新岸线
NUFRONT

NL6621 SDK 用户手册

版本：V1.7

北京新岸线移动多媒体技术有限公司

2015 年 5 月

地址：北京市海淀区清华科技园科技大厦 A 座 16-17 层/D 座 16-17 层　　邮编：100084
Address: 16th -17th Floor A and D Building, SP Tower, Tsinghua Science Park Building 8, No.1 Zhongguancun East Road, Haidian District Beijing 100084, PRC
TEL：（8610）82150688　　FAX：（8610）82150699　　Web：www.nufront.com

目录

图目录

表目录

1　引言

1.1 概述

本文主要描述 NL6621 软件开发包（SDK）的功能和使用方法，该 SDK 集成了 NL6621 硬件驱动（BSP）、实时操作系统、Tcp/ip 协议栈、WiFi 协议栈以及其它公共模块，能够满足大部分应用软件的需求。建议与 NL6621 评估板配合使用，以提高开发效率。

1.2 芯片架构简介

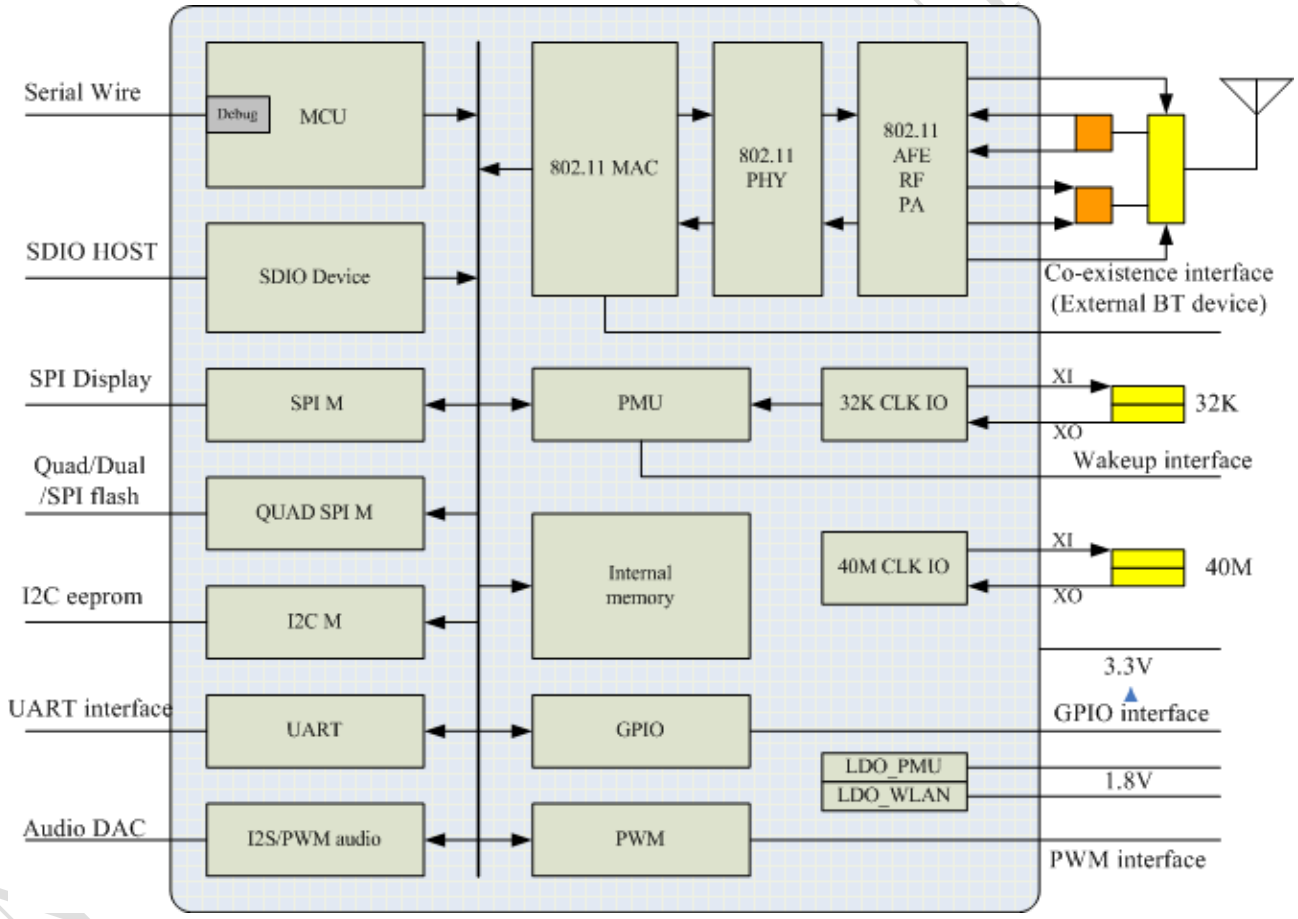


图 1 NL6621 芯片架构

其中，MCU 采用的最高主频为 160MHz

1.3 SDK 基本特征

NL6621 SDK 所支持的功能特征如下：

- 提供 WiFi 协议栈支持
  - ✧ 支持 Sta、Adhoc 以及 SoftAp 三种工作模式
  - ✧ 支持 OPEN、SHARE、WPAPSK、WPA2PSK 安全认证方式
  - ✧ 支持 WEP64/128、TKIP、CCMP 加密通信方式
  - ✧ 支持 802.11 节电管理
  - ✧ 支持隐藏 SSID（SoftAp 模式）
  - ✧ 最多允许 8 个站点接入（SoftAp 模式）
  - ✧ 支持 WPS（Sta 模式）
- 提供 Tcp/Ip 协议栈支持
  - ✧ 支持 IPV4、ICMP、ARP、UDP、TCP
  - ✧ 支持 DHCP Client/Server
  - ✧ 支持 DNS Client/Server
- 提供实时操作系统支持
  - ✧ 支持多任务管理
  - ✧ 支持系统时钟、定时器管理
  - ✧ 支持信号量、互斥锁等同步机制
  - ✧ 支持邮箱、消息队列等异步通信机制
  - ✧ 支持动态内存管理
- 提供丰富的 BSP 接口支持
- 提供应用程序范例

2 SDK 使用说明

2.1 软件架构

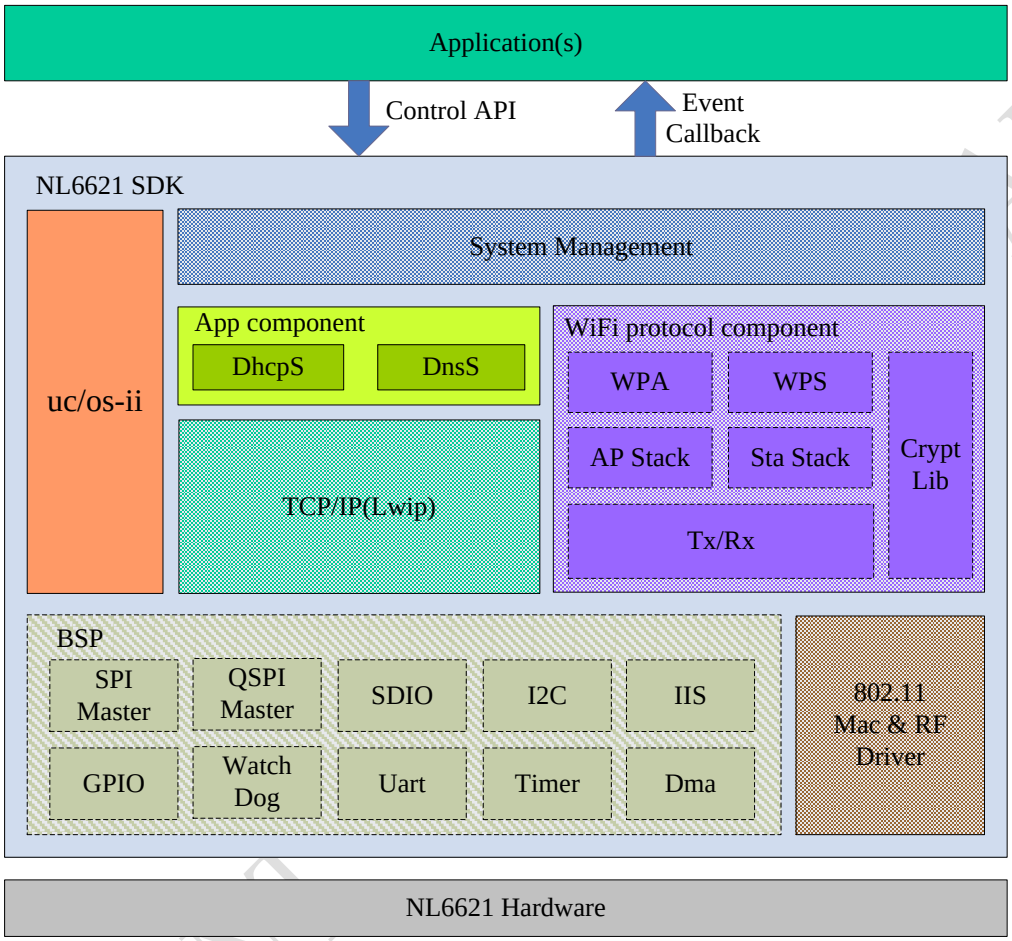


图 2 NL6621 SDK 软件架构

2.2 开发环境

本 SDK 采用的开发环境为 Arm Keil uVersion V4.10

工具链版本：RealView MDK-ARM V4.12

注：不同版本的编译工具生成的代码地址空间可能不同，这将影响对 ROM 代码的调用。因此，不建议使用其它版本的编译工具。

2.3 目录结构

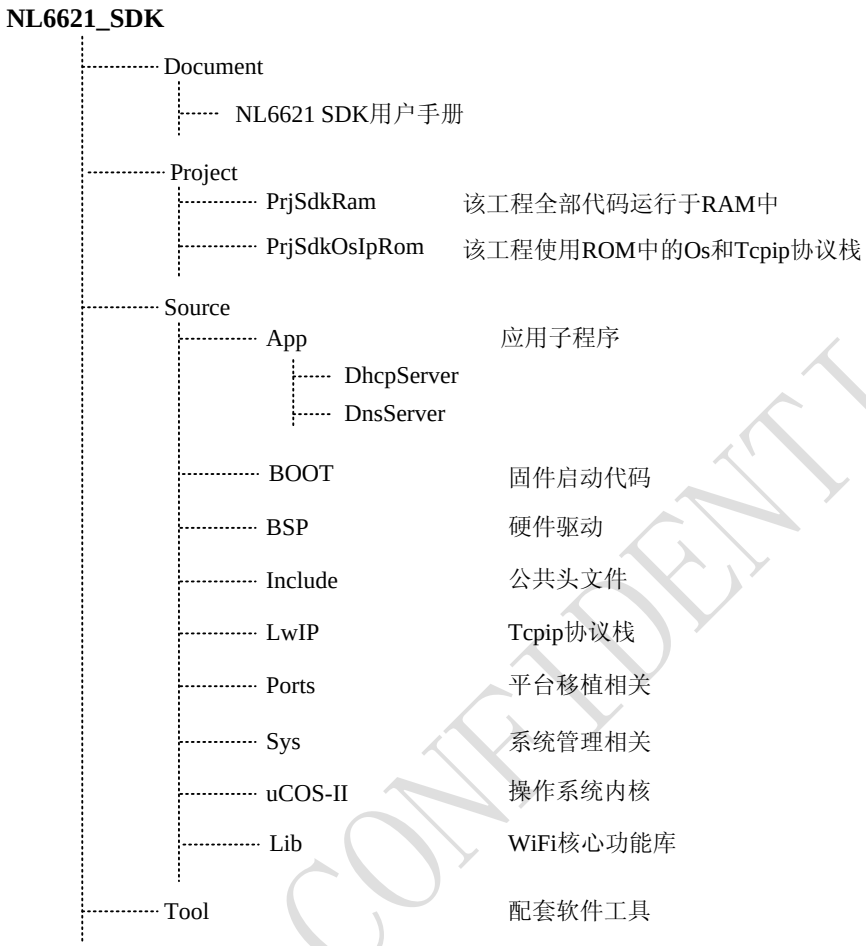


图 3 NL6621 SDK 目录结构

2.4 编译连接

2.4.1 配置文件

Source\Include\ app_cfg.h 文件用来对 SDK 进行功能配置，目前可以配置的功能如下：

表 1 功能配置选项列表

功能	配置选项	说 明
Sta	CONFIG_STA_SUPPORT	Station 协议栈
Ap	CONFIG_AP_SUPPORT	SoftAp 协议栈
StaWps	WSC_STA_SUPPORT	Station 模式下的 Wps 协议
Dhcp Server	DHCP_SERVER_SUPPORT	只在 AP 模式下生效，用于向接入的站点提供

		DHCP 服务
Dns Server	DNS_SERVER_SUPPORT	只在 AP 模式下生效，用于对外提供域名访问方式（只能设置一个域名本 AP）
堆内存大小	OS_DMEM_POOL_SIZE	堆内存大小，以字节为单位，默认为 10Kbytes
测试范例程序	TEST_APP_SUPPORT	本 SDK 提供的测试程序，能够进行简单的 Tcp、Udp 通信测试，详见本文第 5 章
校准发送增益	USE_NV_INFO	校准数据保存在 Flash 中的地址由 Bsp\Flash.h 中的宏 NVM_AUTO_TEST_OFFSET 进行配置，必须与自动化测试软件配置的地址一致。
校验认证码	CHECK_AUTH_CODE	校验由自动化测试软件生成的认证码，如果认证码无效则不允许固件运行。
802.11n 功能	DOT11_N_SUPPORT	打开该宏并替换 lib 文件为 wcore_11n.lib 即可支持 802.11n 功能，但会增加对内存的消耗。

此外，app_cfg.h 中还定义了任务优先级以及栈大小，不允许修改（测试范例所在任务除外），但可以为新增加的任务定义优先级和栈内存，注意不要与现有任务冲突。

2.4.2 库文件

本 SDK 以静态库的方式提供 WiFi 核心功能，库文件路径为 Source\Lib\wcore.lib。为了节省 RAM 资源，该 lib 文件对 WiFi 功能进行了裁剪，默认情况下只包含 Sta 和 Ap 基础协议功能。

2.4.3 编译优化

Keil 中的编译优化选项，会影响编译连接后的整个地址空间使用情况，在需要使用 Rom 固件的条件下，一般情况下不允许修改任何优化选项，目前 SDK 默认采用的优化级别为-O3。

当采用 PrjSdkRam 工程进行开发时，由于该工程不使用 Rom 固件，用户可以修改编译优化选项，该工程默认开启 Cross-Module Optimization 选项，能够删除没有被使用的函数和变量，以达到减少固件镜像文件大小的目的。

注：对部分低版本的开发工具，Cross-Module Optimization 需要手动编译两次，且中间不能做 Clean 操作，高版本工具能够自动完成该过程。

当采用 PrjSdkOsIpRom 工程进行开发时，默认开启了 feedback 编译和连接选项，对工程连续编译两次以上就能够去除无用的符号（函数、变量等），从而达到减少固件大小的目的。

注：PrjSdkOsIpRom 工程 scatter 文件中的加载域被放大到了 0x0004FF00（实际大小为 0x0002FF00），编译优化后生成的 bin 文件大小依然不允许超过 0x0002FF00。

2.4.4 输出文件

编译成功后会生成多个二进制 bin 文件，其中 nft_sd_uapsta.bin 为 ROM 中的 bootloader 程序加载的镜像文件。

2.5 RAM 资源说明

NL6621 的 RAM 情况资源如下表所示。

表 2 NL66221 RAM 资源列表

RAM 名称	大小	用途
CODE SRAM	192K	存放代码或数据
SRAM1	96K	存放数据，只供 CPU 使用
SRAM2	96K	存放数据，供 CPU 与 MAC 共同使用，目前用来存放数据发送队列及其他数据
SRAM3	64K	存放数据，供 CPU 与 MAC 共同使用，目前用来存放数据接收队列及其他数据

注：RAM 地址空间与固件启动方式有关，详细情况参见本文 3.1、3.2 节。

2.6 基本数据类型

本 SDK 对基本数据类型进行了重定义，建议采用本 SDK 定义的数据类型开发程序，以保证程序兼容性和可移植性。

数据类型定义位于文件 Source\Include\ types_def.h 中。

2.7 C 标准库接口

开发工具 Keil 提供了 C 标准库功能，使用前需要添加相关头文件到代码。

另外，本 SDK 自定义了部分标准 C 接口，例如 memcpy、memset 等，以减少标准 C 库的 RAM 开销。

自定义 C 接口位于 source\ uCOS-II\ os_core.c 以及 source\sys\util.c 中，建议优先使用自定义接口。

2.8 堆内存管理

本 SDK 在 uCosII 基础上增加了堆内存管理功能，具体实现位于 Source\Ports\Cortex-M3\os_dmem.c

用户可以通过修改 app_cfg.h 中的 OS_DMEM_POOL_SIZE 来定义堆内存大小。

堆内存分配及释放接口形式如下所示：

```
void *OSMMalloc(INT32U size)
void    OSMFree(void *ptr)
```

2.9 程序调试

2.9.1 JTAG 调试

使用 PrjSdkRam 工程利用 Keil 集成调试工具进行 JTAG 调试，建议仿真器采用 ARM-JLINK。JLINK 设置如下图所示

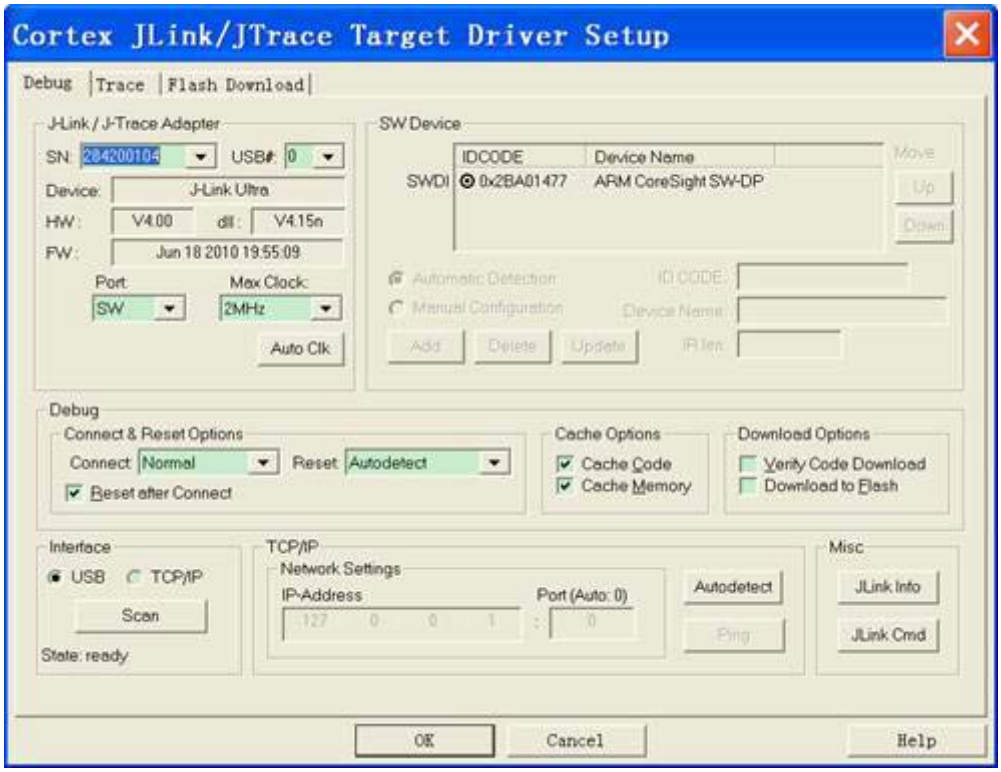


图 4 JLINK Debug 设置

注：在使用 JTAG 前，请注意参考评估板手册中的跳线设置。

2.9.2 UART 调试

使用 PrjSdkOsIpRom 工程编译生成固件镜像（nft_sd_uapsta.bin），并利用 Tool\bootTool.exe 工具通过 UART 接口下载固件进行调试。

bootTool.exe 程序界面如下图所示，该工具支持 UART 加载固件（Uart Boot）和 UART 烧写固件（Uart Burn）功能。

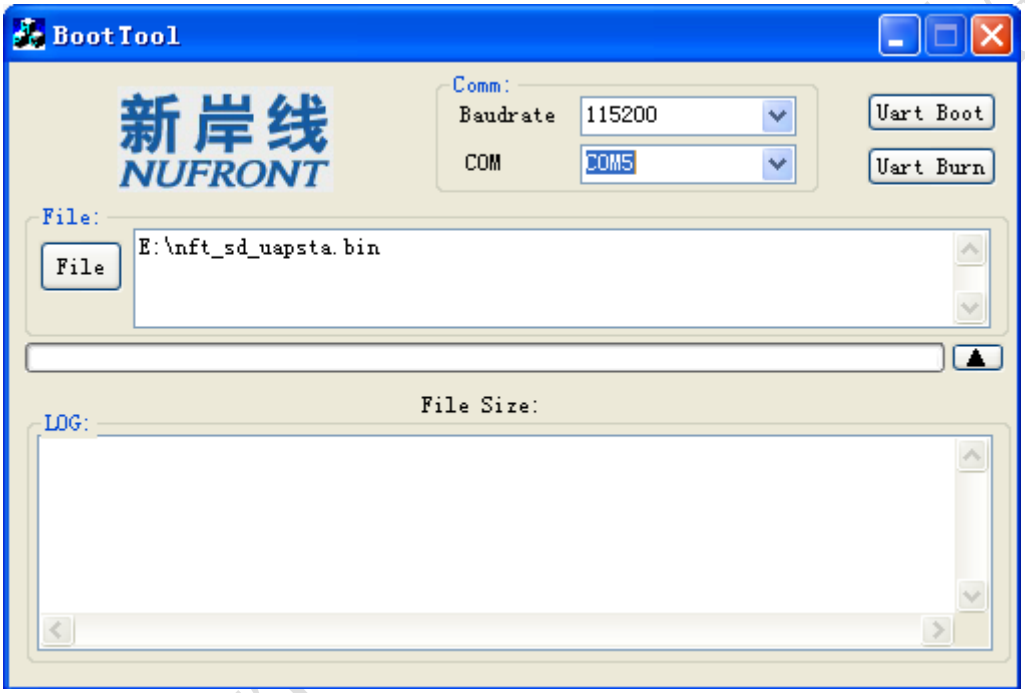


图 5 UART 固件下载工具

1) Uart Boot 操作步骤

- 下载固件之前，请确保评估板重新上电并以 UART 加载固件方式启动（参见 3.1 节）。
- 点击 File 选择需要下载的固件镜像文件。
- 连接评估板 UART 接口与 PC COM 口后，选择对应的 COM 口编号，点击 Uart Boot 则开始固件下载过程。
- 固件下载完成后，会自动跳转到固件镜像入口处开始运行固件程序，此时可以开启串口终端观察程序打印信息。

2) Uart Burn 操作步骤

- 烧写固件之前，先按照 Uart Boot 步骤将 Tool/ burnFlash.bin 文件下载到 RAM 中运行。
- 开启串口终端观察打印信息，当打印“**NULK READY!**”后，则表示成功进入到烧写模式。
- 点击 File 选择需要烧写的固件镜像文件，然后点击 Uart Burn 开始烧写 SPI Flash。

- 烧写完成后，切换到 SPI Flash 加载固件启动方式，然后重新上电即可运行已烧写的固件。

注：当采用 PrjSdkRam 工程生成用于 UART 加载或 SPI Flash 加载的固件镜像文件时，需要将连接器选项中的分散加载文件修改为 scatter_loader.scat，如下图所示。

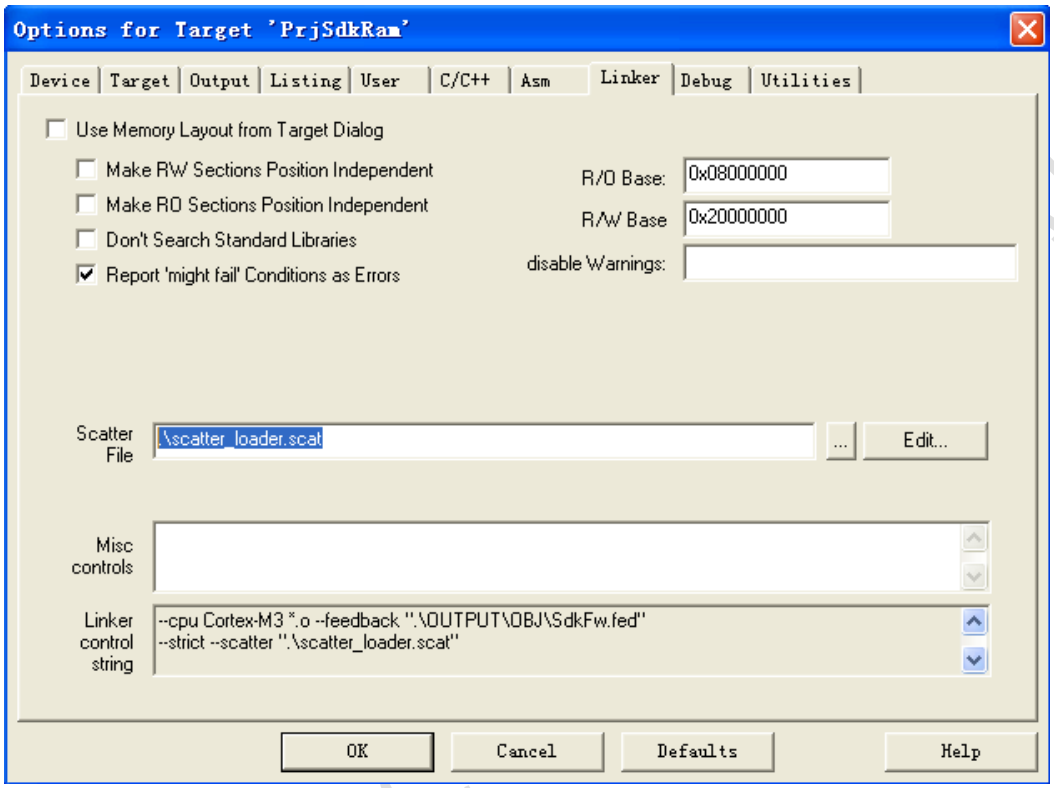


图 6 PrjSdkRam 工程分散加载文件设置

2.9.3 打印调试信息

本 SDK 提供了调试打印接口，位于 source\include\util.h 中，接口形式如下：

```
DBGPRINT(level, fmt, ...)
```

其中 level 为打印级别，定义如下：

```
#define DEBUG_OFF       0
#define DEBUG_ERROR     1
#define DEBUG_WARN      2
#define DEBUG_TRACE     3
#define DEBUG_INFO      4
```

通过定义 DEBUG_ON 和 DEBUG_LEVEL 来决定系统是否打印以及打印级别。

3 应用开发指南

3.1 地址空间

NL6621 芯片支持三种地址空间映射方式（remap= 2’b00, 2’b01, 2’b10），分别应用于各种固件加载方式。地址空间映射情况如下表所示。

表 3 地址空间映射列表

Block	Base address			Actual Size	Comments
	remap = 2’b00	remap = 2’b01	remap = 2’b10		
ROM	0x0000_0000	0x0007_0000	0x0100_0000	64KB	
CODE SRAM	0x0001_0000	0x0000_0000	0x0101_0000	192KB	Code Sram
SRAM0	0x0008_0000		0x0108_0000	96KB	Data Buf SRAM
QSPI	0x0100_0000		0x0000_0000	16MB	External Qspi Flash
SRAM1	0x2000_0000			96KB	Buf SRAM Shared with Mac
SRAM2	0x2004_0000			64KB	Buf SRAM Shared with Mac

3.2 固件启动方式

目前固件启动方式可以有多种，NL6621 上电时由芯片引脚(qspi_hold, qspi_wp, qspi_so)状态来决定，具体定义如下：

表 4 固件启动方式列表

qspi_hold, qspi_wp, qspi_so	Comments	地址空间 remap
`B000	sdio/spi 加载固件	2’b00
`B001	i2c_eeprom 加载固件	2’b00

`B010	spi_flash 加载固件	2’b00
`B011	UART 加载固件	2’b00
`B100	JTAG 加载固件	2’b01
`B101	QSPI Flash 直接运行固件	2’b10

注：

- 采用 sdio/spi、i2c_eeprom、spi_flash 以及 UART 方式加载固件时，均由 ROM 中的 bootloader 完成固件加载过程。固件镜像会被加载到地址为 0x0001_0100 的 CODE SRAM 区域（0x0001_0000~0x0001_0100 被 bootloader 使用）。
- 受 CODE SRAM 空间大小的限制，ROM 中的 bootloader 能够加载的固件镜像大小不能超过 0x2FF00 字节 (0x0040_0000 - 0x0001_0100 = 0x2FF00)。
- 固件镜像必须存放于外部存储介质（Flash、E2prom）的 0 地址。

3.3 ROM 固件说明

3.3.1 ROM 固件地址空间

ROM 固件地址空间划分如下（remap = 2’b00）：

表 5 ROM 地址空间划分

地址区间	存放内容	说明
0x0000_0000 ~ 0x0000_1FFF	Bootloader	用于 sdio/spi、i2c_eeprom、spi_flash 以及 UART 启动方式下加载固件镜像到 CODE SRAM 区
0x0000_2000 ~ 0x0000_FFFF	SDK ROM	为了减少对 RAM 资源的占用，NL6621 针对大部分的应用需求，定制了一版 uCos 以及 Lwip 固件代码，固化为 SDK ROM。

3.3.2 SDK ROM 注意事项

1) PrjSdkOsIpRom 工程的编译优化选项不可修改

由于 SDK ROM 中的代码镜像是由 PrjSdkOsIpRom 工程编译后得到的，因此 PrjSdkOsIpRom 工程的编译优化选项原则上是不可以修改的，否则容易导致 ROM 相关代码的符号地址发生改变，从而使程序运行

失败。

2) uCos 以及 Lwip 的配置选项不可修改

ROM 中固化的 uCos 以及 Lwip 的编译配置选项也已经固定，若固化的配置选项不能满足应用需求，建议放弃使用 SDK ROM，采用 PrjSdkRam 工程进行开发。

uCos 的配置选项位于 Source\uCOS-II\os_cfg.h 中；

Lwip 的配置选项位于 Source\LwIP\port\include\lwipopts.h。

3) SDK ROM 的接口地址区不可修改

SDK ROM 需要调用的接口函数地址以及占用的 RAM 地址空间也已经固定如下：

0x00080000-0x000807FF FW ROM hook function table, 2K

0x00080800-0x00081FFF FW ROM_CODE RW/ZI data

详细的地址空间定义可参考各个工程内的 scatter 文件。

3.4 Flash 空间说明

本 SDK 默认 Flash 地址区间规划如下:

表 6 Flash 地址区间规划

地址区间	存放内容	说明
0x0000_0000 ~ 0x0003_0000	应用固件镜像	该区域存放的固件镜像由 Rom 中的 Bootloader 加载到 RAM 中去执行。
0x0003_2000 ~ 0x0003_3000	射频校准数据	射频校准数据默认保存到该区域，但需要与产测软件的配置保持一致。
其它	用户自定义数据	

3.5 程序入口

汇编入口：CortexM3_startup.s，不建议对该文件进行修改。

C 程序入口：main.c，用户可以对 main 接口进行修改。

main 函数的结构十分简单，用户可在 SystemInit()与 OSStart()之间创建应用程序任务。

例如：

```
INT32 main(VOID)
{
    /* system Init */
    SystemInit();

   /* create application tasks here */
#ifdef TEST_APP_SUPPORT
#ifdef LWIP_SUPPORT
    OSTaskCreate(TestAppMain, NULL, &sAppStartTaskStack[NST_APP_START_TASK_STK_SIZE -1],
                 NST_APP_TASK_START_PRIO);
#endif // LWIP_SUPPORT
#endif //TEST_APP_SUPPORT

    /* Start Os */
    OSStart();

    return 1;
}
```

注：app_main.c（TestAppMain 所在文件）为本 SDK 提供的测试代码，仅供参考，实际应用中可将其删除。

3.6 WiFi 启动过程

应用软件任务可以在系统任务初始化完成后调用 API 接口配置参数和启动 WiFi 功能。

通过 API 接口启动 WiFi 功能的步骤如下：

- 设置系统事件回调函数（InfSysEvtCBSet）
- 配置 WiFi 参数（InfSsidSet、InfAuthModeSet…）
- 配置 Ip 参数（InfIpSet、InfDhcpSrvSet、InfDnsSrvSet）
- 启动 WiFi（InfWiFiStart）

另外，SDK 还提供一个加载默认配置参数的接口 InfLoadDefaultParam，可以通过修改代码的方式自定义默认配置参数，具体参数含义请参考数据结构 CFG_PARAM 定义。

注：应用软件任务需要在系统任务初始化完成后，再进行 WiFi 启动过程，建议适当等待一些时间（1 秒左右）

3.7 网络通信流程

基于本 SDK 的网络通信流程比较简单，下图以应用程序设置启动配置参数为例，指示了从启动 WiFi

到应用软件正常通信的过程。

应用程序需要调用的 WiFi API 接口，请参考本文 5.1 节。

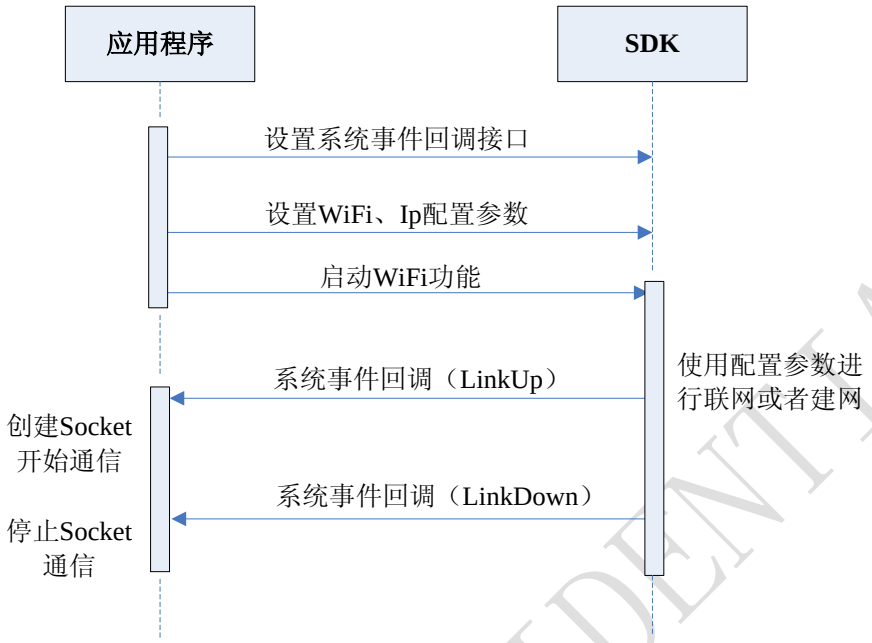


图 7 应用程序通信流程

3.8 系统事件

本 SDK 定义了一些重要的系统事件，通过 InfSysEvtCBSet 设置的应用软件回调函数接口进行通知。目前已定义的系统事件如下表所示。

表 7 系统事件列表

事件名	说 明
SYS_EVT_LINK_DOWN	WiFi 联网失败
SYS_EVT_LINK_UP	WiFi 联网或建网成功
SYS_EVT_JOIN_FAIL	Sta 尝试联网次数超过 ConTryTimes 的限制（由接口 InfConTryTimesSet 配置 ConTryTimes 大小，断网自动重连次数不受 ConTryTimes 限制）
SYS_EVT_DHCP_FAIL	Dhcp Client 失败次数超过 DhcpTryTimes 的限制（由接口 InfIpSet 配置 DhcpTryTimes 大小）
SYS_EVT_SCAN_DONE	扫描完成（只对由应用软件调用 InfScanStart 发起的扫描操作有效）

SYS_EVT_DIRECT_CFG_DONE	Direct Config 过程完成

注：InfSysEvtCBSet 设置的回调函数接口由 SysMgmtMain 任务调用，为避免栈溢出，建议不要在回调函数中实现复杂功能，可以通过发送消息的方式通知应用程序任务去实现.

3.9 I2S 音频接口应用

用户如需在实际应用场景中使用 I2S 接口，可参考 I2S 测试示例代码，调用 I2S 接口的初始化等相关函数，并针对实际场景对相关配置选项进行配置。

1、 开启 I2S 接口相关配置选项。

2、 调用 BSP_I2SInit 函数完成 I2S 初始化（示例代码中在 BSP_Init 函数中调用）;

3、 如需采用 DMA，则应调用 BSP_DmaInit 函数完成 DMA 初始化（示例代码中在 BSP_Init 函数中调用）;

4、 在用户应用线程中调用 BSP_I2SStart 函数使能 I2S 接口，如采用 DMA 则应在该函数中调用 BSP_DmaStart 函数时提供音频数据源地址;

5、 采用 DMA 方式时，因 DMA 搬移以 32bit 为单位，要求音频数据源在 I2S 音频比特位宽的基础上补零以满足 32bit 要求。

   如：I2S 位宽为 16bit，则在将该数据源作为 DMA 源地址时，有效数据为每次搬移的低 16bit，故需要取出每 16bit 音频数据并对其高 16bit 补 0（示例代码 BSP_DmaIntISR 函数中已有示范）。

6、 在一些应用场景，如歌曲播放间隙，可调用 BSP_I2SHalt 暂停 I2S/PWM 接口输出，以使音频接口处于静默状态。如需继续使用 I2S/PWM 接口输出音频数据，则需再次调用 BSP_I2SStart。

7、 对于音源数据不足的情况，可以通过接口 BSP_I2SMute 进入静音状态防止噪音输出，待音源数据恢复后可以通过接口 BSP_I2SSound 取消静音。

3.10 SDIO SPI 传输模式应用

NL6621 芯片的 SDIO Slave 接口支持 1bit 、4bit SD 以及 SPI 传输模式，其中 SPI 模式能够有效的替代 SPI Slave 接口实现与外围芯片 SPI Master 接口的通信，其应用十分广泛。

本 SDK 提供了 SDIO Slave 接口的驱动参考代码，实现了基本的数据读写功能（基于 CMD53），能够演示 1Kbyte 的数据读写操作（需要开启 SDIO_TEST 编译选项）。

针对 SDIO 的 SPI 传输模式，有下列注意事项：

1） 按照 SDIO 协议将 SPI Master 引脚与 SDIO Slave 对应的引脚相连，SDIO 协议中各传输模式下的信号对应关系如下图所示：

Pin	SD 4-bit mode		SD 1-bit mode		SPI mode	
1	CD/DAT[3]	Data line 3	N/C	Not Used	CS	Card Select
2	CMD	Command line	CMD	Command line	DI	Data input
3	VSS1	Ground	VSS1	Ground	VSS1	Ground
4	VDD	Supply voltage	VDD	Supply voltage	VDD	Supply voltage
5	CLK	Clock	CLK	Clock	SCLK	Clock
6	VSS2	Ground	VSS2	Ground	VSS2	Ground
7	DAT[0]	Data line 0	DATA	Data line	DO	Data output
8	DAT[1]	Data line 1 or Interrupt (optional)	IRQ	Interrupt	IRQ	Interrupt
9	DAT[2]	Data line 2 or Read Wait (optional)	RW	Read Wait (optional)	N/C	Not Used
10	NC[1]	For Future Use	NC	For Future Use	NC	For Future Use
11	NC[1]	For Future Use	NC	For Future Use	NC	For Future Use

图 8 SDIO 各传输模式信号对应关系

由上图给出 SDIO 信号对应关系可以得到 SPI Master 与 SDIO Slave 的引脚连接关系如下：

SPI Master　　SDIO Slave

CS　　----------　　DAT3

DO　　----------　　CMD

DI　　----------　　DAT0

SCLK　　----------　　CLK

2） SPI Master 端建议采用 **GPIO 模拟 CS 信号**，有些 SPI Master 的 CS 信号在一个 SDIO 的命令周期中是不连续的，不符合 SDIO 协议。

3） SPI Master 应采用 motorola 格式，CPHA = 1, CPOL = 1。

4） NL6621 的 SDIO 接口所有信号都要上拉，包括 DAT0、DAT1、DAT2、DAT3、CMD、CLK，其中 **DAT1、DAT2 没有被使用也必须上拉**，否则会影响功能。

5） 对 NL6621 的 SDIO Slave 进行枚举之前，需要 SPI Master 端先发送 1 个 byte 周期的时钟信号（此期间 CS 为无效），用于 NL6621 同步芯片内的时钟域。

为了便于用户测试 SDIO 的 SPI 传输模式功能，本 SDK 分别提供了 SPI Master 端的测试代码以及 SDIO Slave 端的测试代码，可以使用两块 6621 开发板进行对接测试，测试步骤如下：

- Master 端开启宏 SPI_SDIO_CMD_TEST 后编译固件程序
- Slave 端开启宏 SDIO_TEST 后编译固件程序

- 连接 SPI Master 与 SDIO Slave 引脚，注意 CS 信号采用 SPI Master 端的 GPIO0 来模拟
- 先启动 SDIO Slave 端的固件程序，然后启动 SPI Master 端的固件程序
- 观察 SPI Master 端的固件程序打印信息

3.11 休眠功能应用

NL6621 支持两种休眠模式：深睡眠 模式和浅睡眠模式。本节将分别介绍二者的工作原理和使用方法。

3.11.1 深睡眠模式

深睡眠模式是一种待机模式，能够最大程度的降低 NL6621 的芯片功耗。在该模式下，CPU 和 RAM 都会掉电，芯片被唤醒后会自动复位 CPU，重新加载固件到 RAM 中去运行。

应用软件通过调用 InfDeepSleepSet 接口来进入深睡眠模式。深睡眠的唤醒方式有以下两种，可以通过 InfDeepSleepSet 接口参数进行设置：

- 通过 Gpio0 唤醒

  Gpio0 需要配置为输入态，当 Gpio0 上电平跳变后的持续时间超过 1 个 32K 时钟周期时即可触发芯片唤醒。

- 通过定时唤醒

注：由于深睡眠模式下，芯片唤醒后需要重新加载固件到 RAM 中去运行，联网状态的恢复时间会比较长，因此该模式不适合需要保持网络连接的场合。

3.11.2 浅睡眠模式

浅睡眠模式是 WiFi 协议规定的一种节电工作模式，需要 STA 联网成功后方可启动。浅睡眠模式下芯片会自动在休眠/唤醒状态间切换，能够保证当前的 WiFi 连接不断，但是通信性能会很受影响。

应用软件通过调用 InfPowerSaveSet 接口来进入或退出浅睡眠模式。浅睡眠模式下只能通过芯片内部的定时器唤醒，应用软件可以通过 InfListenIntervalSet 接口对睡眠周期进行设置。浅睡眠模式唤醒后，软件从进入睡眠之前的位置继续运行。

需要注意的是，WiFi 协议规定当有 STA 进入节电模式时，Ap 只能在每个 DTIM 时刻发送多播帧。因此，如果要侦听 DTIM 时刻的数据，则需要将 ListenDtim 参数设置为 True。该情况下的睡眠周期结束时刻以及 DTIM 时刻都会触发芯片唤醒。

注：浅睡眠模式比较适合于需要保证网络连接不断开，但又有节电要求的场合。但由于浅睡眠模式下的通信性能和实时性都得不到保证，因此不适合作为正常工作模式去使用。应用软件可根据需要在浅睡眠模式和正常工作模式下进行动态切换，例如：应用软件可以在浅睡眠模式下通过 WiFi 网络接收退出浅睡眠模式的消息，并切换到正常工作模式去收发业务数据，完成业务数据收发后再进入浅睡眠模式，如此反复。

3.11.3 有无 32K 晶体

NL6621 芯片进入以上两种睡眠方式后，唤醒逻辑都要依靠 32KHz 时钟来驱动。32KHz 时钟的来源有两种方式：

1） 通过外部 32K 晶体产生

该方式下，内部其它时钟信号都将关闭，能够最大程度降低功耗。

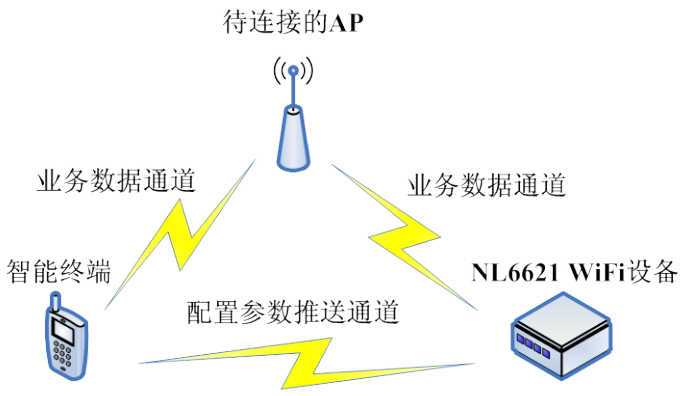
2） 通过内部时钟分频产生

该方式下，内部 40MHz 时钟信号一直保留，会消耗一部分电流，但可以节省一个 32K 晶体。

3.12 DirectConfig 功能应用

DirectConfig 功能用于 NL6621 WiFi 设备在联网之前通过无线方式获得联网需要的配置参数，包括 Ssid 和 Key。该功能需要与客户端应用程序配合使用。客户端可以是 PC、智能手机、PAD 等支持 WiFi 功能的终端设备。一个客户端可以同时配置多个 NL6621 WiFi 设备。

网络拓扑结构如下图所示。



主要工作步骤如下：

1） 智能终端连接到上图中的 AP 设备

2） NL6621 固件启动 DirectConfig 功能

3） 智能终端通过应用软件推送配置参数

4） NL6621 固件获得配置参数，并保存到 SysParam.WiFiCfg 变量中

5） NL6621 固件产生系统事件 SYS_EVT_DIRECT_CFG_DONE

以下是 NL6621 固件通过 DirectConfig 获取配置参数并联网的示例代码

```
VOID TestAppMain(VOID * pParam)
{
    …

    InfSysEvtCBSet(AppEvtCallBack);
    InfLoadDefaultParam();
    InfDirectCfgStart(0, 0);

    …
}

VOID AppEvtCallBack(SYS_EVT_ID event)
{
    switch (event)
    {
        …
        case SYS_EVT_DIRECT_CFG_DONE:
            DBGPRINT(DEBUG_TRACE, "SYS_EVT_DIRECT_CFG_DONE\n");
            InfWiFiStop();
            InfNetModeSet(PARAM_NET_MODE_STA_BSS);
            InfWiFiStart();
            break;

        default:
            break;
    }
}
```

智能终端应用程序的实现请参考 Tool\DirectConfigDemo 程序源码（C 语言）。可以利用 DirectConfigDemo.exe 程序在 PC 机上测试 DirectConfig 功能，使用方法参见其中的《DirectConfigDemo 程序说明》。

另外，本 SDK 还提供了 Android 手机的 DirectConfig 客户端程序 Tool\NLConfigTool.apk 及其源码 Tool\ NLConfigTool.rar，使用方法请参考本文 5.3 节。

注：启动 DirectConfig 功能后，NL6621 会按照信号强度由高到低的顺序依次侦听各信道下 AP 的网络

数据。在 AP 数量比较多的环境下，为了提高 DirectConfig 过程的成功率，建议通过接口 InfDirectCfgStart 的参数 MinRssiFilter 和 TryPeerNum 对需要侦听的 AP 信号强度以及最大个数进行限制。

4 编程接口

4.1 参数配置接口

参数配置接口主要用于对 WiFi 功能启动前的配置参数进行设置，设置完毕后再调用 InfWiFiStart 接口启动 WiFi 功能。

4.1.1 InfLoadDefaultParam

VOID InfLoadDefaultParam (VOID)

程序描述	加载默认配置参数		
输入参数	变量名	类型	描述
返回值类型			
备注			

4.1.2 InfConfigQuery

INT32 InfConfigQuery(CFG_PARAM *pParam)

程序描述	查询当前配置信息		
输入参数	变量名	类型	描述
	pParam	CFG_PARAM *	用来存放配置信息数据结构体指针，该内存由调用方分配和释放。
返回值类型	0：成功 非 0：失败		
备注			

4.1.3 InfNetModeSet

INT32 InfNetModeSet(PARAM_NET_MODE Mode)

程序描述	设置网络模式（STA、ADHOC、AP）

输入参数	变量名	类型	描述
	Mode	PARAM_NET_MODE	网络模式 PARAM_NET_MODE_STA_BSS PARAM_NET_MODE_STA_ADHOC PARAM_NET_MODE_SOFTAP
返回值类型	0：成功 非 0：失败		
备注			

4.1.4 InfPhyModeSet

INT32 InfPhyModeSet(PARAM_PHY_MODE Mode)

程序描述	设置物理层工作模式（802.11b/g/n）		
输入参数	变量名	类型	描述
	Mode	PARAM_PHY_MODE	物理层工作模式 PARAM_PHY_MODE_BGN PARAM_PHY_MODE_BG PARAM_PHY_MODE_B_ONLY
返回值类型	0：成功 非 0：失败		
备注			

4.1.5 InfSsidSet

INT32 InfSsidSet(UINT8 *pSsid, UINT8 SsidLen)

程序描述	设置 SSID		
输入参数	变量名	类型	描述
	pSsid	UINT8 *	Ssid 字符串
	SsidLen	UINT8	Ssid 长度，1~32 字节，不含\0

返回值类型	0：成功 非 0：失败
备注	

4.1.6 InfSsidBrdcastSet

INT32 InfSsidBrdcastSet(UINT8 Enable);

程序描述	设置 SSID 广播使能		
输入参数	变量名	类型	描述
	Enable	UINT8	0-隐藏 SSID 1-广播 SSID
返回值类型	0：成功 非 0：失败		
备注			

4.1.7 InfChannelSet

INT32 InfChannelSet(UINT8 Ch);

程序描述	设置工作信道		
输入参数	变量名	类型	描述
	Ch	UINT8	信道号，1~13
返回值类型	0：成功 非 0：失败		
备注	只能在 Adhoc 或 AP 模式下生效		

4.1.8 InfEncModeSet

INT32 InfEncModeSet(PARAM_ENC_MODE EncMode);

程序描述	设置加密方式		
输入参数	变量名	类型	描述

	EncMode	PARAM_ENC_MODE	加密方式 PARAM_ENC_MODE_OPEN PARAM_ENC_MODE_WEP PARAM_ENC_MODE_TKIP PARAM_ENC_MODE_CCMP PARAM_ENC_MODE_AUTO
返回值类型	0：成功 非 0：失败		
备注	在 STA 模式下，使用 PARAM_ENC_MODE_AUTO 方式，只需指定网络名称和密码，不需要指定具体的加密和认证方式，便可加入网络，使用更加灵活。 在 ADHOC 或者 AP 模式下 PARAM_ENC_MODE_AUTO 方式不生效，默认采用 PARAM_ENC_MODE_OPEN 方式。		

4.1.9 InfAuthModeSet

INT32 InfAuthModeSet(PARAM_AUTH_MODE AuthMode);

程序描述	设置认证方式		
输入参数	变量名	类型	描述
	AuthMode	PARAM_AUTH_MODE	认证方式 PARAM_AUTH_MODE_OPEN PARAM_AUTH_MODE_SHARE PARAM_AUTH_MODE_WPAPSK PARAM_AUTH_MODE_WPA2PSK
返回值类型	0：成功 非 0：失败		
备注			

4.1.10 InfKeySet

INT32 InfKeySet(UINT8 Index, UINT8 *KeyBuf, UINT8 KeyLen);

程序描述	设置密钥数据

输入参数	变量名	类型	描述
	Index	UINT8	密钥索引，仅对 WEP 加密方式有效，0~3
	KeyBuf	UINT8 *	密钥内容
	KeyLen	UINT8	密钥长度，不超过 64 字节
返回值类型	0：成功 非 0：失败		
备注			

4.1.11 InfConTryTimesSet

INT32 InfConTryTimesSet(UINT8 TryTimes);

程序描述	设置 STA 模式下自动联网次数		
输入参数	变量名	类型	描述
	TryTimes	UINT8	联网重试次数，0 表示一直重试，大于 0 表示重试次数
返回值类型	0：成功 非 0：失败		
备注	STA 模式下联网重试超过 TryTimes 时会停止联网过程，并产生 SYS_EVT_JOIN_FAIL 事件		

4.1.12 InfWmmEnableSet

INT32 InfWmmEnableSet(BOOLEAN Enable)

程序描述	设置 WMM 使能		
输入参数	变量名	类型	描述
	Enable	BOOLEAN	NST_TRUE：使能 NST_FALSE：不使能
返回值类型	0：成功 非 0：失败		
备注			

4.1.13 InfWscEnSet

INT32 InfWscEnSet(BOOLEAN WscEn);

程序描述	设置 AP 模式下的 WPS 使能与否		
输入参数	变量名	类型	描述
	WscEn	BOOLEAN	1:WPS enable 0:WPS disable
返回值类型	0：成功 非 0：失败		
备注	仅限 AP 模式下使用。另外遵循 IEEE802.11 协议规定，AP 模式下，如果加密方式为 WEP sharekey，WPS 功能无法使能。		

4.1.14 InfIpSet

INT32 InfIpSet(IP_CFG *IpParam);

程序描述	设置 IP 参数		
输入参数	变量名	类型	描述

	IpParam	IP_CFG *	typedef struct _IP_CFG{
			INT8U Ip[4];
			INT8U Netmask[4];
			INT8U GateWay[4];
			INT8U Dns[4];
			INT8U bDhcp; //dhcps 使能
			INT8U DhcpTryTimes
			INT8U Res_2[2];
			}IP_CFG;

返回值类型	0：成功 非 0：失败		
备注	使能 dhcp 时,通过 dhcp 请求 IP 地址，DhcpTryTimes 规定尝试次数，DhcpTryTimes 为 0 表示一直尝试 dhcp 请求，直到成功分配 IP 地址。		

4.1.15 InfDhcpSrvSet

INT32 InfDhcpSrvSet(UINT8 Enable);

程序描述	Dhcp Server 功能开关		
输入参数	变量名	类型	描述
	Enable	UINT8	0-禁用；1-启用
返回值类型	0：成功 非 0：失败		
备注	只在 AP 模式下生效		

4.1.16 InfDnsSrvSet

INT32 InfDnsSrvSet(UINT8 Enable, UINT8 *DnsName, UINT8 DnsNameLen);

程序描述	Dns Server 功能开关		
输入参数	变量名	类型	描述
	Enable	UINT8	0-禁用；1-启用
	DnsName	UINT8 *	本地 Dns 域名
	DnsNameLen	UINT8	本地 Dns 域名长度
返回值类型	0：成功 非 0：失败		
备注	只在 AP 模式下生效，且只能设置一个域名给 AP		

4.1.17 InfSysEvtCBSet

INT32 InfSysEvtCBSet(SysEvtCallBack pFuncCb);

程序描述	设置系统事件回调函数		
输入参数	变量名	类型	描述
	pFuncCb	SysEvtCallBack	系统事件回调函数指针，用于接收系统事件
返回值类型	0：成功 非 0：失败		
备注			

4.1.18 InfMacAddrSet

INT32 InfMacAddrSet(UINT8 Addr[6]);

程序描述	设置 Mac 地址		
输入参数	变量名	类型	描述
	Addr	UINT8 [6]	待设置的 MAC 地址
返回值类型	0：成功 非 0：失败		
备注	只能在系统任务启动前设置 MAC 地址，建议在 OSStart 之前完成设置		

4.2 WiFi 控制接口

WiFi 控制接口用于启动/停止 WiFi 功能，以及在 WiFi 功能启动后进行实时控制或者查询相关信息。

4.2.1 InfWiFiStart

INT32 InfWiFiStart(VOID);

程序描述	启动 WiFi 功能		
输入参数	变量名	类型	描述
返回值类型	0：成功 非 0：失败		
备注	所有参数设置完成后才可以调用该接口		

4.2.2 InfWiFiStop

INT32 InfWiFiStop(VOID);

程序描述	停止 WiFi 功能		
输入参数	变量名	类型	描述

返回值类型	0：成功 非 0：失败		
备注	重新设置 WiFi 参数前，必须先停止 WiFi 功能。		

4.2.3 InfScanStart

INT32 InfScanStart(SCAN_INFO_TABLE * pScanTable, INT8 MinRssiFilter, INT8 ScanType)

程序描述	发起扫描过程		
输入参数	变量名	类型	描述
	pScanTable	SCAN_INFO_TABLE *	存放扫描结果的 Buffer，pScanTable->InfoCount 指示 Buffer 可存放的最大结果数，取值范围为 1~32
	MinRssiFilter	INT8	通过 rssi 来过滤扫描结果，低于 MinRssiFilter 时，则不保存到结果列表当中，取值范围-1 ~ -127，超出该范围则不作过滤
	ScanType	INT8	0-主动扫描 其它-被动扫描
返回值类型	0：成功 非 0：失败		
备注	1） 该接口只能在调用 InfWiFiStart 成功后调用。 2） 该函数立刻返回，扫描完成后会调用系统事件回调接口通知应用程序，扫描结果保存在 pScanTable 指定的 Buffer 中，pScanTable->InfoCount 指示实际扫描到的结果数量。 3） pScanTable 由调用程序负责内存的分片和释放，且 pScanTable->InfoCount 需要初始化为实际分配的 InfoList 单元个数。		

4.2.4 InfTxRateSet

INT32 InfTxRateSet(PARAM_RATE_MODE RateMode, PARAM_RATE_IDX RateIdx)

程序描述	设置发送速率		
输入参数	变量名	类型	描述
	RateMode	PARAM_RATE_MODE	固定发送速率时采用的调制模式
	RateIdx	PARAM_RATE_IDX	发送速率索引，0 为自动调速 1~8 为固定速率索引
返回值类型	0：成功 非 0：失败		
备注	1） 只有在收到 SYS_EVT_LINK_UP 事件以后调用该接口才能生效 2） 速率设置规则如下： MODE_CCK: RateIdx Idx0~ Idx3 => 1, 2, 5.5, 11 (Mbps) MODE_OFDM: RateIdx Idx0~ Idx7 => 6, 9, 12, 18, 24, 36, 48, 54 (Mbps) MODE_HT: RateIdx Idx0~ Idx7 => Mcs0~Mcs7		

4.2.5 InfTxPwrLevelSet

INT32 InfTxPwrLevelSet(PARAM_TX_PWR_LEVEL TxPwrLevel)

程序描述	设置发送功率级别		
输入参数	变量名	类型	描述
	TxPwrLevel	PARAM_TX_PWR_LEVEL	发送功率级别，分三档：0-低，1- 中，2-高
返回值类型	0：成功 非 0：失败		
备注	每档发送功率级别之间大约相差 5dbm，每个速率下的最大发送功率应由厂测软件进行校准。		

4.2.6 InfPowerSaveSet

INT32 InfPowerSaveSet(UINT8 PowerSaveMode)

程序描述	设置 802.11 协议节电功能		
输入参数	变量名	类型	描述
	PowerSaveMode	UINT8	0- 不使能 1- 传统节电模式，通过向 AP 发送 PS-POLL 获取缓存数据 2- 快速节电模式，通过动态切换到 Active 状态来接收和发送数据。
返回值类型	0：成功 非 0：失败		
备注	只在 STA 模式下生效，节电过程中不断网。		

4.2.7 InfDeepSleepSet

INT32 InfDeepSleepSet(UINT32 WakeUpMode)

程序描述	设置深度休眠		
输入参数	变量名	类型	描述
	WakeUpMode	UINT32	唤醒方式选择： 0x00000000 -由 Gpio0 唤醒 0x8xxxxxxx -由 32K 定时器唤醒, bit0 ~ 25 指示唤醒等待时间(ms), bit26 ~30 保留
返回值类型	0：成功 非 0：失败		

备注	进入深度休眠后，芯片处于最低功耗状态，唤醒后需要重新加载固件程序。 采用 GPIO 唤醒时，要保证对应的 GPIO 配置为输入方向，且 GPIO 信号发生跳变后的保持时间大于一个 32K 时钟周期，信号跳变方向可以任意。

4.2.8 InfPeerRssiGet

INT8 InfPeerRssiGet(UINT8 pMacAddr[6])

程序描述	获取对端站点的信号强度		
输入参数	变量名	类型	描述
	pMacAddr	UINT8[6]	对端 Mac 地址
返回值类型	0：成功 非 0：失败		
备注			

4.2.9 InfCurChGet

UINT8 InfCurChGet(VOID)

程序描述	查询当前工作信道		
输入参数	变量名	类型	描述
返回值类型	信道号		
备注			

4.2.10 InfEfuseInfoGet

INT32 InfEfuseInfoGet(UINT8 Offset, UINT8 Cnt, UINT32* pEfuseInfo)

程序描述	读取 Efuse 信息		
输入参数	变量名	类型	描述

	Offset	UINT8	读取地址偏移，以字为单位，0~7
	Cnt	UINT8	读取字的个数
	pEfuseInfo	UINT32*	存放 Efuse 读取结果的 Buffer
返回值类型	0：成功 非 0：失败		
备注	Efuse 大小一共 32 字节,空间分配情况如下： Byte0 - Byte6：存放 CHIP ID，每个芯片唯一，芯片封测时写入 Byte 7 - Byte 27：开放给用户使用，由厂测软件写入 Byte 28 - Byte 31：存放芯片内部逻辑使用数据，芯片封测时写入		

4.2.11 InfBeaconPeriodSet

INT32 InfBeaconPeriodSet(UINT16 BcnPrd, UINT8 NotifyPeerEn)

程序描述	设置 Beacon 发送周期		
输入参数	变量名	类型	描述
	BcnPrd	UINT16	Beacon 周期，单位为毫秒，最小可设为 100ms
	NotifyPeerEn	UINT8	指示是否让所连接的对端站点断网重连。 0-不重连，1-重连 只在 SoftAp 模式下生效。
返回值类型	0：成功 非 0：失败		
备注	该接口对于 SoftAp 和 Adhoc 模式下的 Beacon 周期均可进行设置。		

4.2.12 InfListenIntervalSet

INT32 InfListenIntervalSet(UINT8 ListenIntv, BOOL_T ListenDtim)

程序描述	设置 STA 协议节电模式下的侦听间隔周期（睡眠周期）		
输入参数	变量名	类型	描述
	ListenIntv	UINT8	侦听间隔周期，有效范围 1~255，典型值为 1~3，以 TBTT 周期为单位。
	ListenDtim	BOOL_T	指示是否侦听 DTIM ture-侦听，false-不侦听

返回值类型	0：成功 非 0：失败
备注	AP 会在 DTIM 时刻发送缓存的广播数据帧，在侦听 DTIM 的情况下，实际的侦听间隔周期取 ListenIntv 与 Ap Dtim 周期的较小值。

4.2.13 InfDirectCfgStart

INT32 InfDirectCfgStart(INT8 MinRssiFilter, UINT8 TryPeerNum)

程序描述	启动 Direct Config 功能		
输入参数	变量名	类型	描述
	MinRssiFilter	INT8	指定需要侦听 AP 的最小信号强度。 有效值范围为 -1~-127（dbm），设置为其它值时，该过滤条件不生效。
	TryPeerNum	UINT8	指定需要侦听 AP 的最大个数。 设置为 0 时，对能够被扫描到的 AP 都进行侦听。
返回值类型	0：成功 非 0：失败		
备注	该接口用于在联网之前通过无线方式获得 STA 联网需要的配置参数。 该接口内部会启动 WiFi 功能，调用该接口之前不可调用 InfWiFiStart。 DirectConfig 过程完成后，会将 ssid 和 key 信息保存到 SysParam.WiFiCfg 变量当中。		

4.2.14 InfSnifferStart

INT32 InfSnifferStart(MonRecvCallBack pFuncCb, UINT8 BindBssid[6], UINT8 Channel)

程序描述	启动嗅探功能		
输入参数	变量名	类型	描述
	pFuncCb	MonRecvCallBack	嗅探器接收数据回调接口指针，回调接口由用户自定义，SDK 在接收到 802.11 协议数据后会调用该接口。
	BindBssid	UINT8[6]	需要嗅探的网络 BSSID
	Channel		需要嗅探的信道
返回值类型	0：成功 非 0：失败		
备注	该嗅探功能一次只能对一个 BSSID 所指示的网络进行嗅探。可以通过调用接口 InfWiFiStop 来停止嗅探功能。		

4.2.15 InfPeerAgeOutSet

INT32 InfPeerAgeOutSet(INT32U Time)

程序描述	该接口用于设置 AP 模式下站点的老化时间		
输入参数	变量名	类型	描述
	Time	UINT32	站点老化时间，以秒为单位，设置为 0 时表示不对站点进行老化处理。
返回值类型	0：成功 非 0：失败		
备注	AP 模式下，当未收到某个接入站点任何数据的时间周期超过老化时间时，则启动对该站点的老化过程		

4.2.16 InfPeerKickOut

INT32 InfPeerKickOut(UINT8 pMacAddr[6])

程序描述	该接口用于 AP 模式下对已接入的站点解除关联		
输入参数	变量名	类型	描述
	pMacAddr	UINT8[6]	待解除关联的站点 MAC 地址
返回值类型	0：成功 非 0：失败		
备注			

4.2.17 InfProtectModeSet

INT32 InfProtectModeSet(PROTECT_MODE Mode)

程序描述	该接口用于设置 B/G 保护模式		
输入参数	变量名	类型	描述
	Mode	PROTECT_MODE	PROTECT_MODE_AUTO：自动使用 B/G 保护 PROTECT_MODE_ALWAYS：强制启用 B/G 保护 PROTECT_MODE_NONE：不启用 B/G 保护
返回值类型	0：成功 非 0：失败		
备注			

4.2.18 InfWPSStart

INT32 InfWPSStart(UINT8 WscMode, UINT32 PinCode)

程序描述	该接口用于 STA/AP 模式下，启动 WPS 操作		
输入参数	变量名	类型	描述

	WscMode	UINT8	0：PARAM_WSC_MODE_DISABLE 1：PARAM_WSC_MODE_PIN: 2：PARAM_WSC_MODE_PBC
	PinCode	UINT32	对端 PIN 码，常见于对端设备的标签或配置界面上，只在启动 AP WPS PIN 模式时进行配置。
返回值类型	0：成功 非 0：失败		
备注	1、AP 模式下该接口需要在配置接口中配置了 InfWSCEnSet 接口，使能 WPS 功能后方可调用。 2、STA PBC、PIN 模式，AP PBC 模式下，调用该接口时 PinCode 参数设置为 0。 3、与 InfWPSStop 接口成对使用。若多次循环调用时，需先调用 InfWPSStop 对上一次的 WPS 操作进行终止，方可启动下一次 WPS 操作		

4.2.19 InfWPSStop

INT32 InfWPSStop(VOID)

程序描述	该接口用于 STA/AP 模式下，停止 WPS 操作		
输入参数	变量名	类型	描述
返回值类型	0：成功 非 0：失败		
备注	与 InfWPSStart 接口成对使用。若多次循环调用时，需先调用该接口对上一次的 WPS 操作进行终止，方可启动下一次 WPS 操作		

4.2.20 InfStaWpsPinGet

INT32 InfStaWpsPinGet(UINT8 pWpsPin[9])

程序描述	STA WPS 模式下，获取自身 PIN 码		
输入参数	变量名	类型	描述
	pWpsPin	UINT8	长度为 9 的数组，组成一个 PinCode 字符串。
返回值类型	0：成功 非 0：失败		
备注	目前 STA PIN 码为采用 MAC 地址计算而成，可调用该接口进行查询。		

4.2.21 InfVendorIESet

INT32 InfVendorIESet(UINT8 * pIePtr, UINT8 IeLen)

程序描述	AP 模式下设置 Beacon 和 Probe Response 自定义信息单元		
输入参数	变量名	类型	描述
	pIePtr	UINT8 *	信息单元数据指针
	IeLen	UINT8	信息单元数据长度
返回值类型	0：成功 非 0：失败		
备注	pIePtr 所指向的内存由接口调用者负责分配，在 AP 停止工作前不可以被释放。		

4.3 操作系统接口

参见《μCOS-II Reference Manual》

4.4 网络编程接口

4.4.1 Socket 宏定义

```
#define  accept(a,b,c)          lwip_accept(a,b,c)
#define  bind(a,b,c)            lwip_bind(a,b,c)
#define  shutdown(a,b)           lwip_shutdown(a,b)
#define  closesocket(s)         lwip_close(s)
#define  connect(a,b,c)         lwip_connect(a,b,c)
#define  getsockname(a,b,c)      lwip_getsockname(a,b,c)
```

```
#define  getpeername(a,b,c)       lwip_getpeername(a,b,c)
#define  setsockopt(a,b,c,d,e)  lwip_setsockopt(a,b,c,d,e)
#define  getsockopt(a,b,c,d,e)  lwip_getsockopt(a,b,c,d,e)
#define  listen(a,b)                 lwip_listen(a,b)
#define  recv(a,b,c,d)               lwip_recv(a,b,c,d)
#define  recvfrom(a,b,c,d,e,f)  lwip_recvfrom(a,b,c,d,e,f)
#define  send(a,b,c,d)               lwip_send(a,b,c,d)
#define  sendto(a,b,c,d,e,f)     lwip_sendto(a,b,c,d,e,f)
#define  socket(a,b,c)               lwip_socket(a,b,c)
#define  select(a,b,c,d,e)         lwip_select(a,b,c,d,e)
#define  ioctlsocket(a,b,c)       lwip_ioctl(a,b,c)
```

4.4.2 函数原型

4.4.2.1 lwip_socket

lwip_socket(int domain, int type, int protocol)

程序描述	建立网络套接字		
输入参数	变量名	类型	描述
	domain	int	地址族 AF_UNSPEC AF_INET
	type	int	Socket 类型 SOCK_RAW SOCK_DGRAM SOCK_STREAM
	protocol	int	协议类型 IPPROTO_IP IPPROTO_TCP IPPROTO_UDP IPPROTO_UDPLITE
返回值类型	int 即 sockId 0：成功 非 0：失败		
备注			

4.4.2.2 lwip_bind

lwip_bind(int s, struct sockaddr *addr, socklen_t addrlen)

程序描述	用于进行网络套接字绑定		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	addr	sockaddr *	套接字地址信息
	addrlen	socklen_t	地址信息实际字节数
返回值类型	int		
备注			

4.4.2.3 lwip_connect

lwip_connect(int s, const struct sockaddr *name, socklen_t namelen)

程序描述	用来将参数 s 的 socket 连至参数 addr 指定的网络地址		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	addr	sockaddr *	套接字地址信息
	addrlen	socklen_t	地址信息实际字节数
返回值类型	int		
备注			

4.4.2.4 lwip_getsockname

lwip_getsockname(int s, struct sockaddr *name, socklen_t *namelen)

程序描述	获取一个本地套接字的名字		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	name	sockaddr *	套接字地址信息
	namelen	socklen_t	名字实际字节数
返回值类型	int		
备注			

4.4.2.5 lwip_getpeername

lwip_getpeername(int s, struct sockaddr *name, socklen_t *namelen)

程序描述	获取一个对端套接字的名字		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	name	sockaddr *	对端套接字地址信息
	namelen	socklen_t	名字实际字节数
返回值类型	int		
备注			

4.4.2.6 lwip_setsockopt

lwip_setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen)

程序描述	设置套接字选项值		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	level	int	选项定义的层次，支持 SOL_SOCKET IPPROTO_TCP IPPROTO_IP IPPROTO_IPV6

	optname	int	需设置的选项 SO_DEBUG SO_ACCEPTCONN SO_REUSEADDR SO_KEEPALIVE SO_DONTROUTE SO_BROADCAST SO_USELOOPBACK SO_LINGER SO_OOBINLINE SO_REUSEPORT
	optval	const void *	指向存放选项值的缓冲区
	optlen	socklen_t	Optval 缓冲区长度
返回值类型	int		
备注			

4.4.2.7 lwip_getsockopt

lwip_getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen)

程序描述	读取套接字选项值		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	level	int	选项定义的层次，支持 SOL_SOCKET IPPROTO_TCP IPPROTO_IP IPPROTO_IPV6

	optname	int	需读取的设置选项名称 SO_DEBUG SO_ACCEPTCONN SO_REUSEADDR SO_KEEPALIVE SO_DONTROUTE SO_BROADCAST SO_USELOOPBACK SO_LINGER SO_OOBINLINE SO_REUSEPORT
	optval	const void *	指向存放选项值的缓冲区
	optlen	socklen_t	Optval 缓冲区长度
返回值类型	int		
备注			

4.4.2.8 lwip_listen

int lwip_listen(int s, int backlog)

程序描述	侦听套接字，用于服务端侦听连接。		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	addr	sockaddr *	本地套接字结构体
	addrlen	socklen_t	套接字结构体总字节数
返回值类型	int		
备注			

4.4.2.9 lwip_accept

lwip_accept(int s, struct sockaddr *addr, socklen_t *addrlen)

程序描述	通常在 listen 之后使用

输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	addr	sockaddr *	对端连接的地址信息
	addrlen	socklen_t *	地址信息长度
返回值类型	int		
备注			

4.4.2.10 lwip_recv

int lwip_recv(int s, void *mem, size_t len, int flags)

程序描述	Read 对方发来的数据		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	mem	void *	接收的缓冲区
	len	size_t	接收长度
	flags	int	接收标志
返回值类型	int		
备注			

4.4.2.11 lwip_recvfrom

int lwip_recvfrom(int s, void *mem, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)

程序描述	Read 对方发来的数据		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	mem	void *	接收的缓冲区
	len	size_t	接收长度
	flags	int	接收标志
	from	sockaddr *	远端套接字信息结构体
	fromlen	socklen_t *	远端套接字信息长度

返回值类型	int
备注	

4.4.2.12 lwip_send

int lwip_send(int s, const void *data, size_t size, int flags)

程序描述	给对方发送数据		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	data	void *	发送数据缓冲区
	size	size_t	发送数据长度
	flags	int	发送标志
返回值类型	int		
备注			

4.4.2.13 lwip_sendto

int lwip_sendto(int s, const void *data, size_t size, int flags, const struct sockaddr *to, socklen_t tolen)

程序描述	给对方发送数据		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	data	void *	发送数据缓冲区
	size	size_t	发送数据长度
	flags	int	发送标志
	to	sockaddr *	发送的目的地套接字信息
	tolen	socklen_t *	发送的目的套接字信息长度

返回值类型	int
备注	

4.4.2.14 lwip_select

int lwip_select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)

程序描述	多 IO 复用侦听，同时侦听多个读写描述符集中的套接字		
输入参数	变量名	类型	描述
	maxfdp1	int	最大套接字号
	readset	fd_set *	读描述符集合
	writeset	fd_set *	写描述符集合
	exceptset	fd_set *	排除描述符集合
	timeval	timeval *	超时设定时间
返回值类型	int		
备注			

4.4.2.15 lwip_ioctl

int lwip_ioctl(int s, long cmd, void *argp)

程序描述	获取网络套接字连接的信息或发送控制连接的参数		
输入参数	变量名	类型	描述
	s	int	标志一个套接字的描述字
	cmd	long	读取或写入命令类型
	argp	void *	命令参数
返回值类型	int		
备注			

4.5 公共接口

4.5.1 定时器

4.5.1.1 NST_InitTimer

VOID NST_InitTimer(IN NST_TIMER ** pTimer,
IN OS_TMR_CALLBACK TimerFunc,
IN PVOID pData,
IN BOOL_T Repeat)

程序描述	创建定时器		
输入参数	变量名	类型	描述
	pTimer	NST_TIMER **	用来存放定时器指针变量地址，定制器指针所指向的变量由接口内部分配内存
	TimerFunc	OS_TMR_CALLBACK	定时器回调函数指针
	pData	PVOID	定时器回调函数中的上下文数据
	Repeat	BOOL_T	循环定时器标记
返回值类型			
备注			

4.5.1.2 NST_DelTimer

VOID NST_DelTimer(IN NST_TIMER ** pTimer, OUT BOOL_T *pCancelled)

程序描述	删除定时器		
输入参数	变量名	类型	描述
	pTimer	NST_TIMER **	待删除的定时器指针地址
输出参数	pCancelled	BOOL_T *	删除成功标记
返回值类型			
备注			

4.5.1.3 NST_SetTimer

VOID NST_SetTimer(IN NST_TIMER * pTimer, IN UINT32 Value)

程序描述	启动定时器		
输入参数	变量名	类型	描述
	pTimer	NST_TIMER *	待启动的定时器指针
	Value	UINT32	定时器超时时间，单位为 ms
返回值类型			
备注	该定时器精度为 10ms		

4.5.1.4 NST_CancelTimer

VOID NST_CancelTimer(IN NST_TIMER * pTimer, OUT BOOL_T *pCancelled)

程序描述	取消定时器		
输入参数	变量名	类型	描述
	pTimer	NST_TIMER *	待取消的定时器指针
	pCancelled	BOOL_T *	取消成功标记
返回值类型			
备注			

4.5.1.5 NST_ModTimer

VOID NST_ModTimer(IN NST_TIMER *pTimer, IN UINT32 Value)

程序描述	重新启动定时器		
输入参数	变量名	类型	描述
	pTimer	NST_TIMER *	待重启的定时器指针
	Value	UINT32	定时器超时时间，单位为 ms
返回值类型			
备注	该定时器精度为 10ms		

4.5.2 互斥锁

4.5.2.1 NST_ALLOC_LOCK

VOID NST_ALLOC_LOCK(OUT NST_LOCK ** Lock)

程序描述	创建互锁		
输入参数	变量名	类型	描述

	Lock	NST_LOCK **	用来存放互锁指针变量地址
返回值类型			
备注			

4.5.2.2 NST_FREE_LOCK

VOID NST_FREE_LOCK(IN NST_LOCK * Lock)

程序描述	释放互锁		
输入参数	变量名	类型	描述
	Lock	NST_LOCK *	待释放的互锁指针
返回值类型			
备注			

4.5.2.3 NST_AQUIRE_LOCK

VOID NST_AQUIRE_LOCK(IN NST_LOCK * Lock)

程序描述	获取互锁		
输入参数	变量名	类型	描述
	Lock	NST_LOCK *	待获取的互锁指针
返回值类型			
备注			

4.5.2.4 NST_RELEASE_LOCK

VOID NST_RELEASE_LOCK(IN NST_LOCK * Lock)

程序描述	释放互锁		
输入参数	变量名	类型	描述
	Lock	NST_LOCK *	待释放的互锁指针
返回值类型			
备注			

4.5.3 内存拷贝

4.5.3.1 NST_MoveMemOvlap

VOID NST_MoveMemOvlap(UINT8 *pDst, UINT8 *pSrc, UINT16 Size)

程序描述	拷贝内存，原内存区与目的内存区可以有重合区域		
输入参数	变量名	类型	描述
	pDst	UINT8 *	目的内存地址
	pSrc	UINT8 *	原内存地址
	Size	Size	拷贝内存长度
返回值类型			
备注			

4.5.3.2 NST_MoveBigMem

VOID NST_MoveBigMem(UINT8 *pDst, UINT8 *pSrc, UINT16 Size)

程序描述	大块内存拷贝		
输入参数	变量名	类型	描述
	pDst	UINT8 *	目的内存地址
	pSrc	UINT8 *	原内存地址
	Size	Size	拷贝内存长度
返回值类型			
备注	当源地址与目的地址都为 4 字节边界对齐时采用字拷贝，否则采用字节拷贝		

4.5.3.3 NST_DmaMoveMem

VOID NST_DmaMoveMem(UINT8 *pDst, UINT8 *pSrc, UINT16 Size)

程序描述	DMA 内存搬移		
输入参数	变量名	类型	描述
	pDst	UINT8 *	目的内存地址
	pSrc	UINT8 *	原内存地址
	Size	Size	拷贝内存长度
返回值类型			
备注	该接口会发生阻塞，直到 DMA 完成		

4.5.4 杂项

4.5.4.1 NST_GetCrc32

INT32 NST_GetCrc32(UINT8 *pData, INT32 DataSize)

程序描述	计算 32 位 CRC		
输入参数	变量名	类型	描述
	pData	UINT8 *	待计算的数据源
	DataSize	INT32	待计算的数据长度
返回值类型	CRC 结果		
备注			

4.5.4.2 AtoH

void AtoH(PSTRING src, PUINT8 dest, int destlen)

程序描述	将 ASCII 码表示的 16 进制数转换为数值形式		
输入参数	变量名	类型	描述
	src	PSTRING	待转换的字符串指针
	dest	PUINT8	存放转换结果的指针
	destlen	int	转换结果长度
返回值类型			
备注			

5 测试代码范例

5.1 TCPIP 测试范例

本 SDK 提供了 TCP/IP 协议功能测试代码（位于 app_main.c 文件中），可进行简单的 TCP、UDP 通信测试。

表 8 TCPIP 测试范例编译选项

配置选项	说 明
TEST_APP_SUPPORT	测试代码总开关（app_cfg.h）
LWIP_SUPPORT	TCPIP 测试代码必须开启 lwip 功能（app_cfg.h）
TEST_TCP_SERVER	开启 TCP 服务器测试功能
TEST_TCP_CLIENT	开启 TCP 客户端测试功能
TEST_UDP_SERVER	开启 UDP 服务器测试功能
TEST_UDP_CLIENT	开启 UDP 客户端测试功能

本测试代码所涉及的配置选项如上表所示，其中 TEST_TCP_SERVER 功能默认开启，运行于主线程，而 TEST_TCP_CLIENT、TEST_UDP_SERVER、TEST_UDP_CLIENT 功能共享一个子线程，只能开启其中之一。

下面对各个部分测试代码进行详细介绍。

5.1.1 TCPServer 测试范例

TCPServer 功能完成 TCP 服务器端的角色，实现面向连接的传输，提供可靠的端到端通信。目前代码中提供了简单的单线程 TCPserver 范例代码，函数名称为 TestTcpServer。该示例代码中示范了一个 TCPServer 是如何建立的，如何创建一个 socket，如何与对端 TCPClient 进行通讯。开启该测试功能需在 app_cfg.h 中打开#define TEST_TCP_SERVER 选项。

下面对代码的具体实现以及使用做个简单介绍：

首先创建一个 socket，并绑 定 socket 到服务器端地址，此处 TcpServer 会自动侦听客户端的连接请求，并接受第一个发起连接的客户端的 socket 连接请求。调用 LWIP 中 recv 函数来实现对 Client 端发出数据的接收。

用户在使用时需要对以下参数根据自己现有的网络情况进行配置。

表 9 TCPServer 配置

配置选项	默认设置	说明
TCP_SERVER_PORT	10000	TCPServer 端口，使用时需要在 Client 端输入该端口号，用户可自行修改。
INADDR_ANY	IPADDR_ANY	允许连接的对端 IP 地址，目前设置为任何 IP 地址均可。
RECV_BUFFER_SIZE	2048	每次接收时可接收的最大包，用户可自行修改

测试时对端 Client 可选用 Iperf 作为测试工具，进行打流操作。

具体命令为：

```
iperf -c （Server 端 IP 地址）-p （Server 端端口号） -i 1 -t 10
```

例如：

```
iperf -c 192.168.0.9 -p 10000 -i 1 -t 10
```

5.1.2 TCPClient 测试范例

TCPClient 与对端 TCPServer 一起完成 TCP 数据流通信。目前代码中提供了简单的 TCPClient 范例代码，函数名称为 TestTcpClient。该示例代码中示范了如何创建一个 socket，如何连接到对端服务器并与对端 TCPServer 进行通讯。开启该测试功能需在 app_cfg.h 中打开#define TEST_TCP_CLIENT 选项。

下面对代码的具体实现以及使用做个简单介绍：

首先创建一个 socket，根据服务器端信息，发起连接，如果当时连接失败会间隔二十秒之后再次发起连接，直到与对端连接成功。调用 LWIP 中 send 函数来实现数据发送。

用户在使用时需要对以下参数根据自己现有的网络情况进行配置。

表 10 TCPClient 配置

配置选项	默认设置	说明
TCP_CLIENT_PORT	10001	对端 TCPServer 端口，用户可自行修改，只需与 TCPServer 端保持一致即可。
TCP_SERVER_IP_ADR	0xC0A80002（192.168.0.2）	对端 TCPServer IP 地址
APP_DATA_LEN	2048	每次发送时可发送的最大包，用户可自行修改
SendData[APP_DATA_LEN]	随机数	该数组为发送数据缓存数组，为 TCPClient 发送数据的来源，

		用户可自行修改代码选择数据源。

测试时对端 Client 可选用 Iperf 作为测试工具，进行打流操作。

具体命令为：

```
iperf -s -p （Server 端端口号） -i 1
```

例如：

```
iperf -s -p 10001 -i 1
```

5.1.3 UDPServer 测试范例

UDPServer 功能完成 UDP 通路中接收 UDP 数据帧端的角色，实现不可靠的端到端通信。目前代码中提供了简单的单线程 UDPServer 范例代码，函数名称为 TestUdpServer。该示例代码中示范了一个 UDP 数据通路中的接收操作。开启该测试功能需在 app_cfg.h 中打开#define TEST_UDP_SERVER 选项。

下面对代码的具体实现以及使用做个简单介绍：

首先创建一个 socket，并绑定 socket 到服务器端地址。调用 LWIP 中 recvfrom 函数来收取对端发出 UDP 数据包。为便于 UDP 测试，在收到 UDP 包之后，直接调用 LWIP 中 sendto 函数将 UDP 数据包发出，形成 UDP 回环，便于对方查看 UDP 环路情况以及丢包率。

用户在使用时需要对以下参数根据自己现有的网络情况进行配置。

表 11 UDPServer 配置

配置选项	默认设置	说明
UDP_SERVER_PORT	10002	UDPServer 端口，使用时需要在对端输入该端口号，用户可自行修改。
INADDR_ANY	IPADDR_ANY	允许连接的对端 IP 地址，目前设置为任何 IP 地址均可。
RECV_BUFFER_SIZE	2048	每次接收时可接收的最大包，用户可自行修改

测试方法：

测试时对端 Client 可选用 Iperf 作为测试工具，进行打流操作。由于有 UDP 环路的建立，亦可选用 TCP&UDP 测试工具进行 UDP 环路测试。

- Iperf 测试方法具体命令为：

```
iperf -c （Server 端 IP 地址）-p （Server 端端口号）-u -i 1
```

例如：

```
iperf -c 192.168.0.9 -p 10002 –u -i 1
```

- TCP&UDP 测试工具

  用户可在网络上下载 TCP&UDP 测试工具，在本测试的对端（即 UDP 数据的发送端）安装并进行测试。

  1、点击创建连接，出现创建连接界面(如下图所示)，类型选取 UDP，目标 IP 为当前作为 UDPServer 的开发平台 IP 地址，端口号为用户指定端口号（默认为 10002），点击“创建”按钮。

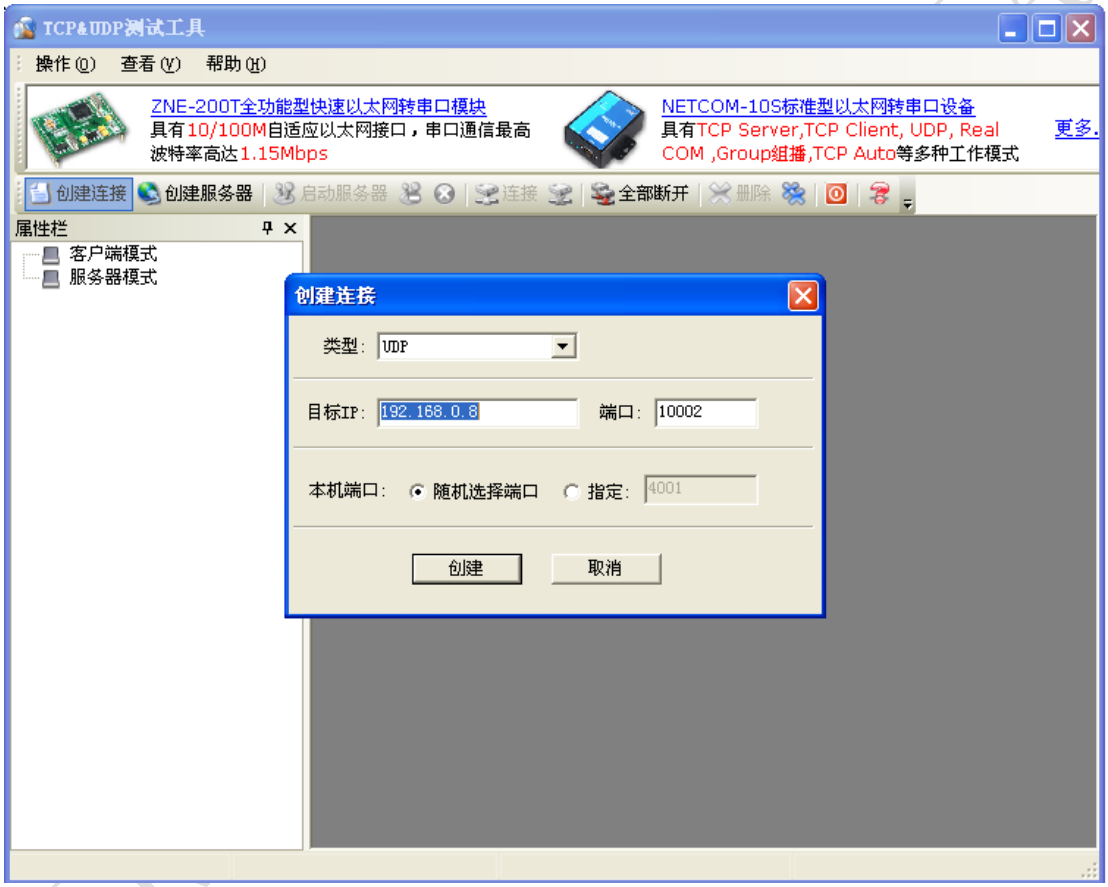


图 9 设置 UDP 参数

2、出现以下界面，点击“创建”，建立一个 UDP 连接，此时即可在发送窗口区输入待发送数据，亦可选择发送文件。

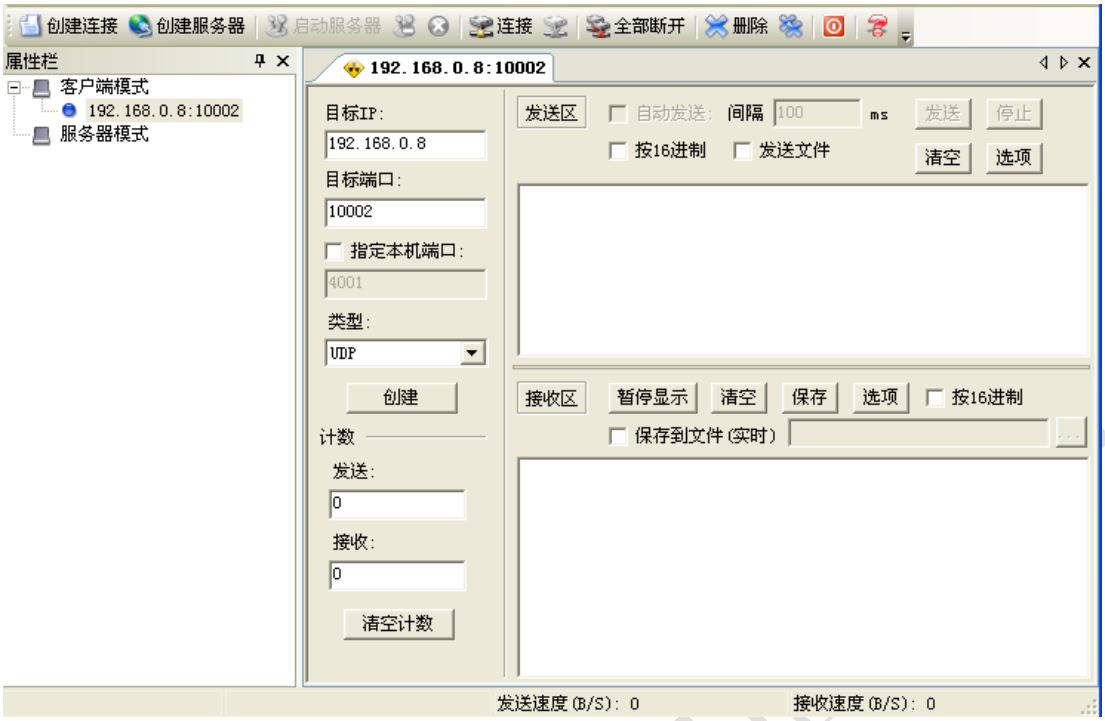


图 10 创建 UDP 连接

3、下图即为进行数据发送之后的界面，由于目前 UDPServer 设计为回环，故在数据未丢失的情况下，只要发送区发出，接收区会收到同样的数据，同时发送计数与接收计数相同。在进行该测试时，无线抓包可以看到双方交互的 UDP 帧。

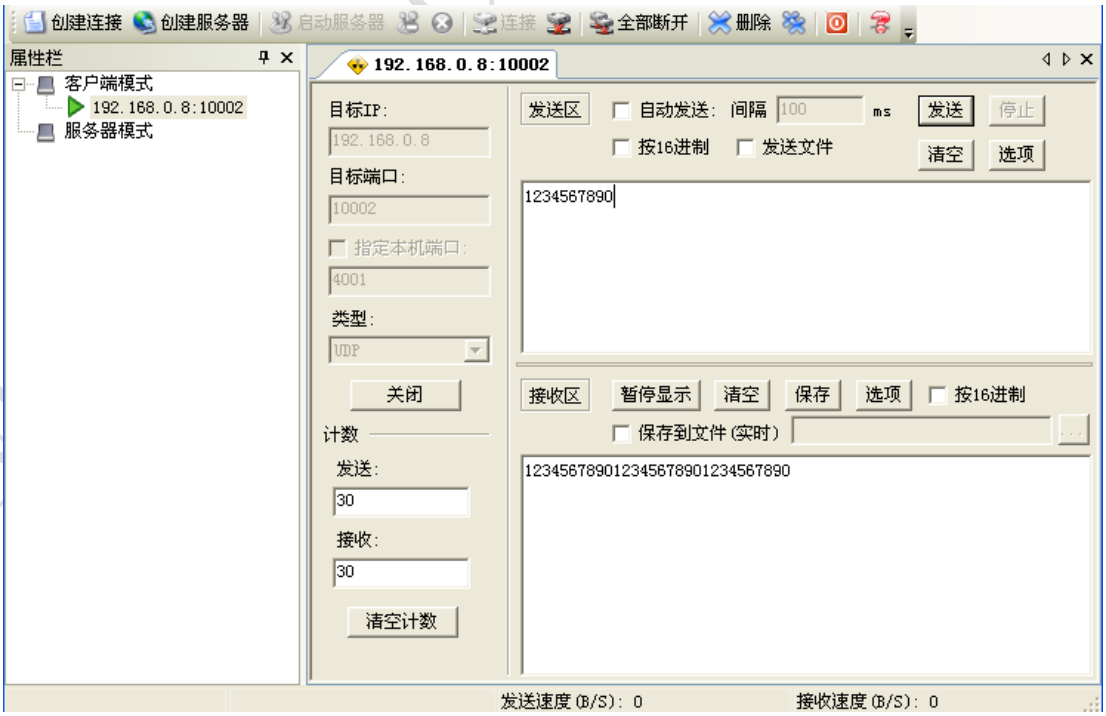


图 11 收发环路

5.1.4 UDPClient 测试范例

UDPClient 功能完成 UDP 通路中发送 UDP 数据帧端的角色，目前代码中提供了简单的 UDPClient 范例代码，函数名称为 TestUdpClient。该示例代码中示范了一个 UDP 数据通路中的发送操作。开启该测试功能需在 app_cfg.h 中打开#define TEST_UDP_CLIENT 选项。

下面对代码的具体实现以及使用做个简单介绍：

首先创建一个 socket，根据服务器端信息，如果当时连接失败会间隔二十秒之后再次发起连接，直到与对端连接成功。调用 LWIP 中 send 函数来实现数据发送。

用户在使用时需要对以下参数根据自己现有的网络情况进行配置。

表 12 UDPClient 配置

配置选项	默认设置	说明
UDP_CLIENT_PORT	10003	对端 UDPServer（UDP 帧接收端）端口，用户可自行修改，只需与 UDPServer（UDP 帧接收端）保持一致即可。
UDP_SERVER_IP_ADR	0xC0A80002（192.168.0.2）	对端 UDPServer（UDP 帧接收端） IP 地址
APP_DATA_LEN	2048	每次发送时可发送的最大包，用户可自行修改
SendData[APP_DATA_LEN]	随机数	该数组为发送数据缓存数组，为 UDPClient 发送数据的来源，用户可自行修改代码选择数据源。

测试时对端 Client 可选用 Iperf 作为测试工具，进行打流操作。

具体命令为：

```
iperf -s -p （Server 端端口号）-u -i 1
```

例如：

```
iperf -s -p 10003 -u -i 1
```

5.2 音频接口测试范例

NL6621 芯片支持 I2s 和 Pwm 音频输出接口，本 SDK 提供了音频接口的测试代码，能够演示循环播放本地音频数据。

I2S 接口测试代码集成于 I2S BSP 驱动内，默认不开启。演示时需要在 BSP_Init 中对 I2S 接口进行初始化，并在应用程序任务中调用 BSP_I2SStart 接口启动音频播放功能。涉及的编译选项说明如下：

表 13 音频接口测试范例编译选项

编译选项	说 明
TEST_APP_SUPPORT	测试代码总开关（app_cfg.h）
HW_I2S_SUPPORT	I2S 测试代码开启（i2s.h）
USE_I2S_DMA	I2S 接口测试时采用 DMA 搬移音频数据（i2s.h）
USE_I2S_DMA_INT	I2S 接口测试时 DMA 采用中断方式（i2s.h）
USE_I2S_16_BIT	I2S 接口可选择采样位宽为 16bit 或 24bit，此选项为设置采样位宽为 16bit（i2s.h），若不设置则为 24bit。
USE_PWM	采用 PWM 输出音频数据（i2s.h）

注：PWM 音频输出是 NL6621 在 I2S 接口的基础上增加的可选输出方式，PWM 输出方式下的操作与 I2S 输出方式大致相同，只有部分寄存器配置的差异，测试代码中以编译选项 USE_PWM 加以区分。

5.2.1 寄存器简介

I2S 接口的操作涉及到以下芯片寄存器，相关寄存器细节请参考 NL6621 datasheet I2S 接口章节。

表 14 I2s 接口相关寄存器

寄存器名称	偏移地址	说明
I2S_CTRL	6'b000000	I2S Control Register
I2S_STAT	6'b000100	I2S Status Register
I2S_SRR	6'b001000	I2S Channels Sample Rate & Resolution Configuration Register
CID_CTRL	6'b001100	Clock, Interrupt and DMA Control Register
TFIFO_STAT	6'b010000	Transmit FIFO Status Register
RFIFO_STAT	6'b010100	Receive FIFO Status Register

TFIFO_CTRL	6'b011000	Transmit FIFO Control Register
RFIFO_CTRL	6'b011100	Receive FIFO Control Register
DEV_CONF	6'b100000	Device Configuration Register

5.2.2 相关函数介绍

- BSP_I2SInit

函数说明：

完成 I2S 接口的初始化操作。

1) 配置 I2S_CTRL 寄存器为 0，复位所有 I2S channel、TFIFO、RFIFO。
2) 配置 I2S_SRR 寄存器，控制采样频率和采样 bit 位宽；
   - 采样频率由 NL6621 芯片 datasheet 中函数计算 eva_i2s_cal_sample_rate(int fclk, int asf)，其中 fclk 为时钟频率，asf 为采样频率

     目前 fclk 默认为 120MHz，以输出 44.1KHz 采样率为例：

     120MHz/44.1kHz/32bit/2 = 0x2B;
   - 采样位宽可设置为 16bit 或 24bit，由 USE_I2S_16_BIT 配置选项确定。
3) 配置 I2S_TFIFO_CTRL，I2S_RFIFO_CTRL 寄存器，设置 almost full 和 almost empty 的门限，如 8，24；

函数原型：

```
void BSP_I2SInit(void) ;
```

- BSP_I2SStart

函数说明：

提供 I2S 接口的使能操作，在使 用 PWM 接口时配置 PWM 寄存器，使用 DMA 时调用 DMA 相关接口启动 DMA。

1) 配置 I2S_CID_CTRL，使能 underrun 与 overrun 中断；
2) 配置 I2S_CTRL 寄存器，选择发送通路，主从配置，正常模式；
3) 如采用 PWM 作为输出方式，配置 I2S_PWM_CONF 寄存器，配置 PWM 使能、设置 PWM 采样率，选择 PWM 通道。
4) 如采用 I2S 作为输出方式，配置 DEV_CONF 寄存器，选择数据传输时物理信号的传输模式，代码

中提供了三种模式配置可供选择，用户可根据实际应用场景进行配置。

表 15　I2s 接口帧格式说明

I2S 传输模式	说 明
TRAN_REC_I2S_MODE	I2S 数字音频输出格式为标准 I2S 模式
TRAN_REC_I2S_LEFT_MODE	I2S 数字音频输出格式为左对齐模式
TRAN_REC_I2S_RIGHT_MODE	I2S 数字音频输出格式为右对齐模式

5)　如采用 DMA 方式在此处调用 DMA 相关接口启动 DMA。

函数原型：

```
void BSP_I2SStart(void) ;
```

- BSP_I2SHalt

函数说明：

提供 I2S 接口的暂停操作，在使 用 PWM 接口时提供 PWM mute 操作，使用 DMA 时提供 DMA suspend 操作（可与 BSP_I2SStart 成对使用）。

函数原型：

```
void BSP_I2SHalt(void) ;
```

- BSP_I2SMute

函数说明：

提供 I2S 接口的静默操作，在使用 PWM 接口时提供 PWM mute 操作，多用于 I2S 接口无音频数据时的短时静默。（可与 BSP_I2SSound 成对使用）。

函数原型：

```
void BSP_I2SMute(void) ;
```

- BSP_I2SSound

函数说明：

提供 I2S 接口在短时静默之后的恢复操作，与 BSP_I2SMute 成对使用。

函数原型：

```
void BSP_I2SSound(void) ;
```

- BSP_I2SDeinit

函数说明：

提供 I2S 接口的卸载操作。

函数原型：

```
void BSP_I2SDeinit(void);
```

- BSP_I2SIntISR

函数说明：

I2S 接口中断处理函数。

在示例代码的 BSP_I2SStart 函数中配置 I2S_CID_CTRL，使能了 underrun 与 overrun 中断。当 I2S 接口出现以上两种异常状况时，该中断处理函数就会被执行，以对异常情况作出处理。此处采用全局变量 i2sinttimes 对 I2S 接口异常中断出现的次数进行了统计，出现该变量为非 0 一般表示目前的音频数据源不足以满足当前采样率下 I2S 接口的播放速度。

函数原型：

```
void BSP_I2SIntISR(void);
```

5.2.3 I2S 测试例程

要使用 I2S 接口测试例程需完成以下操作：

1、 调用 BSP_I2SInit 函数完成 I2S 初始化（示例代码中在 BSP_Init 函数中调用）；

2、 在应用程序任务中调用 BSP_I2SStart 函数使能 I2S 接口；

3、 开启 I2S 接口相关配置选项，SDK 代码中提供了最常用的采用 DMA 中断方式的 I2S 模式。

注：测试代码中定义了 8K byte 的本地音频数据（buf0 数组）用于演示音频播放功能，演示过程会对该数据进行循环播放，正常情况下会输出频繁的铃音。

5.3 DirectConfig 测试范例

本 SDK 包含了 DirectConfig 功能的测试程序，开启该测试功能需在 app_cfg.h 中打开#define TEST_DIRECT_CONFIG 选项。该测试程序需要与手机客户端软件 NLConfigTool 配合使用。

DirectConfig 测试程序实现的流程如下：

1） NL6621 设备启动 TestDirectConfig 过程，打印如下：

```
(9)Direct config start, waiting..../
```

2） 启动 NLConfigTool 软件（确保手机已经连接到 AP），选择直连模式添加，在页面中输入 WiFi 名称和密码后点击 OK

3） NL6621 设备成功获取配置参数，打印如下：

```
(1090)Direct config succeeded
get ssid: 000833d4, len = 7
0x0000 : 6e 75 66 72 6f 6e 74
get key: 000833fd, len = 8
0x0000 : 31 32 33 34 35 36 37 38
```

4） 使用获取的配置参数发起 WiFi 连接过程，打印如下：

```
(1091)Start wifi connection.
```

5） 成功连接到 AP 后，通过 UDP 广播通知 NLConfigTool 客户端程序，打印如下：

```
(1216)[DHCP] MY IP ADDRESS:192.168.0.3
(1216)SYS_EVT_LINK_UP
(1241)Respond with udp broadcast...-
(1743)Respond finished.
```

6） NLConfigTool 软件获得 NL6621 设备连接 WiFi 成功的消息

5.4 SoftApConfig 测试范例

SoftApConfig 功能是除 DirectConfig 之外的另一种 WiFi 参数配置方式，其工作原理如下：

1） 首先让 6621 设备建立一个默认的 SoftAp 网络。

2） 手机断开当前网络并加入 6621 设备建立的 SoftAp

3） 手机应用程序将 WiFi 配置参数发送给 6621 设备

4） 手机连接到之前的网络。

本 SDK 包含了 SoftApConfig 功能的测试程序，开启该测试功能需在 app_cfg.h 中打开#define TEST_SOFTAP_CONFIG 选项。该测试程序需要与手机客户端软件 NLConfigTool 配合使用，具体步骤如下：

1）6621 设备运行测试固件建立默认网络名称为“NL6621”的 SoftAp，等待获取配置信息

```
(231)USE STATIC IP ADDRESS:10.10.10.1 !!!
(231)SYS_EVT_LINK_UP
(332)provisioning...
```

2）手机连接到 6621 设备建立的 SoftAp

3）手机运行 NLConfigTool 软件，选择热点模式添加，输入 WiFi 名称和密码后点击 OK，如下图所示：

5） 6621 设备成功获得配置信息并使用该配置发起联网过程

```
(6520)ssid:test009
(6520)password:1234567890
```

```
(6520)SYS_EVT_LINK_DOWN
(7168)[DHCP] MY IP ADDRESS:192.168.99.111
(7168)SYS_EVT_LINK_UP
```

5.5 Sniffer 功能测试范例

本 SDK 包含了 Sniffer 功能的测试程序，开启该测试功能需在 app_cfg.h 中打开#define TEST_SNIFFER 选项。

Sniffer 测试程序实现的流程如下：

1） 调用 InfWiFiStart 接口启动 WiFi 功能

2） 发起扫描请求

3） 等待扫描完成

4） 调用接口 InfSnifferStart 依次对扫描结果列表中的 AP 所在的网络进行嗅探

5） 打印接收到的 802.11 数据包

5.6 Uart 固件更新测试范例

本 SDK 包含了通过 Uart 接口更新固件的测试程序，开启该测试功能需在 app_cfg.h 中打开#define TEST_UART_UPDATE_FW 选项。

该测试程序与 Tool\bootTool.exe 工具软件配合使用，待测试程序运行后打印出“NULK READY!”，即可在 boolTool.exe 软件界面上执行 **Uart Burn** 操作。

5.7 OTA 固件更新测试范例

本 SDK 包含了通过 WiFi 网络接口更新固件的测试程序，开启该测试功能需在 app_cfg.h 中打开#define TEST_OTA_UPDATE_FW 选项。

该测试程序需要与 Tool\ OTAUpdateFw.apk 手机程序配合使用，OTAUpdateFw.apk 程序界面如下：

测试流程如下：

1）让手机与 6621 被测设备接入同一网络。

2）在手机软件界面上选择好固件 bin 文件路径。

3）等待 6621 被测设备打印“OTA update ready”，点击固件升级按钮开始固件更新过程。

4）固件升级成功后手机软件会有相应提示，6621 被测设备同时会打印出“Firmware update success”。

5.8 微信 Airkiss 测试范例

本 SDK 包含了微信 Airkiss 功能的测试程序，开启该测试功能需在 app_cfg.h 中打开#define TEST_AIRKISS 选项。该测试程序需要与手机客户端软件 WechatAirKiss 配合使用。

Airkiss 测试程序实现的流程如下：

1） NL6621 设备启动 TestAirkiss 过程，打印如下：

```
(10)Test airkiss start
```

2）启动 WechatAirKiss 软件（确保手机已经连接到 AP），在页面中输入 WiFi 名称和密码后点击发送

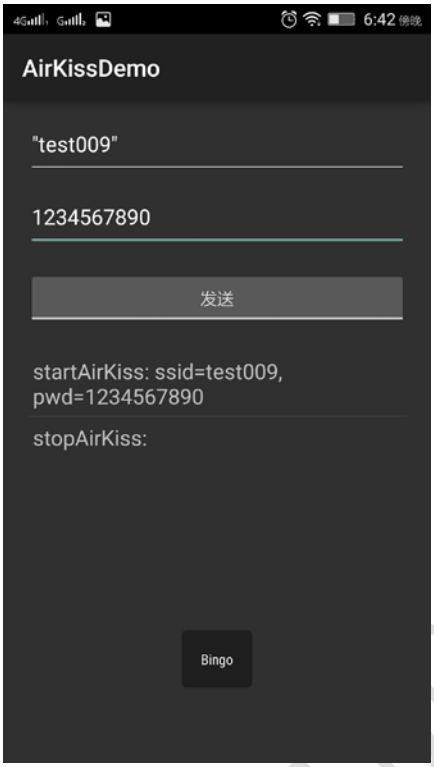
3） NL6621 设备成功获取配置参数，打印如下：

```
(10)Test airkiss start
(198)SYS_EVT_SCAN_DONE
(1186)SYS_EVT_SCAN_DONE
(2106)airkiss ok
```

4） 使用获取的配置参数发起 WiFi 连接过程，并通过 UDP 广播通知客户端程序，打印如下：

```
(2514)[DHCP] MY IP ADDRESS:192.168.99.128
(2514)SYS_EVT_LINK_UP
(2533)Respond with udp broadcast.-
(3034)Respond finished.
(3034)Test airkiss done
```

5） WechatAirKiss 软件获得 NL6621 设备连接 WiFi 成功的消息



附录 A 版本信息

版本	发布日期	说 明
V0.1	2013-9-30	创建
V1.0	2013-10-15	1）调整部分章节顺序 2）增加测试代码范例 3）增加 InfMacAddrSet 编程接口
V1.1	2013-11-16	1） 增加 I2S 音频接口应用说明 2） 增加 SDIO SPI 模式应用说明 3） 增加音频接口测试范例 4） 将 WiFi 功能接口分为参数配置接口和 WiFi 控制接口 5） InfEncModeSet 接口增加自动识别加密方式参数 6） InfScanStart 接口支持 AP 模式下的扫描功能，删除 InfScanResultGet 接口 7） 新增接口 InfPhyModeSet、InfTxRateSet、InfTxPwrLevelSet、InfPowerSaveSet、InfDeepSleepSet、InfPeerRssiGet、InfEfuseInfoGet 8） 增加发送增益校准说明（USE_NV_INFO）
V1.2	2014-02-17	1） 2.4.1 节增加校验认证码功能说明 2） 修改 2.4.2 节库文件说明 3） 修改 2.4.3 编译选项说明 4） 3.8 节增加 I2S 静音/取消静音功能说明 5） 修改 3.9 节 SDIO SPI 传输模式说明，增加测试步骤。 6） 新增接口 InfBeaconPeriodSet 7） 新增接口 InfListenIntervalSet 8） 修改接口 InfPowerSaveSet 参数 9） 增加 3.10 节休眠功能应用 10）增加 3.11 节 DirectConfig 功能应用 11）新增接口 InfDirectCfgStart
V1.3	2014-03-19	1） 删除 InfWscSet，新增接口 InfWscEnSet 2） 新增接口 InfWPSStart 3） 新增接口 InfWPSStop

		4） 新增接口 InfStaWpsPinGet 5） 更新接口 InfScanStart 说明，扫描结果数量最大值改为 32 6） 修改 2.4.3 节，说明 PrjSdkRam 工程的优化选项以减小固件镜像文件大小。 7） 新增接口 InfPeerAgeOutSet 8） 新增接口 InfPeerKickOut 9） 新增接口 InfProtectModeSet 10）新增接口 InfWmmEnableSet
V1.4	2014-09-05	1） 修改 3.11 节，增加 DirectConfig 手机客户端程序说明 2） 增加 5.3 节 DirectConfig 测试范例 3） 新增接口 InfSnifferStart 4） 增加 5.4 节 Sniffer 功能测试范例
V1.5	2014-10-30	1） 增加 5.5 节 Uart 固件更新测试范例 2） 增加 5.6 节 OTA 固件更新测试范例 3） 修改 2.4.1 节，增加 DOT11_N_SUPPORT 配置选项说明
V1.6	2015-3-13	1） 新增接口 InfCurChGet 2） 新增接口 InfVendorIESet 3） 修改 2.4.3 节，增加 PrjSdkOsIpRom 工程编译优化说明 4） 修改 DirectConfig 测试范例 5） 增加 SoftApConfig 测试范例
V1.7	2015-5-11	1） 增加 3.4 节 Flash 空间说明 2） 增加 5.8 节 Airkiss 测试范例