BC26-OpenCPU

快速开发指导

LPWA 模块系列

版本：Quectel_BC26-OpenCPU_快速开发指导_V1.0

日期：2019-03-12

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
上海市徐汇区虹梅路 1801 号宏业大厦 7 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。由于客户操作不当而造成的人身伤害或财产损失，本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。

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。

文档历史

修订记录

版本	日期	作者	变更表述
1.0	2019-03-12	梁维	初始版本

目录

表格索引

图表索引

1 基本概述

本文档介绍如何使用 OpenCPU 软件 SDK 包来快速开始 BC26-OpenCPU 项目的开发。此外，本文还介绍了应用程序的设计和编程的注意事项。

2 OpenCPU 相关文档

表 1：OpenCPU 相关文档

文档名	介绍
Quectel_BC26-OpenCPU_User_Guide	介绍 OpenCPU 平台及项目相关的 API 接口
Quectel_OpenCPU_RIL_Application_Note	文档介绍如何通过 RIL 来封装 RIL 接口，处理 AT 命令的返回值
Quectel_BC26-OpenCPU_DFOTA_应用指导	文档介绍了如何在 OpenCPU 方案中使用 DFOTA
Quectel_BC26-OpenCPU_硬件设计手册	文档定义了 BC26-OpenCPU 模块及其与客户应用连接的空中接口和硬件接口。
Quectel_BC26-OpenCPU_Genie_Log_抓取操作指导	文档介绍了 OpenCPU 方案中如何抓取 Genie Log

3 开始前准备

在使用 OpenCPU 之前，您需要确认是否有如下所列的软件和硬件组件。

3.1. 主机系统

支持以下任意一种主机操作系统和体系结构：

- Microsoft Windows XP
- Windows Vista
- Windows 7 32 bit 或 64 bit
- Windows 10 32 bit 或 64 bit

3.2. 编译器

- GCC 编译器 (Sorcery CodeBench Lite for ARM EABI) 我们已经默认安装在 tools 文件夹下。
- DOS 命令行。

3.3. 编程语言

必须具备基本的 C 语言编程知识，最好能具备一定的多任务操作系统经验。

3.4. 模块硬件

- Quectel BC26-OpenCPU 模块
- Quectel BC26-TE-B
- 其他如 USB 线等

如需如上所述硬件资源，可联系移远通信技术支持。

3.5. OpenCPU SDK

- OpenCPU SDK 软件包

您可以从移远通信技术支持 support@quectel.com 获取软件包。

- App 下载工具

您可以从 SDK 包中的 *tools* 文件夹中获取。

4 编译

本章将介绍如何在命令行中编译 SDK。

4.1. 编译

在 OpenCPU 中，用户只需要通过执行如下命令即可编译 SDK 代码：

```
make clean
make new
```

4.2. 编译输出

编译期间将输出一些编译处理信息在命令行中。所有的警告和错误都会保存在 *build\gcc\build.log* 中，用户可以根据 log 中的错误行和错误提示，快速排查代码错误。

5 下载

5.1. TE-B

如果您使用的是我司 TE-B 套件，请通过 TE-B 上的主串口（CHA）下载固件或者 APP Bin。

5.2. 用户设备

如果模块已焊接到用户设备主板上，请通过 UART0（RXD、TXD）下载固件或者 APP Bin。

5.3. 量产

为提高客户生产效率，移远通信提供了专用夹具和下载工具，可以同时给多个模块下载固件/APP Bin。如有需要，请咨询移远通信技术支持。

6 调试

在 OpenCPU 应用中，通过串口打印跟踪 Log 是主要的调试方法。

BC26-OpenCPU 模块提供三个串口：主串口（UART0）、Debug 串口(UART2)和 AUX 串口（UART1）。在程序中，客户可以调用 Ql_UART_Open 来打开上述端口，并调用 Ql_UART_Write 来输出调试消息。

如果模块出现异常重启、Dump 或者网络业务异常，可以通过上述三个串口或者 Debug USB 口来抓取 Genie Log，具体操作流程请参考***文档 [1]***。

7 关于 OpenCPU SDK

解压缩 SDK 包以后，可查看其目录结构；以 *BC26_OpenCPU_NB1_SDK_V1.3* 为例，其典型的目录层次结构如下：

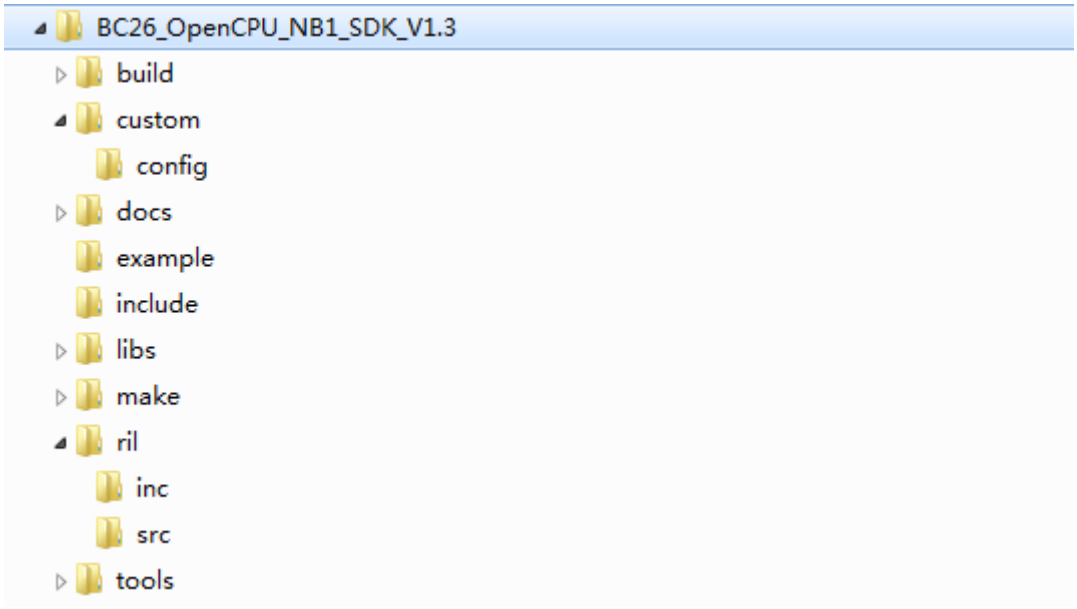

图 1：BC26_OpenCPU_NB1_SDK 目录层次结构

表 2：BC26_OpenCPU_NB1_SDK 目录说明

目录名	描述
BC26_OpenCPU_NB1_SDK_V1.3	OpenCPU 的根目录
build	输出编译的目录； 用户可以看到编译生成的 APP Bin 和编译的 Log 信息。
custom	此目录被设计为项目的根目录； 在子目录 *custom\config*中，可根据需要重新配置应用程序，例如创建任务和任务的堆栈大小、GPIO 初始状态等；用户可以在此目录下创建属于自己的项目结构，并加载到 *Makefile* 中。
docs	存放 OpenCPU 项目相关的文件
example	此目录存放示例代码； 每个示例文件实现了一个独立的功能；每个示例文件都可以编译成一

	个可执行的 APP Bin。
include	存放 Ql 开头 API 接口的头文件
libs	存放 OpenCPU 链接的 lib 库
make	*Makefile* 文件在这个目录中。
ril	存放 OpenCPU RIL 的开源代码； 用户可以轻松地基于 RIL 添加一个新的 API 来处理 AT 命令。
tools	存放 GCC 的免安装包和 QFlash 下载工具包

8 创建用户项目

一般情况下，目录 *sdk\custom*被设计为项目的根目录。在这个目录中，放置了一个程序文件 *main.c* 用于演示 OpenCPU 程序的主要程序框架。

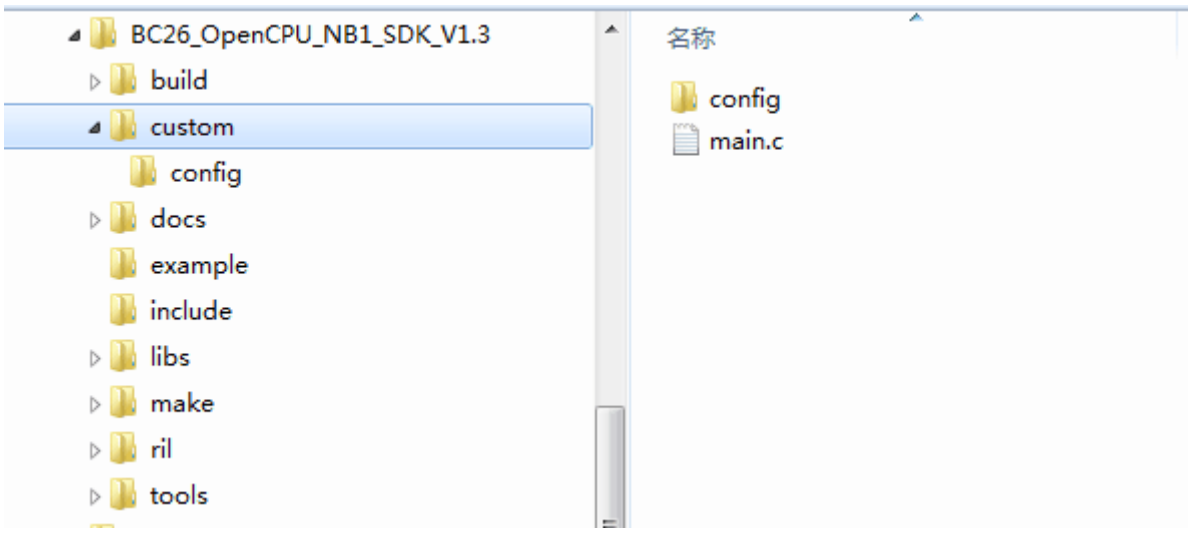

图 2：custom 用户目录

在 *sdk\custom*目录中，可以添加其他模块文件和子目录；具体操作请参考***文档 [2]*** 中的***第 2.5.3 章***。

所有源代码文件都在 *makefile* (*\SDK\make\gcc\gcc_makefile*)下管理；客户可以决定哪些目录以及哪些源代码文件需要在此生成文件中编译；具体操作可参考***文档 [2]*** 中的***第 2.5.3 章***。

另外，*main.c* 工程已被默认加载，用户只需在 *main.c* 中添加代码或者更改 *main.c* 中现有的代码，即可将 *main.c* 用于演示串口的交互。此外，用户还可以添加其他.c 文件，*makefile* 默认会自动编译 *sdk\custom* 中所有新添加的.c 文件。

9 快速编程

本章主要演示如何控制 GPIO 的引脚电平来驱动 LED 灯的亮、灭，让用户对 *OpenCPU SDK* 方案达成初步了解。

用户可以用如下的示例代码来覆盖 *sdk\custom\main.c*，或者删除 *main.c* 并创建一个新的.c 文件以实现通过对 GPIO 引脚电平的控制而控制 LED 等的亮与灭。

9.1. GPIO 控制

9.1.1. 确认需求的头文件

要确认需要哪些头文件，用户需要先了解这个应用程序的需求。对于 GPIO 控制这个应用程序，基本的需求是：通过周期更改 GPIO 电平状态来实现 LED 闪烁。

- 首先，用户需要控制一个 GPIO；相关的 API 和变量定义在 *ql_gpio.h* 中。
- 其次，“周期性”意味着需要一个计时器；相关定义在 *ql_timer.h* 中。
- 最后，应用程序中需要处理 timer 和 UART 的消息，因此 *ql_system.h* 是必须的。此外，还需要打印一些日志信息来调试程序，相关的头文件是 *ql_stdlib.h*、*ql_uart.h* 和 *ql_trace.h*。API 函数的所有返回值都在 *ql_error.h* 中定义。

因此，用户需要包括的头文件如下：

```
#include "ql_stdlib.h"
#include "ql_trace.h"
#include "ql_error.h"
#include "ql_system.h"
#include “ql_uart.h”
#include "ql_gpio.h"
#include "ql_timer.h"
```

9.1.2. 控制 GPIO

在 BC26-TE-B 上，NETLIGHT 引脚已经连接到一个 LED 上，这意味着用户可以控制 NETLIGHT 来实现 LED 闪烁。

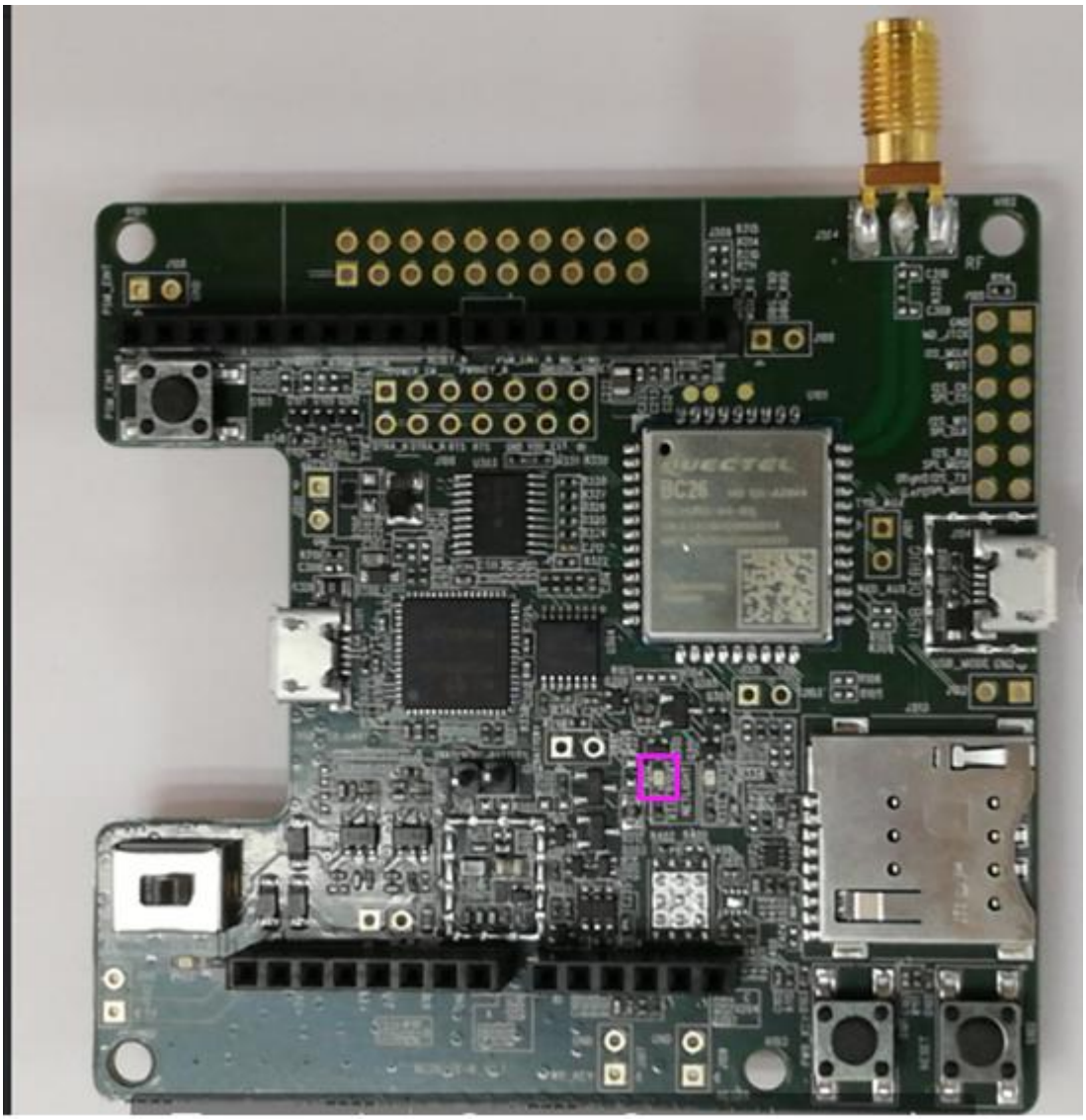

图 3：BC26-TE-B 的 LED 指示灯

步骤一：在程序中配置 GPIO 引脚

```
//Define GPIO pin
static Enum_PinName m_gpioPin = PINNAME_NETLIGHT;
```

步骤二：按照如下方式初始化 GPIO

- “输入/输出”状态初始化为“**输出**”
- “初始电平”状态初始化为“**低电平**”
- “上下拉状态”状态初始化为“**上拉**”

```
//Initialize GPIO
ret = Ql_GPIO_Init(m_gpioPin, PINDIRECTION_OUT, PINLEVEL_LOW, PINPULLSEL_PULLUP);
```

```
if (QL_RET_OK == ret)
{
    APP_DEBUG ("<-- Initialize GPIO successfully -->\r\n");
}else{
    APP_DEBUG ("<-- Fail to initialize GPIO pin, cause=%d -->\r\n", ret);
}
```

接下来，用户需要启动一个计时器，并定期更改 GPIO 口的电平从而实现 LED 闪烁。

9.1.3. 时间和 LED 灯控制

定义一个 500ms 超时的计时器，以控制 LED 亮 500ms、暗 500ms。

步骤一：定义一个计时器和计时器中断处理程序

```
//Define a timer and the handler
static u32 m_myTimerId = 2019;
static u32 m_nInterval = 500;     //500ms
static void Callback_OnTimer(u32 timerId, void* param);
```

步骤二：注册一个 Timer 并开始

```
//Register and start timer
Ql_Timer_Register(m_myTimerId, Callback_OnTimer, NULL);
Ql_Timer_Start(m_myTimerId, m_nInterval, TRUE);
```

步骤三：实现 Timer 的中断处理程序

```
static void Callback_OnTimer(u32 timerId, void* param)
{
    s32 gpioLvl = Ql_GPIO_GetLevel(m_gpioPin);
    if (PINLEVEL_LOW == gpioLvl)
    {
        // Set GPIO to high level, then LED is light
        Ql_GPIO_SetLevel(m_gpioPin, PINLEVEL_HIGH);
        APP_DEBUG ("<-- Set GPIO to high level -->\r\n");
    }else{
        // Set GPIO to low level, then LED is dark
        Ql_GPIO_SetLevel(m_gpioPin, PINLEVEL_LOW);
        APP_DEBUG ("<-- Set GPIO to low level -->\r\n");
    }
}
```

至此，所有的编程工作均已完成，完整代码如下：

```
#include "ql_stdlib.h"
#include "ql_trace.h"
#include "ql_error.h"
#include "ql_system.h"
#include "ql_gpio.h"
#include "ql_timer.h"
#include "ql_uart.h"

// Define APP DEBUG
    #define DEBUG_ENABLE 1
    #if DEBUG_ENABLE > 0
    #define DEBUG_PORT    UART_PORT0
    #define DBG_BUF_LEN     512
    static char DBG_BUFFER[DBG_BUF_LEN];
    #define APP_DEBUG(FORMAT,...) {\
        Ql_memset(DBG_BUFFER, 0, DBG_BUF_LEN);\
        Ql_sprintf(DBG_BUFFER,FORMAT,##__VA_ARGS__); \
        if (UART_PORT2 == (DEBUG_PORT)) \
        {\
            Ql_Debug_Trace(DBG_BUFFER);\
        } else {\
            Ql_UART_Write((Enum_SerialPort)(DEBUG_PORT),                (u8*)(DBG_BUFFER),
Ql_strlen((const char *)(DBG_BUFFER)));\
        }\
    }
    #else
    #define APP_DEBUG(FORMAT,...)
    #endif

    static Enum_SerialPort m_myUartPort    = UART_PORT0;

// Define GPIO pin
static Enum_PinName m_gpioPin = PINNAME_NETLIGHT;

// Define a timer and the handler
static u32 m_myTimerId = 2019;
static u32 m_nInterval = 500;     // 500ms
static void Callback_OnTimer(u32 timerId, void* param);
static void CallBack_UART_Hdlr(Enum_SerialPort port, Enum_UARTEventType msg, bool level, void*
customizedPara);
/*****************************************************************************/
/* The entrance procedure for this example application                       */
/*****************************************************************************/
void proc_main_task(s32 taskId)
```

```
{
    s32 ret;
    ST_MSG msg;

        // Register & open UART port
        ret = Ql_UART_Register(m_myUartPort, CallBack_UART_Hdlr, NULL);
        if (ret < QL_RET_OK)
        {
            Ql_Debug_Trace("Fail to register serial port[%d], ret=%d\r\n", m_myUartPort, ret);
        }
        ret = Ql_UART_Open(m_myUartPort, 115200, FC_NONE);
        if (ret < QL_RET_OK)
        {
            Ql_Debug_Trace("Fail to open serial port[%d], ret=%d\r\n", m_myUartPort, ret);
    }

    APP_DEBUG("OpenCPU: LED Blinking by NETLIGH\r\n");

    // Initialize GPIO
    ret = Ql_GPIO_Init(m_gpioPin, PINDIRECTION_OUT, PINLEVEL_LOW,
            PINPULLSEL_PULLUP);
    if (QL_RET_OK == ret)
    {
        APP_DEBUG ("<-- Initialize GPIO successfully -->\r\n");
    }else{
        APP_DEBUG ("<-- Fail to initialize GPIO pin, cause=%d -->\r\n", ret);
    }

    // Register and start timer
    Ql_Timer_Register(m_myTimerId, Callback_OnTimer, NULL);
    Ql_Timer_Start(m_myTimerId, m_nInterval, TRUE);

    // START MESSAGE LOOP OF THIS TASK
    while(TRUE)
    {
        Ql_OS_GetMessage(&msg);
        switch(msg.message)
        {
        default:
            break;
        }
    }
}
```

```
static void Callback_OnTimer(u32 timerId, void* param)
{
    s32 gpioLvl = Ql_GPIO_GetLevel(m_gpioPin);
    if (PINLEVEL_LOW == gpioLvl)
    {
        // Set GPIO to high level, then LED is light
        Ql_GPIO_SetLevel(m_gpioPin, PINLEVEL_HIGH);
        APP_DEBUG ("<-- Set GPIO to high level -->\r\n");
    }else{
        // Set GPIO to low level, then LED is dark
        Ql_GPIO_SetLevel(m_gpioPin, PINLEVEL_LOW);
        APP_DEBUG ("<-- Set GPIO to low level -->\r\n");
    }
}

    static void CallBack_UART_Hdlr(Enum_SerialPort port, Enum_UARTEventType msg, bool level, void* customizedPara)
    {
    }
```

9.1.4. 运行 APP Bin

用户可以将完整的代码复制到 *sdk\custom\main.c* 以覆盖现有的代码，并将 APP Bin 编译并下载到模块。

当应用程序运行时，可以看到 TE-B 上的 **D303** LED 在 500ms 的时间内闪烁，同时可以看到主串口输出以下调试信息。

```
[2019-01-23_13:14:08:624]OpenCPU: LED Blinking by NETLIGH
[2019-01-23_13:14:08:624]<-- Initialize GPIO successfully -->
[2019-01-23_13:14:09:120]<-- Set GPIO to high level -->
[2019-01-23_13:14:09:620]<-- Set GPIO to low level -->
[2019-01-23_13:14:10:120]<-- Set GPIO to high level -->
[2019-01-23_13:14:10:620]<-- Set GPIO to low level -->
[2019-01-23_13:14:11:120]<-- Set GPIO to high level -->
[2019-01-23_13:14:11:621]<-- Set GPIO to low level -->
[2019-01-23_13:14:12:120]<-- Set GPIO to high level -->
[2019-01-23_13:14:12:621]<-- Set GPIO to low level -->
[2019-01-23_13:14:13:121]<-- Set GPIO to high level -->
```

10 注意事项

10.1. Light Sleep 模式

当模块进入 Light Sleep 模式，用户需要先发一包两个字节（比如 **AT**）的数据唤醒模块，然后再进行数据业务。

10.2. Deep Sleep 模式

模块进入 Deep Sleep 模式后，CPU 会掉电，这时候仅仅只有 RTC 还在运行；如果用户需要保存掉电不丢失的数据，可以使用 API 接口（Ql_SecureData_Read、Ql_SecureData_Store）。模块唤醒后，程序会重新加载，代码重新运行。

10.3. 串口

BC26-OpenCPU 模块提供三个串行端口。默认配置参数为：115200bps 波特率。UART 端口的数据缓冲区大小为 1400B，用户不要发送超过 1400 字节的数据到模块。

当收到 UART 回调中的事件 EVENT_UART_READY_TO_READ 时，应用程序应调用 Ql_UART_Read 从 UART 缓冲区中读取所有数据。

10.4. Timer（定时器）

在 OpenCPU 中，有两种定时器：普通定时器和快速定时器。每个任务有 10 个普通定时器，整个应用程序只有一个快速定时器。普通定时器会由于任务阻塞而延迟，因此普通定时器是任务定时器；而快速计时器不属于任何任务，是由中断触发，因此，快速定时器具有良好的实时性。但是，请不要在中断处理程序中放置太多的工作负载，否则会导致系统异常。

10.5. 动态内存

可以调用 Ql_MEM_Alloc 来指定动态内存的大小，也可以调用 Ql_MEM_Free 来释放内存。应用程序最大可以申请 300KB 的动态内存。

11 附录 A 参考文档

11.1. 参考文档

表 3：参考文档

序号	文档名称	备注
[1]	Quectel_BC26-OpenCPU_Genie_Log_抓取操作指导	文档介绍了 OpenCPU 方案中如何抓取 Genie Log
[2]	Quectel_BC26-OpenCPU_User_Guide	介绍 OpenCPU 平台及项目相关的 API 接口