



鸿蒙开发教程

蓝鹊派-BMP200

Hi3861



蓝鹊派

福州比特元科技有限公司



目录

第一章：环境搭建	4
一、开发环境简介:	4
二、Windows 环境搭建:	4
三、Ubuntu 环境搭建:	7
四、配置 Windows 远程访问 Ubuntu 环境:	18
第二章：创建源码工程	21
一、Windows 环境下按如下步骤配置：（可自行选择代码路径）	21
第三章：编译、烧录源码工程.....	23
一、Windows 环境下按如下步骤进行.....	23
第四章：任务管理	27
一、任务管理介绍	错误!未定义书签。
二、任务运行状态	33
三、任务的一些名词	33
四、API 的介绍	34
五、实验步骤	34
第五章：软件定时器	39
一、实验目的	39
二、软件定时器简介.....	错误!未定义书签。
三、软件定时器运行机制.....	39
四、API 的介绍	39
五、实验步骤	40
第六章：信号量	44
一、实验目的	44
二、信号量的介绍	错误!未定义书签。
三、信号量的使用场景.....	44
三、API 的介绍	44
四、实验步骤	45
第七章：事件管理	49
一、实验目的	49
二、实验原理	49
三、API 的介绍	50
四、实验步骤	51
第八章：互斥锁	54
一、实验目的	54
二、实验原理	54
三、API 的介绍	54
四、实验步骤	55
第九章：消息队列	59
一、实验目的	59
二、实验原理	59
三、API 的介绍	59



四、实验步骤	60
第十章：点亮第一个 LED 灯	64
一、实验目的	64
二、实验原理	64
三、API 的介绍	64
四、实验步骤	65
第十一章：PWM 的使用	68
一、实验目的	68
二、实验原理	68
三、API 的介绍	68
四、实验步骤	69
第十二章：ADC 的使用	72
一、实验目的	72
二、实验原理	72
三、API 的介绍	72
四、实验步骤	72
第十二章：NFC 标签 NT3H 的使用	76
一、实验目的	76
二、实验原理	76
三、API 的介绍	77
四、实验步骤	77
第十三章：智慧路灯	81
一、实验目的	81
二、实验原理	81
三、API 的介绍	82
四、实验步骤	83
第十四章：智慧烟感	86
一、实验目的	86
二、实验原理	86
三、API 的介绍	87
四、实验步骤	88
第十五章：智慧农业	91
一、实验目的	91
二、实验原理	91
三、API 的介绍	94
四、实验步骤	95

第一章：环境搭建

一、开发环境简介：

当前阶段，大部分的开发板源码还不支持在 Windows 环境下进行编译，因此，需要使用 Ubuntu 的编译环境对源码进行编译。所以，本教程采取 Windows+Ubuntu 混合开发环境。

注意：1、操作系统环境命名不能包含中文！！！

2、安装之前请关闭所有杀毒软件！！！

二、Windows 环境搭建：

1、在 Windows 操作系统下安装 DevEco Device Tool 工具。（选择 Windows 版本）

下载链接：<https://device.harmonyos.com/cn/develop/ide#download>

2、下载并且解压完后，双击安装包进行安装，自行选择安装路径。

说明：如果您已安装 DevEco Device Tool 3.0 Beta2 及以前的版本，则在安装新版本时，会先卸载旧版本，卸载过程中出现如下错误提示时，请点击“Ignore”继续安装，该错误不影响新版本的安装。（如图 1.1 所示）

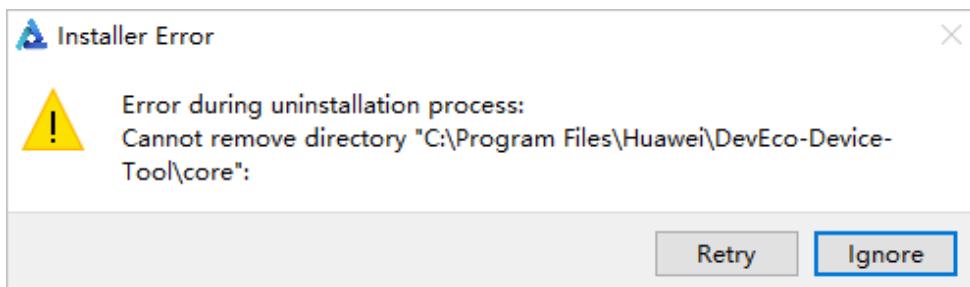


图 1.1

3、根据安装向导提示进行安装。

说明：在弹出 VSCode installation confirm 页面，勾选“Install VScode 1.62.2 automatically”，点击 Next。如果检测到 Visual Studio Code 已安装，且版本为 1.62 及以上，则会跳过该步骤。（如图 1.2 所示）

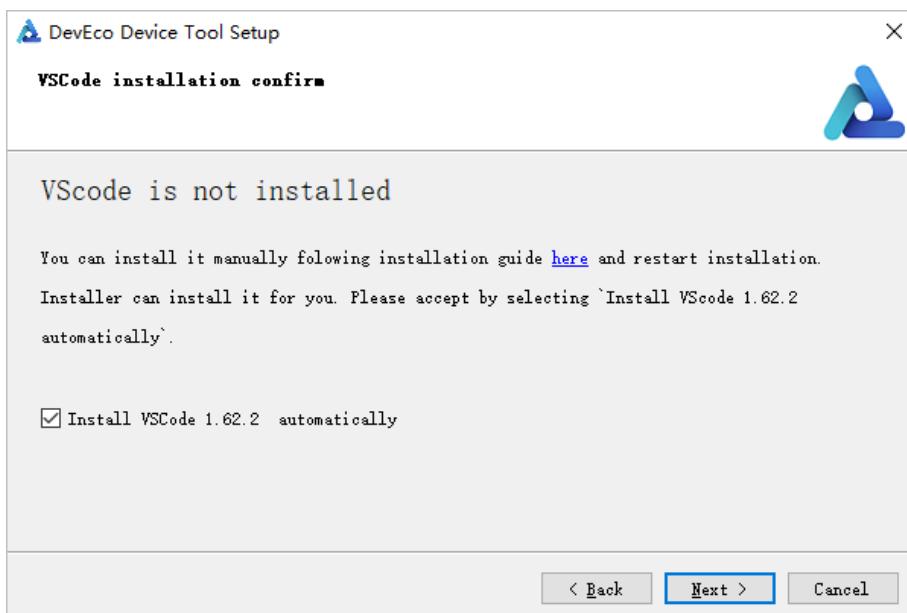


图 2.2



4、在弹出的 Python select page 选择“Download from Huawei mirror”，点击 Next。（如图 1.3 所示）

说明：如果系统已安装可兼容的 Python 版本（Python 3.8~3.9 版本），可选择“Use one of compatible on your PC”。（如果系统是别的版本，也选择 Download from Huawei mirror）

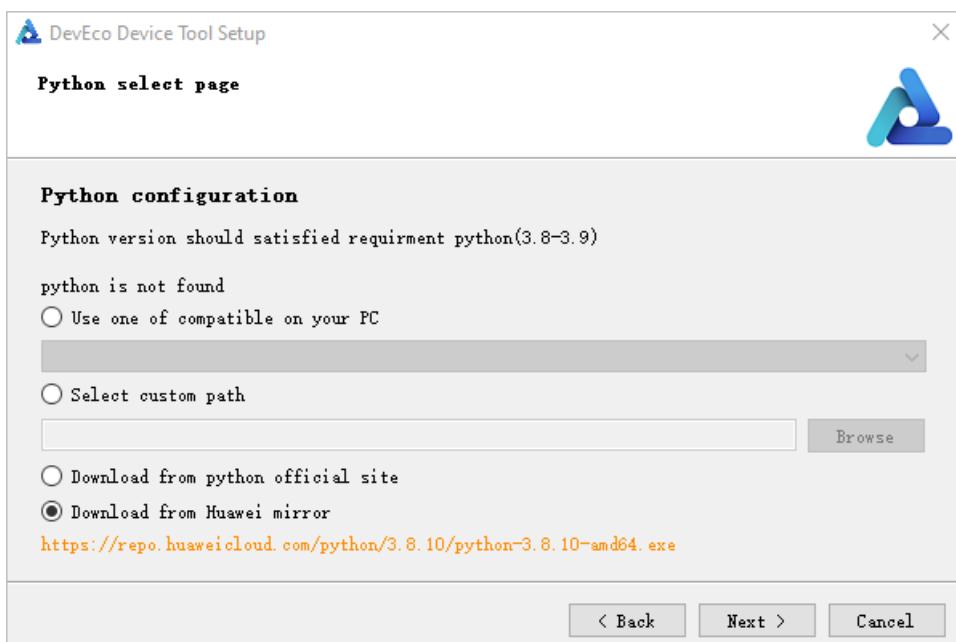


图 3.3

5、在以下界面点击 Next，进行软件下载和安装。（如图 1.4 所示）

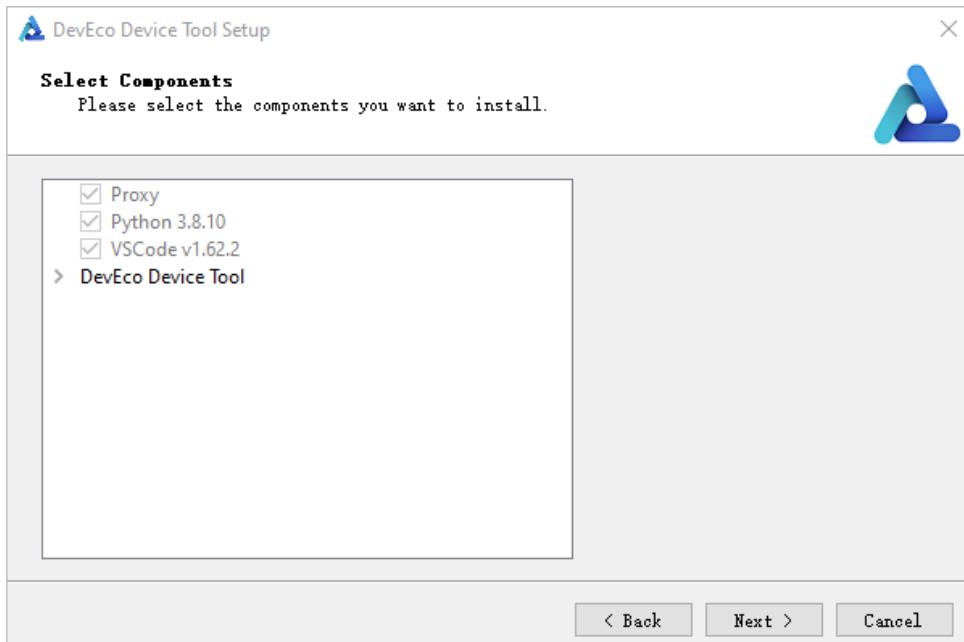


图 4.4



6、继续等待 DevEco Device Tool 安装向导自动安装 DevEco Device Tool 插件，直至安装完成，点击 Finish，关闭 DevEco Device Tool 安装向导。（如图 1.5 所示）



图 5.5

7、打开 Visual Studio Code，进入 DevEco Device Tool 工具界面。至此，DevEco Device Tool Windows 开发环境安装完成。（如图 1.6 所示）

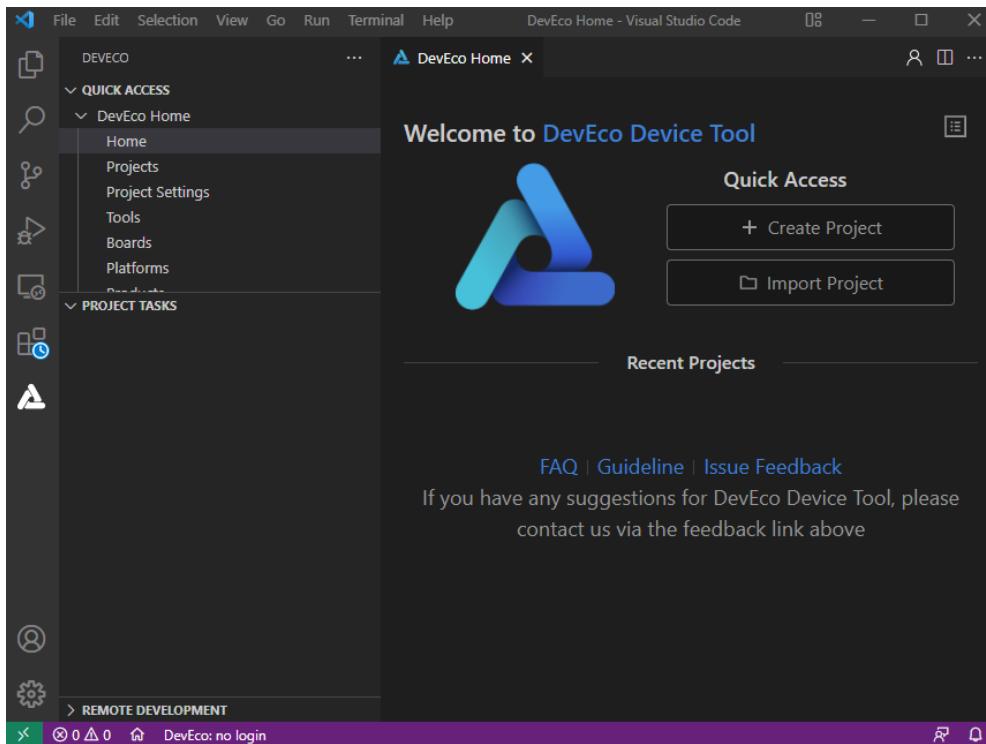


图 6.6

三、Ubuntu 环境搭建：

1. 虚拟机下载：官网自行下载虚拟机并安装，完成后再安装 Ubuntu 镜像。

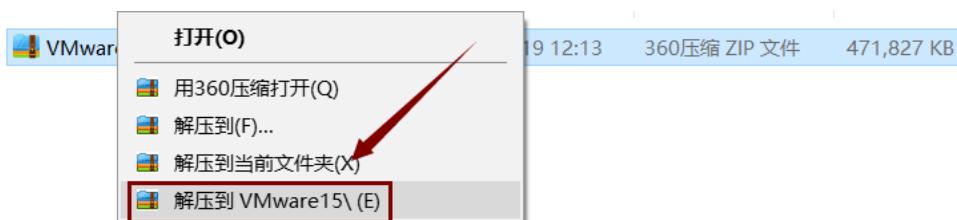
VMware15 下载链接：https://pan.baidu.com/share/init?surl=0vyqm_P-i0Dth9b_b-8EUw

提取码：9y9n

安装教程如下所示：

1. 鼠标右击软件压缩包，选择“解压到 VMware15”。

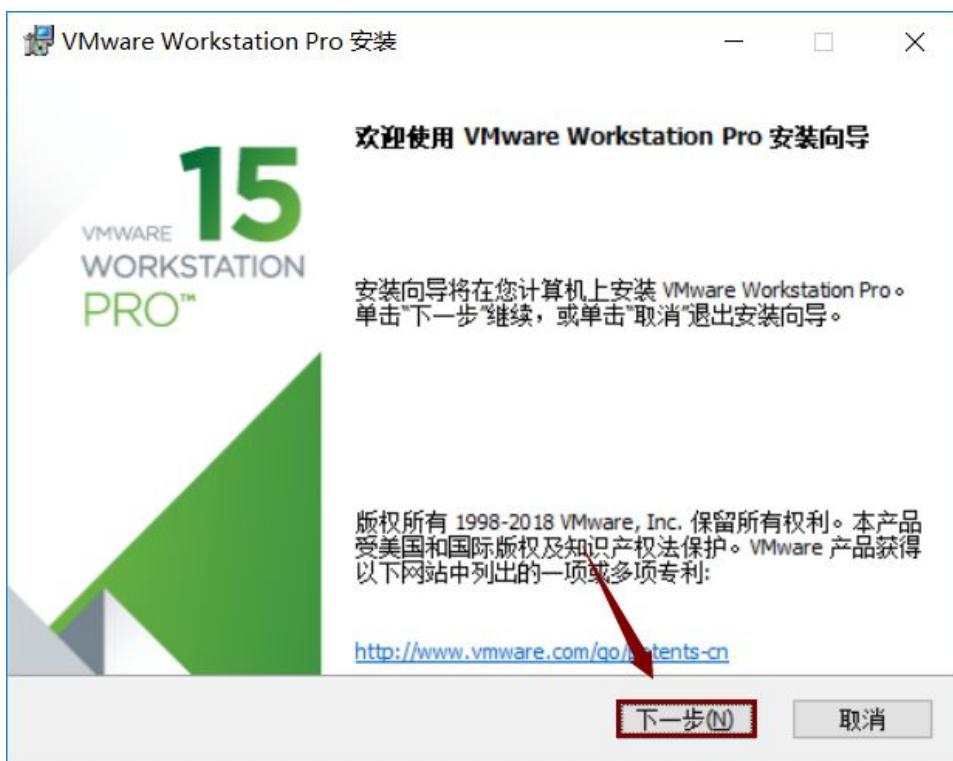
说明：解压密码为 123456



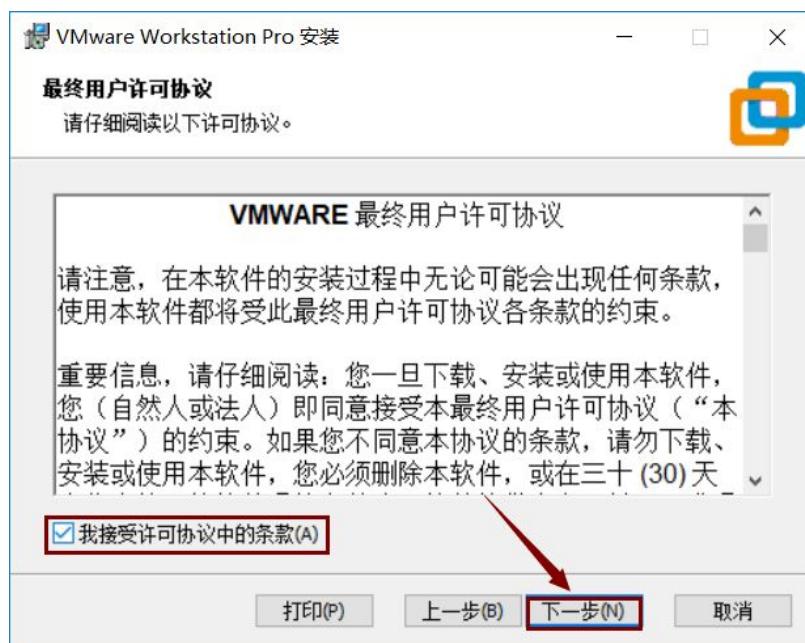
2. 打开解压后的文件夹，选中“VMware-15”可执行文件，鼠标右击选择“以管理员身份运行”。



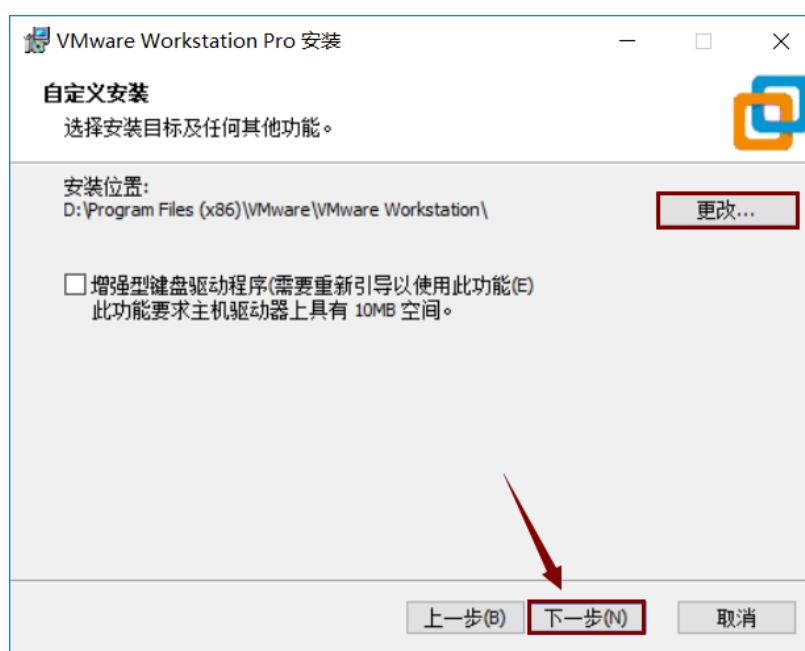
3. 点击“下一步”。

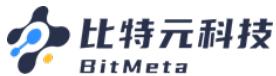


4. 勾选“我接受条款协议中的条款”，然后点击“下一步”。

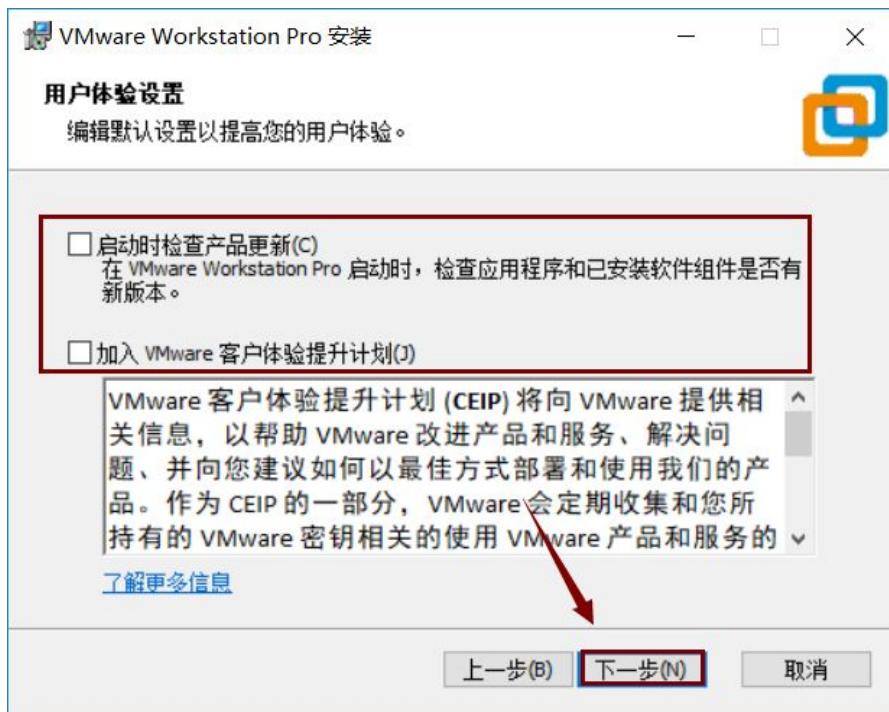


5. 点击“更改”更改软件的安装目录，建议安装在 C 盘之外的其他盘符，然后点击“下一步”。

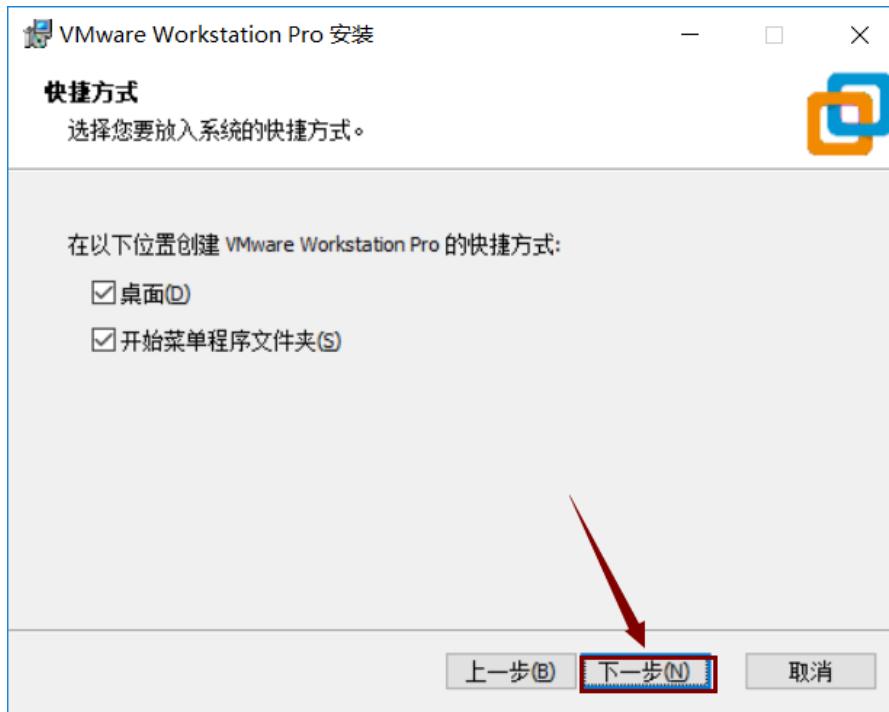




6. 取消勾选，然后点击“下一步”。

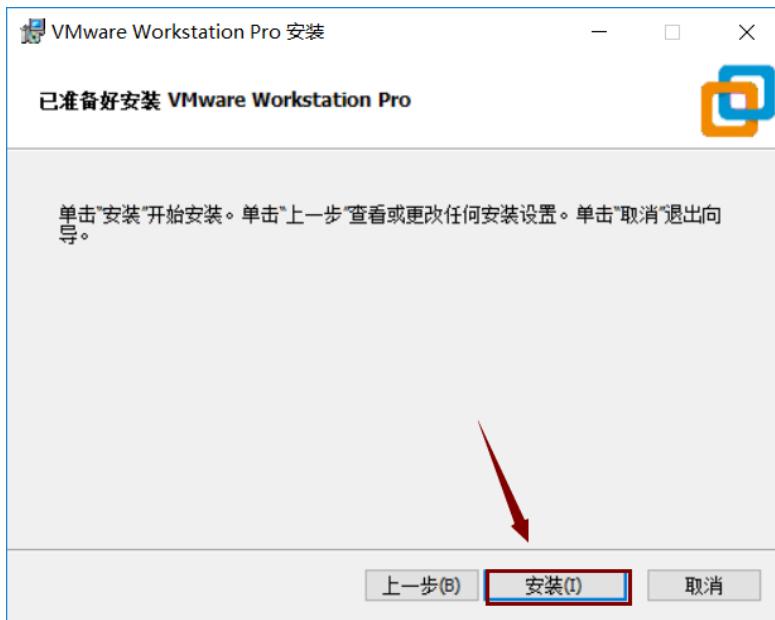


7. 点击“下一步”。



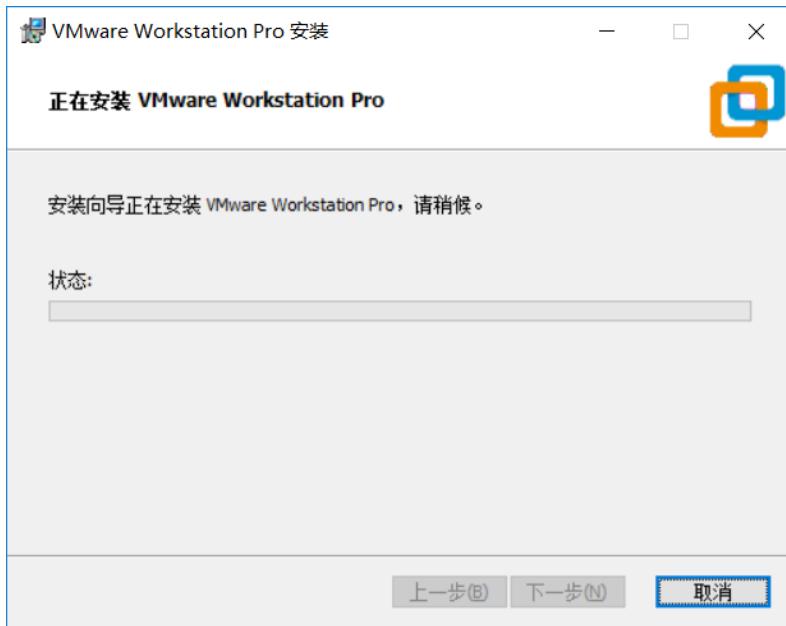


8. 点击“安装”。



9. 软件正在安装，请耐心等待，谢谢。

注意：千万注意下一步，一定要点击许可证！！！

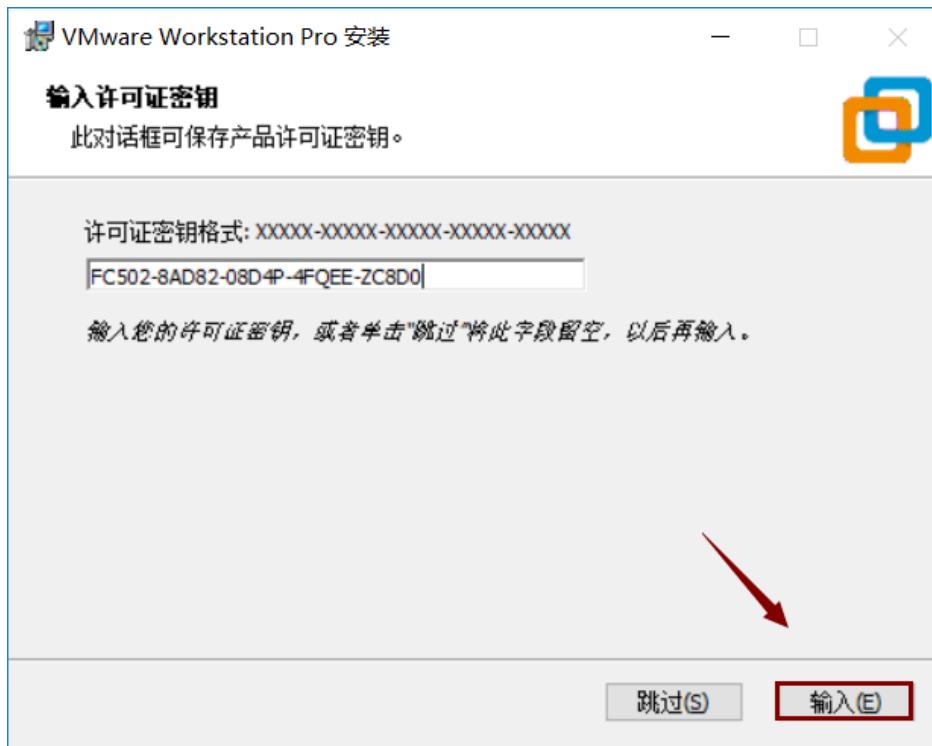


10. 点击“许可证”。

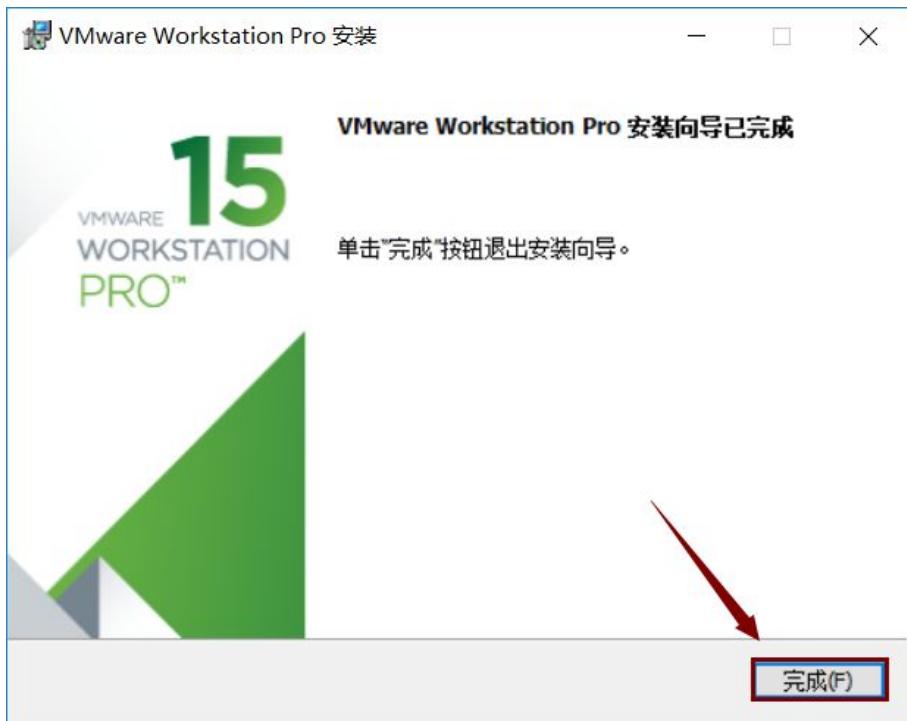
注意：一定要点击许可证！！！



11. 在框中输入“FC502-8AD82-08D4P-4FQEE-ZC8D0”然后点击“输入”。



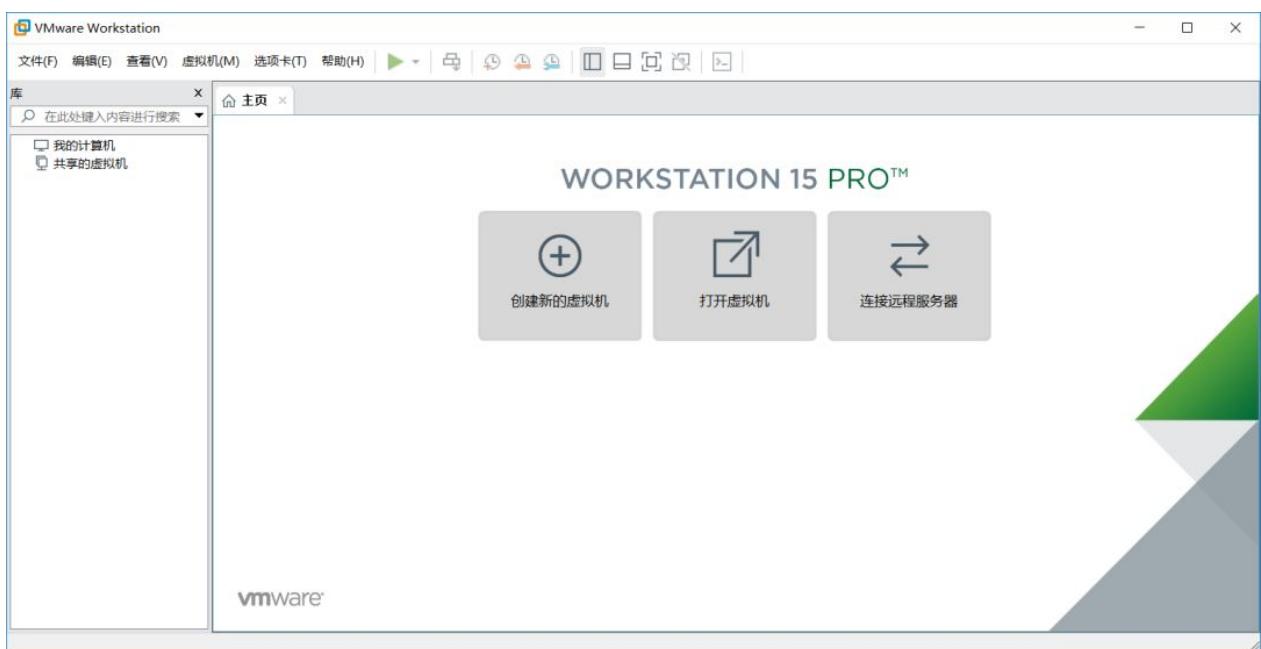
12. 点击“完成”。



13. 在桌面找到“VMware Workstation Pro”双击打开。



14. 安装完成。



2、Ubuntu 镜像如下

OpenHarmony3.0 镜像链接: https://pan.baidu.com/s/1RD8HAfFpVDOWudPWzQLT_A
提取码: 9131

注意: 1、网盘里的东西要全部都下载下来!!!

2、下面的账户密码就是之后要用到的。

Ubuntu 系统账号: hihope

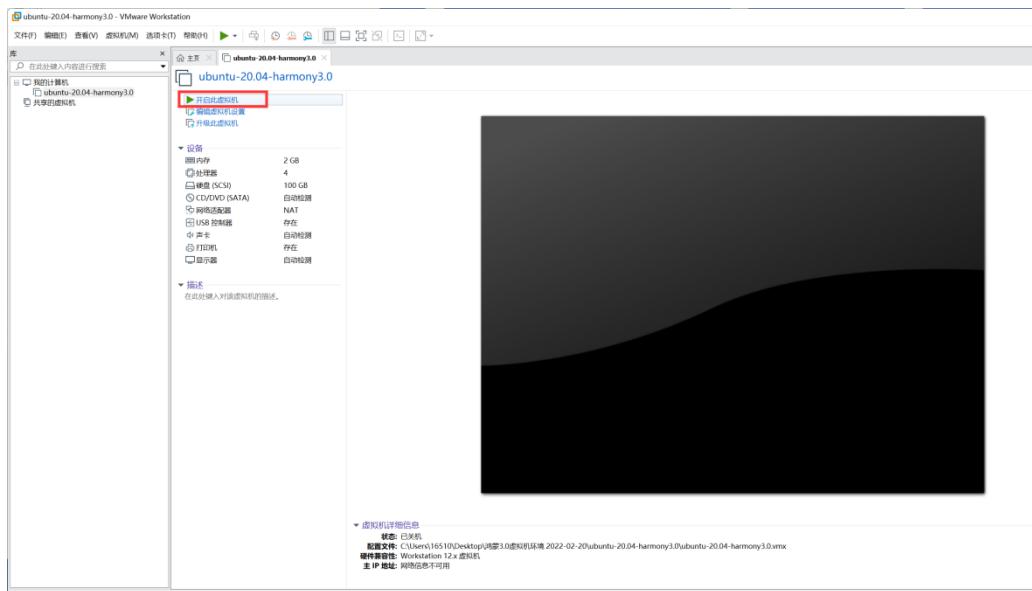
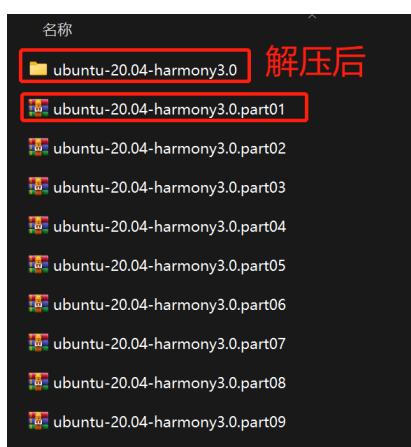
Ubuntu 系统密码: 123456

代码路径: ~/harmony (镜像自带的一份源码)

Tips1: 已经配置好环境, 下载好代码, 编译可通过。

Tips2: 已经下载好了 repo, 开发者只需要自己配置 git, 即可下载代码。

3、下载完后, 解压 part01, 得到下图文件夹。双击桌面虚拟机图标, 开启虚拟机后, 选择下图打开虚拟机, 选择刚刚解压后的文件夹里的鸿蒙虚拟机。







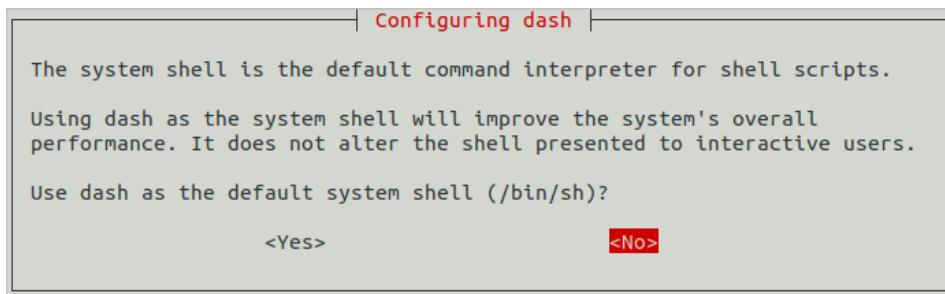
打开鸿蒙 3.0 虚拟机后，开始配置开发环境。打开终端（方法 1，快捷键：CTRL+T，方法 2，选择空白界面，右击鼠标，选择 Open terminal），执行如下命令，确认输出结果为 bash。如果输出结果不是 bash，请根据步骤 2，将 Ubuntu shell 修改为 bash。

命令：ls -l /bin/sh

```
hihope@hihope-virtual-machine:~$ ls -l /bin/sh  
lrwxrwxrwx 1 root root 4 2月 20 12:48 /bin/sh -> bash
```

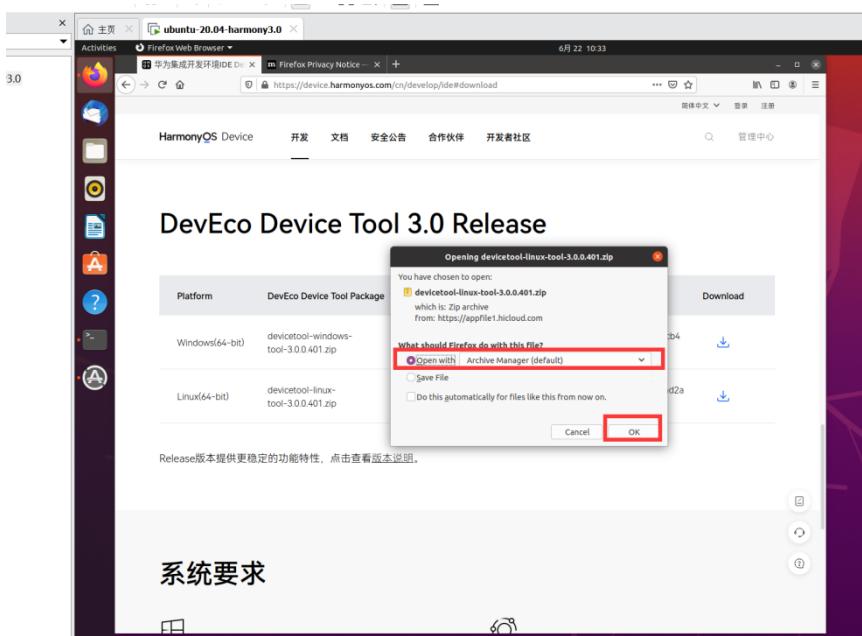
4、打开终端工具，执行如下命令，输入密码，然后选择 No，将 Ubuntu shell 由 dash 修改为 bash。执行后可再次输入上条命令进去确认。（如果已经是 bash，则跳过本步骤）

命令：sudo dpkg-reconfigure dash



5、在虚拟机中下载 DevEco Device Tool 3.0 Release （选择 Linux 版本）

链接：<https://device.harmonyos.com/cn/develop/ide#download>



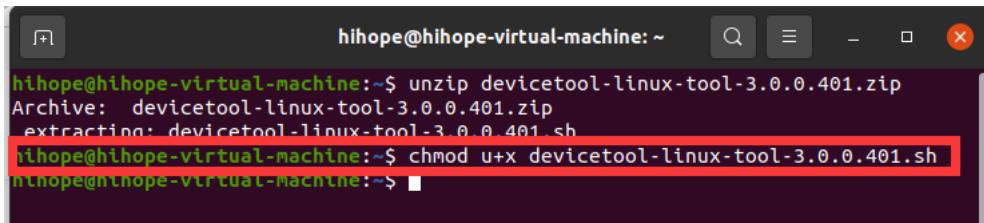
6、解压 DevEco Device Tool 软件包并对解压后的文件夹进行赋权。进入 DevEco Device Tool 软件包目录，执行如下命令解压软件包，其中 devicetool-linux-tool-3.0.0.401.zip 为软件包名称，请根据实际进行修改。

命令：unzip devicetool-linux-tool-3.0.0.401.zip

```
hihope@hihope-virtual-machine:~$ unzip devicetool-linux-tool-3.0.0.401.zip  
Archive: devicetool-linux-tool-3.0.0.401.zip  
extracting: devicetool-linux-tool-3.0.0.401.sh
```

7、进入解压后的文件夹，执行如下命令，赋予安装文件可执行权限，其中 devicetool-linux-tool-3.0.0.401.sh 请根据实际进行修改。

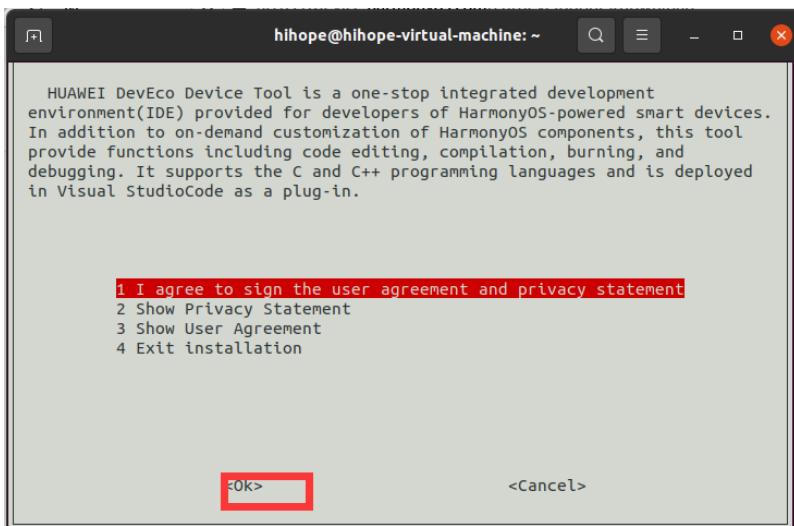
命令： chmod u+x devicetool-linux-tool-3.0.0.401.sh



```
hihope@hihope-virtual-machine:~$ unzip devicetool-linux-tool-3.0.0.401.zip
Archive: devicetool-linux-tool-3.0.0.401.zip
  extracting: devicetool-linux-tool-3.0.0.401.sh
hihope@hihope-virtual-machine:~$ chmod u+x devicetool-linux-tool-3.0.0.401.sh
hihope@hihope-virtual-machine:~$
```

8、执行如下命令，安装 DevEco Device Tool，其中 devicetool-linux-tool-3.0.0.401.sh 请根据实际进行修改。

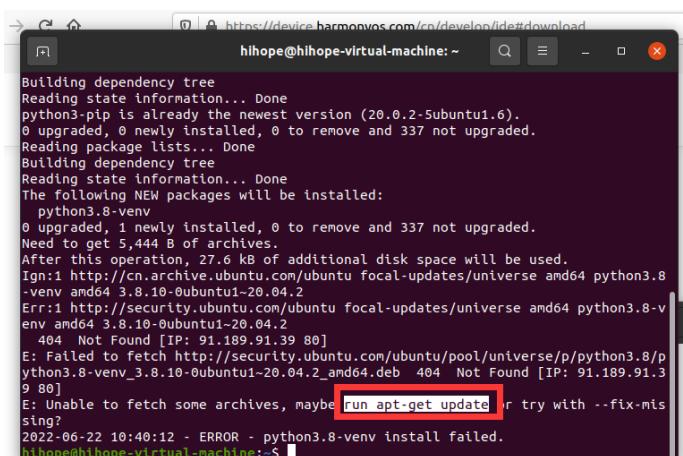
说明： 安装过程中，会自动检查 Python 是否安装，且要求 Python 为 3.8~3.9 版本。如果不满足，则安装过程中会自动安装，提示“Do you want to continue?”，请输入“Y”后继续安装。



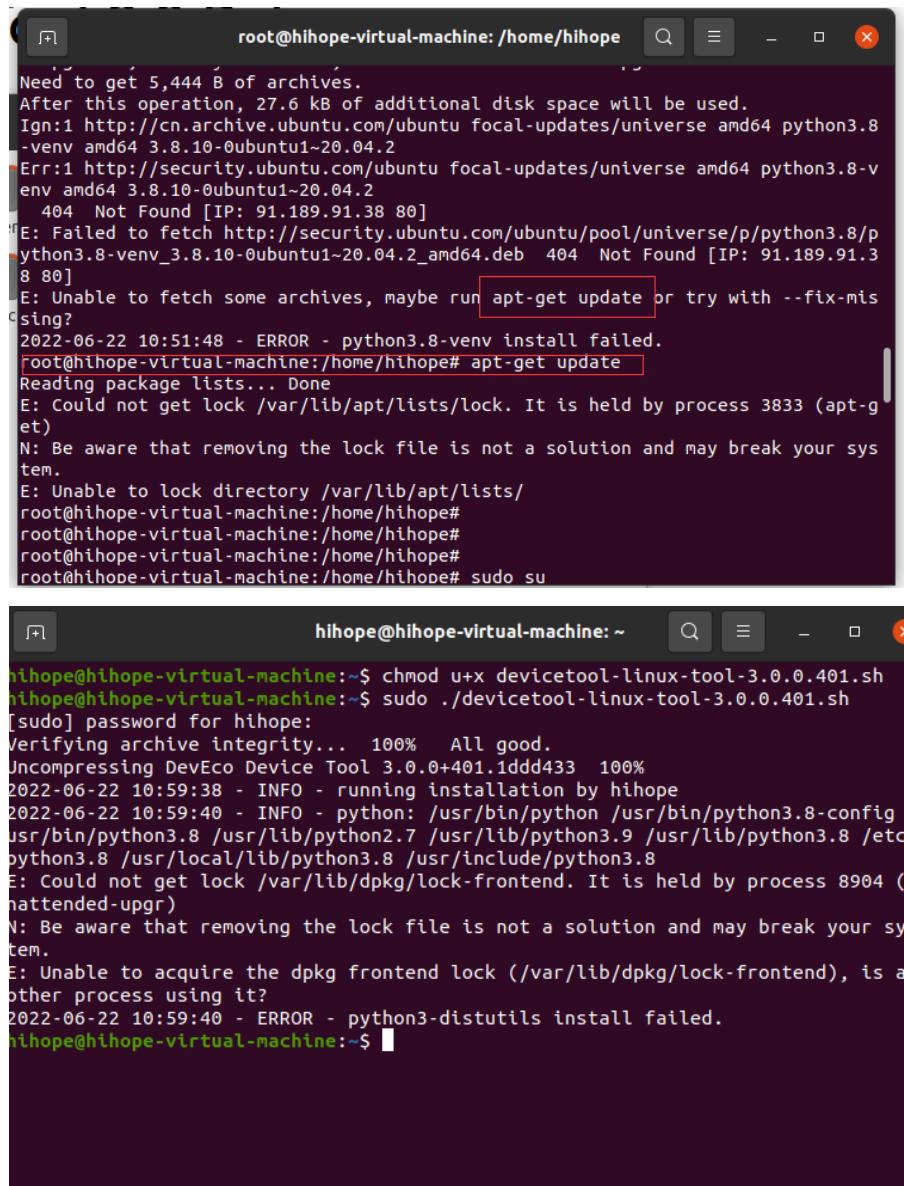
命令： sudo ./devicetool-linux-tool-3.0.0.401.sh

注意：可能出现的问题

问题 1：如第一张图片所示，我们按照提示执行 apt-get update，如果出现第二张图片，我们先执行命令： sudo su，再执行命令： sudo passwd，输入新密码也为 123456，再次执行 apt-get update。如果遇到第三张图片的问题，则重新启动 ubuntu 即可。再重复上述步骤 6 和步骤 7。



```
hihope@hihope-virtual-machine:~$ https://device.harmonysos.com/cn/developer/ide#download
Building dependency tree
Reading state information... Done
python3-pip is already the newest version (20.0.2-5ubuntu1.6).
0 upgraded, 0 newly installed, 0 to remove and 337 not upgraded.
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  python3.8-venv
0 upgraded, 1 newly installed, 0 to remove and 337 not upgraded.
Need to get 5,444 B of archives.
After this operation, 27.6 kB of additional disk space will be used.
Ign:1 http://cn.archive.ubuntu.com/ubuntu focal-updates/universe amd64 python3.8-venv all 3.8.10-0ubuntu1~20.04.2
Err:1 http://security.ubuntu.com/ubuntu focal-updates/universe amd64 python3.8-venv all 3.8.10-0ubuntu1~20.04.2
  404  Not Found [IP: 91.189.91.39 80]
E: Failed to fetch http://security.ubuntu.com/ubuntu/pool/universe/p/python3.8/python3.8-venv_3.8.10-0ubuntu1~20.04.2_amd64.deb 404  Not Found [IP: 91.189.91.39 80]
E: Unable to fetch some archives, maybe run apt-get update or try with --fix-missing?
2022-06-22 10:40:12 - ERROR - python3.8-venv install failed.
hihope@hihope-virtual-machine:~$
```



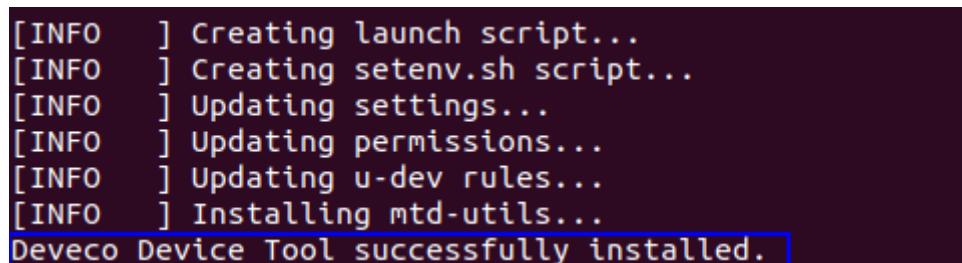
root@hihope-virtual-machine: /home/hihope

```
Need to get 5,444 B of archives.  
After this operation, 27.6 kB of additional disk space will be used.  
Ign:1 http://cn.archive.ubuntu.com/ubuntu focal-updates/universe amd64 python3.8  
-venv amd64 3.8.10-0ubuntu1~20.04.2  
Err:1 http://security.ubuntu.com/ubuntu focal-updates/universe amd64 python3.8-v  
env amd64 3.8.10-0ubuntu1~20.04.2  
        404  Not Found [IP: 91.189.91.38 80]  
E: Failed to fetch http://security.ubuntu.com/ubuntu/pool/universe/p/python3.8/p  
ython3.8-venv_3.8.10-0ubuntu1~20.04.2_amd64.deb  404  Not Found [IP: 91.189.91.3  
8 80]  
E: Unable to fetch some archives, maybe run apt-get update or try with --fix-mis  
sing?  
2022-06-22 10:51:48 - ERROR - python3.8-venv install failed.  
root@hihope-virtual-machine:/home/hihope# apt-get update  
Reading package lists... Done  
E: Could not get lock /var/lib/apt/lists/lock. It is held by process 3833 (apt-g  
et)  
N: Be aware that removing the lock file is not a solution and may break your sys  
tem.  
E: Unable to lock directory /var/lib/apt/lists/  
root@hihope-virtual-machine:/home/hihope#  
root@hihope-virtual-machine:/home/hihope#  
root@hihope-virtual-machine:/home/hihope# sudo su
```


hihope@hihope-virtual-machine: ~

```
hihope@hihope-virtual-machine:~$ chmod u+x devicetool-linux-tool-3.0.0.401.sh  
hihope@hihope-virtual-machine:~$ sudo ./devicetool-linux-tool-3.0.0.401.sh  
[sudo] password for hihope:  
Verifying archive integrity... 100% All good.  
Uncompressing DevEco Device Tool 3.0.0+401.1ddd433 100%  
2022-06-22 10:59:38 - INFO - running installation by hihope  
2022-06-22 10:59:40 - INFO - python: /usr/bin/python /usr/bin/python3.8-config /  
usr/bin/python3.8 /usr/lib/python2.7 /usr/lib/python3.9 /usr/lib/python3.8 /etc/  
python3.8 /usr/local/lib/python3.8 /usr/include/python3.8  
E: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process 8904 (u  
nattended-upgr)  
N: Be aware that removing the lock file is not a solution and may break your sys  
tem.  
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is an  
other process using it?  
2022-06-22 10:59:40 - ERROR - python3-distutils install failed.  
hihope@hihope-virtual-machine:~$
```

安装完成后，当界面输出“Deveco Device Tool successfully installed.”时，表示 DevEco Device Tool 安装成功。(如果此过程有提示别的软件未安装，按照跳出来提示的命令进行对应安装)



```
[INFO    ] Creating launch script...  
[INFO    ] Creating setenv.sh script...  
[INFO    ] Updating settings...  
[INFO    ] Updating permissions...  
[INFO    ] Updating u-dev rules...  
[INFO    ] Installing mtd-utils...  
Deveco Device Tool successfully installed.
```

四、配置 Windows 远程访问 Ubuntu 环境：

1、安装 SSH 服务并获取远程访问的 IP 地址。在 Ubuntu 系统中，打开终端工具，执行如下命令安装 SSH 服务。

说明：如果执行该命令失败，提示 `openssh-server` 和 `openssh-client` 依赖版本不同，请根据 CLI 界面提示信息，安装 `openssh-client` 相应版本后（例如：`sudo apt-get install openssh-client=1:8.2p1-4`），再重新执行该命令安装 `openssh-server`。

命令：`sudo apt-get install openssh-server`

2、执行如下命令，启动 SSH 服务。

命令：`sudo systemctl start ssh`

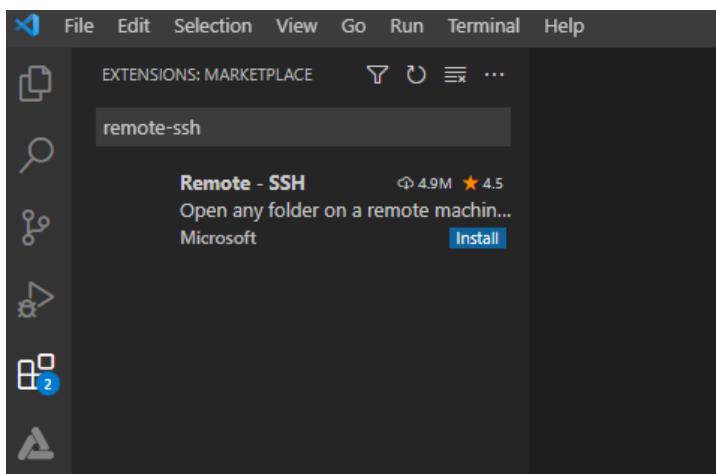
3、执行如下命令，获取当前用户的 IP 地址，用于 Windows 远程访问 Ubuntu 环境。

命令：`ifconfig`

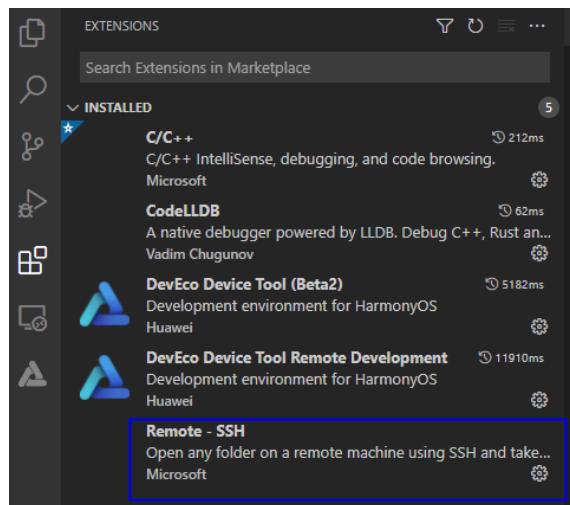
```
hihope@hihope-virtual-machine:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.1.10 netmask 255.255.255.0 broadcast 192.168.74.255
inet6 fe80::fe80:3a39:6c79 prefixlen 64 scopeid 0x20<link>
ether 00:0c:29:7c:b9:42 txqueuelen 1000 (Ethernet)
RX packets 10662 bytes 6990920 (6.9 MB)
RX errors 21 dropped 21 overruns 0 frame 0
TX packets 4489 bytes 414085 (414.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 19 base 0x2000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 715 bytes 63582 (63.5 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 715 bytes 63582 (63.5 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

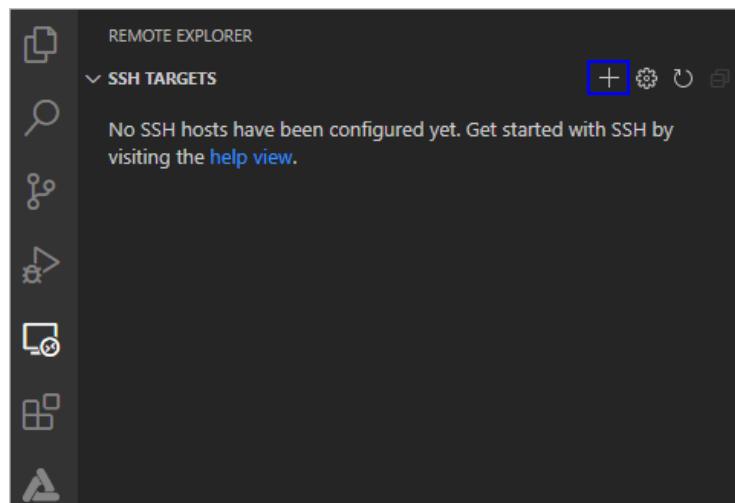
4、安装 Remote SSH，打开 Windows 系统下的 Visual Studio Code，点击 ，在插件市场的搜索输入框中输入“remote-ssh”。



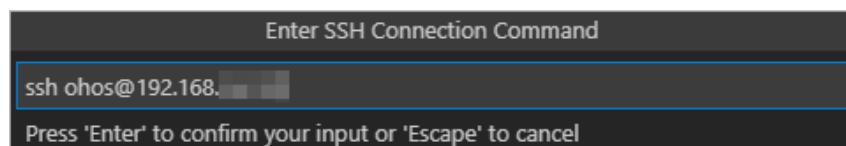
5、点击 `Remote-SSH` 的 `Install` 按钮，安装 `Remote-SSH`。安装成功后，在 `INSTALLED` 下可以看到已安装 `Remote-SSH`。



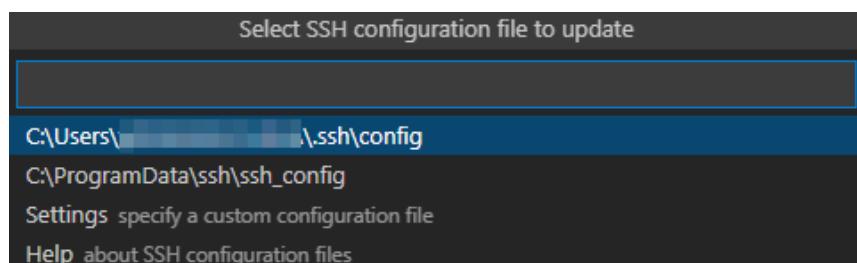
6、远程连接 Ubuntu 环境，打开 Windows 系统的 Visual Studio Code，点击 ，在 REMOTE EXPLORER 页面点击+按钮。



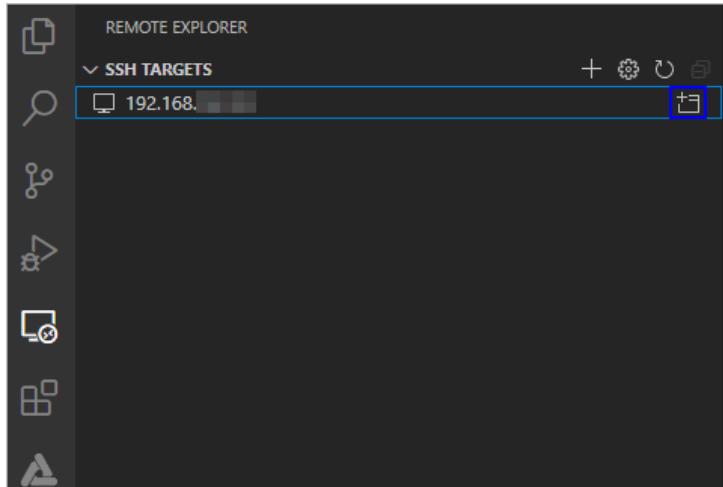
7、在弹出的 SSH 连接命令输入框中输入 “ssh username@ip_address”，其中 ip_address 为要连接的远程计算机的 IP 地址，username 为登录远程计算机的帐号。



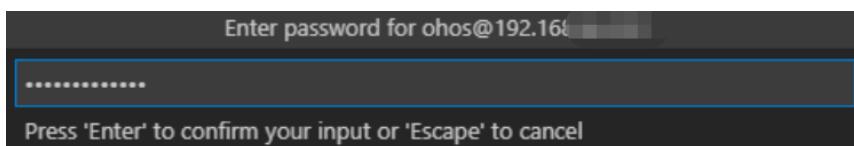
8、在弹出的输入框中，选择 SSH configuration 文件，选择默认的第一选项即可。



9、在 SSH TARGETS 中，找到远程计算机，点击 ，打开远程计算机。



10、在弹出的输入框中，选择 Linux，然后在选择 Continue，然后输入登录远程计算机的密码，连接远程计算机。



连接成功后，等待在远程计算机.vscode-server 文件夹下自动安装插件，安装完成后，根据界面提示在 Windows 系统下重新加载 Visual Studio Code，便可以在 Windows 的 DevEco Device Tool 界面进行源码开发、编译、烧录等操作。

第二章：创建源码工程

一、Windows 环境下按如下步骤配置：（可自行选择代码路径）

1、下载本课程源代码工程（firmware）：

下载链接：https://pan.baidu.com/s/1qLcjuDjsCbxWjID_2JkWKw

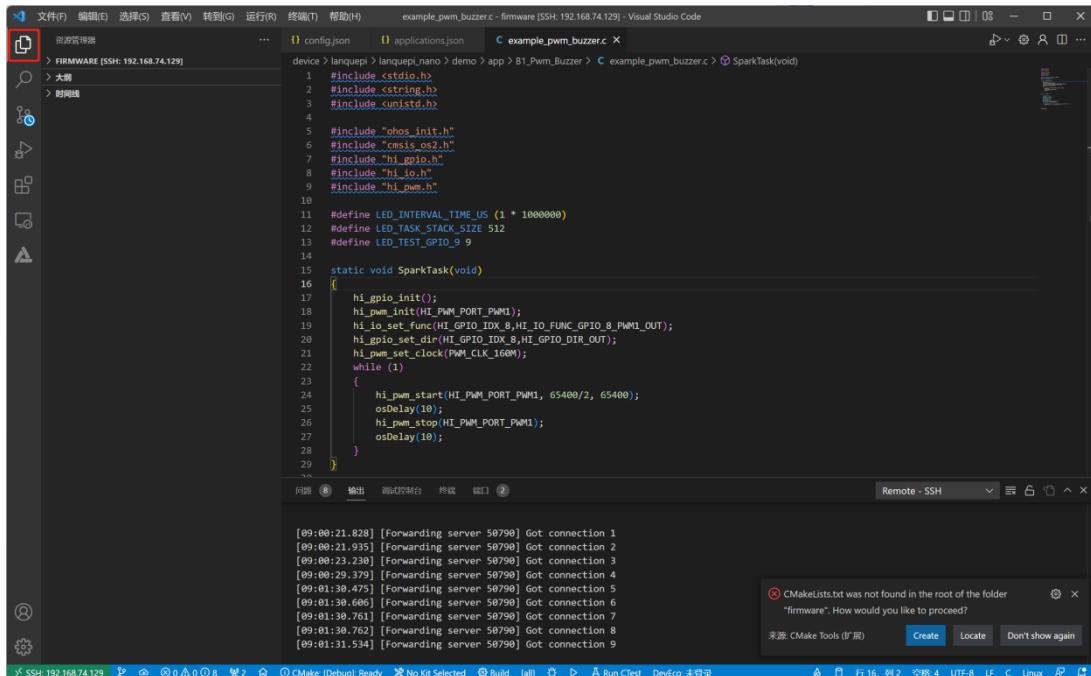
提取码：32di

将下载后的工程放入到虚拟机 Ubuntu 环境下的 Home, 执行如下命令对其进行解压。

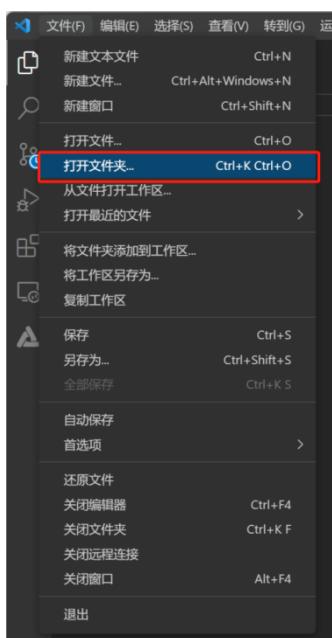
命令：`tar -zxf firmware1.tar.gz`

2、打开 DevEco Device Tool，点击如下截图方框。

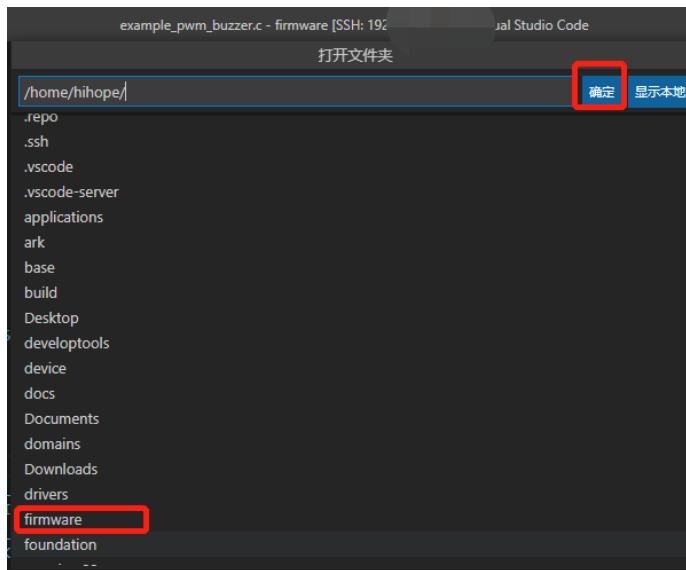
说明：确保工具是已经 SSH 到 Ubuntu 环境下的 VScode 界面。



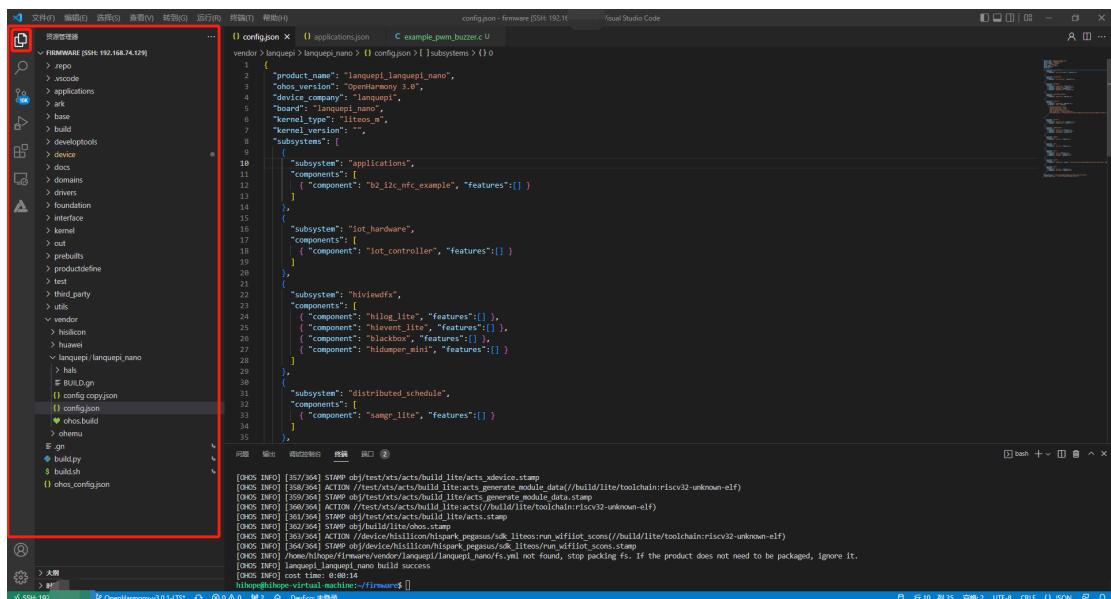
3、点击文件，选择打开文件夹。（代码路径可自行选择，以 firmware 为例）



4、在如下界面，选择 firmware 点击确定即可。



5、成功后出现如下界面。

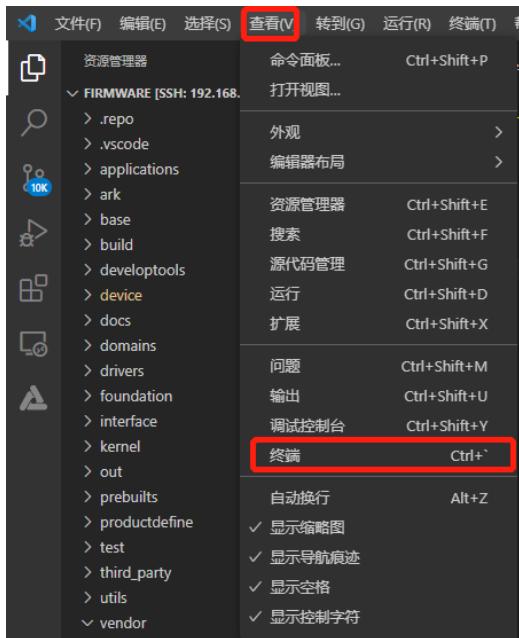


第三章：编译、烧录源码工程

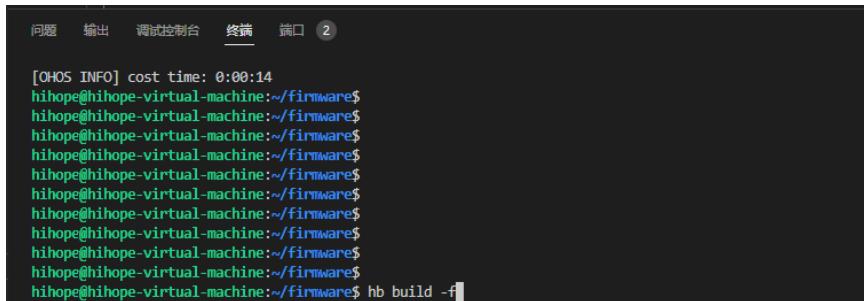
一、Windows 环境下按如下步骤进行

1、打开 DevEco Device Tool，点击查看后点击终端。

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 VScode 界面。



2、点击终端后出现如下界面：

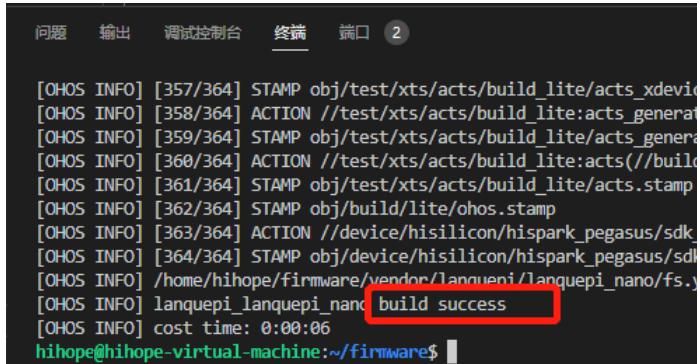


```
[OHO INFO] cost time: 0:00:14
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

3、输入如下命令进行编译

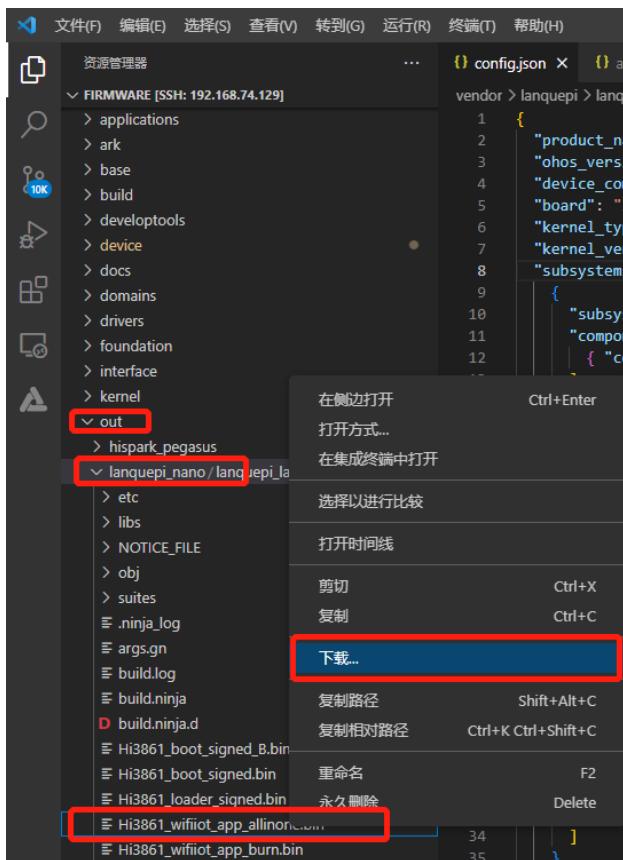
命令：hb build -f

4、编译成功后出现如下界面：

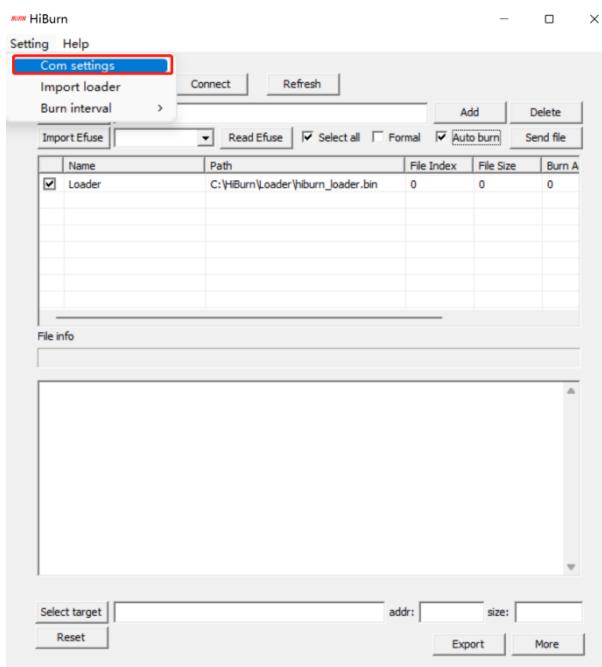


```
[OHO INFO] [357/364] STAMP obj/test/xts/acts/build_lite/acts_xdevice
[OHO INFO] [358/364] ACTION //test/xts/acts/build_lite:acts_generat
[OHO INFO] [359/364] STAMP obj/test/xts/acts/build_lite/acts_genera
[OHO INFO] [360/364] ACTION //test/xts/acts/build_lite:acts(/build
[OHO INFO] [361/364] STAMP obj/test/xts/acts/build_lite/acts.stamp
[OHO INFO] [362/364] STAMP obj/build/lite/ohos.stamp
[OHO INFO] [363/364] ACTION //device/hisilicon/hispark_pegasus/sdk_
[OHO INFO] [364/364] STAMP obj/device/hisilicon/hispark_pegasus/sdk_
[OHO INFO] /home/hihope/firmware/vendor/lanquepi/lanquepi_nano/fs.y
[OHO INFO] lanquepi_lanquepi_nano build success
[OHO INFO] cost time: 0:00:06
hihope@hihope-virtual-machine:~/firmware$
```

5、编译成功后，在资源管理器中找到 OUT，在目录中找到 lanquepi_nano 并打开，找到 Hi3861_wifiiot_app_allinone.bin。右击下载到桌面。



6、在桌面中打开 Hiburn 烧录工具，点击 Setting，对 Com setting 进行设置，将 Baud 设置成 921600，这样下载速度更快。

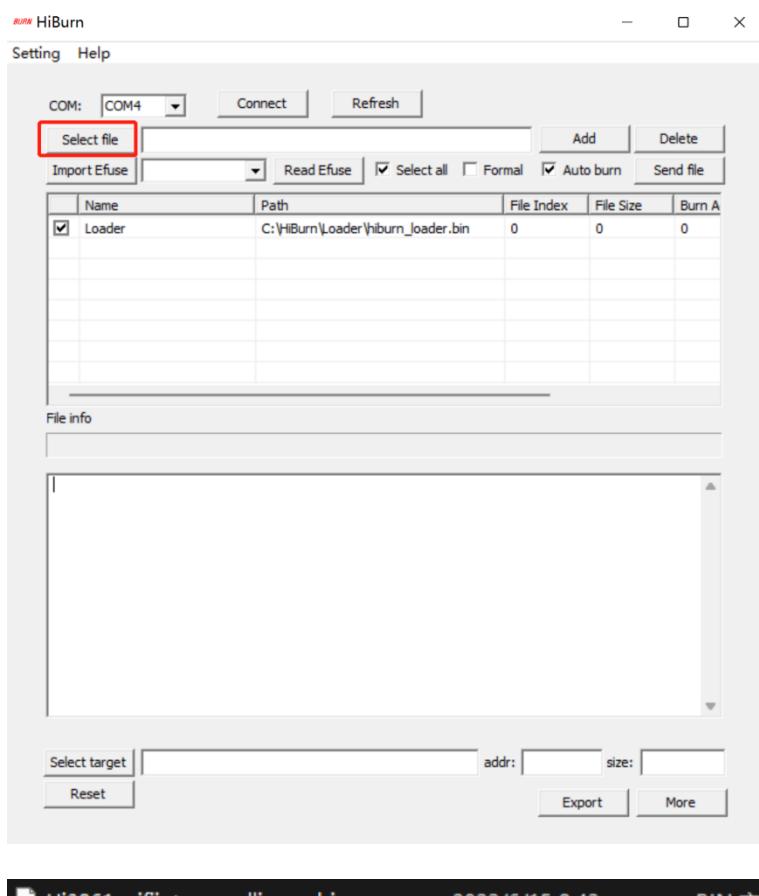




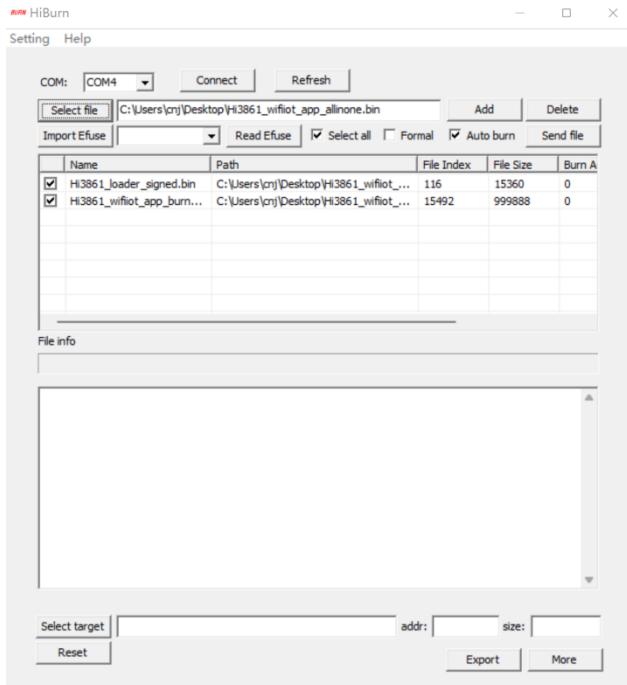
Com settings



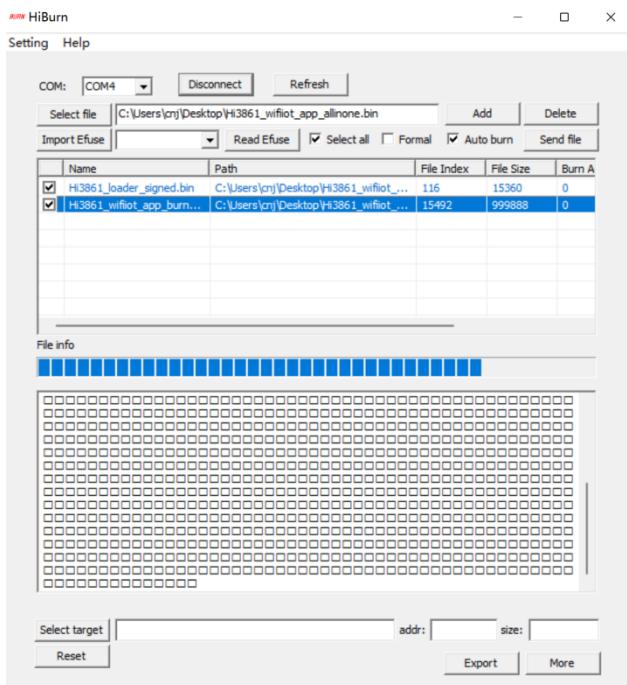
7、设置完成，选择 Select file，找到刚刚下载的.bin 文件后，选择打开，并且在 Hiburn 上进行 connect，并按下板子的复位键，即可烧录代码。



Hi3861_wifiiot_app_allinone.bin 2022/6/15 8:42 BIN 文件 992 KB



8、烧录完成后，选择 Disconnect，按下板子的复位键，即可听到蜂鸣器的声音。



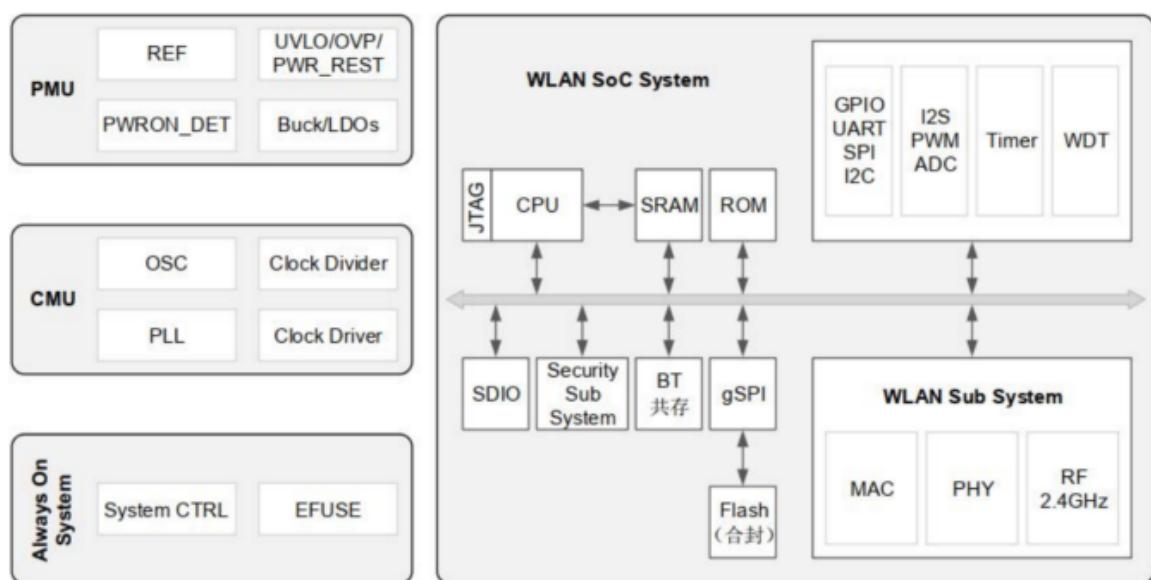
至此，你已经完成开发环境的搭建，可以开启你的鸿蒙开发旅途了。

第四章：Hi-12F 模块介绍

一、Hi-12F 模块简介

Hi-12F 模块搭载 Hi3861V100 核心处理器芯片。该芯片是一款高度集成的 2.4GHz 低功耗 SoC WiFi 芯片，集成了 IEEE 802.11b/g/n 基带和 RF 电路，RF 电路包括功率放大器 PA、低噪声放大器 LNA、RF balun、天线开关以及电源管理等模块；支持 20MHz 标准带宽和 5MHz/10MHz 窄带宽，提供最大 72.2Mbit/s 物理层速率。芯片 WiFi 基带支持正交频分复用(OFDM)技术，并向下兼容直接序列扩频(DSSS)和补码键控(CCK)技术，支持 IEEE 802.11 b/g/n 协议的各种数据速率。

Hi-12F 模块的 Hi3861V100 芯片同时集成高性能 32bit 微处理器、硬件安全引擎以及丰富的外设接口，外设接口包括 SPI、UART、I2C、PWM、GPIO 和多路 ADC，同时支持高速 SDIO2.0 Slave 接口，最高时钟可达 50MHz；芯片内置 SRAM 和 Flash，可独立运行，并支持在 Flash 上运行程序。

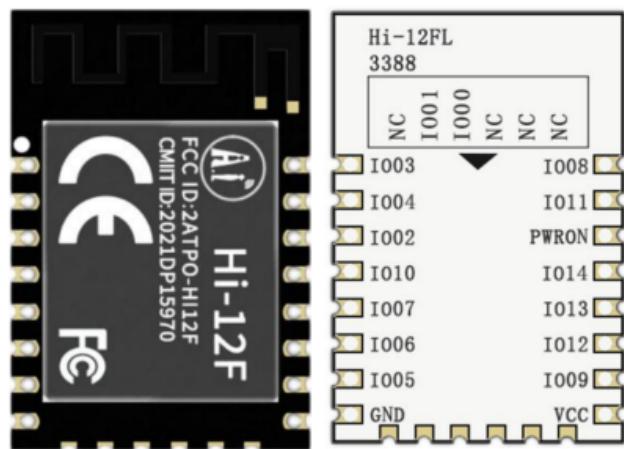


主芯片架构图

二、Hi-12F 模块特性

- 1x12.4GHz 频段(ch1~ch14)
- 支持 IEEE802.11b/g/n 单天线所有的数据速率
- 支持最大速率:72.2Mbps@HT20 MCS7 支持 STBC
- 支持 Short-GI
- 支持 STA 和 AP 形态，作为 AP 时最大支持 6 个 STA 接入
- 支持 WFA WPA/WPA2 personal、WPS2.0
- 高性能 32bit 微处理器，最大工作频率 160MHz，内嵌 SRAM 352KB、ROM 288KB，内嵌 2MB Flash
 - 内部集成 EFUSE，支持安全存储、安全启动、硬件 ID
 - 支持 256 节点 Mesh 组网
 - 支持 AT 指令，可快速上手
 - 开放操作系统 Huawei LiteOs，提供开放、高效、安全的系统开发、运行环境
 - 模组采用 SMD-22 封装
 - 支持 UART/SPI/I2C/GPIO/ADC/PWM/12S/SDIO 接口

三、Hi-12F 模块外观



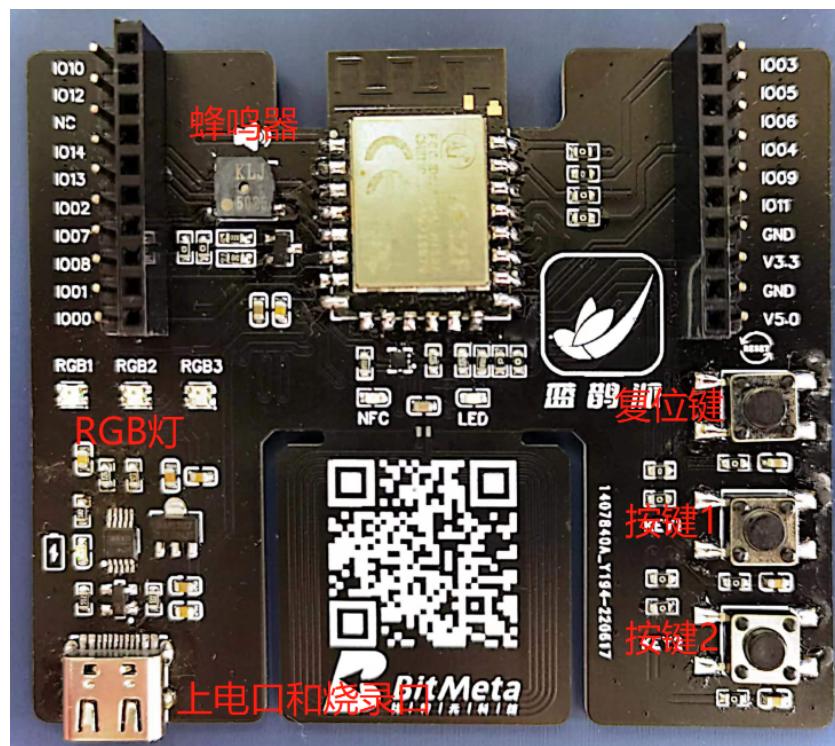
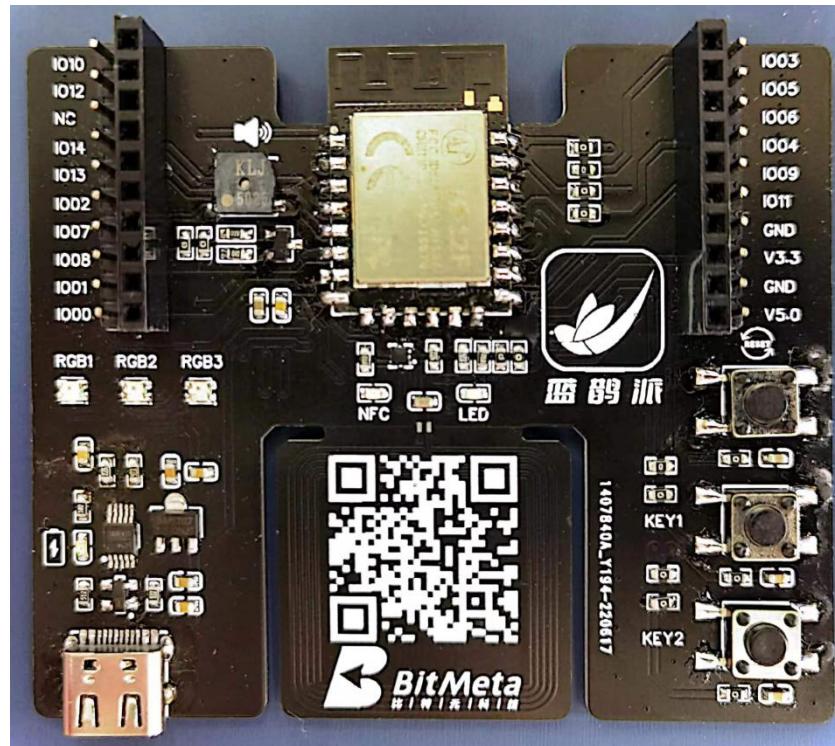
四、Hi-12F 管脚基本功能

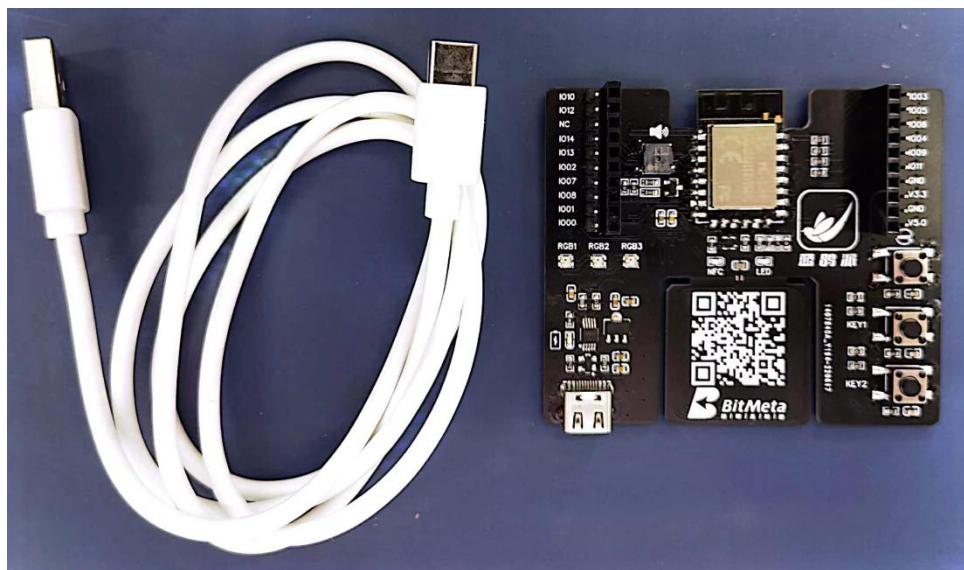
IO 名称 功能复用	UART	SPI	ADC	PWM	I2S	SDIO	I2C
IO00	UART1_TXD	SPI1_CLK		PWM3_OUT			I2C1_SDA
IO01	UART1_RXD	SPI1_RXD		PWM4_OUT			I2C1_SCL
IO02	UART1_RTS	SPI1_TXD		PWM2_OUT			
IO03	UART0_LOG_TXD UART1_CTS	SPI1_CS1		PWM5_OUT			I2C1_SDA
IO04	UART0_LOG_RXD		ADC1	PWM1_OUT			I2C1_SCL
IO05	UART1_RXD	SPI0_CS1	ADC2	PWM2_OUT	I2S0_MCK		
IO06	UART1_TXD	SPI0_CLK		PWM3_OUT	I2S0_TX		
IO07	UART1_CTS	SPI0_RXD	ADC3	PWM0_OUT	I2S0_CLK		
IO08	UART1_RTS	SPI0_TXD		PWM1_OUT	I2S0_WS		
IO09	UART2_RTS	SPI0_TXD	ADC4	PWM0_OUT	I2S0_MCK	SDIO_D2	I2C0_SCL
IO10	UART2_CTS	SPI0_CLK		PWM1_OUT	I2S0_TX	SDIO_D3	I2C0_SDA
IO11	UART2_TXD	SPI0_RXD	ADC5	PWM2_OUT	I2S0_RX	SDIO_CMD	
IO12	UART2_RXD	SPI0_CS1	ADC0	PWM3_OUT	I2S0_CLK	SDIO_CLK	
IO13	UART0_LOG_TXD UART2_RTS		ADC6	PWM4_OUT	I2S0_WS	SDIO_D0	I2C0_SDA
IO14	UART0_LOG_RXD UART2_CTS			PWM5_OUT		SDIO_D1	I2C0_SCL

注意：IO2/IO6/IO8 是硬件配置字，默认状态时下拉，上电时不能处于高电平状态，否则模块无法进入正常工作状态。

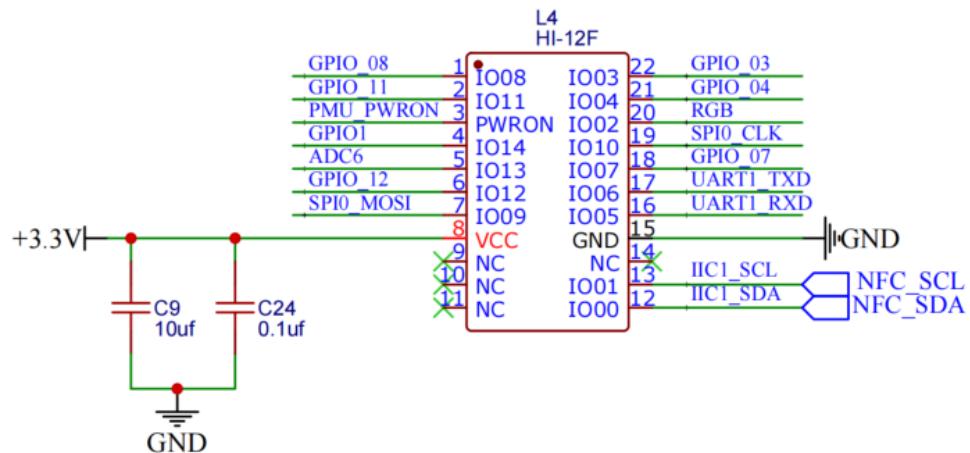
五、蓝鹊派—BMP20 外观

注意：蓝鹊派采用 USB 线即可进行代码的烧录，只需要用一根 Typ-C 数据线将蓝鹊派与电脑相连接即可。（电脑需要安装 CH-340 串口驱动）





六、蓝鹊派—BMP200 最小系统电路图



注意：蓝鹊派—BMP200 整体系统原理图，另附 PDF 文件

七、蓝鹊派—BMP200 配套 E53 三块拓展版

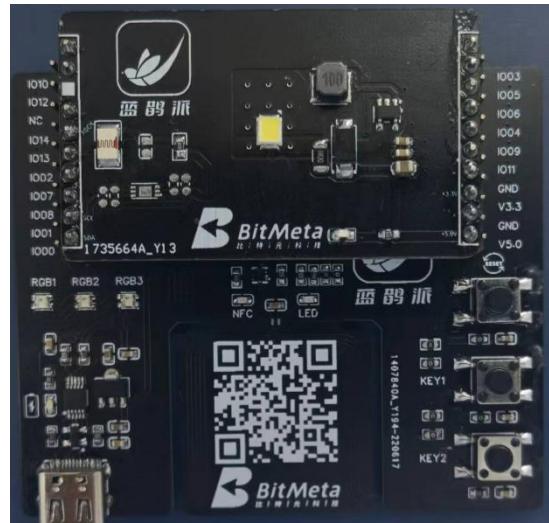
1. E53 介绍

E53 接口标准的 E 取自扩展（Expansion）的英文首字母，板子的尺寸为 $5 \times 3\text{cm}$ ，故采用 E53 作为前缀来命名尺寸为 $5 \times 3\text{cm}$ 类型的案例扩展板，任何一款满足标准设计的开发板均可直接适配 E53 扩展板。

E53 扩展板是根据不同的应用场景来设计的，以最大的程度在扩展板上还原真实应用场景。

E53 扩展接口在电气特性上，包含了常用的物联网感知层传感器通信接口，比如 5V、3.3V、GND、SPI、UART、IIC、ADC、DAC 等等，可以适配各种传感器，还留有 4 个普通 GPIO，如图：

1	SPI_SCK	PWM/GPIO	20
2	SPI_NSS	UART_RX	19
3	NC	UART_TX	18
4	GPIO	PWM/GPIO	17
5	ADC	SPI_MOSI	16
6	DAC	SPI_MISO	15
7	GPIO	GND	14
8	GPIO/PWM	VCC_3.3	13
9	IIC_SCL	GND	12
10	IIC_SDA	VCC_5.0	11



智慧路灯



智慧烟感



智慧农业

注意：将 E53 拓展板连接到主板时，需要注意将 LOGO 的正面朝上，小白点的朝左。智慧农业的小电风扇标有正负，红是正极，黑线是负极。

第五章：任务管理

八、实验目的

- 1、熟悉 HarmonyOS 系统任务管理相关函数的使用。
- 2、熟悉如何使用 VScode 编辑软件。
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、编程、调试。
- 4、检测搭建的软件及硬件环境是否可以正常使用。

九、实验原理

1、任务管理的介绍

• 从系统的角度看，任务是竞争系统资源的最小运行单元。HarmonyOS LiteOS-M 的任务可以使用或等待 CPU、使用内存空间等系统资源，并独立于其它任务运行，给用户提供多个任务，实现任务间的切换和通信。

• LiteOS-M 中的任务是抢占式调度机制，高优先级的任务可打断低优先级任务，低优先级任务必须在高优先级任务阻塞或结束后才能得到调度，同时支持时间片轮转调度方式。

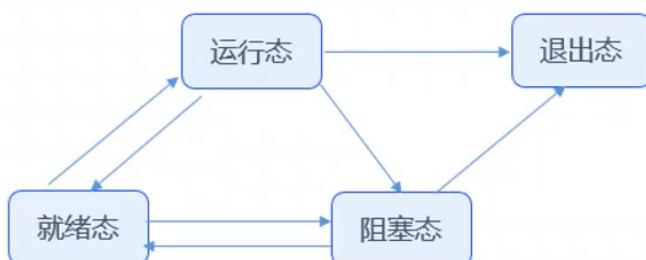
• LiteOS-M 的任务默认有 32 个优先级(0-31)，最高优先级为 0，最低优先级为 31。

2、任务运行状态

任务状态通常分为以下四种：

- 就绪（Ready）：该任务在就绪列表中，只等待 CPU。
- 运行（Running）：该任务正在执行。
- 阻塞（Blocked）：该任务不在就绪列表中。包含任务被挂起、任务被延时、任务正在等待信号量、读写队列或者等待读写事件等。
- 退出态（Dead）：该任务运行结束，等待系统回收资源。

• 任务状态切换如下图所示



3、任务的一些名词

- 任务 ID：在任务创建时通过参数返回给用户，作为任务的一个非常重要的标识。
- 任务优先级：优先级表示任务执行的优先顺序。
- 任务入口函数：每个新任务得到调度后将执行的函数。
- 任务控制块 TCB：每一个任务都含有一个任务控制块(TCB)。TCB 包含了任务上下文栈指针（stack pointer）、任务状态、任务优先级、任务 ID、任务名、任务栈大小等信息。TCB 可以反映出每个任务运行情况。
- 任务栈：每一个任务都拥有一个独立的栈空间，我们称为任务栈。
- 任务上下文：任务在运行过程中使用到的一些资源，如寄存器等，我们称为任务上下文。LiteOS 在任务挂起的时候会将本任务的任务上下文信息，保存在自己的任务栈里面，以便任务恢复后，从栈空间中恢复挂起时的上下文信息，从而继续执行被挂起时被打断的代码。

•任务切换:任务切换包含获取就绪列表中最高优先级任务、切出任务上下文保存、切入任务上下文恢复等动作。

三、API 的介绍

1、osThreadNew:

功能	创建任务，不能在中断服务调用该函数
函数定义	<code>osThreadId_t osThreadNew (osThreadFunc_t func, void *argument, const osThreadAttr_t *attr)</code>
参数	<code>func</code> : 任务函数 <code>argument</code> : 作为启动参数传递给任务函数的指针 <code>attr</code> : 任务入口函数的参数列表
返回	任务 ID

2、osThreadTerminate

功能	删除某个任务（一般是对非自任务操作）
函数定义	<code>osStatus_t osThreadTerminate (osThreadId_t thread_id)</code>
参数	<code>thread_id</code> : 任务 ID
返回	0—成功，非 0—失败

3、osThreadSuspend

功能	任务挂起
函数定义	<code>osStatus_t osThreadSuspend (osThreadId_t thread_id)</code>
参数	<code>thread_id</code> : 任务 ID
返回	0—成功，非 0—失败

4、osThreadResume

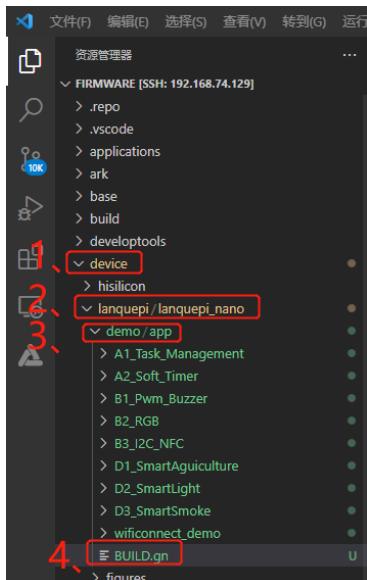
功能	任务恢复
函数定义	<code>osStatus_t osThreadResume (osThreadId_t thread_id)</code>
参数	<code>thread_id</code> : 任务 ID
返回	0—成功，非 0—失败

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 VScode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将 A1_Task_Management:a1_task_management 的注释#删除

```
device > lanquepi > lanquepi_nano > demo > app >  BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#A6_Message_Queue:a6_message_queue",
10    "#B1_First_Led:b1_led",
11    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12    "#B3_ADC:b3_adc",
13    "#B4_I2C_NFC:b4_i2c_nfc",
14    "#B5_RGB:b5_rgb",
15    "#D1_SmartAgriculture:e53_ia1_example",
16    "#D2_SmartLight:d2_smart_light",
17    "#D3_SmartSmoke:d3_smoke",
18  ]
19 }
```

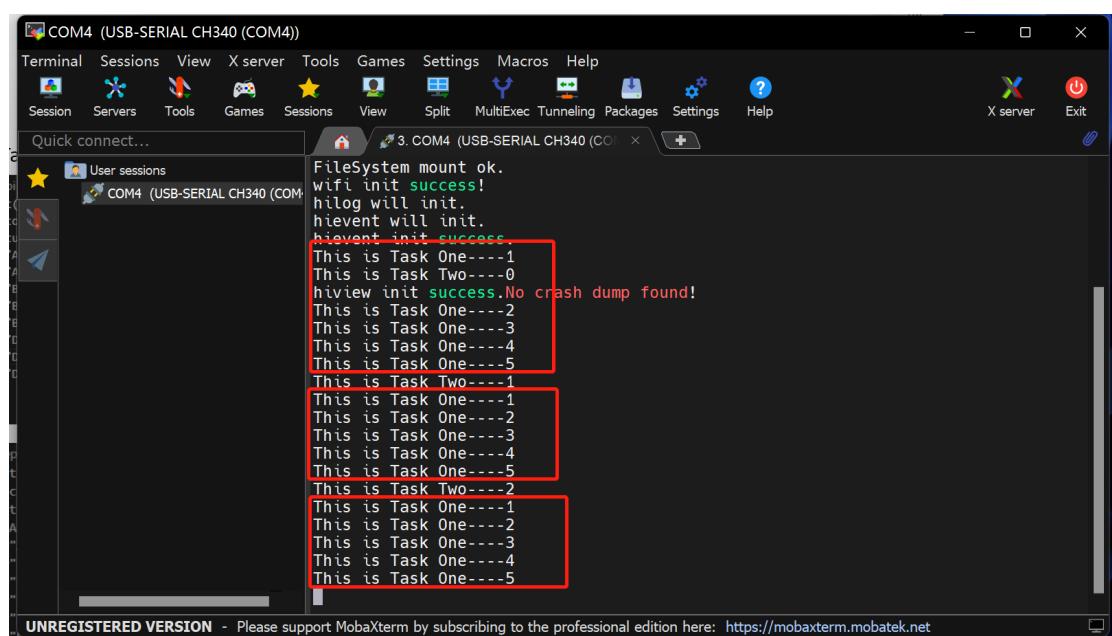
```
device > lanquepi > lanquepi_nano > demo > app >  BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#A6_Message_Queue:a6_message_queue",
10    "#B1_First_Led:b1_led",
11    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12    "#B3_ADC:b3_adc",
13    "#B4_I2C_NFC:b4_i2c_nfc",
14    "#B5_RGB:b5_rgb",
15    "#D1_SmartAgriculture:e53_ia1_example",
16    "#D2_SmartLight:d2_smart_light",
17    "#D3_SmartSmoke:d3_smoke",
18  ]
19 }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照上个章节的烧录过程进行烧录。

6、打开串口工具,出现如下结果，任务一 1 秒执行一次，任务二 5 秒执行一次。





7、本章节源文件代码如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"

#define oneSeconds 1000 * 1000 //延时 1s
#define fiveSeconds 1000 * 5000 //延时 5s

/*任务一*/
void Task_One(void)
{
    int taskOne_Count = 1; //定义一个整型变量，用于计数
    while (1)
    {
        printf("This is Task One----%d\r\n", taskOne_Count++);
        usleep(oneSeconds); //延时 1s
        if (taskOne_Count >= 6) //计数 5 次为一个周期
        {
            taskOne_Count = 1;
        }
    }
}

/*任务二*/
void Task_Two(void)
{
    int taskTwo_Count = 0; //定义一个整型变量，用于计数
    while (1)
    {
        printf("This is Task Two----%d\r\n", taskTwo_Count++);
        usleep(fiveSeconds); //延时 5s
    }
}

/*任务创建*/
static void Thread_example(void)
{
    osThreadAttr_t attr;
    attr.name = "Task_One"; //任务名称
    attr.attr_bits = 0U; //属性位用于设置 cmsis_os2.h 中的 osStatus_t osThreadJoin (osThreadId_t thread_id); 函数是否能使用，  
默认为 0
    attr.cb_mem = NULL; //控制块的指针，不操作设置为 NULL
    attr.cb_size = 0U; //控制块的指针大小，0
    attr.stack_mem = NULL; //任务栈指针，不操作设置为 NULL
    attr.stack_size = 1024 * 4; //任务栈大小：8 字节对齐的大小
    attr.priority = 25; //任务的优先级：25
```



```
if (osThreadNew((osThreadFunc_t)Task_One, NULL, &attr) == NULL) //通过 osThreadNew 函数传递参数设置任务函数，创建任务 1
{
    printf("Failed to create Task_One!\n");
}

attr.name = "Task_Two"; //重新赋值任务名

if (osThreadNew((osThreadFunc_t)Task_Two, NULL, &attr) == NULL)
{
    printf("Failed to create Task_Two!\n");
}

}

SYS_RUN(Thread_example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("a1_task_management"){
    sources = [
        "example_task_management.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```



第六章：软件定时器

一、实验目的

- 1、熟悉 HarmonyOS 系统下软件定时器相关函数的使用。
- 2、熟悉如何使用 VScode 编辑软件。
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、编程、调试。
- 4、检测搭建的软件及硬件环境是否可以正常使用。

二、实验原理

1、软件定时器的介绍

软件定时器，是基于系统 Tick 时钟中断且由软件来模拟的定时器，当经过设定的 Tick 时钟计数值后会触发用户定义的回调函数。定时精度与系统 Tick 时钟的周期有关。

硬件定时器受硬件的限制，数量上不足以满足用户的实际需求，因此为了满足用户需求，提供更多的定时器，LiteOS 操作系统提供软件定时器功能。软件定时器扩展了定时器的数量，允许创建更多的定时业务。

2、软件定时器功能上支持：

- 静态裁剪：能通过宏关闭软件定时器功能。
- 软件定时器创建。
- 软件定时器启动。
- 软件定时器停止。
- 软件定时器删除。
- 软件定时器剩余 Tick 数获取。

3、软件定时器运行机制

软件定时器使用了系统的一个队列和一个任务资源，软件定时器的触发遵循队列先进先出规则，定时时间短的定时器总是比定时时间长的靠近队列头，满足优先被触发的准则。

软件定时器以 Tick 为基本计时单位，当用户创建并启动一个软件定时器时，LiteOS 会根据当前系统 Tick 时间及用户设置的定时时间间隔确定该定时器的到期 Tick 时间，并将该定时器控制结构挂入计时全局链表。

当 Tick 中断到来时，在 Tick 中断处理函数中扫描软件定时器的计时全局链表，看是否有定时器超时，若有则将超时的定时器记录下来。

Tick 中断处理函数结束后，软件定时器任务（优先级为最高）被唤醒，在该任务中调用之前记录下来的定时器的超时回调函数。

三、API 的介绍

1、osTimerNew:

功能	创建定时器，不能在中断服务调用该函数
函数定义	<code>osTimerId_t osTimerNew (osTimerFunc_t func, osTimerType_t type, void *argument, const osTimerAttr_t *attr)</code>
参数	<code>func</code> : 函数指针指向回调函数 <code>type</code> : 定时器类型， <code>osTimerOnce</code> 表示单次定时器， <code>ostimer</code> 周期表示周期性定时器 <code>argument</code> : 定时器回调函数的参数 <code>attr</code> : 计时器属性
返回	定时器 ID

2、osTimerStart

功能	启动定时器，不能在中断服务调用该函数
函数定义	<code>osStatus_t osTimerStart (osTimerId_t timer_id,uint32_t ticks)</code>
参数	<code>timer_id</code> : 定时器 ID <code>ticks</code> : 时间滴答计时器的值
返回	0—成功，非 0—失败

3、osTimerStop

功能	停止定时器
函数定义	<code>osStatus_t osTimerStop (osTimerId_t timer_id)</code>
参数	<code>timer_id</code> : 定时器 ID
返回	0—成功，非 0—失败

4、osTimerDelete

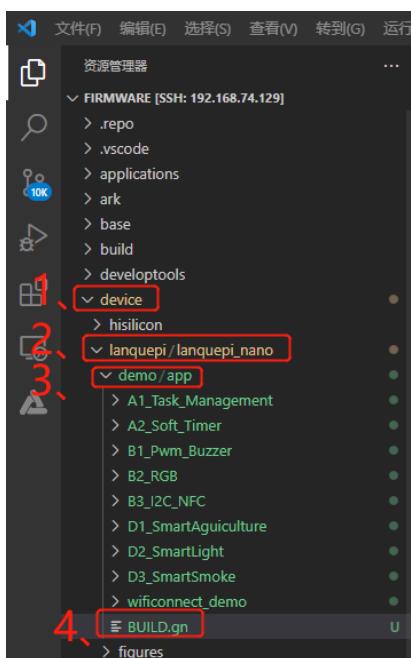
功能	删除定时器
函数定义	<code>osStatus_t osTimerDelete (osTimerId_t timer_id)</code>
参数	<code>timer_id</code> : 定时器 ID
返回	0—成功，非 0—失败

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将 A2_Soft_Timer:a2_soft_timer 的注释#删除

```
device > lanquepi > lanquepi_nano > demo > app > ≡ BUILD.gn
1   import("//build/lite/config/component/lite_component.gni")
2   lite_component("app") {
3     features = [
4       "#"A1_Task_Management:a1_task_management",
5       #"A2_Soft_Timer:a2_soft_timer",
6       "#"A3_Semaphore:a3_semaphore",
7       "#"A4_Event_Management:a4_event_management",
8       "#"A5_Mutex:a5_mutex",
9       "#"A6_Message_Queue:a6_message_queue",
10      "#"B1_First_Led:b1_led",
11      "#"B2_Pwm_Buzzer:b2_pwm_buzzer",
12      "#"B3_ADC:b3_adc",
13      "#"B4_I2C_NFC:b4_i2c_nfc",
14      "#"B5_RGB:b5_rgb",
15      "#"D1_SmartAgriculture:e53_ia1_example",
16      "#"D2_SmartLight:d2_smart_light",
17      "#"D3_SmartSmoke:d3_smart_smoke",
18    ]
19  }
```



```
device > lanquepi > lanquepi_nano > demo > app > ≡ BUILD.gn
1   import("//build/lite/config/component/lite_component.gni")
2   lite_component("app") {
3     features = [
4       "#"A1_Task_Management:a1_task_management",
5       "A2_Soft_Timer:a2_soft_timer",
6       "#"A3_Semaphore:a3_semaphore",
7       "#"A4_Event_Management:a4_event_management",
8       "#"A5_Mutex:a5_mutex",
9       "#"A6_Message_Queue:a6_message_queue",
10      "#"B1_First_Led:b1_led",
11      "#"B2_Pwm_Buzzer:b2_pwm_buzzer",
12      "#"B3_ADC:b3_adc",
13      "#"B4_I2C_NFC:b4_i2c_nfc",
14      "#"B5_RGB:b5_rgb",
15      "#"D1_SmartAgriculture:e53_ia1_example",
16      "#"D2_SmartLight:d2_smart_light",
17      "#"D3_SmartSmoke:d3_smart_smoke",
18    ]
19  }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、打开串口工具,出现如下结果，1S 计时到后会触发 Timer1_Callback()函数，3S 计时到后会触发 Timer2_Callback()函数。

```
ready to OS start
sdk ver:Hi3861V100R001C00SPC025 2020-09-03 18:10:00
formatting spiffs...
FileSystem mount ok.
wifi init success!
hilog will init.
hievent will init.
hievent init success.
hiview init success.This is Timer1_Callback!
No crash dump found!
This is Timer1_Callback!
This is Timer2_Callback!
This is Timer1_Callback!
This is Timer1_Callback!
This is Timer1_Callback!
This is Timer1_Callback!
This is Timer2_Callback!
This is Timer1_Callback!
This is Timer1_Callback!
This is Timer1_Callback!
This is Timer1_Callback!
This is Timer2_Callback!
This is Timer1_Callback!
This is Timer1_Callback!
This is Timer2_Callback!
This is Timer1_Callback!
```

7、本章节源文件代码如下:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```



```
#include "hi_types.h"
#include "ohos_init.h"
#include "cmsis_os2.h"

#define oneSecond 1000

hi_u32 runTime_One,runTime_Two; //定义两个变量，表示执行周期时间

/*定时器 1 回调函数*/
void Timer1_Callback(void *arg)
{
    (void)arg;
    printf("This is Timer1_Callback!\r\n");
}

/*定时器 2 回调函数*/
void Timer2_Callback(void *arg)
{
    (void)arg;
    printf("This is Timer2_Callback!\r\n");
}

/*定时器任务创建*/
static void Timer_example(void)
{
    osTimerId_t timer1,timer2; //标识定时器 ID, timer1、timer2
    osStatus_t status; //CMSIS-RTOS 的返回值，用于检测
    hi_u32 timerDelay; //定义延时变量

    runTime_One = 1U; //Hi3861 1U=10ms,100U=1S.

    timer1 = osTimerNew(Timer1_Callback, osTimerPeriodic, &runTime_One, NULL); //返回值是定时器 ID，赋给 timer1
    if (timer1 != NULL)
    {
        timerDelay = oneSecond; //延时 1S 变量
        status = osTimerStart(timer1, timerDelay); //启动定时器，延时 1S，返回值 0：成功；返回值 1：失败
        if (status != osOK) //如果返回值不为 0，则表示失败，什么都不做
        {
            //Do nothing
        }
    }

    runTime_Two = 1U;

    timer2 = osTimerNew(Timer2_Callback, osTimerPeriodic, &runTime_Two, NULL);
    if (timer2 != NULL)
    {
        timerDelay = 3*oneSecond; //延时 3S 变量
        status = osTimerStart(timer2, timerDelay); //启动定时器，延时 1S，返回值 0：成功；返回值 1：失败
        if (status != osOK) //如果返回值不为 0，则表示失败，什么都不做
        {
            //Do nothing
        }
    }
}
```



```
}
```

```
SYS_RUN(Timer_example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("a2_soft_timer"){

    sources = [
        "example_soft_timer.c"
    ]

    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]

}
```

第七章：信号量

一、实验目的

- 1、熟悉 HarmonyOS 系统下的信号量相关函数的使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1、信号量的介绍

• 信号量（Semaphore）是一种实现任务间通信的机制，实现任务之间同步或临界资源的互斥访问。在多任务操作系统中，不同的任务之间需要同步运行，信号量功能可以为用户提供这方面的支持。

2、信号量的使用方式

信号量可以被任务获取或者申请，不同的信号量通过信号量索引号来唯一确定，每个信号量都有一个计数值和任务队列。通常一个信号量的计数值用于对应有效的资源数，表示剩下的可被占用的互斥资源数，其值的含义分两种情况：

0：表示没有积累下来的 Post 操作，且有可能有在此信号量上阻塞的任务；

正值：表示有一个或多个 Post 下来的释放操作；

当任务申请(Pend)信号量时，如果申请成功，则信号量的计数值递减，如若申请失败，则挂起在该信号量的等待任务队列上，一旦有任务释放该信号量，则等待任务队列中的任务被唤醒开始执行。

3、信号量的使用场景

信号量是一种非常灵活的同步方式，可以运用在多种场合中，实现锁、同步、资源计数等功能，也能方便的用于任务与任务，中断与任务的同步中。

• 互斥锁：用作互斥时，信号量创建后记数是满的，在需要使用临界资源时，先申请信号量，使其变空，这样其他任务需要使用临界资源时就会因为无法申请到信号量而阻塞，从而保证了临界资源的安全。

• 任务间同步：用作同步时，信号量在创建后被置为空，任务 1 申请信号量而阻塞，任务 2 在某种条件发生后，释放信号量，于是任务 1 得以进入 READY 或 RUNNING 态，从而达到了两个任务间的同步。

• 资源计数：用作资源计数时，信号量的作用是一个特殊的计数器，可以递增或者递减，但是值永远不能为负值，典型的应用场景是生产者与消费者的场景。

• 中断与任务的同步：用作中断与任务的同步时，可以在中断未触发时将信号量的值置为 0，从而堵塞断服务处理任务，一旦中断被触发，则唤醒堵塞的中断服务处理任务进行中断处理。

三、API 的介绍

1、osSemaphoreNew

功能	创建信号量，不能在中断服务调用该函数
函数定义	<code>osSemaphoreId_t osSemaphoreNew (uint32_t max_count, uint32_t initial_count, const osSemaphoreAttr_t *attr)</code>
参数	<code>max_count</code> : 信号量计数值的最大值 <code>initial_count</code> : 信号量计数值的初始值 <code>attr</code> : 信号量属性
返回	信号量 ID

2、osSemaphoreAcquire

功能	获取信号量，一直等待，直到由参数 <code>semaphore_id</code> 指定的信号量对象的标记可用为止。 如果参数 <code>timeout</code> 设置为 0，可以从中断服务例程调用
函数定义	<code>osStatus_t osSemaphoreAcquire (osSemaphoreId_t semaphore_id, uint32_t timeout)</code>
参数	<code>semaphore_id</code> : 信号量 ID <code>timeout</code> : 超时值
返回	0—成功，非 0—失败

3、osSemaphoreRelease

功能	释放信号量，该函数可以在中断服务例程调用
函数定义	<code>osStatus_t osSemaphoreRelease (osSemaphoreId_t semaphore_id)</code>
参数	<code>semaphore_id</code> : 信号量 ID
返回	0—成功，非 0—失败

4、osSemaphoreDelete

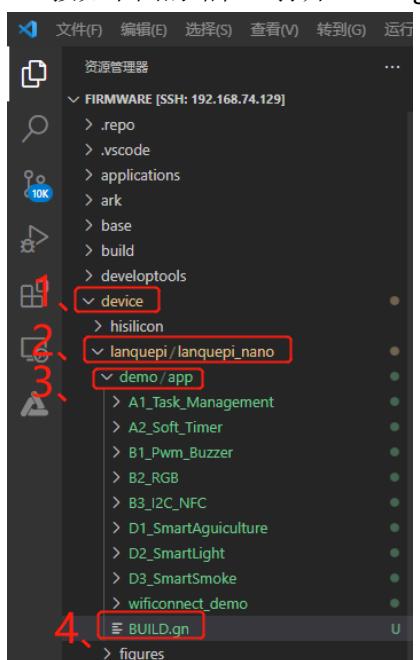
功能	删除信号量
函数定义	<code>osStatus_t osSemaphoreDelete (osSemaphoreId_t semaphore_id)</code>
参数	<code>semaphore_id</code> : 信号量 ID
返回	0—成功，非 0—失败

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将 A3_Semaphore:a3_semaphore 的注释#删除

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     #"A1_Task_Management:a1_task_management",
5     #"A2_Soft_Timer:a2_soft_timer",
6     #"A3_Semaphore:a3_semaphore",
7     #"A4_Event_Management:a4_event_management",
8     #"A5_Mutex:a5_mutex",
9     #"A6_Message_Queue:a6_message_queue",
10    #"B1_First_Led:b1_led",
11    #"B2_Pwm_Buzzer:b2_pwm_buzzer",
12    #"B3_ADC:b3_adc",
13    #"B4_I2C_NFC:b4_i2c_nfc",
14    #"B5_RGB:b5_rgb",
15    #"D1_SmartAgriculture:e53_ia1_example",
16    #"D2_SmartLight:d2_smart_light",
17    #"D3_SmartSmoke:d3_smoke",
18  ]
19 }
```

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     #"A1_Task_Management:a1_task_management",
5     #"A2_Soft_Timer:a2_soft_timer",
6     "A3_Semaphore:a3_semaphore",
7     #"A4_Event_Management:a4_event_management",
8     #"A5_Mutex:a5_mutex",
9     #"A6_Message_Queue:a6_message_queue",
10    #"B1_First_Led:b1_led",
11    #"B2_Pwm_Buzzer:b2_pwm_buzzer",
12    #"B3_ADC:b3_adc",
13    #"B4_I2C_NFC:b4_i2c_nfc",
14    #"B5_RGB:b5_rgb",
15    #"D1_SmartAgriculture:e53_ia1_example",
16    #"D2_SmartLight:d2_smart_light",
17    #"D3_SmartSmoke:d3_smoke",
18  ]
19 }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、打开串口工具,出现如下结果。

```
ready to OS start
sdk ver:Hi3861V100R001C00SPC025 2020-09-03 18:10:00
formatting spiffs...
FileSystem mount ok.
wifi init success!
hilog will init.
hievent will init.
hievent init success.
hiview init success.
Thread_Semaphore1 Release Semap

Thread_Semaphore2 get Semap

Thread_Semaphore3 get Semap
No crash dump found!

Thread_Semaphore1 Release Semap

Thread_Semaphore2 get Semap

Thread_Semaphore3 get Semap

Thread_Semaphore1 Release Semap

Thread_Semaphore2 get Semap
```



7、本章节源文件代码如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"

#define oneSeconds 1000 * 1000 //延时 1s

osSemaphoreId_t semphore1;//标识信号量对象 ID

void Thread_Semaphore1(void)
{
    while (1)
    {
        osSemaphoreRelease(semphore1); //申请两次 semphore1 信号量，使得 Thread_Semaphore2 和 Thread_Semaphore3 能同步执行
        osSemaphoreRelease(semphore1); //此处若只申请一次信号量，则 Thread_Semaphore2 和 Thread_Semaphore3 会交替运行。
        printf("\r\nThread_Semaphore1 Release Semaphore\r\n");
        usleep(oneSeconds);
    }
}

void Thread_Semaphore2(void)
{
    while (1)
    {
        osSemaphoreAcquire(semphore1, osWaitForever); //等待 semphore1 信号量
        printf("\r\nThread_Semaphore2 get Semaphore\r\n");
        usleep(oneSeconds/100);
    }
}

void Thread_Semaphore3(void)
{
    while (1)
    {
        osSemaphoreAcquire(semphore1, osWaitForever); //等待 semphore1 信号量
        printf("\r\nThread_Semaphore3 get Semaphore\r\n");
        usleep(oneSeconds/100);
    }
}

void Semaphore_example(void)
{
    osThreadAttr_t attr;
    attr.name = "Thread_Semaphore1"; //任务名称
    attr.attr_bits = 0U; //属性位用于设置 cmsis_os2.h 中的 osStatus_t osThreadJoin (osThreadId_t thread_id);函数是否能使用，默认为 0
    attr.cb_mem = NULL; //控制块的指针，不操作设置为 NULL
    attr.cb_size = 0U; //控制块的指针大小，0
```



```
attr.stack_mem = NULL;           //任务栈指针，不操作设置为NULL
attr.stack_size = 1024 * 4;       //任务栈大小：8字节对齐的大小
attr.priority = 24;              //任务的优先级：25
if (osThreadNew((osThreadFunc_t)Thread_Semaphore1, NULL, &attr) == NULL)
{
    printf("Failed to create Thread_Semaphore1!\n");
}

attr.name = "Thread_Semaphore2";
if (osThreadNew((osThreadFunc_t)Thread_Semaphore2, NULL, &attr) == NULL)
{
    printf("Failed to create Thread_Semaphore2!\n");
}

attr.name = "Thread_Semaphore3";
if (osThreadNew((osThreadFunc_t)Thread_Semaphore3, NULL, &attr) == NULL)
{
    printf("Failed to create Thread_Semaphore3!\n");
}

semaphore1 = osSemaphoreNew(4, 0, NULL);
if (semaphore1 == NULL)
{
    printf("Failed to create Semaphore1!\n");
}
}

SYS_RUN(Semaphore_example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("a3_semaphore") {
    sources = [
        "example_semaphore.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

第八章：事件管理

一、实验目的

- 1、熟悉 HarmonyOS 系统下的事件相关函数的使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1、事件的基本概念

事件是一种实现任务间通信的机制，可用于实现任务间的同步。事件的特点是：

- 事件通信只能是事件类型的通信，无数据传输，一个任务可以等待多个事件的发生：可以是任意一个事件发生时唤醒任务进行事件处理；也可以是几个事件都发生后才唤醒任务进行事件处理。事件集合用 32 位无符号整型变量来表示，每一位代表一个事件。

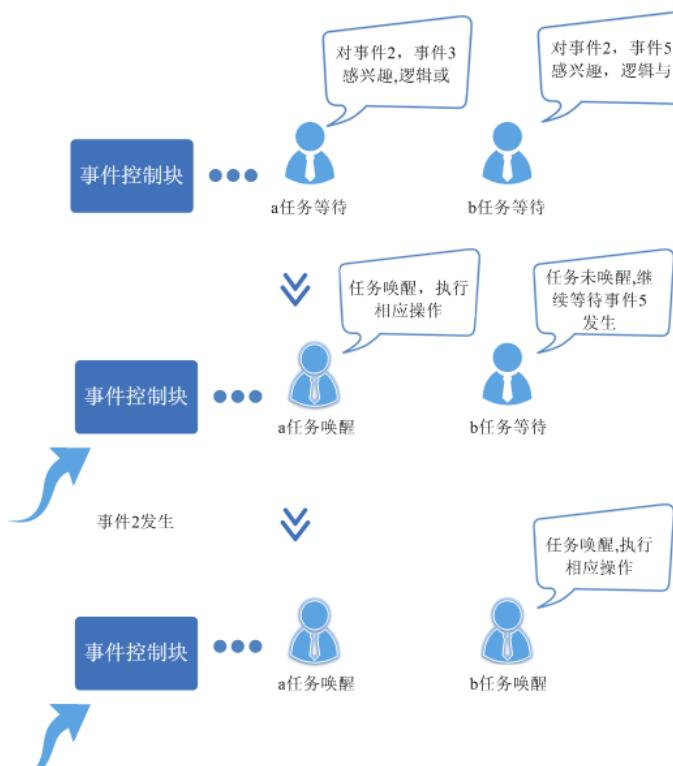
- 多任务环境下，任务之间往往需要同步操作。事件可以提供一对多、多对多的同步操作。一对多同步模型：一个任务等待多个事件的触发；多对多同步模型：多个任务等待多个事件的触发。任务可以通过创建事件控制块来实现对事件的触发和等待操作。LiteOS 的事件仅用于任务间的同步。

2、事件运作机制

读事件时，可以根据入参事件掩码类型 `uwEventMask` 读取事件的单个或者多个事件类型。事件读取成功后，如果设置 `LOS_WAITMODE_CLR` 会清除已读取到的事件类型，反之不会清除已读取到的事件类型，需显式清除。可以通过入参选择读取模式，读取事件掩码类型中所有事件还是读取事件掩码类型中任意事件。

写事件时，对指定事件写入指定的事件类型，可以一次同时写多个事件类型。写事件会触发任务调度。

清除事件时，根据入参事件和待清除的事件类型，对事件对应位进行清 0 操作。





三、API 的介绍

1、osEventFlagsNew

功能	创建事件标记对象，不能在中断服务调用该函数
函数定义	<code>osEventFlagsId_t osEventFlagsNew (const osEventFlagsAttr_t *attr)</code>
参数	<code>attr</code> : 事件标志属性;空:默认值
返回	事件 ID

2、osEventFlagsSet

功能	设置事件标记
函数定义	<code>int32_t osEventFlagsSet (osEventFlagsId_t ef_id, uint32_t flags)</code>
参数	<code>ef_id</code> : 事件标志由 <code>osEventFlagsNew</code> 获得的 ID <code>flags</code> : 指定设置的标志
返回	事件标记

3、osEventFlagsWait

功能	等待事件标记触发，直到设置了由参数 <code>ef_id</code> 指定的事件对象中的任何或所有由参数 <code>flags</code> 指定的事件标志。当这些事件标志被设置，函数立即返回。否则，线程将被置于阻塞状态。
函数定义	<code>uint32_t osEventFlagsWait (osEventFlagsId_t ef_id, uint32_t flags, uint32_t options, uint32_t timeout)</code>
参数	<code>ef_id</code> : 事件标志由 <code>osEventFlagsNew</code> 获得的 ID <code>flags</code> : 指定要等待的标志 <code>options</code> : 指定标记选项 <code>timeout</code> : 超时时间，0 表示不超时
返回	触发事件标记

4、osEventFlagsDelete

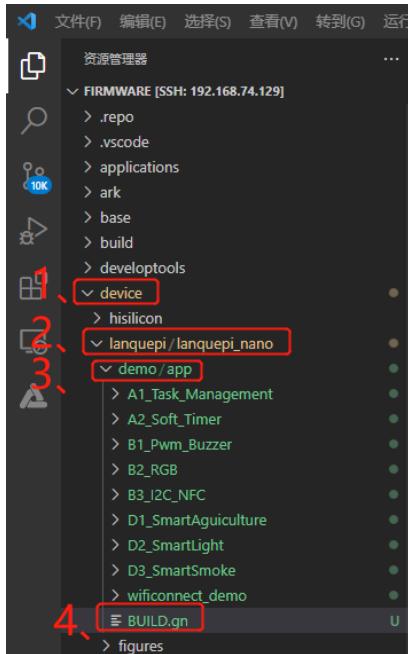
功能	删除事件标记对象
函数定义	<code>osStatus_t osEventFlagsDelete (osEventFlagsId_t ef_id)</code>
参数	<code>ef_id</code> : 事件标志由 <code>osEventFlagsNew</code> 获得的 ID
返回	0—成功，非 0—失败

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将 A4_Event_Management:a4_event_management 的注释#删除

```
device > lanquepi > lanquepi_nano > demo > app > └─ BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     #"A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#A6_Message_Queue:a6_message_queue",
10    "#B1_First_Led:b1_led",
11    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12    "#B3_ADC:b3_adc",
13    "#B4_I2C_NFC:b4_i2c_nfc",
14    "#B5_RGB:b5_rgb",
15    "#D1_SmartAgriculture:e53_ia1_example",
16    "#D2_SmartLight:d2_smart_light",
17    "#D3_SmartSmoke:d3_smoke",
18  ]
19 }
```

```
device > lanquepi > lanquepi_nano > demo > app > └─ BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#A6_Message_Queue:a6_message_queue",
10    "#B1_First_Led:b1_led",
11    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12    "#B3_ADC:b3_adc",
13    "#B4_I2C_NFC:b4_i2c_nfc",
14    "#B5_RGB:b5_rgb",
15    "#D1_SmartAgriculture:e53_ia1_example",
16    "#D2_SmartLight:d2_smart_light",
17    "#D3_SmartSmoke:d3_smoke",
18  ]
19 }
```



4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、打开串口工具,出现如下结果。

```
Receive Flags is 10
Receive Flags is 10
Receive Flags is 10
ready to OS start
sdk ver:Hi3861V100R001C00SPC025 2020-09-03 18:10:00
FileSystem mount ok.
wifi init success!
hilog will init.
hievent will init.
hievent init success.
hiview init success. Receive Flags is 10
No crash dump found!
Receive Flags is 10
```

7、本章节源文件代码如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "hi_types.h"

#define oneSeconds 1000 * 1000 //延时 1s

#define FLAGS 10 //宏定义标志为 10

osEventFlagsId_t event_flag_id; //事件标志 ID 标识事件标志

/*发送事件*/
void Thread_EventSender(void *arg)
{
    (void)arg;
    while (1)
    {
        osEventFlagsSet(event_flag_id, FLAGS); //设置事件标记
        osThreadYield(); //将当前运行的线程设置为就绪状态，挂起线程
        usleep(oneSeconds); //延时 1s
    }
}

/*接收事件*/
void Thread_EventReceiver(void *arg)
{
    (void)arg;
    hi_u32 flags;
    while (1)
    {
```



```
flags = osEventFlagsWait(event_flag_id, FLAGS, osFlagsWaitAny, osWaitForever); //等待事件标记触发
printf("Receive Flags is %d\n", flags); //打印标志
}

}

/*创建事件*/
static void Event_example(void)
{
    event_flag_id = osEventFlagsNew(NULL); //创建事件标记对象
    if (event_flag_id == NULL)
    {
        printf("Failed to create EventFlags!\n");
    }

    osThreadAttr_t attr;
    attr.attr_bits = 0U; //属性位用于设置 cmsis_os2.h 中的 osStatus_t osThreadJoin (osThreadId_t thread_id); 函数是否能使用, 默认认为 0
    attr.cb_mem = NULL; //控制块的指针, 不操作设置为 NULL
    attr.cb_size = 0U; //控制块的指针大小, 0
    attr.stack_mem = NULL; //任务栈指针, 不操作设置为 NULL
    attr.stack_size = 1024 * 4; //任务栈大小: 8 字节对齐的大小
    attr.priority = 25; //任务的优先级: 25
    attr.name = "Thread_EventSender"; //任务名称
    if (osThreadNew(Thread_EventSender, NULL, &attr) == NULL)
    {
        printf("Failed to create Thread_EventSender!\n");
    }

    attr.name = "Thread_EventReceiver"; //任务名称
    if (osThreadNew(Thread_EventReceiver, NULL, &attr) == NULL)
    {
        printf("Failed to create Thread_EventReceiver!\n");
    }
}

SYS_RUN(Event_example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("a4_event_management"){
    sources = [
        "example_event_management.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

第九章：互斥锁

一、实验目的

- 1、熟悉 HarmonyOS 系统下互斥锁的通信机制
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1、互斥锁的基本概念

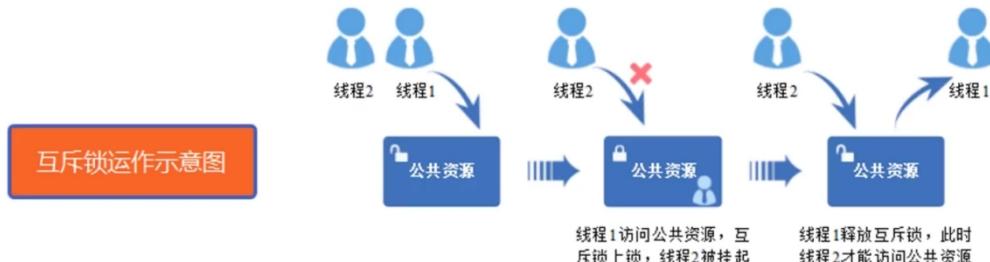
互斥锁又称互斥型信号量，是一种特殊的二值性信号量，用于实现对共享资源的独占式处理。

2、互斥锁的使用方式

- 在任意时刻，互斥锁的状态只有两种：开锁和闭锁。
- 当有任务持有时，互斥锁处于闭锁状态，这个任务获得该互斥锁的所有权。
- 当该任务释放它时，该互斥锁被开锁，任务失去该互斥锁的所有权。
- 当一个任务持有互斥锁时，其他任务将不能再对该互斥锁进行开锁或持有。
- 当一个互斥锁为加锁状态时，此时其他任务如果想访问这个公共资源则会被阻塞，直到互斥锁被持有该锁的任务释放后，其他任务才能重新访问该公共资源，此时互斥锁再次上锁，如此确保同一时刻只有一个任务正在访问这个公共资源，保证了公共资源操作的完整性。

1、互斥锁的运行机制

多任务环境下会存在多个任务访问同一公共资源的场景，而有些公共资源是非共享的，需要任务进行独占式处理。



三、API 的介绍

1、osMutexNew

功能	创建互斥锁，不能在中断服务调用该函数
函数定义	osMutexId_t osMutexNew (const osMutexAttr_t *attr)
参数	attr: 互斥对象的属性
返回	互斥锁 ID

2、osMutexAcquire

功能	函数 osMutexAcquire 一直等待，直到参数 mutex_id 指定的互斥对象可用为止。如果没有其他线程获得互斥锁，该函数立即返回并阻塞互斥锁对象
函数定义	osStatus_t osMutexAcquire (osMutexId_t mutex_id,uint32_t timeout)
参数	mutex_id: 互斥锁 ID timeout: 超时值
返回	0 - 成功, 非 0 - 失败

3、osMutexRelease

功能	释放互斥锁，不能从中断服务例程调用此函数
函数定义	osStatus_t osMutexRelease (osMutexId_t mutex_id)
参数	mutex_id: 互斥锁 ID
返回	0 - 成功, 非 0 - 失败

4、osMutexDelete

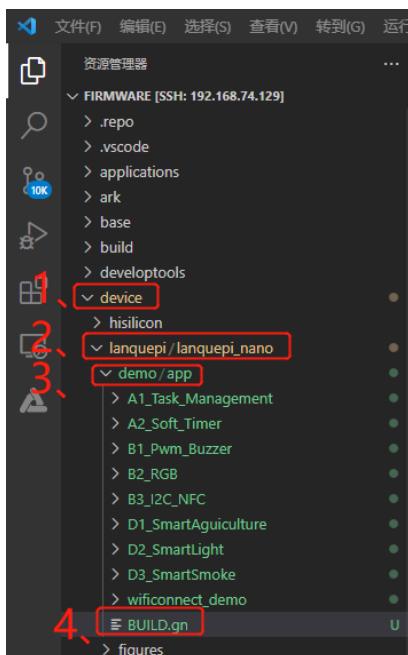
功能	删除互斥锁
函数定义	osStatus_t osMutexDelete (osMutexId_t mutex_id)
参数	mutex_id: 互斥锁 ID
返回	0—成功, 非 0—失败

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将 A5_Mutex:a5_mutex 的注释#删除

```

device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     #"A5_Mutex:a5_mutex",
9     "#A6_Message_Queue:a6_message_queue",
10    "#B1_First_Led:b1_led",
11    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12    "#B3_ADC:b3_adc",
13    "#B4_I2C_NFC:b4_i2c_nfc",
14    "#B5_RGB:b5_rgb",
15    "#D1_SmartAgriculture:e53_ia1_example",
16    "#D2_SmartLight:d2_smart_light",
17    "#D3_SmartSmoke:d3_smoke",
18  ]
19 }

```

```

device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "A5_Mutex:a5_mutex",
9     "#A6_Message_Queue:a6_message_queue",
10    "#B1_First_Led:b1_led",
11    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12    "#B3_ADC:b3_adc",
13    "#B4_I2C_NFC:b4_i2c_nfc",
14    "#B5_RGB:b5_rgb",
15    "#D1_SmartAgriculture:e53_ia1_example",
16    "#D2_SmartLight:d2_smart_light",
17    "#D3_SmartSmoke:d3_smoke",
18  ]
19 }

```

4、打开 VsCode 的终端，输入如下命令，按回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、打开串口工具,出现如下结果：在 Mutex_example 函数中，通过 osMutexNew()函数创建了互斥锁 ID，并创建的三个不同优先级的任务，在第一秒，高优先级和中优先级线程被延迟。因此，低优先级线程可以启动自己的工作，获得互斥锁并在持有它时延迟。在第一秒之后，高优先级和中优先级线程就准备好了。因此高优先级线程获得优先级并尝试获取互斥锁。因为互斥锁已经被低优先级线程所拥有，所以高优先级线程被阻塞，中间优先级线程被执行，并开始执行许多非阻塞的工作，3S 后低优先级释放互斥锁，高优先级线程准备就绪并立即被调度。

```

ready to OS start
sdk ver:Hi3861V100R001C00SPC025 2020-09-03 18:10:00
formatting spiffs...
FileSystem mount ok.
wifi init success!
hilog will init.
hievent will init.
hievent init success.
hiview init success.LowPrioThread is runing.
No crash dump found!
MidPrioThread is runing.
HighPrioThread is runing.
MidPrioThread is runing.
MidPrioThread is runing.
LowPrioThread is runing.
MidPrioThread is runing.
MidPrioThread is runing.
MidPrioThread is runing.
HighPrioThread is runing.
MidPrioThread is runing.
MidPrioThread is runing.
MidPrioThread is runing.
LowPrioThread is runing.
MidPrioThread is runing.
MidPrioThread is runing.

```

7、本章节源文件代码如下：

```

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "hi_types.h"
#include "ohos_init.h"
#include "cmsis_os2.h"

#define oneSeconds 1000 * 1000 //延时 1s

osMutexId_t mutex_id; //互斥体 ID 标识互斥体

void HighPrioThread(void)
{
    usleep(oneSeconds); //延时 1s
    while(1)

```



```
{  
    osMutexAcquire(mutex_id, osWaitForever); //函数 osMutexAcquire 一直等待，直到参数 mutex_id 指定的互斥对象可用为止。如果没有其他线  
程获得互斥锁，该函数立即返回并阻塞互斥锁对象  
    printf("High Thread is runing.\r\n"); //打印到串口观察  
    usleep(3*oneSeconds); //延时三秒  
    osMutexRelease(mutex_id); //释放互斥锁  
}  
}  
  
void MidPrioThread(void)  
{  
    usleep(oneSeconds); //延时 1S  
    while(1)  
    {  
        printf("Mid Thread is runing.\r\n"); //打印到串口观察  
        usleep(oneSeconds); //延时 1S  
    }  
}  
  
void LowPrioThread(void)  
{  
    while(1)  
    {  
        osMutexAcquire(mutex_id, osWaitForever); //函数 osMutexAcquire 一直等待，直到参数 mutex_id 指定的互斥对象可用为止。如果没有其他线  
程获得互斥锁，该函数立即返回并阻塞互斥锁对象  
        printf("Low Thread is runing.\r\n"); //打印到串口观察  
        usleep(3*oneSeconds); //延时 3S  
        osMutexRelease(mutex_id); //释放互斥锁  
    }  
}  
  
void Mutex_example (void)  
{  
    osThreadAttr_t attr;  
    attr.attr_bits = 0U; //属性位用于设置 cmsis_os2.h 中的 osStatus_t osThreadJoin (osThreadId_t thread_id); 函数是否能使用。  
默认为 0  
    attr.cb_mem = NULL; //控制块的指针，不操作设置为 NULL  
    attr.cb_size = 0U; //控制块的指针大小，0  
    attr.stack_mem = NULL; //任务栈指针，不操作设置为 NULL  
    attr.stack_size = 1024*4; //任务栈大小：8 字节对齐的大小  
  
    attr.name = "HighPrioThread"; //任务名称  
    attr.priority = 24; //任务的优先级：24  
    if (osThreadNew((osThreadFunc_t)HighPrioThread, NULL, &attr) == NULL)  
    {  
        printf("Failed to create HighPrioThread!\n");  
    }  
}
```



```
attr.name = "MidPrioThread"; //任务名称
attr.priority = 25;           //任务的优先级: 25
if (osThreadNew((osThreadFunc_t)MidPrioThread, NULL, &attr) == NULL)
{
    printf("Failed to create MidPrioThread!\n");
}

attr.name = "LowPrioThread"; //任务名称
attr.priority = 26;          //任务的优先级: 26
if (osThreadNew((osThreadFunc_t)LowPrioThread, NULL, &attr) == NULL)
{
    printf("Failed to create LowPrioThread!\n");
}

mutex_id = osMutexNew(NULL); //创建互斥锁
if (mutex_id == NULL)
{
    printf("Failed to create Mutex!\n");
}
}

SYS_RUN(Mutex_example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("a5_mutex"){
    sources = [
        "example_mutex.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

第十章：消息队列

一、实验目的

- 1、熟悉 HarmonyOS 系统下消息队列的通信机制
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1、消息队列的基本概念

消息队列，是一种常用于任务间通信的数据结构，实现了接收来自任务或中断的不固定长度的消息，并根据不同的接口选择传递消息是否存放在自己空间。任务能够从队列里面读取消息，当队列中的消息是空时，挂起读取任务；当队列中有新消息，挂起的读取任务被唤醒并处理新消息。

2、消息队列的特点

用户在处理业务时，消息队列提供了异步处理机制，允许将一个消息放入队列，但并不立即处理它，同时队列还能起到缓冲消息作用。LiteOS 中使用队列数据结构实现任务异步通信工作，具有如下特性：

- 消息以先进先出方式排队，支持异步读写工作方式。
- 读队列和写队列都支持超时机制。
- 发送消息类型由通信双方约定，可以允许不同长度（不超过队列节点最大值）消息。
- 一个任务能够从任意一个消息队列接收和发送消息。
- 多个任务能够从同一消息队列接收和发送消息。
- 当队列使用结束后，如果是动态申请的内存，需要通过释放内存函数回收。

2、消息队列的运行机制

• 创建队列时，根据用户传入队列长度和消息节点大小来开辟相应的内存空间以供该队列使用，返回队列 ID。

• 在队列控制块中维护一个消息头节点位置 Head 和一个消息尾节点位置 Tail 来表示当前队列中消息存储情况。Head 表示队列中被占用消息的起始位置。Tail 表示队列中被空闲消息的起始位置。刚创建时 Head 和 Tail 均指向队列起始位置。

- 写队列时，根据 Tail 找到被占用消息节点末尾的空闲节点作为数据写入对象。
- 读队列时，根据 Head 找到最先写入队列中的消息节点进行读取。
- 删除队列时，根据传入的队列 ID 寻找到对应的队列，把队列状态置为未使用，释放原队列所占的空间，对应的队列控制头置为初始状态。

三、API 的介绍

1、osMessageQueueNew

功能	创建消息队列，不能在中断服务调用该函数
函数定义	osMessageQueueId_t osMessageQueueNew (uint32_t msg_count,uint32_t msg_size,const osMessageQueueAttr_t *attr)
参数	msg_count: 队列中的最大消息数 msg_size: 最大消息大小(以字节为单位) attr: 消息队列属性;空:默认值
返回	消息队列 ID

2、osMessageQueuePut

功能	发送消息，如果参数 timeout 设置为 0，可以从中断服务例程调用
----	-------------------------------------

函数定义	osStatus_t osMessageQueuePut (osMessageQueueId_t mq_id,const void *msg_ptr,uint8_t msg_prio,uint32_t timeout)
参数	<p>mq_id: 由 osMessageQueueNew 获得的消息队列 ID msg_ptr: 要发送的消息 msg_prio: 指优先级 timeout: 超时值</p>
返回	0 - 成功, 非 0 - 失败

3、osMessageQueueGet

功能	获取消息, 如果参数 timeout 设置为 0, 可以从中断服务例程调用
函数定义	osStatus_t osMessageQueueGet (osMessageQueueId_t mq_id,void *msg_ptr,uint8_t *msg_prio,uint32_t timeout)
参数	<p>mq_id: 由 osMessageQueueNew 获得的消息队列 ID msg_ptr: 指针指向队列中获取消息的缓冲区指针 msg_prio: 指优先级 timeout: 超时值</p>
返回	0 - 成功, 非 0 - 失败

4、osMessageQueueDelete

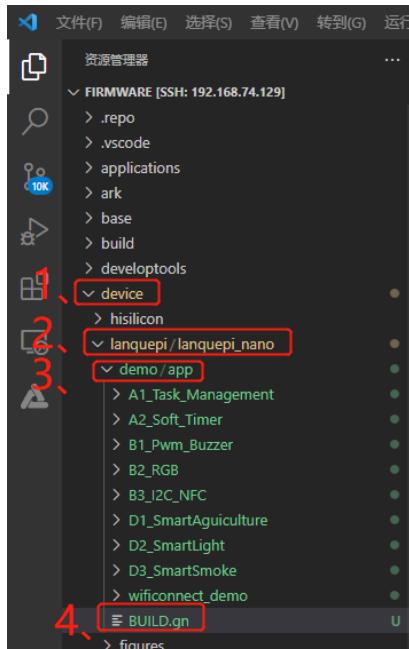
功能	删除消息队列
函数定义	osStatus_t osMessageQueueDelete (osMessageQueueId_t mq_id)
参数	mq_id: 消息队列 ID
返回	0—成功, 非 0—失败

四、实验步骤

1、打开 VsCode

说明: 确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径, 打开 BUILD.gn 文件。



3、将 A6_Message_Queue:a6_message_queue 的注释#删除



```
device > lanquepi > lanquepi_nano > demo > app >  ≡ BUILD.gn
1   import("//build/lite/config/component/lite_component.gni")
2   lite_component("app") {
3     features = [
4       "#A1_Task_Management:a1_task_management",
5       "#A2_Soft_Timer:a2_soft_timer",
6       "#A3_Semaphore:a3_semaphore",
7       "#A4_Event_Management:a4_event_management",
8       "#A5_Mutex:a5_mutex",
9       "#A6_Message_Queue:a6_message_queue",
10      "#B1_First_Led:b1_led",
11      "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12      "#B3_ADC:b3_adc",
13      "#B4_I2C_NFC:b4_i2c_nfc",
14      "#B5_RGB:b5_rgb",
15      "#D1_SmartAgriulture:d1_smart_agriculture",
16      "#D2_SmartLight:d2_smart_light",
17      "#D3_SmartSmoke:d3_smart_smoke",
18    ]
19  }
```

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3     features = [
4         "#A1_Task_Management:a1_task_management",
5         "#A2_Soft_Timer:a2_soft_timer",
6         "#A3_Semaphore:a3_semaphore",
7         "#A4_Event_Management:a4_event_management",
8         "#A5_Mutex:a5_mutex",
9         "A6_Message_Queue:a6_message_queue",
10        "#B1_First_Led:b1_led",
11        "#B2_Pwm_Buzzer:b2_pwm_buzzer",
12        "#B3_ADC:b3_adc",
13        "#B4_I2C_NFC:b4_i2c_nfc",
14        "#B5_RGB:b5_rgb",
15        "#D1_SmartAguiculture:d1_smart_agriculture",
16        "#D2_SmartLight:d2_smart_light",
17        "#D3_SmartSmoke:d3_smart_smoke",
18    ]
19 }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、打开串口工具,出现如下结果:在 Message_example 函数中,通过 osMessageQueueNew() 函数创建了消息队列 ID, Thread_MsgQueue1() 函数中通过 osMessageQueuePut() 函数向消息队列中发送消息。在 Thread_MsgQueue2() 函数中通过 osMessageQueueGet() 函数读取消息队列中的消息并打印出来。示例代码编译烧录代码后,按下开发板的 RESET 按键,通过串口助手查看日志,会打印从消息队列中获取的消息。



7、本章节源文件代码如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"

#define oneSeconds 1000 * 1000 //延时 1s

#define MESSAGE_QUEUE_OBJ 16 //宏定义消息队列对象

typedef struct
{
    char *Buf;
    uint8_t Index;
} Message_Queue_OBJ_Temp; //定义结构体（需要传消息的 BUF, ）

Message_Queue_OBJ_Temp msg; //声明结构体 msg

osMessageQueueId_t mid_MsgQueue; //消息队列 ID 标识消息队列

void Thread_MsgQueue1(void *argument)

{
    (void)argument;

    msg.Buf = "Hello Lanquepi-M_Nano!"; //需要传递的消息

    msg.Index = 0U; //消息下标

    while (1)
    {
        osMessageQueuePut(mid_MsgQueue, &msg, 0U, 0U); //发送消息
        osThreadYield(); //挂起线程
        osDelay(oneSeconds); //延时 1s
    }
}

void Thread_MsgQueue2(void *argument)

{
    (void)argument;
    osStatus_t status; //定义状态用于检测

    while (1)
    {
        status = osMessageQueueGet(mid_MsgQueue, &msg, NULL, 0U); //获取消息，并将返回值传给 status
        if (status == osOK) //如果返回值为 0，即表示成功
        {
            printf("Message Queue Get msg:%s\n", msg.Buf); //打印需要传递的消息
        }
    }
}

static void Message_example(void)

{
    mid_MsgQueue = osMessageQueueNew(MESSAGE_QUEUE_OBJ, 100, NULL); //创建消息队列
    if (mid_MsgQueue == NULL)
    {

```



```
    printf("Failed to create Message Queue!\n");
}

osThreadAttr_t attr;
attr.attr_bits = 0U; //属性应用于设置 cmsis_os2.h 中的 osStatus_t osThreadJoin (osThreadId_t thread_id); 函数是否能使用, 默认为 0

attr.cb_mem = NULL; //控制块的指针, 不操作设置为 NULL
attr.cb_size = 0U; //控制块的指针大小, 0
attr.stack_mem = NULL; //任务栈指针, 不操作设置为 NULL
attr.stack_size = 1024 * 10; //任务栈大小: 8 字节对齐的大小
attr.priority = 25; //任务的优先级: 25
attr.name = "Thread_MsgQueue1"; //任务名称

if (osThreadNew(Thread_MsgQueue1, NULL, &attr) == NULL)
{
    printf("Failed to create Thread_MsgQueue1!\n");
}

attr.name = "Thread_MsgQueue2";
if (osThreadNew(Thread_MsgQueue2, NULL, &attr) == NULL)
{
    printf("Failed to create Thread_MsgQueue2!\n");
}
}

APP_FEATURE_INIT(Message_example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("a6_message_queue"){
    sources = [
        "example_message_queue.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

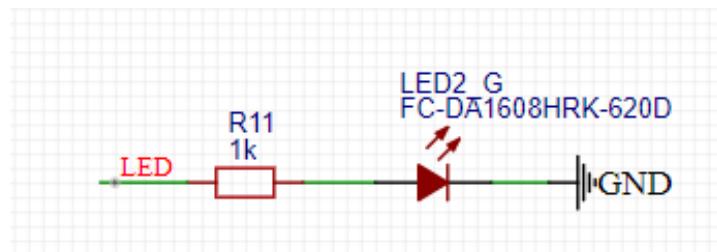
第十一章：点亮第一个 LED 灯

一、实验目的

- 1、熟悉 HarmonyOS 系统下 GPIO 口的函数使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

使用板载的 LED 来验证 GPIO 的输出功能，在 Lanquepi-M_Nano 开发板上 LED 的连接电路图如下图所示，LED 的控制引脚与主控芯片的 GPIO_7 连接，所以需要编写软件去控制 GPIO_7 输出高低电平实现 LED 灯的亮灭。



三、API 的介绍

1、hi_gpio_init

功能	初始化 GPIO 外设
函数定义	hi_u32 hi_gpio_init(hi_void)
参数	无
返回	错误码

2、hi_io_set_func

功能	设置 GPIO 引脚功能
函数定义	hi_u32 hi_io_set_func(hi_io_name id, hi_u8 val)
参数	id: 表示 GPIO 引脚号 val: 表示 IO 复用功能
返回	错误码

3、hi_gpio_set_dir

功能	设置 GPIO 输出方向
函数定义	hi_u32 hi_gpio_set_dir(hi_gpio_idx id, hi_gpio_dir dir)
参数	id: 表示 GPIO 引脚号 dir: 表示 GPIO 输出方向
返回	错误码

4、hi_gpio_set_output_val

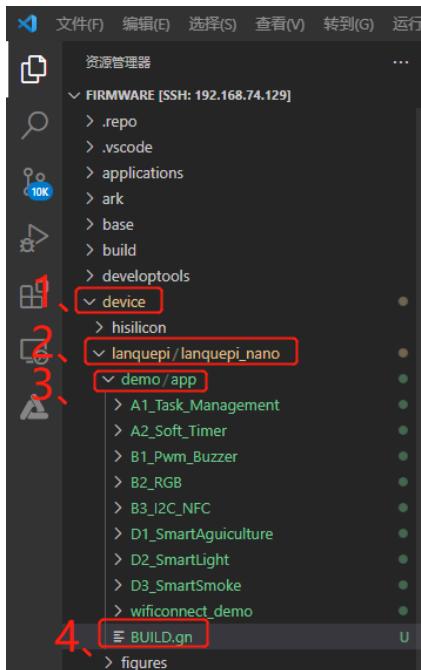
功能	设置 GPIO 引脚输出电平值
函数定义	hi_u32 hi_gpio_set_output_val(hi_gpio_idx id, hi_gpio_value val)
参数	id: 表示 GPIO 引脚号 val: 表示 GPIO 输出电平值
返回	错误码

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将的 B1_First_Led:b1_led 注释#删除

```

1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     # "A1_Task_Management:a1_task_management",
5     # "A2_Soft_Timer:a2_soft_timer",
6     # "A3_Semaphore:a3_semaphore",
7     # "A4_Event_Management:a4_event_management",
8     # "A5_Mutex:a5_mutex",
9     # "B1_First_Led:b1_led",
10    # "B2_Pwm_Buzzer:b2_pwm_buzzer",
11    # "B4_RGB:b4_rgb",
12    # "B3_I2C_NFC:b3_i2c_nfc",
13    # "D1_SmartAguiculture:d1_smart_aguiculture",
14    # "D2_SmartLight:d2_smart_light",
15    # "D3_SmartSmoke:d3_smart_smoke",
16  ]
17 }

```

```

device > lanquepi > lanquepi_nano > demo > app > ┌ BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     # "A1_Task_Management:a1_task_management",
5     # "A2_Soft_Timer:a2_soft_timer",
6     # "A3_Semaphore:a3_semaphore",
7     # "A4_Event_Management:a4_event_management",
8     # "A5_Mutex:a5_mutex",
9     "B1_First_Led:b1_led",
10    # "B2_Pwm_Buzzer:b2_pwm_buzzer",
11    # "B4_RGB:b4_rgb",
12    # "B3_I2C_NFC:b3_i2c_nfc",
13    # "D1_SmartAguiculture:d1_smart_aguiculture",
14    # "D2_SmartLight:d2_smart_light",
15    # "D3_SmartSmoke:d3_smart_smoke",
16  ]
17 }

```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、出现如下结果：按下开发板的 RESET 按键，开发板的 LED 灯开始每秒闪烁。



7、本章节源文件代码如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "hi_gpio.h"
#include "hi_io.h"
#include "hi_pwm.h"

#define oneSeconds 1000 * 1000 //延时 1s

/*创建 LED 任务*/
static void LedTask(void)
{
    hi_gpio_init(); //初始化 GPIO 口
    hi_io_set_func(HI_GPIO_IDX_7,HI_IO_FUNC_GPIO_7_GPIO); //GPIO 口的功能设置为普通 IO 口
    hi_gpio_set_dir(HI_GPIO_IDX_7,HI_GPIO_DIR_OUT); //GPIO 的方向设置为输出

    while (1)
    {
        hi_gpio_set_ooutput_val(HI_GPIO_IDX_7,1); //GPIO 输出的电平设置为高电平
        usleep(oneSeconds); //延时一秒
        hi_gpio_set_ooutput_val(HI_GPIO_IDX_7,0); //GPIO 输出的电平设置为低电平
        usleep(oneSeconds); //延时一秒
    }
}

/*创建线程*/
static void ledExample(void)
{
    // 指定线程的属性
    osThreadAttr_t attr;
```



```
attr.name = "LedTask";           //线程名字
attr.attr_bits = 0U;             //线程堆栈线程属性位
attr.cb_mem = NULL;              //线程控制块的内存
attr.cb_size = 0U;               //线程控制块的内存大小
attr.stack_mem = NULL;            //线程堆栈的内存
attr.stack_size = 512;             //线程堆栈的大小
attr.priority = osPriorityNormal; //线程优先级
if (osThreadNew((osThreadFunc_t)LedTask, NULL, &attr) == NULL)
{
    printf("Failed to create LedTask!\r\n");
}
SYS_RUN(ledExample);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("b1_led"){
    sources = [
        "example_led.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

第十二章：PWM 的使用

一、实验目的

- 1、熟悉 HarmonyOS 系统下 GPIO 以及 PWM 的函数使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

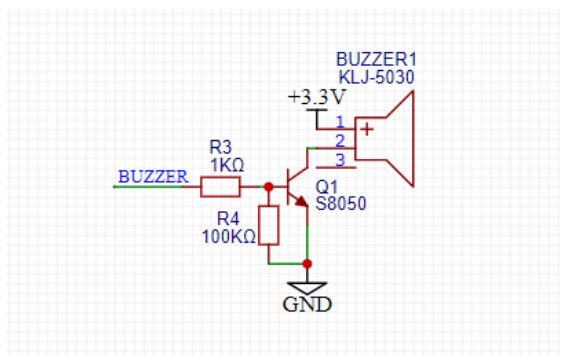
二、实验原理

脉冲宽度调制（PWM），是英文“Pulse Width Modulation”的缩写，简称脉宽调试。是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的方法。广泛应用于从测量、通信到功率控制与变换的许多领域中。

脉冲调制有两个重要的参数，第一个就是输出频率，频率越高，则模拟的效果越好。第二个就是占空比。占空比就是改变输出模拟效果的电压大小。占空比越大则模拟出的电压越大。

本章节，将使用 PWM 波去驱动无源蜂鸣器，无源内部不带震荡源，所以如果用直流信号无法令其鸣叫，必须用 2K-5K 的方波去驱动它。

使用板载的无源蜂鸣器来验证 PWM 的输出功能，在 Lanquepi-M_Nano 开发板上无源蜂鸣器的连接电路图如下图所示，无源蜂鸣器的控制引脚与主控芯片的 GPIO_8 连接，所以需要编写软件去控制 GPIO_8 输出 PWM 实现无源蜂鸣器的驱动。



三、API 的介绍

1、hi_gpio_init

功能	初始化 GPIO 外设
函数定义	hi_u32 hi_gpio_init(hi_void)
参数	无
返回	错误码

2、hi_io_set_func

功能	设置 GPIO 引脚功能
函数定义	hi_u32 hi_io_set_func(hi_io_name id, hi_u8 val)
参数	id: 表示 GPIO 引脚号 val: 表示 IO 复用功能
返回	错误码

3、hi_gpio_set_dir

功能	设置 GPIO 输出方向
函数定义	hi_u32 hi_gpio_set_dir(hi_gpio_idx id, hi_gpio_dir dir)
参数	id: 表示 GPIO 引脚号

	dir: 表示 GPIO 输出方向
返回	错误码

4、hi_pwm_init

功能	PWM 的初始化
函数定义	hi_u32 hi_pwm_init(hi_pwm_port port)
参数	port: 表示 PWM 端口号
返回	错误码

5、hi_pwm_set_clock

功能	PWM 的时钟设置
函数定义	hi_u32 hi_pwm_set_clock(hi_pwm_clk_source clk_type)
参数	clk_type: 表示 PWM 时钟类型，默认为 160M 时钟
返回	错误码

6、hi_pwm_start

功能	根据输入参数输出 PWM 信号
函数定义	hi_u32 hi_pwm_start(hi_pwm_port port, hi_u16 duty, hi_u16 freq)
参数	port: PWM 端口号 duty: 占空比 freq: 分频倍数
返回	错误码

6、hi_pwm_stop

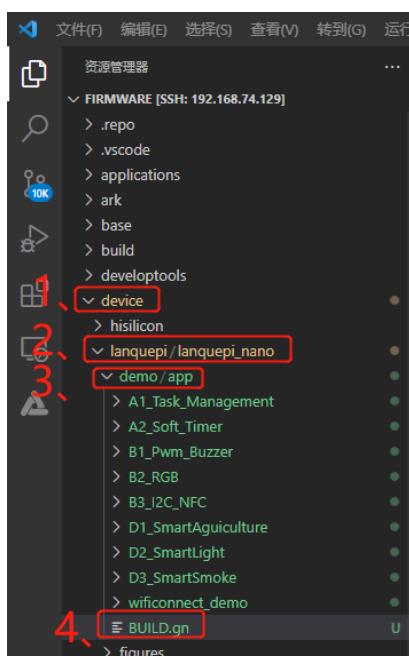
功能	关闭 PWM 输出信号
函数定义	hi_u32 hi_pwm_stop(hi_pwm_port port)
参数	port: 初始化 PWM 端口号
返回	错误码

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。





3、将的 B1_First_Led:b1_led 注释#删除

```
device > lanquepi > lanquepi_nano > demo > app >  BUILD.gn
1  import("//build/lite/config/component/lite_component.gni")
2  lite_component("app") {
3      features = [
4          "#A1_Task_Management:a1_task_management",
5          "#A2_Soft_Timer:a2_soft_timer",
6          "#A3_Semaphore:a3_semaphore",
7          "#A4_Event_Management:a4_event_management",
8          "#A5_Mutex:a5_mutex",
9          "#B1_First_Led:b1_led",
10         "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11         "#B4_RGB:b4_rgb",
12         "#B3_I2C_NFC:b3_i2c_nfc",
13         "#D1_SmartAguiculture:d1_smart_aguiculture",
14         "#D2_SmartLight:d2_smart_light",
15         "#D3_SmartSmoke:d3_smart_smoke",
16     ]
17 }
```

```
device > lanquepi > lanquepi_nano > demo > app >  BUILD.gn
1  import("//build/lite/config/component/lite_component.gni")
2  lite_component("app") {
3      features = [
4          "#A1_Task_Management:a1_task_management",
5          "#A2_Soft_Timer:a2_soft_timer",
6          "#A3_Semaphore:a3_semaphore",
7          "#A4_Event_Management:a4_event_management",
8          "#A5_Mutex:a5_mutex",
9          "#B1_First_Led:b1_led",
10         "B2_Pwm_Buzzer:b2_pwm_buzzer",
11         "#B4_RGB:b4_rgb",
12         "#B3_I2C_NFC:b3_i2c_nfc",
13         "#D1_SmartAguiculture:d1_smart_aguiculture",
14         "#D2_SmartLight:d2_smart_light",
15         "#D3_SmartSmoke:d3_smart_smoke",
16     ]
17 }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、出现如下结果：按下开发板的 RESET 按键，开发板的蜂鸣器开始滴答滴答的响。

7、本章节源文件代码如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "hi_gpio.h"
#include "hi_io.h"
#include "hi_pwm.h"

#define one_MS 1U           //1U 等于 10ms
#define oneMS 1000          //延时 1ms
#define oneSeconds 1000 * 1000 //延时 1s

/*创建蜂鸣器任务*/
static void BuzzerTask(void)
{
    hi_gpio_init();           //GPIO 初始化
    hi_pwm_init(HI_PWM_PORT_PWM1); //PWM1 初始化
```



```
hi_io_set_func(HI_GPIO_IDX_8,HI_IO_FUNC_GPIO_8_PWM1_OUT); //引脚 GPIO8 设置为 PWM1 功能
hi_gpio_set_dir(HI_GPIO_IDX_8,HI_GPIO_DIR_OUT); //GPIO8 方向设置成输出
hi_pwm_set_clock(PWM_CLK_160M); //PWM 时钟设置为 160M
while (1)
{
    hi_pwm_start(HI_PWM_PORT_PWM1, 65400/2, 65400); //开启 PWM
    usleep(100*oneMS); //延时 10ms
    hi_pwm_stop(HI_PWM_PORT_PWM1); //关闭 PWM
    usleep(100*oneMS); //延时 10ms
}
/*创建线程*/
static void Buzzer_Example(void)
{
    // 指定线程的属性
    osThreadAttr_t attr;
    attr.name = "BuzzerTask"; //线程名字
    attr.attr_bits = 0U; //线程堆栈线程属性位
    attr.cb_mem = NULL; //线程控制块的内存
    attr.cb_size = 0U; //线程控制块的内存大小
    attr.stack_mem = NULL; //线程堆栈的内存
    attr.stack_size = 512; //线程堆栈的大小
    attr.priority = osPriorityNormal; //线程优先级
    if (osThreadNew((osThreadFunc_t)BuzzerTask, NULL, &attr) == NULL)
    {
        printf("Failed to create BuzzerTask!\r\n");
    }
}
SYS_RUN(Buzzer_Example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("b2_pwm_buzzer"){
    sources = [
        "example_pwm_buzzer.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

第十三章：ADC 的使用

一、实验目的

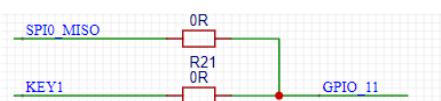
- 1、熟悉 HarmonyOS 系统下 ADC 函数的使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

ADC(Analog-to-Digital Converter)，即模拟-数字转换器，可以将连续变化的模拟信号转换为离散的数字信号，进而使用数字电路进行处理，称之为数字信号处理。

本章节，将使用 ADC 去读取 GPIO11 引脚的电压，GPIO11 同时是 Lanquepi-M_Nano 开发板上的按键 Key1，因此通过操作按键，即可读取出高低电平，并将读取到的电压值打印到串口上。

通过查看芯片手册可知 GPIO_11 对应的是 ADC_5。



引脚	Uart	SPI	ADC	PWM	I2S	SDIO	I2C
GPIO_04	LOG_RXD_0		ADC_1				
GPIO_02				PWM_2	MCK_0		
GPIO_05	RXD_1	CSI_0	ADC_2	PWM_2	TX_0		
GPIO_06	TXD_1	CLK_0		PWM_3			
GPIO_14	LOG_RXD_0/CTS_2			PWM_5	RX_0	D1	SCL_0
GPIO_11	TXD_2	RXD_0	ADC_5	PWM_2	CLK_0	CMD	
GPIO_12	RXD_2	CSI_0	ADC_0	PWM_3	WS_0	CLK	
GPIO_13	LOG_TXD_0/RTS_2		ADC_6	PWM_4		D0	SDA_0

三、API 的介绍

1、hi_adc_read

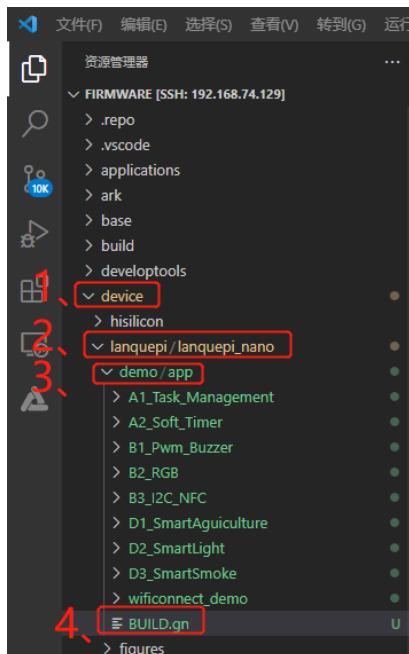
功能	初始化 GPIO 外设
函数定义	<pre>hi_u32 hi_adc_read(hi_adc_channel_index channel, hi_u16 *data, hi_adc_equ_model_sel equ_model,hi_adc_cur_bais cur_bais, hi_u16 delay_cnt);</pre>
参数	<p>channel: 表示 ADC 通道 data: 表示指向存储读取数据的地址的指针 equModel: 表示平均算法的次数 curBais: 表示模拟功率控制模式 rstCnt: 指示从重置到转换开始的时间计数。一次计数等于 334 纳秒。值的范围必须从 0 到 0xFF</p>
返回	错误码

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将的 B3_ADC:b3_adc 注释#删除

```
device > lanquepi > lanquepi_nano > demo > app > └─ BUILD.gn
1  import("//build/lite/config/component/lite_component.gni")
2  lite_component("app") {
3    features = [
4      "#A1_Task_Management:a1_task_management",
5      "#A2_Soft_Timer:a2_soft_timer",
6      "#A3_Semaphore:a3_semaphore",
7      "#A4_Event_Management:a4_event_management",
8      "#A5_Mutex:a5_mutex",
9      "#B1_First_Led:b1_led",
10     "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11     "#B3_ADC:b3_adc", //注释掉
12     "#B4_RGB:b4_rgb",
13     "#B3_I2C_NFC:b3_i2c_nfc",
14     "#D1_SmartAgriculture:d1_smart_agriculture",
15     "#D2_SmartLight:d2_smart_light",
16     "#D3_SmartSmoke:d3_smart_smoke",
17   ]
18 }
```

```
device > lanquepi > lanquepi_nano > demo > app > └─ BUILD.gn
1  import("//build/lite/config/component/lite_component.gni")
2  lite_component("app") {
3    features = [
4      "#A1_Task_Management:a1_task_management",
5      "#A2_Soft_Timer:a2_soft_timer",
6      "#A3_Semaphore:a3_semaphore",
7      "#A4_Event_Management:a4_event_management",
8      "#A5_Mutex:a5_mutex",
9      "#B1_First_Led:b1_led",
10     "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11     "B3_ADC:b3_adc", //注释掉
12     "#B4_RGB:b4_rgb",
13     "#B3_I2C_NFC:b3_i2c_nfc",
14     "#D1_SmartAgriculture:d1_smart_agriculture",
15     "#D2_SmartLight:d2_smart_light",
16     "#D3_SmartSmoke:d3_smart_smoke",
17   ]
18 }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```



5、按照烧录章节的烧录过程进行烧录。

6、出现如下结果：打开串口，使用板载用户按键 Key1 来模拟 GPIO 口电压的变化，所以需要编写软件去读取 ADC Channel 5 的电压，程序设计时先将 GPIO_11 上拉，使 GPIO_11 的电压一直处于高电平，当按键按下时 GPIO_11 接地，此时 GPIO_11 的电压变为 0 V。

```
*ADC_example*
voltage:3.296V
*ADC_example*
voltage:3.298V
*ADC_example*
voltage:3.298V
*ADC_example*
voltage:3.298V
*ADC_example*
voltage:3.298V
*ADC_example*
voltage:3.298V
*ADC_example*
voltage:0.193V
*ADC_example*
voltage:0.193V
*ADC_example*
voltage:3.298V
*ADC_example*
voltage:3.298V
*ADC_example*
voltage:0.195V
*ADC_example*
voltage:0.195V
*ADC_example*
voltage:3.298V
```

7、本章节源文件代码如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include "ohos_init.h"
#include "ohos_types.h"
#include "cmsis_os2.h"
#include "hi_task.h"
#include "hi_gpio.h"
#include "hi_io.h"
#include "hi_adc.h"
#include "hi_types.h"

#define oneSeconds 1000 * 1000 //延时 1s

/*获取电压值函数*/
static float GetVoltage(void)
{
    hi_u32 read_Temp;      //unsigned int
    hi_u16 data;          //unsigned short
    /*
    该函数通过使用 hi_adc_read() 函数来读取 ADC_CHANNEL_5 的数值存储在 data 中,
    HI_ADC_EQU_MODEL_8 表示 8 次平均算法模式,
    HI_ADC_CUR_BAIS_DEFAULT 表示默认的自动识别模式,
    最后通过 data * 1.8 * 4 / 4096.0 计算出实际的电压值。
    */
    read_Temp = hi_adc_read(HI_ADC_CHANNEL_5,&data,HI_ADC_EQU_MODEL_8,HI_ADC_CUR_BAIS_DEFAULT,0xFF);
    return (float) data * 1.8 * 4 / 4096.0;
}

static void ADCTask(void)
{
```



```
float voltage_Value;

hi_io_set_pull(HI_IO_NAME_GPIO_11, HI_IO_PULL_UP);      //上拉,让按键未按下时 GPIO_11 保持高电平状态

while (1)

{

    printf("*ADC_example*\r\n");           //打印*ADC_example*

    voltage_Value = GetVoltage();          //获取电压值

    printf("voltage: %.3fV\r\n", voltage_Value); //打印获取的电压值

    usleep(oneSeconds);                  //延时 1s

}

}

static void ADC_Example(void)

{

    osThreadAttr_t attr;

    attr.name = "ADCTask";           //线程名字

    attr.attr_bits = 0U;             //线程堆栈线程属性位

    attr.cb_mem = NULL;             //线程控制块的内存

    attr.cb_size = 0U;              //线程控制块的内存大小

    attr.stack_mem = NULL;           //线程堆栈的内存

    attr.stack_size = 1024*8;        //线程堆栈的大小

    attr.priority = 24;              //线程优先级

    if (osThreadNew((osThreadFunc_t)ADCTask, NULL, &attr) == NULL)

    {

        printf("Failed to create ADCTask!\n");

    }

}

SYS_RUN(ADC_Example);
```

8、本章节 BUILD.gn 目录如下：

```
static_library("b2_pwm_buzzer"){

    sources = [
        "example_pwm_buzzer.c"
    ]

    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
```

第十四章：NFC 标签 NT3H 的使用

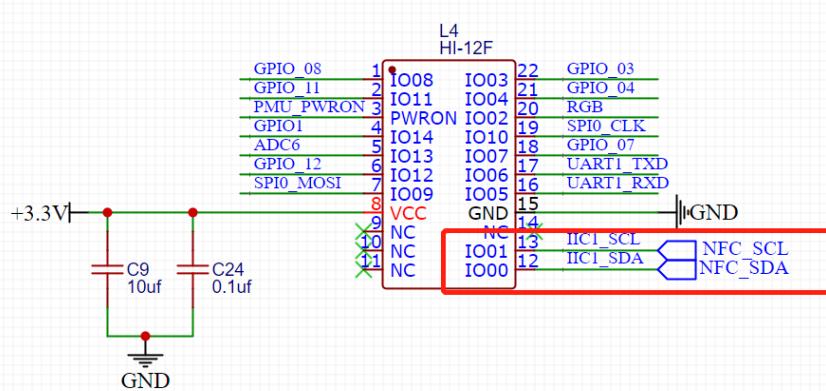
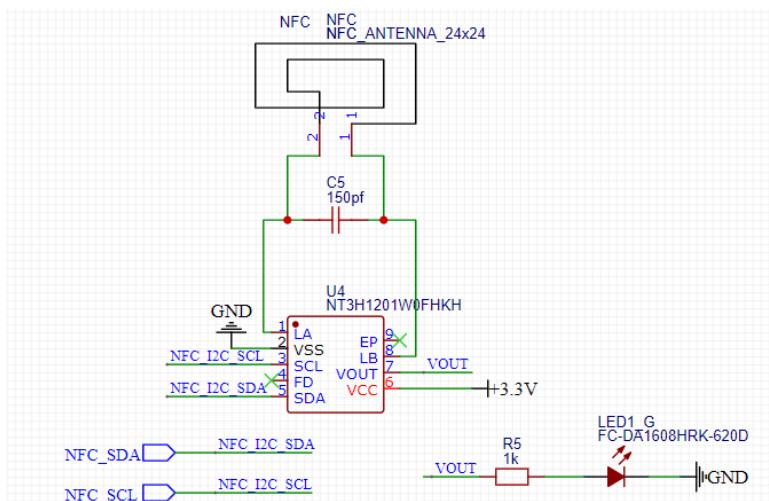
一、实验目的

- 1、熟悉 HarmonyOS 系统下 IIC 函数以及 NFC 标签 NT3H 的使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1.I2C 的简介：I2C(Inter-Integrated Circuit, 内部集成电路) 总线是一种由飞利浦 Philip 公司开发的串行总线。是两条串行的总线，它由一根数据线 (SDA) 和一根 时钟线 (SCL) 组成。两条线都需要上拉电阻。I2C 总线上可以接多个 I2C 设备，每个器件都有一个唯一的地址识别。同一时间只能有一个主设备，其他为从设备。通常 MCU 作为主设备控制，外设作为从设备。即模拟-数字转换器，可以将连续变化的模拟信号转换为离散的数字信号，进而使用数字电路进行处理，称之为数字信号处理。

本章节，将使用 I2C 去读写 NFC，通过查看芯片手册和原理图可知 GPIO_0 对应的是 IIC_SDA,GPIO_1 对应的是 IIC_SCL。



3、NFC 标签 NT3H1201 的简介

- NT3H1x01W0FHK NFC 芯片，是一款简单，低成本的 NFC 标签。特点：
- 工作频率：13.56MHz；
- NT3H1101（NT3H1201）支持接触式和非接触式接口，IIC 从机接口支持标准模式（100KHz）和高速模式（高达 400KHz）；



- 用户读写区：1904 bytes;
- SRAM：64 bytes;
- NT3H1101（NT3H1201）NFC 标签可直接作为标准 IIC EEPROM 使用；
- 外部连接板载 NFC 射频天线。

三、API 的介绍

1、hi_i2c_init

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_init(hi_i2c_idx id, hi_u32 baudrate)
参数	id: I2C 设备 ID baudrate: I2C 频率
返回	错误码

2、hi_i2c_write

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_write(hi_i2c_idx id, hi_u16 device_addr, const hi_i2c_data *i2c_data)
参数	id: I2C 设备 ID deviceAddr: I2C 设备地址 i2cData: 表示写入的数据
返回	错误码

3、hi_i2c_read

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_read(hi_i2c_idx id, hi_u16 device_addr, const hi_i2c_data *i2c_data)
参数	id: I2C 设备 ID deviceAddr: I2C 设备地址 i2cData: 表示要读取的数据指向的指针
返回	错误码

4、hi_i2c_set_baudrate

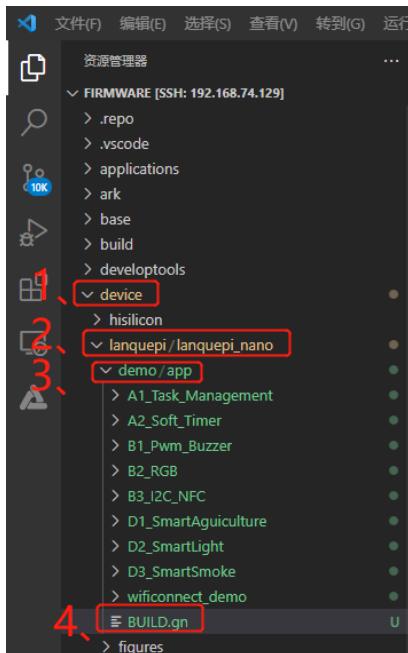
功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_set_baudrate(hi_i2c_idx id, hi_u32 baudrate)
参数	id: I2C 设备 ID baudrate: I2C 频率
返回	错误码

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 VScode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将的 B4_I2C_NFC:b4_i2c_nfc 注释#删除

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#B1_First_Led:b1_led",
10    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11    "#B3_ADC:b3_adc",
12    #"B4_I2C_NFC:b4_i2c_nfc",
13    "#B5_RGB:b5_rgb",
14    "#D1_SmartAguiculture:d1_smart_aguiculture",
15    "#D2_SmartLight:d2_smart_light",
16    "#D3_SmartSmoke:d3_smart_smoke",
17  ]
18 }
```

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#B1_First_Led:b1_led",
10    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11    "#B3_ADC:b3_adc",
12    "B4_I2C_NFC:b4_i2c_nfc",
13    "#B5_RGB:b5_rgb",
14    "#D1_SmartAguiculture:d1_smart_aguiculture",
15    "#D2_SmartLight:d2_smart_light",
16    "#D3_SmartSmoke:d3_smart_smoke",
17  ]
18 }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

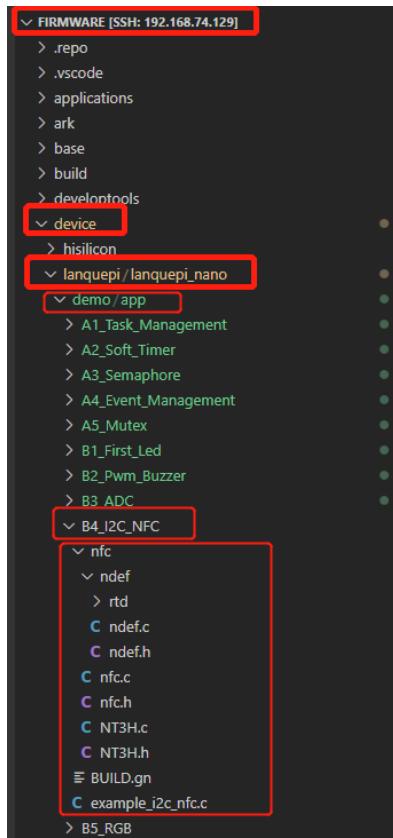
6、出现如下图结果：打开串口会出现提示请使用手机靠近 NFC 标签。当手机靠近 NFC 标签时，手机上则会跳出相应信息，如图二所示。



```
i2c
=====
*****I2C_NFC_example*****
=====
Please use the mobile phone with NFC function close to the development board!
=====
*****I2C_NFC_example*****
=====
Please use the mobile phone with NFC function close to the development board!
=====
*****I2C_NFC_example*****
=====
Please use the mobile phone with NFC function close to the development board!
=====
*****I2C_NFC_example*****
=====
Please use the mobile phone with NFC function close to the development board!
```



7、本章节源文件代码目录如下：



9、本章节 BUILD.gn 目录如下：

```

static_library("b4_i2c_nfc"){
  sources = [
    "nfc/NT3H.c",
    "nfc/nfc.c",
    "nfc/ndef/rtd/nfcForum.c",
    "nfc/ndef/rtd/rtdText.c",
    "nfc/ndef/rtd/rtdUri.c",
    "nfc/ndef/ndef.c",
    "example_i2c_nfc.c"
  ]
  cflags = [ "-Wno-unused-variable" ]
  cflags += [ "-Wno-unused-but-set-variable" ]
  cflags += [ "-Wno-unused-parameter" ]
  include_dirs = [
    "./include",
    "//utils/native/lite/include",
    "//device/lanquepi/lanquepi_nano/sdk_liteos/include",
    "nfc/ndef",
    "nfc/ndef/rtd/",
    "nfc"
  ]
}

```

第十五章：智慧路灯

一、实验目的

- 1、熟悉 HarmonyOS 系统下 IIC 函数的使用以及 BH1750 光强传感器的使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1、IIC 的简介：I2C(Inter-Integrated Circuit, 内部集成电路) 总线是一种由飞利浦 Philip 公司开发的串行总线。是两条串行的总线，它由一根数据线 (SDA) 和一根 时钟线 (SCL) 组成。两条线都需要上拉电阻。I2C 总线上可以接多个 I2C 设备，每个器件都有一个唯一的地址识别。同一时间只能有一个主设备，其他为从设备。通常 MCU 作为主设备控制，外设作为从设备。即模拟-数字转换器，可以将连续变化的模拟信号转换为离散的数字信号，进而使用数字电路进行处理，称之为数字信号处理。

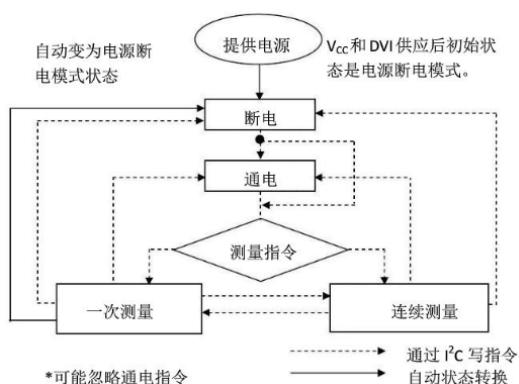
本章节，将使用 IIC 去读写 BH1750，通过查看芯片手册和原理图可知 GPIO_0 对应的是 IIC_SDA,GPIO_1 对应的是 IIC_SCL。

2、BH1750 光强度传感器的介绍

BH1750FVI 是一种用于两线式串行总线接口的数字型光强度传感器集成电路。这种集成电路可以根据收集的光线强度数据来调整液晶或者键盘背景灯的亮度。利用它的高分辨率可以探测较大范围的光强度变化。传感器特点：

- 支持 I2CBUS 接口
- 接近视觉灵敏度的光谱灵敏度特性
- 输出对应亮度的数字值
- 对应广泛的输入光范围。（相当于 1-65535lx）
- 通过降低功率功能，实现低电流化。
- 通过 50Hz/60Hz 除光噪音功能实现稳定的测定。
- 支持 1.8v 逻辑输入接口。
- 无需其他外部件。
- 光源依赖性弱。
- 有两种可选的 I2Cslave 地址。
- 可调的测量结果影响较大的因素为光入口大小。
- 使用这种功能计算 1.1lx 到 100000lx 马克斯/分钟的范围。
- 最小误差变动在±20%。
- 受红外线影响很小。

3、测量程序步骤





4、指令集合

指令	功能代码	注释
断电	0000_0000	无激活状态。
通电	0000_0001	等待测量指令。
重置	0000_0111	重置数字寄存器值，重置指令在断电模式下不起作用。
连续 H 分辨率模式	0001_0000	在 1lx 分辨率下开始测量。 测量时间一般为 120ms。
连续 H 分辨率模式 2	0001_0001	在 0.5lx 分辨率下开始测量。 测量时间一般为 120ms。
连续 L 分辨率模式	0001_0011	在 41lx 分辨率下开始测量。 测量时间一般为 16ms。
一次 H 分辨率模式	0010_0000	在 1lx 分辨率下开始测量。 测量时间一般为 120ms。 测量后自动设置为断电模式。
一次 H 分辨率模式 2	0010_0001	在 0.5lx 分辨率下开始测量。 测量时间一般为 120ms。 测量后自动设置为断电模式。
一次 L 分辨率模式	0010_0011	在 41lx 分辨率下开始测量。 测量时间一般为 16ms。 测量后自动设置为断电模式。
改变测量时间(高位)	01000_MT[7, 6, 5]	改变测量时间 ※ 请参考“根据光学扇窗的影响调整测量结果。”
改变测量时间(低位)	011_MT[4, 3, 2, 1, 0]	改变测量时间 ※ 请参考“根据光学扇窗的影响调整测量结果。”

5、测量模式说明

测量模式	测量时间.	分辨率
H-分辨率模式 2	典型时间： 120ms.	0.5 lx
H-分辨率模式	典型时间： 120ms.	1 lx.
L-分辨率模式	典型时间： 16ms.	4 lx.

我们建议您使用 H 分辨率模式。

H 分辨率模式下足够长的测量时间（积分时间）能够抑制一些噪声（包括 50Hz/60Hz）。同时，H 分辨率模式的分辨率在 1lx 下，适用于黑暗场合下（少于 10 lx）。H 分辨率模式 2 同样适用于黑暗场合下的检测。

三、API 的介绍

1、hi_gpio_init

功能	初始化 GPIO 外设
函数定义	hi_u32 hi_gpio_init(hi_void)
参数	无
返回	错误码

2、hi_io_set_func

功能	设置 GPIO 引脚功能
函数定义	hi_u32 hi_io_set_func(hi_io_name id, hi_u8 val)
参数	id: 表示 GPIO 引脚号 val: 表示 IO 复用功能

返回	错误码
----	-----

3、hi_gpio_set_dir

功能	设置 GPIO 输出方向
函数定义	hi_u32 hi_gpio_set_dir(hi_gpio_idx id, hi_gpio_dir dir)
参数	id: 表示 GPIO 引脚号 dir: 表示 GPIO 输出方向
返回	错误码

4、hi_i2c_init

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_init(hi_i2c_idx id, hi_u32 baudrate)
参数	id: I2C 设备 ID baudrate: I2C 频率
返回	错误码

5、hi_i2c_write

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_write(hi_i2c_idx id, hi_u16 device_addr, const hi_i2c_data *i2c_data)
参数	id: I2C 设备 ID deviceAddr: I2C 设备地址 i2cData: 表示写入的数据
返回	错误码

6、hi_i2c_read

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_read(hi_i2c_idx id, hi_u16 device_addr, const hi_i2c_data *i2c_data)
参数	id: I2C 设备 ID deviceAddr: I2C 设备地址 i2cData: 表示要读取的数据指向的指针
返回	错误码

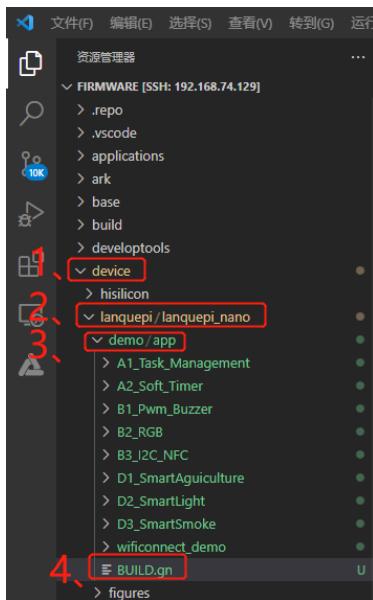
7、hi_i2c_set_baudrate

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_set_baudrate(hi_i2c_idx id, hi_u32 baudrate)
参数	id: I2C 设备 ID baudrate: I2C 频率
返回	错误码

四、实验步骤
1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将的 D2_SmartLight:d2_smart_light 注释#删除

```

device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#B1_First_Led:b1_led",
10    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11    "#B3_ADC:b3_adc",
12    "#B4_I2C_NFC:b4_i2c_nfc",
13    "#B5_RGB:b5_rgb",
14    "#D1_SmartAguiculture:d1_smart_aguiculture",
15    "#D2_SmartLight:d2_smart_light",
16    "#D3_SmartSmoke:d3_smart_smoke",
17  ]
18 }

```

```

device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#B1_First_Led:b1_led",
10    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11    "#B3_ADC:b3_adc",
12    "#B4_I2C_NFC:b4_i2c_nfc",
13    "#B5_RGB:b5_rgb",
14    "#D1_SmartAguiculture:d1_smart_aguiculture",
15    "D2_SmartLight:d2_smart_light",
16    "#D3_SmartSmoke:d3_smart_smoke",
17  ]
18 }

```

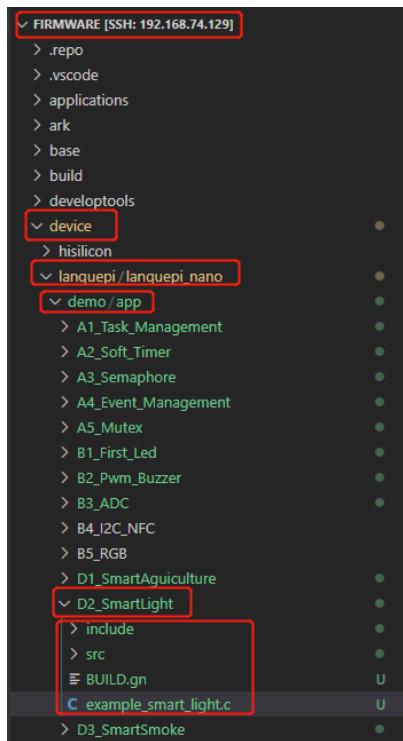
4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、出现如下结果：串口会显示出当前的光强度值，开发板上根据光强度会出现对应的实验现象，当光强度值大于 500 时，灯关闭，当光强度值小于 500 时，灯打开。

7、本章节源文件代码目录如下：



8、本章节 BUILD.gn 目录如下：

```
static_library("d2_smart_light"){
    sources = [
        "example_smart_light.c",
        "./src/bh1750.c"
    ]
    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

第十六章：智慧烟感

一、实验目的

- 1、熟悉 HarmonyOS 系统下 ADC 函数的使用以及 MQ2 烟雾传感器的使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1、ADC 的介绍

ADC(Analog-to-Digital Converter)，即模拟-数字转换器，可以将连续变化的模拟信号转换为离散的数字信号，进而使用数字电路进行处理，称之为数字信号处理。

本章节，将使用 ADC 去读取 GPIO13 引脚的电压，通过查看芯片手册可知 GPIO_13 对应的是 ADC_6。

引脚	Uart	SPI	ADC	PWM	I2S	SDIO	I2C
GPIO_04	LOG_RXD_0		ADC_1				
GPIO_02				PWM_2	MCK_0		
GPIO_05	RXD_1	CSI_0	ADC_2	PWM_2	TX_0		
GPIO_06	TXD_1	CLK_0		PWM_3			
GPIO_14	LOG_RXD_0/CTS_2			PWM_5	RX_0	D1	SCL_0
GPIO_11	TXD_2	RXD_0	ADC_5	PWM_2	CLK_0	CMD	
GPIO_12	RXD_2	CSI_0	ADC_0	PWM_3	WS_0	CLK	
GPIO_13	LOG_RXD_0/RTS_2		ADC_6	PWM_4		D0	SDA_0

4、MQ2 烟雾传感器的介绍

MQ-2 气体传感器所使用的气敏材料是在清洁空气中电导率较低的二氧化锡(SnO₂)。当传感器所处环境中存在可燃气体时，传感器的电导率随空气中可燃气体浓度的增加而增大。使用简单的电路即可将电导率的变化转换为与该气体浓度相对应的输出信号。MQ-2 气体传感器可用于家庭和工厂的气体泄漏检测，适宜对液化气、丁烷、丙烷、甲烷、酒精、氢气、烟雾等的探测，对天然气和其它可燃蒸汽的检测也很理想。这种传感器可检测多种可燃气体，是一款适合多种应用的低成本传感器。

3、MQ-2 烟雾传感器模块特点：

- 具有信号输出指示。
- 双路信号输出（模拟量输出及 TTL 电平输出）。
- TTL 输出有效信号为低电平。（当输出低电平时信号灯亮，可直接接单片机）
- 模拟量输出 0~5V 电压，浓度越高电压越高。
- 对液化气，天然气，城市煤气有较好的灵敏度。
- 结果受温湿度影响。



- $Rs/R0$ 与 ppm 的计算公式: $ppm = 613.9f * \text{pow}(RS/R0, -2.074f)$

二、API 的介绍

1、hi_gpio_init

功能	初始化 GPIO 外设
函数定义	hi_u32 hi_gpio_init(hi_void)
参数	无
返回	错误码

2、hi_io_set_func

功能	设置 GPIO 引脚功能
函数定义	hi_u32 hi_io_set_func(hi_io_name id, hi_u8 val)
参数	id: 表示 GPIO 引脚号 val: 表示 IO 复用功能
返回	错误码

3、hi_gpio_set_dir

功能	设置 GPIO 输出方向
函数定义	hi_u32 hi_gpio_set_dir(hi_gpio_idx id, hi_gpio_dir dir)
参数	id: 表示 GPIO 引脚号 dir: 表示 GPIO 输出方向
返回	错误码

4、hi_pwm_init

功能	PWM 的初始化
函数定义	hi_u32 hi_pwm_init(hi_pwm_port port)
参数	port: 表示 PWM 端口号
返回	错误码

5、hi_pwm_set_clock

功能	PWM 的时钟设置
函数定义	hi_u32 hi_pwm_set_clock(hi_pwm_clk_source clk_type)
参数	clk_type: 表示 PWM 时钟类型, 默认为 160M 时钟
返回	错误码

6、hi_pwm_start

功能	根据输入参数输出 PWM 信号
函数定义	hi_u32 hi_pwm_start(hi_pwm_port port, hi_u16 duty, hi_u16 freq)
参数	port: PWM 端口号 duty: 占空比 freq: 分频倍数
返回	错误码

6、hi_pwm_stop

功能	关闭 PWM 输出信号
函数定义	hi_u32 hi_pwm_stop(hi_pwm_port port)
参数	port: 初始化 PWM 端口号
返回	错误码

7、hi_adc_read

功能	初始化 GPIO 外设
----	-------------

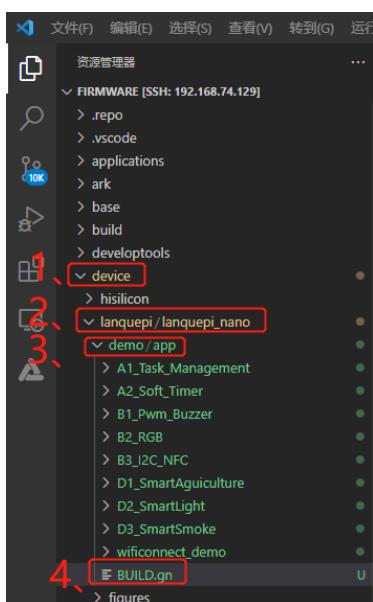
函数定义	hi_u32 hi_adc_read(hi_adc_channel_index channel, hi_u16 *data, hi_adc_equ_model_sel equ_model,hi_adc_cur_bais cur_bais, hi_u16 delay_cnt);
参数	<p>channel: 表示 ADC 通道</p> <p>data: 表示指向存储读取数据的地址的指针</p> <p>equModel: 表示平均算法的次数</p> <p>curBais: 表示模拟功率控制模式</p> <p>rstCnt: 指示从重置到转换开始的时间计数。一次计数等于 334 纳秒。值的范围必须从 0 到 0xFF</p>
返回	错误码

四、实验步骤

1、打开 VsCode

说明: 确保工具是已经 SSH 到 Ubuntu 环境下的 vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将的 D3_SmartSmoke:d3_smoke 注释#删除

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     #"A1_Task_Management:a1_task_management",
5     #"A2_Soft_Timer:a2_soft_timer",
6     #"A3_Semaphore:a3_semaphore",
7     #"A4_Event_Management:a4_event_management",
8     #"A5_Mutex:a5_mutex",
9     #"B1_First_Led:b1_led",
10    #"B2_Pwm_Buzzer:b2_pwm_buzzer",
11    #"B3_ADC:b3_adc",
12    #"B4_I2C_NFC:b4_i2c_nfc",
13    #"B5_RGB:b5_rgb",
14    #"D1_SmartAguiculture:d1_smart_aguiculture",
15    #"D2_SmartLight:d2_smart_light",
16    #"D3_SmartsSmoke:d3_smart_smoke", #注释掉
17  ]
18 }
```

```
device > lanquepi > lanquepi.nano > demo > app >  BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3     features = [
4         "#A1_Task_Management:a1_task_management",
5         "#A2_Soft_Timer:a2_soft_timer",
6         "#A3_Semaphore:a3_semaphore",
7         "#A4_Event_Management:a4_event_management",
8         "#A5_Mutex:a5_mutex",
9         "#B1_First_Led:b1_led",
10        "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11        "#B3_ADC:b3_adc",
12        "#B4_I2C_NFC:b4_i2c_nfc",
13        "#B5_RGB:b5_rgb",
14        "#D1_SmartAgriculture:d1_smart_agriculture",
15        "#D2_SmartLight:d2_smart_light",
16        "#D3_SmartSmoke:d3_smoke",
17    ]
18 }
```

4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

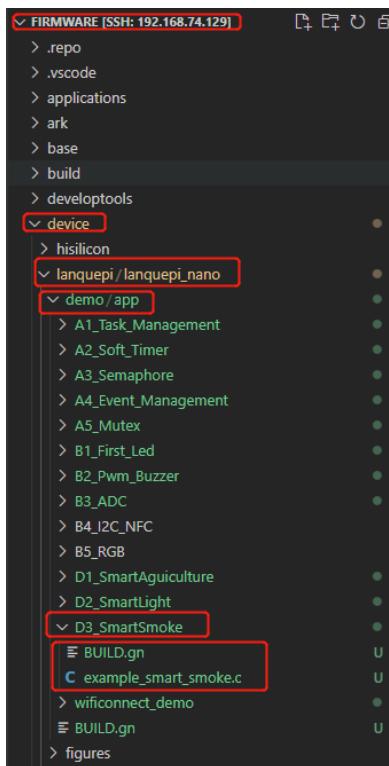
5、按照烧录章节的烧录过程进行烧录。

6、出现如下结果：串口会显示出当前 ADC 采集到的电压以及对应的烟雾值，当烟雾值大于 25 时，蜂鸣器报警。

```
当前电压: 0.49[PPM]: 20.17
ready to OS start
sdk ver:Hi3861V100R001C00SPC025 2020-09-03 18:10:00
FileSystem mount ok.
wifi init success!
hilog will init.
hievent will init.
hievent init success.>> 智慧烟感
>> 开始初始化I0口~
>> 创建烟感数据收集任务~

hiview init success.当前电压: 0.50当前电压: 0.49[PPM]: 19.84
No crash dump found!
当前电压: 0.50[PPM]: 20.33
当前电压: 0.83[PPM]: 69.01
当前电压: 0.65[PPM]: 37.04
当前电压: 0.54[PPM]: 24.90
```

7、本章节源文件代码目录如下：



8、本章节 BUILD.gn 目录如下：

```
static_library("d3_smart_smoke"){

    sources = [
        "example_smart_smoke.c"
    ]

    include_dirs = [
        "./include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```

第十七章：智慧农业

一、实验目的

- 1、熟悉 HarmonyOS 系统下 IIC 函数的使用以及 BH1750 光强传感器和 SHT30 温湿度传感器的使用
- 2、熟悉如何使用 VScode 编辑软件
- 3、熟悉 HarmonyOS 系统的编译流程和设计步骤，能够进行设计、变成、调试
- 4、检测搭建的软件及硬件环境是否可以正常使用

二、实验原理

1、IIC 的简介：I2C(Inter-Integrated Circuit, 内部集成电路) 总线是一种由飞利浦 Philip 公司开发的串行总线。是两条串行的总线，它由一根数据线（SDA）和一根 时钟线（SDL）组成。两条线都需要上拉电阻。I2C 总线上可以接多个 I2C 设备，每个器件都有一个唯一的地址识别。同一时间只能有一个主设备，其他为从设备。通常 MCU 作为主设备控制，外设作为从设备。即模拟-数字转换器，可以将连续变化的模拟信号转换为离散的数字信号，进而使用数字电路进行处理，称之为数字信号处理。

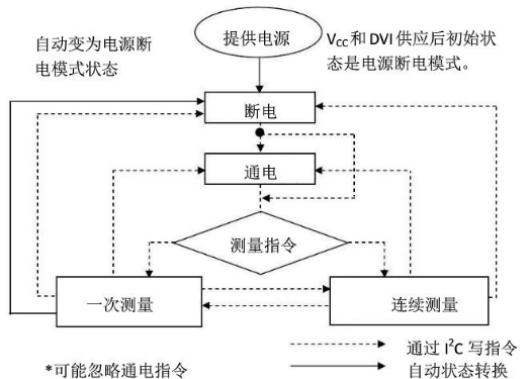
本章节，将使用 IIC 去读写 BH1750，通过查看芯片手册和原理图可知 GPIO_0 对应的是 IIC_SDA,GPIO_1 对应的是 IIC_SCL。

2、BH1750 光强度传感器的介绍

BH1750FVI 是一种用于两线式串行总线接口的数字型光强度传感器集成电路。这种集成电路可以根据收集的光线强度数据来调整液晶或者键盘背景灯的亮度。利用它的高分辨率可以探测较大范围的光强度变化。传感器特点：

- 支持 I2CBUS 接口
- 接近视觉灵敏度的光谱灵敏度特性
- 输出对应亮度的数字值
- 对应广泛的输入光范围。（相当于 1-65535lx）
- 通过降低功率功能，实现低电流化。
- 通过 50Hz/60Hz 除光噪音功能实现稳定的测定。
- 支持 1.8v 逻辑输入接口。
- 无需其他外部件。
- 光源依赖性弱。
- 有两种可选的 I2Cslave 地址。
- 可调的测量结果影响较大的因素为光入口大小。
- 使用这种功能计算 1.1lx 到 100000lx 马克斯/分钟的范围。
- 最小误差变动在±20%。
- 受红外线影响很小。

3、测量程序步骤



4、指令集合

指令	功能代码	注释
断电	0000_0000	无激活状态。
通电	0000_0001	等待测量指令。
重置	0000_0111	重置数字寄存器值，重置指令在断电模式下不起作用。
连续 H 分辨率模式	0001_0000	在 1lx 分辨率下开始测量。 测量时间一般为 120ms。
连续 H 分辨率模式 2	0001_0001	在 0.5lx 分辨率下开始测量。 测量时间一般为 120ms。
连续 L 分辨率模式	0001_0011	在 41lx 分辨率下开始测量。 测量时间一般为 16ms。
一次 H 分辨率模式	0010_0000	在 1lx 分辨率下开始测量。 测量时间一般为 120ms。 测量后自动设置为断电模式。
一次 H 分辨率模式 2	0010_0001	在 0.5lx 分辨率下开始测量。 测量时间一般为 120ms。 测量后自动设置为断电模式。
一次 L 分辨率模式	0010_0011	在 41lx 分辨率下开始测量。 测量时间一般为 16ms。 测量后自动设置为断电模式。
改变测量时间(高位)	01000_MT[7, 6, 5]	改变测量时间 ※ 请参考“根据光学扇窗的影响调整测量结果。”
改变测量时间(低位)	011_MT[4, 3, 2, 1, 0]	改变测量时间 ※ 请参考“根据光学扇窗的影响调整测量结果。”

5、测量模式说明

测量模式	测量时间.	分辨率
H-分辨率模式 2	典型时间： 120ms.	0.5 lx
H-分辨率模式	典型时间： 120ms.	1 lx.
L-分辨率模式	典型时间： 16ms.	4 lx.

我们建议您使用 H 分辨率模式。

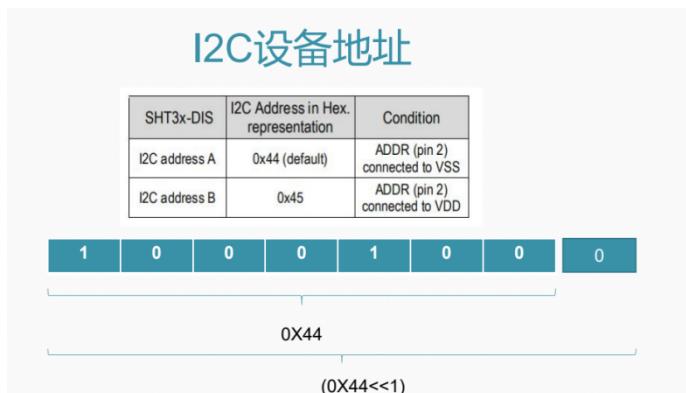
H 分辨率模式下足够长的测量时间（积分时间）能够抑制一些噪声（包括 50Hz/60Hz）。同时，H 分辨率模式的分辨率在 1lx 下，适用于黑暗场合下（少于 10 lx）。H 分辨率模式 2 同样适用于黑暗场合下的检测。

5、SHT30 温湿度传感器的介绍

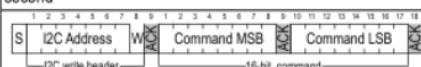
SHT30 温湿度传感器是一个完全校准的、现行的、带有温度补偿的数字输出型传感器，具有 2.4V-5.5V 的宽电压支持，使用 IIC 接口进行通信，最高速率可达 1M 并且有两个用户可选地址。

6、SHT30 的相关知识

- SHT30 的设备地址：HT30_Addr 的地址是 0x44，换成二进制是 0100 0100，右移一位也就是 1000 1000，也就是 0x88。这里是因为开启信号的时候，发送的 8 位数据是前七位是地址，就是 0x44 的后 7 位 100 0100，后面加一个 0 表示写，所以开启信号是 1000 1000 既 0x88。



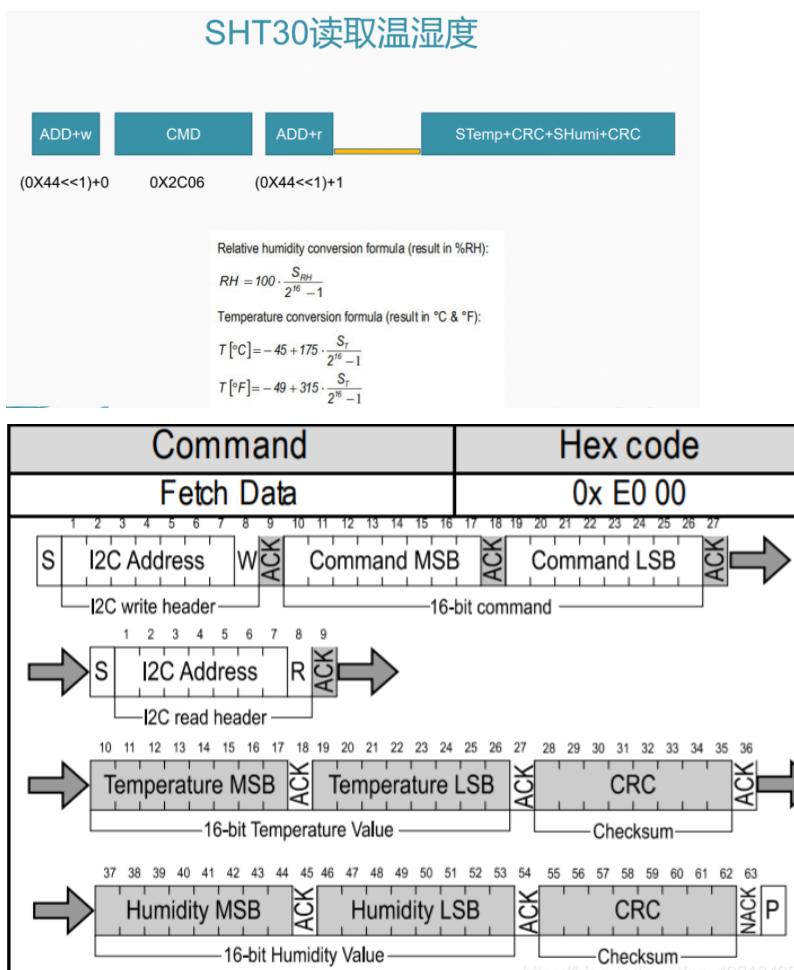
初始化 SHT30，并设置采集周期。命令是 I2C 的地址和高 8 位和低 8 位，这里选择的是 0x22, 0x36。重复性高，采集频率每秒 2 次。

Condition		Hex. code	
Repeatability	mps	MSB	LSB
High			32
Medium			24
Low			2F
High			30
Medium	0.5	0x20	
Low			26
High			2D
Medium	1	0x21	
Low			36
High			20
Medium	2	0x22	
Low			2B
High			34
Medium	4	0x23	
Low			22
High			29
Medium	10	0x27	
Low			37
High			21
Medium			2A
e.g. 0x2130: 1 high repeatability mps - measurement per second			
			

- SHT30 读取温度：发送命令 0xE000 可以读取测量的数据，包括如下：

- 1、温度高 8 位
- 2、温度低 8 位
- 3、温度 CRC
- 4、湿度高 8 位
- 5、湿度低 8 位
- 6、湿度 CRC

先根据 CRC 检测温度，如果没有问题赋值，其次根据 CRC 检测湿度，如果没问题赋值。



三、API 的介绍

1、hi_gpio_init

功能	初始化 GPIO 外设
函数定义	hi_u32 hi_gpio_init(hi_void)
参数	无
返回	错误码

2、hi_io_set_func

功能	设置 GPIO 引脚功能
函数定义	hi_u32 hi_io_set_func(hi_io_name id, hi_u8 val)
参数	id: 表示 GPIO 引脚号 val: 表示 IO 复用功能
返回	错误码

3、hi_gpio_set_dir

功能	设置 GPIO 输出方向
函数定义	hi_u32 hi_gpio_set_dir(hi_gpio_idx id, hi_gpio_dir dir)
参数	id: 表示 GPIO 引脚号 dir: 表示 GPIO 输出方向
返回	错误码



4、hi_i2c_init

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_init(hi_i2c_idx id, hi_u32 baudrate)
参数	id: I2C 设备 ID baudrate: I2C 频率
返回	错误码

5、hi_i2c_write

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_write(hi_i2c_idx id, hi_u16 device_addr, const hi_i2c_data *i2c_data)
参数	id: I2C 设备 ID deviceAddr: I2C 设备地址 i2cData: 表示写入的数据
返回	错误码

6、hi_i2c_read

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_read(hi_i2c_idx id, hi_u16 device_addr, const hi_i2c_data *i2c_data)
参数	id: I2C 设备 ID deviceAddr: I2C 设备地址 i2cData: 表示要读取的数据指向的指针
返回	错误码

7、hi_i2c_set_baudrate

功能	用指定的频率初始化 I2C 设备
函数定义	hi_u32 hi_i2c_set_baudrate(hi_i2c_idx id, hi_u32 baudrate)
参数	id: I2C 设备 ID baudrate: I2C 频率
返回	错误码

8、hi_i2c_writeread

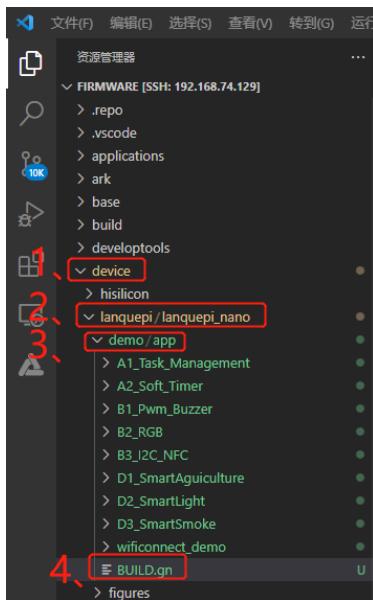
功能	I2C 向从机发送数据，然后接收从机数据
函数定义	hi_u32 hi_i2c_writeread(hi_i2c_idx id, hi_u16 device_addr, const hi_i2c_data *i2c_data)
参数	id: I2C 设备 ID deviceAddr: I2C 设备地址 i2cData: 表示写入的数据
返回	错误码

四、实验步骤

1、打开 VsCode

说明：确保工具是已经 SSH 到 Ubuntu 环境下的 Vscode 界面。

2、按如下图的路径，打开 BUILD.gn 文件。



3、将的 D1_SmartAguiculture:d1_smart_aguiculture 注释#删除

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#B1_First_Led:b1_led",
10    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11    "#B3_ADC:b3_adc",
12    "#B4_I2C_NFC:b4_i2c_nfc",
13    "#B5_RGB:b5_rgb",
14    "#D1_SmartAguiculture:d1_smart_aguiculture",
15    "#D2_SmartLight:d2_smart_light",
16    "#D3_SmartSmoke:d3_smart_smoke",
17  ]
18 }
```

```
device > lanquepi > lanquepi_nano > demo > app > BUILD.gn
1 import("//build/lite/config/component/lite_component.gni")
2 lite_component("app") {
3   features = [
4     "#A1_Task_Management:a1_task_management",
5     "#A2_Soft_Timer:a2_soft_timer",
6     "#A3_Semaphore:a3_semaphore",
7     "#A4_Event_Management:a4_event_management",
8     "#A5_Mutex:a5_mutex",
9     "#B1_First_Led:b1_led",
10    "#B2_Pwm_Buzzer:b2_pwm_buzzer",
11    "#B3_ADC:b3_adc",
12    "#B4_I2C_NFC:b4_i2c_nfc",
13    "#B5_RGB:b5_rgb",
14    "D1_SmartAguiculture:d1_smart_aguiculture",
15    "#D2_SmartLight:d2_smart_light",
16    "#D3_SmartSmoke:d3_smart_smoke",
17  ]
18 }
```

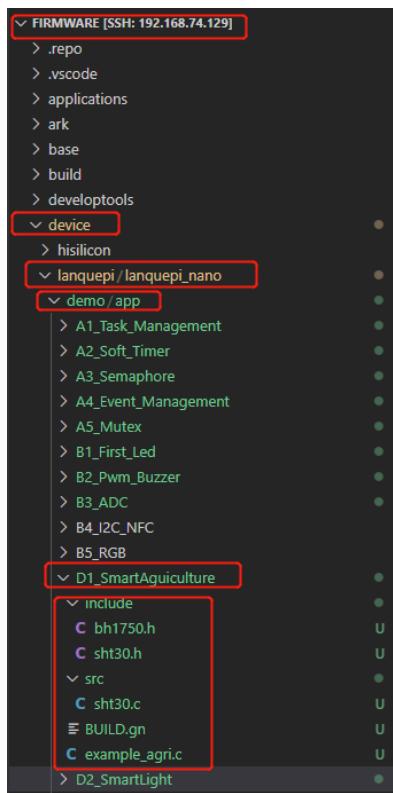
4、打开 VsCode 的终端，输入如下命令，按下回车进行编译。

```
hihope@hihope-virtual-machine:~/firmware$ hb build -f
```

5、按照烧录章节的烧录过程进行烧录。

6、出现如下结果：串口会显示出当前的光强度值和温湿度值。开发板上根据光强度会出现对应的实验现象，当光强度值大于 500 时，灯关闭，当光强度值小于 500 时，灯打开。当温湿度低于设定值时，蜂鸣器会响起，同时电机开始转动。

7、本章节源文件代码目录如下：



8、本章节 BUILD.gn 目录如下：

```
static_library("d1_smart_agriculture"){

    sources = [
        "example_agri.c",
        "src/sht30.c"
    ]

    include_dirs = [
        "include",
        "//utils/native/lite/include",
        "//device/lanquepi/lanquepi_nano/sdk_liteos/include"
    ]
}
```