NearLink_DK_WS63 星闪开发板

使用说明书 V1.0

更改记录

版本	日期	作者	审核者	备注
V1.0	2024-07-15	赵鹏飞		

目录

1. 介绍

1.1. 产品概述

型号：NearLink_DK_WS63E

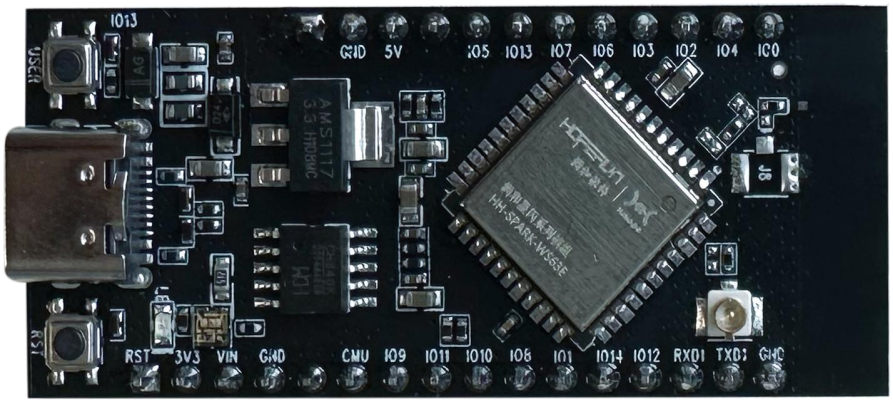


图 1-1 NearLink_DK_WS63E 星闪开发板

NearLink_DK_WS63E采用海思星闪WS63E的解决方案，具备对802.11b/g/n/ax无线通信协议的支持，同时兼容BLE5.3协议，具备BLE Mesh和BLE网关能力；支持SLE1.0协议及SLE网关功能；具备雷达人体活动检测能力；可基于OpenHarmony轻量系统开发物联网场景功能，是物联网智能终端领域的理想选择。

1.2. 产品特点

NearLink_DK_WS63E具有以下特点：

- **稳定、可靠的通信能力**

✧ 支持复杂环境下 TPC、自动速率、弱干扰免疫等可靠性通信算法

- **灵活的组网能力**

✧ 支持 BLE Mesh 组网

✧ 支持 Wi-Fi、BLE 或 SLE 三种组网方式

- **完善的网络支持**

✧ 支持 IPv4/IPv6 网络功能

✧ 支持 DHCPv4/DHCPv6 Client/Server

✧ 支持 DNS Client 功能

✧ 支持 mDNS 功能

✧ 支持 CoAP/MQTT/HTTP/JSON 基础组件

- **强大的安全引擎**

✧ 硬件实现 AES128/256 加解密算法

✧ 硬件实现 HASH-SHA256、HMAC_SHA256 算法

✧ 硬件实现 RSA、ECC 签名校验算法

✧ 硬件实现真随机数生成，满足 FIPS140-2 随机测试标准

✧ 硬件支持 TLS/DTLS 加速

✧ 硬件支持国密算法 SM2、SM3、SM4

✧ 内部集成 EFUSE，支持安全存储、安全启动、硬件 ID

✧ 内部集成 MPU 特性，支持内存隔离特性

- **开放的操作系统**

✧ 开放操作系统 OpenHarmony，提供开放、高效、安全的系统开发、运行环境

✧ 丰富的低功耗、小内存、高稳定性、高实时性机制

✧ 灵活的协议支撑和扩展能力

✧ 二次开发接口

✧ 多层级开发接口：操作系统适配接口和系统诊断接口、链路层接口、网络层接口

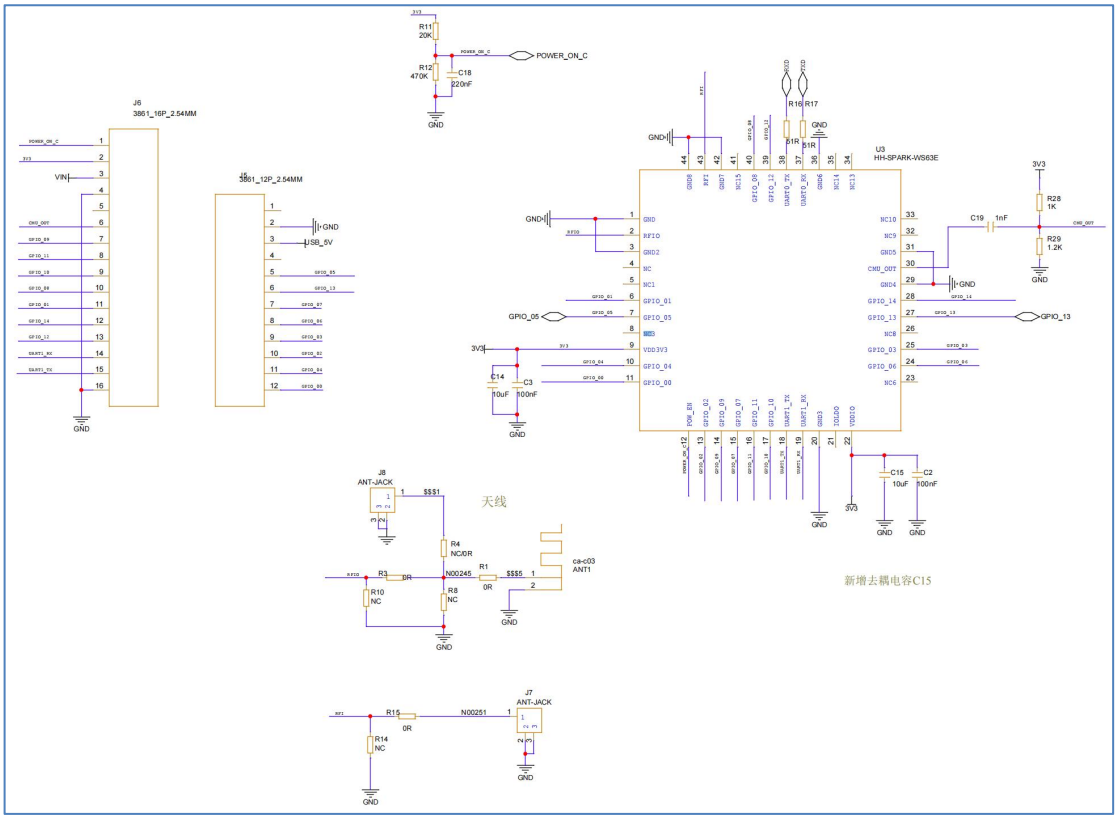
1.3. 主要规格

表1-1 NearLink_DK_WS63E 星闪开发板主要规格

模块	规格描述
CPU 子系统	● 高性能 32bit 微处理器，最大工作频率 240MHz ● 内嵌 SRAM 606KB、ROM 300KB ● 内嵌 4MB Flash
外围接口	● 1 个 SPI 接口、1 个 QSPI 接口、2 个 I2C 接口、1 个 I2S 接口、3 个 UART 接口、19 个 GPIO 接口、6 路 ADC 输入、8 路 PWM（注：上述接口通过复用实现） ● 外部晶体时钟频率 24MHz、40MHz
Software	● Wi-Fi 模式 STA, Soft-AP and sniffer modes ● 安全机制 WPS / WEP / WPA / WPA2 / WPA3 ● 加密类型 UART Download ● 软件开发 SDK ● 网络协议 IPv4, TCP/UDP/HTTP/FTP/MQTT
Wi-Fi	● 1×1 2.4GHz 频段（ch1～ch14） ● PHY支持 IEEE 802.11b/g/n/ax MAC 支持 IEEE 802.11d/e/i/k/v/w ● 支持 802.11n 20MHz/40MHz 频宽，支持 802.11ax 20MHz频宽 ● 支持最大速率：150Mbps@HT40 MCS7， 114.7Mbps@HE20 MCS9 ● 内置 PA 和 LNA，集成 TX/RX Switch、Balun 等 ● 支持 STA 和 AP 形态，作为 AP 时最大支持 6 个 STA 接入 ● 支持 A-MPDU、A-MSDU ● 支持 Block-ACK ● 支持 QoS，满足不同业务服务质量需求 ● 支持 WPA/WPA2/WPA3 personal、WPS2.0 ● 支持 RF 自校准方案 ● 支持 STBC 和 LDPC ● 支持雷达感知功能
蓝牙	● 低功耗蓝牙 Bluetooth Low Energy（BLE） ● 支持 BLE 4.0/4.1/4.2/5.0/5.1/5.2 ● 支持 125Kbps、500Kbps、1Mbps、2Mbps 速率 ● 支持多路广播 ● 支持 Class 1 ● 支持高功率 20dBm ● 支持 BLE Mesh，支持 BLE 网关
星闪	● 星闪低功耗接入技术 Sparklink Low Energy（SLE） ● 支持 SLE 1.0 ● 支持 SLE 1MHz/2MHz/4MHz，最大空口速率 12Mbps ● 支持 Polar 信道编码 ● 支持 SLE 网关
其他信息	● 电源电压输入：典型值5V ● 工作温度：-40℃～+85℃

2. 硬件

2.1. 原理图



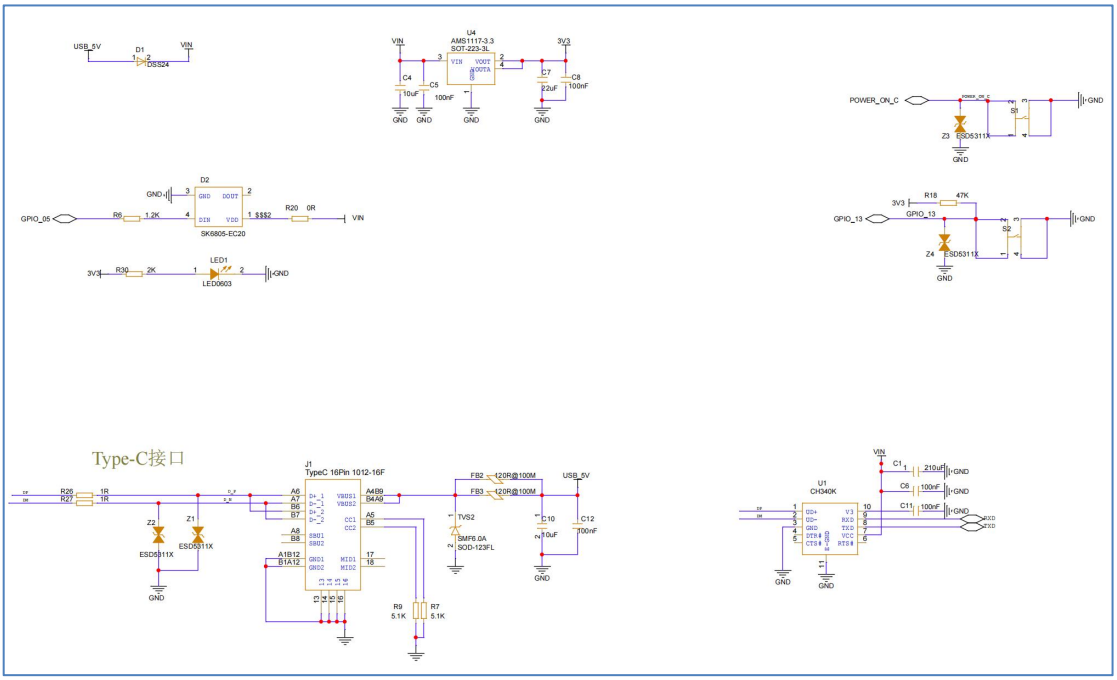


图 2-1　NearLink_DK_WS63E 星闪开发板原理图

2.2. pinout

图 2-2 NearLink_DK_WS63E 星闪开发板接口

表 2-1 NearLink_DK_WS63E 星闪开发板接口

序号	名称	类型	功能
1	NC		未连接
2	GND	PWR	电源地
3	5V	PWR	电源，5V
4	NC		未连接
5	GPIO05	IO	GPIO05，SSI_DATA，SPI1_IO2，UART2_CTS，PWM5,DFT_JTAG_TCK
6	GPIO13	IO	GPIO13，UART_CTS，DFT_JTAG_TDO，JTAG_TMS
7	GPIO07	IO	GPIO07，PWM7，UART2_RXD，SPI0_SCK，I2S_MCLK，ADC0
8	GPIO06	IO	GPIO06，PWM6，UART2_RTS，SPI1_SCK，DFT_JTAG_TDI，SPI0_OUT
9	GPIO03	IO	GPIO03，PWM3，SPI1_IO1
10	GPIO02	IO	GPIO02，PWM2，SPI_IO3
11	GPIO04	IO	GPIO4，SSI_CLK，PWM4，SPI1_IO1，DFT_JTAG_TMS，JTAG_ENABLE
12	GPIO00	IO	GPIO00，PWM0，SPI1_CSN，JTAG_TDI
13	GND	PWR	电源地
14	TXD1	IO	UART1_TXD，GPIO15，I2C1_SDA
15	RXD1	IO	UART1_RXD，GPIO16，I2C1_SCL
16	GPIO12	IO	GPIO12，PWM4，I2S_DI，ADC5
17	GPIO14	IO	GPIO14，DFT_JTAG_TRSTN，UART1_RTS
18	GPIO01	IO	GPIO01，PWM1，SPI1_IO0，JTAG_MODE
19	GPIO08	IO	GPIO08，PWM0，UART2_TXD，SPI0_CS1_N，ADC1
20	GPIO10	IO	GPIO10，PWM2，SPI0_CS0_N，I2S_SCLK，ADC3
21	GPIO11	IO	GPIO11，PWM3，SPI0_IN，I2S_LRCLK，ADC4
22	GPIO09	IO	GPIO09，PWM1，SPI0_OUT，I2S_DO，JTAG_TDO，ADC2
23	NC		未连接
24	NC		未连接
25	GND	PWR	电源地
26	VIN	PWR	电源，5V
27	3V3	PWR	电源，3.3V
28	RST	IO	主芯片的 RESET 信号

2.3. 尺寸图

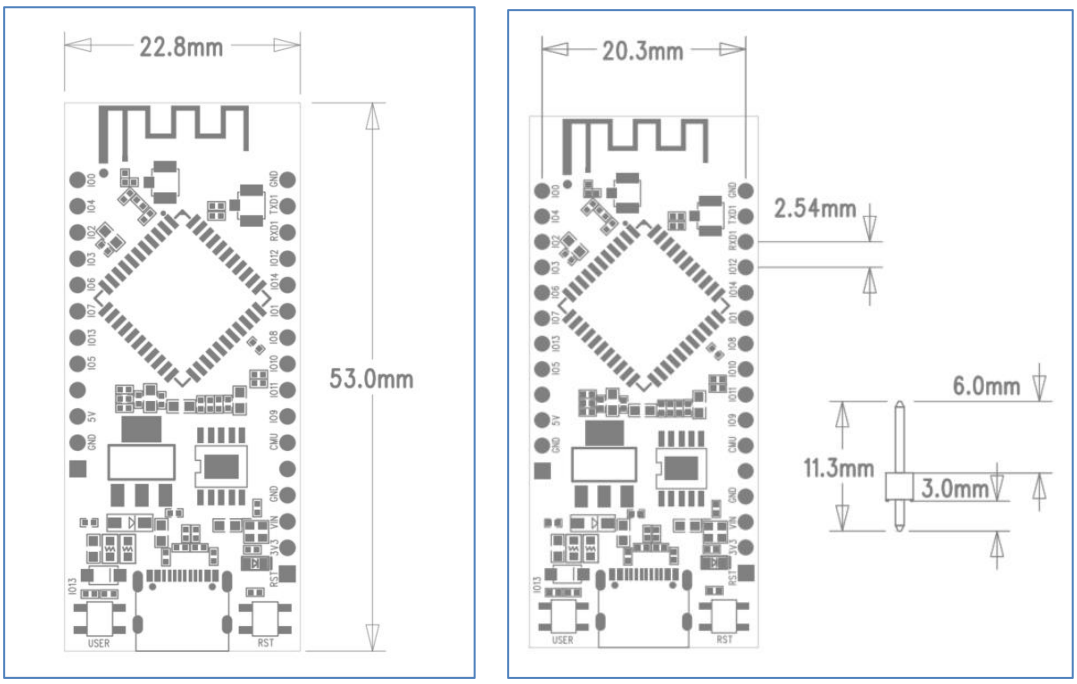


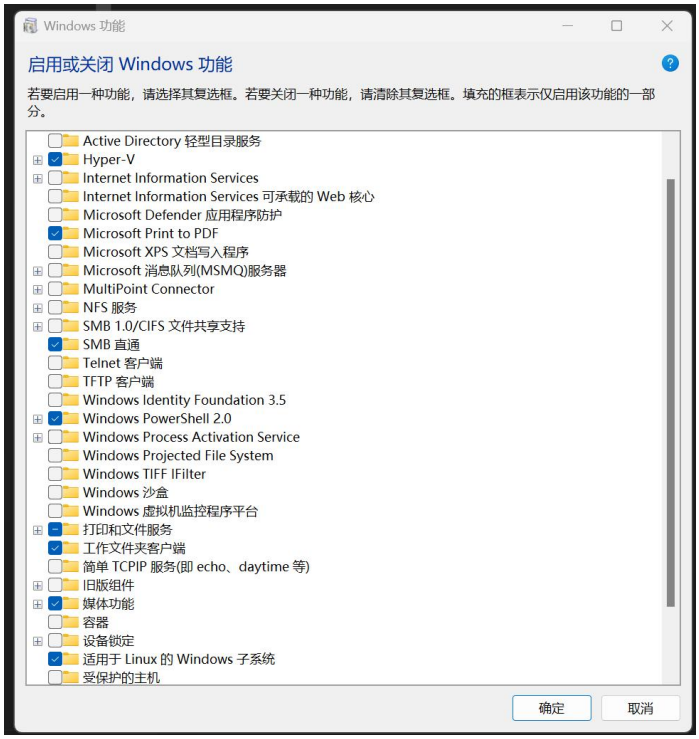
图 2-3 NearLink_DK_WS63E 星闪开发板及排针尺寸

3. 软件

3.1. WSL-OpenHarmony开发环境搭建

3.1.1. 系统设置

勾选Hyper-V和适用于Linux的Windows子系统



3.1.2. 安装WSL2

（1）打开PowerShell elevated(管理员权限)

（2）输入 wsl --set-default-version 2

（3）输入 wsl --install -d Ubuntu-22.04

（4）若遇到无法解析服务器的名称或地址错误出现,则按如下步骤进行，否则跳过。

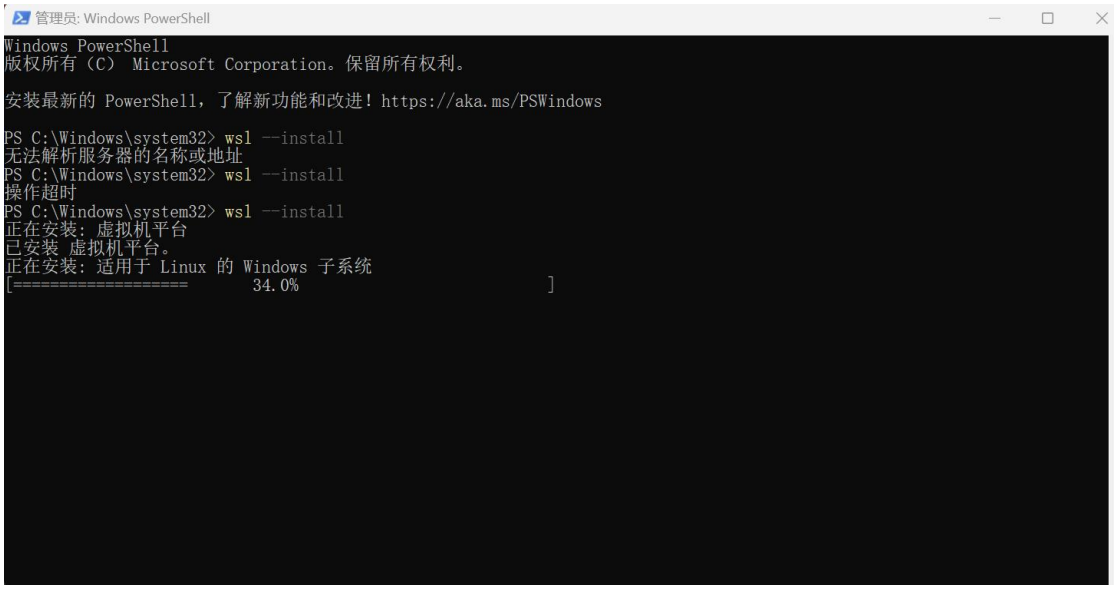
① 打开C:\Windows\System32\drivers\etc,复制hosts文件至桌面并以记事本打开

② 在打开的hosts文件中添加如下内容

185.199.110.133 raw.githubusercontent.com

③ 保存并关闭记事本，将保存的hosts文件复制回C:\Windows\System32\drivers\etc目录下替换原hosts文件

④ 重新打开PowerShell elevated(管理员权限)，输入wsl --install -d Ubuntu-22.04,等待安装完成。



（5）安装完成,重启电脑,设置用户名和密码

（6）输入wsl --shutdown关闭WSL。

（7）输入wsl --export Ubuntu-22.04 D:\Ubuntu-22.04.tar.gz 导出Ubuntu-22.04至D盘根目录下。

（8）输入wsl --unregister Ubuntu-22.04 卸载Ubuntu-22.04

（9）输入wsl --import Ubuntu-22.04 D:\Ubuntu-22.04 D:\Ubuntu-22.04.tar.gz导入Ubuntu-22.04至D:\Ubuntu-22.04目录下，该目录按需修改

（10）输入ubuntu2204 config --default-user 用户名修改用户名,此处用户名为上述步骤设置的用户名,否则为root用户

3.2. 安装OpenHarmony开发环境

（1）终端输入ubuntu2204进入Ubuntu-22.04

（2）设置Ubuntu-22.04为国内镜像源

sudo vim /etc/apt/sources.list修改为如下内容

```
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-updates main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-backports main restricted universe multiverse

# 以下安全更新软件源包含了官方源与镜像站配置，如有需要可自行修改注释切换
deb http://security.ubuntu.com/ubuntu/ jammy-security main restricted universe multiverse
# deb-src http://security.ubuntu.com/ubuntu/ jammy-security main restricted universe multiverse
```

（3）输入sudo apt update && sudo apt upgrade -y更新系统包

（4）修改Python错误/home/yoiannis/.local/lib/python3.10/site-packages/prompt_toolkit/styles/from_dict.py

```
- from collections import Mapping
+ from collections.abc import Mapping

from .base import Style, DEFAULT_ATTRS, ANSI_COLOR_NAMES
from .defaults import DEFAULT_STYLE_EXTENSIONS
from .utils import merge_attrs, split_token_in_parts
from six.moves import range
```

```
sudo python -m pip install -i https://pypi.tuna.tsinghua.edu.cn/simple --upgrade pip
sudo python -m pip install --upgrade pip
sudo pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

（6）安装库和工具集

```
sudo apt-get update && sudo apt-get install binutils binutils-dev git git-lfs gnupg flex bison gperf build-essential zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z1-dev ccache libgl1-mesa-dev libxml2-utils xsltproc unzip m4 bc gnutls-bin python3-pip ruby genext2fs device-tree-compiler make libffi-dev e2fsprogs pkg-config perl openssl libssl-dev libelf-dev libdwarf-dev u-boot-tools mtd-utils cpio doxygen liblz4-tool openjdk-8-jre gcc g++ texinfo dosfstools mtools default-jre default-jdk libncurses5 apt-utils wget scons tar rsync git-core libxml2-dev lib32z-dev grsync xxd libglib2.0-dev libpixman-1-dev kmod jfsutils reiserfsprogs xfsprogs squashfs-tools pcmciautils quota ppp libtinfo-dev libtinfo5 libncurses5-dev libncursesw5 libstdc++6 gcc-arm-none-eabi vim ssh locales libxinerama-dev libxcursor-dev libxrandr-dev libxi-dev
```

3.3. 星闪模组特有开发环境

（1）输入python3 -m pip install scons安装scons

（2）输入pip3 install setuptools安装setuptools

（3）输入pip3 install pycryptodome安装pycryptodome

（4）输入pip3 install six --upgrade --ignore-installed six安装six

（5）输入pip3 install ecdsa安装ecdsa

（6）输入pip3 install pycparser安装pycparser

3.4. 获取星闪模组SDK,设置编译工具链

（1） 解压sdk tar -zvf oh_sdk_XXXXXX.tar.gz, 切换到sdk目录下 cd sdk_temp

（2） 构建工具下载 ./build/prebuilts_download.sh

（3） 安装编译工具 sudo apt install cmake

(4) 安装编译工具 python3 -m pip install --user build/hb

(5) 设置编译工具链环境变量,当前源码所在路径可用pwd命令查看

```
yoiannis@DESKTOP-I6HONSH:~/OpenHarmony/sdk_temp$ ls
applications  base    build.py  commonlibrary  device   drivers     ide        kernel          out         productdefine  third_party
arkcompiler   build   build.sh  developtools   domains  foundation  interface  napi_generator  prebuilts   test           vendor
yoiannis@DESKTOP-I6HONSH:~/OpenHarmony/sdk_temp$ pwd
/home/yoiannis/OpenHarmony/sdk_temp
yoiannis@DESKTOP-I6HONSH:~/OpenHarmony/sdk_temp$
```

vim ~/.bashrc
export PATH=/{当前源码所在路径}/device/soc/hisilicon/ws63v100/sdkv100/tools/bin/compiler/riscv/cc_riscv32_musl_b090/cc_riscv32_musl/bin:$PATH
export PATH=~/.local/bin:$PATH
source ~/.bashrc

```
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
export PATH=/home/yoiannis/OpenHarmony/sdk_temp/device/soc/hisilicon/ws63v100/sdkv100/tools/bin/compiler/riscv/cc_riscv32_musl_b090/cc_riscv32_musl/bin:$PATH
export PATH=~/.local/bin:$PATH
-- INSERT (paste) --                                                                    119,14        Bot
```

(6) 执行 riscv32-linux-musl-gcc -v检查编译工具链是否配置成功,配置成功显示如下。

```
yoiannis@DESKTOP-I6HONSH:~/OpenHarmony/sdk_temp$ riscv32-linux-musl-gcc -v
Using built-in specs.
COLLECT_GCC=riscv32-linux-musl-gcc
COLLECT_LTO_WRAPPER=/home/yoiannis/OpenHarmony/sdk_temp/device/soc/hisilicon/ws63v100/sdkv100/tools/bin/compiler/riscv/cc_riscv32_musl_b090/cc_riscv32_musl/bin/../libexec/gcc/riscv32-linux-musl/7.3.0/lto-wrapper
Target: riscv32-linux-musl
Configured with: /devcloud/workspace/cc_source/build/cc_riscv32_musl/../../open_source/cc_riscv32_musl_build_src/gcc/configure --build=x86_64-pc-linux-gnu --host=x86_64-pc-linux-gnu --target=riscv32-linux-musl --with-arch=rv32imc --prefix=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/cc_riscv32_musl --disable-multilib --disable-__cxa_atexit --with-gnu-as --with-gnu-ld --disable-libmudflap --enable-libgomp --enable-libssp --disable-threads --enable-shared --with-abi=ilp32 --enable-poison-system-directories --enable-symvers=gnu --with-pkgversion='build ver100.090 2023-05-17' --enable-languages=c,c++ --with-headers=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/cc_riscv32_musl/sysroot/usr/include --with-sysroot=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/cc_riscv32_musl/sysroot --with-build-sysroot=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/cc_riscv32_musl/sysroot --with-gmp=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/obj/host-libs/usr --with-mpfr=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/obj/host-libs/usr --with-mpc=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/obj/host-libs/usr --with-isl=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/obj/host-libs/usr --with-build-time-tools=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/cc_riscv32_musl/riscv32-linux-musl/bin --libdir=/devcloud/workspace/cc_source/build/cc_riscv32_musl/riscv32_elf_musl_build_dir/cc_riscv32_musl/lib --disable-libitm --with-system-zlib
Thread model: single
gcc version 7.3.0 (build ver100.090 2023-05-17)
```

3.5. 编译

(1) 在源码目录下执行hb set,选择mini,选择对应开发板

```
yoiannis@DESKTOP-I6HONSH:~/OpenHarmony/sdk_temp$ hb set
OHOS Which os_level do you need?  mini
OHOS Which product do you need?  (Use arrow keys)

hihope
 > nearlink_dk_3863
   neptune_iotlink_demo
   nearlink_dk_bs21
   nearlink_dk_3863_xts
```

(2) 执行hb build -f编译,等待编译成功

3.6. 安装Windows烧录调试工具

3.6.1. 安装串口终端工具和串口驱动程序

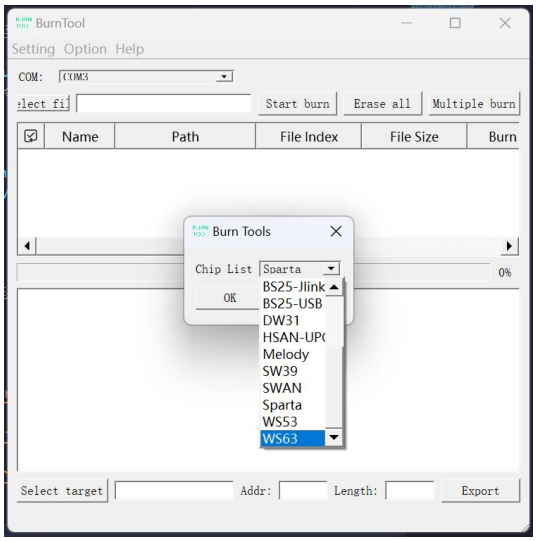
（1） 学校或家庭用户到xshell官网下载xshell,下载完成后双击安装即可。

（2） 下载CH340驱动,下载完成后双击安装即可。

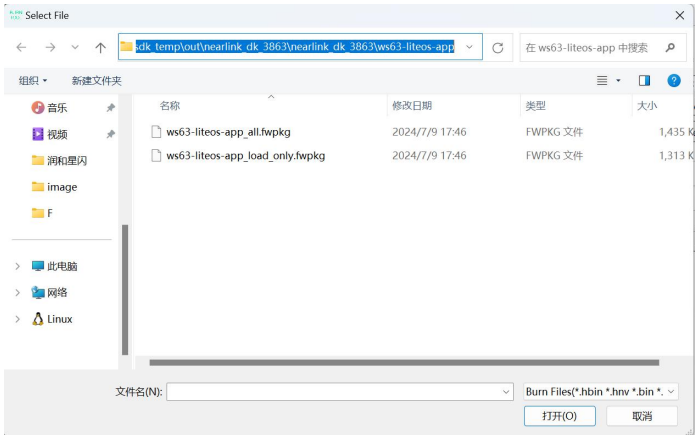
（3） 下载BurnTool烧录工具并解压。

3.6.2. 连接与烧录

（1）打开烧录工具，选择对应的串口,打开烧录工具,点开Option选项,选择对应的芯片,WS63E与WS63属于同一款系列，芯片选择WS63即可。

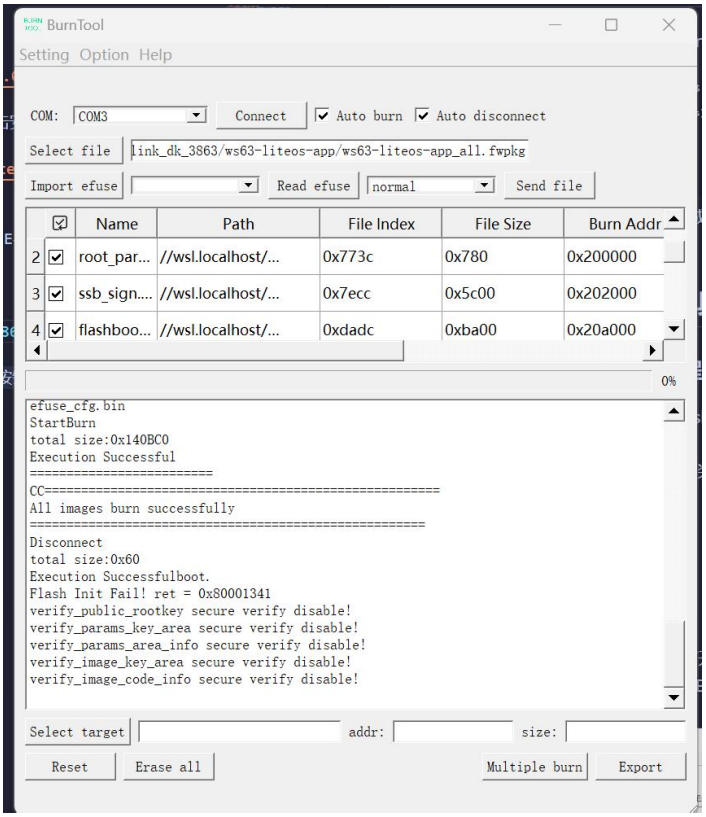


（2）选择烧录文件,示例路径为：\\wsl.localhost\Ubuntu-22.04\home\yoiannis\OpenHarmony\sdk_temp\out\nearlink_dk_3863\nearlink_dk_3863\ws63-liteos-app

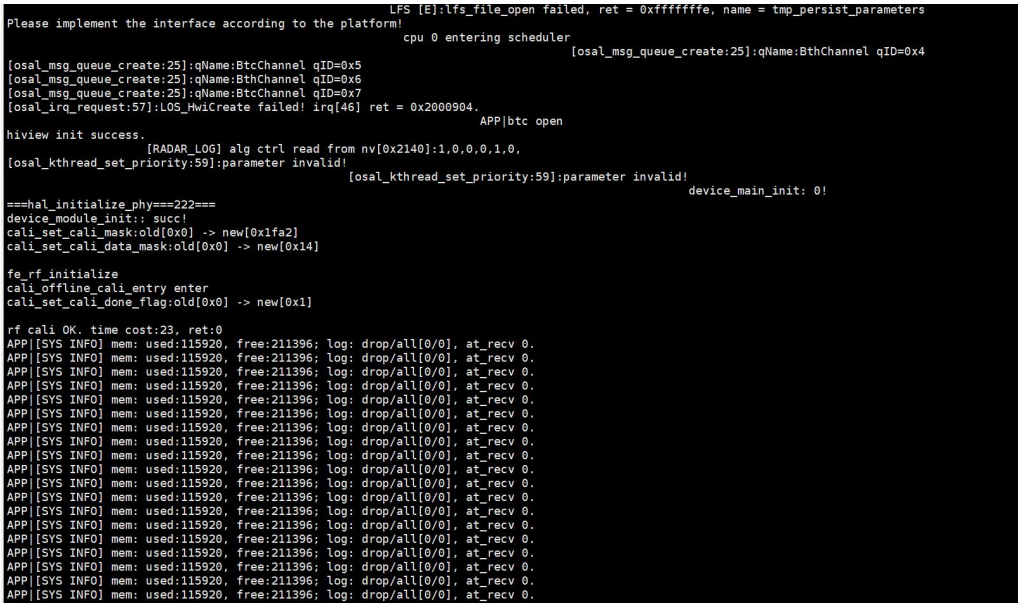


（3）勾选Auto Burn和Auto disconnect选项,点击connect连接,单按开发板RST按键开始烧录。

（4）烧录完成结果如下：



（4） 打开串口工具,波特率选择115200,上电后可以看到相关的串口打印。



3.7. HelloWorld工程创建

1. 在applications/sample/wifi-iot/app目录下新建hello_world文件夹

2. 在hello_world文件夹下新建hello_world.c和hello_world.h文件

```
/* hello_world.h */
/* 系统头文件 */
#include <stdio.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
/* 定义任务栈大小和优先级 */
#define TASK_STACK_SIZE         1024
#define TASK_PRIO               20
```

```
/* hello_world.c */
#include "hello_world.h"
/**
@brief 输出“Hello, World!”消息
该函数用于演示最基本的打印操作，向用户展示如何通过C程序打印出经典的
 “Hello, World!”
**/
static void *HelloWorldTask(const char *arg)
{
    (void)arg;
    while (1)
    {
        printf("[HelloWorldEntryExample] Task is running.....!\r\n");
        usleep(300000);
    }
    return NULL;
}
/**
 * @brief 创建并启动"Hello World"任务线程
 * 本函数通过CMSIS-OS API创建一个名为"HelloWorldTask"的任务线程，该线程负责打印"Hello World"消息。
 * 设置线程的属性，包括线程名、属性位、回调内存、回调大小、栈内存、栈大小和优先级。
 * 如果线程创建失败，将通过printf输出错误消息。
**/
static void HelloWorldEntry(void)
{
    osThreadAttr_t attr; /* 设置线程属性 */
    attr.name = "HelloWorldTask";           /**< 线程名称 */
    attr.attr_bits = 0U;                    /**< 线程属性位，此处设为0 */
    attr.cb_mem = NULL;                     /**< 线程控制块内存地址，此处设为NULL，由OS自动分配 */
    attr.cb_size = 0U;                      /**< 线程控制块大小，此处设为0，由OS自动确定 */
    attr.stack_mem = NULL;                  /**< 线程栈内存地址，此处设为NULL，由OS自动分配 */
    attr.stack_size = TASK_STACK_SIZE;  /**< 线程栈大小，单位通常为字节 */
    attr.priority = TASK_PRIO;              /**< 线程优先级 */

    /* 创建线程 */
    if (osThreadNew((osThreadFunc_t)HelloWorldTask, NULL, &attr) == NULL) {
        printf("[HelloWorldEntryExample] Failed to create HelloWorldEntry!\n");
    }
}
/* 系统初始化完成后，自动调用 */
SYS_RUN(HelloWorldEntry);
```


3. 在hello_world文件中新建BUILD.gn编译脚本文件

```
# 静态库构建规则：helloWorld
static_library("helloWorld") {
    # 源代码文件列表
    # 这里指定了构建静态库所需的源代码文件，这些文件将会被编译并链接到静态库中。
    sources = [
        "hello_world.c",
    ]

    # 头文件搜索路径
    # 这些路径指定了编译器在查找源文件中包含的头文件时需要搜索的目录。
    # 包含这些路径可以使编译器找到项目中或其他依赖库中定义的公共接口。
    include_dirs = [
        "//commonlibrary/utils_lite/include",
        "//kernel/liteos_m/kal/cmsis",
        "./",
    ]
}
```

4. 在applications/sample/wifi-iot/app/BUILD.gn文件中添加编译规则

```
import("//build/lite/config/component/lite_component.gni")

lite_component("app") {
    features = [
        "hello_world:helloWorld",  # 同级目录下 hello_world 文件夹下的 helloWorld 静态库
    ]
}
```

5. 将静态库文件中添加编译进系统

修改device\soc\hisilicon\ws63v100\sdkv100\libs_url\ws63\cmake\ohos.cmake文件添加 "helloWorld"静态库，如下：

```
elseif(${TARGET_COMMAND} MATCHES "ws63-liteos-app")
set(COMPONENT_LIST "begetutil"  "hilog_lite_static" "samgr_adapter" "bootstrap" "fsmanager_static"
"hal_update_static" "hilog_static" "inithook"  "samgr_source"
        "broadcast" "hal_file_static"  "init_log"  "native_file" "udidcomm"
        "cjson_static" "hal_sys_param" "hichainsdk" "hota" "init_utils"  "param_client_lite"
        "hiview_lite_static" "hal_sysparam" "hievent_lite_static" "huks_3.0_sdk"  "samgr" "blackbox_lite"
"hal_iothardware" "wifiservice"
        "hidumper_mini" "helloWorld")
endif()
```

修改device\soc\hisilicon\ws63v100\sdkv100\build\config\target_config\ws63\config.py文件在'ws63-liteos-app'中'ram_component': []'添加"helloWorld"静态库，如下：

```
'ws63-liteos-app': {
    'base_target_name': 'target_ws63_app_rom_template',
    'os': 'liteos',
    'defines': [
        ......
    ],
    'ram_component': [
        .......
        'xo_trim_port',
        "mqtt",
        "helloWorld" # 添加 helloWorld 静态库
    ],
    'ccflags': [
        "-DBOARD_ASIC", '-DPRE_ASIC',
    ],
    'application': 'application',
    'bin_name': 'ws63-liteos-app',
    'smaller': True,
    'hso_enable_bt': True,
    'hso_enable': True,
    'codesize_statistic': True,
    'nv_update':True,
    'generate_efuse_bin': True,
    'copy_files_to_interim': True
},
```

在SDK源码目录下执行hb build -f或python build.py wifiiot,等待编译完成,执行烧录

6. 烧录完成后，通过串口工具查看输出信息

```
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
APP|[SYS INFO] mem: used:73900, free:3635416; log: drop/all[0/0], at_recv 0.
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
[HelloWorldEntryExample] Task is running.....!
```

3.8. GPIO点灯测试

1. 引脚定义

主板GPIO	主板	显示板	NFC板	环境监测板	智能红绿灯板	炫彩灯板	可复用功能
GPIO07	-	-	-	-	RED(红灯)	-	-
GPIO010	-	-	-	-	YELLOW(黄灯)	-	-
GPIO011	-	-	-	-	GREEN(绿灯)	-	-

备注:

- 表格中未列出的引脚和功能可能包含NFC板、环境监测板、智能红绿灯板等其他功能板的信息。
- 特定引脚的具体编号和功能请参考实际硬件文档或技术规格。

2.添加GPIO驱动程序

API：

API名称	说明
unsigned int GpioInit(void);	GPIO模块初始化
unsigned int GpioSetDir(WifiIotGpioIdx id, WifiIotGpioDir dir);	设置GPIO引脚方向，id参数用于指定引脚，dir参数用于指定输入或输出
unsigned int GpioSetOutputVal(WifiIotGpioIdx id, WifiIotGpioValue val);	设置GPIO引脚的输出状态，id参数用于指定引脚，val参数用于指定高电平或低电平
unsigned int IoSetFunc(WifiIotIoName id, unsigned char val);	设置引脚功能，id参数用于指定引脚，val用于指定引脚功能
unsigned int GpioDeinit(void);	解除GPIO模块初始化

（1） 在applications/sample/wifi-iot/app目录下新建GPIO点灯工程

（2） 添加iot_gpio_ex.h文件

（3） 添加app_demo_led_control.cLED点灯文件

（4） 参考02.HelloWorld工程添加编译构建文件，将工程编译进系统,烧录查看效果

代码参考: [demo/led_demo · HiHopeORG/NearLink - 码云 - 开源中国 (gitee.com)]