BC660K-GL QuecOpen API Reference Manual

NB-IoT Module Series

Version: 1.0

Date: 2021-06-02

Status: Released

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://www.quectel.com/support/sales.htm.

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/technical.htm
Or email to support@quectel.com.

General Notes

Quectel offers the information as a service to its customers. The information provided is based upon customers' requirements. Quectel makes every effort to ensure the quality of the information it makes available. Quectel does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information. All information supplied herein is subject to change without prior notice.

Disclaimer

While Quectel has made efforts to ensure that the functions and features under development are free from errors, it is possible that these functions and features could contain errors, inaccuracies and omissions. Unless otherwise provided by valid agreement, Quectel makes no warranties of any kind, implied or express, with respect to the use of features and functions under development. To the maximum extent permitted by law, Quectel excludes all liability for any loss or damage suffered in connection with the use of the functions and features under development, regardless of whether such loss or damage may have been foreseeable.

Duty of Confidentiality

The Receiving Party shall keep confidential all documentation and information provided by Quectel, except when the specific permission has been granted by Quectel. The Receiving Party shall not access or use Quectel's documentation and information for any purpose except as expressly provided herein. Furthermore, the Receiving Party shall not disclose any of the Quectel's documentation and information to any third party without the prior written consent by Quectel. For any noncompliance to the above requirements, unauthorized use, or other illegal or malicious use of the documentation and information, Quectel will reserve the right to take legal action.

Copyright

About the Document

Revision History

Version	Date	Author	Description
-	2021-06-02	Dakai XU	Creation of the document
1.0	2021-06-02	Dakai XU	First official release

Contents

Table Index

1 Introduction

Quectel BC660K-GL module support QuecOpen® solution. QuecOpen® is an open-source embedded development platform based on FreeRTOS system. It is intended to simplify the design and development of IoT applications. For more information on QuecOpen®, see ***document [1]***.

The QuecOpen® solution embeds the application program directly into the Quectel communication module and can run without an external MCU. At present, it has been widely used in M2M fields, such as smart home, smart city, tracker & tracing, automotive, and smart energy, etc.

This document mainly introduces the QuecOpen® solution supported by Quectel's BC660K-GL NB-IoT module, such as system architecture, data types, system configuration, and the use of related API functions.

2 QuecOpen® Platform

2.1. System Architecture

The following figure shows the fundamental principle of QuecOpen software architecture.

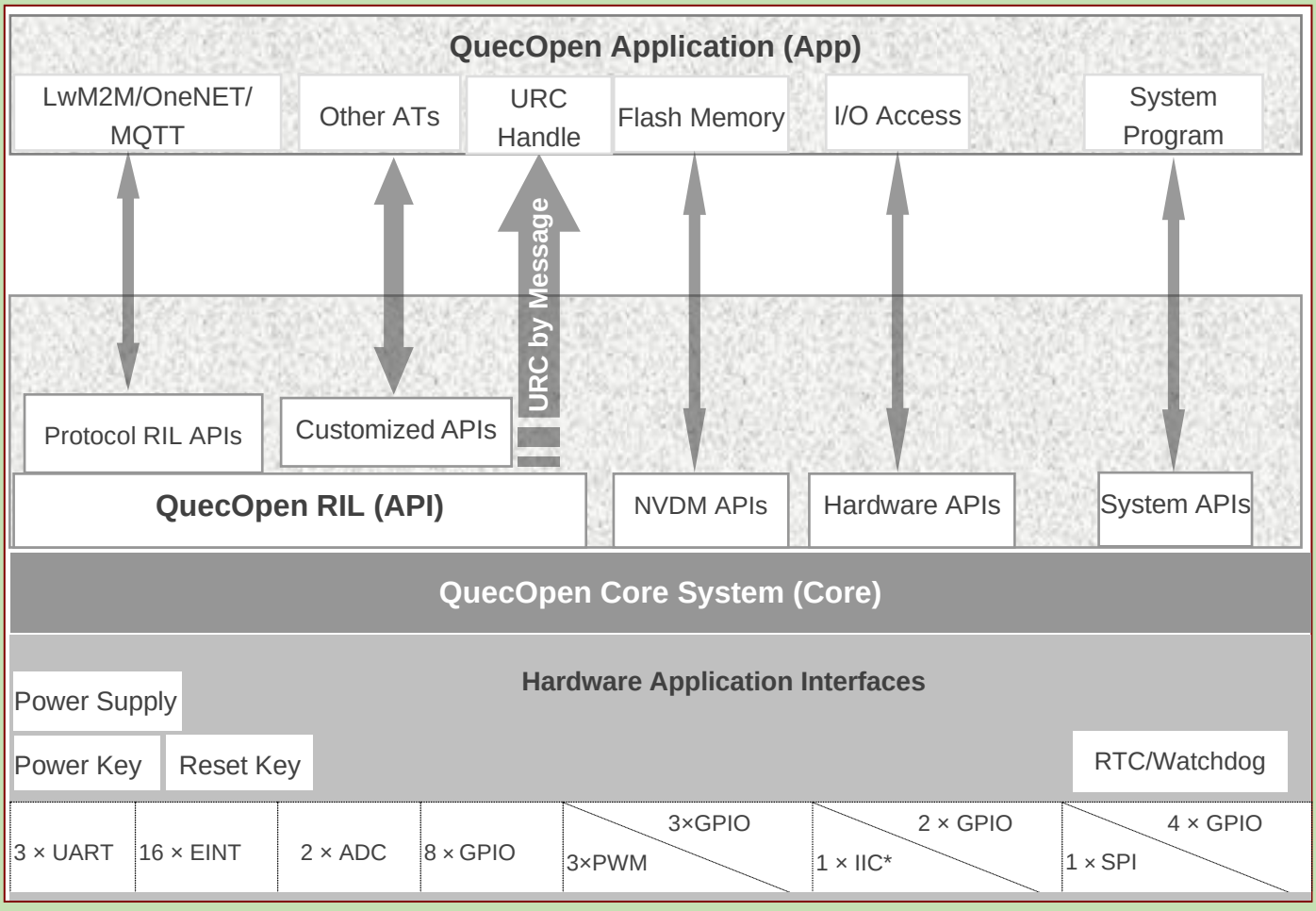

Figure 1: QuecOpen Software Architecture

EINT, PWM, I2C, SPI, part of ADC and GPIO of BC660K-GL QuecOpen module are multiplexed interfaces.

QuecOpen core system is a combination of hardware and software of NB-IoT module. It is constructed by the built-in ARM Cortex-M3 processor and FreeRTOS operating system. With the character of

micro-kernel, real-time, and multitasking, etc. QuecOpen APIs are designed for accessing hardware resources, radio communications resources, or external devices. All APIs are introduced in ***Chapter 4***.

QuecOpen RIL is an open source layer, which enables developers to simply call APIs to send AT commands and get the response when API returns. Additionally, new APIs can be easily added to implement AT commands sending requirements.

In the QuecOpen RIL mechanism, all URC messages of the module have already been reinterpreted and the result is informed to App by system message. All URC messages are reported to the application by the system message MSG_ID_URC_INDICATION, and are reported to the main task by default.

2.2. Open Resources

2.2.1. Processor

32-bit ARM Cortex-M3 processor with a maximum frequency of 204 MHz.

2.2.2. Memory Schemes

BC660K-GL QuecOpen module builds in 4 MB Flash and 272 KB RAM.

- APP Storage Space: 200 KB, used to store the developer's bin files.
- RAM Space: 30 KB, used to store dynamic memory.

2.3. Peripherals

2.3.1. UART

BC660K-GL QuecOpen module provides 3 UART ports: the main UART port, the auxiliary UART port and debug UART port; the debug UART port can only be used to print EPAT debugging information.

2.3.2. GPIO

BC660K-GL QuecOpen module provides 16 I/O pins that can be configured to general-purpose I/O. The pins can be accessed by the GPIO APIs; for the usage of the GPIO interface and related APIs, please refer to ***Chapter 4.8.2***.

2.3.3. EINT

BC660K-GL QuecOpen module provides 16 I/O pins that can be configured to external interrupt functions. All other GPIOs except GPIO1, RXD_AUX and TXD_AUX support the configuration of external interrupt functions, but only one pin interrupt is opened at most. The EINT pin can be accessed by APIs; for the usage of the EINT interface and related APIs, please refer to ***Chapter 4.8.3***.

2.3.4. PWM

BC660K-GL QuecOpen module provides 3 pins that support the PWM function, and you can select 32 K or 26 M clock sources, but only one PWM pin can be selected for output at the same time. The PWM pins can be accessed by APIs; for the usage of the PWM interface and related APIs, please refer to ***Chapter 4.8.4***.

2.3.5. ADC

BC660K-GL QuecOpen module provides 2 I/O pins that support dedicated ADC (ADC0, ADC1) function. The voltage value of the corresponding pin can be directly read by APIs; for the usage of the ADC interface and related APIs, please refer to ***Chapter 4.8.5***.

2.3.6. I2C

BC660K-GL QuecOpen module provides a hardware I2C interface and also supports an analog I2C interface. For the usage of the I2C interface and related APIs, please refer to ***Chapter 4.8.6***.

2.3.7. SPI

BC660K-GL QuecOpen module provides a hardware SPI interface, and supports analog SPI interface. For the usage of the SPI interface and related APIs, please refer to ***Chapter 4.8.7***.

3 Base Data Types

3.1. Required Header File

In QuecOpen, the base data types are defined in *ql_type.h* header file.

3.2. Base Data Type

The basic data types defined are shown in the following table.

Table 1: Base Data Type

Type	Description
bool	Boolean variable (should be TRUE or FALSE). This variable is declared as follows: typedef unsigned char ubool;
ascii	8-bit signed ASCII code. This variable is declared as follows: typedef char ascii;
s8	8-bit signed integer. This variable is declared as follows: typedef signed char s8;
u8	8-bit unsigned integer. This variable is declared as follows: typedef unsigned char u8;
s16	16-bit signed integer. This variable is declared as follows: typedef signed short s16;
u16	16-bit unsigned integer. This variable is declared as follows: typedef unsigned short u16;
s32	32-bit signed integer. This variable is declared as follows:

	typedef long s32;
u32	32-bit unsigned integer. This variable is declared as follows: typedef unsigned long u32;
u64	64-bit unsigned integer. This variable is declared as follows: typedef unsigned long long u64;
s64	64-bit signed integer. This variable is declared as follows: typedef long long s64;
ticks	32-bit unsigned integer. This variable is declared as follows: typedef unsigned int ticks;

4 APIs

4.1. FreeRTOS API

BC660K-GL QuecOpen module supports functions with messages, mutual exclusion, semaphores, and events. These interfaces are usually used for multi-task programming. The operating system of BC660K-GL is FreeRTOS, which opens all OS related APIs to users, so you can directly use FreeRTOS or CMSIS related APIs.

Therefore, developers can also refer to the API introduction officially released on FreeRTOS operating system. The example *example_multitask.c* in BC660K-GL QuecOpen SDK shows how to use related APIs.

4.2. Time API

The time APIs of BC660K-GL QuecOpen module can be used to set UTC time, get UTC time, convert calendar time to seconds, etc.

4.2.1. Header File

ql_time.h file is the header file of the time API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.2.2. Function Overview of Time

Table 2: Function Overview of Time

Function	Description
Ql_TIME_Get()	Get the current UTC time and time zone
Ql_TIME_Set()	Set the current UTC time and time zone

Ql_Mktime()	Get total seconds of UTC time elapsed since 1970.01.01 00:00:00

4.2.3. Usage

The time format of the SDK is expressed in the format of the structure *ST_TIME*. The local time is converted from UTC time to the local time zone. The calendar time is the total number of seconds from the standard time point to the current time. Generally, 00:00:00 on January 1st, 1970 is set as the standard point in time. *ST_Time* is defined as follows:

```
typedef struct
{
    u16     year;
    u8      month;
    u8      day;
    u8      hour;
    u8      minute;
    u8      second;
    s32     time_zone;      //The range is from -47 to 48, one digit expresses 15 minutes, for
                              example: 32 indicates "GMT+8"
}ST_Time
```

- **Parameter**

Type	Parameter	Description
u16	*year*	year
u8	*month*	month
u8	*day*	day
u8	*hour*	hour
u8	*minute*	minute
u8	*second*	second
s32	*time_zone*	Time zone. One digit expresses 15 minutes, for example: 32 indicates "GMT+8". Range: -47–48.

4.2.4. API Description

4.2.4.1. Ql_TIME_Get

This function gets the current UTC time and time zone.

- **Prototype**

```
s32 Ql_TIME_Get(ST_Time * dateTime)
```

- **Parameter**

dateTime:
[Out] The UTC time and time zone. The default time is 2000.1.1 00:00:00.

- **Return Value**

0	Successful execution.
-2	Invalid parameter.
-81	Get current datetime error.

4.2.4.2. Ql_TIME_Set

This function sets the current UTC time and time zone.

- **Prototype**

```
s32 Ql_TIME_Set(ST_Time * dateTime)
```

- **Parameter**

datetime:
[In] The UTC time and time zone.

- **Return Value**

0	Successful execution.
-2	Invalid parameter.
-82	Set current datetime error.

NOTE

Only the year setting greater than 2000 is supported.

4.2.4.3. Ql_Mktime

This function gets total seconds of UTC time elapsed since 1970.01.01 00:00:00.

- **Prototype**

```
s32 Ql_Mktime(u32 * seconds)
```

- **Parameter**

seconds:
[Out] Seconds of UTC time elapsed since 1970.01.01 00:00:00.

- **Return Value**

0 successful execution.
-2 Invalid parameter.
-83 Get current datetime of second error.

4.3. Timer API

BC660K-GL QuecOpen module supports two kinds of timers: software timers and hardware timers; among them, there are 30 software timers and 5 hardware timers.

NOTE

The accuracy of the hardware timer is greater than that of the software timer. It is recommended to use the hardware timer for application scenarios that require greater timing accuracy.

4.3.1. Header File

ql_timer.h file is the header file of the timer API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.3.2. Function Overview of Timer

Table 3: Function Overview of Timer

Function	Description
Ql_TIMER_Register()	Register a software timer
Ql_TIMER_RegisterFast()	Register a hardware timer
Ql_TIMER_Start()	Start up a specified timer
Ql_TIMER_Stop()	Stop a specified timer
Ql_TIMER_Delete()	Delete a specified timer

4.3.3. Usage

Ql_TIMER_Register() can initialize a timer and register a callback function. *Ql_TIMER_Start()* can start the created timer, and *Ql_TIMER_Stop()* can stop the running timer.

Callback function is defined as follows:

```
typedef void(*Callback_Timer_OnTimer)(u32 timerId, void* param)
```

4.3.4. API Description

4.3.4.1. Ql_TIMER_Register

This function registers a software timer.

- **Prototype**

```
s32 Ql_TIMER_Register(u32 timerId, Callback_Timer_OnTimer callback_onTimer, void* param)
```

- **Parameter**

timerId:
[In] Timer ID. It must be ensured that the ID is the only one under QuecOpen task. It should also differ from the ID registered by *Ql_Timer_RegisterFast()*.

callback_onTimer:
[In] Callback function.

param:
[In] A customized parameter that can be passed into the callback functions.

- **Return Value**

0 Successful execution.
-2 The parameter error.
-71 The timer ID is already being used or the timer is started or stopped.
-73 All timers are used up.

4.3.4.2. Ql_TIMER_RegisterFast

This function registers a hardware timer.

- **Prototype**

```
s32 Ql_TIMER_RegisterFast(u32 timerId, Callback_Timer_OnTimer callback_onTimer, void* param)
```

- **Parameter**

timerId:
[In] Timer ID. It must be ensured that the ID is the only one under QuecOpen task. It should also differ from the ID registered by *Ql_Timer_RegisterFast()*.

callback_onTimer:
[In] Callback function.

param:
[In] One customized parameter that can be passed into the callback functions.

- **Return Value**

0 Indicates successful execution.
-2 Indicates the parameter error.
-71 Indicates that the timer ID is already being used or the timer is started or stopped.
-73 Indicates all timers are used up.

4.3.4.3. Ql_TIMER_Start

This function starts up a specified timer.

- **Prototype**

```
s32 Ql_TIMER_Start(u32 timeId, u32 interval, bool autoRepeat)
```

- **Parameter**

timerId:
[In] Timer ID, which must be registered.

interval:
[In] The timer interval. Unit: ms. The minimum interval must be greater than or equal to 1ms.

autoRepeat:
[In] TRUE or FALSE. TRUE indicates the timer is executed repeatedly; FALSE indicates the timer is executed once.

- **Return Value**

0 Successful execution.
-2 The parameter error.
-71 The timer ID is already being used or the timer is started or stopped.
-73 All timers are used up.

4.3.4.4. Ql_TIMER_Stop

This function stops a specified timer that must be a registered timer.

- **Prototype**

```
s32 Ql_TIMER_Stop(u32 timeId)
```

- **Parameter**

timerId:
[In] Timer ID.

- **Return Value**

0 Indicates successful execution.
-2 Indicates the parameter error.
-71 Indicates that the timer ID is already being used or the timer is started or stopped.
-73 Indicates all timers are used up.

NOTE

The non-reloading timer automatically terminates when the timing is reached.

4.3.4.5. Ql_TIMER_Delete

This function deletes a specified timer.

- **Prototype**

```
s32 Ql_TIMER_Delete(u32 timeId)
```

- **Parameter**

timerId:
[In] Timer ID. It must be a registered timer.

- **Return Value**

0	Successful execution.
-2	The parameter error.
-71	The timer ID is already being used or the timer is started or stopped.
-73	All timers are used up.

4.4. RTC API

BC660K-GL QuecOpen supports an RTC timer. When the module enters deep sleep mode, the RTC will still work normally. When the RTC timing time is reached, the module will wake up from deep sleep.

4.4.1. Header File

ql_rtc.h file is the header file of the RTC API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.4.2. Function Overview of RTC

Table 4: Function Overview of RTC

Function	Description
Ql_RTC_Register()	Register the RTC timer. When the RTC timer expires, the registered callback function will be called.
Ql_RTC_Start()	Start up an RTC timer.
Ql_RTC_Stop()	Stop an RTC timer.

4.4.3. Usage

You can refer to the following steps to complete the RTC operation:

- Call *Ql_RTC_Register()* to register the RTC timer and its callback function.
- Call *Ql_RTC_Start()* to start the RTC timer.
- Call *Ql_RTC_Stop()* to stop the RTC timer.

Callback function is defined as follows:

```
typedef void(* callback_rtc_func)(u32 rtcId, void* param)
```

4.4.4. API Description

4.4.4.1. Ql_RTC_Register

This function registers the RTC timer. When the RTC timer expires, the registered callback function will be called.

- **Prototype**

```
s32 Ql_RTC_Register(u32 rtc_id, callback_rtc_func callback_timer, void* param)
```

- **Parameter**

rtc_id:
[In] RTC timer ID, only one channel is supported.

callback_timer:
[Out] Callback function. Notify the application when the time of the RTC arrives.

param:
[In] One customized parameter that can be passed into the callback functions.

- **Return Value**

0 Successful execution
-2 The parameter error
-61 RTC is used up
-64 RTC is registered

4.4.4.2. Ql_RTC_Start

This function starts up an RTC timer.

- **Prototype**

```
s32 Ql_RTC_Start(u32 rtc_id, u32 interval, bool auto_repeat)
```

- **Parameter**

rtc_id:
[In] Timer ID, which must be registered.

interval:
[In] The timer interval. Unit: ms. The minimum interval must be greater than or equal to 1 ms.

auto_Repeat:
[In] TRUE or FALSE. TRUE indicates the timer is automatically executed repeatedly; FALSE indicates the timer is executed once.

- **Return Value**

0 Indicates successful execution.
-2 Indicates the parameter error.
-62 indicates RTC is running.

4.4.4.3. Ql_RTC_Stop

This function stops an RTC timer.

- **Prototype**

```
s32 Ql_RTC_Stop(u32 rtc_id)
```

- **Parameter**

rtc_id:
[In] The unique ID of the RTC timer.

- **Return Value**

0 Indicates successful execution.
-2 Indicates the parameter error.
-63 Indicates RTC is not running.

4.5. Deep Sleep API

The BC660K-GL QuecOpen module supports the sleep mechanism. When all tasks on the AP side are in the suspended state and the network side is in the PSM state, the module will actively enter the deep sleep mode to reduce power consumption.

In addition to exiting the deep sleep mode through the RTC timer timeout, it also supports waking up the module by PSM_EINT pin. When the module is in deep sleep mode, give the PSM_EINT pin a falling edge to wake up the module.

4.5.1. Header File

ql_power.h file is the header file of the deep sleep API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.5.2. Function Overview of Deep Sleep

Table 5: Function Overview of Deep Sleep

Function	Description
Ql_GetWakeUpReason()	Get a reason for power on
Ql_SleepEnable()	Enable the module into sleep mode
Ql_SleepDisable()	Disable the module entering sleep mode, including deep sleep and light sleep

Ql_GetPowerVol()	Query the voltage value of power supply
Ql_DeepSleep_Register()	Register the callback function of the deep sleep mode, the callback function needs to be called before the module enters the deep sleep mode.

4.5.3. Usage

- Call *Ql_SleepDisable()* to prohibit the module from entering sleep mode.
- Call *Ql_SleepEnable()* to enable the module to enter sleep mode.
- Call *Ql_DeepSleep_Register()* to register a callback function. When the module enters deep sleep mode, the callback function reports the status to the APP.

Callback function for the module to enter the deep sleep mode is defined as follows:

```
typedef void (*Callback_DeepSleep_Func) (u8* buffer,u32 length)
```

The wake-up reasons are defined as follows:

```
Typedef enum {
    //* First power on or hardware reset */
    QL_POWERON                  = 0,
    //*   software reset */
    QL_SOFT_RESET               = 1,
    //* Wake up by RTC timer from deep sleep. TAU/ALARM(oc,onet)/USER RTC */
    QL_RTCWAKEUP                = 2,
    //* Wake up by UART0-RXD from deep sleep */
    QL_RXD_WAKEUP               = 3,
    //* Wake up by PSM_EINT0 from deep sleep */
    QL_PSM_EINT0_WAKEUP         = 4,
    //* Wake up by PSM_EINT1 from deep sleep */
    QL_PSM_EINT1_WAKEUP         = 5,

    /* others reset */
    QL_UNKNOWN_PWRON,
} Enum_WakeUp_Reason
```

- **Parameter**

Parameter	Description
QL_POWERON	First power on or reset
QL_SOFT_RESET	Software reset

QL_RTCWAKEUP	Wake up by RTC timer from deep sleep
QL_RXD_WAKEUP	Wake up by UART0-RXD from deep sleep
QL_PSM_EINT0_WAKEUP	Wake up by PSM_EINT0 from deep sleep
QL_PSM_EINT1_WAKEUP	Wake up by PSM_EINT1 from deep sleep
QL_UNKNOWN_PWRON	Unknown power on

4.5.4. API Description

4.5.4.1. Ql_GetWakeUpReason

This function gets a reason for module power-on.

- **Prototype**

```
Enum_WakeUp_Reason Ql_GetWakeUpReason(void)
```

- **Parameter**

Void.

- **Return Value**

please refer to *Enum_WakeUp_Reason* in *ql_power.h*.

4.5.4.2. Ql_SleepEnable

This function enables the module into sleep mode.

- **Prototype**

```
s32 Ql_SleepEnable(void)
```

- **Parameter**

Void.

- **Return Value**

0 Successful execution.

4.5.4.3. Ql_SleepDisable

This function disables the module entering sleep mode, including deep sleep and light sleep.

- **Prototype**

```
s32 Ql_SleepDisable(void)
```

- **Parameter**

Void.

- **Return Value**

0 Successful execution.

NOTE

This interface prevents the module from entering light sleep and deep sleep modes. If you find that the module cannot enter sleep mode normally, please check whether this function is called in the App.

4.5.4.4. Ql_DeepSleep_Register

This function registers the callback function of the deep sleep mode, the callback function needs to be called before the module enters the deep sleep mode.

- **Prototype**

```
s32 Ql_DeepSleep_Register(Callback_DeepSleep_Func callback_deepsleep)
```

- **Parameter**

callback_deepsleep:
[out] Callback function.

- **Return Value**

0 Successful execution.
-1 Failed execution.

NOTES

1. This function only prompts the module to enter the deep sleep mode. The callback function is triggered only when the module enters deep sleep, and then the module enters deep sleep; please do not do too much complicated business in the callback function.
2. API functions cannot be called in *callback_ds_event*. It is recommended to do some variable operations.

4.6. Power Management API

4.6.1. Header File

ql_power.h file is the header file of the power management API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.6.2. Usage

Power management includes the operation of getting the power supply voltage.

4.6.3. API Description

4.6.3.1. Ql_GetPowerVol

This function queries the voltage value of power supply.

- **Prototype**

```
s32 Ql_GetPowerVol(u32* voltage)
```

- **Parameter**

voltage:
[Out] Voltage value of power supply. Unit: mV.

- **Return Value**

0 Successful execution.
-1 Failed execution.

4.7. Network Management API

4.7.1. Header File

ql_ps.h file is the header file of the network management API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.7.2. Function Overview of Network Management

Table 6: Function Overview of Network Management

Function	Description
Ql_PSCallback_Register()	Register psCallback which indicates some PS event
Ql_GetSN()	Get SN of the module
Ql_GetIMEI()	Get IMEI of the module
Ql_GetIMSI()	Get IMSI of the card, which can only be gotten after identifying the SIM card successfully
Ql_GetICCID()	Get ICCID of the card, which can only be gotten after identifying the SIM card successfully
Ql_GetCeregState()	Get EPS registration status of the module
Ql_GetLocInfo()	Get the current cell information of the module, which needs to be queried after registering network
Ql_GetECL()	Get the coverage level of the environment where the module is currently located
Ql_GetSignalInfo()	Ges the network signal status of the cell where the module is currently located
Ql_GetCellInfo()	Get the relevant information of the cell where the module is currently located

4.7.3. Usage

Network management includes operations related to the network, such as getting cell information, network status, etc. For more details, please refer to the API description, or refer to *example_ps.c* in the SDK.

The group masks are defined as follows:

```
typedef enum {
    GROUP_BASE_MASK = (0X01 << U_CAC_BASE),
```

```
    GROUP_DEV_MASK    = (0X01 << U_CAC_DEV),
    GROUP_MM_MASK     = (0X01 << U_CAC_MM),
    GROUP_PS_MASK     = (0X01 << U_CAC_PS),
    GROUP_SIM_MASK    = (0X01 << U_CAC_SIM),
    GROUP_SMS_MASK    = (0X01 << U_CAC_SMS),
    GROUP_ALL_MASK    = 0X7FFFFFFF
} ENUMgroupMask
```

- **Parameter**

Parameter	Description
GROUP_BASE_MASK	
GROUP_DEV_MASK	
GROUP_MM_MASK	
GROUP_PS_MASK	
GROUP_SIM_MASK	
GROUP_SMS_MASK	
GROUP_ALL_MASK	

4.7.4. API Description

4.7.4.1. Ql_PSCallback_Register

This function registers psCallback which indicates some PS event.

- **Prototype**

```
s32 Ql_PSCallback_Register(ENUMgroupMask event,psCallback callback)
```

- **Parameter**

event:
[In] Please refer to *ENUMgroupMask* in *ql_ps.h*.

callback:
[In] Callback function.

- **Return Value**

-2 Invalid parameter.

4.7.4.2. Ql_GetSN

This function gets SN of the module.

- **Prototype**

```
s32 Ql_GetSN(char* str)
```

- **Parameter**

str:
[Out] String type of SN.

- **Return Value**

0 Successful execution.
-1 Failed execution.

4.7.4.3. Ql_GetIMEI

This function gets IMEI of the module.

- **Prototype**

```
s32 Ql_GetIMEI(char* str)
```

- **Parameter**

str:
[Out] String type of IMEI.

- **Return Value**

0 Successful execution.
-1 Failed execution.

4.7.4.4. Ql_GetIMSI

This function gets IMSI of the SIM card, which can only be got after identifying the SIM card successfully.

- **Prototype**

```
s32 Ql_GetIMSI(char* str)
```

- **Parameter**

str:
[Out] String type of IMSI.

- **Return Value**

0 Successful execution.
-1 Failed execution.

4.7.4.5. Ql_GetICCID

This function gets ICCID of the SIM card, which can only be gotten after identifying the SIM card successfully.

- **Prototype**

```
s32 Ql_GetICCID(char* str)
```

- **Parameter**

str:
[Out] String type of ICCIDI.

- **Return Value**

0 Successful execution.
-1 Failed execution.

4.7.4.6. Ql_GetCeregState

This function gets CEREG registration status of the module.

- **Prototype**

```
s32 Ql_GetCeregState(u8 *state)
```

- **Parameter**

state:
[Out] EPS registration status

0	Not registered, MT is not searching the network currently
1	Registered, home network
2	Not registered, but MT is trying to attach or search the network to register currently
3	Registration rejected
4	Unknown (for example: beyond the coverage of E-UTRAN)
5	Registration, roaming status

- **Return Value**

0	Successful execution.
-1	Failed execution.

4.7.4.7. Ql_GetLocInfo

This function gets the current cell information of the module, which needs to be queried after registering network.

- **Prototype**

```
s32 Ql_GetLocInfo(u16 *tac, u32 *cellId)
```

- **Parameter**

tac:
[Out] Tracking area code. Two bytes in hexadecimal format (for example, "00C3" is equal to 195 in decimal).

cellId:
[Out] E-UTRAN cell ID. Four bytes in hexadecimal format.

- **Return Value**

0	Successful execution.
-1	Failed execution.

4.7.4.8. Ql_GetECL

This function gets the coverage level of the environment where the module is currently located.

- **Prototype**

```
s32 Ql_GetECL(void)
```

- **Parameter**

Void.

- **Return Value**

0 Good coverage
1 Medium coverage
2 Poor coverage

4.7.4.9. Ql_GetSignalInfo

This function gets the network signal status of the cell where the module is currently located.

- **Prototype**

```
s32 Ql_GetSignalInfo(u8 *csq, s8 *snr, s8 *rsrp)
```

- **Parameter**

csq:
[Out] Pointer to received signal strength.

0	-113 dBm or less
1	-111 dBm
2–30	-109 dBm to -53 dBm
31	-51 dBm or greater
99	Unknown

snr:
[Out] Pointer to signal-to-noise ratio (SNR) of serving cell. Range: -30–30. Unit: dB. It can be negative.

rsrp:
[Out] Pointer to signal received power (RSRP).
When data needs to be sent, it is recommended to be greater than -115 dBm.

0	-140 dBm or below

1 -140 dBm ≤ *rsrp* <-139 dBm
2 -139 dBm ≤ *rsrp* <-138 dBm
…
95 -46 dBm ≤ *rsrp* <-45 dBm
96 -45 dBm ≤ *rsrp* <-44 dBm
97 -44 dBm ≤ *rsrp*
255 Unknown

1) AS NB extended the RSRP value in: TS 36.133-v14.5.0, Table 9.1.4-1
-17 *rsrp* < -156 dBm
-16 -156 dBm ≤ *rsrp* < -155 dBm
...
-3 -143 dBm ≤ *rsrp* < -142 dBm
-2 -142 dBm ≤ *rsrp* < -141 dBm
-1 -141 dBm ≤ *rsrp* < -140 dBm
0 *rsrp* < -140 dBm
1 -140 dBm ≤ *rsrp* < -139 dBm
2 -139 dBm ≤ *rsrp* < -138 dBm
...
95 -46 dBm ≤ *rsrp* < -45 dBm
96 -45 dBm ≤ *rsrp* < -44 dBm
97 -44 dBm ≤ *rsrp*
2) If not valid, set to 127

- **Return Value**

0 Successful execution.
-1 Failed execution.

4.7.4.10. Ql_GetCellInfo

This function gets the relevant information of the cell where the module is currently located.

- **Prototype**

```
s32 Ql_GetCellInfo(Ql_CellListInfo_t *bcListInfo)
```

- **Parameter**

bcListInfo:
[Out] Service cell information.

- **Return Value**

0 Successful execution.
-1 Failed execution.

NOTE

To facilitate the understanding of the network environment, the network quality can be evaluated according to the following general rules:

4. Strong: RSRP ≥ -100 dBm, and SNR ≥ 3 dB
5. Medium: -100 dBm ≥ RSRP ≥ -110 dBm and 3 dB> SNR> -3 dB
6. Weak: RSRP <-115 dBm or SNR <-3 dB

4.8. Peropheral Interface API

The peripheral interfaces of the BC660K-GL QuecOpen module include UART, GPIO, EINT, PWM, ADC, I2C and SPI.

4.8.1. UART

4.8.1.1. Header File

ql_uart.h file is the header file of the UART API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.8.1.2. Function Overview of UART

Table 7: Function Overview of UART

Function	Description
Ql_UART_Open()	Open the UART port and configures the baud rate at the same time
Ql_UART_Write()	End data to the specified UART port
Ql_UART_Close()	Close the specified UART port

4.8.1.3. Usage

The BC600K-GL QuecOpen module provides 3 UARTs. The main UART port and the auxiliary UART port

can be used as ordinary communication ports, and the debug UART port can only be used as EPAT tool debugging ports. In addition, the main UART port can be used as a download port and a wake-up port for light sleep/deep sleep mode.

Before using the UART port, you need to open the corresponding UART port and configure the UART port parameters. The following steps as follows:

Step 1: Open the UART port. Call *Ql_UART_Open()* to open the UART port, and configure the baud rate and callback function at the same time. You can also call *Ql_UART_OpenEx()* to open the UART port and configure the baud rate, data bit, stop bit, parity bit, etc.
Step 2: UART print output. Call *Ql_UART_Write()* to print data out by the specified UART port.
Step 3: Close the UART port. When the UART port is no longer needed, call *Ql_UART_Close()* to close the specified UART port.

The module supports three UART ports, the enumeration is defined as follows:

```
typedef enum
{
      UART_PORT0 = 0,
      UART_PORT1,
      UART_PORT2,
       UART_NONE,
}Enum_SerialPort;
```

- **Parameter**

Parameter	Description
UART_PORT0	Main UART port
UART_PORT1	Auxiliary UART port
UART_PORT2	Debug UART port, only for EPAT printing log

Callback function is defined as follows:

```
typedef void (* CallBack_UART_Notify)(u32 event, void* dataPtr, u32 dataLen);
```

4.8.1.4. API Description

4.8.1.4.1. Ql_UART_Open

This function opens the UART port and configures the baud rate at the same time.

- **Prototype**

```
s32 Ql_UART_Open(Enum_SerialPort port,u32 baudrate,CallBack_UART_Notify uart_receive_call_back)
```

- **Parameter**

port:
[In] Port name.

baudrate:
[In] Baud rate. The maximum value is 921600. Unit: bps.

uart_receive_call_back:
[In] Callback function, when the serial port receives data, call the interrupt interface.

- **Return Value**

0 Indicates successful execution
-2 Indicates the parameter error
-16 Indicates UART initialization error.
-17 Indicates port is not supported.
-18 Indicates baud rate is not supported.

4.8.1.4.2. Ql_UART_Write

This function sends data to the specified UART port.

- **Prototype**

```
s32 Ql_UART_Write(Enum_SerialPort port,u8 *buff,u32 buff_len)
```

- **Parameter**

port:
[In] Specified UART port.

buff:
[In] The data to be sent.

buff_len:
[In] The length of the data to be sent.

- **Return Value**

0 Indicates successful execution.
-2 Indicates the parameter error.
-17 Indicates port is not supported.

4.8.1.4.3. Ql_UART_Close

This function closes the specified UART port.

- **Prototype**

```
s32 Ql_UART_Close(Enum_SerialPort port)
```

- **Parameter**

port:
[In] Serial port number.

- **Return Value**

0 Indicates successful execution.
-2 Indicates the parameter error.
-17 Indicates port is not supported.

4.8.2. GPIO

4.8.2.1. Header File

ql_gpio.h file is the header file of the GPIO API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.8.2.2. Function Overview of GPIO

Table 8: Function Overview of GPIO

Function	Description
Ql_GPIO_Init()	Enable the GPIO function of the specified pin, and initialize the configurations, including direction, level and pull selection
Ql_GPIO_SetLevel()	Set the level of a specified GPIO
Ql_GPIO_GetLevel()	Get the level of a specified GPIO

Ql_GPIO_GetDirection()	Get the direction of a specified GPIO
Ql_GPIO_Uninit()	Release a specified GPIO

4.8.2.3. Usage

The BC660K-GL QuecOpen module opens 16 pins that can be configured as GPIO, and all open pins can be configured by the GPIO-related API interface. The configuration method of GPIO function is as follows:

Step 1: GPIO initialization. Call *Ql_GPIO_Init()* to set the specified pin as GPIO function, including direction, initial level and pull-up configuration.
Step 2: GPIO control. When the pin is initialized as a GPIO, the developer can call the GPIO-related API to control and get the direction, level, and up-down mode of the GPIO.
Step 3: Release GPIO. If you no longer use this pin as a GPIO, but need to multiplex this pin for other functions (such as PWM, EINT), you can call *Ql_GPIO_Uninit()* to release the pin first.

4.8.2.4. API Description

4.8.2.4.1. Ql_GPIO_Init

This function enables the GPIO function of the specified pin, and initializes the configurations, including direction, level and pull selection.

- **Prototype**

```
s32 Ql_GPIO_Init(Enum_PinName pinName, Enum_PinDirection dir, Enum_PinLevel level, Enum_PinPullSel pullsel)
```

- **Parameter**

pinName:
[In] Pin name. Please refer to the *Enum_PinName* in the SDK.

dir:
[In] The direction of GPIO. Please refer to the *Enum_PinName* in the SDK.

level:
[In] The initial level of GPIO. Please refer to the *Enum_PinName* in the SDK.

pullsel:
[In] Pull selection. Please refer to the *Enum_PinName* in the SDK.

- **Return Value**

0 Successful execution.
-10 The input GPIO is invalid.
-12 the GPIO is used in other places. For example, this GPIO has been using as EINT.

4.8.2.4.2. Ql_GPIO_SetLevel

This function sets the level of a specified GPIO.

- **Prototype**

```
s32 Ql_GPIO_SetLevel(Enum_PinName pinName, Enum_PinLevel level)
```

- **Parameter**

pinName:
[In] Pin name. Please refer to the *Enum_PinName* in the SDK.

level:
[In] The level of GPIO. Please refer to the *Enum_PinName* in the SDK.

- **Return Value**

0 Successful execution.
-2 The input GPIO is invalid.
-13 It cannot be operated, maybe the GPIO is not initialized.

4.8.2.4.3. Ql_GPIO_GetLevel

This function gets the level of a specified GPIO.

- **Prototype**

```
s32 Ql_GPIO_GetLevel(Enum_PinName pinName)
```

- **Parameter**

pinName:
[In] Pin name. Please refer to the *Enum_PinName* in the SDK.

- **Return Value**

The level value of the specified GPIO, which is in *Enum_PinDirection* in the SDK.
-2 The input GPIO is invalid.

-13 It cannot be operated, maybe the GPIO is not initialized.

4.8.2.4.4. Ql_GPIO_GetDirection

This function gets the direction of a specified GPIO.

- **Prototype**

```
s32 Ql_GPIO_GetDirection(Enum_PinName pinName)
```

- **Parameter**

pinName:
[In] Pin name. Please refer to the *Enum_PinName* in the SDK.

- **Return Value**

The direction of the specified GPIO, which is in *Enum_PinDirection* in the SDK.
-2 The input GPIO is invalid.
-13 It cannot be operated, maybe because the GPIO is not initialized.

4.8.2.4.5. Ql_GPIO_Uninit

This function releases a specified GPIO。

- **Prototype**

```
s32 Ql_GPIO_Uninit(Enum_PinName pinName)
```

- **Parameter**

pinName:
[In] Pin name. Please refer to the *Enum_PinName* in the SDK.

- **Return Value**

0 Successful execution.
-2 The input GPIO is invalid.
-13 It cannot be operated, maybe the GPIO is not initialized.

4.8.3. EINT

4.8.3.1. Header File

ql_gpio.h file is the header file of the ENIT API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.8.3.2. Function Overview of EINT

Table 9: Function Overview of EINT

Function	Description
Ql_EINT_Init()	Initialize an interrupt function
Ql_EINT_Uninit()	Release a specified EINT pin

4.8.3.3. Usage

BC660K-GL QuecOpen module opens 16 pins that can be multiplexed as interrupt pins, but only 3 channels of EINT can be configured at the same time. You can register and use the EINT function of the relevant pins by the EINT API function. The usage of API functions are as follows:

Step 1: Initialize the interrupt configuration. Call *Ql_EINT_Init()* to configure the pin's interrupt mode and debounce time, etc.

Step 2: Release the specified EINT pin. When a pin is no longer needed to use the interrupt function, *Ql_EINT_Uninit()* can be called to release the pin; after that, the pin can be reused for other functions.

The format of the interrupt service function is as follows:

```
typedef void (*Callback_EINT_Handle)(Enum_PinName pinName)
```

4.8.3.4. API Description

4.8.3.4.1. Ql_EINT_Init

This function initializes an interrupt function.

- **Prototype**

```
s32 Ql_EINT_Init(Enum_PinName pinName ,Enum_EintType eintType ,u32 swDebounce, Callback_EINT_Handle eint_isr)
```

- **Parameter**

pinName:
[In] Pin name, only pins that support interrupt function can be selected. Please refer to the *Enum_PinName* in the SDK.

eintType:
[In] Interrupt type. Please refer to the *Enum_PinName* in the SDK.

swDebounce:
[In] Software debounce. Range: 0–1000.

eint_isr:
[In] Interrupt service function.

- **Return Value**

0 Successful execution.
-2 The input IO is invalid.
-8 *swDebounce* is more than 1000 ms.
-11 EINT pin is more than the maximum (maximum is 3 now).
-12 It cannot be operated, maybe the GPIO is already initialized.

4.8.3.4.2. Ql_EINT_Uninit

This function releases a specified EINT pin.

- **Prototype**

```
s32 Ql_EINT_Uninit(Enum_PinName pinName)
```

- **Parameter**

pinName:
[In] Pin name, only pins that support interrupt function can be selected. Please refer to the *Enum_PinName* in the SDK.

- **Return Value**

0 Successful execution.

-2 The input GPIO is invalid.
-13 It cannot be operated, maybe the GPIO is not initialized.

4.8.4. PWM

4.8.4.1. Header File

ql_pwm.h file is the header file of the PWM API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.8.4.2. Function Overview of PWM

Table 10: Function Overview of PWM

Function	Description
Ql_PWM_Init()	Initialize the specified pin as the PWM function
Ql_PWM_Output()	Control the output of the PWM waveform
Ql_PWM_UpdateDutyCycle()	Update the duty cycle of the PWM waveform
Ql_PWM_Uninit()	Release the pins that have been initialized as PWM functions

4.8.4.3. Usage

BC660K-GL QuecOpen module provides 3 pins that support PWM function: SPI_SCLK, GPIO4 and MAIN_RTS. The PWM function of these pins can be configured by API. The clock source of PWM supports 32 KHz and 26 MHz, and the support range is 2 Hz–6.5 MHz. The configuration steps of PWM function are as follows:

Step 1: Initialize the PWM pin. Call *Ql_PWM_Init()* to configure the PWM duty cycle and frequency.
Step 2: Control the PWM waveform output. Call *Ql_PWM_Output()* to turn on/off the PWM waveform output.
Step 3: Update the duty cycle of the PWM waveform. Call *Ql_PWM_UpdateDutyCycle()* to update the high-level duty cycle of the PWM waveform.
Step 4: Release the PWM pin. Call *Ql_PWM_Uninit()* to release the PWM pin.

4.8.4.4. API Description

4.8.4.4.1. Ql_PWM_Init

This function initializes the specified pin as the PWM function.

- **Prototype**

```
s32 Ql_PWM_Init(Enum_PinName pwmPinName, Enum_PwmSource pwmSrcClk, u32 freq, u32 dutyCyclePercent)
```

- **Parameter**

pwmPinName:
[In] The pin name that supports the PWM function. Currently only PINNAME_SPI_SCLK, PINNAME_GPIO4 and PINNAME_MAIN_RTS support PWM multiplexing.

pwmSrcClk:
[In] PWM source clock. Please refer to the *Enum_PinName* in the SDK.

freq:
[In] The frequency of the PWM output. Range: 2–6500000. Unit: MHz.
dutyCyclePercent:
[In] The number of clock cycles to stay at High and low ratio. Range: 0-100.

- **Return Value**

0 Successful execution.
-2 The input pin is invalid.
-7 *freq* or *dutyCyclePercent* error.
-10 The input pin does not support PWM.
-12 The pin is already initialized.

4.8.4.4.2. Ql_PWM_Output

This function controls the output of the PWM waveform.

- **Prototype**

```
s32 Ql_PWM_Output(Enum_PinName pwmPinName,bool pwmOnOff)
```

- **Parameter**

pwmPinName:
[In] The pin name that supports the PWM function.

pwmOnOff:
[In] PWM output status.

TRUE	Turn on PWM waveform output
FALSE	Turn off PWM waveform output

- **Return Value**

0	Successful execution.
-2	The input pin is invalid.
-10	The input pin does not support PWM.
-13	The pin is not in PWM mode or not initialized.

4.8.4.4.3. Ql_PWM_UpdateDutyCycle

This function updates the duty cycle of the PWM waveform.

- **Prototype**

```
s32 Ql_PWM_UpdateDutyCycle(Enum_PinName pwmPinName, u32 dutyCyclePercent)
```

- **Parameter**

pwmPinName:
[In] The pin name that supports the PWM function.

dutyCyclePercent:
[In] The number of clock cycles to stay at high and low ratio. Range: 0-100.

- **Return Value**

0	Successful execution.
-2	The input pin is invalid.
-7	*dutyCyclePercent* error.
-10	The input pin does not support PWM.
-13	The pin is not in PWM mode or not initialized.

4.8.4.4.4. Ql_PWM_Uninit

This function releases the pins that have been initialized as PWM functions.

- **Prototype**

```
s32 Ql_PWM_Uninit(Enum_PinName pwmPinName)
```

- **Parameter**

pwmPinName:
[In] The pin name that supports the PWM function.

- **Return Value**

0 Successful execution.
-2 The input pin is invalid.
-10 The input pin does not support PWM.
-13 The pin is not in PWM mode or not initialized.

4.8.5. ADC

4.8.5.1. Header File

ql_adc.h file is the header file of the ADC API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.8.5.2. Function Overview of ADC

Table 11: Function Overview of ADC

Function	Description
Ql_ADC_Open()	Open an ADC channel
Ql_ADC_Read()	Read the ADC value of the specified pin
Ql_ADC_Close()	Close the ADC function of the specified pin
Ql_Get_Temperature()	Get the temperature value of the specified ADC pin

4.8.5.3. Usage

BC660K-GL QuecOpen module provides 2 pins that support ADC sampling function. The supported sampling range is 0–1200 mV. When the pin voltage exceeds 1200 mV, the sampling voltage will remain at 1200 mV.

4.8.5.4. API Description

4.8.5.4.1. Ql_ADC_Open

This function opens an ADC channel.

- **Prototype**

```
s32 Ql_ADC_Open(Enum_ADCPin adcPin)
```

- **Parameter**

adcPin:
[In] ADC pin name. Please refer to the *Enum_ADCPin* in the SDK.

- **Return Value**

0 Successful execution.
-2 The input pin is invalid.
-38 The ADC channel is not opened.

4.8.5.4.2. Ql_ADC_Read

This function reads the ADC value of the specified pin.

- **Prototype**

```
s32 Ql_ADC_Read(Enum_ADCPin adcPin,u32 *adcValue)
```

- **Parameter**

adcPin:
[In] ADC pin name. Please refer to the *Enum_ADCPin* in the SDK.

adcValue:
[out] ADC value. Unit: mV.

- **Return Value**

0 Successful execution.
-2 The input pin is invalid.
-36 ADC channel conversion stop.
-37 ADC channel conversion timeout.

4.8.5.4.3. Ql_ADC_Close

This function closes the ADC function of the specified pin.

- **Prototype**

```
s32 Ql_ADC_Close(Enum_ADCPin adcPin)
```

- **Parameter**

adcPin:
[In] ADC pin name. Please refer to the *Enum_ADCPin* in the SDK.

- **Return Value**

0	Successful execution.
-2	The input pin is invalid.
-38	The ADC channel is not opened.

4.8.5.4.4. Ql_Get_Temperature

This function gets the temperature value of the specified ADC pin.

- **Prototype**

```
s32 Ql_Get_Temperature(s32* tempValue)
```

- **Parameter**

tempValue:
[Out] Temperature value. Range: -40–85. Unit: °C.

- **Return Value**

0	Successful execution.
-1	Get failed.
-2	The input pin is invalid.
-37	ADC channel conversion timeout.
-39	The ADC channel conversion is running.

4.8.6. I2C

4.8.6.1. Header File

ql_i2c.h file is the header file of the I2C API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.8.6.2. Function Overview of I2C

Table 12: Function Overview of I2C

Function	Description
Ql_IIC_Init()	Initialize the I2C channel
Ql_I2C_Config()	Configure the I2C interface
Ql_I2C_Write()	Write data to a specified slave via I2C interface
Ql_I2C_Read()	Read data from specified slave via I2C interface
Ql_I2C_Write_Read()	Read the data from the specified register (or address) of slave device
Ql_I2C_Uninit()	Release the specified I2C channel

4.8.6.3. Usage

The BC660K-GL QuecOpen module supports 1 hardware I2C interface, and also supports analog I2C interface; analog I2C can use simulated by any two GPIO pins. To use the I2C function, including initialization, configuration, write operation and read operation, the steps are as follows:

Step 1: Initialize I2C interface. *Call Ql_I2C_Init()* to initialize an I2C channel, including GPIO pins for I2C function and I2C type.

Step 2: Configure I2C interface. Call *Ql_I2C_Config()* to configure communication parameters. See the API description for extended information.

Step 3: Read the data from slave. Call *Ql_I2C_Read()* to read data from the specified slave device. The following figure shows the direction of data exchange.

Step 4: Write data to slave. Call *Ql_I2C_Write()* to write data to the specified slave device. The following

figure shows the direction of data exchange.

Step 5: Write data into the designated register of the slave. Call *Ql_I2C_Write()* to write data into the specified register of the slave device. The figure below shows the direction of data exchange.

Step 6: Read the data from the specified register of the slave. Call *Ql_I2C_Write_Read()* to read data from the register of the slave device. The following figure shows the direction of data exchange.

Step 7: Release the I2C channel. Call *Ql_I2C_Uninit()* to release the specified I2C channel.

4.8.6.4. API Description

4.8.6.4.1. Ql_I2C_Init

This function initializes the I2C channel.

- **Prototype**

```
s32 Ql_IIC_Init(u32 chnnlNo,PinName pinSCL,PinName pinSDA, u32 IICtype)
```

- **Parameter**

chnnlNo:
[In] I2C channel number. Range: 0–254.

pinSCL:
[In] I2C SCL pin. Hardware I2C only supports I2C_SCL; analog I2C can use any available GPIO pins.

pinSDA:
[In] I2C SDA pin. Hardware I2C only supports I2C_SDA; analog I2C can use any available GPIO pins.

I2Ctype:
[In] I2C type.

0	Analog I2C, any available pins can be used.
1	For hardware I2C, only I2C_SCL and I2C_SDA pins can be used as clock and data lines respectively.

- **Return Value**

0 Successful execution.
-2 The channel number is be used.
-4 Not supported.
-21 Unsupported channel.
-22 The channel is already in use.
-23 Exceeding the maximum total.
-24 Invalid pin.
-25 Hardware error.
-26 Hardware I2C initialization failed.

4.8.6.4.2. Ql_I2C_Config

This function configures the I2C interface.

- **Prototype**

```
s32 Ql_I2C_Config(u32 chnnlNo, bool isHost, Enum_I2cSpeed I2CSpeed)
```

- **Parameter**

chnnlNo:
[In] I2C channel number. Range: 0–254.

isHost:
[In] Operating mode. Only 1 (host mode) is supported.

I2CSpeed:
[In] Communication rate. The communication rate is just for hardware I2C controller, and the parameter can be ignored if using analog I2C.

- **Return Value**

0 Successful execution.
-2 The input pin is invalid.
-26 Hardware I2C initialization failed.
-27 Unsupported configuration parameter.
-28 The I2C channel ID does not exist.

4.8.6.4.3. Ql_I2C_Write

This function writes data to a specified slave via I2C interface.

- **Prototype**

```
s32 Ql_I2C_Write(u32 chnnlNo, u8 slaveAddr, u8 *pData, u32 len)
```

- **Parameter**

chnnlNo:
[In] I2C channel number. Range: 0–254.

slaveAddr:
[In] Slave address.

pData:
[In] Data to be written to the slave.

len:
[In] The length of the data to be written.

- **Return Value**

If write succeed, return the length of the write data.
-2 Parameter error.
-4 Not supported.
-24 Invalid pin.
-25 Hardware error.
-28 The I2C channel ID does not exist.

4.8.6.4.4. Ql_I2C_Read

This function reads data from specified slave via I2C interface.

- **Prototype**

```
s32 Ql_I2C_Read(u32 chnnlNo, u8 slaveAddr, u8 *pBuffer, u32 len)
```

- **Parameter**

chnnlNo:
[In] I2C channel number. Range: 0–254.

slaveAddr:
[In] Slave address.

pBuffer:
[In] Data to be read.

len:
[In] The length of the data to be read. If *I2Ctype* = 1, then 1 < *len* <8.

- **Return Value**

If read succeed, return the length of the read data.
-2 Parameter error.
-4 Not supported.
-24 Invalid pin.
-25 Hardware error.
-28 The I2C channel ID does not exist.

4.8.6.4.5. Ql_I2C_Write_Read

This function reads the data from the specified register (or address) of the slave device.

- **Prototype**

```
s32 Ql_I2C_Write_Read(u32 chnnlNo, u8 slaveAddr, u8 * pData, u32 wrtLen, u8* pBuffer, u32 rdLen)
```

- **Parameter**

chnnlNo:
[In] I2C channel number. Range: 0–254.

slaveAddr:
[In] Slave address.

pData:
[In] Data to be written.

wrtLen:
[In] The length of the data to be written. If *I2Ctype* = 1, then 1 < *wrtLen* <8.

pBuffer:
[In] Data to be read.

rdLen:
[In] The length of the data to be read. If *I2Ctype* = 1, then 1 < *rfLen* <8.

- **Return Value**

If no error, return the length of the read data.
-2 Parameter error.

-4 Not supported.
-25 Hardware error.
-28 The I2C channel ID does not exist.

4.8.6.4.6. Ql_I2C_Uninit

This function releases the specified I2C channel.

- **Prototype**

```
s32 Ql_I2C_Uninit(u32 chnnlNo)
```

- **Parameter**

chnnlNo:
[In] I2C channel number. Range: 0–254.

- **Return Value**

0 Successful execution.
-22 The channel is already in use.
-24 Invalid pin.
-28 The I2C channel ID does not exist.

4.8.7. SPI

4.8.7.1. Header File

ql_spi.h file is the header file of the SPI API, located in the directory *\PLAT\project\qcx212_0h00\quectel_project\quec_open\include*.

4.8.7.2. Function Overview of SPI

Table 13: Function Overview of SPI

Function	Description
Ql_SPI_Init()	Initialize the SPI channel
Ql_SPI_Config()	Configure the SPI interface
Ql_SPI_Write()	Write data to the specified slave via SPI interface

Ql_SPI_Read()	Read data from the specified slave device via SPI interface
Ql_SPI_WriteRead()	Read data from the slave device in a half-duplex mode
Ql_SPI_Unini()	Release the SPI pins

4.8.7.3. Usage

BC660K-GL QuecOpen module supports 1 hardware SPI interface, and also supports analog SPI interface; all available GPIO pins can be configured for analog SPI. Hardware SPI and analog SPI are used in the same way, and both SPIs need to be initialized and configured before reading and writing. When the SPI function is not required, the function can be called to release the corresponding pins.

Step 1: Initialize the SPI interface. Call *Ql_SPI_Init()* to initialize the configuration of the SPI channel, including the SPI channel number, the specified pins for SPI and SPI type.
Step 2: Configure parameters. Call *Ql_SPI_Config()* to configure the communication parameters for the SPI interface, including working mode, clock polarity, clock phase and communication rate.
Step 3: Write data to the slave device. Call *Ql_SPI_Write()* to write data to the specified slave device.
Step 4: Write and read data. Call *Ql_SPI_WriteRead()* to complete the write and read operations to the specified device.
Step 5: Release the SPI interface. Call *Ql_SPI_Uninit()* to release the specified SPI interface.

4.8.7.4. API Description

4.8.7.4.1. Ql_SPI_Init

This function initializes the SPI channel.

- **Prototype**

```
s32 Ql_SPI_Init(u32 chnnlNo, Enum_PinName pinClk, Enum_PinName pinMiso, Enum_PinName pinMosi, Enum_PinName pinCs, bool spiType)
```

- **Parameter**

chnnlNo:
[In] SPI channel number. Range: 0–254.

pinClk:
[In] SPI CLK pin. Hardware SPI only supports PINNAME_SPI_SCLK, and analog SPI can be any available GPIO pin.

pinMiso:
[In] SPI MISO pin. Hardware SPI only supports PINNAME_SPI_MISO, and analog SPI can be any available GPIO pin.

pinMosi:
[In] SPI MOSI pin. Hardware SPI only supports PINNAME_SPI_MISO, and analog SPI can be any available GPIO pin.

pinCs:
[In] SPI CS pin. Hardware SPI only supports PINNAME_SPI_MISO, and analog SPI can be any available GPIO pin.

spiType:
[In] SPI type.
0 Analog SPI
1 Hardware SPI

- **Return Value**

0 Successful execution.
-2 Parameter error.
-4 Unsupported features.
-7 The current operation is invalid.
-46 Invalid pins.
-47 Hardware initialization failed.

4.8.7.4.2. Ql_SPI_Config

This function configures the SPI interface.

- **Prototype**

```
s32 Ql_SPI_Config(u32 chnnlNo, bool isHost, bool cpol, bool cpha, u32 clkSpeed)
```

- **Parameter**

chnnlNo:
[In] SPI channel number. Range: 0–254.

isHost:
[In] Working mode. Only support mode 1 (host mode).

cpol:
[In] Clock polarity.

cpha:
[In] Clock phase.

clkSpeed:
[In] Communication rate (only applicable to hardware SPI). Range: 1–52. Unit: MHz.

- **Return Value**

0 Successful execution.
-2 Parameter error.
-4 Unsupported features.
-7 The current operation is invalid.
-53 Unsupported channel.

4.8.7.4.3. Ql_SPI_Write

This function writes data to the specified slave via SPI interface.

- **Prototype**

```
s32 Ql_SPI_Write(u32 chnnlNo,u8 * pData,u32 len)
```

- **Parameter**

chnnlNo:
[In] SPI channel number. Range: 0–254.

pData:
[In] Data to be written.

len:
[In] The length of the data to be written.

- **Return Value**

0 Successful execution.
-2 Parameter error.
-4 Unsupported features.
-7 The current operation is invalid.
-53 Unsupported channel.

4.8.7.4.4. Ql_SPI_Read

This function reads data from the specified slave device via SPI interface.

- **Prototype**

```
s32 Ql_SPI_Read(u32 chnnlNo, u8* pBuffer, u32 rdLen)
```

- **Parameter**

chnnlNo:
[In] SPI channel number. Range: 0–254.

pBuffer:
[Out] Read buffer of reading data from slave device.

len:
[In] The length of the data to be read.

- **Return Value**

0	Successful execution.
-2	Parameter error.
-4	Unsupported features.
-7	The current operation is invalid.
-53	Unsupported channel.

4.8.7.4.5. Ql_SPI_WriteRead

This function reads data from the slave device in a half-duplex mode.

- **Prototype**

```
s32 Ql_SPI_WriteRead(u32 chnnlNo, u8 *pData, u32 wrtLen, u8 * pBuffer, u32 rdLen)
```

- **Parameter**

chnnlNo:
[In] I2C channel number. Range: 0–254.

pData:
[In] Data to be written.

wrtLen:
[In] The length of the data to be written.

pBuffer:
[In] Data to be read.

rdLen:
[In] The length of the data to be read.

- **Return Value**

0 Successful execution.
-2 Parameter error.
-4 Unsupported features.
-7 The current operation is invalid.
-53 Unsupported channel.

NOTE

1. When the length of the written data is greater than the length of the read data, the read data with the extra length will be set as 0xFF.
2. When the length of the read data is greater than the length of the write data, the write data with the excess length will be set as 0xFF.

4.8.7.4.6. Ql_SPI_Uninit

This function releases the SPI pins.

- **Prototype**

```
s32 Ql_SPI_Uninit(u32 chnnlNo)
```

- **Parameter**

chnnlNo:
[In] SPI channel number. Range: 0–254.

- **Return Value**

0 Successful execution.
-2 Parameter error.
-4 Unsupported features.
-7 The current operation is invalid.
-53 Unsupported channel.

4.9. RIL API

4.9.1. Header File

ql_ril.h file is the header file of the time API, located in the directory *PLAT\project\qcx212_0h00\quectel_project\quec_open\ril\inc*.

4.9.2. Function Overview of RIL

Table 14: Function Overview of RIL

Function	Description
Ql_RIL_Initialize()	Initialize the RIL channel
Ql_RIL_SendATCmd()	Send AT commands

4.9.3. Usage

The BC660K-GL QuecOpen module sends AT commands via the RIL channel to communicate with the kernel. The RIL channel needs to be initialized before sending the AT commands.

Step 1: After the main task receives the MSG_ID_RIL_READY message, call *Ql_RIL_Initialize()* to complete the initialization of the RIL interface.

Step 2: After the initialization is completed, call *Ql_RIL_SendATCmd()* to send the AT command to the kernel and process the return data of the AT command in the interface callback.

4.9.4. API Description

4.9.4.1. Ql_RIL_Initialize

This function initializes the RIL channel.

- **Prototype**

```
extern void Ql_RIL_Initialize(void)
```

- **Parameter**

Void.

- **Return Value**

Void.

NOTE

Execute *Ql_RIL_Initialize()* before sending the AT command.

4.9.4.2. Ql_RIL_SendATCmd

This function sends AT commands.

- **Prototype**

```
s32 Ql_RIL_SendATCmd(char* atCmd, u32 atCmdLen, Callback_ATResponse atRsp_callBack, void* userData, u32 timeOut)
```

```
typedef void (*Callback_ATResponse)(char* line, u32 len, void* userData)
```

- **Parameter**

atCmd:
[In] AT command to be sent.

atCmdLen:
[In] The length of AT command.

atRsp_callBack:
[In] Callback function.

userData:
[In/Out] Customized parameters, pass in the null pointer type parameter of the callback.

timeOut:
[In] Timeout for the Interface. Unit: ms. If it is set to 0, the default value is 3 minutes.

- **Return Value**

0 Execute AT command successfully, and the response is **OK**.
-1 Failed to execute AT command, or the response is **ERROR**.
-2 Send AT command timeout.
-3 Sending command.

-4 Invalid input parameter.
-5 RIL channel is not initialized

NOTES

1. This function must be called after the RIL channel has been initialized.
2. This function is a synchronous function and cannot be called until the command execution result is returned.

5 Appendix A References

Table 15: Related Documents

SN	Document Name	Description
[1]	Quectel_BC660K-GL_QuecOpen_Quick_Start_Guide	BC260Y-CN QuecOpen module quick start guide
[2]	Quectel_BC660K-GL_AT_Commands_Manual	BC660K-GL Module AT Commands Manual
[3]	Quectel_BC660K-GL_QuecOpen_Hardware_Design	BC260Y-CN QuecOpen module hardware design

Table 16: Terms and Abbreviations

Abbreviation	Description
ADC	Analog-to-digital Converter
API	Application Programming Interface
App	Application
EINT	External Interrupt
EPS	Evolved Packet System
E-UTRAN	Evolved Universal Terrestrial Radio Access Network
DFOTA	Delta Firmware Upgrade Over-the-Air
GPIO	General-purpose Input/Output
I2C	Inter-Integrated Circuit
ICCID	Integrated Circuit Card Identifier
ID	Mostly refers to Identifier in terms of software
IMEI	International Mobile Equipment Identity

IMSI	International Mobile Subscriber Identity
I/O	Input/Output
IoT	Internet of Things
ISR	Interrupt Service Routine
M2M	Machine to Machine
MCU	Microcontroller Control Unit
NB-IoT	Narrowband Internet of Things
OS	Operating System
PSM	Power Saving Mode
PWM	Pulse Width Modulation
RAM	Random Access Memory
RIL	Radio Interface Layer
RSRP	Reference Signal Received Power
RTC	Real-Time Clock
SDK	Software Development Kit
SIM	Subscriber Identity Module
SN	Serial Number
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
TAU	Tracking Area Update.
UART	Universal Asynchronous Receiver/Transmitter
URC	Unsolicited Result Code
UTC	Coordinated Universal Time