



完 美 无 线 体 验

FIBOCOM L610 系列

OpenCPU 编程指南

文档版本：V1.0.2

更新日期：2020-04-23



适用型号

序号	产品型号	说明
1	L610 系列	用于 L610 OpenCPU 固件项目

版权声明

版权所有©2020 深圳市广和通无线股份有限公司。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

商标申明



为深圳市广和通无线股份有限公司的注册商标，由所有人拥有。

版本记录

文档版本	编写人	主审人	批准人	更新日期	说明
V1.0.2	董福	王海亮 程涛涛 梁景钦	邓利斌	2020-04-24	1、修正 7.2 节 2、新增 9.26 节 3、新增 8.1—8.7 章节
V1.0.1	董福	王海亮 程涛涛 梁景钦	邓利斌	2020-04-23	1、修正 9.6 节 I2C 2、修正 9.8 节 UART 3、新增 9.24 节 4、新增 9.25 节 5、修正第 10 章
V1.0.0	董福	王海亮 程涛涛 梁景钦	邓利斌	2020-02-20	第一版发布

目录

1	前言	6
2	L610 平台简介	7
3	L610 及平台支持的基本功能	8
4	开发板简介	9
5	常用工具简介	10
6	OpenCPU 简介	11
6.1	OpenCPU 开机运行	12
6.2	OpenCPU event 通知机制	13
7	OpenCPU FLASH 和 RAM	14
7.1	OpenCPU FLASH	14
7.2	OpenCPU RAM	15
8	搭建 OpenCPU 开发环境和编译	17
8.1	简介	17
8.2	开发环境搭建	17
8.3	ubuntu 下载地址	17
8.4	ubuntu 安装	17
8.5	所需软件下载	17
8.6	用户程序编译	18
8.7	如何调试	19
8.8	如何添加代码	19
8.9	如何添加库	20
8.10	如何修改生成的镜像名称	22
9	已封装的 API 功能说明	23
9.1	OSI 接口	23
9.2	PDP/LWIP 相关接口	23
9.3	OTA/固件升级	25
9.4	休眠/唤醒	26
9.5	定时器	26
9.6	I2C	27
9.7	SPI	28
9.8	UART	28
9.9	USB	29

9.10	ADC.....	29
9.11	GPIO	30
9.12	看门狗	30
9.13	MQTT	30
9.14	OneNET	31
9.15	LCD	31
9.16	摄像头	32
9.17	LBS/WIFISCAN	33
9.18	CELLINFO.....	34
9.19	发送 AT 命令的 API	34
9.20	文件系统.....	34
9.21	接收异步事件和接收 AT 响应 callback.....	35
9.22	Debug API	36
9.23	OTHER API	37
9.24	SSL socket API.....	38
9.25	HTTP API	38
9.26	客户定制 API	38
10	其他需求功能.....	40
11	参考 demo.....	41
12	其他注意事项.....	44
12.1	prvInvokeGlobalCtors()接口使用	44
13	Q&A	45
13.1	如何烧录	45

1 前言

本文旨在帮助客户快速了解 L610 OpenCPU 模块开发环境、调用 API 和 demo 使用。

2 L610 平台简介

L610 是基于 LTE cat1 开发的一款模组，芯片具备 LTE(CAT1)/GSM 双模通信能力，支持 TDD/FDD（理论 DL10Mbps、UL5Mbps）数据通信能力，支持 volte、CSFB、SMS、TCP、UDP、FTP、CoAP、LWM2M、MQTT、HTTP、HTTPS、DNS、PING、IPV4V6 等协议，满足终端通信设备客户需求。

3 L610 及平台支持的基本功能

功能	说明
CPU	ARM5, 主频 500MHZ
Modem	Lte Cat1 (DL: 10Mbps,UL: 5Mbps) TDD&FDD,GSM
OS	FreeRtos Kernel: V10.0.1
Memory	内部集成 8MB FLASH 和 16MB PSRAM, 最大支持 16MB 外挂 flash
Band	Lte CAT1:Band1/3/5/8/34/39/40/41;GSM850/900/1800/1900, 均不支持分集接收
支持协议	TCP/UDP/FTP/CoAP/LWM2M/MQTT/HTTP/HTTPS/DNS/ICMP/PING/IPV4/IPV6
多媒体	QVGA15fps@MPEG
Security	支持 Security Boot
Connectivity	支持内置 WIFISCAN 暂不支持 BT 暂不支持外挂 WIFI
LCD	QVGA@SPI, HVGA@MIPI
Camera	0.3M
LWIP	支持内置协议栈 V2.0.1
定位	支持 LBS, 支持内置 WIFISCAN, 不支持 GPS
Audio	MP3、WAV、PCM、AMR (平台开发中, V1.3 基线支持)
Size	8.9*8.9mm, BGA, 301Ball, 0.4mm
DECODEC	H.264/H.263/MPEG4/MJPG@15fps
INCODEC	MPEG4encodeQVGA@12fps MJPEGencodeQVGA@15fps
Others	支持 IRAT 和 VOLTE, CSFB(GSMCSCALL), PPP, ECM(linux), SMS, 数据通信

4 开发板简介

请参阅《FIBOCOM ADP-L610-CN-00 使用指南》。

FIBOCOM
Confidential

5 常用工具简介

CP 工具: ARMTRACER(ArmTracer_V6.1.5_User.7z),如遇权限问题找 FAE 提供新签名的工具;

AP 工具: coolwatcher(windows: UIS8910DM_cooltools_win32_R2.0.0002/

Linux: 8910DM_cooltools-custom-xenial_P3.R2.0.0002.tgz)

烧录工具: RESEARCHDOWNLOAD_R23.0.0001.7z

Windows 驱动: 8910_module_usb_driver_20200113_signed.7z

Linux(Ubuntu:16.04)驱动: L610_linux_driver.sh

Windows 下抓取 log 指导见: 《FIBOCOM L610 系列 LOG 抓取指南_Windows》

Linux 下抓取 log 指导见: 《FIBOCOM L610 log 抓取指南_Linux》

主版本及 OpenCPU 客户应用程序下载指导: 《FIBOCOM L610 系列 版本下载指南_Windows》

FIBOCOM
Confidential

6 OpenCPU 简介

使用 OpenCPU，客户无需 MCU，无需通过 AT 指令集与模块通信，而是客户的应用程序作为主固件程序的一部分调用集成开发好的 API，从而实现想要的功能。比如启/停内置协议栈拨号联网,应用程序调用封装好 LWIP 内部接口的 API 进行数据收发，调用封装好的 HTTP、MQTT、FTP 等 API 接口实现不同的业务需求，调用封装好的 UART、I2C、SPI 等 API 接口实现对外围设备的控制，比如 LCD、camera 等，应用程序调用封装好的 OSAPI 进行创建线程，消息队列，信号量，调用封装好的 GPIOAPI 接口对 GPIO 的管理等等需求。API 实现功能与 AT 指令实现功能等同，几乎所有的 AT 指令集能做的事均可开发成 API 供客户应用程序调用，从而满足客户各类应用场景。

常规AT指令集系统

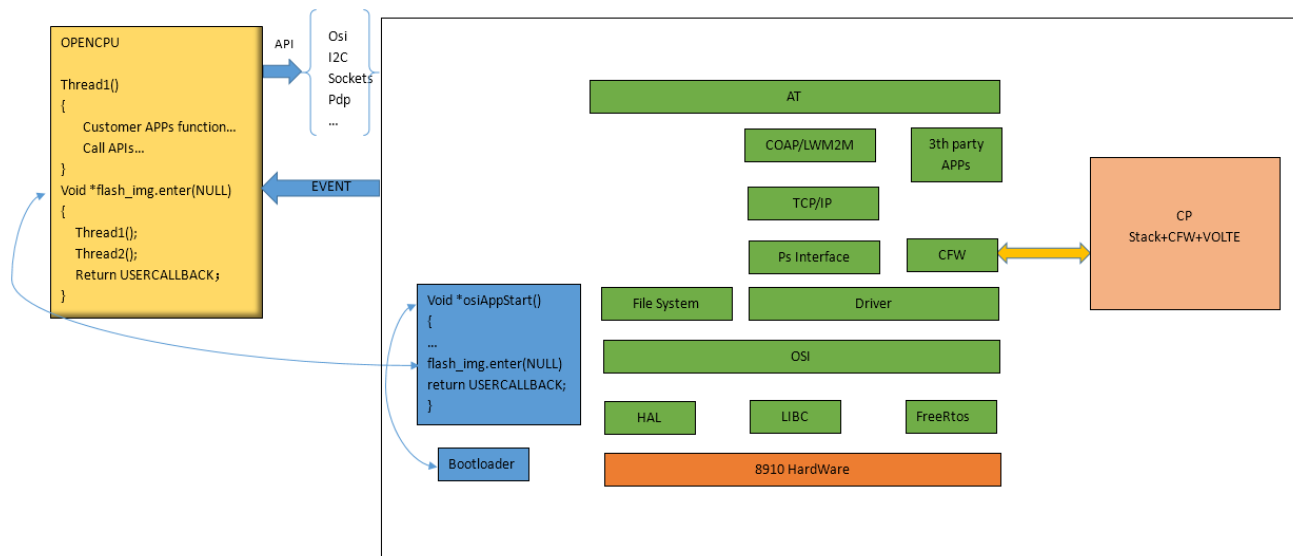


Opencpu API 系统



OpenCPU 好处是节省 MCU 硬件，从而大大节省客户成本，MCU 收发 AT 想要做的事均能通过软件去实现。OpenCPU 集成客户应用程序和调用封装好的 API，客户端开发需要能够独立编译，生成可执行文件烧录进 FLASH 能够正常加载运行。

6.1 OpenCPU 开机运行



如上图所示：客户的应用程序在 **bootloader** 之后加载主版本 **application** 程序的时候会去加载存放在 **FLASH 0x2C0000**(应用程序相对起始地址)应用程序，并对存放在此处起始地址几百 **byte** 的头部校验，检验客户应用程序准确无误后，跳转至客户应用程序起始栈地址运行，即客户看到的应用程序入口函数。

```

Void *appimg_enter(void *param)
{
    ...
    return user_callback;
}
    
```

将函数结构体变量地址作为入口函数的返回值；

函数结构体内部定义两个 **callback**，一个是异步事件 **event** 上报，一个是接收调 **AT API** 发 **AT** 命令的 **response** 的 **callback**，具体使用请参见代码；



注意：

客户应用程序的入口函数 **void *appimg_enter(void *param)**对于系统而言仅仅就是一个空函数，参考 **demo** 可能就是打印一行 **log**，开机加载执行运行退出后就结束了，不会再次加载；

所以客户的应用程序必须跑在此接口内创建的线程内，线程内调 **API** 接口处理应用程序各个功能，比如联网发数据等。

6.2 OpenCPU event 通知机制

沿用以往项目实现方案：继续保留 OpenCPU 接收异步接口事件通知机制以及调用发 AT 命令的 API 接收 AT 响应的机制；

API 分 block 和 non-block 的，因为客户调用 API 时，有些 API 立即执行完毕立即告知结果。有些 API 会通知其他线程处理或 callback 里处理结果。根据执行结果跟 API 执行结束是否同步，区分 API 为 block 和 non-block 接口。

对于 non-block API 执行结束，一件事还没有有结果，这件事是否成功或失败或需要执行计算后的结果，则需要通过 event 通知机制通知到客户应用程序。客户应用程序加信号量等待或加延时等待执行结果。

定义接收 event 的回调函数：

```
struct FIBO_CALLBACK_S
{
    fibo_at_resp_t at_resp;
    fibo_signal_t fibo_signal;
};
```

其中 at_resp 是处理调发 AT 的 API 发出某一条 AT 时接收 AT 返回 response 的回调函数。

fibo_signal 是处理接收异步 API 执行结果上报 event 的回调函数。

请参考 demo 代码。

SignalEvent 定义在 fibo_OpenCPU_comm.h

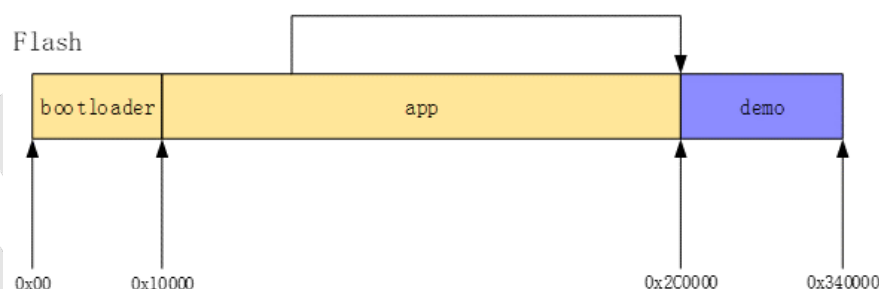
7 OpenCPU FLASH 和 RAM

7.1 OpenCPU FLASH

目前平台内部集成 8MBFLASH，各个软件模块占用 FLASH 划分如下：

Partition	Range	Size	Mount Point
bootloader	0..0x10000	64KB	
application	0x10000..0x340000	3.2MB	
system FS	0x340000..0x4a0000	1.4MB	/
modem FS	0x4a0000..0x7e0000	3.2MB	/modem
factory FS	0x7e0000..0x800000	128KB	/factory

其中从 application 里划分 512KB 空间给 OpenCPU 使用：



FLASH 软件模块宏定义：

```
// Flash layout
#define CONFIG_NOR_PHY_ADDRESS 0x60000000
#define CONFIG_NOR_EXT_PHY_ADDRESS 0x70000000
#define CONFIG_BOOT_FLASH_OFFSET 0x0
#define CONFIG_BOOT_FLASH_SIZE 0x10000
#define CONFIG_APP_FLASH_OFFSET 0x10000
#define CONFIG_APP_FLASH_SIZE 0x2b0000
#define CONFIG_APPIMG_FLASH_OFFSET 0x2c0000
#define CONFIG_APPIMG_FLASH_SIZE 0x80000
#define CONFIG_FS_SYS_FLASH_OFFSET 0x340000
#define CONFIG_FS_SYS_FLASH_SIZE 0x160000
#define CONFIG_FS_MODEM_FLASH_OFFSET 0x4a0000
#define CONFIG_FS_MODEM_FLASH_SIZE 0x340000
#define CONFIG_FS_FACTORY_FLASH_OFFSET 0x7e0000
#define CONFIG_FS_FACTORY_FLASH_SIZE 0x20000
```



注意：

其中 FS 分区实测剩余 1.2MB 左右空间，还会划分一部分空间给到主版本 FOTA 差分包和客户应用程序 FOTA 升级包用，FOTA 升级包大小不固定，所以不能全部划分 FS 剩余空间给客户存储固定文件使用。

目前 TTS (text-to-sound) 的库也编译进主版本, 实际存放在 application flash 空间, 会占用 700K 左右 flash 空间, 导致实际剩余 application flash 空间很小, 大约 100K 左右, 如果不需要支持 TTS 的客户对 FLASH 空间有要求的客户, 可以移除 TTS。

7.2 OpenCPU RAM

由于 OpenCPU 客户应用程序可独立主版本在外部独自编译, 所以必须得给这部分代码分配一块独立的静态存储区域的 RAM 资源(默认 64K, 可改大), 编译期间给代码里定义的初始化的全局变量计算可分配静态存储区域内存。



目前给客户分配的 RAM 静态存储区大小为 64K, 只会限制客户定义的总的全局变量大小及其他放在静态 RAM 存储区域变量的大小, 动态存储区堆栈资源与主版本共享剩余内存资源, 大约 2.5~2.8M。

- 扩大 OpenCPU RAM 静态存储空间, 堆栈所在动态存储空间就会变少。
- 单个线程最大可申请 250K 左右大小空间。
- 如果客户在编写应用程序代码时, 如果有很大的全局变量或其他存放在静态存储区的变量数据, 建议使用指针, 通过 MALLOC 申请 heap 资源。指针类型数据变量实际只占用静态存储区几个 byte, 超过 64K, 编译会报错。
- 建议客户创建线程申请内存资源大小 1k---十几 K, 无需申请大几十 K 甚至上百 K, 如有很大的局部数组变量或很大的结构体变量, 建议使用指针, 通过 MALLOC 申请 heap 资源。如果很大的数组变量超过线程申请的线程运行深度, 会导致线程内存溢出而 dump, 系统将无法正常运行;
- 建议创建多线程协同工作, 通过消息队列机制驱动不同线程协同工作。



注意:

64K RAM 指的是客户 APP 设置全局变量或者静态变量所能占用的上限, 这个和提供给客户的 fibo_malloc 的 RAM 有本质区别。fibo_malloc 是申请主固件的内存区域, 目前最大有 2.5M 空间可供

客户申请。

FIBOCOM
Confidential

8 搭建 OpenCPU 开发环境和编译

8.1 简介

L610 OpenCPU 系列模块的程序分为内部程序和外部程序。

1) 内部程序，也称固件程序，在模块生产时植入 FLASH 相应的区域。

2) 外部程序，也称应用程序或用户程序，在模块的 FLASH 中独立分配的区域。其中，用户程序的源代码是开放的，用户可以根据自己的需要，依照开发接口对用户程序进行定制、编译，并下载到模块中运行。

8.2 开发环境搭建

必须使用 ubuntu 16.04 版本，其它版本不支持。

8.3 ubuntu 下载地址

ubuntu 下载地址：

<http://mirrors.aliyun.com/ubuntu-releases/16.04/>

<http://old-releases.ubuntu.com/releases/16.04.4/>

对应具体需要下载的 ubuntu iso 名字为：

[ubuntu-16.04-desktop-amd64.iso](#)



注意：

为了避免不必要的麻烦，请务必使用此 iso 进行 ubuntu 的安装。

8.4 ubuntu 安装

使用一个 U 盘，使用 UltraISO.exe 工具将 iso 制作成 U 盘启动安装。

选择一台服务器作为裸机（个人电脑也可以，经测试，用户程序的编译速度在个人电脑也很快速），插入 U 盘，BIOS 选择从 U 盘启动，进行 ubuntu 的安装。

8.5 所需软件下载

Ubuntu 安装完毕，需要可以连接 internet。连接 internet 成功后，使用 ubuntu 的 root 权限，执行以下指令：

```
apt install build-essential python3 python3-tk qtbase5-dev
```

```
apt install libc6:i386 libstdc++6:i386 zlib1g:i386
```

以上用于安装编译用户程序所必须的库以及软件。

8.6 用户程序编译

会为用户提供一个压缩包：core_sdk.tar.gz

编译步骤：

1) 将 core_sdk.tar.gz 拷贝到 ubuntu 某个目录下，比如：/home/user/

2) 在 ubuntu shell 界面进行操作：

```
cd /home/user/
```

```
tar xzvf core_sdk.tar.gz //此指令用于解压缩
```

3) 解压缩完成，依次运行如下编译指令：

```
. tools/core_launch.sh
```

```
cout
```

```
cmake ../../ -G Ninja
```

```
ninja
```

如下图所示：

```
lobin@fibocom:~/core_sdk$ . tools/core_launch.sh
lobin@fibocom:~/core_sdk$ cout
lobin@fibocom:~/core_sdk/out/appimage_debug$ cmake ../../ -G Ninja
-- Configuring done
-- Generating done
-- Build files have been written to: /home/lobin/core_sdk/out/appimage_debug
lobin@fibocom:~/core_sdk/out/appimage_debug$
lobin@fibocom:~/core_sdk/out/appimage_debug$ ninja
ninja: no work to do.
lobin@fibocom:~/core_sdk/out/appimage_debug$
lobin@fibocom:~/core_sdk/out/appimage_debug$ ninja clean
[1/1] Cleaning all built files...
Cleaning... 15 files.
lobin@fibocom:~/core_sdk/out/appimage_debug$
lobin@fibocom:~/core_sdk/out/appimage_debug$ ninja
[6/6] Linking C executable hex/hello_flash.elf
lobin@fibocom:~/core_sdk/out/appimage_debug$
lobin@fibocom:~/core_sdk/out/appimage_debug$
```



注意：

以上前三条指令都是用于编译环境配置，第四条指令（ninja）用于编译链接。若已编译完成，想重新编译则使用：

```
ninja clean
```

```
ninja
```

4) 编译生成文件存于：./out/appimage_debug/hex/

其中最关键文件为 hello_flash.pac。此文件用于下载到设备上启动运行。

8.7 如何调试

与内部程序的调试一致，使用 coolwatcher 工具进行 log 抓取。这样的好处是，内部程序和外部程序可同时调试并抓取 log，并可生成一个 log 文件。为以后解决内部程序和外部程序的问题提供方便。

打印接口可参考 hello_world.c 文件中：OSI_LOGI

hello_world.c 文件所在目录：与解压缩的 core_sdk.tar.gz 在同一个目录下。

extern INT32 fibo_textTrace(INT8 *fmt, ...)。

其打印数据会显示到 coolwatcher 工具上。

8.8 如何添加代码

客户拿到 tar 包后，通过 tar-zxvfcore_sdk.tar.gz 解压到某个文件夹后，可以看到有如下文件：

cmake	2020-02-28 15:26
components	2020-02-28 15:26
ldscripts	2020-02-28 15:26
prebuilts	2020-02-28 15:26
tools	2020-02-28 15:26
.clang-format	2019-12-23 9:32
.gitignore	2019-12-23 9:32
CMakeLists.txt	2019-12-23 9:32
core_sdk.tar.gz	2020-02-28 15:26
demo.c	2019-12-23 9:32
hello_world.c	2020-02-14 17:12

其中 hello_world.c 和 demo.c 是示例代码；

实际客户开发中可自己定义.c 和.h 文件名，以及可以新建文件夹存放客户的应用程序；

比如在 CMakeLists.txt 所在路径下创建 test 文件夹，该文件夹创建 src 文件夹和 include 文件夹，分别存放.c 文件和.h 文件，亦可只需创建一个文件夹里面同时存放.c 和.h，比如 src 里面创建 test.c 和 test2.c；

只需修改 CMakeLists.txt 宏 CONFIG_APPIMG_LOAD_FLASH 控里添加，以及添加新增加的头文件路径。



注意：

所有代码只允许一个入口函数 void*appimg_enter(void*param)，其他.c 文件里也有则编译会报错，注意该入口函数所在文件名及其路径；

目前支持此宏 CONFIG_APPIMG_LOAD_FLASH，不支持 CONFIG_APPIMG_LOAD_FILE，所以无需把代码添加到 CONFIG_APPIMG_LOAD_FILE 所控的 cmake 代码里。

```
if(CONFIG_APPIMG_LOAD_FLASH)
    set(target hello_flash)
    add_appimg(${target} ${flash_ldscrip
        hello_world.c
        demo.c
        test/src/test.c
        test/src/test1.c)
```

```
project(core C CXX ASM)

include_directories(components/include components/newlib/include test/include)

function(cpp_only target file)
```

添加新增头文件路径，如果新建多个头文件路径，在此处往后添加即可。

如果客户应用程序文件过多，可以在 CMakeLists.txt 里增加一个变量 LIB_SOURCES_FILE，记录所有.c 路径；后在 add_appimg 里引用此变量即可。

见如下图：

比如 abc 文件夹下的.c,如果有多个.c 所在文件夹，在 file 那行脚本语句里追加即可；

```
113:
114:     add_custom_command(OUTPUT ${pac_file}
115:         COMMAND python3 ${pacgen_py} pac-gen ${pac_config} ${pac_file}
116:         DEPENDS ${pacgen_py} ${pac_config} ${pac_dep}
117:         WORKING_DIRECTORY ${SOURCE_TOP_DIR}
118:     )
119:     add_custom_target(${target}_pacgen ALL DEPENDS ${pac_file})
120: endif()
121:
122: file(GLOB LIB_SOURCES_FILE ${CMAKE_SOURCE_DIR}/abc/*.c)
123:
124:
125: if(CONFIG_APPIMG_LOAD_FLASH)
126:     set(target hello_flash)
127:     add_appimg(${target} ${flash_ldscript}
128:         ${LIB_SOURCES_FILE})
129:     target_link_libraries(${target} PRIVATE ${libc_file_name} ${libm_file_name} ${libgcc_file_name})
130:
131:     set(pac_config ${out_hex_dir}/${target}.json)
132:     set(pac_file ${out_hex_dir}/${target}.pac)
133:     pac_init_fdl(init_fdl ${pac_config})
134:     execute_process(
135:         COMMAND python3 ${pacgen_py} ${init_fdl}
136:         cfg-image -i APPIMG -a ${CONFIG_APPIMG_FLASH_ADDRESS} -s ${CONFIG_APPIMG_FLASH_SIZE}
137:         -p ${out_hex_dir}/${target}.img ${pac_config}
138:         dep-gen --base ${SOURCE_TOP_DIR} ${pac_config}
139:         OUTPUT_VARIABLE pac_dep
140:         OUTPUT_STRIP_TRAILING_WHITESPACE
141:         WORKING_DIRECTORY ${SOURCE_TOP_DIR}
142:     )
143:
```

8.9 如何添加库

首先在 cmake/toolchain-gcc.cmake 里增加库，并把库文件放在 components/newlib/armca5 路径下；

比如加载新的库 libtest.c,在此文件里声明 libtest_file_name, 并在 CMakeLists.txt 引用即可。

```
14: set(CMAKE_SYSTEM_NAME Generic)
15:
16: find_program(CMAKE_C_COMPILER ${CROSS_COMPILE}gcc)
17: find_program(CMAKE_CXX_COMPILER ${CROSS_COMPILE}g++)
18: find_program(CMAKE_OBJCOPY ${CROSS_COMPILE}objcopy)
19: find_program(CMAKE_OBJDUMP ${CROSS_COMPILE}objdump)
20: find_program(CMAKE_AR ${CROSS_COMPILE}ar)
21: find_program(CMAKE_RANLIB ${CROSS_COMPILE}ranlib)
22: find_program(CMAKE_READelf ${CROSS_COMPILE}readelf)
23: find_program(CMAKE_NM ${CROSS_COMPILE}nm)
24: find_program(CMAKE_LD ${CROSS_COMPILE}ld)
25:
26: set(CMAKE_EXECUTABLE_SUFFIX_C .elf)
27: set(CMAKE_EXECUTABLE_SUFFIX_CXX .elf)
28:
29: if(CONFIG_CPU_ARM_CA5)
30:     set(abi_options -mcpu=cortex-a5 -mtune=generic-armv7-a -mthumb -mfpu=neon-vfpv4
31:         -mfloat-abi=hard -mno-unaligned-access)
32:     set(partial_link_options)
33:     set(libc_file_name ${CMAKE_CURRENT_SOURCE_DIR}/components/newlib/armca5/libc.a)
34:     set(libm_file_name ${CMAKE_CURRENT_SOURCE_DIR}/components/newlib/armca5/libm.a)
35:     set(libtest_file_name ${CMAKE_CURRENT_SOURCE_DIR}/components/newlib/armca5/libtest.a)
36: endif()
37:
38: if(CONFIG_CPU_ARM_CM4F)
39:     set(abi_options -mcpu=cortex-m4 -mtune=cortex-m4 -mthumb -mfpu=fpv4-sp-d16
40:         -mfloat-abi=hard -mno-unaligned-access)
41:     set(partial_link_options)
42:     set(libc_file_name ${CMAKE_CURRENT_SOURCE_DIR}/components/newlib/armcm4f/libc.a)
43:     set(libm_file_name ${CMAKE_CURRENT_SOURCE_DIR}/components/newlib/armcm4f/libm.a)
44: endif()
45:
46: if(CONFIG_CPU_MIPS_XCPU)
47:     set(abi_options -march=xcpu -mtune=xcpu -EL -mips16 -msoft-float -mno-gpopt -G0
48:         -mdisable-save-restore)
49:     set(partial_link_options -EL)
50:     set(libc_file_name ${CMAKE_CURRENT_SOURCE_DIR}/components/newlib/xcpu/libc.a)
51:     set(libm_file_name ${CMAKE_CURRENT_SOURCE_DIR}/components/newlib/xcpu/libm.a)
52: endif()
```

其次在 CMakeLists.txt 里引用此库。

```
if(CONFIG_APPING_LOAD_FLASH)
    set(target hello_flash)
    add_appimg(${target} ${flash_ldscript}
        hello_world.c
        demo.c)
    target_link_libraries(${target} PRIVATE ${libc_file_name} ${libm_file_name} ${libgcc_file_name} ${libtest_file_name})
    set(pac_config ${out_hex_dir}/${target}.json)
    set(pac_file ${out_hex_dir}/${target}.pac)
    pac_init_fdl(init_fdl ${pac_config})
    execute_process(
        COMMAND python3 ${pacgen_py} ${init_fdl}
        cfd-image -i APPING -a ${CONFIG_APPING_FLASH_ADDRESS} -s ${CONFIG_APPING_FLASH_SIZE}
        -p ${out_hex_dir}/${target}.img ${pac_config}
        dep-gen --base ${SOURCE_TOP_DIR} ${pac_config}
        OUTPUT_VARIABLE pac_dep
        OUTPUT_STRIP_TRAILING_WHITESPACE
        WORKING_DIRECTORY ${SOURCE_TOP_DIR}
    )
endif()
```



注意:

如果编译 ld 时提示标准 C 库未定义, 则需将标准 C 库函数放到 core_export.list, 重新出主版本方能支持链接失败问题。

8.10 如何修改生成的镜像名称

如果客户想改自己的应用程序镜像文件名称，则可按照如下方式修改。

```
if(CONFIG_APPIMG_LOAD_FLASH)
    set(target hello_flash)
    add_appimg("${target}" "${flash_ldscript}"
        hello_world.c
        demo.c
        test/src/test.c
        test/src/test1.c)
```

随便修改，比如改为 abc

或直接修改生成的 image*.PAC 文件名称。

FIBOCOM
Confidential

9 已封装的 API 功能说明

具体各个 API 请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

具体各个子模块测试代码请参考提供给客户的 demo 代码。

如下是对各个子模块接口简介和简述开发调试注意事项。

9.1 OSI 接口

- 功能及特性:

这一部分主要是封装 freeRTOS 的 API，比如创建线程。创建消息队列，接收消息队列等，信号量，锁等

上层应用程序建议多线程协同工作，不同线程间通过消息队列触发，共享全局变量时注意加锁；

- API 及涉及 signal event:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》；

无 signal event。

- 其他注意事项:

(1) 请参考 test 代码 osi_demo.c。

(2) 单个线程栈深最大可申请 250K 左右，不建议申请这么大，申请 1K 至十几 K 即可，如果线程内函数内部有过大局部变量，建议使用 fibo_malloc 申请 heap 资源。

(3) 如果动态创建线程或信号量或 fibo_malloc 申请一块内存资源等，请务必及时删除线程或信号量及时回收内存资源，防止一直循环动态创建申请内存资源导致内存耗尽而系统无法正常运行。

(4) 建议线程一次性创建后一直有效。

(5) 建议通过 fibo_get_heapinfo 接口不定时查询内存 freesize 剩余大小从而判断内存是否有泄露。

9.2 PDP/LWIP 相关接口

- 功能及特性:

这一部分主要是封装的拨号连接和内置协议栈接口。

- API 及涉及 signal event:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

典型 API: 为了保持跟以往项目接口一致性，如下接口继续保留，但受平台限制，接口均是异步

INT32 fibo_PDPActive();

该接口主要是激活 LTEattach 建立的那路默认承载（epsid5）并映射到 CID1 上，如果非特殊卡及特殊网络，APN 和 PDP 鉴权等参数可置 NULL。

如果有特殊 SIM 卡或特殊网络只支持 LTEattach 建立的那路默认承载且对 APN 及 PDP 鉴权有要求的，请用此接口；

其中 IP 无法通过接口退出时输出，通过 GAPP_SIG_PDP_ACTIVE_ADDRESS 信号上报或发 fibo_PDPStatus 查询；

INT32 fibo_PDPRelease();

主动断开 PDP，跟上面激活接口对应使用，默认断开激活的 attach 那路默认承载 CID1；

INT32 fibo_asyn_PDPActive();

定义并激活某一路 PDP，集成 cgdcont 和 cgact 共同功能，设置一路 CID 并激活；主要用于多路业务场景。

INT32 fibo_asyn_PDPRelease();

断开某一路 PDP 连接，需指定 CID。



注意：

客户无特殊需求，建议使用 fibo_PDPActive();接口即可。

PDP 异步通知事件：

GAPP_SIG_PDP_RELEASE_IND: PDP 主动断开成功后上报此 signal;

GAPP_SIG_PDP_ACTIVE_IND: PDP 激活成功上报此 signal;

GAPP_SIG_PDP_ACTIVE_OR_DEACTIVE_FAIL_IND:

PDP 激活或失败时上报此 signal，携带两个参数上来，一个是标记哪一路 CID 激活/去激活失败，一个是标记当前是做激活还是去激活；

GAPP_SIG_PDP_DEACTIVE_ABNORMALLY_IND: PDP 异常被动断开，非客户程序主动发起的断开，比如网络侧发起的断开 PDP，或长时间无信号掉网或其他原因导致的 CP 侧本地断开 PDP（断开 AP 和 CP 的连接并清除 context）。

- 其他注意事项：

(1) 建立 sockets 前，必须激活 PDP，激活 PDP 前必须注册上网络，顺序是：

fibo_getRegInfo→PDP_active→fibo_getHostByName→sockets 建立→收发数据---按需 closesockets

先查询注册状态，注册上后做 PDP 激活，如果服务器是 URL，则需 DNS 查询，否则不需要，

建立 **sockets** 连接，收发数据，按需要看是否需要主动 **closesockets**。

- (2) 支持多路业务并发最大 7 路，**sockets** 最大支持 24 路，多路业务共享这 24 路 **sockets**，用完为止；基本上很少遇到多业务并发的客户，一般只激活一路即可，建立一路 **sockets** 业务。
- (3) **PDP** 激活后，无需断开，如果主动调 **release** 断开，则下次做数据业务前必须重新激活；
- (4) 如果由于网络原因或长时间无信号掉网收到 **CP** 发来的 **PDP** 断开指示，我们会发 **GAPP_SIG_PDP_DEACTIVE_ABNORMALLY_IND** 通知 **OpenCPU** 应用 **PDP** 异常断开，应用收该 **signal** 后同时判断网络处于注册上后再次发起 **PDP** 连接；
- (5) 平台受限，**PDP** 激活和去激活接口均是异步的；
- (6) 可通过 **fibocom_PDPStatus** 接口查询当前某一路激活状态及 **IP** 信息；
- (7) 收到 **PDP** 激活指示后底层 **net** 线程正在做 **IP/DNS** 适配 **netif** 过程及 **netifbringup** 的动作，上层立即去做 **sockets** 连接 **DNS** 查询等可能失败，由于虚拟网卡可能还未 **bringup**，建议上层稍等 500ms 左右建立 **sockets**。



注意：

PDP 激活务必保证是注册状态的，建立 **sockets** 连接做数据业务，**DNS** 查询等务必保证 **PDP** 是激活状态的。

9.3 OTA/固件升级

- 功能及特性：

支持客户 **APP** 升级和主固件升级，共 4 个 **API** 接口。

- **API** 及涉及 **signal event**：

共 4 个 **API** 接口，详细说明如下：

- 1) **INT32 fibocom_open_ota**(**UINT8 type**,**UINT8 *url**,**UINT8 *filename**,**UINT8 *username**,**UINT8 *password**);

此 **API** 通过 **HTTP** 或者 **FTP** 方式下载并自动升级客户的 **APP**；

所填参数为用于下载的 **HTTP** 或 **FTP** 参数，具体请参考 **API** 手册；

一旦下载完成，会对客户 **APP** 做有效性检查。检查成功，则模块自动重启进行新旧 **APP** 的替换工作。替换完成，模块会正常启动。

- 2) **INT32 fibocom_ota_handle**(**INT8 *data**,**UINT32 len**);

客户 **APP** 升级接口，参数 **data** 为客户 **APP** 的码流，**len** 为 **data** 的长度；

此 **API** 不包含下载过程，只实现升级流程。下载过程需要客户自行完成，比如使用 **TCP**、**UDP** 进行 **APP** 下载。

- 3) **INT32 fibocom_firmware_ota**(**UINT8 type**,**UINT8 *url**,**UINT8 *filename**,**UINT8 *username**,**UINT8 *password**);

此 **API** 通过 **HTTP** 或者 **FTP** 方式下载主固件差分包并自动升级主固件。

下载完成后，会检查差分包的有效性。如果无效，则直接返回-1；如果有效，则重启在 **bootloader** 阶段进行主固件的升级操作。升级完成之后正常启动模块。

升级过程会花费 1 到 2 分钟时间。

4) INT32 fibo_firmware_handle(INT8 *data, UINT32 len);

主固件升级函数，**data** 为主固件差分包数据流，**len** 为 **data** 的长度。

同样，此函数不包含下载流程。下载可有客户自行完成。

● 其他注意事项：

不论下载的是客户 **APP** 还是主固件的差分包，一定要保证其正确性。如果错误，则升级失败，函数都会返回-1。如果成功，都会返回 0。

9.4 休眠/唤醒

● 功能及特性：

这一部分主要是用于请求模块进入休眠和唤醒模块（如果有其他任务在占用资源，模块不一定能睡下去）。

● API 及涉及 signal event:

INT32 fibo_setSleepMode(UINT8 time), 当 **time** 不为 0 的时候，模块将在 **time**（单位秒）之后释放 **uart** 的资源，请求模块进入 **sleep**; **time** 为 0，即 **uart** 抢占资源，锁住模块，是模块唤醒。

● 其他注意事项：

- (1) **Open** 项目由于开机并没有初始化 **uart**，**uart** 的初始化动作将有客户来进行，所以会出现未初始化 **uart** 之前，这条指令无效的情况。
- (2) 按键，**UART** 发数据等会退出休眠。
- (3) 在客户未下发 **CFUN0**（打开飞行模式）情况下，无线是保活的，也就是说，来电来短信，来下行数据业务等能唤醒休眠。

9.5 定时器

● 功能及特性：

用于适当延迟后，执行回调函数。

● API 及涉及 signal event:

主要有 3 个 API，以下详细说明：

1) UINT32 fibo_timer_new(UINT32 ms, void(*fn), (void*arg), void*arg);

创建定时器，第一个参数为时长；第二个为执行的回调函数；第三个为回调函数的参数。

此定时器精确到 1ms；

函数返回为定时器 ID，在释放定时器的时候会使用，需要保存。

此为硬件定时器，非软件定时器。

- 2) `UINT32 fibo_timer_period_new(UINT32ms,void(*fn),(void*arg),void*arg);`

与 `fibo_timer_new` 类似，唯一区别会循环执行回调函数。循环的时长与参数 1 直接相关。

- 3) `INT32 fibo_timer_free(UINT32 timerid);`

释放定时器。入参为创建定时器的返回值。

- 其他注意事项：

`fibo_timer_new` 创建的定时器只要执行回调函数，即可不用再调用 `fibo_timer_free` 进行释放，OS 会自动执行释放。但是循环定时器，必须调用 `fibo_timer_free` 才能释放。

为了避免由于忘记释放循环定时器导致的问题，建议使用 `fibo_timer_new` 进行定时器创建。

9.6 I2C

- 功能及特性：

用于对 I2C 设备进行数据读写。

- API 及涉及 signal event:

主要有如下四个 API

- 1) `INT32 i2c_open(HAL_I2C_BPS_Tbps)`，打开 I2C 设备，`HAL_I2C_BPS_T` 是一个结构体，里面包含元素是两个 `int` 型的数，第一个是 `i2cname`，取值为（1-3），分别为 I2C1，I2C2 和 I2C3，另一个 `int` 数代表速率，取值为 0（100KBPS）和 1（400KBPS）。

- 2) `INT32 i2c_close(void)`，关闭 I2C，返回结果 0 成功，<0 失败。

- 3) `INT32 i2c_send_data(UINT32 slaveAddr,UINT32 memAddr,BOOL is16bit,UINT8 *pData,UINT8 datalen)`

给 I2C 设备发送数据，参数说明如下：

<slaveAddr>I2C 设备 7 位从机地址。

<memAddr>设备的寄存器地址。

<is16bit>表示 `memAddr` 是否为 16bit，true 为是 16bit，false 为 8bit，由客户 I2C 设备决定

<pData>要写入 I2C 设备的数据。

<datalen>要写入 I2C 设备的数据长度。最大长度为 8。

- 4) `INT32 i2c_get_data(UINT32 slaveAddr,UINT32 memAddr,BOOL is16bit,UINT8 *pData,UINT8 datalen)`

读取 I2C 设备数据，参数说明如下：

<slaveAddr>I2C 设备 7 位从机地址。

<memAddr>设备的寄存器地址。

<is16bit>表示 `memAddr` 是否为 16bit，true 为是 16bit，false 为 8bit，由客户 I2C 设备决定

<pData>保存读取的数据。

<datalen>要读取 I2C 设备的数据长度。最大长度为 8。

- 其他注意事项:

因 I2C 是客户控制的设备, 所以, 地址信息及位数由客户设备决定, 所以一定要保证传递的信息正确, 否则会有读取失败的情况。

9.7 SPI

- 功能及特性:

NA

- API 及涉及 signal event:

NA

- 其他注意事项:

主版本和 OpenCPU 版本均不支持通用 SPI, 平台没有通用 SPI 驱动; 但支持专用 SPI, 如 LCD、外挂 FLASH、camera 等均有专用 SPI。

9.8 UART

- 功能及特性:

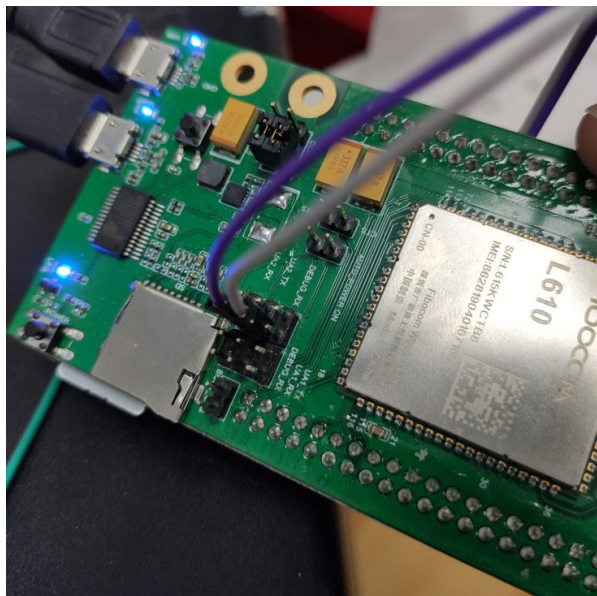
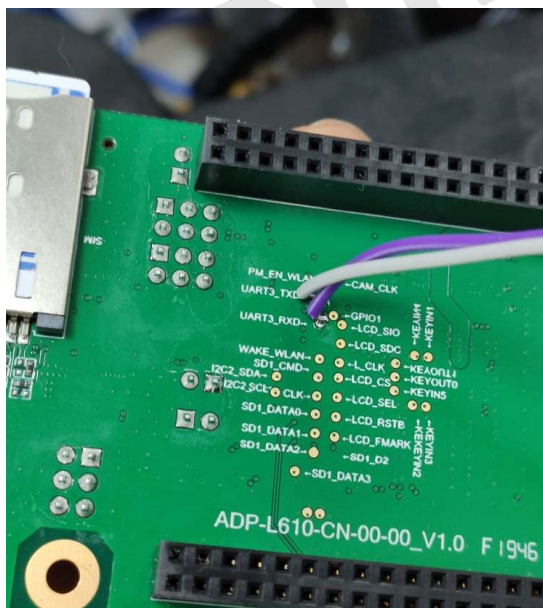
串口使用及跳线规则

- API 及涉及 signal event:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》

- 其他注意事项:

(1)、UART 总共有三个口可以使用, 分别是 0,1 和 2 口, 0 和 1 口可以使用跳帽完成 (详见开发板上 debug 跳线名称标注 0 是 UA1_TX/UA1_RX, 1 是 UA2_TX/UA2_RX), 2 是 UART3 口接线方式如下:



(2)、UART 初始化时，需要指定接收 buffer 的大小和接收的超时时间，只有在超时时间到或者接收 buffer 满了的情况下，才会调用接收回调函数。

```
struct hal_uart_config_s
{
    uint32_t baud;           ///< baudrate, 0 for auto baud
    hal_uart_data_bits_t data_bits; ///< data bits
    hal_uart_stop_bits_t stop_bits; ///< stop bits
    bool cts_enable;         ///< enable cts or not
    bool rts_enable;         ///< enable rts or not
    size_t rx_buf_size;      ///< rx buffer size
    size_t tx_buf_size;      ///< tx buffer size
    uint32_t recv_timeout;   ///< ms
};
```

9.9 USB

- 功能及特性:

NA

- API 及涉及 signal event:

NA

- 其他注意事项:

没有单独开发 USB API 接口供外界设备使用，请基于实际硬件 USB 口及其驱动情况功能使用及调试。

9.10 ADC

- 功能及特性:

这一部分主要讲述 ADC 功能，主要有获取 ADC 外部负载，已经获取芯片 VBat 电压，获取芯片内部温度接口（后续添加，当前版本没有）。

- API 及涉及 signal event:

INT32 fibo_hal_adc_init(void)//初始化 ADC 功能

INT32 fibo_hal_adc_deinit(void)//去初始化 ADC 功能

INT32 fibo_hal_adc_get_data_polling(hal_adc_channel_t channel,UINT32 *data)

//获取 ADC 通路上面的负载值

/*获取 VBat 电压，芯片内部温度接口后续添加*/

- 其他注意事项:

fibo_hal_adc_get_data_polling 这个接口获取 ADC 负载值的时候，一定要注意，当前获取的 ADC 通道上面要有有效负载，范围 0-5V，不可以去量一个悬空脚的值，会有很大误差。

9.11 GPIO

- 功能及特性:

这一部分讲述 GPIO 功能的使用和注意事项

- API 及涉及 signal event:

INT32 fibo_gpio_mode_set(SAPP_IO_ID_T id,UINT8 mode);//设置 GPIO 的工作模式

INT32 fibo_gpio_cfg(SAPP_IO_ID_T id,SAPP_GPIO_CFG_T cfg);

//设置为 GPIO 模式后，设置输入输出方向

INT32 fibo_gpio_set(SAPP_IO_ID_T id,UINT8 level);//设置为 GPIO 的输出模式后，拉高拉低

INT32 fibo_gpio_get(SAPP_IO_ID_T id,UINT8 *level);//设置为 GPIO 模式后，获取电平高低

INT32 fibo_gpio_pull_disable(SAPP_IO_ID_T id);

//释放 GPIO 工作模式到默认工作模式（前提是模式客户手动从默认模式复用到其他模式）

INT32 fibo_gpio_isr_init(SAPP_IO_ID_T id,oc_isr_t *isr_cfg);//设置 GPIO 为中断模式

INT32 fibo_gpio_isr_deinit(SAPP_IO_ID_T id);//将 GPIO 中断模式 disable

- 其他注意事项:

在 fibo_gpio_mode_set 这一接口中，千万注意，要配合我司的 gpio 功能复用表来使用，不是所有 GPIO 管脚的 mode0 来当做 GPIO 的！

9.12 看门狗

- 功能及特性:

此功能包括软件了开/关看门狗的接口。

- API 及涉及 signal event:

fibo_watchdog_enable，开启看门狗，有唯一一个参数设置看门狗的时间，程序会自己喂狗。

fibo_watchdog_disable 关闭看门狗。

- 其他注意事项:

时间设置单位为秒，不可设置为 0。

9.13 MQTT

- 功能及特性:

支持 MQTT

- API 及涉及 signal event:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》

- 其他注意事项:

- (1) 使用请参考 `oc_mqtt_demo.c`
- (2) MQTT 业务请先做 PDP 激活

9.14 OneNET

- 功能及特性:

这一部分主要是封装 OneNET 的 API，比如创建实例，建立链接，接收消息，对实例读写等操作。

- API 及涉及 signal event:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

典型 api:

`fiboneonet_create` 可使用参数创建实例，也可将参数默认为 0, 0, 0, 0 创建默认实例

`fiboneonet_observe_response` 与 `discover/read/write` 等一样 msgid 要使用服务器下发的 msgid，可以回调信息 `sig_res_callback` 中取得。

- 其他注意事项:

- (1) 调用接口时 PDP 必须激活，否则不会有响应。
- (2) 需在中移动 onenet 开发者中心注册设备，对实例进行操作。
- (3) 服务器下发消息后需尽快回应，否则超时后回应则服务器没有响应需重新下发 api 或更新注册

9.15 LCD

- 功能及特性:

针对当前的 L610 平台，仅支持 SPI 连接方式的 LCD 屏，支持三线或四线 SPI 连接。LCD 背光使用 PWM 来控制，可以实现屏幕的亮暗调节。API 实现了 LCD 的初始化，点阵的点亮及帧数据的刷新。本平台支持四个图层叠加，layer0 为摄像头图像显示层，支持 yuv 格式，为最底层；layer1—layer3 为图片显示层，支持 RGB 格式，layer3 为最顶层。帧数据常用的格式有 RGB 和 YUV，我们使用 RGB565 和 YUV422 格式。目前平台支持的 LCD 包括 st7789v1、st7789v2，分辨率为 240*320，每帧数据最大为 240*320*2Byte，SPI 采用 50M 时钟，保证帧数据的快速传输及刷新。

- API 及涉及 signal event:

INT32 `fiboneonet_lcd_init(void)`

该接口主要是初始化 lcd，不能重复执行。

INT32 `fiboneonet_lcd_deinit(void)`

该接口主要是去初始化 lcd，不能重复执行。

INT32 `fiboneonet_lcd_sleep(BOOL mode)`

该接口主要是设置 lcd 休眠唤醒，mode 为 true 时进入休眠，为 false 时退出休眠。

INT32 `fiboneonet_lcd_frame_transfer(const lcd_frame_buffer_t *pstFrame, const t_lcd_display_t *pstWindow)`

该接口是传输帧数据到 lcd 的缓存，用于单独刷新帧数据到 lcd 图层，yuv 格式的数据放在 layer0，rgb 格式的数据放在 layer3，具体入参定义请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

INT32 fibo_lcd_SetOverLay(const lcdFrameBuffer_t *pstFrame)

该接口是设置 lcd 的 layer0—layer2，用于多个图层叠加显示。帧数据采用由底层到顶层的顺序，yuv 格式的数据放在 layer0，RGB 格式的数据依次由 layer1 到 layer2 放置。设置好底层后，再调用 fibo_lcd_FrameTransfer 设置最顶层和显示窗口，这样就可以多个图层叠加显示了。

INT32 fibo_lcd_SetBrushDirection(lcdDirect_t direct_type)

该接口用于设置 lcd 的横竖屏模式，及刷屏方向，具体参数定义请参考 《FIBOCOM L610 系列 OpenCPU API 应用指南》。

INT32 fibo_lcd_SetPixel(UINT16 ulx,UINT16 uly,UINT16 ulcolor)

该接口用于点亮单个像素，显示指定的颜色。

INT32 fibo_lcd_FillRect(const lcdDisplay_t *pstWindow,UINT16 ulBgcolor)

该接口用于用指定的颜色填充一块区域。

INT32 fibo_lcd_DrawLine(UINT16 ulSx,UINT16 ulSy,UINT16 ulEx,UINT16 ulEy,UINT16 ulColor)

该接口用于画一条线，从（ulSx,ulSy）到（ulEx,ulEy），使用指定的颜色。

- 其他注意事项：

注意帧数据及色码的大小端是否与 L610 一致。

9.16 摄像头

- 功能及特性：

针对当前的 L610 平台，有专门的 Camera 控制器，它是通过 SPI 来连接摄像头的传输模块，作为图像数据的输入口；然后使用 I2C 来连接摄像头的配置模块，作为配置数据的输出口。L610 与 Camera 的 pin 脚对应关系如下图所示：

L610-CN PIN#	默认功能	复用功能1	CAM外设
CAM接法			
6	NET_STATUS	CAM_SI1	DATA1
23	SD_DET	CAM_SCK	PCLK
61	STATUS	CAM_RSTL	RST
62	UART1_RING	CAM_PWDN	PWDN
63	UART1_DCD	CAM_REFCLK	MCLK
66	UART1_DTR	CAM_SIO	DATA0
41	I2C1_SCL		SCL
42	I2C1_SDA		SDA

目前支持的摄像头为 GC032a，像素为 30W，分辨率为 640*480，帧率最大支持 19fps。摄像头图像格式采用 YUV422 格式，配合 LCD 屏显示，实现 Preview 和 Capture 功能。

- API 及涉及 signal event:

INT32 fibo_camera_init(void)

该接口用于摄像头的初始化，不能重复执行。

INT32 fibo_camera_deinit(void)

该接口用于摄像头的去初始化，不能重复执行。

INT32 fibo_camera_StartPreview(void)

该接口用于启动摄像头的 preview 模式。

INT32 fibo_camera_StopPreview(void)

该接口用于停止摄像头的 preview 模式。

INT32 fibo_camera_GetPreviewBuf(UINT16 **pPreviewBuf)

该接口用于获取 preview 模式时的帧数据，当获取成功时，pPreviewBuf 为指向帧数据地址的指针。

INT32 fibo_camera_PrevNotify(UINT16 *pPreviewBuf)

该接口用于通知摄像头获取帧数据，在 fibo_camera_GetPreviewBuf 后，如果还需要继续获取帧数据，则需要调用该函数。

INT32 fibo_camera_CaptureImage(UINT16 **pFrameBuf)

该接口用于捕获一个 image。

INT32 fibo_camera_GetSensorInfo(CAM_DEV_T *pstCamDevice)

该接口用于获取摄像头的配置信息，包括摄像头图像长、宽等。

- 其他注意事项：

请参考 oc_lcd_demo.c

9.17 LBS/WIFISCAN

- 功能及特性：

支持通过 LBS 和 WIFISCAN 定位；目前国内只支持高德地图，且内置我司从高德公司买的 key，该 key 支持终端数有限，建议客户自己同支持 LBS 和 WIFI 定位的地图服务商申请自己的 key，通过 fibo_lbs_set_key 接口设置下去。

- API 及涉及 signal event:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

内部封装发 AT 指令的 API 发 AT 命令执行定位；

通过 AT 接收响应的 callback 接收定位经纬度信息。

- 其他注意事项：

- (1) 调此接口定位时 PDP 必须激活；
- (2) 5 表示 lbs 通过谷歌服务器获取定位信息，国内不支持；
- (3) 6 表示 lbs 通过高德地图获取定位信息，国内验证 PASS；
- (4) 7 表示通过 WIFISCAN 通过高德地图获取定位信息；
- (5) WIFISCAN 可参看《FIBOCOM L610 系列 应用指南_WIFISCAN 定位技术》

9.18 CELLINFO

- 功能及特性:

获取服务小区和邻区信息及注册信息;

- API 及涉及 signal event:

`fibocom_getRegInfo`//获取注册状态及基本的消息信息

`fibocom_getCellInfo`//获取较完整的服务小区和邻区信息

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

无 signal event 上报;

- 其他注意事项:

请参考 `pdp_sockets_demo.c` 及 `lbs_demo.c` 里面有相关的参考接口使用示例。

9.19 发送 AT 命令的 API

- 功能及特性:

本节讲述封装可以发送 AT 的 API 及接收 AT 响应的 API。

- API 及涉及 signal event:

`INT32 fibocom_at_send(const UINT8 *cmd,UINT16 len);`

其中 `cmd` 是构造 AT 命令的指针, `len` 是 AT 命令总子串的长度。

接收 `ATResponsecallback`:

- 其他注意事项:

(1) 大部分 AT 是阻塞的, 也就是 AT 命令返回 OK 或 ERROR 后才释放 AT 通道, 否则发送其他 AT 将会阻塞在 AT 通道上, 建议一条 AT 响应后再发下条 AT。

(2) 其次 AT 返回 OK 或 ERROR 在先, 事件响应在后, 建议加延时接收异步事件响应, 待接收到上一条异步事件子串消息后再发送下一条 AT 指令。

9.20 文件系统

- 功能及特性:

这一部分讲述文件系统的操作

- API 及涉及 signal event:

`INT32 fibocom_file_open(const INT8 *pathname,UINT32 opt);`

`INT32 fibocom_file_close(INT32 fd);`

`INT32 fibocom_file_read(INT32 fd,UINT8 *buff,UINT32 size);`

`INT32 fibocom_file_write(INT32 fd,UINT8 *buff,UINT32 size);`

```
INT32 fibo_file_seek(INT32 fd,INT32 offset,UINT8 opt);
INT32 fibo_file_rename(const INT8 *lpOldName,const INT8 *lpNewName);
INT32 fibo_file_rmdir(const char *name);
INT32 fibo_file_mkdir(const char *name);
INT32 fibo_file_exist(const INT8 *pszFileName);
INT32 fibo_file_getSize(const INT8 *pszFileName);
INT32 fibo_file_delete(const INT8 *pathName);
```

- 其他注意事项:

(1) 客户使用一定要参考 open 的 API 手册使用。

9.21 接收异步事件和接收 AT 响应 callback

- 功能及特性:

本节讲述接收处理异步事件上报的 callback; 以及接收 AP 响应的 callback。

为何会用到异步事件上报的 callback, 如 6.2 节所讲, 由于很多 API 是非阻塞的, 函数执行很快结束退出, 而该 API 所涉及的事情交由主版本其他线程处理, 其他线程处理甚至需要等待网络的响应, 网络响应时间处理很难控制, 所以等待网络响应后一级一级线程上报成功或失败的结果, 主版本内部收到事件成功或失败的响应后通过 signal event 上报给 OpenCPU 应用程序。

所以 event 是标记事件成功失败的关键, 也是客户的一个重要参考。

- API 及涉及 signal event:

涉及代码:

其中 **user_callback** 是一个函数结构体, 内部定义了两个结构体变量, 并且分别赋值回调函数 signal 接收回调函数 sig_res_callback 和 AT 响应回调函数 at_res_callback。

//接收异步事件响应的回调函数

```
static void sig_res_callback(GAPP_SIGNAL_ID_T sig, va_list arg)
{
    switch(sig)
    {
        //for customer, you can do anything here

        //fibo_PDPActiveIpaddressresopnseevent
        case EVENT_ID:
        {
            ...
        }
    }
}
```

```

        break;
    default:
        break;
}
}

//接收 ATresponse 的回调函数
static void at_res_callback(UINT8*buf,UINT16len)
{
    //for customer, you can do anything here
    OSI_PRINTF("FIBO<--%s",buf);
}
//定义变量 user_callback
static FIBO_CALLBACK_T user_callback={
    .fibo_signal=sig_res_callback,
    .at_resp=at_res_callback;
}

//客户程序入口函数
void* appimg_enter(void*param)
{
    ....
    Return(void*)&user_callback;
}

```

- 其他注意事项:

EVENTID:

参见代码: fibo_OpenCPU_comm.h 及《FIBOCOM L610 系列 OpenCPU API 应用指南》。

9.22 Debug API

- 功能及特性:

为了方便客户代码加打印信息,方便调试,提供如下接口可以打印 log:

- API 及涉及 signal event:

INT32 fibo_textTrace(INT8*fmt,...)

比如: fibo_textTrace("[%s-%d",__FUNCTION__,__LINE__)打印某函数名称及所在文件行号;

或直接参考 demo 里的 logtrace 接口

```
OSI_PRINTF("[%s-%d",__FUNCTION__,__LINE__);
```

示例:

```
OSI_PRINTF("[%s-%d",Parm1=%d,Parm1=%s",__FUNCTION__,__LINE__,Parm1,Parm2);
```

如%s 打印__FUNCTION__实际的函数名称。

如%d 打印所在文件__LINE__代表的行号。

如%d 打印整形数据 Parm1。

如%s 打印非空指针 Parm2 内存字符串，注意最大支持 255 字符串打印；

上述两个接口使用起来非常方便。

上述 log 打印直接可以通过 coolwatcher 工具抓到，并直接过滤关键字 view 客户代码加的 debuglog 信息，coolwatcher 使用可参见《FIBOCOM L610 系列 LOG 抓取指南_Windows》。

- 其他注意事项:

无 signal event。

9.23 OTHER API

- 功能及特性:

这一部分主要是一些客户常用 API，比如查询信号强度、SIM 卡状态、ICCID 等。

- API 及涉及 signal event:

涉及 API:

```
extern INT32 fibo_cfun_one(); //关闭飞行模式
```

```
extern INT32 fibo_cfun_zero(); //打开飞行模式
```

```
extern INT32 fibo_get_ccid(uint8_t*ccid); //获取 SIM 卡 ICCID
```

```
extern INT32 fibo_get_sim_status(uint8_t *pucSimStatus); //获取 SIM 卡状态
```

```
extern INT32 fibo_get_imsi(UINT8 *imsi); //获取 SIM 卡 IMSI
```

```
extern INT32 fibo_get_csq(INT32 *rssi,INT32 *ber); //获取信号响度，同 AT+CSQ
```

- 其他注意事项:

- (1) 注意获取 SIM 卡状态的接口，如果硬件上 DISABNLESIM 卡热插拔，则查询的 SIM 卡状态非即时状态。
- (2) 请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》

9.24 SSL socket API

- 功能及特性:

这一部分新增建立 SSL socket API;

- API 及涉及 signal event:

涉及 API:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》 SSL 章节

- 其他注意事项:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》

9.25 HTTP API

- 功能及特性:

这一部分新增 HTTP 相关 API;

- API 及涉及 signal event:

涉及 API:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》 HTTP 章节。

- 其他注意事项:

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

9.26 客户定制 API

- 功能及特性:

根据客户的硬件设计, 定制的 API, 仅适用于客户硬件;

- API 及涉及 signal event:

涉及 API:

```
extern UINT8 fibo_read_key(void); //获取按键值
extern UINT8 fibo_get_pwrkeystatus(void); //获取开关键状态
extern UINT8 fibo_get_Usbisinsert(void); //获取 USB 插入状态
extern void fibo_SetLcdBright(UINT8 level); //设置 LCD 背光亮暗
extern bool fibo_Rtc2Second(uint8_t * DateTimeBuf, uint32_t * OutSec); //日期转秒
extern bool fibo_Second2Rtc(uint32_t SecondTime, uint8_t * DateTimeBuf); //秒转日期
extern INT32 fibo_GetBatVolResult(void); //获取电池电压
extern UINT8 fibo_GetBatRemainlvl(void); //获取电池剩余电量
extern UINT8 fibo_GetBatChargPinStatus(void); //获取电池充电状态
```

```
extern bool  fibo_SetBatChargSel(UINT8 level); //设置电池充电选择
```

```
extern INT32 fibo_GetHardVer(void); //获取硬件版本
```

```
extern bool  fibo_pwrkey_init(void); //初始化开关按键。
```

- 其他注意事项：

请参考《FIBOCOM L610 系列 OpenCPU API 应用指南》。

FIBOCOM
Confidential

10 其他需求功能

- TTS

主版本支持。

OpenCPU 已支持 API。

- HTTPS

主版本支持 HTTPS。

OpenCPU 已支持 API。

- TLS

主版本支持；

OpenCPU 正在封装 API。

- FTP

主版本支持；

OpenCPU 没有封装 API，功能使用请调发 AT 的 API 发 AT 命令实现。

请参考《FIBOCOM L610 Series AT Commands_FTP》。

- 天翼云

主版本支持；

OpenCPU 没有封装 API，功能使用请调发 AT 的 API 发 AT 命令实现。

请参考《FIBOCOM L610 Series AT Commands_eCloud》。



















- 阿里云

主版本已支持。

OpenCPU 已支持。

11 参考 demo

目前提供给客户的 demo 有：

-  at_demo.c
-  demo.c
-  hello_world.c
-  lbs_demo.c
-  oc_camera_demo.c
-  oc_file_demo.c
-  oc_gpio_demo.c
-  oc_lcd_demo.c
-  oc_mqtt_demo.c
-  oc_uart_demo.c
-  onenet_demo.c
-  osi_demo.c
-  ota_demo.c
-  pdp_sockets_demo.c
-  pdp_sockets_select_demo.c
-  rtc_demo.c
-  select_demo.c
-  sig_demo.c

如果需要验证某个功能，只需将相应的.c 文件名改为 `hello_world.c`，放在 `CMakeLists.txt` 所在路径，编译生成 PAC 直接烧录，开机运行调试即可；可参考上述代码开发。

示例 1: at_demo.c 用来测试调 AT 的 API 发 at 命令并通过接收 AT response 的 callback 接收 AT 响应;

```
#define OSI_LOG_TAG OSI_MAKE_LOG_TAG('F', 'I', 'B', 'O')

#include "osi_api.h"
#include "osi_log.h"
#include "osi_pipe.h"
#include "at_engine.h"
#include <string.h>
#include "fibo_opencpu.h"

static void at_res_callback(UINT8 *buf, UINT16 len)
{
    OSI_PRINTF1("FIBO <--%s", buf);
}

static void prvVirtAtEntry(void *param)
{
    const uint8_t *cmd = (const uint8_t *)"ATI\r\n";
    fibo_at_send(cmd, strlen((const char *)cmd));
    fibo_thread_delete();
}

static FIBO_CALLBACK_T user_callback = {
    .at_resp = at_res_callback};

void *appimg_enter(void *param)
{
    OSI_LOGI(0, "application image enter, param 0x%x", param);
    //init
    fibo_thread_create(prvVirtAtEntry, "at", 1024, NULL, OSI_PRIORITY_NORMAL);
    return (void *)&user_callback;
}

void appimg_exit(void)
{
    OSI_LOGI(0, "application image exit");
}
```

可以看到，发 AT 的 API 跑在一个线程内，构造 AT 指令，调 fibo_at_send 接口将 AT 发至 modem；通过全局结构体函数变量 user_callback 中的一个.at_resp 回调函数接收 AT 的响应（见 demo 中 at_res_callback 接口）。

示例 2: osi_demo.c, 请详见代码，此处不再粘贴出来；demo 代码主要测试创建线程，信号量，消息队列，消息发送和接收以及锁等功能；

Lbs_demo.c 主要是检测注册状态，注册上后激活 PDP，并执行 lbs 或 wifiscan 定位，并通过 ATcallback 接收经纬度信息。

Pdp_sockets_demo.c:主要测 PDP 激活和创建 sockets 收发数据,DNS 查询等功能

Pdp_sockets_select_demo.c:

select_demo.c:

主要描述 select 接口的使用。

Ota_demo.c 主要讲述 APP 和主固件 OTA 升级参考应用。

其他接口请参见代码，负责各个模块的测试示例：



注意：

提供的客户 demo 代码仅仅是作为 APItest 参考使用，不做上层应用程序功能参考，功能还需客户自己开发。

FIBOCOM
Confidential

12 其他注意事项

12.1 prvInvokeGlobalCtors()接口使用

注意参考 demo 里，在应用程序 void*appimg_enter(void*param)入口函数里请调用 prvInvokeGlobalCtors()此接口。

目的是用来处理 C++全局构造，代码可以仿照 demo 添加进客户的代码里。

FIBOCOM
Confidential

13 Q&A

13.1 如何烧录

请参考《FIBOCOM L610 系列 版本下载指南_Windows》。

FIBOCOM
Confidential