**鸿蒙开发教程**

**蓝鹊派-BMP600**

**ESP32**



**福州比特元科技有限公司**

目录

**[第一章：搭建开发环境](#_Toc137225523)** [3](#_Toc137225523)

**[一、 蓝鹊派BMP600开发板简介](#_Toc137225524)** [3](#_Toc137225524)

**[二、 使用镜像搭建Ubuntu20开发环境](#_Toc137225525)** [3](#_Toc137225525)

**[三、 手动搭建 Ubuntu20 开发环境](#_Toc137225526)** [4](#_Toc137225526)

**[四、 烧录镜像](#_Toc137225527)** [6](#_Toc137225527)

**[五、 运行镜像](#_Toc137225528)** [11](#_Toc137225528)

**[第二章：内核编程-Thread多线程](#_Toc137225529)** [16](#_Toc137225529)

**[一、Thread struct分析](#_Toc137225530)** [16](#_Toc137225530)

**[二、Thread API分析](#_Toc137225531)** [16](#_Toc137225531)

**[三、软件设计](#_Toc137225532)** [18](#_Toc137225532)

**[四、编译调试](#_Toc137225533)** [18](#_Toc137225533)

**[第三章：内核编程-](#_Toc137225534)****[Semaphore信号量](#_Toc137225534)** [20](#_Toc137225534)

**[一、Semaphore API分析](#_Toc137225535)** [20](#_Toc137225535)

**[二、软件设计](#_Toc137225536)** [21](#_Toc137225536)

**[三、编译调试](#_Toc137225537)** [21](#_Toc137225537)

**[第四章：内核编程- Message消息队列](#_Toc137225538)** [23](#_Toc137225538)

**[一、Message API分析](#_Toc137225539)** [23](#_Toc137225539)

**[二、软件设计](#_Toc137225540)** [25](#_Toc137225540)

**[三、编译调试](#_Toc137225541)** [25](#_Toc137225541)

**[第五章：驱动编程-ADC土壤湿度传感器的使用](#_Toc137225542)** [26](#_Toc137225542)

**[一、接口说明](#_Toc137225543)** [26](#_Toc137225543)

**[二、功能设计](#_Toc137225544)** [26](#_Toc137225544)

**[三、编译调试](#_Toc137225545)** [27](#_Toc137225545)

**[第六章：驱动编程-SPI全彩灯带的使用](#_Toc137225546)** [28](#_Toc137225546)

**[一、接口说明](#_Toc137225547)** [28](#_Toc137225547)

**[二、功能设计](#_Toc137225548)** [29](#_Toc137225548)

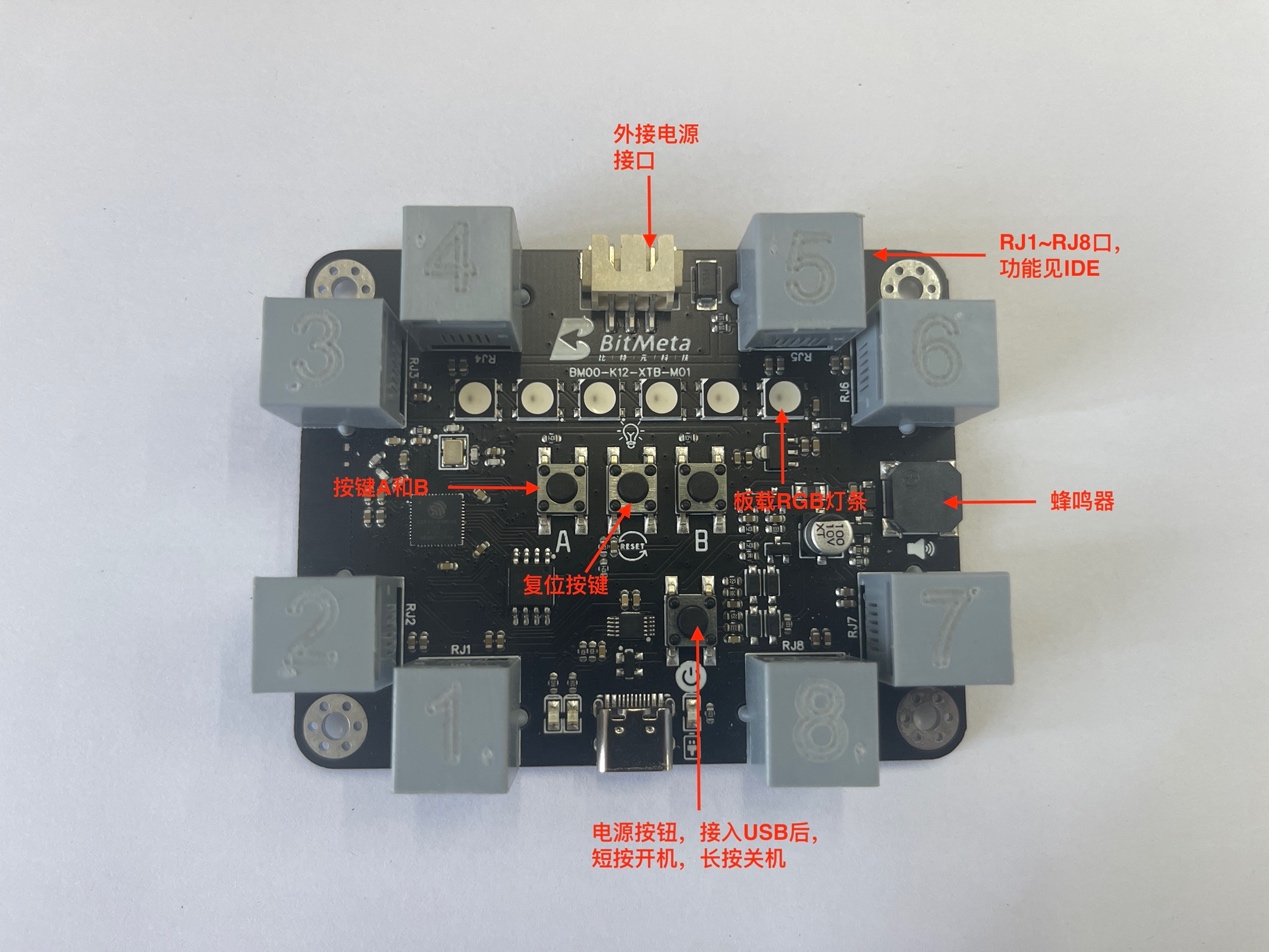
**[三、编译调试](#_Toc137225549)** [29](#_Toc137225549)

**[第七章：综合实验-智能编程鱼缸](#_Toc137225550)** [30](#_Toc137225550)

**[一、编译调试](#_Toc137225551)** [31](#_Toc137225551)

**第一章：搭建开发环境**

1. **蓝鹊派BMP600开发板简介**



蓝鹊派 BMP600 开发板是基于 ESP32U4WDH 推出的物联网设备开发套件，集成2.4GHz Wifi和蓝牙双模，具有超高的射频性能、稳定性、通用性和可靠性，以及超低的功耗，适用于各种应用场景；蓝鹊派 BMP600 开发板还支持锂电池供电和充放电管理；开发套件提供一个开箱即用的智能硬件解决方案，方便开发者验证和开发自己的软件和功能，缩短产品研发周期并快速推向市场。

1. **使用镜像搭建Ubuntu20开发环境**

此种方式对于初学者最为简单，不用手动安装各种开发工具链，只需将下面的网盘中的镜像下载下来，导入 VMware 虚拟机中即可使用。Ubuntu账户密码：123456

镜像下载地址：

链接：https://pan.baidu.com/s/1uhC6IUSiLL8Vh5JPQ6vzIg?pwd=xje9

提取码：xje9

图 1.6

1. **手动搭建 Ubuntu20 开发环境**

系统环境要求：建议[Ubuntu20.04](https://mirrors.aliyun.com/oldubuntu-releases/releases/20.04.3/) 版本及以上

**更新Ubuntu源**

* 打开sources.list文件。

sudo gedit /etc/apt/sources.list

* 将与系统对应版本源复制并覆盖至上述打开的文件中，保存关闭，执行如下命令。（建议使用[ubuntu源](https://gitee.com/link?target=https%3A%2F%2Fdeveloper.aliyun.com%2Fmirror%2Fubuntu)）。

sudo apt update

**安装依赖工具与库**

* 安装ccache（用于加速编译）。

sudo apt install ccache

* 安装git工具并配置用户名和邮箱。

sudo apt install git git-lfs

git config --global user.email "xxx"

git config --global user.name "xxx"

* 设置Python软链接为Python3.8。

sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.8 1

* 安装并升级Python包管理工具(pip3)。

sudo apt-get install python3-setuptools python3-pip -y

sudo pip3 install --upgrade pip

* 安装Python3工具包。

pip3 install --trusted-host mirrors.aliyun.com -i http://mirrors.aliyun.com/pypi/simple jinja2 ohos-build==0.4.6 esptool

* 将hb工具添加至环境变量。

gedit ~/.bashrc #打开环境配置文件

export PATH=$PATH:~/.local/bin #添加该语句至文件末尾，保存退出

source ~/.bashrc #使环境配置文件生效

**安装repo工具**

如果已经安装并拉取过OpenHarmony代码，请忽略该步骤。

如果是通过apt install安装的repo，请卸载后按照下述步骤重新安装。

sudo apt install curl -y

curl -s https://gitee.com/oschina/repo/raw/fork\_flow/repo-py3 > repo

chmod a+x repo

sudo mv repo /usr/local/bin/

**安装交叉编译工具链**

新建一个目录，用来存放下载的编译工具链。

mkdir -p ~/download && cd ~/download

下载交叉编译工具链压缩包。

wget https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8\_4\_0-esp-2021r2-linux-amd64.tar.gz

将交叉编译工具添加至环境变量。

* 解压工具链。

sudo tar axvf xtensa-esp32-elf-gcc8\_4\_0-esp-2021r2-linux-amd64.tar.gz -C /opt/

* 打开配置文件。

gedit ~/.bashrc

* 在文件末尾添加。

export PATH=$PATH:/opt/xtensa-esp32-elf/bin

* 使配置生效。

source ~/.bashrc

**获取源码**

# 移动到用户主目录

cd ~

# 下载源代码

wget https://www.biteyuan.net/download/BitMeta.tar.xz

# 解压源代码并进入代码目录

tar -xvf BitMeta.tar.xz

cd BitMeta

# 安装hb工具

pip3 install build/lite

# 设置根路径

hb set -root .

**构建源码**

* 在源码根目录，执行hb set命令并选择openvalley下项目iotlink。

hb set

* 执行hb build -f脚本构建产生固件。

hb build -f

构建成功会提示类似如下信息。

[OHOS INFO] iotlink build success

[OHOS INFO] cost time: 0:00:28

* 查看生成的固件。

ls out/niobeu4/iotlink/bin/

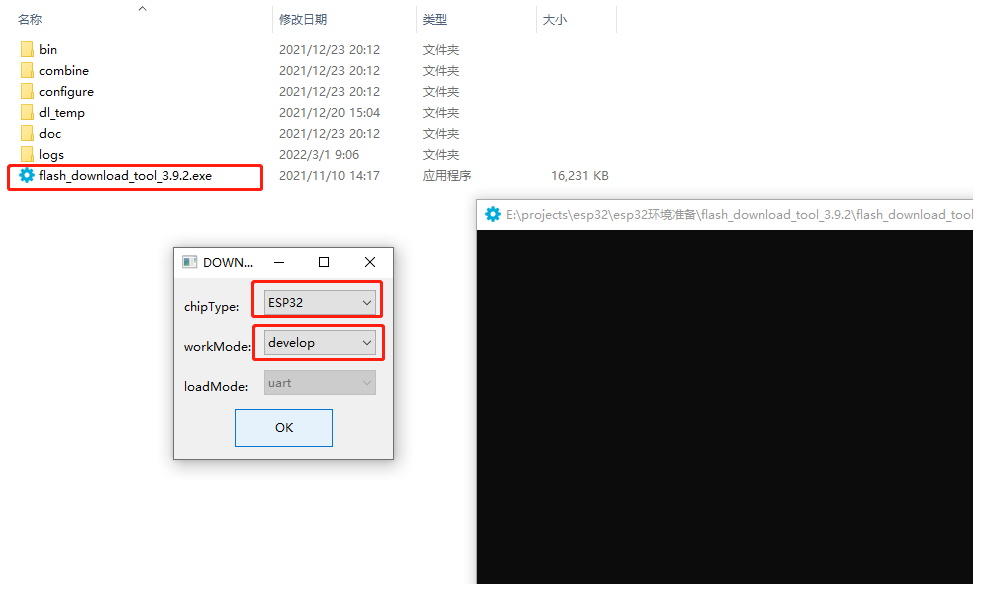
| **固件名称** | **用途** |
| --- | --- |
| OHOS\_Image.bin | OpenHarmony内核及应用镜像文件 |
| bootloader.bin | 启动引导镜像文件 |
| partitions.bin | 分区表镜像文件 |

1. **烧录镜像**

Windows下可以使用Flash\_Download\_Tool工具进行烧录，点击[这里](https://gitee.com/link?target=https%3A%2F%2Fwww.espressif.com.cn%2Fsites%2Fdefault%2Ffiles%2Ftools%2Fflash_download_tool_3.9.2_0.zip)开始下载。

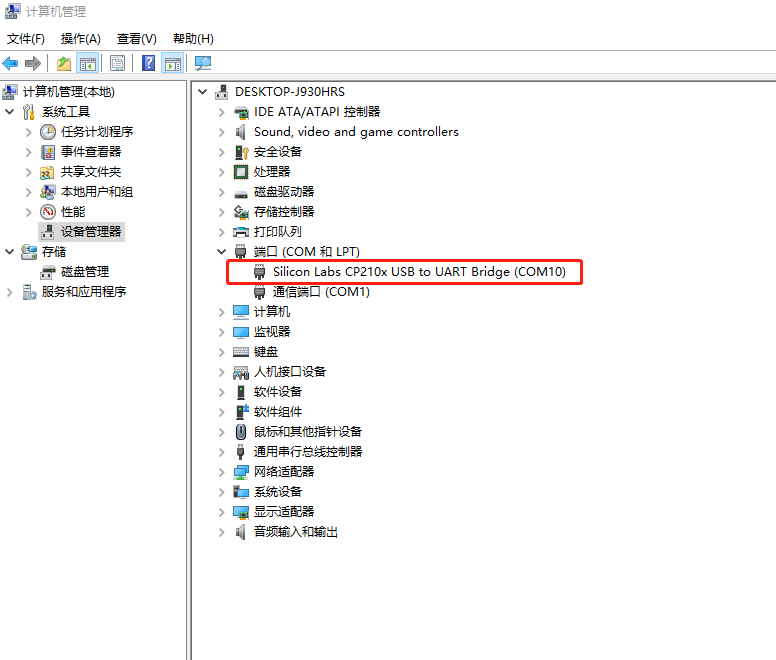
* 在windows下解压flash\_download\_tool\_3.9.2.rar。
* 双击解压后得到的烧录工具flash\_download\_tool\_3.9.2.exe，chipType选择 ESP32, workMode选择 develop，进入主界面。

**图1** Flash\_Download\_Tool打开页面图



* 连接开发板的TypeC口到PC，查看设备管理器，确定串口端口号。

**图2** 设备管理器图



* 进入下载页面，填入需要烧录的 bin 文件和对应的烧录地址，并根据自己实际 需求填入 SPI SPEED、SPI MODE、COM 及 BAUD。

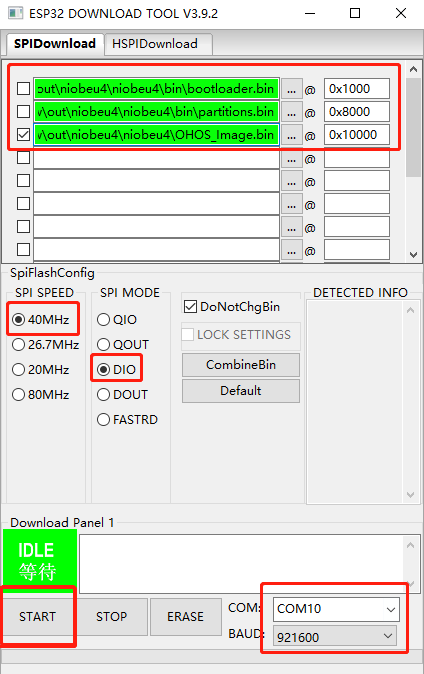
固件对应烧录地址如下。

bootloader.bin ----------> 0x1000

partitions.bin ----------> 0x8000

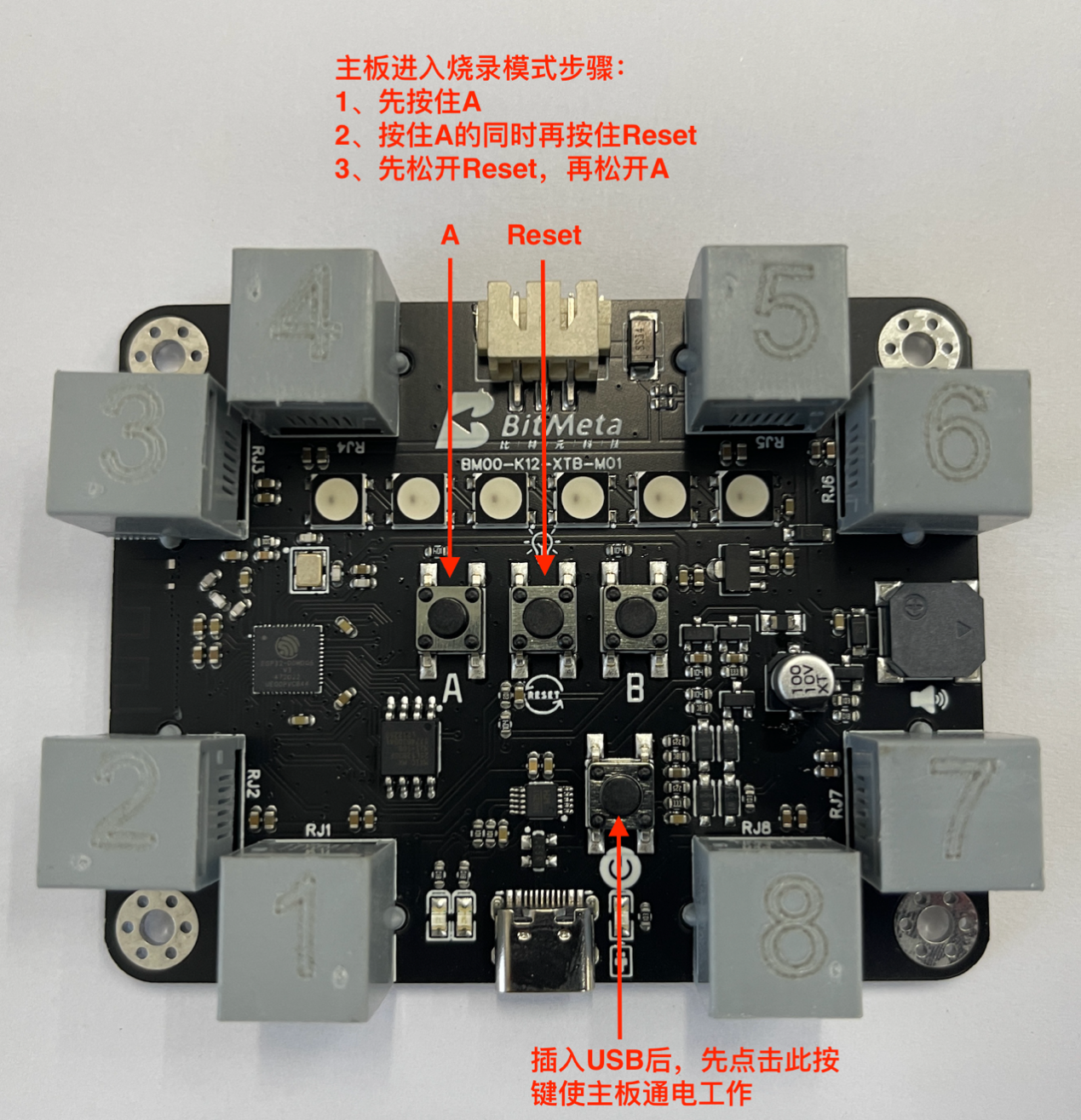
OHOS\_Image.bin ----------> 0x10000

**图3** 烧录设置页面图

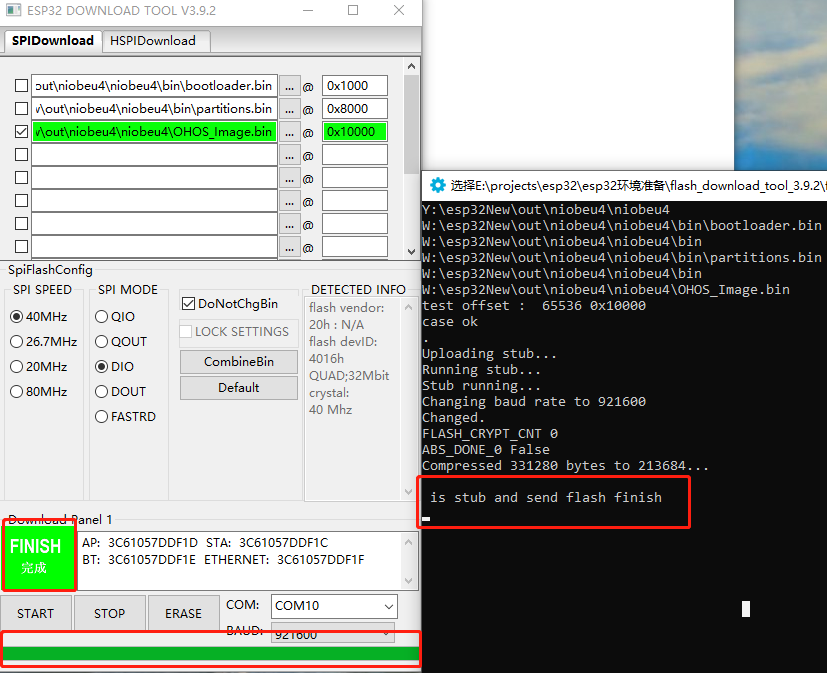


* 点击 START 开始下载。下载过程中，下载工具会读取 flash 的信息和芯片的 MAC 地址。我们可以通过勾选框选择是否烧录该文件，一般我们在首次烧录时会全部烧录，为了加快开发效率，调试时只需烧录OHOS\_Image.bin即可。下载完成后，可以看到如下提示。

**图4** 点击 START 后，按键使主板进入烧录状态



**图5** 烧录成功页面图



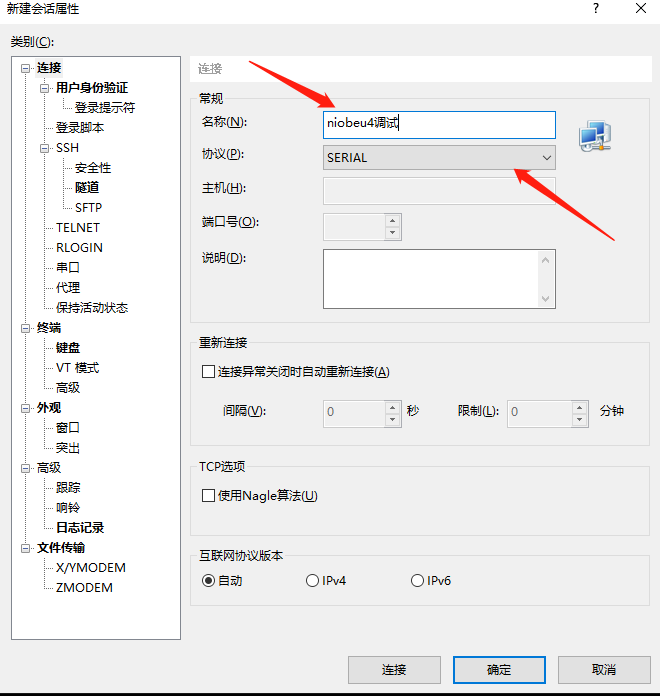
**五、 运行镜像**

烧录完成后需要按下复位键复位设备，程序才会启动。

推荐使用xshell进行串口调试，点击[这里](https://gitee.com/link?target=https%3A%2F%2Fwww.xshell.com%2Fzh%2Ffree-for-home-school%2F)进行下载。

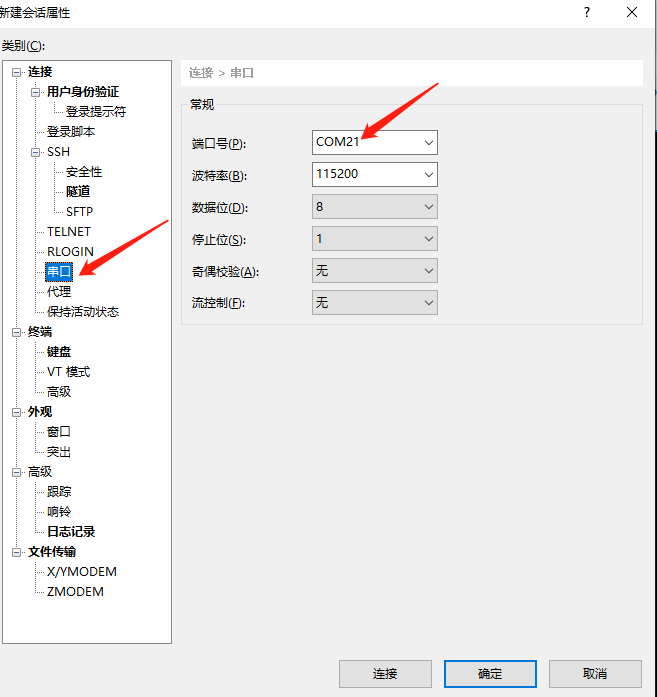
* 新建会话（菜单栏->文件->新建），设置会话名称并选择协议为SERIAL。

**图8** xshell页面新建会话图



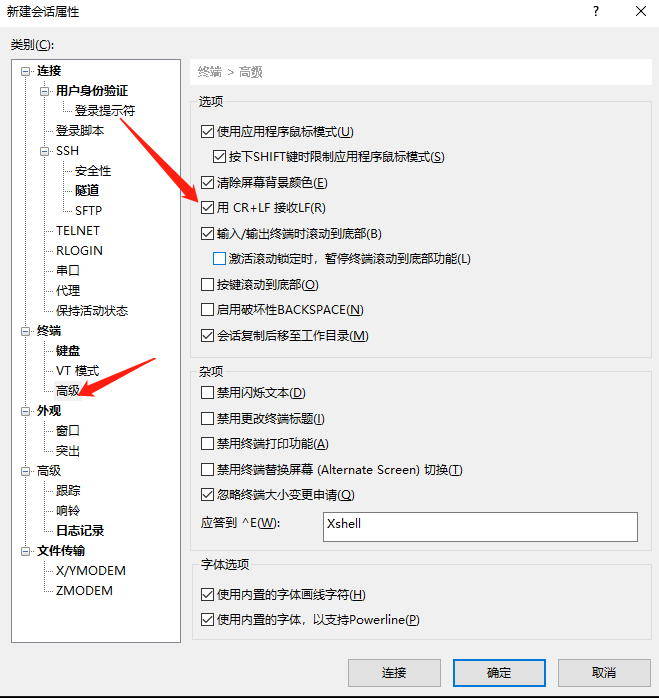
* 配置串口参数，选择正确的端口号，波特率设置为115200，如下图所示。

**图9** xshell页面串口参数配置图



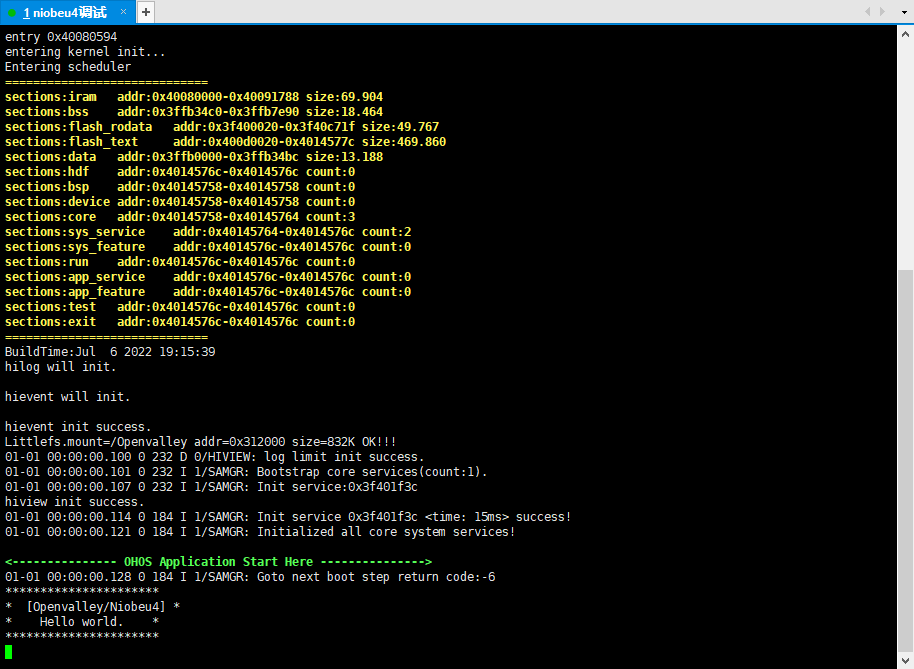
* 由于系统打印信息中没有使用\r\n换行，会导致log显示不对齐。需要设置终端属性，用CR+LF接收LF(R)。

**图10** xshell页面设置终端属性图



* 复位设备，日志打印如下所示。

**图11** 日志打印图



**第二章：内核编程-Thread多线程**

本示例将演示如何在BMP600开发板上使用liteos-m接口进行多线程开发

**一、Thread struct分析**

TSK\_INIT\_PARAM\_S

typedef struct tagTskInitParam {

TSK\_ENTRY\_FUNC pfnTaskEntry;

UINT16 usTaskPrio;

UINT32 uwArg;

UINTPTR stackAddr;

UINT32 uwStackSize;

CHAR \*pcName;

UINT32 uwResved;

} TSK\_INIT\_PARAM\_S;

描述：

名字 描述

pfnTaskEntry 新建线程的函数入口

usTaskPrio 线程优先级，0-31

uwArg 线程函数传参

stackAddr 堆栈起始地址

uwStackSize 堆栈大小，当传入值为0时，使用默认值LOSCFG\_BASE\_CORE\_TSK\_DEFAULT\_STACK\_SIZE

pcName 线程名称

**二、Thread API分析**

LOS\_TaskCreate()

UINT32 LOS\_TaskCreate(UINT32 \*taskID, TSK\_INIT\_PARAM\_S \*taskInitParam);

描述：

函数LOS\_TaskCreate通过将线程添加到活动线程列表并将其设置为就绪状态来启动线程函数。当创建的thread函数的优先级高于当前运行的线程且线程调度未被锁定时，创建的thread函数立即启动并成为新的运行线程。 可以在RTOS启动(调用 osKernelStart)之前安全地调用该函数，但不能在内核初始化 (调用 osKernelInitialize)之前调用该函数。

注意 :不能在中断服务调用该函数

名字 描述

taskID 指针，指向线程id

taskInitParam 新建线程的相关属性以及线程的函数入口和参数等

LOS\_TaskSuspend()

UINT32 LOS\_TaskSuspend(UINT32 taskID);

描述：

根据参数taskID，挂起相应任务。此时任务taskID如果处于运行态，会由运行态切换到阻塞态，并插入到阻塞队列中，而cpu会转去执行就绪列表中最高优先级的任务。

注意 :不能在任务调度处于关闭的状态下调用（即不能在调用osKernelLock之后调用）

名字 描述

taskID 线程id

LOS\_TaskResume()

UINT32 LOS\_TaskResume(UINT32 taskID)

描述：

根据参数taskID，恢复相应任务。此时任务taskID如果处于阻塞态，会由阻塞态切换到就绪态，并插入到就绪队列中，如果此时就绪队列中没有比该任务taskID更高优先级的任务，则任务taskID会由就绪态切换到运行态。

名字 描述

taskID 线程id

osThreadJoin()

UINT32 LOS\_TaskJoin(UINT32 taskID, UINTPTR \*retval);

描述：

挂起当前任务，等待指定任务taskID运行结束并回收其任务控制块资源。

名字 描述

taskID 线程id

retval 线程taskID结束后的返回值

LOS\_TaskDelete()

UINT32 LOS\_TaskDelete(UINT32 taskID);

描述：

删除指定线程/任务，释放其任务堆栈和任务控制块的资源

名字 描述

taskID 线程id

LOS\_TaskLock()

VOID LOS\_TaskLock(VOID);

描述：

锁任务调度。在任务调度被锁的状态下，不能调用LOS\_TaskSuspend，LOS\_TaskDelay

LOS\_TaskUnlock()

UINT32 LOS\_TaskUnlock(VOID);

描述：

解锁任务调度。

**三、软件设计**

主要代码分析

在OS\_Thread\_example函数中，通过osThreadNew()函数创建了threadHi和threadLo两个不同优先级的任务，包括任务延迟、挂起、恢复等操作，，threadHi和threadLo启动后会输出打印日志，通过打印日志可熟悉任务优先级调度机制。

**四、编译调试**

修改 vendor/openvalley/niobeu4/kernel\_configs/debug.config 文件

在文件末尾添加宏 LOSCFG\_NIOBEU4\_APPLICATION\_002=y

同时将其他案例宏注释屏蔽

回到源码根目录BitMeta，执行hb build -f进行编译。

运行结果

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:

LOS\_TaskLock() Success!

curTask pro = 15

Example\_TaskHi create Success!

Example\_TaskLo create Success!

Enter TaskHi Handler.

Enter TaskLo Handler.

TaskHi LOS\_TaskDelay Done.

TaskHi LOS\_TaskSuspend Success.

TaskHi LOS\_TaskResume Success.

Join Example\_TaskHi Success!

**第三章：内核编程-** **Semaphore信号量**

本示例将演示如何在BMP600开发板上使用liteos-m 接口进行信号量开发

**一、Semaphore API分析**

LOS\_SemCreate()

UINT32 LOS\_SemCreate(UINT16 count, UINT32 \*semHandle);

描述：

信号量创建，从未使用的信号量链表中获取一个信号量，并设定初值count，并将新建的信号量ID赋值给semHandle所指向的空间

注意 :不能在中断服务调用该函数

名字 描述

count 信号量计数值的初始值，其取值范围：[0, OS\_SEM\_COUNTING\_MAX\_COUNT)

semHandle 指针，指向存放信号量ID的对象

LOS\_BinarySemCreate()

UINT32 LOS\_BinarySemCreate(UINT16 count, UINT32 \*semHandle);

描述：

创建二值信号量，从未使用的信号量链表中获取一个二值信号量，并设定初值count，并将新建的信号量ID赋值给semHandle所指向的空间

注意 :不能在中断服务调用该函数

名字 描述

count 信号量计数值的初始值，其取值范围:[0,1]

semHandle 指针，指向存放信号量ID的对象

LOS\_SemPost()

UINT32 LOS\_SemPost(UINT32 semHandle);

描述：

释放信号量

注意 :该函数可以在中断服务例程调用

名字 描述

semHandle 信号量ID

返回 0 - 成功，非0 - 失败

LOS\_SemPend()

UINT32 LOS\_SemPend(UINT32 semHandle, UINT32 timeout);

描述：

获取信号量

名字 描述

semHandle 信号量ID

timeout 超时值

返回 0 - 成功，非0 - 失败

LOS\_SemGetValue()

UINT32 LOS\_SemGetValue(UINT32 semHandle, INT32 \*currVal);

描述：

获取当前可被使用的共享资源数

**二、软件设计**

主要代码分析

在semp\_example函数中，创建了两个信号量semMutex和semSync，通过semSync来实现线程Thread\_semp1和Thread\_semp2之间的同步；通过semMutex来实现线程Thread\_sempA、Thread\_sempB、Thread\_sempC、Thread\_sempD对全局变量g\_str的互斥访问。

**三、编译调试**

修改 vendor/openvalley/niobeu4/kernel\_configs/debug.config 文件

在文件末尾添加宏 LOSCFG\_NIOBEU4\_APPLICATION\_006=y

同时将其他案例宏注释屏蔽

回到源码根目录BitMeta，执行hb build -f进行编译。

运行结果

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:

entry Thread\_Semp2 : try get semSync, timeout 100 ticks!

Thread\_Semp2: failed to get semSync! and try get semSync wait forever.

Thread\_Semp1 Release semSync.

Thread\_Semp2: wait\_forever and get semSync.

Thread\_SempA read g\_str = hello!

Thread\_SempB read g\_str = hello!

Thread\_SempC read g\_str = hello!

Thread\_SempD read g\_str = hello!

**第四章：内核编程- Message消息队列**

本示例将演示如何在BMP600开发板上使用cmsis 2.0 接口进行消息队列开发

**一、Message API分析**

LOS\_QueueCreate()

UINT32 LOS\_QueueCreate(const CHAR \*queueName,

UINT16 len,

UINT32 \*queueID,

UINT32 flags,

UINT16 maxMsgSize);

描述：

创建消息队列

注意 :不能在中断服务调用该函数

名字 描述

queueName 消息队列名称，暂时未使用

len 消息队列长度，该值的范围：[1,0xffff]

queueID 消息队列ID，属输出

flags 队列模式，暂未用，可传0

maxMsgSize 消息的大小，该值范围：[1,0xffff-4]

LOS\_QueueReadCopy()

UINT32 LOS\_QueueReadCopy(UINT32 queueID, VOID \*bufferAddr, UINT32 \*bufferSize, UINT32 timeOut);

描述：

读取消息队列，用于读取指定队列中的数据，并将获取到的数据存储到bufferAddr指定的地址中。 要读取的数据的地址和大小由用户定义。

注意 :不能在中断服务调用该函数

名字 描述

queueID 队列ID

bufferAddr 指针，存放读取到的消息

bufferSize 在读取之前维护缓冲区的期望大小，和读取后消息的实际大小

timeOut 超时值，范围：[0,LOS\_WAIT\_FOREVER]，单位tick

LOS\_QueueWriteCopy()

UINT32 LOS\_QueueWriteCopy(UINT32 queueID, VOID \*bufferAddr, UINT32 bufferSize, UINT32 timeOut);

描述：

发送消息，用于将bufferSize指定的大小和存储在bufferAddr指定地址的数据写入队列中。

注意 :如果参数timeout设置为0，可以从中断服务例程调用

名字 描述

queueID 消息队列ID

bufferAddr 要发送的消息

bufferSize 消息大小

timeout 超时值

LOS\_QueueRead()

UINT32 LOS\_QueueRead(UINT32 queueID, VOID \*bufferAddr, UINT32 bufferSize, UINT32 timeOut);

描述：

获取消息，用于读取指定队列中的数据地址，并将其存储到由bufferAddr指定的地址中。

注意 :不能在中断服务调用该函数

名字 描述

queueID 消息队列ID

bufferAddr 存储获取数据的起始地址。起始地址不能为空。

bufferSize 传入的缓冲区大小，不能为0，范围:[1,0xffffffff]

timeout 超时值，范围：[0,LOS\_WAIT\_FOREVER]，单位tick

返回 0 - 成功，非0 - 失败

LOS\_QueueWrite()

UINT32 LOS\_QueueWrite(UINT32 queueID, VOID \*bufferAddr, UINT32 bufferSize, UINT32 timeOut);

描述：

发送消息

**二、软件设计**

主要代码分析

创建一个队列，两个任务。任务1调用写队列接口发送消息，任务2通过读队列接口接收消息。

**三、编译调试**

修改 vendor/openvalley/niobeu4/kernel\_configs/debug.config 文件

在文件末尾添加宏 LOSCFG\_NIOBEU4\_APPLICATION\_007=y

同时将其他案例宏注释屏蔽

回到源码根目录BitMeta，执行hb build -f进行编译。

运行结果

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:

start queue example.

create the queue success.

recv message: test message

delete the queue success.

**第五章：驱动编程-ADC土壤湿度传感器的使用**

简介

本案例程序将演示ADC采集相关程序。

**一、接口说明**

AdcOpen()

int32\_t AdcOpen(uint32\_t number)

描述：

开启ADC函数，使用ADC的时候调用。

名字 描述

number 设置 1：ADC序号, 范围[1,2]

AdcClose()

void AdcClose(DevHandle handle)

描述：

关闭ADC函数，关闭ADC的时候调用。

名字 描述

handle 设备句柄

AdcRead()

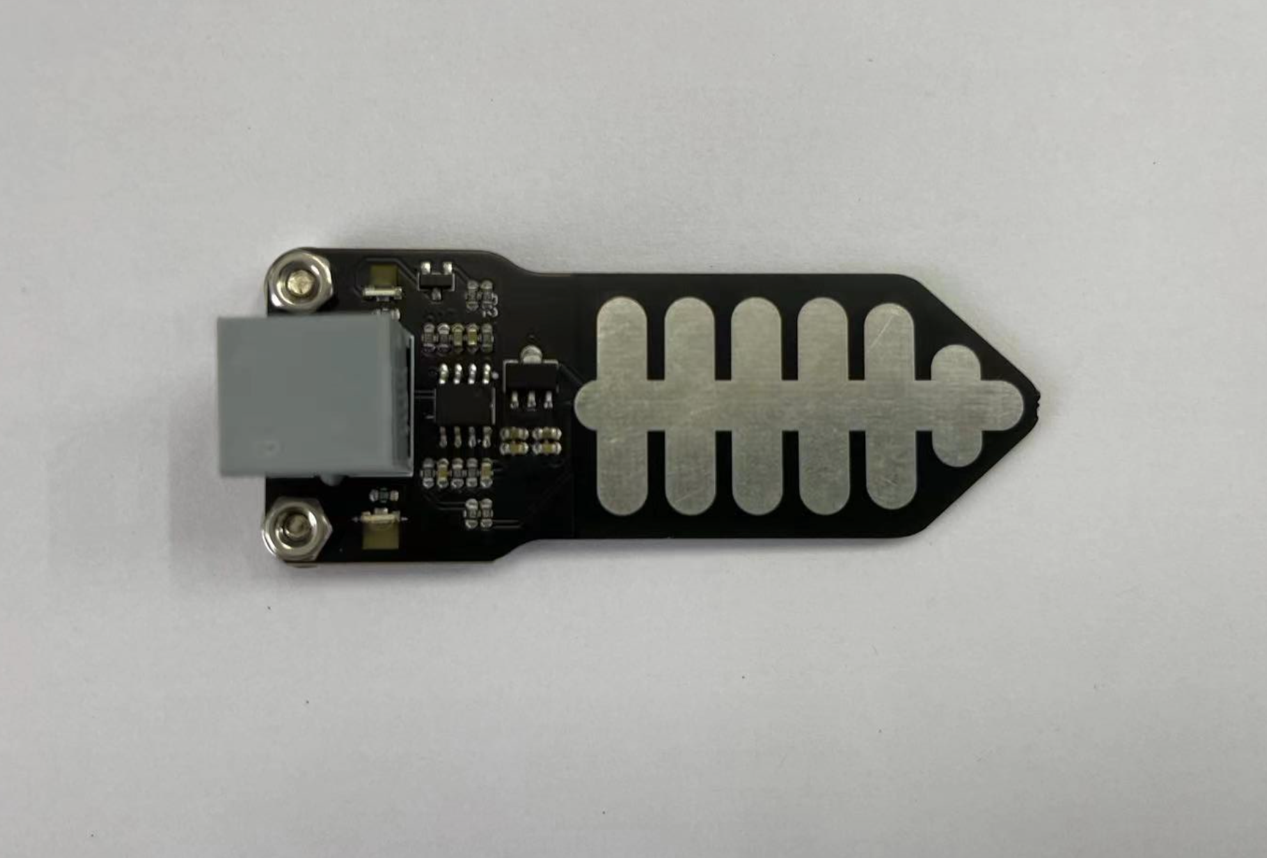
int32\_t AdcRead(DevHandle handle, uint32\_t channel, uint32\_t \*val)

描述：

关闭ADC函数，关闭ADC的时候调用。

**二、功能设计**

将如下图所示的土壤湿度传感器，用RJ11 线接到主板除4/7两口以外的任一口，主控MCU使用ADC读取值并打印到串口，过程中可以往传感器上滴水并擦干，观察打印值的变化。



**三、编译调试**

修改 vendor/openvalley/niobeu4/kernel\_configs/debug.config 文件

在文件末尾添加宏 LOSCFG\_NIOBEU4\_APPLICATION\_b8=y

同时将其他案例宏注释屏蔽

回到源码根目录BitMeta，执行hb build -f进行编译。

运行结果

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:

the soil humidity value is 113.6

the soil humidity value is 79.2

the soil humidity value is 138.5

the soil humidity value is 122.3

the soil humidity value is 141.1

the soil humidity value is 82.6

**第六章：驱动编程-SPI全彩灯带的使用**

简介

SPI（Serial Peripheral Interface） 协议是由摩托罗拉公司提出的通讯协议，即串行外围设备接口，是一种高速全双工的通信总线。它被广泛地使用在 ADC、LCD 等设备与 MCU 间，要求通讯速率较高的场合。

芯片的管脚上只占用四根线。 MISO： 主器件数据输出，从器件数据输入。 MOSI：主器件数据输入，从器件数据输出。 SCK： 时钟信号，由主设备控制发出。 NSS（CS）： 从设备选择信号，由主设备控制。当NSS为低电平则选中从器件。

**一、接口说明**

SpiOpen()

DevHandle SpiOpen(const struct SpiDevInfo \*info);

描述：

spi open获取SPI设备的句柄。

名字 描述

info 指向SPI设备信息的指针

SpiTransfer()

int32\_t SpiTransfer(DevHandle handle, struct SpiMsg \*msgs, uint32\_t count);

描述：

uart transfer SPI设备的自定义传输函数。

名字 描述

handle 指向SPI设备句柄的指针

msgs 要传输的数据的指针

count 消息结构数组的长度

SpiClose()

void SpiClose(DevHandle handle);

描述：

spi close关闭SPI设备的句柄。

**二、功能设计**

将如下图所示的全彩灯带用RJ11 线接到主板的第8口，主控MCU使用SPI控制灯带的亮灭。



**三、编译调试**

修改 vendor/openvalley/niobeu4/kernel\_configs/debug.config 文件

在文件末尾添加宏 LOSCFG\_NIOBEU4\_APPLICATION\_b15=y

同时将其他案例宏注释屏蔽

回到源码根目录BitMeta，执行hb build -f进行编译。

运行结果

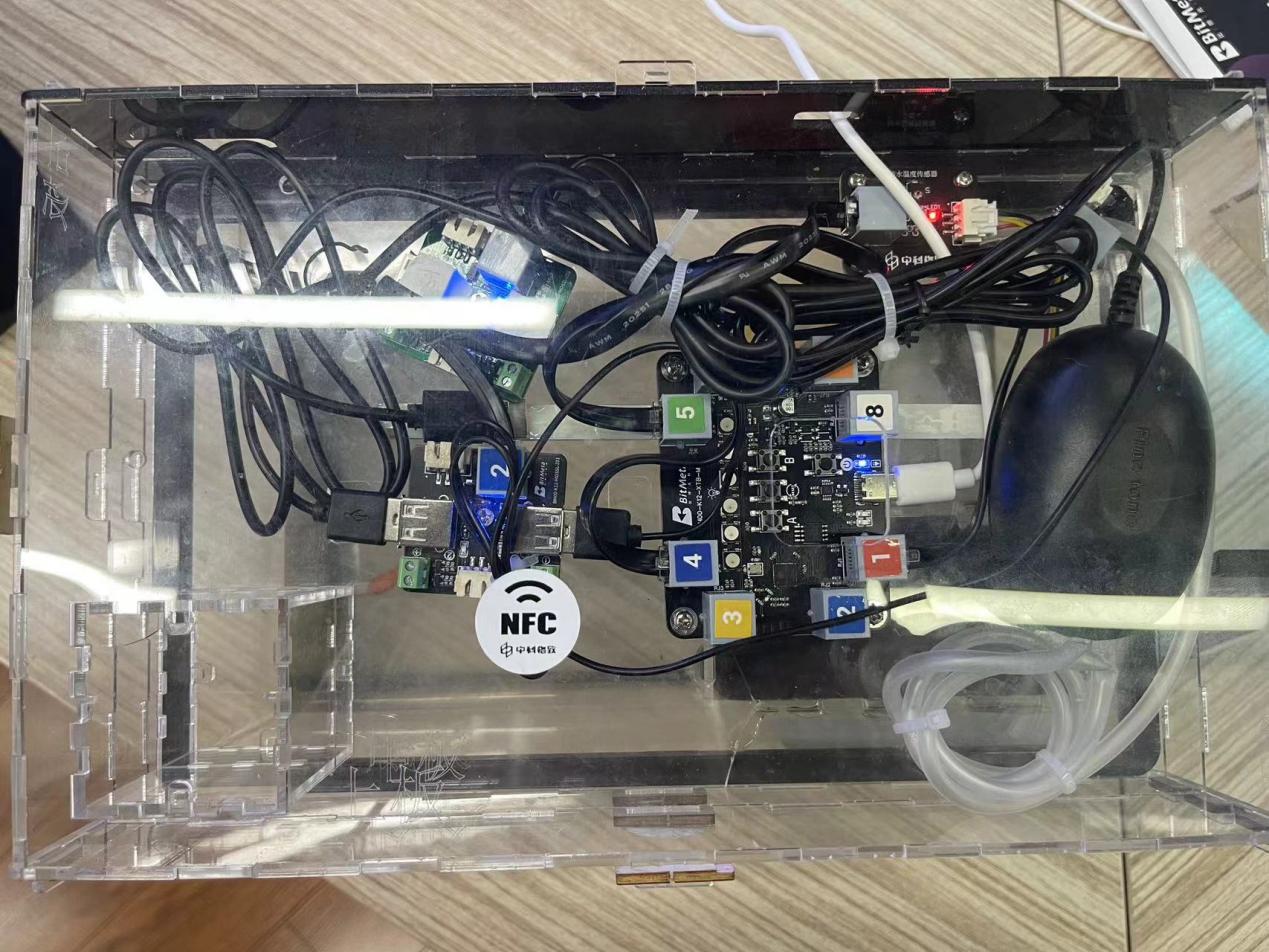
示例代码编译烧录代码后，按下开发板的RESET按键，查看灯带的亮灭。

**第七章：综合实验-智能编程鱼缸**

简介

本章我们使用之前学过的知识，结合BMP600主板加传感器：水泵、气泵、加热棒、防水温度计、全彩灯带、继电器，来实现智能编程鱼缸的功能，如定时换氧、自动调节水温等功能。智能鱼缸加上结构后的成品如下图所示。





**一、编译调试**

修改 vendor/openvalley/niobeu4/kernel\_configs/debug.config 文件

在文件末尾添加宏 LOSCFG\_NIOBEU4\_APPLICATION\_b17=y

同时将其他案例宏注释屏蔽

回到源码根目录BitMeta，执行hb build -f进行编译。

运行结果

示例代码编译烧录代码后，按下开发板的RESET按键，查看鱼缸相关功能。