG("gpio_int_demo_task_created_failed");
    ...}
}
```

备注

上述两个 Demo 已在 `ql_init_demo_thread` 线程中默认不启动，如下所示。

```
static void ql_init_demo_thread(void *param)
{
    ...QL_INIT_LOG("init demo thread enter, param: 0x%x", param);
    #if 0
    ...ql_gpio_app_init();
    ...ql_gpioint_app_init();
    #endif

    #ifdef QL_APP_FEATURE_LEDCFG
    ...ql_ledcfg_app_init();
    #endif









    #ifdef QL_APP_FEATURE_AUDIO
    ...//ql_audio_app_init();
    #endif
    #ifdef QL_APP_FEATURE_LCD
    ...//ql_lcd_app_init();
    #endif
    ...//ql_nw_app_init();
    ...//ql_datacall_app_init();
    ...//ql_osi_demo_init();
}
```

4.2. GPIO 相关调试

4.2.1. 调试准备

ECx00U 系列和 EGx00U QuecOpen 模块可通过使用移远通信 LTE OPEN EVB 板进行 GPIO 功能调试。

编译版本烧录到模块中，使用 USB 线连接 LTE OPEN EVB 的 USB 端口和 PC，USB 的 AP Log 端口主要用于显示系统调试信息，通过 *cooltools* 可以查看到上述两个 Demo 的相关信息，log 的抓取方法请参考文档 [6]。

-  Qectel Modem (COM8)
-  Qectel USB AP Log Port (COM14)
-  Qectel USB AT Port (COM5)
-  Qectel USB CP Log Port (COM11)
-  Qectel USB Diag Port (COM12)
-  Qectel USB MOS Port (COM13)
-  Qectel USB Serial-1 Port (COM9)
-  Qectel USB Serial-2 Port (COM10)

4.2.2. GPIO 功能调试

开机后自动启动 `ql_gpio_app_init()`，通过 log 信息可以看到 GPIO0 和 GPIO1 被循环设置为输出、输入，且引脚电平值循环改变。也可以使用万用表或者示波器测量对应的引脚，查看设置是否正确。

通过调试串口打印出的 log 如下图所示：

	Description
[ql_GPIODEMO][ql_gpio_demo_thread, 189]	GPIO0/1 pin is TEST1 -> TEST2
[ql_GPIODEMO][ql_gpio_demo_thread, 103]	gpio[0] output low-level
[ql_GPIODEMO][ql_gpio_demo_thread, 104]	gpio[0] set dir:[1], lvl:[0]
[ql_GPIODEMO][ql_gpio_demo_thread, 110]	gpio[0] output low-level
[ql_GPIODEMO][ql_gpio_demo_thread, 111]	gpio[0] get dir:[1], lvl:[0]
[ql_GPIODEMO][ql_gpio_demo_thread, 103]	gpio[1] output low-level
[ql_GPIODEMO][ql_gpio_demo_thread, 104]	gpio[1] set dir:[1], lvl:[0]
[ql_GPIODEMO][ql_gpio_demo_thread, 110]	gpio[1] output low-level
[ql_GPIODEMO][ql_gpio_demo_thread, 111]	gpio[1] get dir:[1], lvl:[0]
[ql_GPIODEMO][ql_gpio_demo_thread, 124]	gpio[0] output high-level
[ql_GPIODEMO][ql_gpio_demo_thread, 125]	gpio[0] set dir:[1], lvl:[1]
[ql_GPIODEMO][ql_gpio_demo_thread, 131]	gpio[0] output high-level
[ql_GPIODEMO][ql_gpio_demo_thread, 132]	gpio[0] get dir:[1], lvl:[1]
[ql_GPIODEMO][ql_gpio_demo_thread, 124]	gpio[1] output high-level
[ql_GPIODEMO][ql_gpio_demo_thread, 125]	gpio[1] set dir:[1], lvl:[1]
[ql_GPIODEMO][ql_gpio_demo_thread, 131]	gpio[1] output high-level
[ql_GPIODEMO][ql_gpio_demo_thread, 132]	gpio[1] get dir:[1], lvl:[1]
[ql_GPIODEMO][ql_gpio_demo_thread, 145]	gpio[0] input pull-down
[ql_GPIODEMO][ql_gpio_demo_thread, 146]	gpio[0] set dir:[0], pull:[1]
[ql_GPIODEMO][ql_gpio_demo_thread, 153]	gpio[0] input pull-down
[ql_GPIODEMO][ql_gpio_demo_thread, 154]	gpio[0] get dir:[0], pull:[1], lvl:[0]
[ql_GPIODEMO][ql_gpio_demo_thread, 145]	gpio[1] input pull-down
[ql_GPIODEMO][ql_gpio_demo_thread, 146]	gpio[1] set dir:[0], pull:[1]
[ql_GPIODEMO][ql_gpio_demo_thread, 153]	gpio[1] input pull-down
[ql_GPIODEMO][ql_gpio_demo_thread, 154]	gpio[1] get dir:[0], pull:[1], lvl:[0]
[ql_GPIODEMO][ql_gpio_demo_thread, 167]	gpio[0] input pull-up

4.2.3. GPIO 中断调试

开机后自动启动 `ql_gpioint_app_init()`，GPIO0 和 GPIO1 被设置为中断。请手动触发对应的引脚（连接至 1.8 V 电源或者地线），即可看到中断回调被触发，并打印相关日志。前 20 秒两个 GPIO 都可以触发中断，之后每 20 秒只能使用一个 GPIO 触发中断。

通过调试串口打印出的 log 如下图所示：

```
GPIO int status 0x20000000
GPIO int status 0x20000000
GPIO int status 0x20000000
GPIO int status 0x20000000
GPIO int status 0x20000000
GPIO int status 0x20000000
GPIO int status 0x40000000
GPIO int status 0x40000000
GPIO int status 0x40000000
GPIO int status 0x40000000
GPIO int status 0x40000000
GPIO int status 0x40000000
GPIO int status 0x40000000
GPIO int status 0x40000000
```

备注

1. 因为上述两个 Demo 默认关闭，需要测试时请打开预处理。
2. 因为其他 Demo 中也可能会配置本 Demo 中的引脚，故仅使用本 Demo 进行测试。

5 附录 参考文档和术语缩写

表 3：参考文档

序号	文档名称	描述
[1]	Quectel_ECx00U&EGx00U 系列_QuecOpen_快速开发指导	EC200U 系列、EC600U 系列、EG500U-CN、EG700U-CN QuecOpen 模块快速开发指导
[2]	Quectel_EC200U 系列_QuecOpen_GPIO 配置	EC200U 系列 QuecOpen 模块 GPIO 配置
[3]	Quectel_EC600U 系列_QuecOpen_GPIO 配置	EC600U 系列 QuecOpen 模块 GPIO 配置
[4]	Quectel_EG500U-CN_QuecOpen_GPIO 配置	EG500U-CN QuecOpen 模块 GPIO 配置
[5]	Quectel_EG700U-CN_QuecOpen_GPIO 配置	EG700U-CN QuecOpen 模块 GPIO 配置
[6]	Quectel_ECx00U&EGx00U 系列_QuecOpen_Log 抓取操作指导	EC200U 系列、EC600U 系列、EG500U-CN、EG700U-CN QuecOpen 模块 log 抓取操作指导

表 4：术语缩写

缩写	英文全称	中文全称
API	Application Programming Interface	应用程序接口
AP	Application Processor	应用处理器
APP	Application	应用
EVB	Evaluation Board	评估板
GPIO	General-Purpose Input/Output	通用型输入/输出
IoT	Internet of Things	物联网
RTOS	Real-Time Operating System	实时操作系统
PC	Personal Computer	个人计算机
PSM	Power Saving Mode	省电模式

SDK	Software Development Kit	软件开发工具包
USB	Universal Serial Bus	通用串行总线
UART	Universal Asynchronous Receiver/Transmitter	通用异步收发传输器