**Secret Status：Top Secret ( √ ) Secret** **( √ ) Restrict () Public ( √√ )**

**RKNanoD MP3 SDK软件开发手册**

**(Version 1.0)**

**Product R&D Dept.III**

**Rockchip Electronics Co.,Ltd**

**<www.rock-chips.com>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Status：**  **[√] development**  **[√]** **public**  **[√] modify** | **File Tag：** | **Software development kit guide** | |
| **Version：** | **1.0** | |
| **Author：** | **Ma Longchang** | |
| **Data：** | **2016-4-16** | |
| **Verify:** | **Zheng Yongzhi** | **2016-4-15** |

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Remarks** |
| **1.0** | **Ma Longchang** | **2016-4-15** | **Original** |
|  |  |  |  |
|  |  |  |  |

目录

[1 简述 5](#_Toc446332269)

[2 支持的特性 6](#_Toc446332270)

[3 目录结构 7](#_Toc446332271)

[4 软件配置 10](#_Toc446332272)

[4.1 调试配置 10](#_Toc446332277)

[4.2 存储设备选择配置 10](#_Toc446332278)

[4.3 应用模块选择 11](#_Toc446332279)

[5 编译工程 13](#_Toc446332280)

[6 Debug调试 15](#_Toc446332281)

[7 生成固件与烧写固件 16](#_Toc446332282)

[7.1 EMMC Flash固件生成工具 16](#_Toc446332290)

[7.2 SPI NOR Flash 固件生成工具 17](#_Toc446332291)

[7.3 烧写固件 18](#_Toc446332292)

[7.3.1 单机烧写工具 18](#_Toc446332293)

[7.3.2 工厂量产烧写工具 19](#_Toc446332294)

[8 资源打包 21](#_Toc446332295)

[8.1 图片资源打包 21](#_Toc446332296)

[8.2 字符串资源打包 22](#_Toc446332297)

[9 A-B核交互 25](#_Toc446332298)

[9.1 双核结构 25](#_Toc446332299)

[9.2 Mailbox简介 25](#_Toc446332300)

[9.3 Mailbox的使用 26](#_Toc446332301)

[9.4 A-B核通过Mailbox的交互 27](#_Toc446332302)

[9.4.1 解码部分 27](#_Toc446332303)

[9.4.2 文件操作部分 33](#_Toc446332304)

[10 增加新的解码库 39](#_Toc446332305)

[10.1 添加常规格式音频解码 39](#_Toc446332306)

[10.2 添加用户支持的音频解码 45](#_Toc446332307)

# 简述

## 1.1 SDK 概述

RKNanoD MP3 SDK是一个基于RKNanoD 双核Cortex-M3芯片的多媒体音频软件开发包。该SDK包含了音频解码，编码，图片，FM，电子书等功能。支持24bit HiFi无损格式音频解码以及16bit/24bit高质量录音功能。

## 1.2设计目标

RKNanoD MP3 SDK 是为了解决广大HiFi音频播放的应用需求，提供一个开发周期短、可靠性高、品质优良的产品解决方案。

# 支持的特性

RKNanoD SDK主要是用作为HIFI Mp3播放器使用。目前支持的音频格式有：MP3、WAV、AAC、OGG、FLAC、APE、HIFI FLAC、HIFI APE、HIFI ALAC、DSD、FM收音机、WAV编码（录音）、MP3编码（录音）、电子书（txt）等。详见特征见下表。

|  |  |  |
| --- | --- | --- |
| **特性** | **详细特征** | **备注** |
| **显示** | 64K Pixel Color LCD | Max to 320x240 dot(160×128/320×240 ) |
| **USB** | FS/HS Device 2.0 |  |
| **USB协议** | MSC |  |
| **存储** | EMMC Flash | Max to 256GB |
| SPI Flash |  |
| SD Card | Max to 256GB |
| **音频解码器** | 16 bits | MP3 / WAV / APE / FLAC / AAC / OGG |
| 24 bits | MP3 / WAV / AAC / OGG  HIFI APE / HIFI FLAC / HIFI ALAC |
| **EQ** | Effect Sound |  |
| **媒体库** | Title/Artists / Albums / Genres | Max to 8192 Songs |
| **ID3** | ID3V1 / ID3V2.3 / ID3V2.4 / Album art | Album art only support ID3JEPG |
| **收音机** | FM Radio | China: 87.0MHz – 108.0MHz Japan: 76.0MHz – 108.0MHz |
| **录音** | WAV Encode | 8KHz~192KHz，16bit/24bit |
| MP3 Encode | 8KHz~48KHz，16bit |
| **电子书** | \*.txt file / Book Mark |  |

# 目录结构

RKNanoD MP3 SDK目录结构分为：

**Common**：包括OS、驱动、Fat32 文件系统、音频、视频解码库等底层开发包。

**SDK\_160\_128**：资源、脚本、工程目录、固件存放以及应用相关等。

下面表格中将简单描述一下各个不同目录的用途。

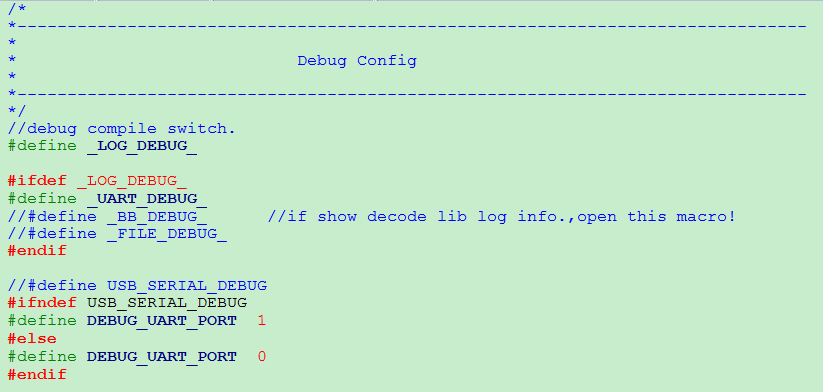
|  |  |  |  |
| --- | --- | --- | --- |
| **目录名** | **子目录** | **子目录** | **详细描述** |
| Common | BBsystem |  | B Core解码相关的接口文件 |
| BootLoader |  | 启动代码，用来JTAG调试 |
| Codec | Audio | 音频解码器相关的解码库与API接口文件 |
| Image | 图片解码相关的解码库与API接口文件 |
| Radio | FM相关控制的接口文件 |
| UsbControl | USB控制相关API接口文件 |
| Video | 视频解码相关的解码库与API接口文件 |
| Display |  | 与显示相关的所有接口文件 |
| Driver |  | 各个不同外设驱动接口文件 |
| FileSys |  | Fat32文件系统相关的接口文件 |
| FwUpdate |  | 执行本地固件升级相关的接口文件 |
| Include |  | 全局相关的头文件 |
| Plug |  |  |
| SortFileInfo |  | 媒体库相关的排序接口文件 |
| System | Debug | 调试接口 |
| FileSeek | 文件搜索相关 |
| ModuleOverlay | 模块调度相关 |
| Os | 系统相关 |
| SysService | 外设检测等系统服务相关代码 |
| SDK\_160\_128 | Build |  | 编译工程目录 |
| Development |  | 固件生成与烧写相关的工具 |
| Firmware |  | 用于存放生成的下载到设备的固件 |
| Resource |  | 存放图片与字符串生成工具及生成后文件 |
| Scatter |  | 存放设备内存组织的脚本文件 |
| UI | Browser | 浏览设备中目录结构的应用接口文件 |
| ChargeWin | 充电相关的UI接口文件 |
| Dialog | 存放用来在LCD上显示对话框的接口文件 |
| MainMenu | 选择主应用模块的接口文件 |
| Medialib | 媒体库相关的接口文件 |
| MusicWin | 音乐播放器相关的应用UI接口文件 |
| PicWin | 图片摸相关的应用UI接口文件 |
| RadioWin | FM应用UI接口文件 |
| RecordWin | 录音应用UI接口文件 |
| SetMenu | 系统设置应用UI接口文件 |
| TextWin | 电子书应用UI接口文件 |
| USB | USB应用相关的UI接口文件 |
| VideoWin | 视频播放器应用相关的UI接口 |

# SDK软件配置

RKNanoD MP3 SDK支持可配置的软件应用。在“SDK\_160\_128/sysconfig.h”中详细列出了软件可控制的软件配置选项。用户可根据自身开发产品需求增减选择所需的功能应用。详细描述如下：



## 调试配置



Debug 调试配置，默认使用uart串口调试，串口调试中默认使用uart 1作为调试口。如上图，定义DEBUG\_UART\_PORT 为1。

其中需要注意：如果想要显示解码库中的log，需要打开\_BB\_DEBUG\_宏，这样解码相关的log将会打印出来。

## 存储设备选择配置

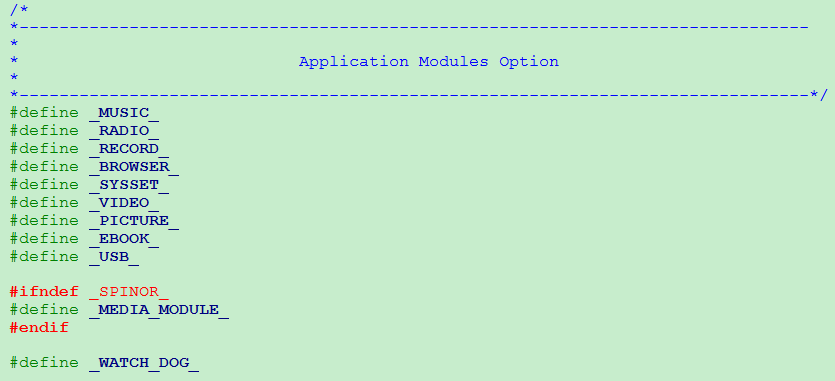
如下图，根据设备硬件的情况选择Flash配置。

* 1. 如果使用EMMC Flash，打开\_EMNC\_宏定义。如果使用SPI NOR Flash 打开\_SPINOR\_宏定义，二者只能选择其一。目前不支持NAND Flash。SD卡的支持可根据用户产品定义选择支持与否。
  2. 如果选择了使用EMMC Flash，可选择定义固件存放的位置，默认存放在EMMC Flash中，不容许修改。用户可选择定义资源文件的存放位置，默认存放在固件中。
  3. 如果选择了使用SPI NOR Flash，用户可选择固件存放的位置，默认存放在SPI Nor Flash中，不容许修改。需要根据SPI NOR Flash的存储容量大小，用户可选择定义资源文件的存放位置，如果SPI NOR Flash 容量比较大（大于8M）可选择将资源文件与固件打包放在Flash中，否则只能放在文件系统中（SD卡中）。
  4. 此处SDCARD\_PORT的定义根据用户自身硬件设计的不同选择不同的定义，如果硬件设计中将SDCARD的部分与EMMC电路共用，需要定义SDCARD\_PORT 为1，否则使用默认值 0。



## 应用模块选择

如下图，定义了NanoD SDK中支持的应用模块。用户可根据自身需求现在打开或关闭。



# SDK双核交互

RKNanoD是一款基于ARM cortex-M3的双核处理器。其中system core（A核）为主处理器，芯片上电后首先启动系统核，加载系统软件运行；calculate core（B核）为从处理器，由主处理器通过软件进行启动和关闭。在RKNanoD MP3 SDK中，system core用以运行系统管理、调度以及应用软件，calculate core作为从处理器进行音频解码、编码等算法运算处理。A、B核之间通过mailbox进行命令交互。RKNanoD为A、B核分配了可独立支配的使用内存，同时也支持A、B核相互访问相应的内存、外设等接口。

## 双核结构

RK-NanoD的双核架构如下面的框图。该框图简单描述了A 核作为系统核，B核作为运算核，以及A-B核之间通过Mailbox作为桥梁通讯的过程。



## Mailbox简介

Mailbox是一个简单的APB外设模块，它通过写命令操作产生一个中断在双核CPU之间进行通讯。Mailbox的寄存器可以被两个CPU通过APB接口访问。

Mailbox有以下特点：

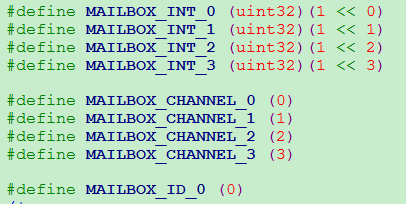
* 支持双核访问
* 支持4个通道，每一个通道包括了一个命令字，一个数组字寄存器和一位标识中断的标志位
* System core的4个中断，用以接收calculate core发出的请求
* Calculate core中有4个中断，用以接收system core发出的请求



Mailbox内部模块图

## Mailbox的使用

RKNanoD 中有一个mailbox，该mailbox有4个通道，每个通道都有独立可设的中断，每个通道的定义如下代码所定义的：



下面表格中详述了Mailbox每个通道的使用用途：

|  |  |  |
| --- | --- | --- |
| 通道号 | 用途 | 备注 |
| 0 | B核启动以及A核调整频率通知B核  时使用的通道 |  |
| 1 | 音量解码中，A核与B核命令与数据交互时使用的通道 |  |
| 2 | 文件操作时，A核与B核命令与数据操作交互时使用的通道 |  |
| 3 | 调试B核代码时使用的通道。主要是用来debug使用。 | B核的打印输出是将打印的内容通过该通道发送到A核，A核托管来打印输出的。 |

接口说明：

|  |  |
| --- | --- |
| 接口 | 说明 |
| MailBoxWriteB2ACmd | B核向A核写命令 |
| MailBoxWriteB2AData | B核向A核写数据 |
| MailBoxWriteA2BCmd | A核向B核写命令 |
| MailBoxWriteA2BData | A核向B核写数据 |
| MailBoxReadB2ACmd | B核读取A核的命令 |
| MailBoxReadB2AData | B核读取A核的数据 |
| MailBoxReadA2BCmd | A核读取B核的数据 |
| MailBoxReadA2BData | A核读取B核的数据 |

注意：Mailbox的命令发送必须与数据一起发送，不能只发送命令或数据中的一种。

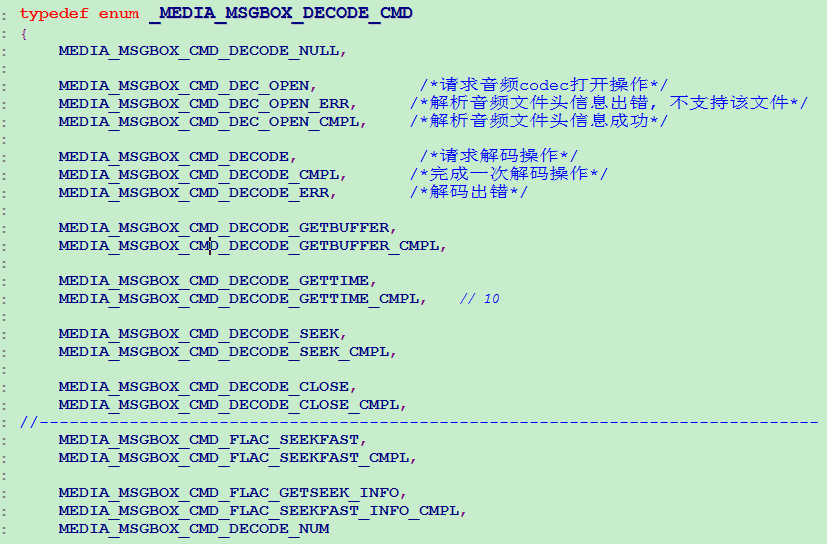
## 双核交互

本章节主要描述A-B核通过Mailbox相互交互的过程，将以音频解码为例，剖析A-B核通过mailbox相互通信，将解码命令与解码数据传递，完成整个解码的控制与交互。

### 解码控制交互命令

* 解码交互命令

首先看一下Audio\_main.h中定义的音频解码使用到的解码命令。



解码命令

具体命令的解释如下表：

|  |  |  |
| --- | --- | --- |
| 命令  （MEDIA\_MSGBOX\_CMD\_\*） | 说明 | 方向  （命令发送的方向A→B或B→A） |
| DECODE\_NULL | 空命令 | 命令字不能为“0” |
| DEC\_OPEN | 打开解码器 | A → B |
| DEC\_OPEN\_ERR | 打开解码失败 | B → A |
| DEC\_ OPEN\_CMPL | 打开解码成功 | B → A |
| DECODE | 请求B核解码 | A → B |
| DECODE\_CMPL | 解码完成 | B → A |
| DECODE\_ERR | 解码出错 | B → A |
| DECODE\_GETBUFFER | 请求B核解码数据 | A → B |
| DECODE\_GETBUFFER\_CMPL | 解码数据获取完成 | B → A |
| DECODE\_GETTIME | 请求获取当前解码时间 | A → B |
| DECODE\_GETTIME\_CMPL | 当前时间获取完成 | B → A |
| DECODE\_SEEK | 指定位置解码 | A → B |
| DECODE\_SEEK\_CMPL | 指定位置完成 | B → A |
| DECODE\_CLOSE | 请求解码关闭 | A → B |
| DECODE\_CLOSE\_CMPL | 解码关闭完成 | B → A |

A-B核的解码命令交互

如上表所述，A核向B核通过Mailbox发送解码相关的各种请求命令，B核负责解码运算完成之后给出结果，同时将结果和状态通过Mailbox发回给A核，这样一问一答的交互方式就是通过Mailbox为桥梁完成。

* 解码交互过程

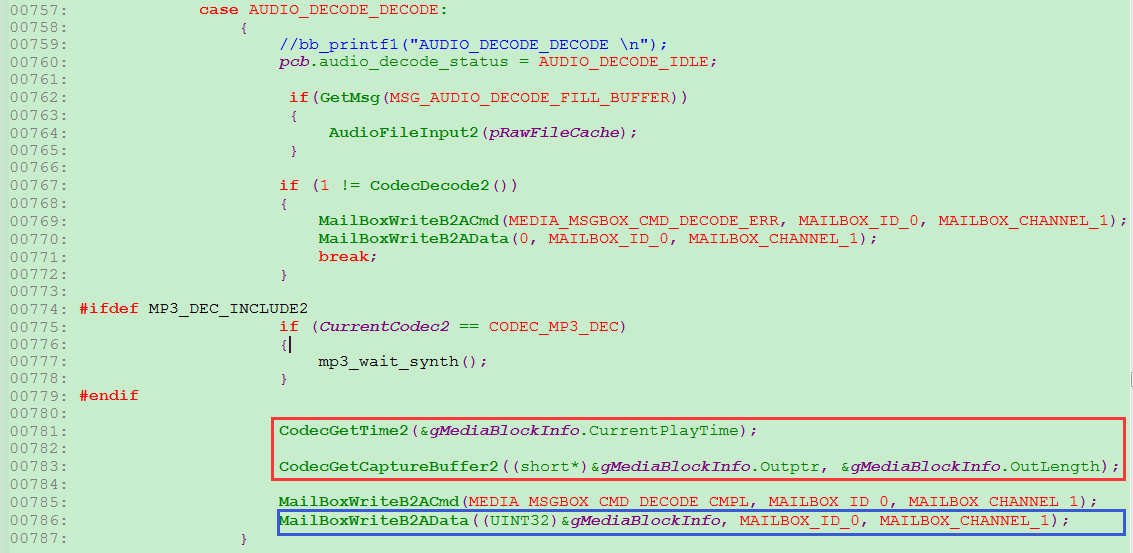
详细的A-B核交互过程如下图所示：



音频解码时A-B核通过mailbox的实际交互过程

注意:

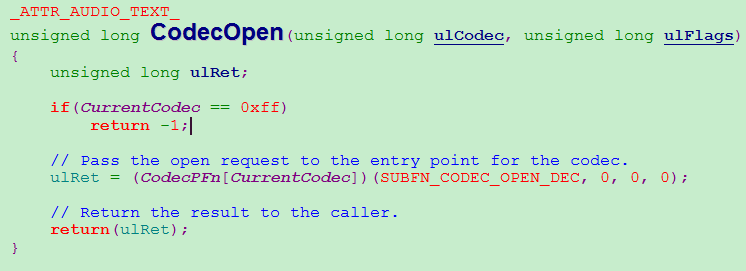
从上图中可以看到，在RKNanoD MP3 SDK中，没有获取时间的请求与回复，也没有解码buffer数据的请求与回复，这是因为获取时间的操作和解码后的数据buffer获取，均放在了DECODE的命令交互中，每完成一次解码，B核就将时间和解码的数据buffer存放在结构体中，同mailbox的data命令发送给A核。具体代码可以在main2.c中看到。如下：



B核解码过程

* 解码交互的示例代码

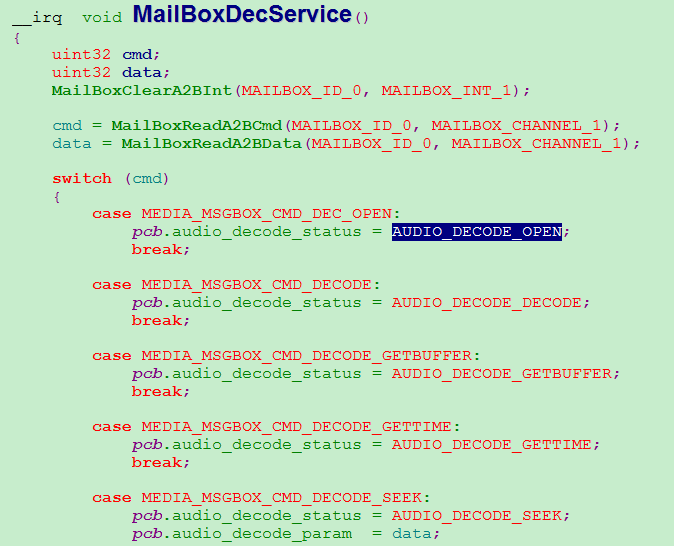
打开解码器（A核部分）：CodecOpen中调用不同解码器的SUBFN\_CODEC\_OPEN\_DEC参数。此处通过mailbox发送向B核发送MEDIA\_MSGBOX\_CMD\_DEC\_OPEN命令。



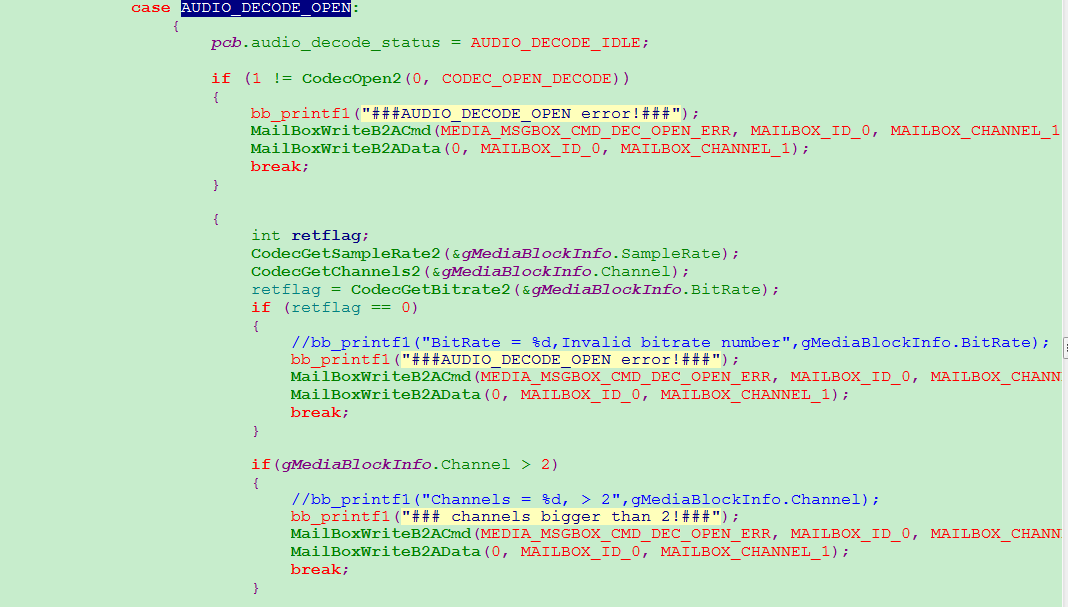


打开解码器（B核部分）：mailbox收到A核发送的命令后产生中断，B核代码根据接收命令处理相应流程。

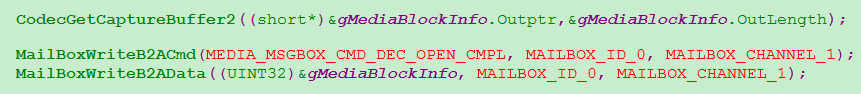
B核mailbox中断服务程序：



B核打开解码器出错处理：



B核打开解码器成功处理：

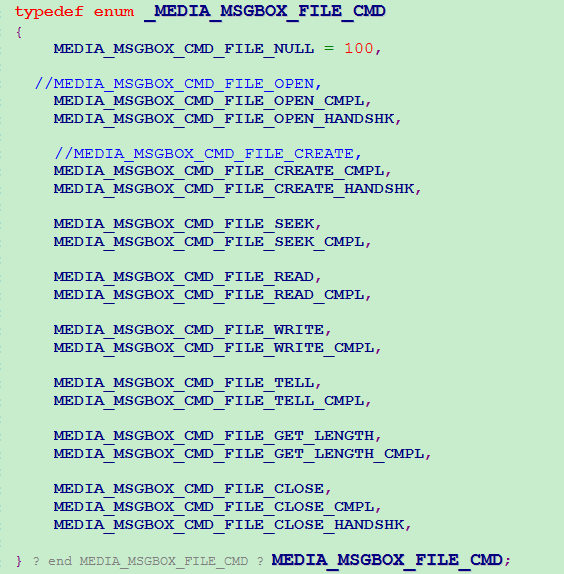


其他解码命令的交互类与此类似，这里不做赘述。

### 文件数据流交互命令

* 文件数据流操作命令

B核在解码过程中，需要A核传输音频压缩数据。A核从文件系统读取文件数据，通过命令交互方式将音频数据传送给B核，B核在解码过程中对文件进行解析，根据解码需求向A核请求信的数据。在文件file.h中定义了A-B核之间对文件操作的交互命令，这些命令也是通过mailbox作为桥梁发送与传递的。



文件命令

具体命令的解释如下表：

|  |  |  |
| --- | --- | --- |
| 命令  （MEDIA\_MSGBOX\_CMD\_\*） | 说明 | 方向  （命令发送的方向A→B或B→A） |
| FILE\_NULL | 空命令 |  |
| FILE\_OPEN\_CMPL | 文件打开完成 | A → B |
| FILE \_ OPEN\_HANDSHK | 文件打开完成确认 | B → A |
| FILE\_CREATE\_CMPL | 文件创建完成 | A → B |
| FILE\_CREATE\_HANDSHK | 文件创建确认 | B → A |
| FILE\_SEEK | 文件搜寻 | B → A |
| FILE\_SEEK\_CMPL | 文件搜寻完成 | A → B |
| FILE\_READ | 请求文件读取 | B → A |
| FILE\_READ\_CMPL | 文件读取完成 | A → B |
| FILE\_WRITE | 请求写入文件 | B → A |
| FILE\_WRITE\_CMPL | 文件写入完成 | A → B |
| FILE\_TELL | 文件当前位置请求 | B → A |
| FILE\_TELL\_CMPL | 文件当前位置完成 | A → B |
| FILE\_GETLENGTH | 请求文件大小 | B → A |
| FILE\_GETLENGTH\_CMPL | 文件大小获取完成 | A → B |
| FILE\_CLOSE | 请求文件关闭 | B → A |
| FILE\_CLOSE\_CMPL | 文件关闭完成 | A → B |
| FILE\_CLOSE\_HANDSHK | 文件关闭确认 | B → A |

A-B核的文件操作的交互命令

一般**文件操作都是B核向A核发送文件操作请求，A核接收到B核操作请求后做响应处理，再将处理结果回复给B核。**

* 文件数据流操作交互过程

1. B核之间通过mailbox交互文件操作的过程详见下图：



* 文件操作A、B核交互示例代码

这里以打开文件、文件读取为例简单说明A、B通过mailbox交互的过程。

**打开文件交互过程：**

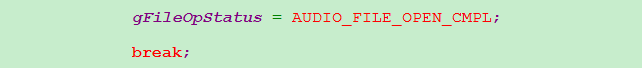
打开文件操作（A核部分）：A核先向B核发送文件打开的命令，以及文件打开后的文件句柄。



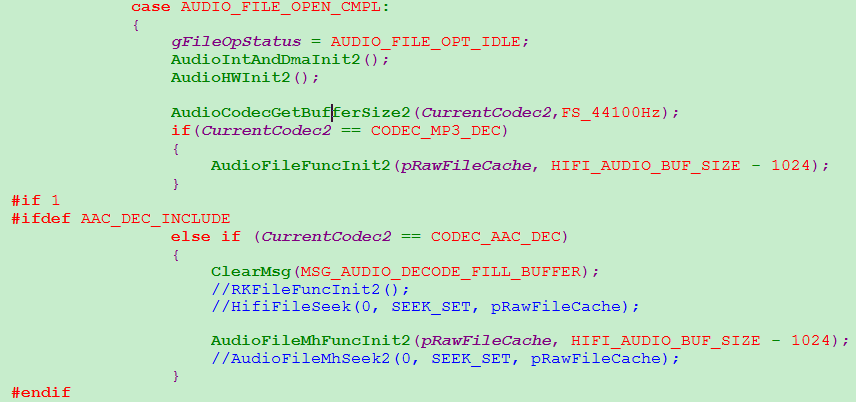
打开文件（B核部分）：mailbox收到A核发送的命令后产生中断，B核代码根据接收命令处理相应流程。

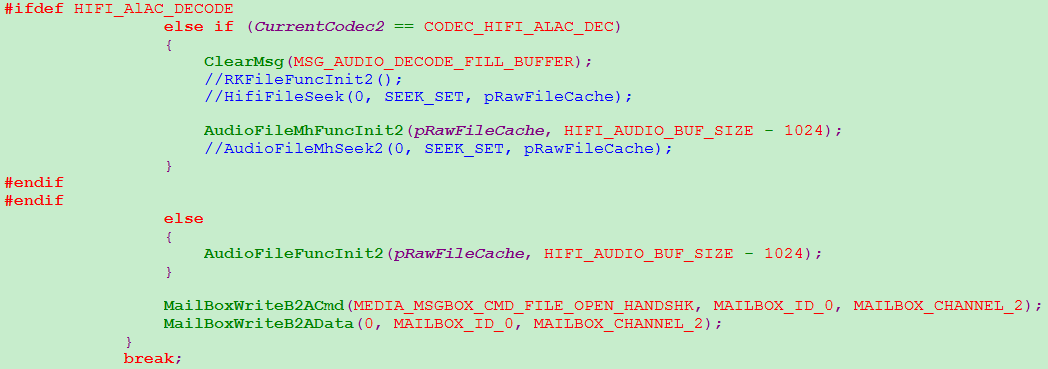
B核mailbox中断服务程序：





B核打开解码器接收到打开文件命令后，开始进行解码器相关初始化的处理：





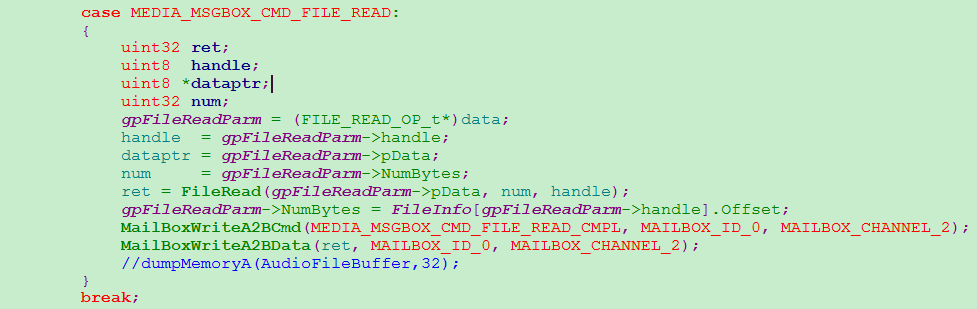
B核初始化处理完成之后向A核发送MEDIA\_MSGBOX\_CMD\_FILE\_OPEN\_HANDSHK 握手命令，表示B核接收到文件打开的操作，并相应做好了准备工作。A核接收到响应信号之后开始准备下一步动作。至此文件打开的A、B核交互操作完成。

**文件读取交互过程：**

文件读取（B核部分）：这里B核解码过程中发现解码数据不够，会向A核发送文件读取命令。如下：

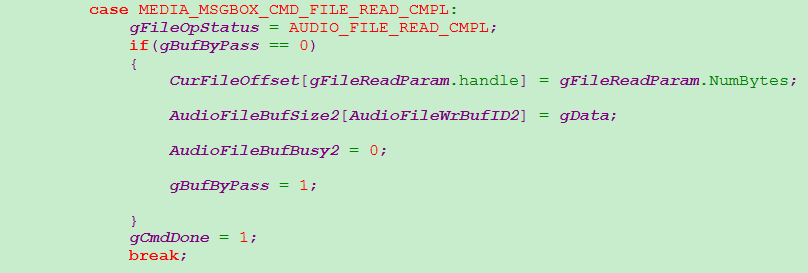


文件读取（A核部分）：A核接收到该命令后执行文件读取操作。在mailbox注册的文件操作的中断服务AudioDecodingInputFileBuffer中：



读取操作执行完成之后，A核会向B核发送MEDIA\_MSGBOX\_CMD\_FILE\_READ\_CMPL命令和读取的数据长度。

文件读取（B核部分）：此时B核接收到A核命令和数据后，将数据流写入自己的buffer中：



# 如何增加新的解码库

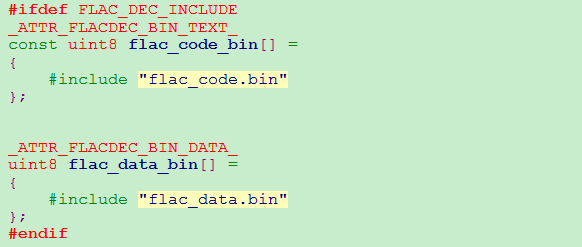
目前NanoD MP3 SDK软件包中支持的解码库格式有 16bit的MP3、WAV、AAC、OGG、FLAC、APE以及支持24bit的HIFI APE、HIFI FLAG、HIFI ALAC格式。实际应用中因为不同的客户还有其他音频格式可能需要支持，需要用户自己开发添加。本文档就是介绍怎样在NanoD MP3 SDK中快速的添加其他音频格式的解码库。

下文中分两中情况描述添加的过程。第一种情况，添加移植常规支持的音频格式。第二种情况，添加用户自定义开发的音频格式。

## 添加常规格式音频解码

添加常规格式音频解码，是指从已知的所支持的音频格式中添加需要支持的音频解码。此处以添加支持常见的FLAC音频格式为例，需要SDK修改地方如下：

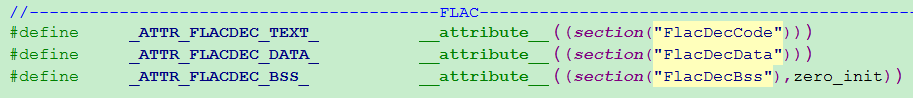
* + **BB\_Core.c** 添加B核编译后生成的flac的解码库二进制文件。

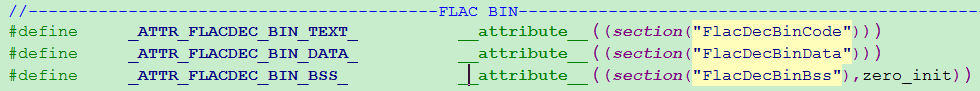


* + **Audio\_main.h** 中添加flac的代码段以及解码库段的定义，如下：

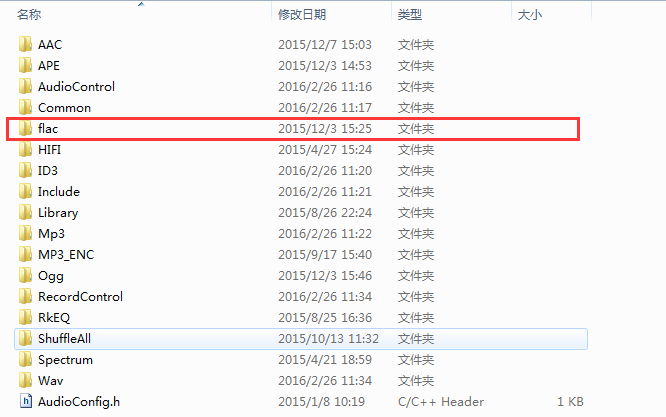
添加解码宏定义



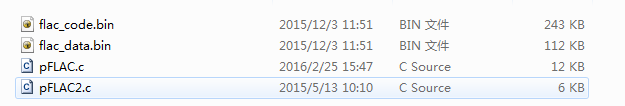




* + 在**Common\Codec\Audio** 文件夹下添加flac的解码库文件以及A 、B核用到的解码文件。



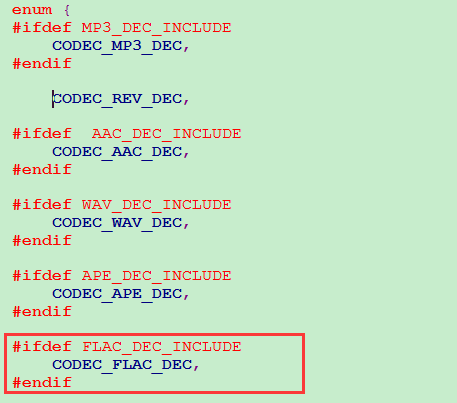
Flac文件夹中包含以下文件：解码库bin文件以及A、B核解码交互接口文件。



* + **Global.h** 中添加支持flac格式的后缀字符串“FLA”

****

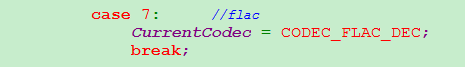
* + **Audio\_global.h**中添加一下Flac的codec类型：



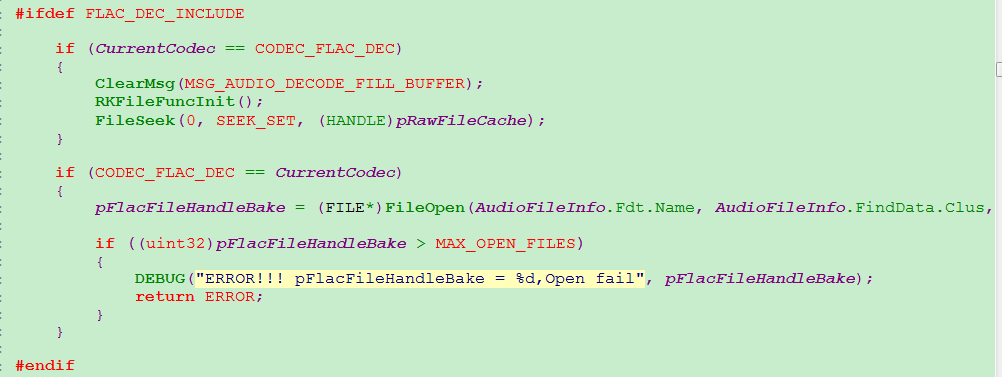
需要注意的是，A核与B核的codec ID号必须保持一致。

* + Audio\_Contol.c中

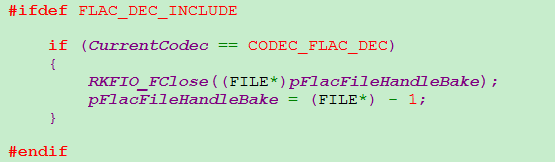
**AudioCodec**接口中添加flac格式类型的选择：



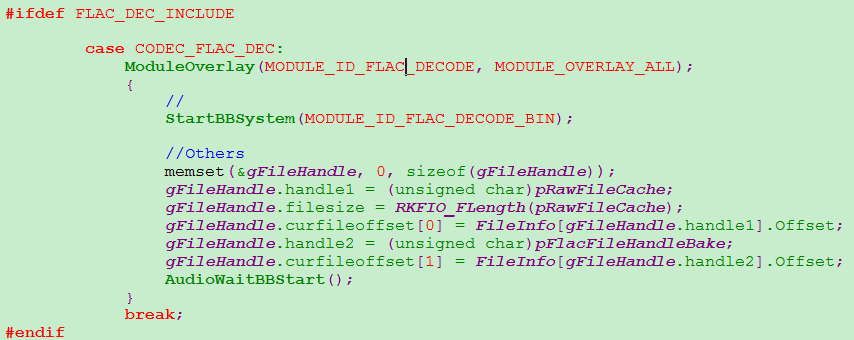
**AudioFileOpen**接口中添加flac格式的文件打开处理：



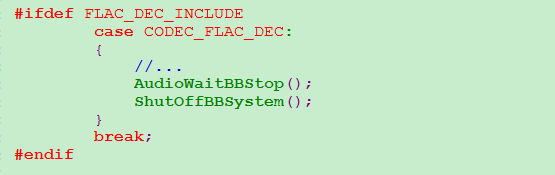
**AudioFileClose**接口中添加flac格式的文件关闭处理：



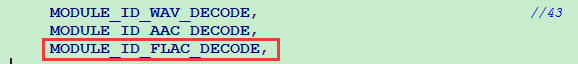
**AudioHWInit**接口中添加flac的启动B核初始化处理：

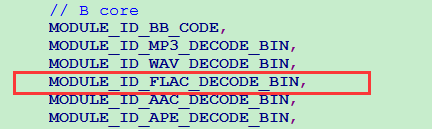


**AudioHWDeinit**接口中添加flac格式的关闭B核的反初始化处理：

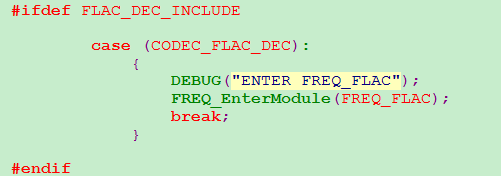


* + **ModulInfoTab.h**文件中添加flac格式的一般处理代码模块以及解码模块的ID：

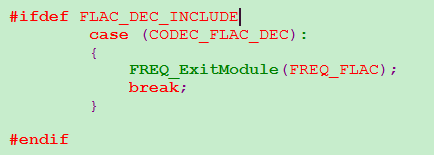




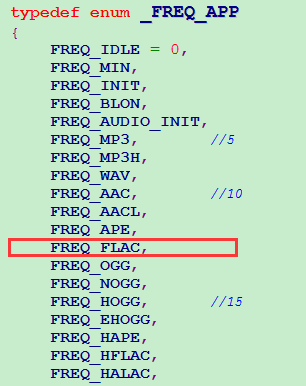
**AudioFREQInit**接口中添加Flac格式的解码A、B核频率设置：



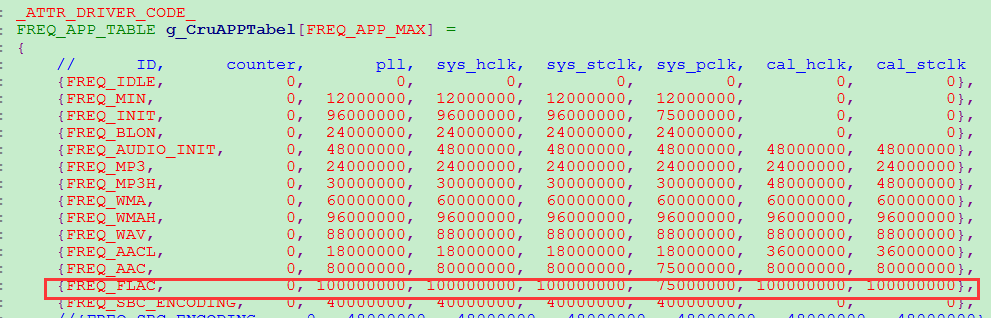
**AudioFREQDeInit**接口中添加flac格式的模块卸载A、B核的频率：



* + 这其中需要添加FREQ\_FLAC的定义，在**PowerManager.h**中添加

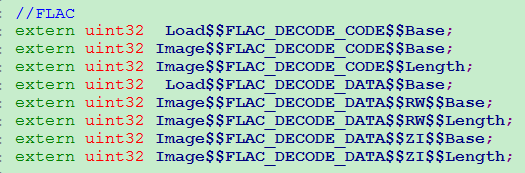


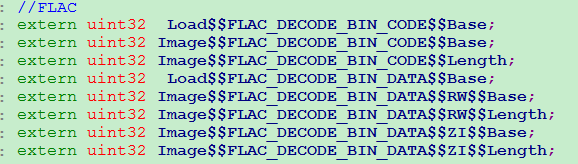
同时在PowerManager.c中添加Flac对应的频率配置表：



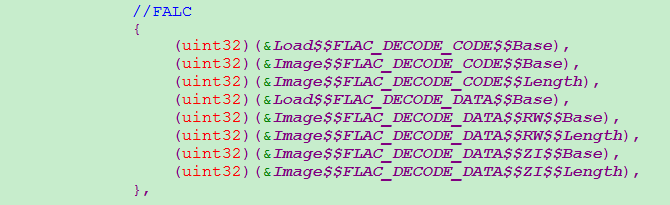
* + ScatterLoader.c中添加flac代码段与flac解码库的代码段：

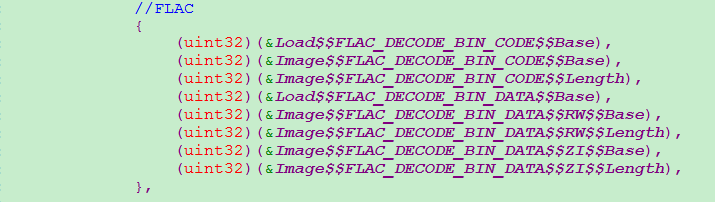
加载段声明：



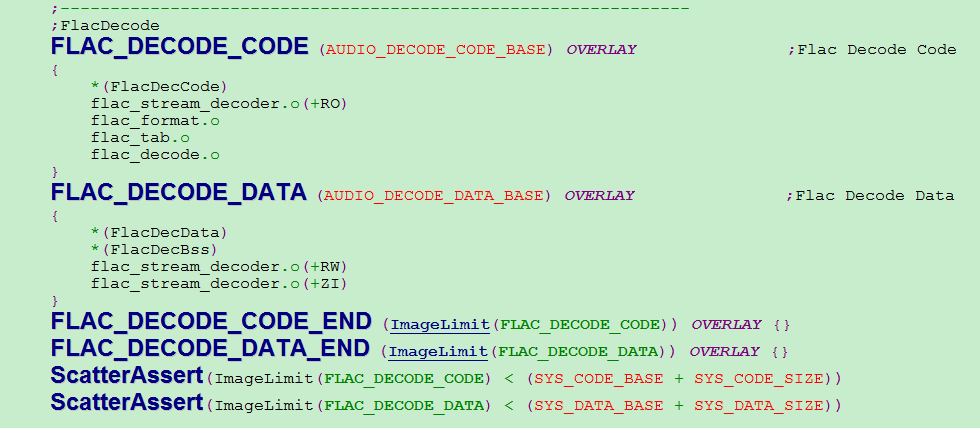


加载段内容：

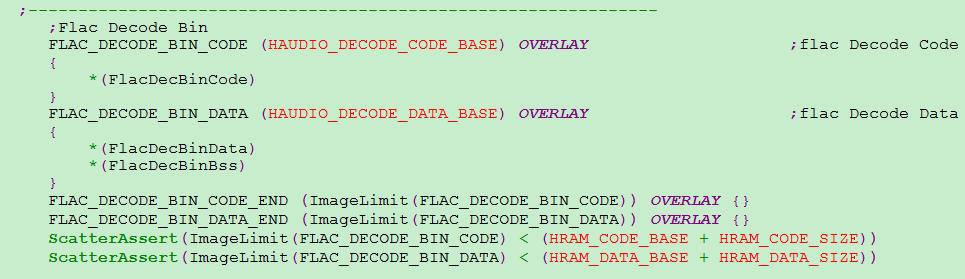




* + BuildAll.scr脚本文件中添加flac代码段的定义：



添加Flac解码库代码段的定义：

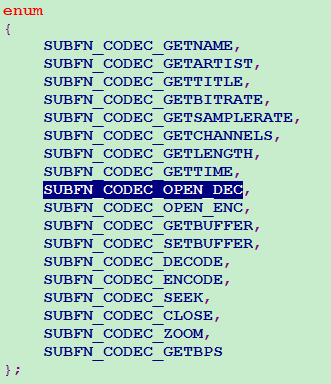


至此，添加常规音频格式的工作就已经完成了。下文将介绍用户自己添加非常规格式音频解码的步骤。

## 添加用户支持的音频解码

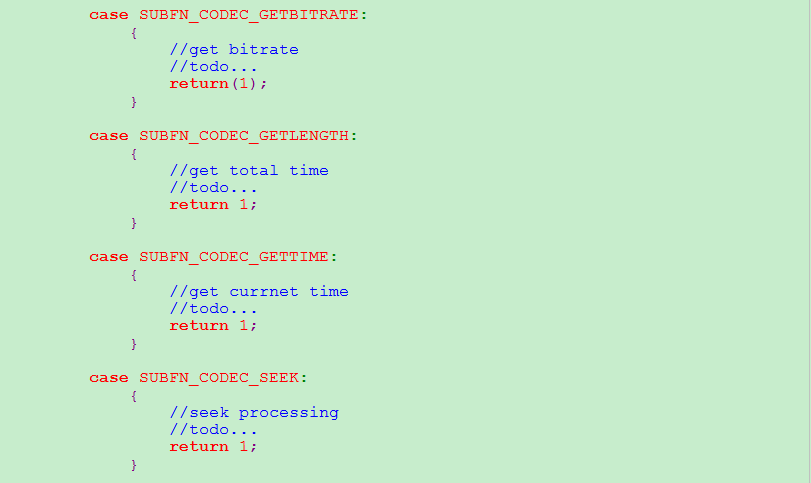
因为产品的差异化需要，在开发中，客户有可能要添加自己支持的非常规的音频格式解码，以便使自己的产品在满足不同消费群体与消费对象中展现独特的一面。下面将介绍一下这方面的实现步骤。

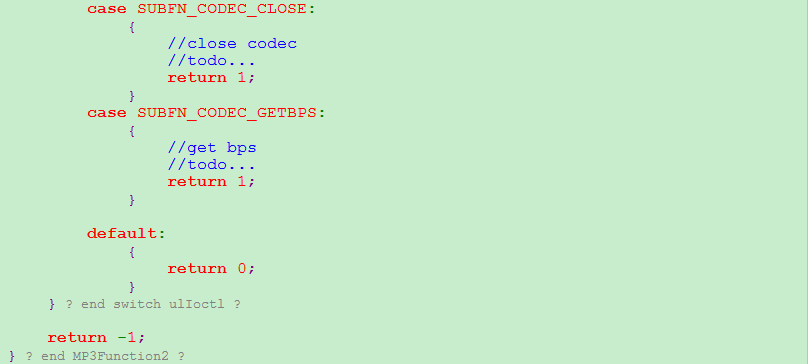
同时，用户添加支持的非常规的音频格式的解码时，需严格按照本RKNanoD平台解码中解码流程中的接口要求，提供对应的API，负责无法与RKNanoD平台的解码流程相兼容。具体以mp3为例，下图是RKNanoD解码器接口定义：



此为完整的解码过程中需要提供的API功能，实际中可能不需要这么多，比如mp3解码的接口：



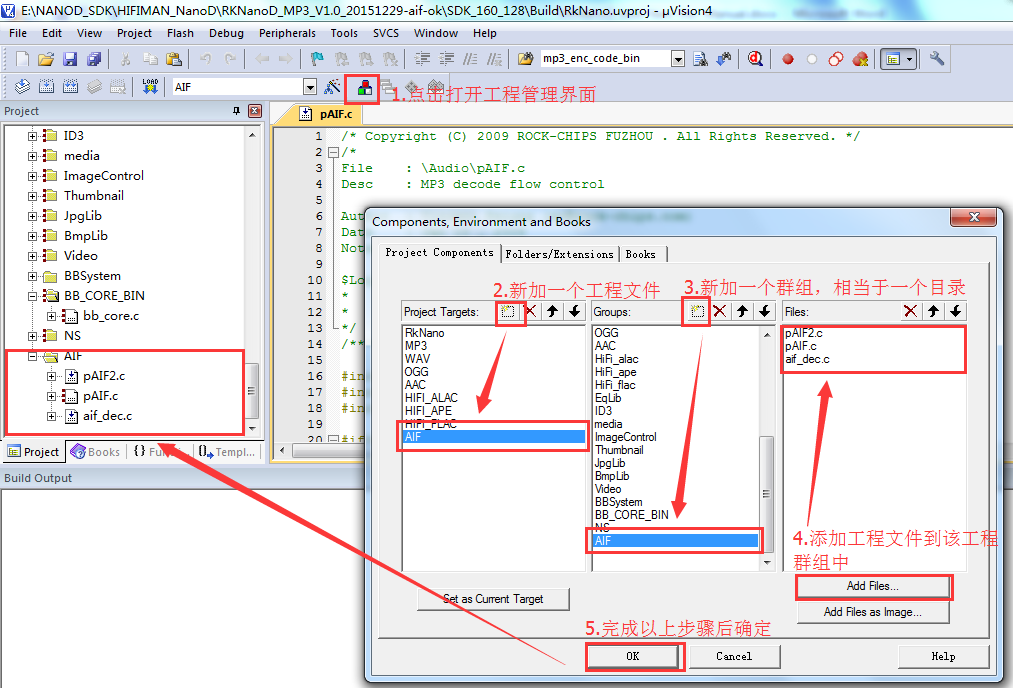




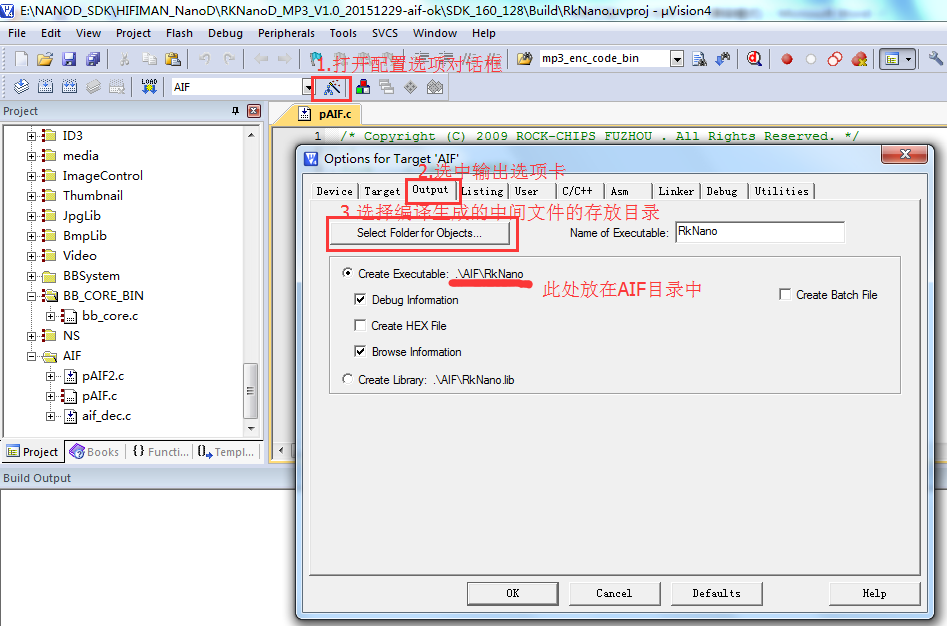
下文以添加AIF格式解码库工程为例，说明如何添加一个非常规的音频解码库工程，生成解码库。当然大部分工作同上一节介绍的，这里不再赘述，只陈述如何添加工程，如何配置工程，如何生成解码库等工作。

* + 创建B核编译工程：

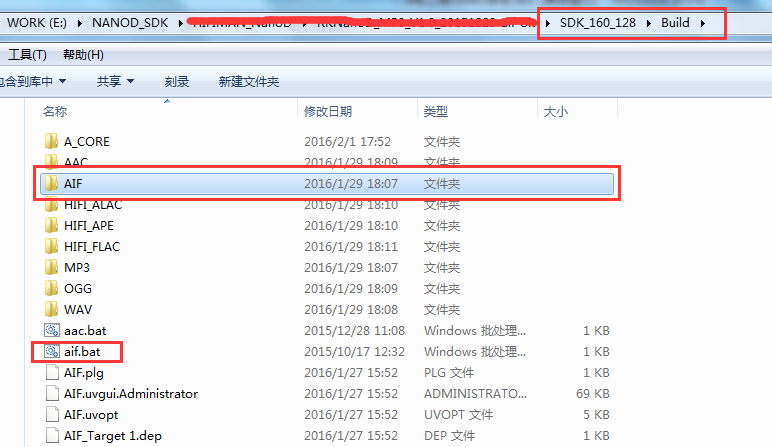
打开工程文件后，按如下图示操作，添加aif的B核编译工程。



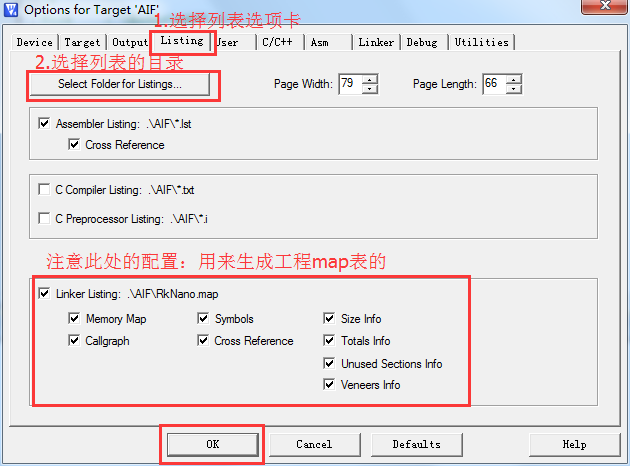
* + 配置AIF解码工程：



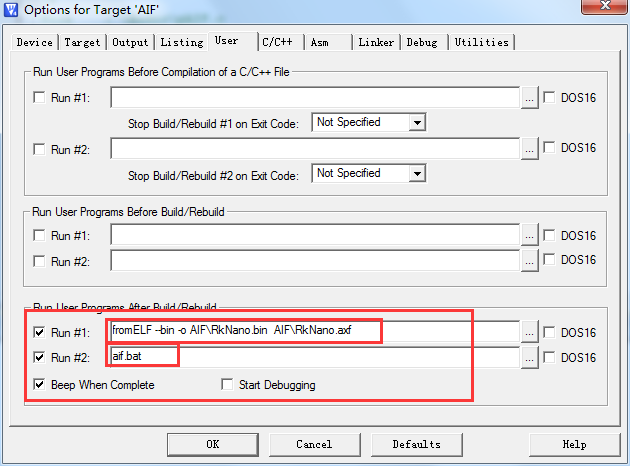
这里需要到\SDK\_160\_128\Build目录下，新建一个输出接收目录，这里我们新建一个AIF的目录，该目录用来存放编译AIF解码库工程生成的中间文件。如下图



配置工程列表的配置：这里选择列表的目录时，就选择上面我们新创建的AIF的目录。



配置用户自定义命令，用以B核编译完成生成可执行的bin文件：



RUN#1中填写:**fromELF --bin -o AIF\RkNano.bin AIF\RkNano.axf**

RUN#2 中填写**：aif.bat**

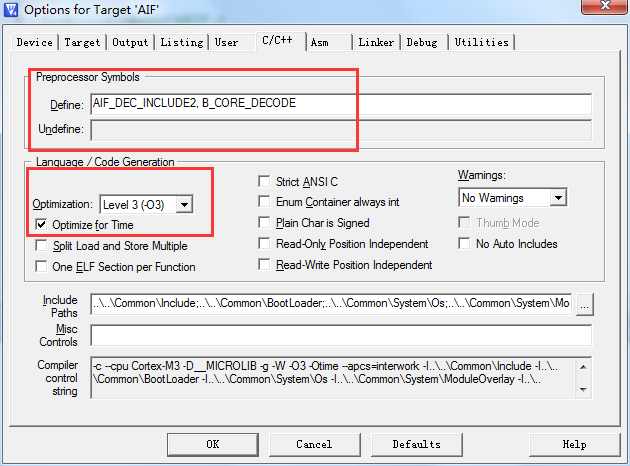
打开\SDK\_160\_128\Build目录，创建aif.bat，内容如下（注意路径的正确性）：



该命令执行完成之后，会将Build\AIF目录下编译的转换为可执行bin文件，并拷贝到音频解码器指定目录common\codec\audio\aif\目录下。

配置编译环境如下图所示：

这里需要预定一个编译宏 ：AIF\_DEC\_INCLUDE2，B\_CORE\_DECODE



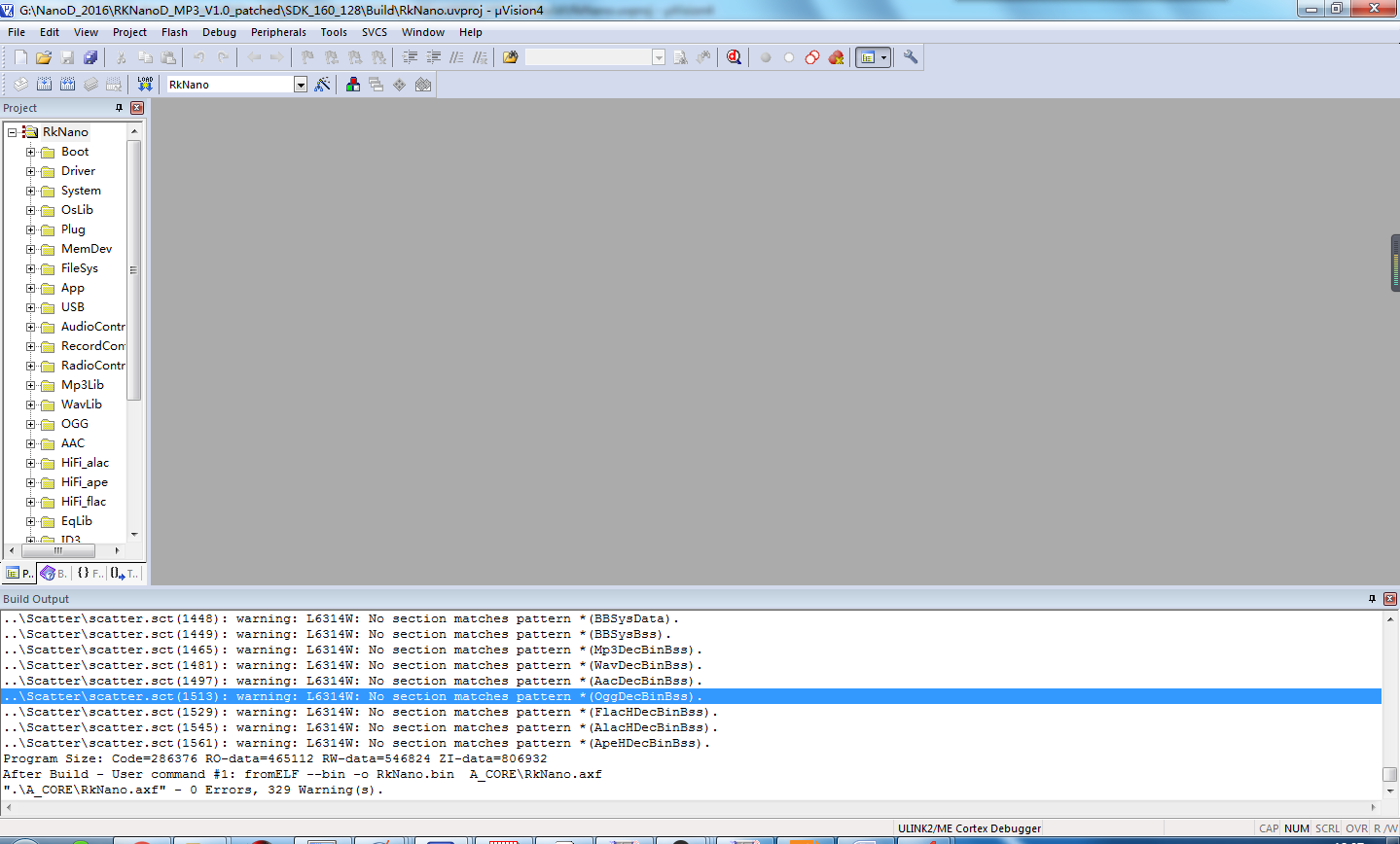
新建一个解码库工程的软件配置工作就到此结束了，代码中添加的工作， 已经在上节中介绍过了，这里不再详述。

B核解码器工程创建完成之后，优先编译B核解码器工程，生成的二进制文件会更新到A核指定目录，此时由于该二进制代码变动，要求A核再重新编译才能使用。

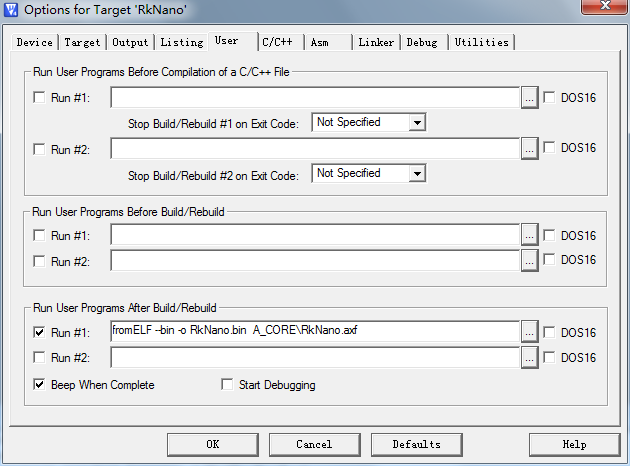
# 工程编译

RKNanoD SDK 使用 ARM的KeilμVision 4 （最好4.72.10版本）编译开发。工程文件路径：SDK\_160\_128/Build/RkNano.uvproj。

打开工程编译软件，工程界面如下图：



点击“build target”或者“rebuild All”按钮，编译整个工程，下面的输出窗口将会显示编译结果以及链接的结果，“scatter.sct”的警告信息对编译的结果没有影响，如果编译链接均成功，将会在A\_Core文件目录下生产RkNano.axf 和RkNano.bin。工程配置信息如下：



需要注意的是，首先需要全部编译B核工程，生成相应的运行于B和的bin文件之后，再编译A核工程；如果B核代码没有修改，可以直接编译A核工程。

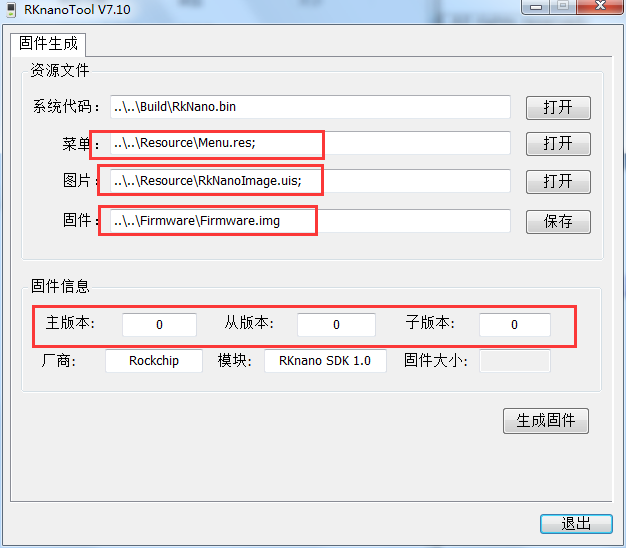
# 生成固件与烧写固件

RKNanoD MP3 SDK中提供了固件的生成与烧写工具。软件工程编译链接后，会在Build目录中生成了RkNano.bin文件，该文件是可运行在NanoD芯片上的二进制可执行文件，固件则包括了可执行的二进制文件以及图片资源和字符串菜单资源。

## EMMC Flash固件生成工具

“SDK\_160\_128\Development\firmware\_generate\_eMMC”目录中RKnanoTool.exe 工具是用来生成EMMC flash下的固件工具。如下图：为EMMC Flash下的固件生成工具界面。

注意：固件信息中主版本号、从版本号均可以设置为2位有效数字。子版本号可以设置为4位有效数字。

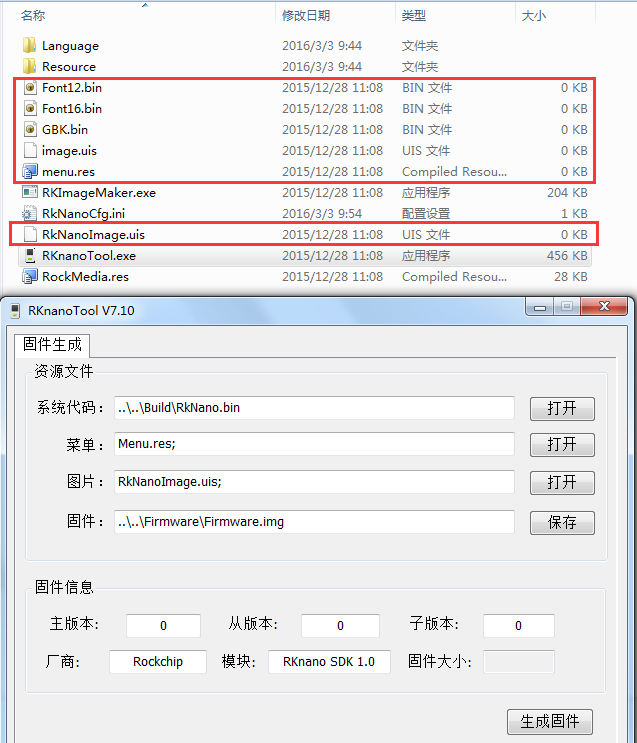


## SPI NOR Flash 固件生成工具

“SDK\_160\_128\Development\firmware\_generate\_SPI”目录中RKnanoTool.exe 工具是用来生成SPI Nor flash下的固件工具。如下图：为SPI Nor Flash下的固件生成工具界面。

**注意**：

* + - 1. 其中固件信息中主版本号、从版本号均可以设置为2位有效数字。子版本号可以设置为4位有效数字。
      2. 这里因为SPI Nor Flash的存储容量有限，固件与图片资源和菜单资源无法一起打包生成，但是生成工具又必须要这些文件，所以这里创建文件大小为0的空文件，以便能成功生成固件。

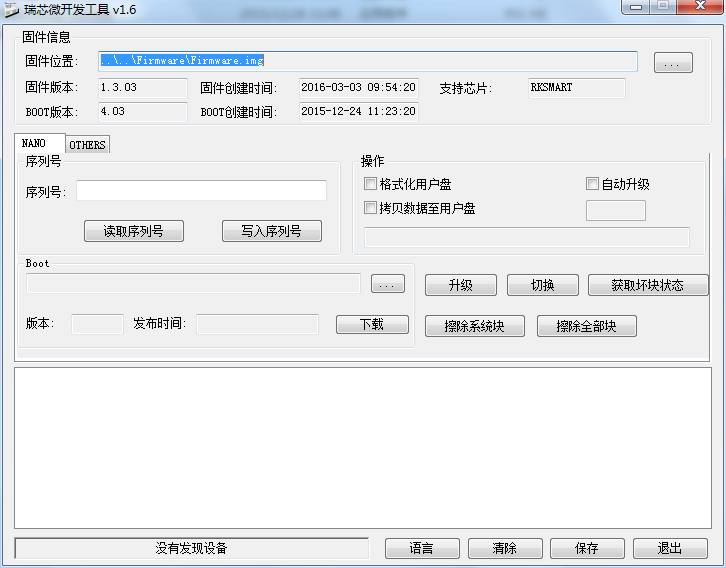


## 烧写固件

固件烧写工具根据使用用途的不同，分为两种。用户开发调试阶段的烧写工具，使用单机版的烧写工具，这种烧写工具只能一次识别、烧写一个设备。用户在开发完成后，在工厂批量生产时，需要使用工厂量产版的烧写工具，这种烧写工具可以一次识别、烧写8个设备。下面将分别做一下介绍。

### 单机烧写工具

下图为RKNanoD的单机固件烧写工具。工具存放在“SDK\_160\_128\Development\firmware\_upgrade” 中，RKDevelopTool.exe打开界面如下：



**固件位置：**用来选择将要烧写到设备中的固件。用户可指定路径来选择不同的固件烧写。默认固件路径存放在“..\..\Firmware\Firmware.img”

**固件版本：**生成工具界面中用户自定义的版本号，会在这里显示。

**固件创建时间：**显示最后一次固件生成的时间。

**Boot版本：**显示最新的boot版本号。

**Boot创建日期：**显示Boot最后创建的日期。

**序列号：**目前保留，NanoD中未使用到。

**切换：**用于切换升级模式，在电脑连接PC识别到MSC设备时，通过切换操作可以进入MaskRom升级模式。

**升级：**下载更新固件。

**擦除系统块：**擦除固件部分的存储块区域。

**擦除全部块：**擦除所有的存储块区域。

### 工厂量产烧写工具

在目录SDK\_160\_128\Development中有一个目录FactoryTool\_v1.41，该目录存放的是量产版的烧写工具。

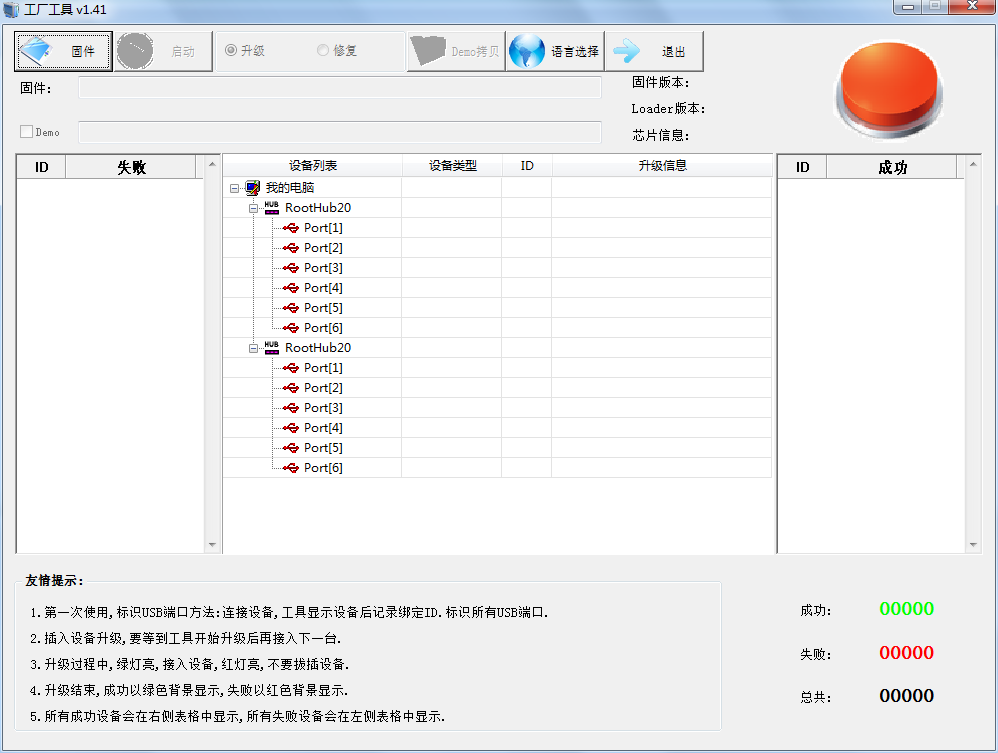
工具界面中重要功能按钮介绍如下：

**固件**：用于指定加载需要烧写的固件。

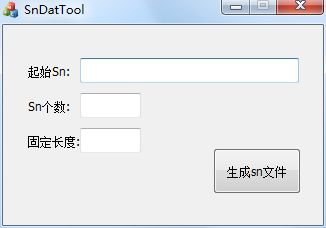
**启动**：当识别到设备插入，会在设备列表中显示出来，等所有设备都识别成功后，选择启动去烧写固件。

**注意**：

* 量产版本的烧写工具需要根据config.ini来配置。详细的配置见该文件各个配置选项的说明。
* 如果用户的产品需要支持SN（序列号）功能，需要根据用户自定义的SN定义规则生成SN工具。Rockchip在SDK\_160\_128\Development\SnDatTool目录中提供了SN数据生成工具的一个Demo VC工程以及说明文档，用户可根据自身需求做出修改，编译改VC工程，生成SnDatTool.exe工具，使用该工具生成sn.dat数据文件，将该文件放在FactoryTool\_v1.41目录下，配合量产工具使用即可。



使用SN数据生成工具额界面如下：



# 资源打包方法

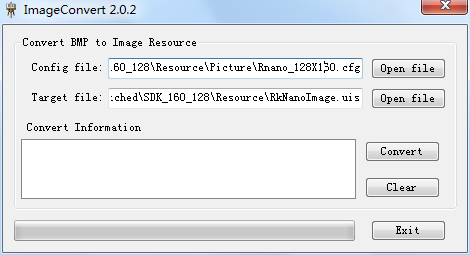
NanoD SDK支持用户界面操作，这其中会显示图片与字符串。图片与字符串会作为固件的一部分打包进固件，保存在flash中。下面将分别介绍图片资源的打包与字符串资源的打包。

## 图片资源打包

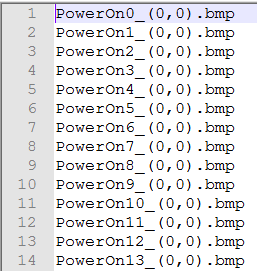
图片资源存放在路径“..\SDK\_160\_128\Resource\Picture”目录下。所有的图片均为24bit位深像素点的bmp格式。注意图片命名格式，根据应用的不同，图片名称中需用括号标注图片的左上角起始坐标。

具体如：MasterName\_SlaveName\_(x, y).bmp。其中MasterName与SlaveName用来生成图片的ID，会在程序中使用。x,y为图片在LCD中起始左边。

图片资源打包工具使用ImageConvert.exe,打开工具后界面如下图所示：



**Config file** 选择Picture目录下的Rnano\_xxx.cfg 配置文件,这里根据LCD显示屏的大小选择正确的配置文件。配置文件是组织需要打包的所有图片的列表，如图为打开Rnano\_128x160.cfg的一部分的列表。



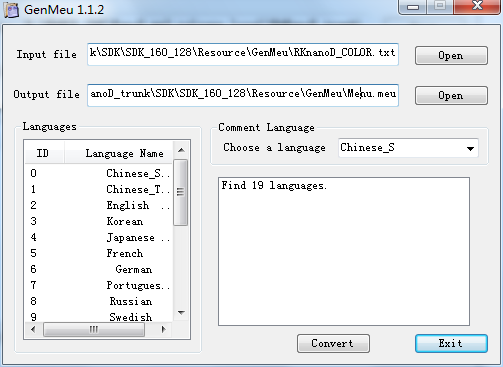
**Target file**为需要生成的图片资源输出文件，名字可以有用户指定，后缀名为\*.uis。

**Convert：**开始转换的功能按钮。

图片资源转换工具从配置文件“Rnano\_xxx.cfg”中依次获取图片名，当所有的图片完成打包后会生成文件“xxx.uis”这里默认使用“RKnanoImg.uis”和“ImageResourceID.h”。“ImageResourceID.h”是用来保存图片的ID编号及ID名称。方便软件代码中使用ID编号来显示对于的图片。“RKnanoImg.uis”于固件一起打包存放在外部存储设备中。

## 字符串资源打包

字符串资源打包工具GenMeu.exe界面如下图所示：



其中，

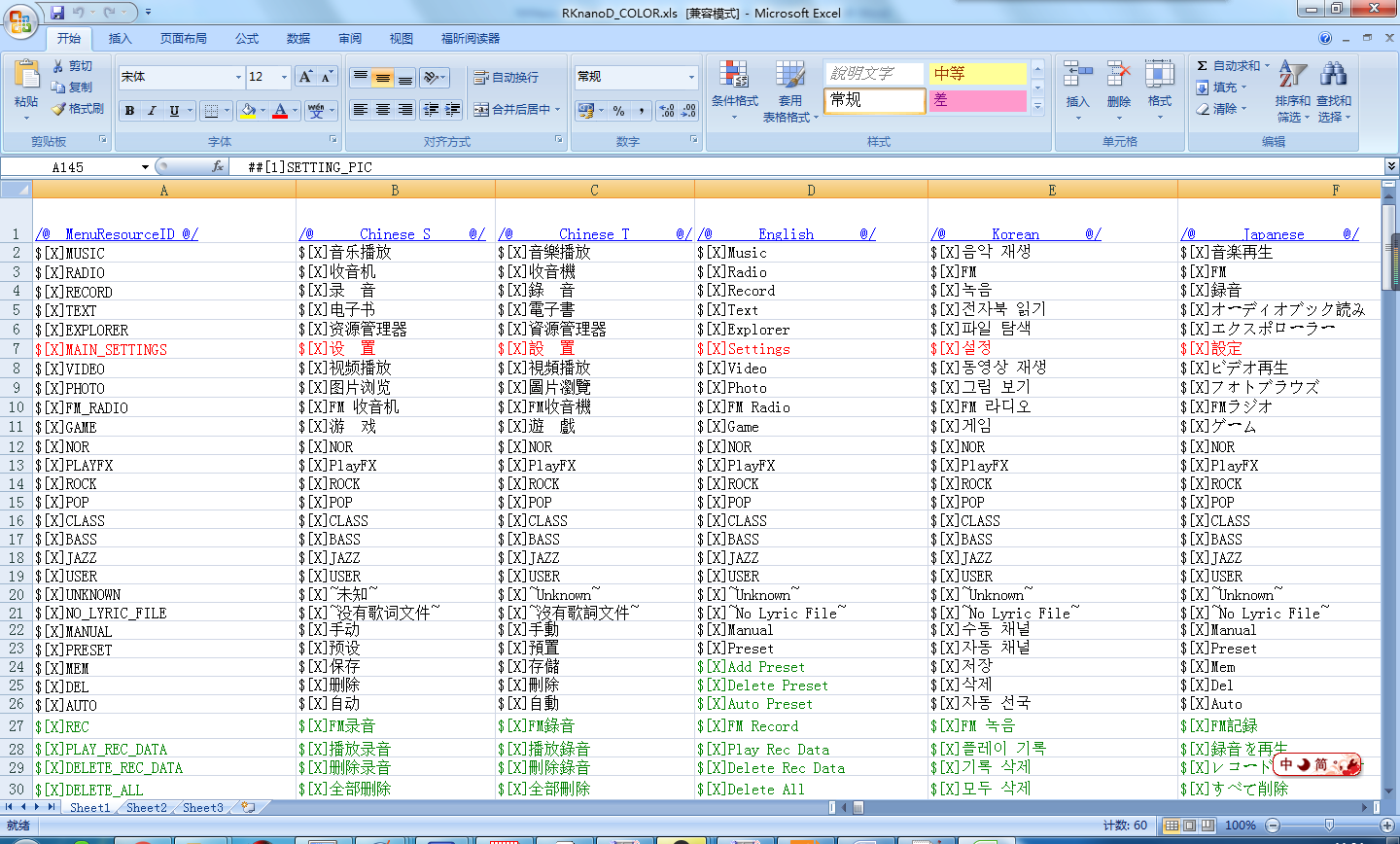
**Input file：**需要转换的Unicode文本文件。

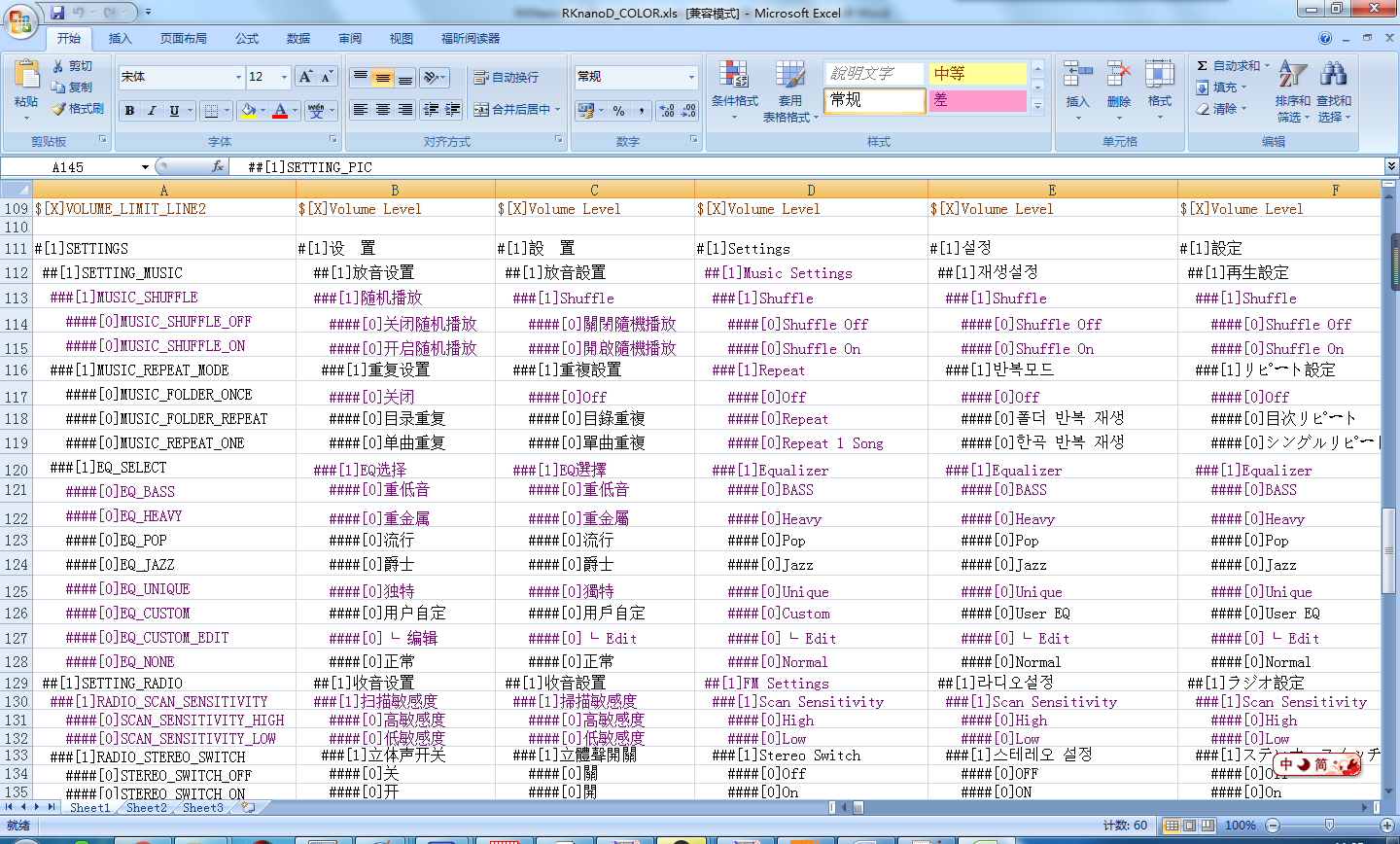
**Output file：**打包好的字符串数据文件。

**Language：**显示语言名与ID，语言名与ID来自输入的文本文件。

**Comment Language：**输出文件“MenuResourceID.h”中显示注释的语言

字符串是按照特定格式组织的excel文件。如下：





第一列字符串定义字符串的ID名称，其他列为多国语言的字符串。

第一行字符串用“/@” 和“@/”是语言名称。

以“$[X]”开始的字符串是用来在LCD上显示的字符串信息。使用“##[1]” 或“###[0]”是设置菜单字符串。

“#”的个数提示了设置菜单的级数关系，比如：

“#”： 一级菜单

“##”： 二级菜单

“###”： 三级菜单

“[ ]”中的数字表示了该菜单项下是否有子菜单：

1：有子菜单

0：没有子菜单

在使用GenMenu.exe工具之前，必须将excel文件转换为Unicode文件文件，方能使用。选择excel开始菜单，选择“另存为…”保存类型选择Unicode 文本选项，保存即可。如图：

