

# MC25&M25-OpenCPU

## 快速开发指导

**GSM/GPRS/GNSS 模块系列**

版本：MC25&M25-OpenCPU\_快速开发指导\_V1.0

日期：2019-07-17

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](mailto: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](mailto: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	2019-07-17	梁维/闻超	初始版本

# 目录

文档历史 .....	2
目录 .....	3
表格索引 .....	5
图片索引 .....	6
<b>1 基本概述 .....</b>	<b>7</b>
<b>2 开发前准备 .....</b>	<b>8</b>
2.1. 主机系统 .....	8
2.2. 编译器及调试工具 .....	8
2.3. 编程语言 .....	8
2.4. 模块硬件 .....	8
2.5. OpenCPU SDK .....	9
<b>3 关于 OpenCPU SDK .....</b>	<b>10</b>
<b>4 开发环境搭建 .....</b>	<b>12</b>
4.1. 编译环境 .....	12
4.1.1. CSDTK 编译器安装 .....	12
4.2. 调试环境 .....	13
4.2.1. CoolWatcher 工具安装 .....	13
4.2.2. CoolWatcher 工具配置 .....	14
<b>5 编译 .....</b>	<b>16</b>
5.1. 编译 .....	16
5.2. 编译输出 .....	16
<b>6 下载 .....</b>	<b>17</b>
6.1. TE-A .....	17
6.2. 单独模块 .....	17
6.2.1. MC25&M25-OpenCPU 模块 .....	18
<b>7 调试 .....</b>	<b>19</b>
7.1. 串口打印调试 .....	19
7.2. 使用 CoolWatcher 调试及下载 .....	19
7.2.1 使用 CoolWatcher 调试应用程序 .....	19
7.2.2 使用 CoolWatcher 下载应用程序 .....	22
<b>8 创建用户工程 .....</b>	<b>24</b>
<b>9 快速编程 .....</b>	<b>25</b>
9.1. GPIO 编程 .....	25
9.1.1 确认需求的头文件 .....	25
9.1.2 GPIO 编程 .....	25
9.1.3 定时器和 LED 灯控制 .....	26
9.1.4 运行程序 .....	29

9.2.	GPRS 编程.....	30
9.2.1	确认需求的头文件 .....	30
9.2.2	定义 PDP 上下文和 GPRS 配置信息.....	30
9.2.3	定义服务器的 IP 地址和端口 .....	31
9.2.4	定义接收缓存区 .....	31
9.2.5	为 GPRS 和 socket 定义回调函数.....	31
9.2.6	OpenCPU RIL 编程 .....	31
9.2.7	URC 消息编程 .....	32
9.2.8	GPRS 编程 .....	33
9.2.9	Socket 编程 .....	34
10	编程注意事项.....	37
10.1.	外部看门狗* .....	37
10.2.	重启方案.....	37
10.3.	客户任务注意事项 .....	37
10.4.	定时器 .....	38
10.5.	串口功能.....	38
10.6.	GPRS 和 TCP 功能.....	38
10.7.	动态内存分配 .....	39
10.8.	程序调试.....	39
11	附录 .....	40

## 表格索引

表 1: MC25_OPENCPU_GS5_SDK_V1.0 目录描述.....	10
表 2: MC25&M25-OPENCPU 模块调试串口引脚定义.....	18
表 3: TRACE 工具主操作界面介绍 .....	20
表 4: 参考文档 .....	40

## 图片索引

图 1: MC25_OPENCPU_GS5_SDK_V1.0 目录结构.....	10
图 2: CSDTK 编译器安装.....	12
图 3: COOLWATCHER 工具安装.....	13
图 4: COOLWATCHER 工具配置 .....	14
图 5: COOLWATCHER 工具操作界面.....	15
图 6: TE-A.....	17
图 7: QFLASH 工具 .....	18
图 8: 启动 TRACE 工具.....	19
图 9: TRACE 工具主操作界面 .....	20
图 10: 设置 TRACE LEVEL.....	21
图 11: TRACE 工具.....	22
图 12: 下载进度 .....	23
图 13: CUSTOM 目录（以 MC25_OPENCPU_GS5_SDK_V1.0 为例） .....	24
图 14: GSM EVB 的 LED 指示灯 .....	25

# 1 基本概述

本文档介绍如何使用 OpenCPU 软件开发工具包（SDK）快速开发 OpenCPU 项目，介绍了 SDK 文件夹的结构、开发环境的搭建、编译、下载和调试的具体步骤以及 OpenCPU 应用程序设计及编程的注意事项。

该文档适用于 MC25-OpenCPU 及 M25-OpenCPU 模块。



## 2 开发前准备

进行 OpenCPU 项目开发之前，需要确认是否具备如下软硬件配置。

### 2.1. 主机系统

支持以下任意一种主机操作系统：

- Windows 7 32 bit 或 64 bit
- Windows 10 32 bit 或 64 bit

### 2.2. 编译器及调试工具

- CSDTK 编译器。
- CoolWatcher 调试工具。

如需获取相关软件包，可联系移远通信技术支持 [support@quectel.com](mailto:support@quectel.com)。

### 2.3. 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。

### 2.4. 模块硬件

- 移远通信 MC25&M25-OpenCPU 模块
- 移远通信 GSM EVB
- 其他配件，如电源适配器、串口线等

如需上述硬件资源，可联系移远通信技术支持 [support@quectel.com](mailto:support@quectel.com)。

## 2.5. OpenCPU SDK

- OpenCPU SDK 软件包

如需获取 SDK 软件包，可联系移远通信技术支持 [support@quectel.com](mailto:support@quectel.com)。

- 下载工具

相关下载工具可从 SDK 软件包中的 *tools* 文件夹中获取。

### 3 关于 OpenCPU SDK

MC25&M25-OpenCPU SDK 分别为 *MC25\_OpenCPU\_GS5\_SDK* 和 *M25\_OpenCPU\_GS5\_SDK*。解压缩 SDK 软件包以后，即可查看其目录结构。

以 *MC25\_OpenCPU\_GS5\_SDK\_V1.0* 为例，其典型的目录层次结构如下图所示。

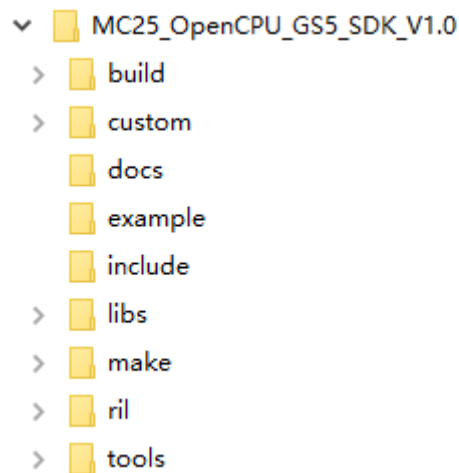


图 1: *MC25\_OpenCPU\_GS5\_SDK\_V1.0* 目录结构

表 1: *MC25\_OpenCPU\_GS5\_SDK\_V1.0* 目录描述

目录名	描述
<i>MC25_OpenCPU_GS5_SDK_V1.0</i>	OpenCPU SDK 根目录。
<i>build</i>	存放编译输出结果。
<i>custom</i>	存放用户配置相关文件。 在子目录 <i>custom\config</i> 的相关文件中，用户可根据需要关闭 OpenCPU 功能、添加新任务、任务栈空间大小、GPIO 初始状态等。 所有为用户提供的配置文件的文件名均以“custom_”开头。
<i>docs</i>	存放 OpenCPU 项目相关说明文档。
<i>example</i>	此目录用于存放示例代码。 每个示例文件均实现一个独立而完整的功能；对 <i>makefile</i> 文件进行修改后，每个示例文件均可编译为可执行的二进制文件。

<i>include</i>	存放所有的 API 头文件。
<i>libs</i>	存放所需库文件。
<i>make</i>	存放编译脚本文件和 <i>makefile</i> 文件。
<i>ril</i>	存放 OpenCPU RIL 源代码。 根据该源码，用户可以基于 RIL 的基础功能来发送 AT 命令，或者实现自己的 API 接口。
<i>tools</i>	存放相关开发工具，如下载工具、DFOTA 包制作工具等。

更多有关 *makefile* 文件修改以及系统配置信息的细节，例如新任务添加、任务栈空间大小及 GPIO 初始状态配置等，请参考文档[1]和[2]。

# 4 开发环境搭建

## 4.1. 编译环境

### 4.1.1. CSDTK 编译器安装

在 C 盘根目录创建 **CSDTK4** 文件夹，并将最新版本 CSDTK 工具包解压至 **CSDTK4** 文件夹内。

如需获取软件包，可联系移远通信技术支持 [support@quectel.com](mailto:support@quectel.com)。



图 2: CSDTK 编译器安装

## 4.2. 调试环境

### 4.2.1. CoolWatcher 工具安装

CoolWatcher 是平台调试的主要工具。

如下图，在 C 盘根目录创建 *cooltools* 文件夹，将最新版本 *cooltools* 工具包解压至 *cooltools* 文件夹。  
CoolWatcher 工具默认包含在 *cooltools* 工具包内。

如需获取软件包，可联系移远通信技术支持 [support@quectel.com](mailto:support@quectel.com)。

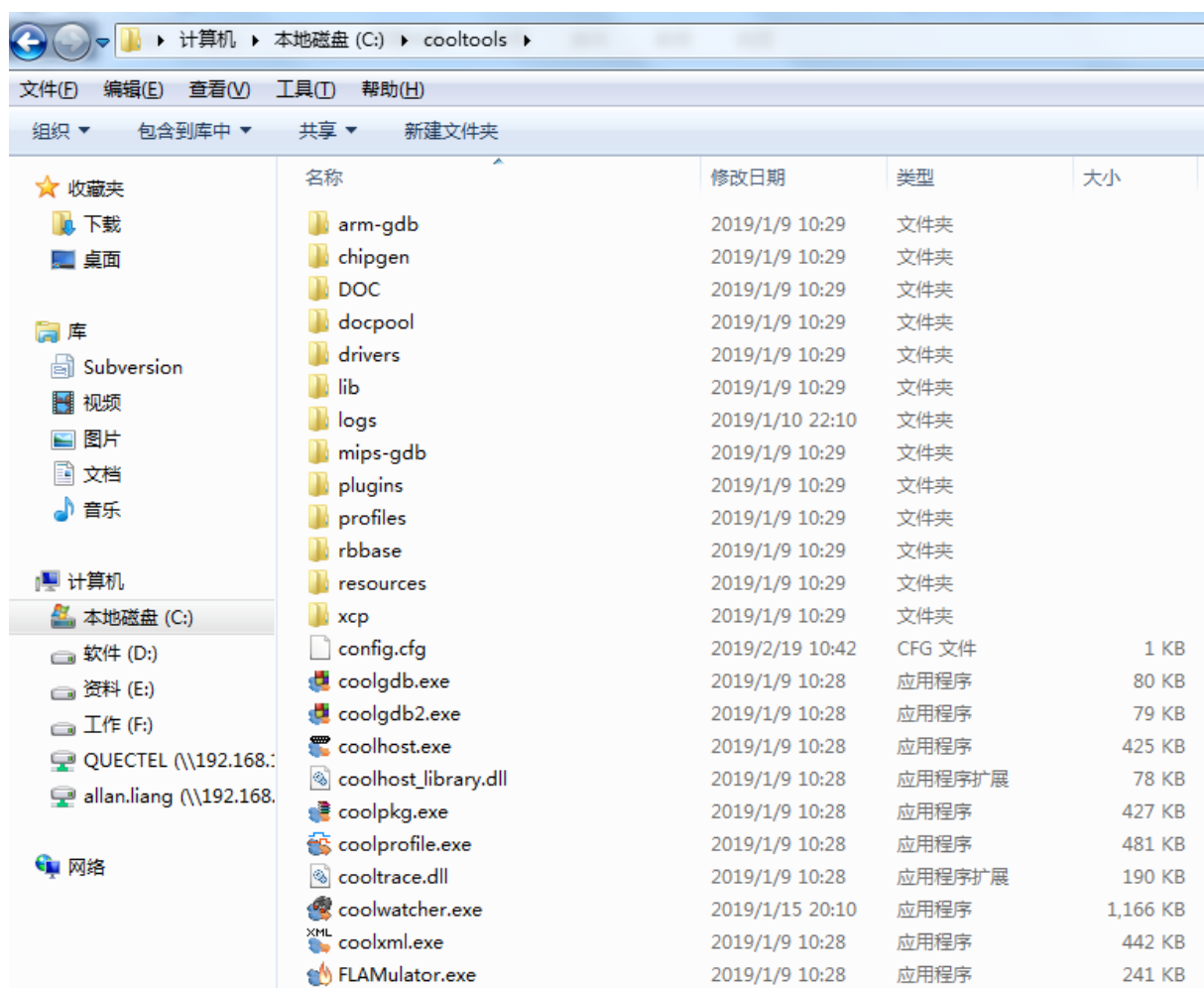


图 3: CoolWatcher 工具安装

### 4.2.2. CoolWachter 工具配置

CoolWachter 工具配置方法如下：

运行 `coolwatcher.exe`，在“Profiles”标签里选择对应平台“8955”，并在“lastcomport”栏输入正确的串口号（调试串口）。之后运行 CoolWachter 时，将默认使用上次设置的信息；如果平台或串口有变更，则需要重新设置。

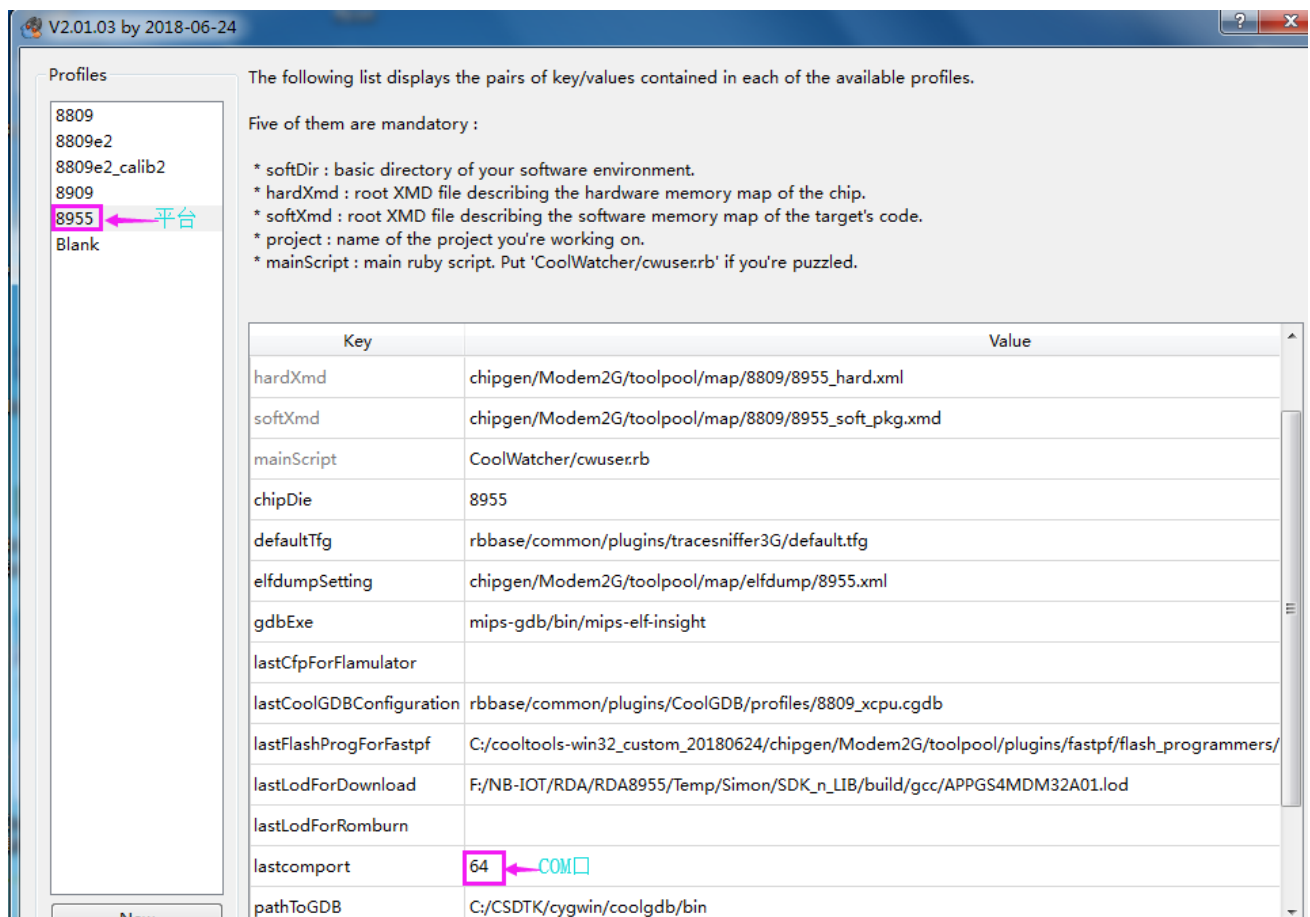


图 4：CoolWachter 工具配置

上述操作完成后，点击“OK”，进入CoolWachter工具操作界面：

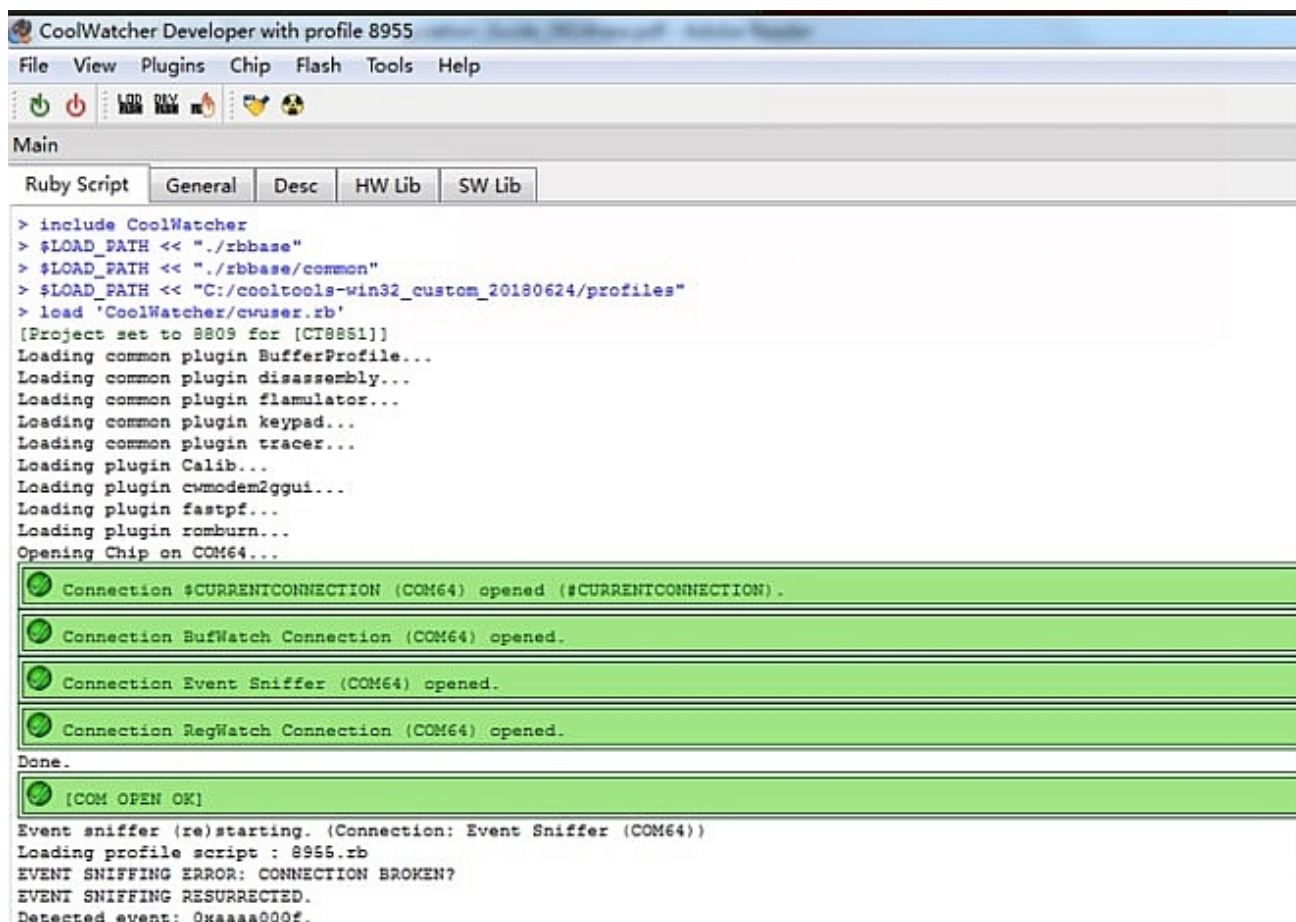


图 5: CoolWatcher 工具操作界面



# 5 编译

本章将介绍如何通过命令行编译 OpenCPU 应用程序。

## 5.1. 编译

OpenCPU SDK 默认包含命令行快捷方式“MS-DOS”。用户只需执行简单的命令，即可对 SDK 代码进行编译。清除编译文件及编译新程序的命令定义如下：

```
make clean
make new
```

如编译成功，输出结果如下图所示（以 MC25-OpenCPU 为例）。编译生成的文件位于 `\sdk\build\gcc`。



```

-----
- GCC Compiling Finished Sucessfully.
- The target image is in the build/gcc directory.
-----
BIN: build/gcc/PPGS5MDM32A01.bin
LOD: build/gcc/PPGS5MDM32A01.lod
Start address: 137560064
Sector size: 4096
C:\Users\Q\Desktop\MC25_OpenCPU_GS5_SDK_V1.0_BETA0424A>
  
```

## 5.2. 编译输出

编译期间，命令行将输出一些编译器处理信息。所有警告及错误信息都保存在 `\sdk\build\gcc\build.log` 路径下。用户可以根据 `build.log` 中的错误行和错误提示，快速排查代码错误。



### 6.2.1. MC25&M25-OpenCPU 模块

本节将介绍如何通过 MC25&M25-OpenCPU 模块的调试串口下载程序。

模块调试串口引脚定义如下：

表 2：MC25&M25-OpenCPU 模块调试串口引脚定义

模块名称	Tx 引脚	Rx 引脚
MC25-OpenCPU	29	30
M25-OpenCPU	39	38

请使用移远通信提供的 QFlash 工具（如下图）下载烧写应用程序。该工具可从 SDK 包中的 *tools* 文件夹获取，具体使用方法请参考文档[3]。

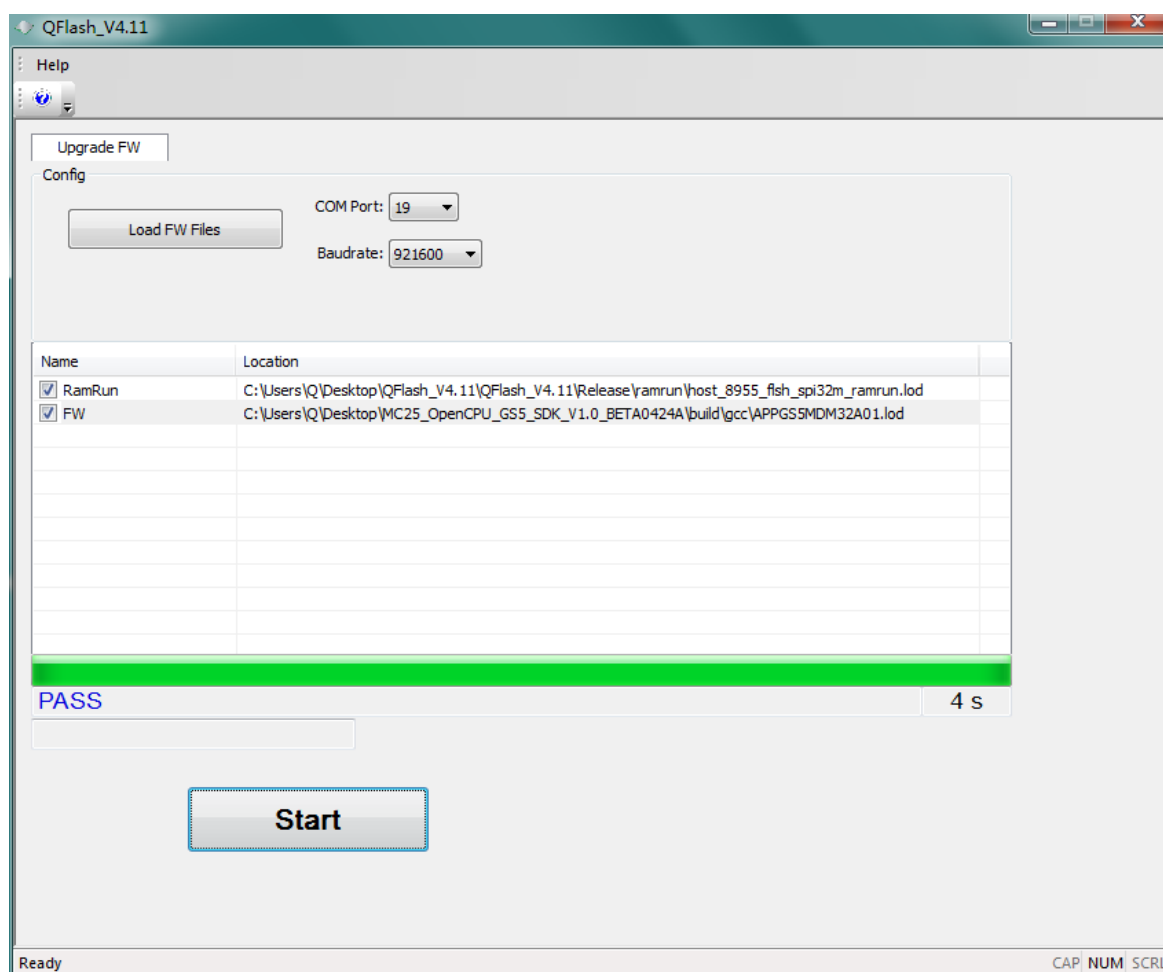


图 7：QFlash 工具

# 7 调试

MC25&M25-OpenCPU 应用程序的调试主要有两种方式：串口打印调试，以及使用 CoolWatcher 工具进行调试。MC25&M25-OpenCPU 模块提供三个串口：主串口（UART1）、调试串口（UART2）和辅助串口（UART3），均可用于调试程序。

## 7.1. 串口打印调试

用户可以在程序中调用 `QI_UART_Open()` 打开 UART1（主串口）或 UART3（辅助串口），再调用 `QI_UART_Write()` 输出调试信息，并通过 QCOM 工具（用户也可选择其他串口调试工具）接收调试信息。

## 7.2. 使用 CoolWatcher 调试及下载

用户可以在程序中调用 `QI_Debug_Trace()` 接口从 UART2（调试串口）打印日志，但需要使用 CoolWatcher 工具来捕获日志。

### 7.2.1 使用 CoolWatcher 调试应用程序

本节将介绍如何使用 CoolWatcher 工具进行调试。

如下图，在 CoolWatcher 工具主操作界面菜单栏选择“Plugins”，点击“Activate Tracer”启动 Trace 工具。

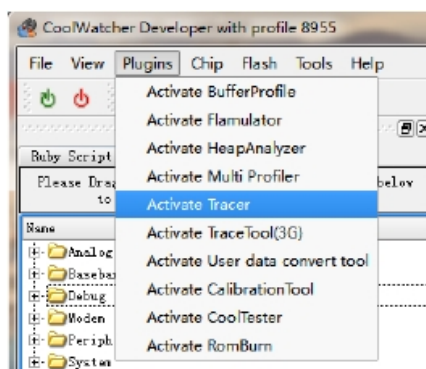


图 8：启动 Trace 工具

Trace 工具主操作界面如下图所示。

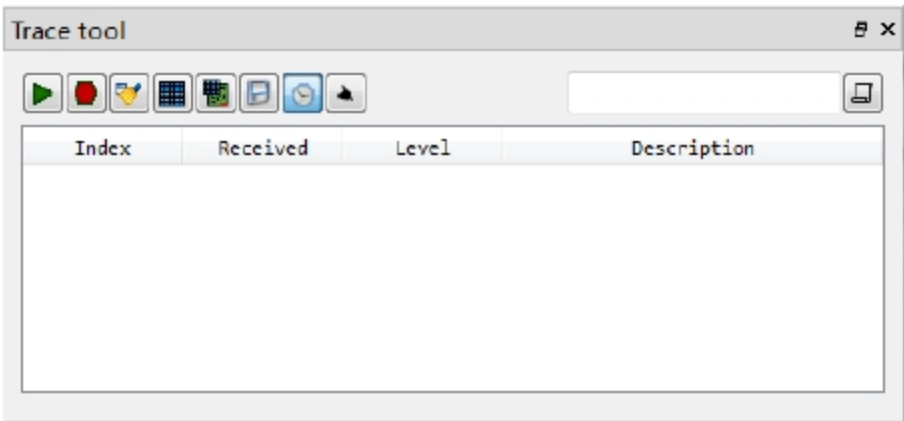


图 9: Trace 工具主操作界面

表 3: Trace 工具主操作界面介绍

功能	说明
	开始 Trace
	停止 Trace
	清空
	设置 Trace Level
	复用 Trace Level
	保存
	启动/关闭 Received 列
	启动/关闭 Comment 列
Index	序号
Received	PC 接收 Trace 的时间
Level	Trace Level
Description	描述

Trace 工具的使用步骤如下：

1. 进入 Trace 工具主操作界面，点击按钮，设置 Trace Level。用户需在下图左侧表格中选择关注

的 Level。

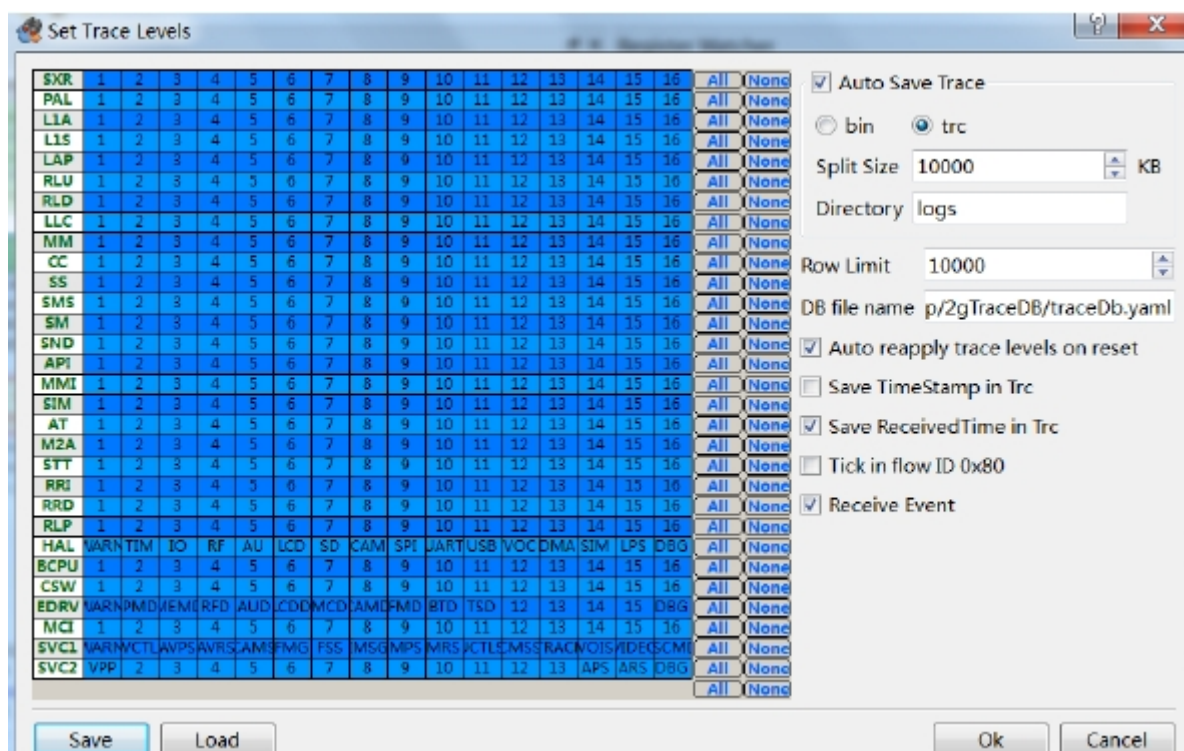



图 10: 设置 Trace Level

- **Auto Save Trace:** 是否自动保存 Trace。如果勾选此选项，Trace 将自动保存。
- **bin/rtr:** Trace 文件类型。 .bin 为二进制文件， .rtr 为文本文件。
- **Split Size:** Trace 文件大小。文件大小超过该值时，文件将被自动切分。
- **DB file name:** DB 文件。
- **Row Limit:** Trace 信息表格（如图 11）的最大行数。
- **Auto reapply trace levels on reset:** 重启后，是否自动使用重启前的 Trace Level 配置信息。
- **Save TimeStamp in Trc:** 是否保存时间戳。
- **Save ReceivedTime in Trc:** 是否保存工具解析 Trace 的时间。
- **Tick in flow ID 0x80:** 该配置项需与 .lod 保持一致。如果 .lod 中有时间戳，则选中本项，否则不选。
- **Receive Event:** 是否接收 Event。
- **Save 按钮:** 保存 Level 配置。
- **Load 按钮:** 加载 Level 配置。

2. 回到 Trace 工具主操作界面，点击  按钮，开始 Trace 功能。Trace 信息将在表格中显示，如下图：

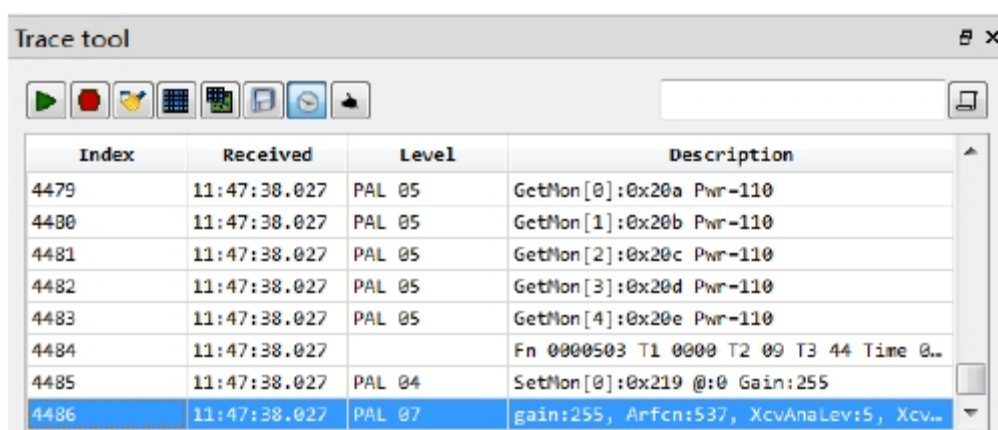


图 11: Trace 工具

3. 用户可以使用右上角的输入框对打印的日志进行搜索和筛选。
4. 点击  按钮，保存日志为.trc 文件。
5. 点击  按钮，结束 Trace 功能。

## 备注

1. .trc 文件可以使用文本编辑工具打开，如 Notepad++。
2. Trace 工具的默认配置信息请参考 `rbbase\common\plugins\tracer\` 目录下的文件。

## 7.2.2 使用 CoolWatcher 下载应用程序

1. 点击 CoolWatcher 工具栏按钮 ，从相应的文件加载路径，选择内核固件包中的.lod 文件或应用程序.lod 文件。
2. 点击工具栏按钮 ，选择相应的 Flash programmer (`host_8955_flash_spi32m_ramrun.lod`)。
3. 点击工具栏按钮 ，开始将软件版本或应用程序烧录到模块中。此时程序右下角出现一个下载进度条（如下图），下载完成后，进度条显示 100%，CoolWatcher 操作界面的“Ruby Script”区域将显示完成信息。





图 12: 下载进度

### 备注

1. 烧录过程开始前不需要重启模块，烧录成功后模块将自动重启，不需手动重启。
2. 此方式仅适用于代码调试，正式生产时请使用 QFlash 工具进行下载。



## 8 创建用户工程

默认情况下，`SDK\custom`被设计为存放用户工程的根目录。该目录下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。

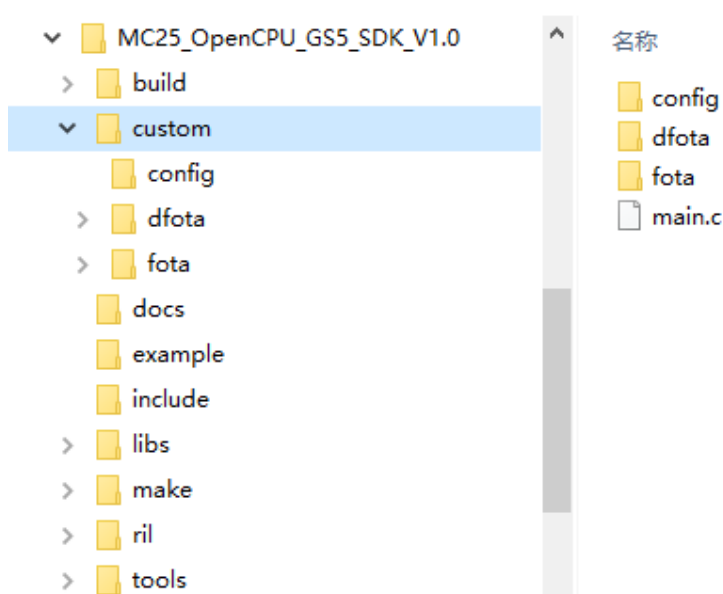


图 13: `custom` 目录（以 `MC25_OpenCPU_GS5_SDK_V1.0` 为例）

用户可以在 `SDK\custom` 下目录添加其他文件夹和子目录。具体编译方法以及 `makefile` 文件的修改方法请参考文档[1]和[2]。

SDK 中提供一个默认的用户工程，用户只需在 `main.c` 文件中添加代码或直接替换该文件即可。此外，用户也可以添加其他.c 文件，该目录下新添加的.c 文件将会被自动编译。

# 9 快速编程

本章节设计了两个例程，用于展示如何基于 OpenCPU SDK 编写应用程序。

章节 9.1.主要演示如何通过控制 GPIO 的引脚电平来驱动 LED 灯的亮、灭；章节 9.2.演示如何使用 GPRS 网络建立 TCP 连接、向 TCP 服务器传输数据。

## 9.1. GPIO 编程

### 9.1.1 确认需求的头文件

要确认需要哪些头文件，用户需要先了解应用程序的基本需求。此例程的基本需求是：通过周期性更改 GPIO 电平状态来实现 LED 闪烁。

因此，需要包含的头文件如下：

```
#include "ql_stdlib.h"
#include "ql_trace.h"    //打印日志相关
#include "ql_error.h"    //API 接口的所有返回值都在 ql_error.h 中定义
#include "ql_system.h"   //OpenCPU 应用程序都有一个消息循环过程
#include "ql_uart.h"     //串口相关
#include "ql_gpio.h"     //GPIO 相关
#include "ql_timer.h"    //定时器相关
```

### 9.1.2 GPIO 编程

移远通信 GSM EVB 上，NETLIGHT 引脚已连接到 LED 上，这意味着用户可以控制 NETLIGHT 引脚的电平来实现 LED 的闪烁。



图 14: GSM EVB 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 = QI_GPIO_Init(m_gpioPin, PINDIRECTION_OUT, PINLEVEL_LOW, PINPULLSEL_PULLUP);
if (QL_RET_OK == ret)
{
    QI_Debug_Trace("<-- Initialize GPIO successfully -->\r\n");
}else{
    QI_Debug_Trace("<-- Fail to initialize GPIO pin, cause=%d -->\r\n", ret);
}
```

步骤三：启动一个定时器，并定期改变 GPIO 口的电平，从而实现 LED 闪烁。

### 9.1.3 定时器和 LED 灯控制

定义一个 500ms 超时的定时器（Timer），用以控制 LED 亮 500ms、暗 500ms。

步骤一：定义一个定时器和定时器中断处理程序。

```
//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：注册并打开一个定时器。

```
//Register and start a timer
QI_Timer_Register(m_myTimerId, Callback_OnTimer, NULL);
QI_Timer_Start(m_myTimerId, m_nInterval, TRUE);
```

步骤三：实现定时器的中断处理程序。

```
static void Callback_OnTimer(u32 timerId, void* param)
{
    s32 gpioLvl = QI_GPIO_GetLevel(m_gpioPin);
    if (PINLEVEL_LOW == gpioLvl)
    {
```

```

        //Set GPIO to high level, then LED is light
        QI_GPIO_SetLevel(m_gpioPin, PINLEVEL_HIGH);
        QI_Debug_Trace("<-- Set GPIO to high level -->\r\n");
    }else{
        //Set GPIO to low level, then LED is dark
        QI_GPIO_SetLevel(m_gpioPin, PINLEVEL_LOW);
        QI_Debug_Trace("<-- 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_PORT1
#define DBG_BUF_LEN 512
static char DBG_BUFFER[DBG_BUF_LEN];
#define APP_DEBUG(FORMAT,...) {\
    QI_memset(DBG_BUFFER, 0, DBG_BUF_LEN);\
    QI_sprintf(DBG_BUFFER,FORMAT,##__VA_ARGS__); \
    if (UART_PORT2 == (DEBUG_PORT)) \
    {\
        QI_Debug_Trace(DBG_BUFFER);\
    } else {\
        QI_UART_Write((Enum_SerialPort)(DEBUG_PORT), (u8*)(DBG_BUFFER), QI_strlen((const char *) (DBG_BUFFER)));\
    }\
}
#else
#define APP_DEBUG(FORMAT,...)
#endif

static Enum_SerialPort m_myUartPort = UART_PORT1;

//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 = QI_UART_Register(m_myUartPort, CallBack_UART_Hdlr, NULL);
    if (ret < QL_RET_OK)
    {
        QI_Debug_Trace("Fail to register serial port[%d], ret=%d\r\n", m_myUartPort, ret);
    }
    ret = QI_UART_Open(m_myUartPort, 115200, FC_NONE);
    if (ret < QL_RET_OK)
    {
        QI_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 = QI_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
    QI_Timer_Register(m_myTimerId, Callback_OnTimer, NULL);
    QI_Timer_Start(m_myTimerId, m_nInterval, TRUE);
}
```

```
//Start message loop of this task
while(TRUE)
{
    QI_OS_GetMessage(&msg);
    switch(msg.message)
    {
        default:
            break;
    }
}

static void Callback_OnTimer(u32 timerId, void* param)
{
    s32 gpioLvl = QI_GPIO_GetLevel(m_gpioPin);
    if (PINLEVEL_LOW == gpioLvl)
    {
        //Set GPIO to high level, then LED is light
        QI_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
        QI_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 运行程序

将完整的代码复制到 `SDK\custom\main.c` 以覆盖已有的代码；编译代码，将生成的可执行文件(.lod)烧录到模块中。了解更多编译和下载的相关信息，请参考[文档\[1\]](#)和[\[2\]](#)。

应用程序运行时，GSM EVB 上的 **D502 LED** 每 500ms 闪烁一次。同时，主串口输出以下日志信息：

```
[2019-02-13_13:14:08:624]OpenCPU: LED Blinking by NETLIGH
[2019-02-13_13:14:08:624]<-- Initialize GPIO successfully -->
[2019-02-13_13_13:14:09:120]<-- Set GPIO to high level -->
[2019-02-13_13:14:09:620]<-- Set GPIO to low level -->
[2019-02-13_13:14:10:120]<-- Set GPIO to high level -->
```

```
[2019-02-13_13:14:10:620]<-- Set GPIO to low level -->
[2019-02-13_13:14:11:120]<-- Set GPIO to high level -->
[2019-02-13_13:14:11:621]<-- Set GPIO to low level -->
[2019-02-13_13:14:12:120]<-- Set GPIO to high level -->
[2019-02-13_13:14:12:621]<-- Set GPIO to low level -->
[2019-02-13_13:14:13:121]<-- Set GPIO to high level -->
```

## 9.2. GPRS 编程

### 9.2.1 确认需求的头文件

要确认需要哪些头文件，用户需要先了解应用程序的基本需求。此例程的基本需求是：通过 `socket` 连接 TCP 服务器，并向 TCP 服务器传输字符数据。

因此，需要包含的头文件如下所示：

```
#include "custom_feature_def.h" //OpenCPU RIL 功能相关宏定义
#include "ril.h"                //OpenCPU RIL 功能
#include "ql_stdlib.h"          //标准库
#include "ql_trace.h"           //输出打印信息
#include "ql_error.h"           //所有的 API 的返回值定义在 ql_error.h 中
#include "ql_system.h"          //OpenCPU 系统相关
#include "ql_gprs.h"            //GPRS 相关
#include "ql_socket.h"          //Socket 连接相关
```

### 9.2.2 定义 PDP 上下文和 GPRS 配置信息

```
#define PDP_CONTEXT_ID 0
static ST_GprsConfig m_GprsConfig = {
    "CMNET", //APN name
    "",      //User name for APN
    "",      //Password for APN
    0,
    NULL,
    NULL,
};
```

此处设定 APN 为中国移动“CMNET”。用户可根据具体情况修改。

### 9.2.3 定义服务器的 IP 地址和端口

```
static u8 m_SrvADDR[20] = "116.247.104.27";
static u32 m_SrvPort = 6003;
```

此处定义了一个移远通信公共服务器和 **socket** 端口。用户可使用自身的服务器。

### 9.2.4 定义接收缓存区

Socket 连接建立时，需要一个缓存区来接收来自服务器的数据。

```
#define SOC_RECV_BUFFER_LEN 1460
static u8 m_SocketRcvBuf[SOC_RECV_BUFFER_LEN];
```

### 9.2.5 为 GPRS 和 **socket** 定义回调函数

```
static void Callback_GPRS_Deactivated(u8 contextId, s32 errCode, void* customParam);
static void Callback_Socket_Close(s32 socketId, s32 errCode, void* customParam );
static void Callback_Socket_Read(s32 socketId, s32 errCode, void* customParam );
static void Callback_Socket_Write(s32 socketId, s32 errCode, void* customParam );
```

- *Callback\_GPRS\_Deactivated*: 当 GPRS 网络掉网时，该回调函数会被调用。
- *Callback\_Socket\_Close*: 当 **socket** 连接断开时，该回调函数会被调用。
- *Callback\_Socket\_Read*: 当 **socket** 上接收到数据时，该回调函数会被调用。
- *Callback\_Socket\_Write*: 如果调用 *QI\_SOC\_Write* 发送数据到 **socket** 时 **socket** 正处于忙碌状态，那么，此后如果 **socket** 空闲，该回调函数将会被调用，以通知应用程序该 **socket** 可用。

### 9.2.6 OpenCPU RIL 编程

使用 OpenCPU RIL 功能之前，应用程序需在主任务收到 MSG\_ID\_RIL\_READY 消息时调用 *QI\_RIL\_Initialize()* 对 RIL 相关功能进行初始化。

```
//Start message loop of this task
while(TRUE)
{
    QI_OS_GetMessage(&msg);
    switch(msg.message)
    {
        case MSG_ID_RIL_READY:
            QI_Debug_Trace("<-- RIL is ready -->\r\n");
            QI_RIL_Initialize();
            break;
```



## 9.2.7 URC 消息编程

接入 GPRS 网络前，需要等待模块注册到 GPRS 网络。如果模块成功注册到 GPRS 网络，应用程序会收到 URC 消息 URC\_GPRS\_NW\_STATE\_IND。在此之前，系统启动过程中，应用程序会收到其他一些指示模块初始化状态的 URC 消息，比如 **AT+CFUN** 状态、(U)SIM 卡状态和 GSM 网络变化状态，开发人员可根据需要进行相应的处理。

以下代码完整展示了 URC 消息相关的使用方法：

```

/*****
/* The entrance procedure for this example application
*****/

void proc_main_task(s32 taskId)
{
    ST_MSG msg;

    QI_Debug_Trace("OpenCPU: Simple GPRS-TCP Example\r\n");

    //Start message loop of this task
    while(TRUE)
    {
        QI_OS_GetMessage(&msg);
        switch(msg.message)
        {
            case MSG_ID_RIL_READY:
                QI_Debug_Trace("<-- RIL is ready -->\r\n");
                QI_RIL_Initialize();
                break;
            case MSG_ID_URC_INDICATION:
                //QI_Debug_Trace("<-- Received URC: type: %d, -->\r\n", msg.param1);
                switch (msg.param1)
                {
                    case URC_SYS_INIT_STATE_IND:
                        QI_Debug_Trace("<-- Sys Init Status %d -->\r\n", msg.param2);
                        break;
                    case URC_SIM_CARD_STATE_IND:
                        QI_Debug_Trace("<-- SIM Card Status:%d -->\r\n", msg.param2);
                        break;
                    case URC_GSM_NW_STATE_IND:
                        QI_Debug_Trace("<-- GSM Network Status:%d -->\r\n", msg.param2);
                        break;
                    case URC_GPRS_NW_STATE_IND:
                        QI_Debug_Trace("<-- GPRS Network Status:%d -->\r\n", msg.param2);
                        if (NW_STAT_REGISTERED == msg.param2)
                        {

```

```

        GPRS_Surf();
    }
    break;
case URC_CFUN_STATE_IND:
    QI_Debug_Trace("<-- CFUN Status:%d -->\r\n", msg.param2);
    break;
default:
    QI_Debug_Trace("<-- Other URC: type=%d\r\n", msg.param1);
    break;
}
break;
default:
    break;
}
}
}

```

### 9.2.8 GPRS 编程

模块成功注册到 GPRS 网络之后，用户可以进行 GPRS 相关的编程。本节介绍了 GPRS 编程的主要步骤，具体 GPRS API 相关使用情况，请参考文档[1]和[2]。

**步骤一：**注册 GPRS 相关的回调函数。

```

ST_PDPContxt_Callback callback_gprs_func = {
    //Callback_GPRS_Actived,
    NULL,
    Callback_GPRS_Deactivated
};

ST_SOC_Callback callback_soc_func = {
    //Callback_socket_connect,
    NULL,
    Callback_Socket_Close,
    //Callback_socket_accept,
    NULL,
    Callback_Socket_Read,
    Callback_Socket_Write
};

//Register GPRS callback
ret=QI_GPRS_Register(PDP_CONTEXT_ID, &callback_gprs_func, NULL);
if (GPRS_PDP_SUCCESS == ret)
{
    QI_Debug_Trace("<-- Register GPRS callback function -->\r\n");
}

```

```

}else{
    QI_Debug_Trace("<-- Fail to register GPRS, ret=%d. -->\r\n", ret);
    return;
}

```

步骤二：配置 PDP 上下文。

```

ret=QI_GPRS_Config(PDP_CONTEXT_ID, &m_GprsConfig);
if (GPRS_PDP_SUCCESS == ret)
{
    QI_Debug_Trace("<-- Configure GPRS PDP -->\r\n");
}else{
    QI_Debug_Trace("<-- Fail to configure GPRS PDP, ret=%d. -->\r\n", ret);
    return;
}

```

步骤三：激活 PDP。

```

ret=QI_GPRS_Activate(PDP_CONTEXT_ID);
if (ret == GPRS_PDP_SUCCESS)
{
    m_GprsActState = 1;
    QI_Debug_Trace("<-- Activate GPRS successfully. -->\r\n\r\n");
}else{
    QI_Debug_Trace("<-- Fail to activate GPRS, ret=%d. -->\r\n\r\n", ret);
    return;
}

```

步骤四：反激活 PDP。

```

ret=QI_GPRS_Deactivate(PDP_CONTEXT_ID);
QI_Debug_Trace("<-- Deactivate GPRS, ret=%d -->\r\n\r\n", ret);

```

## 9.2.9 Socket 编程

激活 GPRS PDP 之后，用户可以进行 TCP/UDP socket 编程。本节介绍了 Socket 编程的主要步骤。具体 socket API 相关使用情况，请参考[文档\[1\]](#)和[\[2\]](#)。

步骤一：注册 socket 相关的回调函数。

```

ret=QI_SOC_Register(callback_soc_func, NULL);
if (SOC_SUCCESS == ret)
{
    QI_Debug_Trace("<-- Register socket callback function -->\r\n");
}else{
    QI_Debug_Trace("<-- Fail to register socket callback, ret=%d. -->\r\n", ret);
}

```

```

        return;
    }

```

步骤二：创建 socket。

```

m_SocketId=QI_SOC_Create(PDP_CONTEXT_ID, SOC_TYPE_TCP);
if (m_SocketId >= 0)
{
    QI_Debug_Trace("<-- Create socket successfully, socket id=%d. -->\r\n", m_SocketId);
}else{
    QI_Debug_Trace("<-- Fail to create socket, ret=%d. -->\r\n", m_SocketId);
    return;
}

```

步骤三：连接到 socket 服务器。

```

ret=QI_SOC_Connect (m_SocketId,(u32)m_ipAddress, m_SrvPort);
if (SOC_SUCCESS == ret)
{
    m_SocketConnState=1;
    QI_Debug_Trace("<-- Connect to server successfully -->\r\n");
}else{
    QI_Debug_Trace("<-- Fail to connect to server, ret=%d -->\r\n", ret);
    QI_Debug_Trace("<-- Close socket.-->\r\n");
    QI_SOC_Close(m_SocketId);
    m_SocketId=-1;
    return;
}

```

Socket 连接到服务器之后，可以发送数据到服务器或从服务器接收数据。

步骤四：发送 socket 数据。

```

char pchData[200];
s32  dataLen=0;
u64  ackNum=0;
QI_memset(pchData, 0x0, sizeof(pchData));
dataLen += QI_sprintf(pchData + dataLen, "%s", "Q u e c t e l");
ret=QI_SOC_Send(m_SocketId, (u8*)pchData, dataLen);
if (ret ==dataLen)
{
    QI_Debug_Trace("<-- Send socket data successfully. --> \r\n");
}else{
    QI_Debug_Trace("<-- Fail to send socket data. --> \r\n");
}

```

此代码展示了如何发送数据“Quectel”到服务器。

数据发送之后，可调用 `QL_SOC_GetAckNumber()` 来检测服务器是否收到数据。此外，也可调用 `QL_SOC_Close()` 来关闭 socket 连接，或调用 `QL_GPRS_DeactivateEx()` 来反激活 GPRS PDP。

本例完整的代码请参考 SDK 中的 `example_tcp_demo.c` 文件，该代码支持编译及运行，具体用法请参照上述例程。

# 10 编程注意事项

## 10.1. 外部看门狗\*

为防止应用程序出现无法恢复的逻辑异常，建议用户在产品设计中增加外部看门狗芯片，相关设计请参考文档[6]和[7]。外部看门狗溢出时，将复位模块的 VBAT 引脚，实现断电重启，从而复位模块。

### 备注

“\*” 表示正在开发中。

## 10.2. 重启方案

为保证模块持续稳定运行，推荐用户在软件设计中增加重启方案。模块发生网络故障时，用户可采用模块重启方案，尝试让模块自动恢复。

## 10.3. 客户任务注意事项

客户可以在 *custom\_task\_cfg.h* 文件中自定义任务，例如：

```
TASK_ITEM (proc_name, subtask1_id, 5*1024, DEFAULT_VALUE1, DEFAULT_VALUE2)
```

其中 *proc\_name* 为该任务的入口函数的名称，任务添加后必须实现此函数，否则编译将会报错；*subtask1\_id* 为该任务的 task ID，用户任务之间发送消息，可使用此 task ID。

如果任务中有消息需要处理，用户应在任务中增加 *QI\_OS\_GetMessage()* 接口（定时器、TCP 等接口的回调均通过消息触发，如果没有此接口，以上功能无法正常运行）。调用此接口后，如果任务中没有消息需要处理，任务将会被阻塞，让出 CPU 使用权。

## 10.4. 定时器

OpenCPU 中提供两种定时器：普通定时器和快速定时器。一个任务中最多可以注册 10 个普通定时器，整个应用程序只能注册一个快速定时器。

普通定时器在注册时需要绑定当前任务的 `task ID`，中断服务函数中无法注册定时器。使用 `QI_Timer_Start` 和 `QI_Timer_Stop` 的任务必须跟该定时器注册时的任务相同，否则将返回调用失败。

普通定时器运行期间，如果任务不能及时调用 `QI_OS_GetMessage()` 接口处理定时器的超时消息，则会导致定时器的回调函数无法得到及时处理。

快速定时器直接通过硬件中断触发，实时性较高，但请不要在中断服务函数中处理过多的业务，否则会引起系统异常。

## 10.5. 串口功能

MC25&M25-OpenCPU 提供三个串口，主串口 (UART1)，调试串口 (UART2) 和辅助串口 (UART3)。UART1 和 UART3 可以用于应用开发，默认波特率为 115200bps，串口数据格式为 8N1 (8 位数据位，无奇偶校验，1 位停止位)，串口数据缓冲区大小为 4096 字节；调试串口只能用于烧录程序以及配合 CoolWatcher 工具调试程序。

如果串口的回调接口收到消息 `EVENT_UART_READY_TO_READ`，则提示用户有数据需要读取，用户应该循环读取串口数据，直到读取串口接收的全部数据；否则再次接收到数据时，将不会上报消息通知应用层，可能导致串口出现假死。

## 10.6. GPRS 和 TCP 功能

执行 GPRS 和 TCP 回调函数的任务，即是注册这些回调函数的任务。如果当前任务不能及时处理收到的消息，则会导致这些回调函数的执行出现延迟。

模块中 GPRS 和 TCP 的同步 API 接口会阻塞任务执行，为了避免任务阻塞较长时间，对于一般网络业务的相关接口，建议用户使用异步 API 接口。

GPRS 同步 API 接口主要有：

- `QI_GPRS_Activate (u8 contextId)` 最长超时等待时间 180s
- `QI_GPRS_Deactivate (u8 contextId)` 最长超时时间 90s

TCP 同步 API 接口主要有：

- *QI\_SOC\_Connect* (*s32 socketId*, *u32 remoteIP*, *u16 remotePort*) 最长超时时间 75s
- *QI\_IpHelper\_GetIPByHostName* (*u8 contextId*, *u8 requestId*, *u8 \*hostname*, *u32\* ipCount*, *u32\* ipAddress*) 最长超时时间 60s

## 10.7. 动态内存分配

用户可以调用 *QI\_MEM\_Alloc()*来指定动态内存的大小，也可以调用 *QI\_MEM\_Free()*来释放内存。应用程序最大可以申请 500KB 的动态内存。

## 10.8. 程序调试

在 OpenCPU 应用程序中，通过串口打印 **Trace** 日志是主要的调试方法。一般情况下，用户可以使用 *APP\_DEBUG* 接口打印信息到串口调试工具中，并通过日志信息检查业务流程是否正常。如果模块出现异常重启、Dump、挂网或数据业务异常等问题，移远通信希望用户可以通过 **CoolWatcher** 工具提供一份内核日志，用于辅助分析问题。



# 11 附录

表 4: 参考文档

序号	文档	描述
[1]	Quectel_MC25-OpenCPU_User_Guide	介绍 OpenCPU 平台及项目相关的 API 接口
[2]	Quectel_M25-OpenCPU_User_Guide	介绍 OpenCPU 平台及项目相关的 API 接口
[3]	Quectel_MC25&M25&M56-R-OpenCPU_QFlash_User_Guide	介绍如何在 OpenCPU 方案中使用 QFlash 工具下载烧写程序
[4]	Quectel_OpenCPU_RIL_Application_Note	介绍如何通过 RIL 来封装 RIL 接口、处理 AT 命令的返回值
[5]	Quectel_MC25&M25&M56-R-OpenCPU_DFOTA_Application_Note_V1.0	介绍如何在 OpenCPU 方案中使用 DFOTA
[6]	Quectel_MC25-OpenCPU_参考设计手册	MC25-OpenCPU 参考设计手册
[7]	Quectel_M25-OpenCPU_参考设计手册	M25-OpenCPU 参考设计手册