机芯智能
MACHINE INTELLIGENCE

入门编程指南

版本 V1.0

免责申明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

文中所得测试数据均为机芯实验室测试所得，实际结果可能略有差异。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

最终解释权归深圳市机芯智能有限公司所有。

注 意

由于产品版本升级或其他原因，本手册内容有可能变更。深圳市机芯智能有限公司保留在没有任何通知或者提示的情况下对本手册的内容进行修改的权利。本手册仅作为使用指导，深圳市机芯智能有限公司尽全力在本手册中提供准确的信息，但是深圳市机芯智能有限公司并不确保手册内容完全没有错误，本手册中的所有陈述、信息和建议也不构成任何明示或暗示的担保。

目　　录

1. 开发环境搭建.....4
1.1 开发环境选择.....4
1.2 选择交叉编译器.....4
2. SDK 说明.....7
2.1 目录结构.....7
2.2 SDK 相关配置文件.....8
3. 编译工程文件.....11
3.1 编译指令.....11
3.2 示例工程编译.....12
4. 烧录固件.....14

1. 开发环境搭建

1.1 开发环境选择

Windows 环境： Cygwin + GCC

Linux 环境： Ubuntu14.2 以上 + GCC

1.2 选择交叉编译器

交叉编译器选择: gcc-arm-none-eabi-4_9-2015q2

Windows 版本

https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q2-update/+download/gcc-arm-none-eabi-4_9-2015q2-20150609-win32.zip

Linux 版本

https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q2-update/+download/gcc-arm-none-eabi-4_9-2015q2-20150609-linux.tar.bz2

（1）Linux Toolchain 安装与配置

A、下载 Linux 版本的工具链压缩包

gcc-arm-none-eabi-4_9-2015q2-20150609-linux.tar.bz2

B、进入控制台程序，将压缩包复制到在个人指定目录下，并解压

```
#tar -xvf gcc-arm-none-eabi-4_9-2015q2-20150609-linux.tar.bz2 ./
```

C、将交叉编译器添加到环境变量中，在控制台输入 vim ~/.profile,在文件中添加

```
export PATH=$PATH:/编译器的实际路径/gcc-arm-none-eabi-4_9-2015q2/bin
```

D、保存后退出执行 source ~/.profile 脚本，更新环境变量配置。

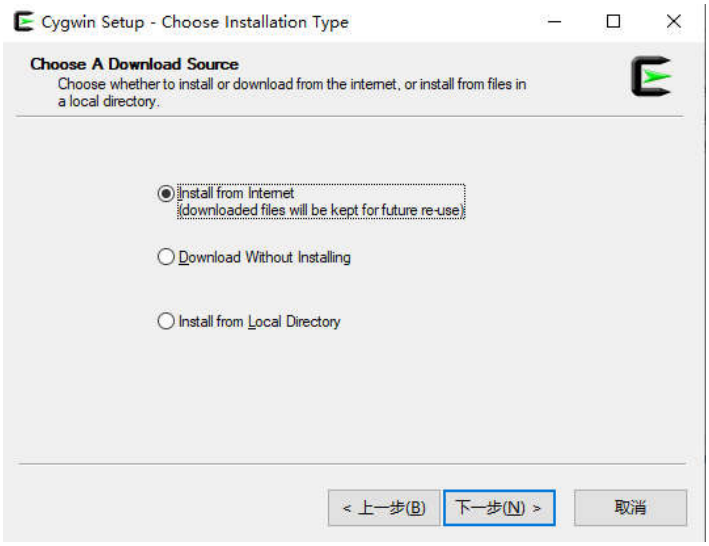
（2）Windows Toolchain 安装与配置

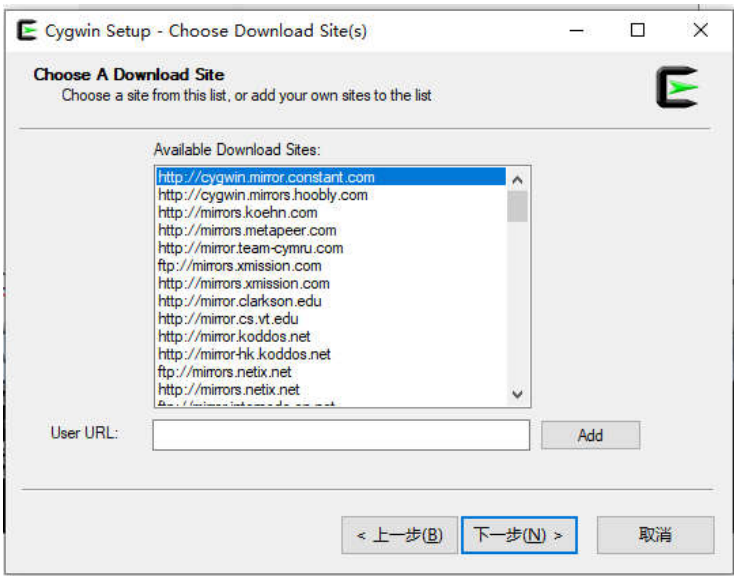
A、安装 Cygwin 开发环境

i、Cygwin 开发环境下载 Cygwin.7z

Cygwin 官网下载地址：https://cygwin.com

ii、下载得到 setup-x86.exe 文件，运行该文件，有三种安装模式选择：一是 Install from Internet，这种模式直接从 Internet 安装，适合网速较快的情况；二是 Download Without Installing，这种模式只从网上下载 Cygwin 的组件包，但不安装；三是 Install from Local Directory，这种模式与上面第二种模式对应，当你的 Cygwin 组件包已经下载到本地，则可以使用此模式从本地安装 Cygwin。选择适合的一种进行安装，下图以第一种为例。

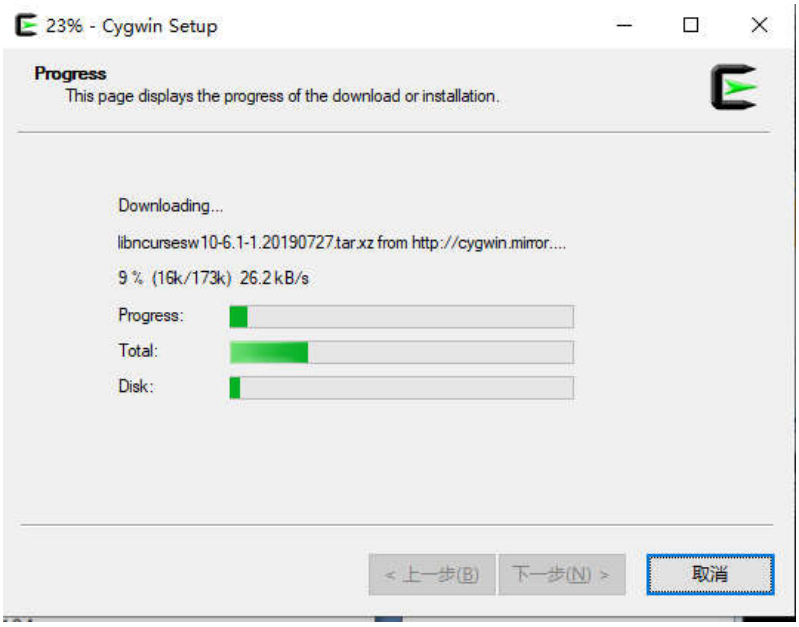




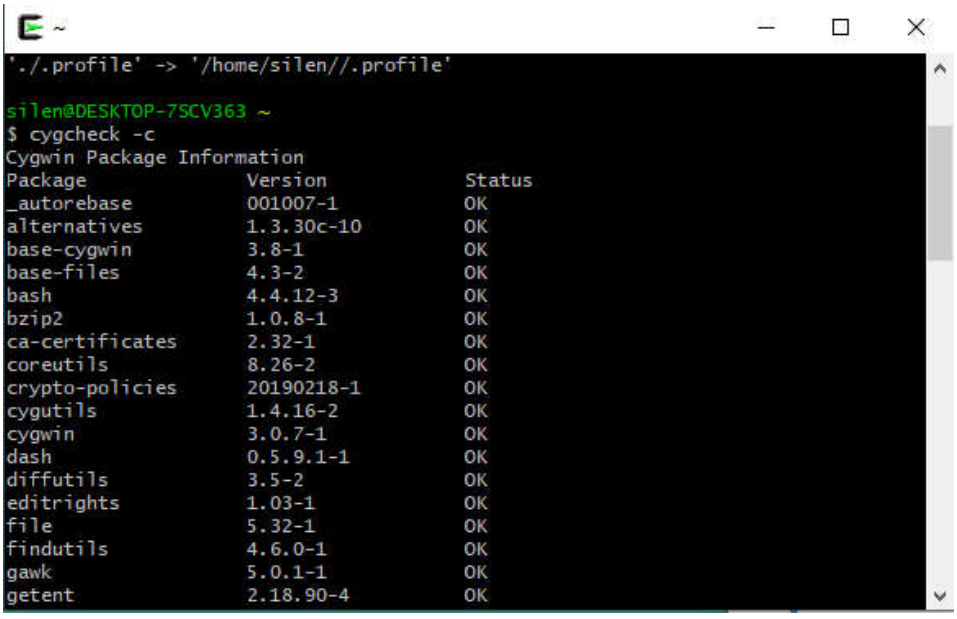
iii、根据提示输入安装路径，例如安装至 C 盘根目录: C:\

(注意：安装包存放路径及安装路径不可存在中文以及空格，安装路径必须是已存在的路径，否则无法安装)

iv、按提示一步步进行安装，安装完毕后桌面会产生图标，表示 Cygwin 组件已安装完成。安装确认及进度如下图所示：



vi、验证 Cygwin 安装完整性，双击桌面 Cygwin 图标进入 Cygwin 终端，输入 cygcheck -c 命令，系统将列出已安装组件，使用此工具包安装的 Cygwin 自动安装了常用工具，满足开发的基本需求。



B、Windows Toolchain 安装与配置

I、下载工具链压缩包并解压到个人指定的安装目录

II、修改系统启动配置文件~/.profile（没有的话自己添加），将 GCC 执行文件路径添加到 PATH 环境变量中，如下所示：

```
export PATH=$PATH:/编译器的实际路径/gcc-arm-none-eabi-4_9-2015q2/bin
```

III、在 console 执行 source ~/.profile 脚本，更新环境变量配置（下次重新登录时不需要）。

2. SDK 说明

2.1 SDK 下载

SDK 下载地址：https://github.com/XradioTech/xradio-skylark-sdk

2.2 目录结构

```
.
├── bin                          # bin文件目录，存放预置bin文件
├── chip.mk
├── config.mk
├── gcc.mk
├── include                      # 头文件目录，存放模块对外头文件
├── lib                          # 库文件目录，存放预置库文件和编译生成的库文件
│   ├── libaac.a
│   ├── libadt.a
│   └── ......
├── project                      # 工程总目录
│   ├── bootloader               # bootloader工程
│   ├── common                   # 工程公用代码
│   ├── demo                     # 演示工程总目录，该目录下每个目录对应一个工程
│   │   ├── hello_demo
│   │   ├── wlan_demo
│   │   └── ......
│   ├── example                  # 示例工程总目录，该目录下每个目录对应一个工程
│   │   ├── uart
│   │   ├── wlan
│   │   └── ......
│   ├── image_cfg
│   │   └── image.cfg            # 默认镜像配置文件
│   ├── linker_script
│   │   └── gcc
│   │       ├── appos.ld         # 工程默认链接脚本
│   │       └── bootloader.ld    # bootloader链接脚本
```

```
|       ├──── project.mk
|       └──── ......
├──── src
|       ├──── driver
|       |       ├──── chip                  # 芯片外设驱动
|       |       └──── component             # 扩展外设驱动
|       ├──── image                       # image模块
|       ├──── kernel                      # 内核
|       ├──── ota                         # OTA模块
|       └──── ......
|   ├──── ......
└──── tools                             # 镜像打包及烧录等工具
```

2.3 SDK 相关配置文件

A、配置文件列表

sdk-code/config.mk - 全局配置文件，配置一些全局参数，一般不需要修改

sdk-code/gcc.mk - 全局的编译规则，一般只需要修改CC_DIR的定义，将其指向对应路径

sdk-code/src/lib.mk - 模块（library）通用编译规则，一般不需要修改

sdk-code/project/prjconfig.mk - 工程默认配置选项，一般不需要修改

sdk-code/project/project.mk - 工程通用编译规则，一般不需要修改

sdk-code/project/xxx/gcc/localconfig.mk - 工程配置选项（最终定义，覆盖其他定义）

sdk-code/project/xxx/gcc/Makefile - 工程Makefile

sdk-code/project/xxx/prj_config.h - 工程功能属性配置文件，由其他c文件所用

B、工具链配置

相关文件

- sdk-code/gcc.mk

打开 sdk 源码下的 gcc.mk 文件，更改 CC_DIR 变量的值为个人所安装的交叉编译的 bin 文件夹所在的路径，如下所示：

```
# ------------------------------------------------------------------------------
# cross compiler
# ------------------------------------------------------------------------------
CC_DIR := ~/gcc-arm-none-eabi-4_9-2015q2/bin
CC_PREFIX := $(CC_DIR)/arm-none-eabi-
```

C、工程配置

相关文件

- 公共配置文件 sdk-code/config.mk
- 工程配置文件 sdk-code/project/prjconfig.mk
- 工程私有配置文件 sdk-code/project/xxx/gcc/localconfig.mk

其中，根据自己的需求进行配置只需要修改 localconfig.mk 文件即可，相关配置如下（根据自己的需求进行配置）：

```
# ------------------------------------------------------------------------
# override global config options
# ------------------------------------------------------------------------
# enable/disable wlan station mode, default to y
export __CONFIG_WLAN_STA := y
# enable/disable wps for wlan station mode, default to n
export __CONFIG_WLAN_STA_WPS := n
# enable/disable wlan hostap mode, default to y
export __CONFIG_WLAN_AP := y
# enable/disable XIP, default to y
export __CONFIG_XIP := y
# enable/disable OTA, default to n
export __CONFIG_OTA := y
```

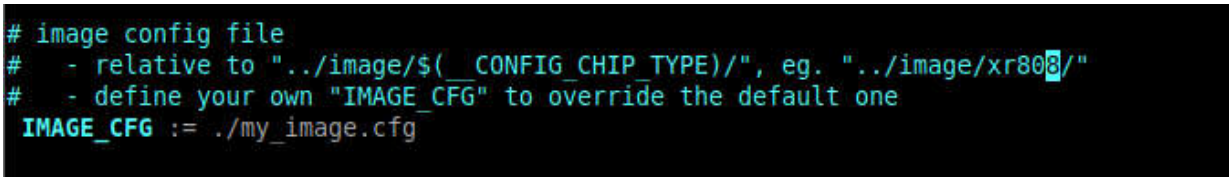
D、打包设置

相关文件

- 通用配置文件 sdk-code/project/image_cfg/xr808/image.cfg
- 专用配置文件 sdk-code/project/xxx/gcc/Makefile 中指定

Image.cfg 是工具编译完成之后打包固件使用的配置文件，内容如下：

可以在 Makefile 中自定义打包固件的配置文件路径，例如，在 sdk-code/project/demo/wlan_demo/image/xr808/下创建一个 my_image.cfg 配置文件，想要使用这个文件进行打包，需要在 sdk-code/project/demo/wlan_demo/gcc/Makefile 中，将该配置文件的路径添加进去，如下图所示，不修改 Makefile 的话默认使用的是 sdk-code/project/image_cfg/xr808 下 image.cfg。



注意：如果使用的是相对路径，一定要注意当 Makefile 执行到此处时已经进入 sdk-code/project/demo/wlan_demo/image/xr808/这个路径下了，所以配置路径的时候要以 sdk-code/project/demo/wlan_demo/image/xr808/ 这个路径为当前路径，添加你的 image.cfg 配置文件所在的路径，例如我的 my_image.cfg 文件当前存放在 sdk-code/project/demo/wlan_demo/image/xr808/这个路径下，所以我配置的时候直接./my_image.cfg 下即可。用绝对路径就不必考虑这个问题。（大部分情况建议使用默认的 image.cfg 配置文件）

Image.cfg 字段内容含义如下：

magic - 固件魔数，表头标识，固定为 AWIH
version - 版本号
count - 打包文件列表即 section 字段中文件的数目
section - 文件列表
id - 该段固件的 ID， ID 定义参考在 sdk-code/include/sys/image.h 中定义

bin - 该段固件 bin 文件名称，表示当前 bin 文件所代表的固件，名字不能超过 16 字符

cert - 该固件 OEM 签名证书，如果没有证书填 null，名字不能超过 16 字符

flash_offs - 该段固件存放在 FLASH 中的位置偏移，以 KB 为单位

sram_offs - 该段固件加载到 SRAM 中的地址偏移，以 B 为单位， 0xFFFFFFFF 为无效值

ep - 该段固件的 ENTRY POINT， 0xFFFFFFFF 为无效值，使用时忽略

attr - 表示该段固件属性，暂未使用，模式配 1

E、连接配置文件

相关文件

- 通用配置文件 sdk-code/project/linker_script/gcc/xr808/appos.ld
- 专用配置文件 sdk-code/project/xxx/gcc/Makefile 中指定

链接脚本文件的主要目的是描述如何将输入文件中的 section 映射到输出文件中，并控制输出文件的存储布局。

目前应用系统 code 和 data 如果不超过 448KB 的 SRAM 空间可参考同路径的 appos.ld 文件，如果超过则需要开启 FLASH XIP 功能，此时 ld 文件请参考同路径的 appos_xip.ld 文件，对 MEMORY 定义和 SECTION 中的.xip 字段进行内容修改即可。

3. 编译工程文件

3.1 编译指令

```
$ cd ${prj_gcc_path}     #进到事例工程的目录的gcc文件夹下 eg. cd project/wlan_demo/gcc
$ make config       # make config后会自动运行`./configure.sh`选择芯片型号和时钟频率
$ make config_clean        #清除配置信息
$ make lib                 #构建库并将其复制到”lib”文件夹下
$ make lib_clean           #删除”make lib”生成的”src”中的文件
$ make lib_install_clean #删除由make lib生成的” lib”中的库
$ make                     # 编译生成二进制文件
```

$ make clean	# 删除”make”编译生成的二进制文件
$ make image	#创建生成镜像文件
$ make image_clean	# 删除”make image”生成的镜像文件
$ make objdump	#生成反汇编文件
$ make build	# `make lib && make && make image`都执行
$ make build_clean	# `make image_clean clean lib_clean lib_install_clean`都执行

3.2 示例工程编译

以 wlan_demo 为例：

在编译之前先确定下交叉编译器的路径是否已经更改过来了

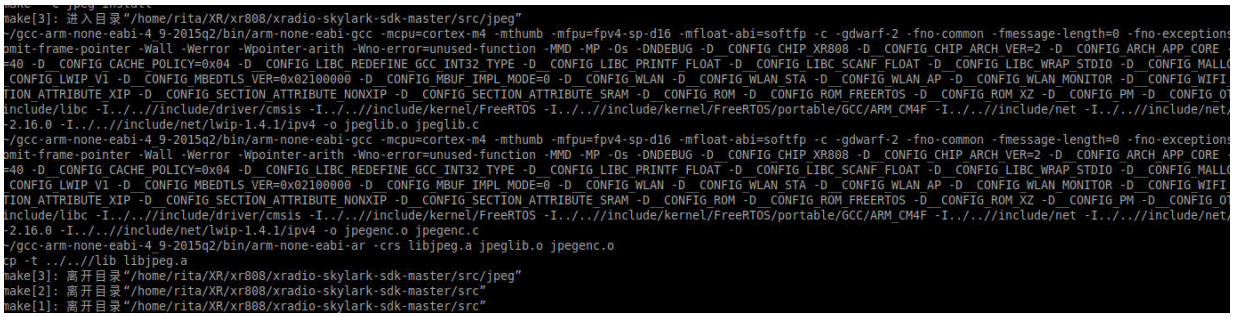
```
rita@rita:xradio-skylark-sdk-master$ ls
bin  chip.mk  config.mk  configure.sh  gcc.mk  include  lib  project  README.md  src  tools
rita@rita:xradio-skylark-sdk-master$ vim gcc.mk
```

```
# ------------------------------------------------------------------------
# cross compiler
# ------------------------------------------------------------------------
CC_DIR := ~/gcc-arm-none-eabi-4_9-2015q2/bin
CC_PREFIX := $(CC_DIR)/arm-none-eabi-
```

接着进入 wlan_demo 的 gcc 文件夹所在的路径下，输入 make config，选择芯片的型号和时钟频率。

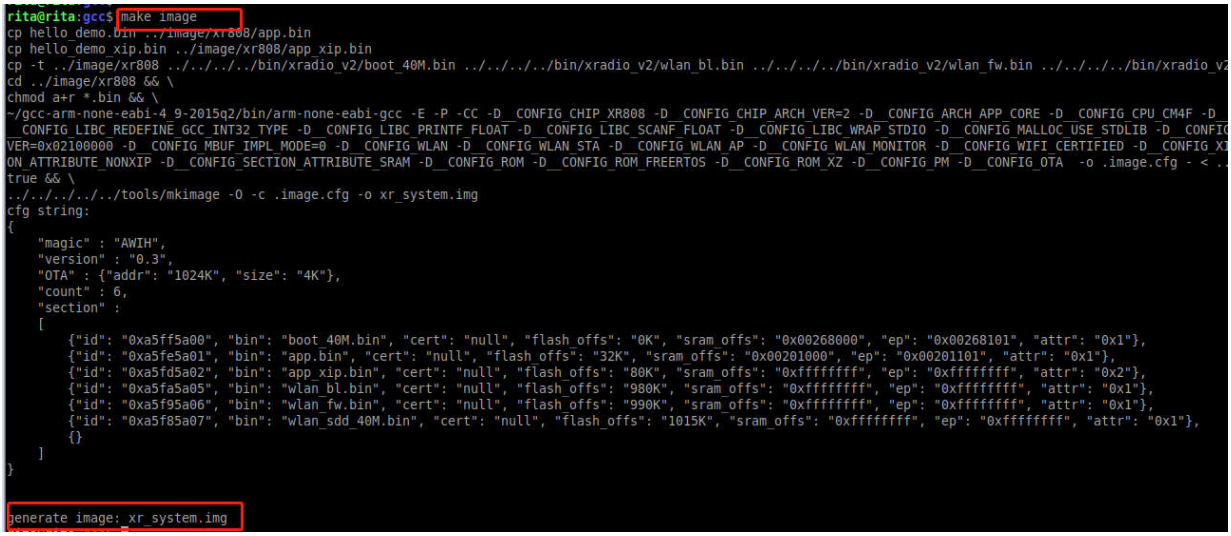
```
rita@rita:xradio-skylark-sdk-master$ cd project/demo/wlan_demo/gcc/
rita@rita:gcc$ make config
*
* XRADIO SDK Configuration
*
Chip
  1. XR872
  2. XR808
choice[1-2]: 2

External high speed crystal oscillator
  1. 24M
  2. 26M
  3. 40M
choice[1-3]: 3
rita@rita:gcc$
```

输入 make lib，编译库文件



输入 make 编译生成二进制文件

输入 make image 打包二进制文件成镜像文件 xr_system.img，这里使用的是 image_cfg/xr808/image.cfg 的默认打包配置文件，如果要使用自定义的需在 Makefile 中添加自定义的配置文件路径。

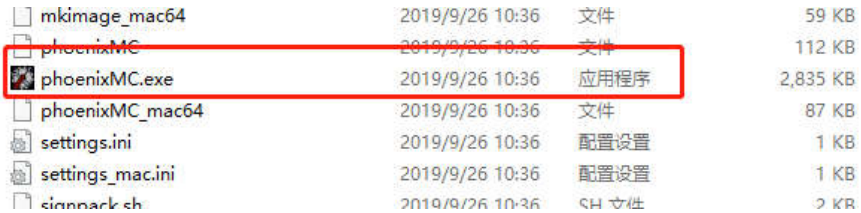


也可以直接使用 make build 指令一键编译生成镜像文件

```
../../../../../tools/mkimage -O -c .image.cfg -o xr_system.img
cfg string:
{
    "magic" : "AWIH",
    "version" : "0.3",
    "OTA" : {"addr": "1024K", "size": "4K"},
    "count" : 6,
    "section" :
    [
        {"id": "0xa5ff5a00", "bin": "boot_40M.bin", "cert": "null", "flash_offs": "0K", "sram_offs": "0x0026
        {"id": "0xa5fe5a01", "bin": "app.bin", "cert": "null", "flash_offs": "32K", "sram_offs": "0x00201000
        {"id": "0xa5fd5a02", "bin": "app_xip.bin", "cert": "null", "flash_offs": "80K", "sram_offs": "0xffff
        {"id": "0xa5fa5a05", "bin": "wlan_bl.bin", "cert": "null", "flash_offs": "980K", "sram_offs": "0xfff
        {"id": "0xa5f95a06", "bin": "wlan_fw.bin", "cert": "null", "flash_offs": "990K", "sram_offs": "0xfff
        {"id": "0xa5f85a07", "bin": "wlan_sdd_40M.bin", "cert": "null", "flash_offs": "1015K", "sram_offs":
        {}
    ]
}


generate image: xr_system.img
```

4. 烧录固件

烧录工具及工具库下载: phoenixMC_demo_release.zip(若该链接不可用，可在 sdk 源码的 tools 文件夹下找到烧录工具)

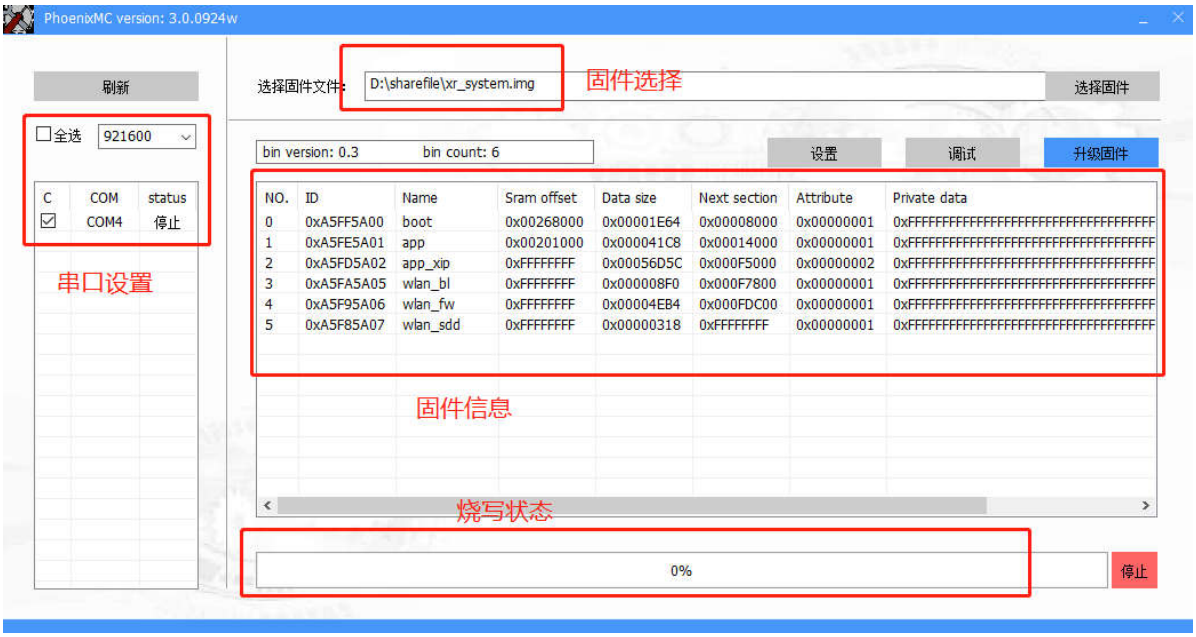


烧录工具为 PhoenixMC，其功能为通过串口将指定的合法固件文件烧录到进入烧录模式的目标主机中，其功能点包括串口设置，固件选择，升级确认，状态显示，固件信息等，其中：

串口设置：串口的波特率目前 有三种选择： 9600， 115200，921600，速度越快，烧录时间越少，具体情况根据具体的 FLASH 表现而定。

固件选择：选择要烧录的固件，既编译生成的“img”文件

烧写状态： 在状态提示栏里会提示当前的烧写步骤，当烧写成功时，进度条会达成100%的进度并显示为绿色，当烧写失败时，进度条显示为红色并报告错误。



PhoenixMC 工具具备可同时烧录多个目标主机的功能，将 PC 通过多个串口与多个设备相连，并刷新串口列表，同时勾选多个对应的 COM 口，点击升级时将同时对多个设备进行烧录，在烧录的过程中，点击不同的 COM 口时，在进度条上将显示各自 COM 口的烧录进度，最终实现多个设备的烧写。

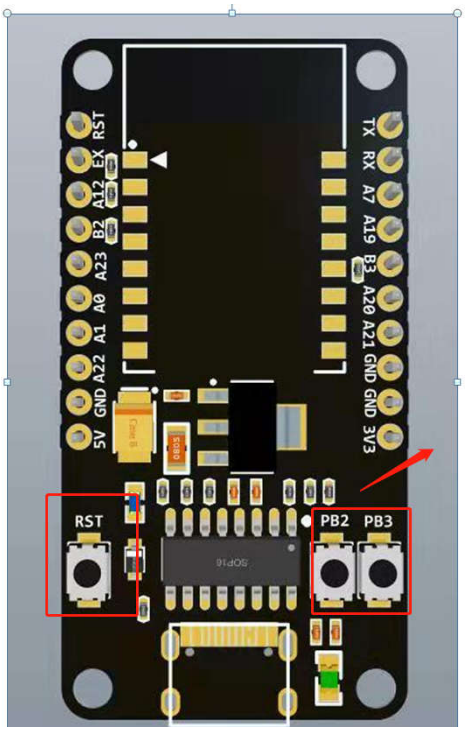
以 MICROWE 开发板为例具体烧录步骤如下：

（1）首先先进入升级模式，进入烧录模式有以下几种情况：

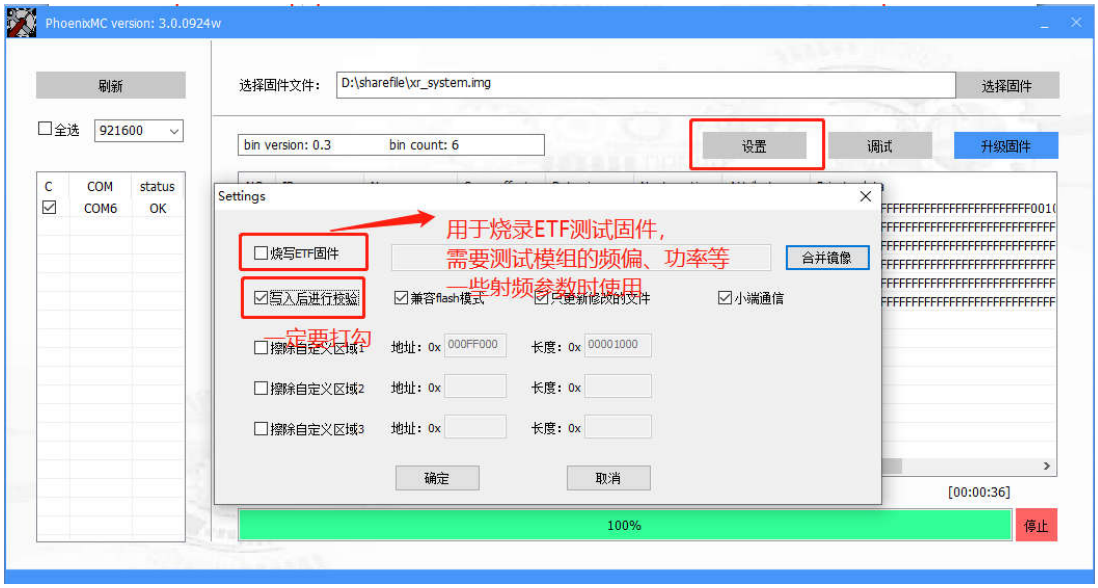
A、没有烧录过固件的模组，一上电自动进入升级模式

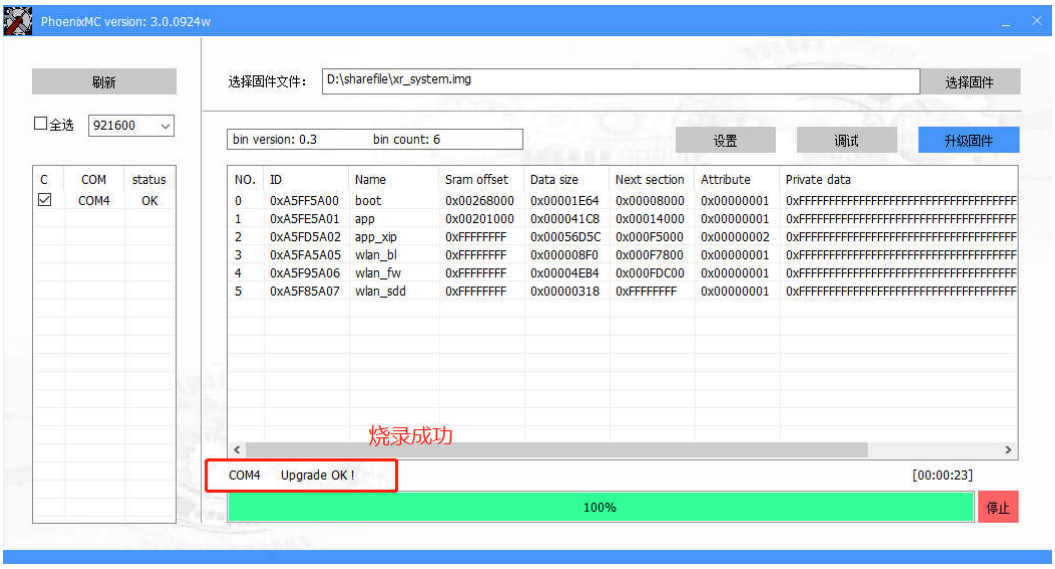
B、可以正常运行的模组，在串口控制台发送“upgrade”指令，收到“ok”，表示进入升级模式，或者按下复位键点击烧录工具的“升级固件”按钮，会自动发送“upgrade”给模组，进入升级模式。

C、以上两种情况都不行的情况下，先同时按下 PB2 PB3 按键，再按下 RST 键，接着松开 RST 键，再松开 PB2 PB3 按键，即可进入升级模式。

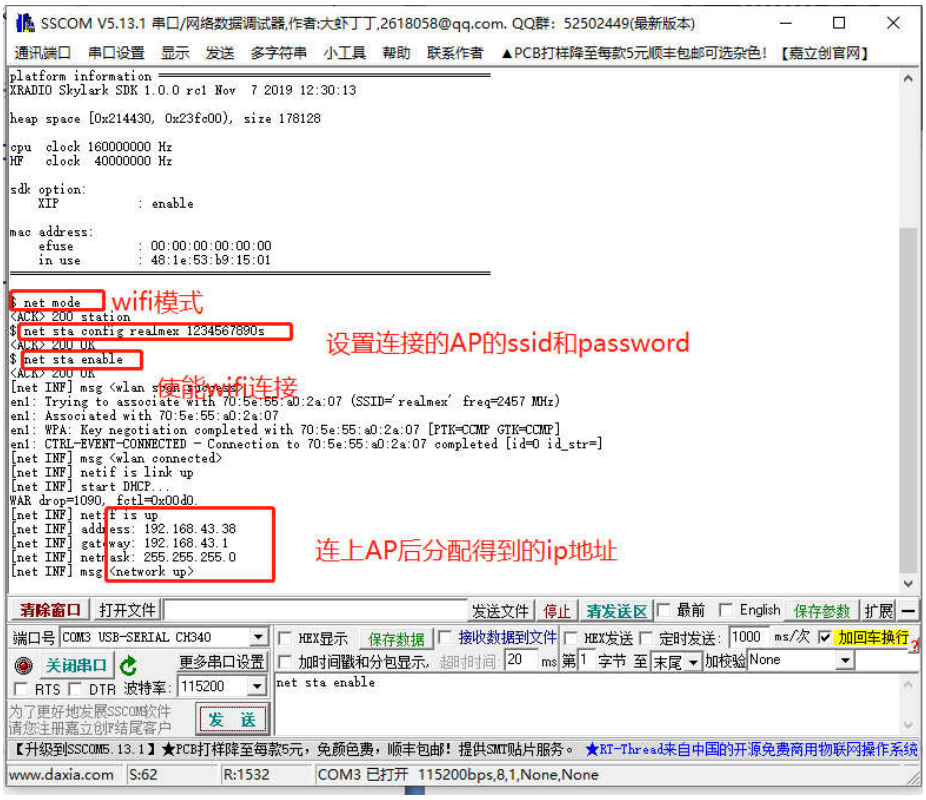


（2）串口设置、固件选择、设置，写入后校验一定要打勾，ETF 测试固件用于测试模组射频参数，如果有需要烧写 ETF 固件，勾选此项，同时会弹出对话框给用户选择所需的 ETF 固件。设置好后出现 upgrade ok，则表示烧录成功。





（3）通过串口查看调试信息，在串口控制台发送相关指令，即可实现对模组的相关配置。例如 net mode 是查看 wifi 当前模式是 STA 模式还是 AP 模式，net sta config <ssid><password> 是设置要连接的路由器的账号和密码,net sta enable 是 wifi 连接，通过串口打印的信息可以看出模组已连上路由，且获取到路由分配的 ip 地址。（更多控制台指令见 SDK 源码中 project/demo/xxx/command.c 文件，用户可根据自己的需求添加修改）



烧写失败原因：

（1）工具烧录 3%失败，报‘sync fail’

1. 该问题属于工具与设备协议同步失败
2. 同步失败的原因有几种，分别如下：

✧ 串口连接错误:

- ➢ 检查串口 TX,RX 是否连接正确；
- ➢ 检查 COM 口是否被其他工具占用；
- ➢ 使用低速波特率 115200 再次尝试

✧ 设备未进入到升级模式，确保设备进升级模式有以下几种：

- ➢ 从未升级的芯片或 Flash 上为空白内容的芯片上电会自动进入升级模式
- ➢ 按以下时序强制芯片上电进入升级模式：
  - 拉低 CHIP-PWD，即复位 IC
  - 同时拉低 PB02，PB03 两个 IO 为低电平
  - 释放 CHIP-PWD，即启动 IC，启动过程中检测到 PB02,PB03 的状态 IC 会主动进入升级模式
  - IC 启动后可以释放 PB02,PB03 管脚(外置 Flash 使用此 IO 时必须释放)
  - 使用工具开始升级
- ➢ 如果 IC 已经正常启动，并在控制台输入 upgrade<回车>后能够收到反应 OK，则芯片进入升级模式，此时断开串口工具即可升级

（2）升级过程中报‘write block x error’

升级工具与设备同步成功，但是烧录过程失败说明传输不可靠，检查线材波特率是否准确，或者降低波特率再次尝试烧录。