BC20-OpenCPU

快速开发指导

NB-IoT 模块系列

版本：BC20-OpenCPU_快速开发指导_V1.0

日期：2019-09-26

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。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。

版权所有 ©上海移远通信技术股份有限公司 2019，保留一切权利。
Copyright © Quectel Wireless Solutions Co., Ltd. 2019.

文档历史

修订记录

版本	日期	作者	变更表述
1.0	2019-09-26	顾雪峰	初始版本

目录

表格索引

图表索引

1 基本概述

本文档主要介绍如何使用 OpenCPU SDK 软件包快速开始移远通信 BC20 模块的 OpenCPU 方案开发。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

文档名称	介绍
Quectel_BC20-OpenCPU_用户指导	介绍 BC20-OpenCPU 模块相关 API 接口的说明和使用指导
Quectel_BC26&BC20-OpenCPU_RIL_应用指导	介绍如何通过 RIL 来封装 RIL 接口，处理相关项目 AT 命令的返回值
Quectel_BC26&BC20-OpenCPU_DFOTA_应用指导	介绍如何在 OpenCPU 方案中使用 DFOTA
Quectel_BC20-OpenCPU_硬件设计手册	定义 BC20-OpenCPU 模块及其与客户应用连接的空中接口和硬件接口
Quectel_BC20-OpenCPU_Genie_Log_抓取操作指导	介绍在 OpenCPU 方案中如何抓取 Genie Log

3 开发准备

在使用 OpenCPU 方案之前，需要确认是否具备以下章节所列的软件和硬件组件。

3.1. 主机系统

开发系统须为以下任意一种主机操作系统：

- 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），默认已安装在 *tools* 文件夹下。
- DOS 命令行

3.3. 编程语言

开发者必须具备基本的 C 语言编程知识，并建议具备一定的多任务操作系统经验。

3.4. 模块硬件

- 移远通信 BC20-OpenCPU 模块
- 移远通信 BC20-TE-B
- 其他（如 USB 线等）

如需上述硬件资源，可联系移远通信技术支持。

3.5. OpenCPU SDK

- OpenCPU SDK 软件包

请联系移远通信技术支持（support@quectel.com）获取所述软件包。

- 固件下载工具

用户可以从 SDK 包中的 *tools* 文件夹内获取所述固件下载工具。

4 编译

本章介绍如何在命令行中编译 SDK 代码。

4.1. 编译

在 OpenCPU 中，执行如下命令即可编译 SDK 代码：

```
make clean
make new
```

4.2. 编译输出

编译期间将在命令行输出一些编译处理信息。所有的警告和错误均会保存在 *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。

如果编译过程中出现任何编译错误，可从 *build.log* 文件中获取错误行号和相应的错误提示。

5 下载

5.1. 通过 TE-B 下载

若通过 BC20-TE-B 套件进行固件或 APP Bin 文件下载，则请通过 TE-B 上的调试串口（Interface 0）进行下载。

5.2. 通过用户设备下载

若模块已焊接到用户设备主板上，请通过模块的调试串口（TXD_DBG、RXD_DBG）下载固件或者 APP Bin。

5.3. 量产项目的软件下载

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，可以一次性将固件和 APP Bin 文件下载到多个模块。如有需要，请咨询移远通信技术支持。

6 调试

在 OpenCPU 应用开发过程中，主要通过串口打印跟踪 Log 进行调试。

BC20-OpenCPU 模块提供三个可配置的串口：主串口（UART1）、调试串口（UART0）和辅助串口（UART3）。在 OpenCPU 应用程序中，可通过调用 *Ql_UART_Open* 打开上述端口，调用 *Ql_UART_Write* 输出调试消息。

如果模块出现异常重启、死机或者网络业务异常，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 OpenCPU SDK 目录结构

获取 SDK 包后解压文件将可查看 OpenCPU SDK 软件包的目录结构，其典型的目录层次结构如下：

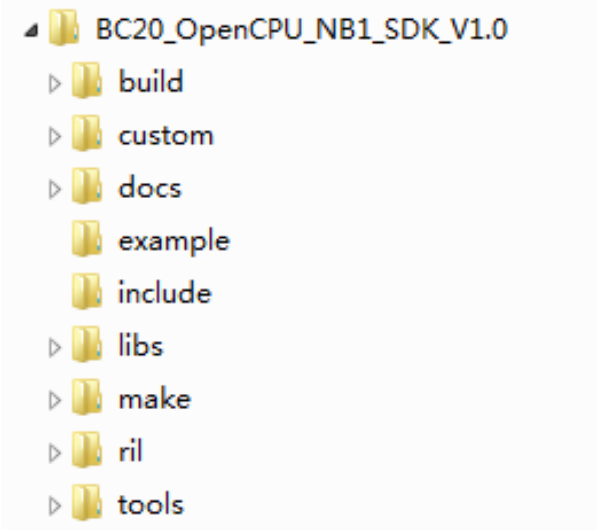

图 1：BC20_OpenCPU_NB1_SDK 目录层次结构

表 2：BC20_OpenCPU_NB1_SDK 目录说明

目录名	描述
BC20_OpenCPU_NB1_SDK	OpenCPU 的根目录
build	输出编译的接口，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 BC20-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	存放以 QI 开头的 API 接口的头文件。
libs	存放 OpenCPU 链接的 lib 库。

make	存放 *Makefile* 文件。
ril	存放 OpenCPU RIL 的开源代码，基于 RIL 用户可以轻松地添加一个新的 API 来处理 AT 命令。
tools	存放 GCC 的免安装包和 QFlash 工具包。

8 创建用户项目

通常，目录 *BC20_OpenCPU_NB1_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 应用的主要程序框架。

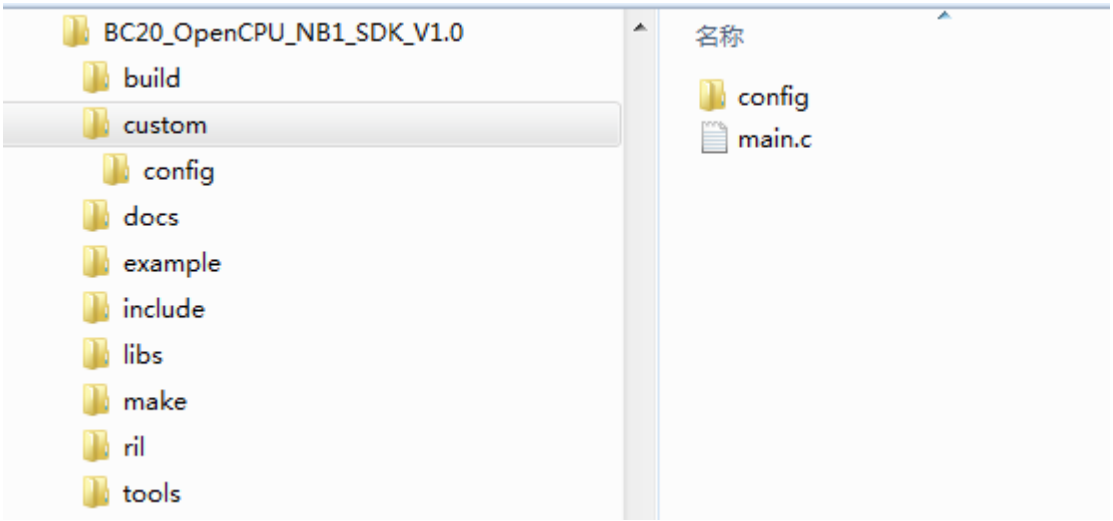

图 2：Custom 用户目录

在 *BC20_OpenCPU_NB1_SDK\custom*目录中，可以添加其他功能子目录；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（*BC20_OpenCPU_NB1_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***。

在 *BC20_OpenCPU_NB1_SDK\custom*目录下，已默认加载 main.c 工程；用户仅需在 *main.c* 中添加代码或者更改 *main.c* 中现有的代码，即可实现串口的交互。用户也可以添加其他.c 文件，*makefile* 默认会自动编译 *BC20_OpenCPU_NB1_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 引脚的电平来驱动 LED 灯的亮灭，并给出示例应用程序，以指导如何通过 OpenCPU SDK 进行编程。

用户可以使用***第9.1.4 章***所示的示例代码来覆盖 *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，从而控制 LED 灯的亮灭。

9.1. GPIO 控制

9.1.1. 确认需要的头文件

要确认需要哪些头文件（.h 文件），用户需要先了解这个应用程序的需求。对于示例应用程序，要求通过周期性更改 GPIO 电平状态实现 LED 灯的亮灭。

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”表明需要一个计时器；相关定义在 *ql_timer.h* 中。

最后，应用程序中需要处理计时器和串口的消息，因此 *ql_system.h* 是必需的。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 BC20-TE-B 上，模块的 NETLIGHT 引脚已被连接至如下图所示的 LED（D304）上；因此通过设置模块 NETLIGHT 引脚的电平，即可实现 LED 灯的亮灭控制。

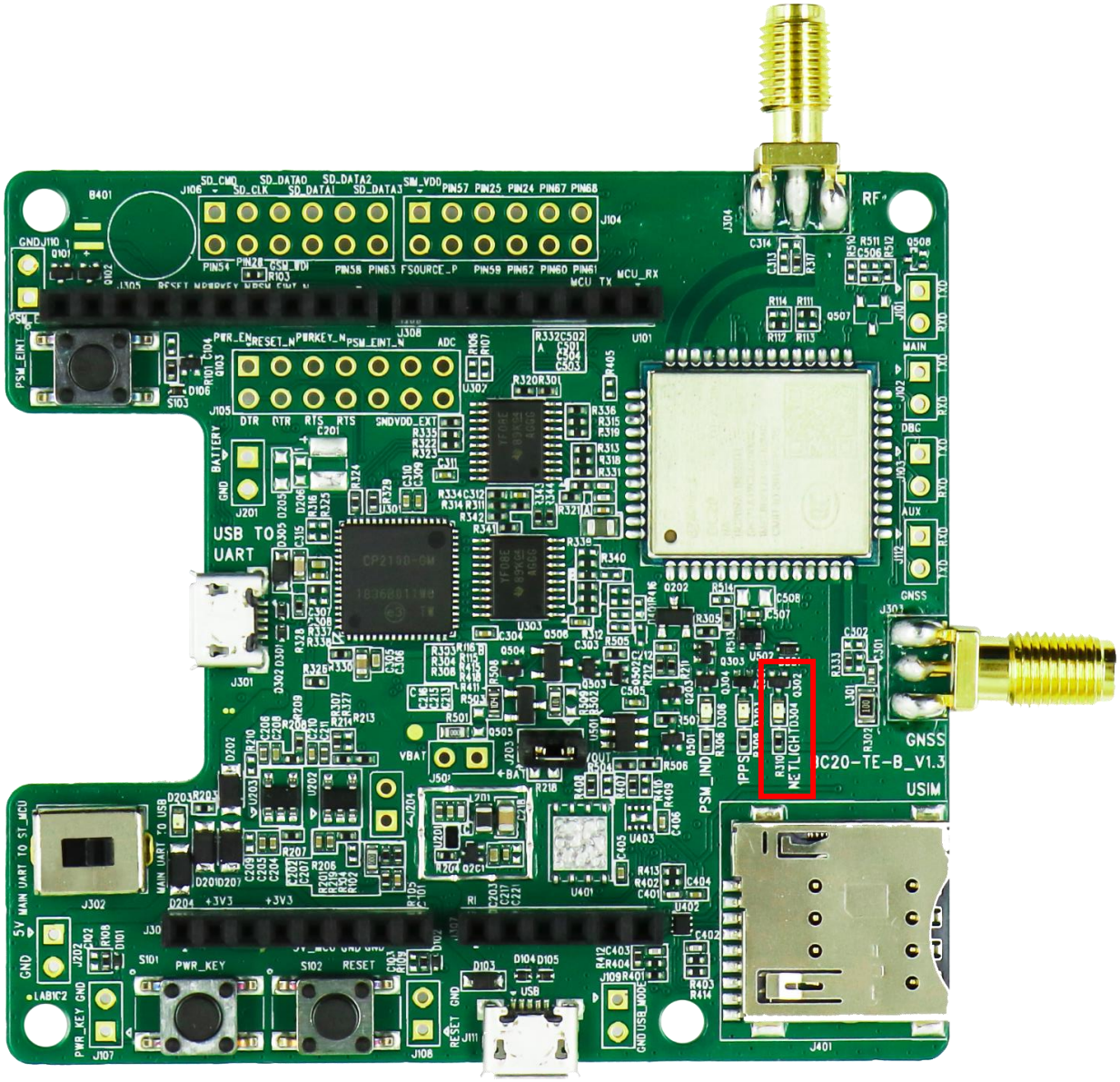

图 3：BC20-TE-B 的 LED 指示灯

步骤一：在程序中配置 GPIO 引脚，代码如下所示：

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

步骤二：按照如下方式初始化 GPIO 引脚状态：

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

以上步骤完成后，需启动一个计时器，并定期更改 GPIO 引脚的电平从而实现 LED 灯的亮灭；详细步骤请见随后的章节。

9.1.3. 定义计时器

定义一个 500ms 的超时计时器，以控制 LED 亮 500ms、灭 500ms。

步骤一：定义一个计时器和计时器中断处理程序，如下所示：

```
//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：注册一个计时器并开始计时，注册代码如下所示：

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

步骤三：实现计时器的中断处理程序，实现代码如下所示：

```
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");
    }
```

```
}
```

9.1.4. 完整示例代码

至此，所有的编程工作均已完成，完整示例代码如下所示：

```
#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)
        {
        }
```

9.1.5. 运行 APP Bin

将如下所示完整的代码复制到 *sdk\custom\main.c* 以覆盖现有的代码，并将 APP Bin 编译并下载到模块中，即可实现 APP Bin 文件的运行。

当运行应用程序时，可以看到 BC20-TE-B 上的 LED（D304）闪烁 500 秒，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 NETLIGHT
[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

本章介绍应用程序设计和编程过程中的主要注意事项。

10.1. Light Sleep 模式

当模块进入 Light Sleep 模式，用户需要先发送一组包含两个字节（比如 **AT**）的数据唤醒模块，然后再进行数据业务。

请注意目前仅支持通过主串口（UART1）发送数据，以唤醒模块。

实际应用中，当模块从 Deep Sleep 模式唤醒后，可调用 Ql_SleepDisable 禁止模块进入 Light Sleep 和 Deep Sleep 模式，以避免影响业务的处理；当业务流程处理完，再调用 Ql_SleepEnable 使能模块进入 Light Sleep 和 Deep Sleep 模式。

10.2. Deep Sleep 模式

模块进入 Deep Sleep 模式后，CPU 会掉电，此时仅 RTC 仍在运行。如需保存某些重要数据以防在 CPU 掉电后丢失，可以使用 API 接口（Ql_Flash_Read、Ql_Flash_Write）读取和保存最大达 4KB 的数据。模块从 Deep Sleep 模式唤醒后，程序会重新加载，代码重新运行。

10.3. 串口

BC20-OpenCPU 模块提供三个可配置的串行端口。UART 端口的数据缓冲区大小为 1400 字节，请不要发送超过 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，应用程序应及时调用 Ql_UART_Read 从 UART 缓冲区读取所有数据。

10.4. 定时器

在 OpenCPU 中，有两种定时器：普通定时器和快速定时器。每个任务最多支持 10 个普通定时器，整个 SDK 工程最多支持 6 个快速定时器。

普通定时器会由于任务阻塞无法及时处理定时器消息而延迟，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，由中断触发，因此快速定时器具有很高的实时性。但请不要在中断处理程序中处理太多的工作负载，否则可能会导致系统异常。

10.5. 动态内存

调用 Ql_MEM_Alloc 可指定动态内存的大小，也可调用 Ql_MEM_Free 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

表 3：参考文档

序号	文档名称	备注
[1]	Quectel_BC20-OpenCPU_Genie_Log_抓取操作指导	介绍在 OpenCPU 方案中如何抓取 Genie Log
[2]	Quectel_BC20-OpenCPU_用户指导	介绍 BC20-OpenCPU 模块相关 API 接口的说明和使用指导

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
CPU	Central Processing Unit	中央处理单元
DOS	Disk Operating System	磁盘操作系统
UART	Universal Asynchronous Receiver/Transmitter	通用异步收发器
USB	Universal Serial Bus	通用串行总线
GPIO	General-Purpose Input/Output	通用输入/输出
LED	Light Emitting Diode	发光二极管
SDK	Software Development Kit	软件开发包