Build a Smarter World

EC2X-QuecOpen

I2C 开发指导

LTE 系列

版本：EC2X-QuecOpen_I2C_开发指导_V1.0

日期：2018-02-28

状态：临时文件

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://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://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

文档历史

修订记录

版本	日期	作者	变更表述
1.0	2018-02-28	高飞虎	初始版本

目录

表格索引

图片索引

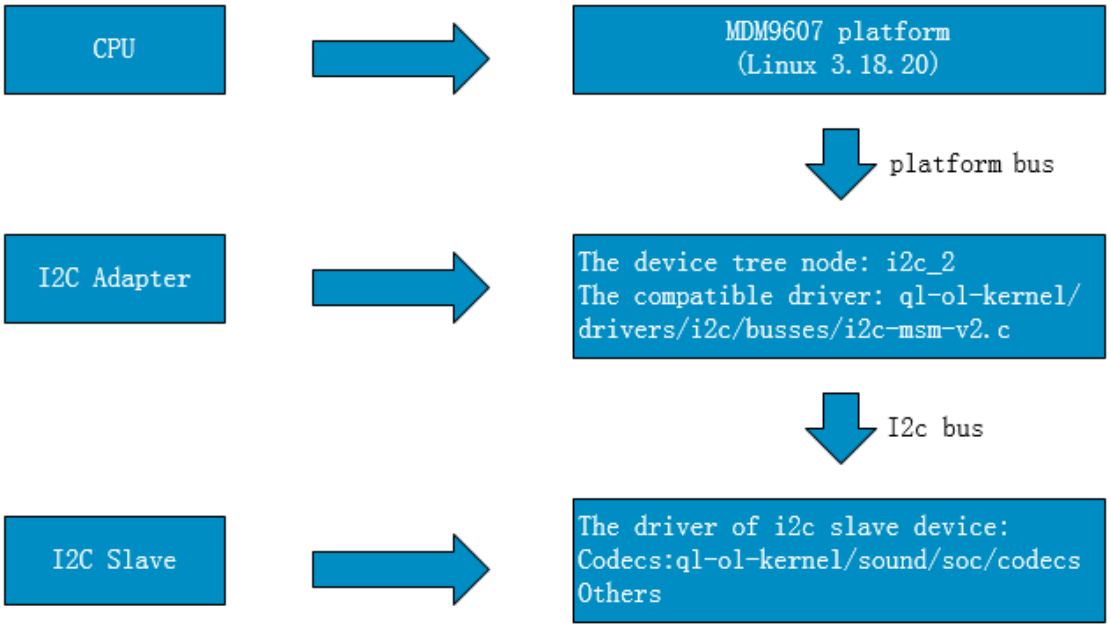
1 引言

文档从用户开发角度出发，介绍了硬件，软件驱动层，软件应用层等；可以帮助客户简易而快速的进行开发。

2 EC20 R2.1-QuecOpen I2C 说明

1. EC20 R2.1-QuecOpen 模块提供一路 I2C 接口，且模块在与 I2C 接口有关的应用中只能作为主设备；
2. 时钟支持：标准(100 kHz)， fast (400 kHz)， fast+ (1 MHz)；默认 clock 400K
3. 支持 7 位设备寻址，即一个 I2C 总线最多可以挂 2^7-1=127 个从设备，常用的 I2C 从设备如：codec, sensor 等；
4. 单次传输的最大长度是 2^16 - 1 bit；

平台 i2c 驱动架构



3 硬件电路设计推荐

3.1 带外部 Codec 芯片的 PCM 和 I2C 接口的参考设计

下图为带外部 Codec 芯片的 PCM 和 I2C 接口的参考设计：

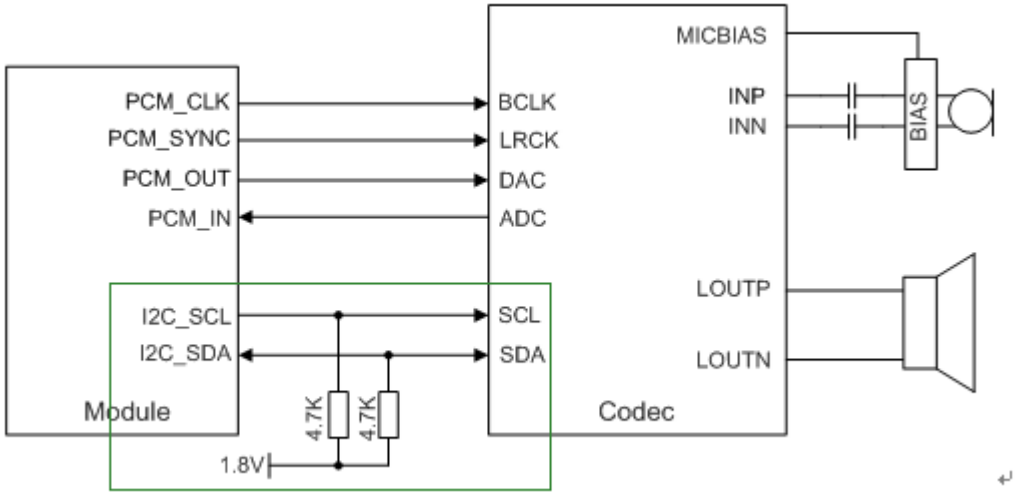


Figure 1：外部 Codec 芯片的 PCM 和 I2C 接口电路

4 驱动层及设备树软件适配

4.1 I2C 管脚使用

1. 以下表格中，非默认的复用功能需要在软件配置后才有效，请参考对应的功能章节进行软件配置；
2. I2C_SCL：需要外部 1.8V 上拉，不用则悬空;
3. I2C_SDA：需要外部 1.8V 上拉，不用则悬空;
4. I2C 接口可复用为 GPIO 和 UART CTS/RTS，默认功能为 I2C;
5.具体管脚使用参考《Quectel_EC20 R2.1_QuecOpen_GPIO_Assignment_Speadsheet》。

Table 1：I2C 接口管脚定义

引脚名	引脚号	I/O	功能描述		
			复用功能 1（默认）	复用功能 2	复用功能 3
I2C_SCL	41	OD	I2C_SCL_BLSP2	GPIO_7	UART_CTS_BLSP2
I2C_SDA	42	OD	I2C_SDA_BLSP2	GPIO_6	UART_RTS_BLSP2

4.2 I2C 设备树配置方法

4.2.1 I2C 控制器配置说明

Linux 的 I2C 体系结构分为 3 个组成部分：

I2C 核心：I2C 核心提供了 I2C 总线驱动和设备驱动的注册，注销方法，I2C 通信方法，与具体控制器无关的代码以及探测设备，检测设备地址的上层代码等。

I2C 总线（控制器）驱动：是对 I2C 硬件体系控制器端的实现，控制器由 CPU 控制，也可以直接集成在 CPU 内部。

I2C 设备驱动：即客户的 I2C 从设备驱动，是对 I2C 硬件体系结构中设备端的实现，设备一般挂接在受 CPU 控制的 I2C 控制器器上，通过 I2C 控制器与 CPU 交换数据。

以上三部分，一般用户只需要关心和修改 I2C 设备驱动；

I2C 总线驱动：即 I2C 控制器,mdm9607 平台使用的是 i2c-msm-v2 控制器；其硬件参数配置，如所兼容的 driver，引脚的选择，寄存器地址，CLK，中断号，DMA 引擎 API 的参数，以及系统休眠和工作时的管脚配置等 QuecOpen 都已经做好了，用户不需要关心和修改；

```
i2c_2: i2c@78b6000 { /* BLSP1 QUP4 */
        compatible = "qcom,i2c-msm-v2";
        #address-cells = <1>;
        #size-cells = <0>;
        reg-names = "qup_phys_addr";
        reg = <0x78b6000 0x600>;
        interrupt-names = "qup_irq";
        interrupts = <0 96 0>;
        qcom,clk-freq-out = <400000>;
        qcom,clk-freq-in  = <19200000>;
        clock-names = "iface_clk", "core_clk";
        clocks = <&clock_gcc clk_gcc_blsp1_ahb_clk>,
                 <&clock_gcc clk_gcc_blsp1_qup2_i2c_apps_clk>;

        pinctrl-names = "i2c_active", "i2c_sleep";
        pinctrl-0 = <&i2c_2_active>;
        pinctrl-1 = <&i2c_2_sleep>;
        qcom,noise-rjct-scl = <0>;
        qcom,noise-rjct-sda = <0>;
        qcom,master-id = <86>;
        dmas = <&dma_blsp1 14 64 0x20000020 0x20>,
               <&dma_blsp1 15 32 0x20000020 0x20>;
        dma-names = "tx", "rx";
```

另外，除非用户在 mdm9607 平台上根本不使用 I2C 控制器，那么用户可以用以下方式来关闭 I2C 控制器:以下方法至少执行一个

1.关闭控制器设备节点

```
--- a/ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi
+++ b/ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi
@@ -38,7 +38,7 @@

 //2016-01-21, modify by jun.wu, change i2c-4 to i2c-2
 &i2c_2 {
-       status = "ok";
+       status = "disabled";
 };
```

2.make kernel_menuconfig 去掉 I2C_MSM_V2 内核选项;

4.2.2 I2C 从设备的配置说明

默认情况下，在设备树 mdm9607.dtsi 的 I2c 控制器节点下，已经挂了几款 codec 从设备，指明了兼容的 driver 和从设备地址;

```
        //2016-02-23, add by jun.wu
        alc5616_codec@1b{
                compatible = "quec,quec-alc5616-i2c";
                reg = <0x1b>;
        };

        //2016-02-23, add by jun.wu
        nau8814_codec@1a{
                compatible = "quec,quec-nau8814-i2c";
                reg = <0x1a>;
        };

        //2016-12-21, add by sundy.wang
        tlv320aic3x_codec@18{
                compatible = "quec,quec-tlv320aic3x-i2c";
                reg = <0x18>;
        };

        quec_stub_codec@1{
                compatible = "quec,quec-stub-i2c";
                reg = <0x01>;
        };
```

若用户需要新增 i2c 设备，请从设备供应商处获取驱动和配置手册。

5 QuecOpen 应用层 API

5.1 用户编程说明

QuecOpen 项目 SDK 中提供了一套完整的用户编程接口；
参考路径：ql-ol-sdk/ql-ol-extsdk/

```
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk$ ls
docs  example  include  lib  target  tools
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk$
```

图中的 lib 目录下包含 quectel 提供的 API 接口库；include 目录是所有 API 的头文件；example 目录是提供的按功能划分的 API 使用示例；

这里我只介绍关于 I2C 相关的接口以及参考示例；
在进行 I2C 应用程序的编写需要依赖库 libql_peripheral.a；
头文件：ql_i2c.h

5.2 I2C API 介绍

当 I2C 控制器正常工作，i2c 从设备如 codec 也挂在总线上后，可以直接使用下面接口进行 codec 与 cpu 的通信；

int Ql_I2C_Init(char *dev_name);
初始化 i2c 设备
参数：dev_name: 设备名，如/dev/i2c-2 for EC20; /dev/i2c-4 for AG35
返回值：设备文件描述符，错误返回-1；

int Ql_I2C_Read(int fd, unsigned short slaveAddr,
unsigned char ofstAddr,
unsigned char* ptrBuff,
unsigned short length);
从某个 i2c 设备的某个偏移地址上读取指定字节长度的数据；出错时返回-1；
参数：fd: 设备文件描述符
slaveAddr: 设备地址，0x18(codec3104), 0x1A(codec8814), 0x1B(codec5616)
ofstAddr：偏移地址, (注：codec5616 一个寄存器有两个字节)
ptrBuff: 指针指向读取的数据
length: 读取的长度

int Ql_I2C_Write(int fd, unsigned short slaveAddr,
unsigned char ofstAddr,
unsigned char* ptrData,
unsigned short length);

向某个 i2c 设备的某个偏移地址上写入指定字节长度的数据；出错时返回-1；
参数：fd: 设备文件描述符
slaveAddr: 设备地址，0x18(codec3104), 0x1A(codec8814), 0x1B(codec5616,
ofstAddr：偏移地址，(注：codec5616 一个寄存器有两个字节)
ptrBuff： 待写数据指针
length： 写入长度

int Ql_I2C_Deinit(int fd);
关闭 i2c 设备.

参考：ql-ol-extsdk/example/i2c

6 I2C 功能测试验证

6.1 example 介绍及编译

ql-ol-extsdk/example/i2c 示例中，向指定 i2c 总线上的某个地址所在的从设备的一个寄存器地址写入一个字节数据，并再次读取出来；

```
#define I2C_DEV         "/dev/i2c-2"    //i2c-2 on EC20xx, i2c-4 on AG35
#define I2C_SLAVE_ADDR  0x18    //codec 3104
#define WHO_AM_I        0x02
#define WHO_AM_I_VALUE  0x12
```

进入到 ql-ol-sdk/ql-ol-extsdk/example/i2c 目录，make 生成 example_i2c 可执行程序，可以编译的前提必须是之前进行了交叉编译环境的初始化

source ql-ol-crosstool/ql-ol-crosstool-env-init

```
~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/i2c$ make
abi-gcc  -march=armv7-a -mfloat-abi=softfp -mfpu=neon  -O2 -fexpensive-optimizati
.h  -I./ -I./inc -I../../include -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol
/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include
fp-neon-oe-linux-gnueabi/usr/include -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/q
/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon
-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include/qmi -I
-neon-oe-linux-gnueabi/usr/include/qmi-framework   -c  -I./ -I./inc -I../../inclu
/lib/interface/inc -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/sys
FAG0130/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/i
rmv7a-vfp-neon-oe-linux-gnueabi/usr/include/data -I/home/gale/MDM9x07/SDK_FAG0130
lude/dsutils -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/sysroots/
G0130/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/inc
tsdk/example/i2c/../../lib -lrt -lpthread example_i2c.c
abi-gcc  -march=armv7-a -mfloat-abi=softfp -mfpu=neon -L./ -L/home/gale/MDM9x07/S
-L./ -L/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/i2c/../../li
le/i2c/../../lib/libql_peripheral.a  -o example_i2c
~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/i2c$ ls
ple_i2c.c  example_i2c.o  Makefile
~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/i2c$
```

6.2 功能测试

example 以 codec3104 为例：

1. 编译上传 example_i2c 到模块
   使用 adb push <example_i2c 在上位机路径> <模块内部路径，如/usrdata>
   或者
   使用串口协议 rz 上传

2. 若使用 OPEN_EVB，需要用跳线帽连接 J0201 的 I2C 排针

GPIO_6 连接 I2C_SDA
GPIO_7 连接 I2C_SCL
来连通硬件通路

3. 插上 codec3104 在 OPEN_EVB
4. 执行 example_i2c，如下图

```
root@mdm9607-perf:~# ./example_i2c
< Ql_I2C_Init=3 >
address=0x18, offset=2, byte_num=1, value[1]=18, value[2]=0
< write i2c value=0x12, iRet=1 >
< read i2c iRet=2, value=0x012 >
```

7 I2C 驱动调试方法

以上内容足以帮助用户让设备正常工作起来，但是总是会遇到一些意外的问题，不管是用户不当操作还是代码上的问题，那么我们需要通过一些调试手段来定位问题。

7.1 一般调试方法

1. QuecOpen 提供的 SDK 中，kernel log 的默认消息级别为 4(KERN_WARNING)，即内核在调用 printk() 时如果未指定消息级别，则默认为 4；
2. 控制台的默认打印级别的 7（KERN_DEBUG），即小于 7 的 kernel log 会被内核代码执行到，虽然执行到但此时是存放在内核 log_buffer 里面，当我们使用 dmesg 时才会把 buffer 的 log 输出到标准输出；

那么如果要想打开已经编译进 kernel 的 debug log，直接命令行 dmesg –n 8 修改；
或者在代码中默认修改：

```
--- a/ql-ol-kernel/include/linux/printk.h
+++ b/ql-ol-kernel/include/linux/printk.h
@@ -40,7 +40,7 @@ static inline const char *printk_skip_level(const char *buffer
 #define CONSOLE_LOGLEVEL_SILENT  0 /* Mum's the word */
 #define CONSOLE_LOGLEVEL_MIN     1 /* Minimum loglevel we let people use */
 #define CONSOLE_LOGLEVEL_QUIET   4 /* Shhh ..., when booted with "quiet" */
-#define CONSOLE_LOGLEVEL_DEFAULT 7 /* anything MORE serious than KERN_DEBUG */
+#define CONSOLE_LOGLEVEL_DEFAULT 8 /* anything MORE serious than KERN_DEBUG */
 #define CONSOLE_LOGLEVEL_DEBUG 10 /* issue debug messages */
 #define CONSOLE_LOGLEVEL_MOTORMOUTH 15 /* You can't shut this one up */
```

然而在很多驱动模块中，都是会定义自己的 DEBUG 编译宏，如果不打开这个宏，连 printk(KERN_DEBUG) 的代码都不会编译到,下面选中调试选项：
make kernel_menuconfig 选中下面三个 I2C 调试选项并编译烧录；

```
        <*> I2C support
        [*]   Enable compatibility bits for old user-space
        <*>   I2C device interface
        < >   I2C bus multiplexing support
        [*]   Autoselect pertinent helper modules
              I2C Hardware Bus support  --->
        < >   I2C/SMBus Test Stub
        [*]   I2C Core debugging messages
        [*]   I2C Algorithm debugging messages
        [*]   I2C Bus debugging messages
```

此时会看到较多的调试消息：

```
root@mdm9607-perf:/sys/kernel/debug/tracing/options# /home/root/example_i2c
< Ql_I2C_Init=3 >
address=0x18, offset=2, byte_num=1, value[1]=18, value[2]=0
< write i2c value=0x12, iRet=1 >
< read i2c iRet=2, value=0x012 >
root@mdm9607-perf:/sys/kernel/debug/tracing/options# dmesg -c
[ 2217.891271] i2c i2c-2: ioctl, cmd=0x707, arg=0xbee7cc34
[ 2217.891407] i2c i2c-2: master_xfer[0] W, addr=0x18, len=2
[ 2217.891511] i2c-msm-v2 78b6000.i2c: #2775 pm_runtime: resuming...
[ 2217.891572] i2c-msm-v2 78b6000.i2c: #2690 resuming...
[ 2217.892016] i2c-msm-v2 78b6000.i2c: xfer() mode:0 msg_cnt:1 rx_cbt:0 tx_cnt:2
[ 2217.892088] i2c-msm-v2 78b6000.i2c: #708 Starting FIFO transfer
[ 2217.892150] i2c-msm-v2 78b6000.i2c: QUP state after programming for next transfers
[ 2217.892207] i2c-msm-v2 78b6000.i2c: tag.val:0x2833081 tag.len:4 (null)
[ 2217.892260] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x02833081
[ 2217.892316] i2c-msm-v2 78b6000.i2c: data: 0x2 0x12 0x35 0xcf
[ 2217.892370] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x00001202
[ 2217.893014] i2c-msm-v2 78b6000.i2c: NONE: msgs(n:1 cur:0 tx) bc(rx:0 tx:2) mode:FIFO slv_addr:0x18 MSTR_STS:0
[ 2217.893470] i2c i2c-2: ioctl, cmd=0x707, arg=0xbee7cc30
[ 2217.893538] i2c i2c-2: master_xfer[0] W, addr=0x18, len=1
[ 2217.893590] i2c i2c-2: master_xfer[1] R, addr=0x18, len=1
[ 2217.893776] i2c-msm-v2 78b6000.i2c: xfer() mode:0 msg_cnt:2 rx_cbt:1 tx_cnt:1
[ 2217.893831] i2c-msm-v2 78b6000.i2c: #708 Starting FIFO transfer
[ 2217.893892] i2c-msm-v2 78b6000.i2c: QUP state after programming for next transfers
[ 2217.893948] i2c-msm-v2 78b6000.i2c: tag.val:0x1823081 tag.len:4 (null)
[ 2217.894001] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x01823081
[ 2217.894056] i2c-msm-v2 78b6000.i2c: data: 0x2 0x83 0x35 0xcf
[ 2217.894109] i2c-msm-v2 78b6000.i2c: tag.val:0x1873181 tag.len:4 (null)
[ 2217.894161] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x87318102
[ 2217.894214] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x00000001
[ 2217.894660] i2c-msm-v2 78b6000.i2c: NONE: msgs(n:2 cur:0 tx) bc(rx:1 tx:1) mode:FIFO slv_addr:0x18 MSTR_STS:0
[ 2217.894789] i2c-msm-v2 78b6000.i2c: #490 IN-FIFO :0x00120187
[ 2217.894844] i2c-msm-v2 78b6000.i2c: (null)
[ 2218.140356] i2c-msm-v2 78b6000.i2c: #2763 pm_runtime: suspending...
[ 2218.140483] i2c-msm-v2 78b6000.i2c: #2665 suspending...
root@mdm9607-perf:/sys/kernel/debug/tracing/options#
```

7.2 使用 kernel tracer 进行调试

QuecOpen SDK 默认开启了 kernel hacking 中的 kernel debugfs 和 kernel tracer 功能，kernel tracer 功能很强大，常用来调试内核；

```
root@mdm9607-perf:/sys/kernel/debug/tracing# ls
README                instances             trace
available_events      options               trace_clock
available_tracers     per_cpu               trace_marker
buffer_size_kb        printk_formats        trace_options
buffer_total_size_kb  saved_cmdlines        trace_pipe
current_tracer        saved_cmdlines_size   tracing_cpumask
events                saved_tgids           tracing_on
free_buffer           set_event             tracing_thresh
root@mdm9607-perf:/sys/kernel/debug/tracing#
```

1. 打开内核栈追踪
   echo 1 > options/stacktrace
2. 打开 printk 输出的 log
   echo 1 > events/printk/enable
3. 打开 i2c event 调试
   echo 1 > events/i2c/enable
4. 发起一次 i2c 访问

5. cat trace 可以内核接口的调用状态；