

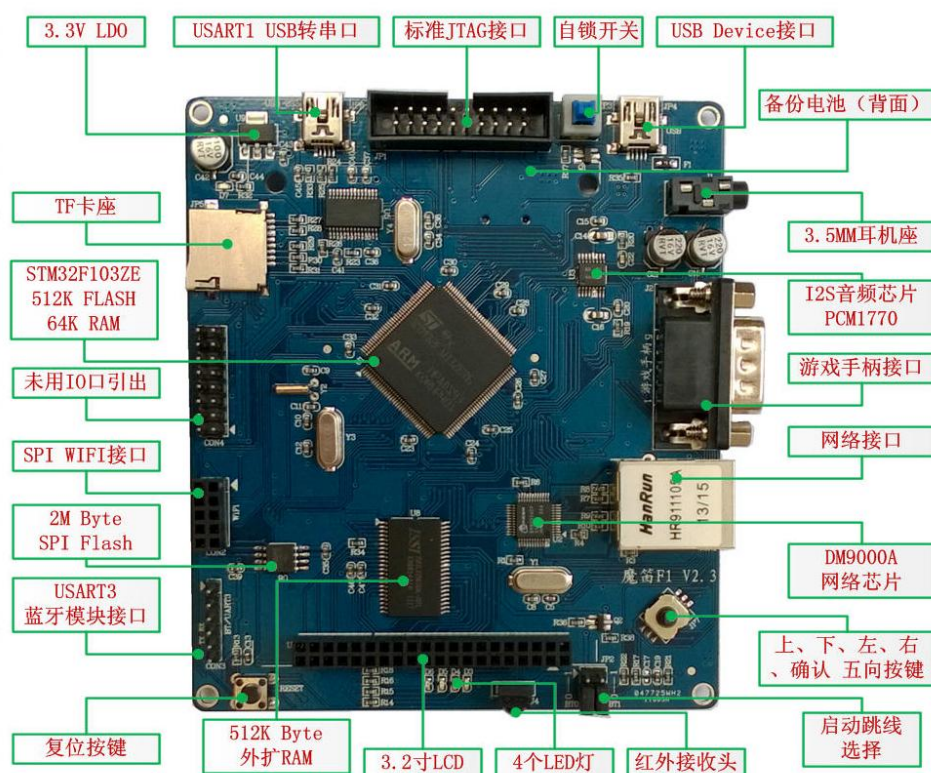
## 魔笛 F1 开发板使用说明

### 一、硬件资源介绍

魔笛 F1 开发板是在原 stm32 radio 项目中提炼、改良后的一款用于 RT-Thread 入门学习、提高的综合性开发板。

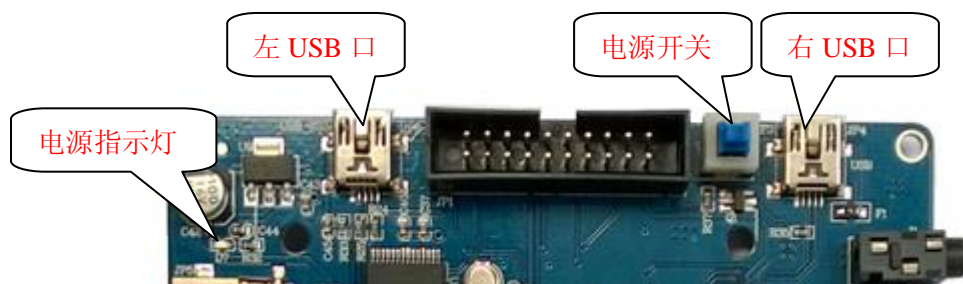
作为一款专门针对 RT-Thread 学习的开发板，魔笛 F1 并不像其他学习板一样拥有众多“华丽”的外扩设备及“夸张”的外观，我们只需具备去玩转 RT-thread 所需的基本外设足以。

魔笛 F1 硬件资源如下：



### 供电方式说明

魔笛 F1 开发板上面共有两个 USB 口，我们可以使用这两个 USB 口中任意一个口进行供电，接上 USB 线后，按下板子上的自锁开关板子即上电，上电后，板子左上角的电源指示灯会亮起



因为左边 USB 接口还兼 USB 转串口功能（USART1），所以我们建议用左边 USB 口来供电，同时开启串口终端来观察程序的运行信息。

## 启动方式说明

开发板上主芯片（STM32F103ZET6）的启动方式使用板子上右下角两个跳线帽来选择：



启动模式选择管脚		启动模式
BT1	BT0	
X	0	用户 FLASH
0	1	系统 FLASH（串口 ISP）
1	1	内嵌 SRAM

我们默认选择 用户 FLASH 启动。

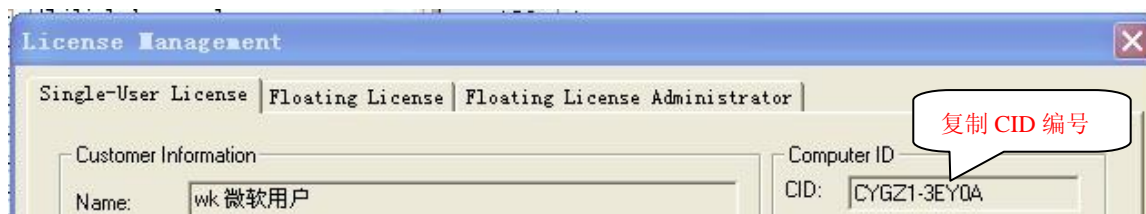
## 二、 开发环境准备

### 安装并破解 KEIL MDK

我们的开发环境默认使用 KEIL MDK，附赠资源包中 TOOL 文件夹下有 MDK 的下载地址，最好下载最新版的 MDK 使用，因为发现有些人使用低版本的 MDK 出现一些莫名其妙的现象，笔者使用的 MDK 版本是 MDK4.54。

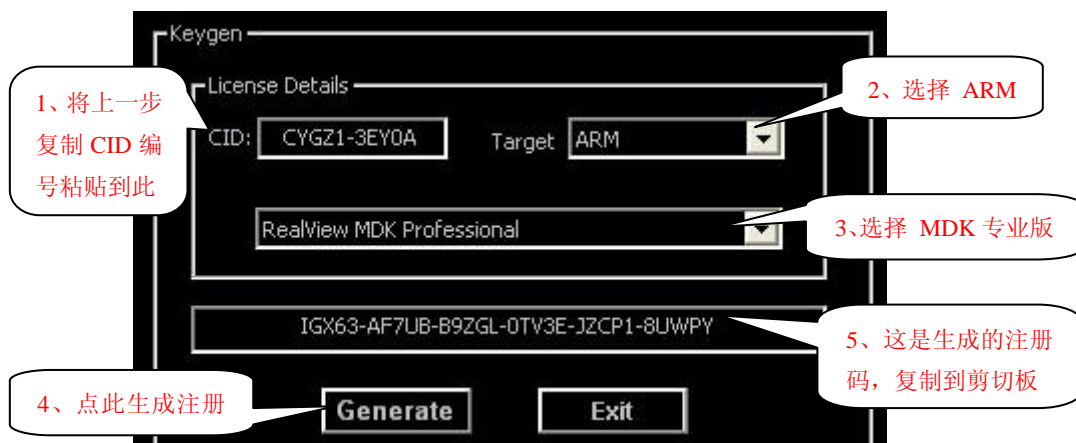
#### 1、安装 keil MDK

使用自行下载的 MDK 最新版，或使用我提供的 MDK4.54 版，装完后打开 MDK 软件（安装时的路径最好不要包含中文路径名和空格），点击 file->license management，复制如下图所示的 CID 编号到剪切板，后续破解时用到。

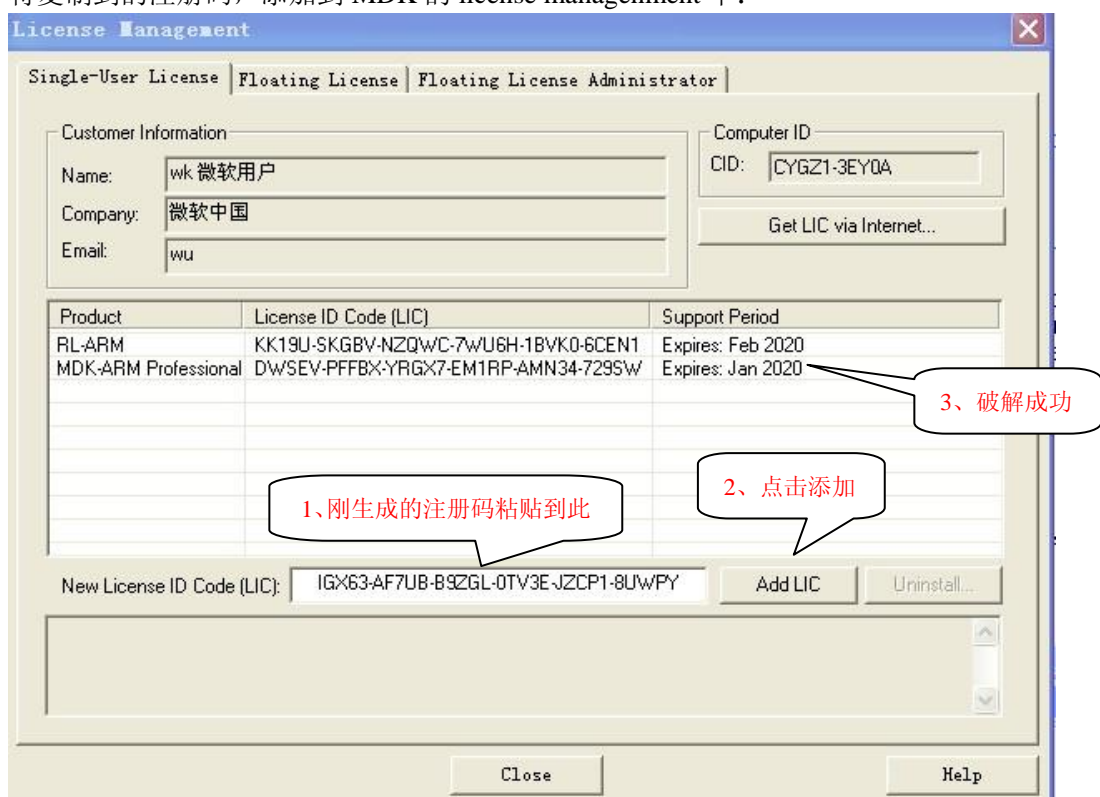


#### 2、破解 MDK

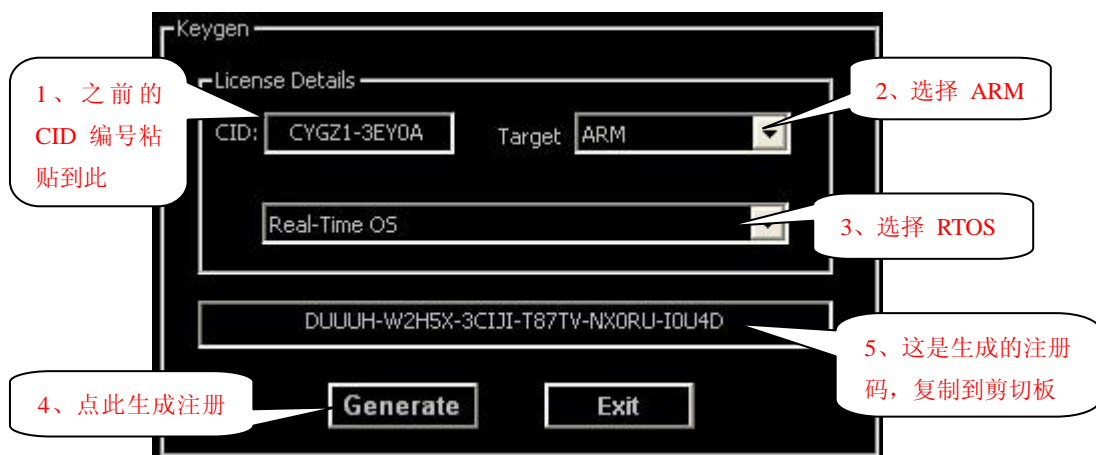
找到附赠资源包《工具软件/keil 相关工具》文件夹下 MDK\_注册码生成器，双击打开，按下图选择要破解的目标，并生成注册码



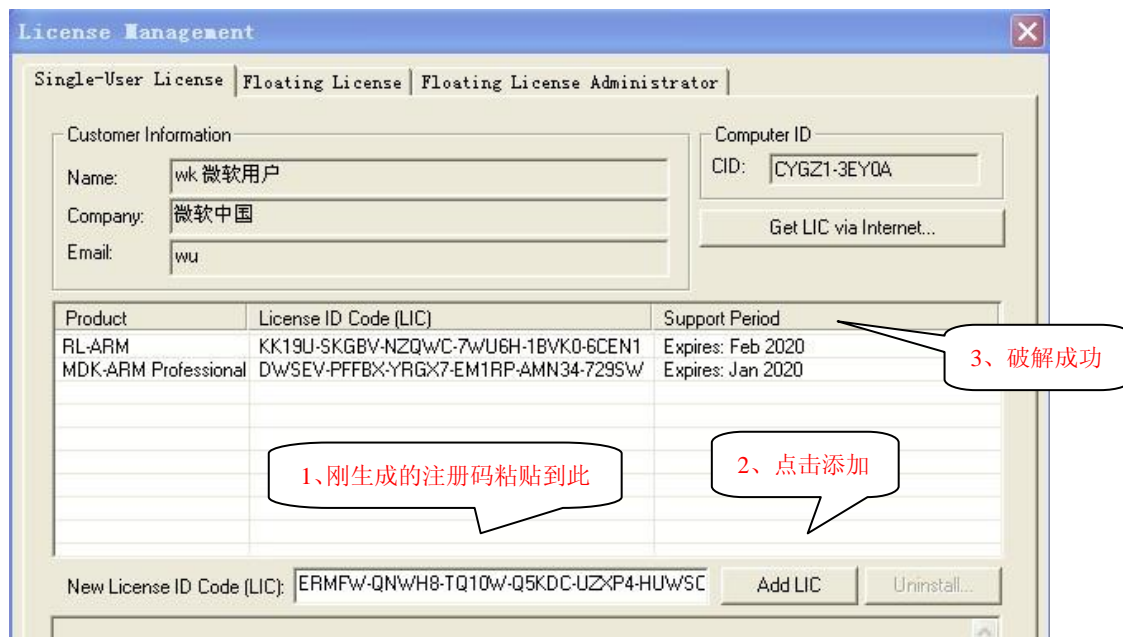
将复制到的注册码，添加到 MDK 的 license management 中：



由于我们要用 os，所以还要再破解另一项：



再次将复制到的注册码，添加到 MDK 的 license management 中：



## 安装串口驱动

考虑到调试的方便，魔笛 F1 开发板板载 USB 转串口芯片 PL2303\CH340G(V2.4 以后版本用 CH340G)，为了能够正常使用 USB 转串口功能，需要安装附赠资源包中《工具软件\串口工具》文件夹中 PL2303\CH340G 驱动。

装好驱动后，重启 PC，用 USB 线连接开发板左边 USB 口到 PC，这时候，在 PC 的设备管理器中会出现一个 COM 口，此 COM 口就是板上 USB 转串口芯片在 PC 上枚举出来的，我们后续就用这个 COM 口来和 PC 进行串口交互。

笔者电脑端在连接后的 COM 口如下：



## 串口终端设置说明

我们在程序调试过程中会经常用到串口来打印程序运行信息，RT-Thread 下的命令行组件也会用到串口终端来运行命令，所以在调试过程中连接一个串口终端软件是很必要的。笔者建议使用 **SecureCRT** 这个串口终端软件，此软件我们在《工具软件\串口工具》

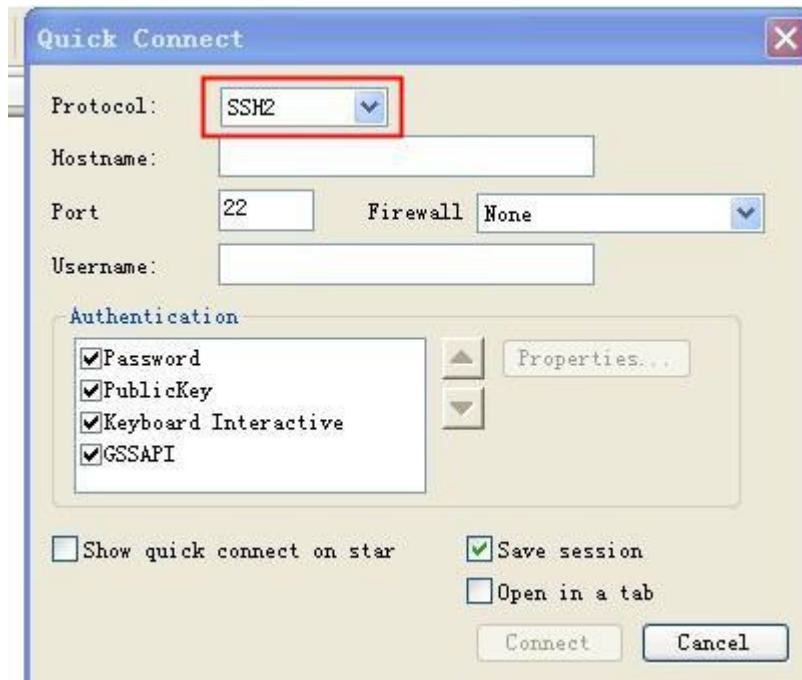


目录中有提供。

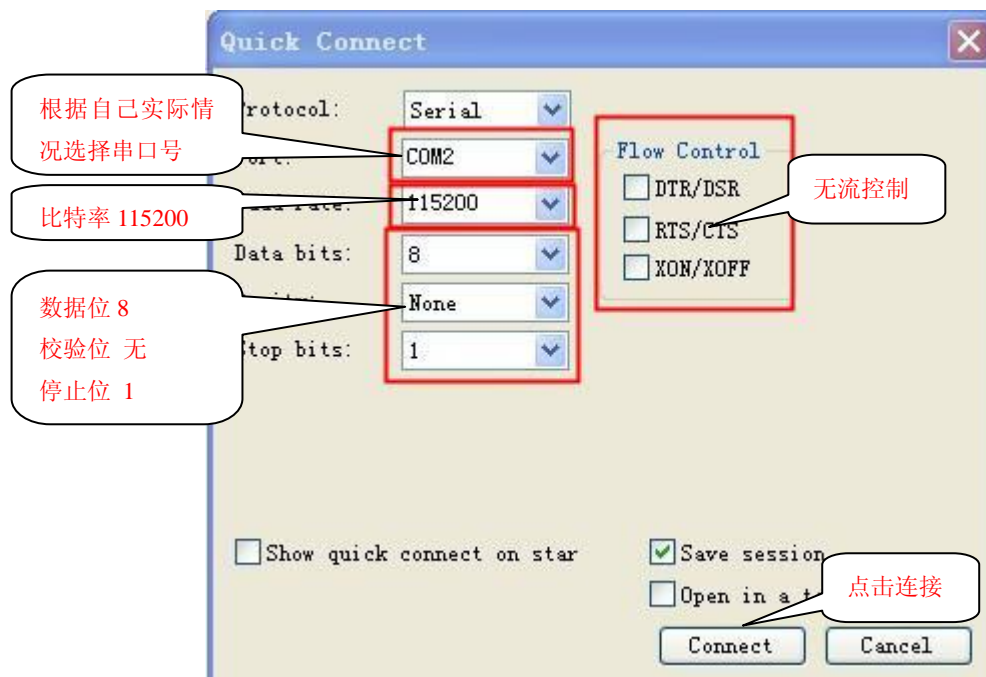
双击 SecureCRT.exe，点击如下图红色框中的 quick connect



弹出如下界面：



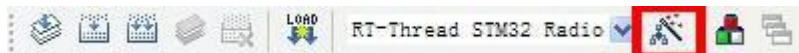
Protocol 中选择 serial 串口，然后按照如下配置设置，并点击连接



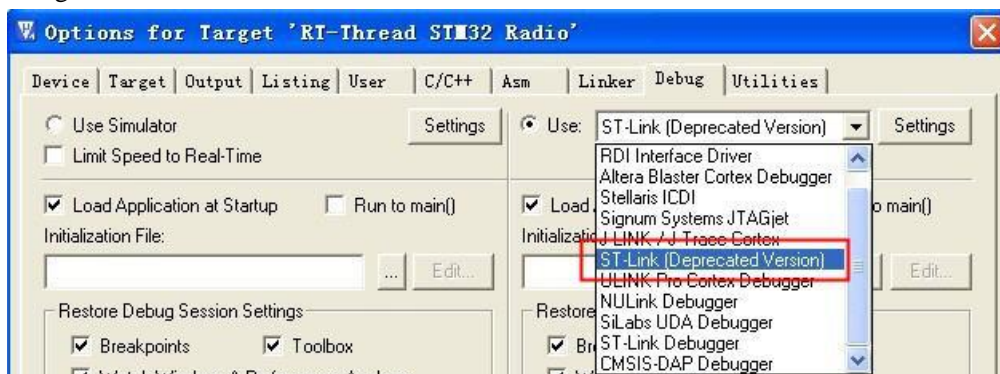
## 程序下载、调试方法

### 1、使用 STLink

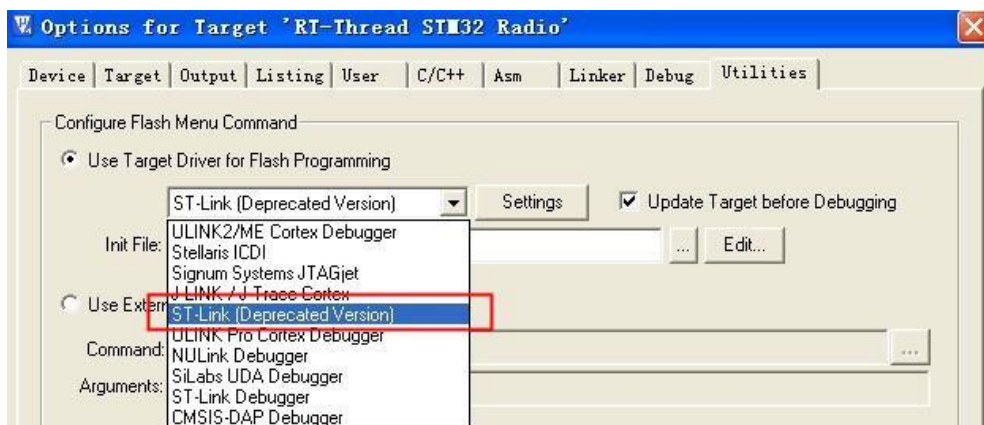
启动跳线选择为 用户 flash，通过 stlink 连接开发板与 PC，并给开发板供电。  
打开某个例程工程，点击如下的图标，进行开发工具选择



在 Debug 标签下，选择 ST-LINK



同时也在 Utilities 标签下，选择 ST-LINK

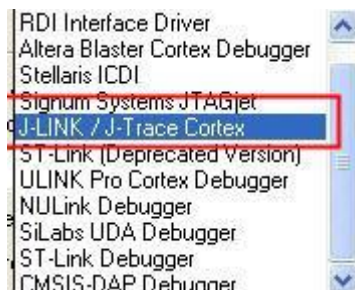


完成后，选择确定，编译好程序后，点击如下图中 debug 按钮进行下载、调试。



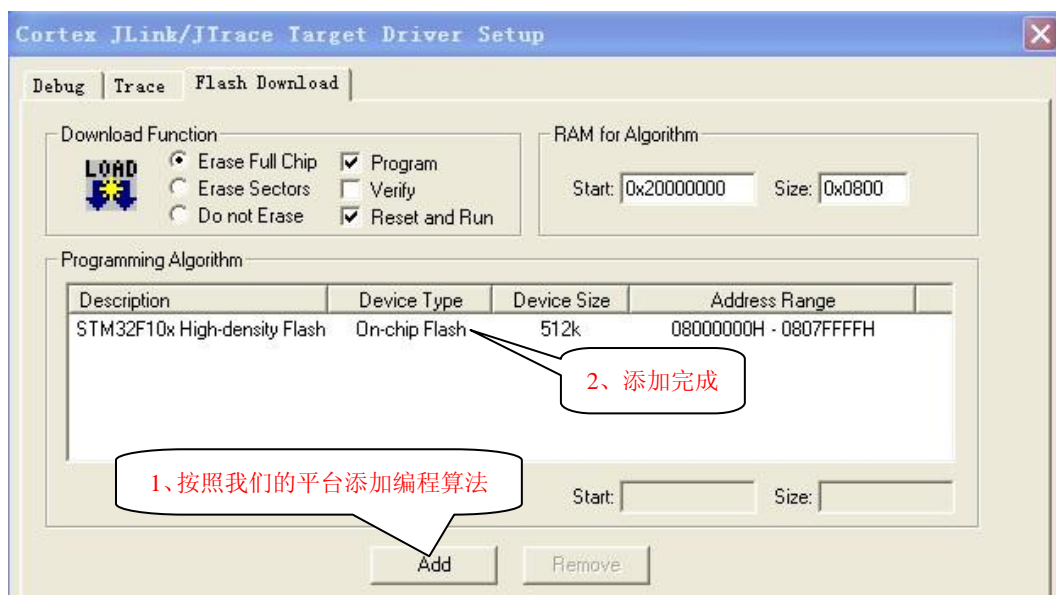
### 2、使用 JLink

和 stlink 的设置步骤类似，在 OPTION 中 debug 和 Utilities 标签下分别选择 jlink 作为下载工具：



和 stlink 不同的是，使用 jlink 时，要选择 flash 编程算法。这个是在 debug 标签下的 flash

download 标签下设置:



完成后, 选择确定, 编译好程序后, 点击如下图中 load 按钮进行下载



调试时点 debug 按钮进行仿真:



### 3、使用串口 ISP 方式

如果用户没有 stlink 或 jlink, 那么请看《相关学习资料/STM32\_FLASH 的 3 种烧写方式.pdf》中的串口 ISP 下载方法介绍, 并结合目录

《工具软件/串口工具/Flash\_Loader\_Demonstrator\_v2.5.0\_Setup.exe》来进行串口 ISP 下载。

#### 例程中 RT-Thread 版本介绍

目前例程中都用的 RT-Thread 最新发布版即: 1.2.0 正式版, 实际上我们在学习过程中具体用哪一个版本都无所谓, 因为里面的东西都一样, 只是一些功能上的升级和 bug 修正。

特殊说明: 由于网络收音机例程存在时间太久, 已无法升级到新版本, 现在网络收音机例程用的内核版本是 1.1.0 版。

## 三、 例程系统文件说明

魔笛 F1 开发板所用到的大多例程都需要一些系统文件, 比如和 UI (图形界面) 中用到的中文字库、文件系统中用到的 unicode 转换表、触摸校准后的配置文件、还有一些背景图片等等。这些系统文件在《魔笛 F1 开发板配套例程/需要拷贝至 SDCARD 的文件》目录中。

这些文件发货前已经 copy 到了板上自带的 tf 卡中, 如果用户不小心格式化了, 可以将这些文件再 copy 进去。

## 四、 配套例程体验

## 音频综合例程

### 1、网络收音机例程

网络收音机例程是移植自官方最新版收音机程序，笔者对其相关功能进行了一些优化、修改。这个例程向大家展示 RT-Thread 内核、命令行组件（finsh）、文件系统组件（elmfat）、网络组件（Lwip）、图形界面组件（RTgui）结合起来后的综合应用。这个例子的详细设计分析、需求等见配套资料中的文档《基于 ARM 的网络收音机设计.pdf》。

例子主要具备如下功能：

- ✓ 主控芯片通过 DM9000AEP 把互联网上的网络音频流（支持 shoutcast 协议流媒体）抓取下来，先在板子上外扩的 SRAM 上缓存。然后在 STM32F103ZE 上通过 mp3 软件解码器（helix 解码库）解码变成 PCM 音频，通过 DMA 方式送到 PCM1770 进行回放；
- ✓ 支持将近 30 个豆瓣电台点播和其他电台点播；
- ✓ 可以流畅播放 SD 卡上的 mp3、wav 格式文件。（支持大于 4G 的 SD 卡）；
- ✓ 通过 3.2 寸 240\*320 的 TFT 触摸屏、五向按键、红外遥控器进行控制；
- ✓ 红外遥控器具备自学习功能；
- ✓ 系统信息显示：网络信息、内存使用信息；
- ✓ LCD 背光、音量可调，触摸灵敏度可随时校准；
- ✓ 读卡器功能，可通过 USB Device 接口将板上的 TF 在 PC 端枚举出一个盘符，让用户通过 PC 向板上 TF 卡拷贝资料。

上电后系统进入初始界面：

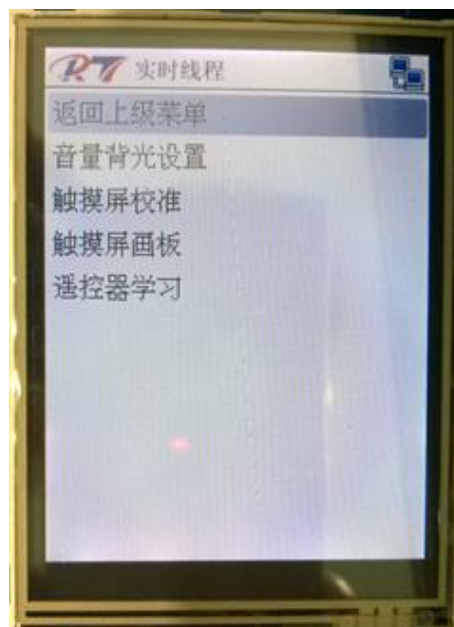


这时候可以按**五向按键**（右下角的 JP7）的左、右键进入系统功能选择菜单，用选择键（五向按键垂直向下按即为确认键）选择相应的功能：





- <豆瓣电台>: 播放豆瓣电台，播放时左、右键可以调节音量
- <其他电台>: 播放其他 shoutcast 音频流电台
- <更新电台>: 更新 其他电台 的播放列表，执行此操作后 TF 卡中会产生 radio.pls 文件，用户可以在 radio.pls 文件中加入其他的电台地址
- <播放文件>: 播放 TF 卡中的 wav 或 mp3 文件，播放时左、右键可以调节音量
- <浏览图片>: 浏览 TF 卡中 picture 文件夹中的 HDC 格式文件，配套资料的《工具软件\HDC 图片转换》中的工具用来转换 HDC 格式图片
- <设备信息>: 显示系统信息
- <系统设置>: 背光、音量调节、触摸校准、遥控器学习



- <USB 联机>: 通过 USB Device 接口将 TF 卡在 PC 端枚举出盘符，让开发板充当一个“读卡器”

例子中用到的中文字库、背景图片等存放在板子自带的 TF 卡中。

默认状态下收音机可以获取直接从路由器端自动获得 IP 地址, 当我们想使用固定 IP 时, 可以按照如下说明修改代码下 rtconfig.h 文件:

```
/* Using DHCP */
// #define RT_LWIP_DHCP //屏蔽之
/* ip address of target */
#define RT_LWIP_IPADDR0 192 //下面按照实际设置
#define RT_LWIP_IPADDR1 168
#define RT_LWIP_IPADDR2 1
#define RT_LWIP_IPADDR3 30

/* gateway address of target */
#define RT_LWIP_GWADDR0 192
#define RT_LWIP_GWADDR1 168
#define RT_LWIP_GWADDR2 1
#define RT_LWIP_GWADDR3 1
```

## 2、wav 音频播放例程

### wav 介绍

Wav 是 windos 的声音文件, 后缀为 wav(\*.wav), 有时也直接称为 PCM (脉冲编码调制) 声音文件, 是没有经过任何压缩的声波数据文件。Wav 文件中的音频数据直接对应着模拟声音被量化的数字量, 我们可以直接将其送入 DA 芯片进行播放, 而无需特殊处理。

### Wav 文件结构

Wav 格式文件由 4 部分组成:

RIFF Header (8 bytes)	RIFF Type(8 bytes)	Fmt chunk(24 bytes)	Data chunk(音频数据+8 bytes)
-----------------------	--------------------	---------------------	--------------------------

第一部分 RIFF Header 共 8 bytes, 前 4 bytes 内容永远为 RIFF, 后 4 bytes 为后续文件的总数据量;

第二部分 RIFF Tyres 共 4 bytes, 内容永远为 WAVE;

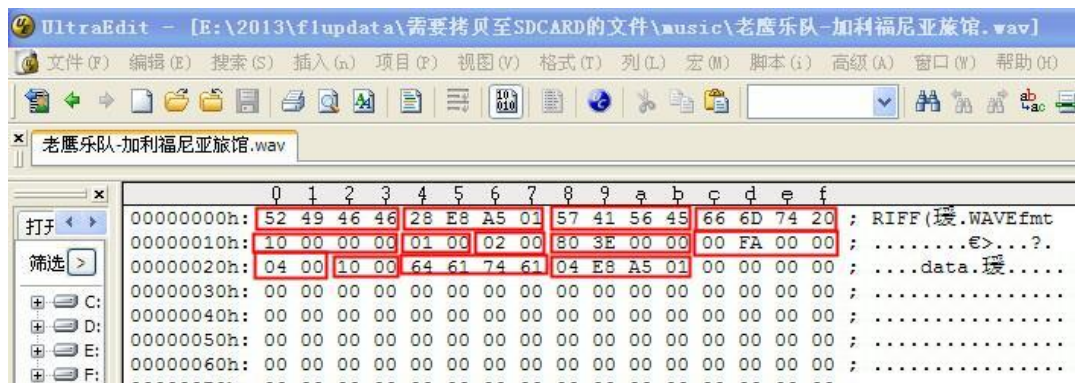
第三部分 Fmt chunk, 通常为 24 个字节, 具体内容见下表:

地址	字节数	名称	说明
0x0c	4	chunkid	永远是 fmt
0x10	4	chunksize	指 fmt chunk 的字节数, 不包括 chunked 和 chunksize
0x14	2	wformattag	格式标志, 当为 0x01 时表示音频数据没有被压缩
0x16	2	wchannels	声道数, 1 表示单声道, 2 表示立体声, 4 表示 4 声道
0x18	4	dwsamplespersec	采样频率, 如 16000HZ、44100HZ 等
0x1c	4	dwavgbyrespersec	每秒播放的字节数= dwsamplespersec* wblockalign
0x20	2	wblockalign	表示采集一个点时的总字节数, 8 位单声道为 1, 16 位双声道为 4, 16 位单声道为 2
0x22	2	wbitspersample	音频采样大小即分辨率, 指每个采样点所表示的位数

第四部分 data chunk 是整个音频文件的主体, 由 3 部分组成:

地址	字节数	名称	说明
0x24	4	chunkid	永远都是 data
0x28	4	chunksize	音频数据的大小, 即后续剩余的所有字节
0x2c	Chunk size	Wformdata[]	实际的 PCM 音频数据

我们可以用《配套资料\工具软件\UE 中文版编辑器》来查看 wav 文件的内容，这里来对照分析一下，加深大家的理解：



上图中用红色框将各部分都框了出来，我们来一一对一下：

52 49 46 46 为 ASCII 码的 “RIFF”；

28 E8 A5 01 小端模式是 01 A5 E8 28 表示后续文件的大小为 27650088 字节即 26.3M；

57 41 56 45 为 ASCII 码的 “WAVE”；

66 6D 74 20 为 ASCII 码的 “fmt” (0x20 表示空格)；

10 00 00 00 小端模式是 00 00 00 10，表示 chunksize 为 16；

01 00 小端模式是 00 01，即 1，就是 wformattag 为 1，表示没被压缩的 PCM 数据；

02 00 小端模式是 00 02，即 2 声道；

80 3E 00 00 小端模式是 00 00 3E 80，即采样频率 16K Hz；

00 FA 00 00 小端模式是 00 00 FA 00，表示每秒播放的字节数为 64000；

04 00 小端模式是 00 04，即 wblockalign 为 4，16 位采样深度、双声道的刚好就是 4(16/8\*2)；

10 00 小端模式是 00 10，表示采样深度（分辨率）为 16bit，即 2 字节；

64 61 74 61 为 ASCII 码的 “data”

04 E8 A5 01 小端模式是 01 A5 E8 04，表示后续 PCM 音频数据大小为 27650052 字节；

### Wav 例程播放流程

- 1、上电扫描 根目录下 music 文件夹中文件，如果是 wav 文件，则保存文件名供后续播放使用，这些文件名保存在根目录 filelist.txt 文件中；
- 2、如果上一步发现有可播放文件，则先读出 0x2c 个字节来分析歌曲信息（采样深度、声道数等），如果歌曲非 16bit 采样深度，则退出播放下一首；
- 3、如果歌曲采样深度为 16bit，则直接读取 PCM 数据通过 DMA 方式送到 I2S 从设备用于播放；
- 4、播放过程中左右键用于换歌，上下键用于调整音量。

## 3、libmad、helix mp3 解码库例程

### MP3 介绍

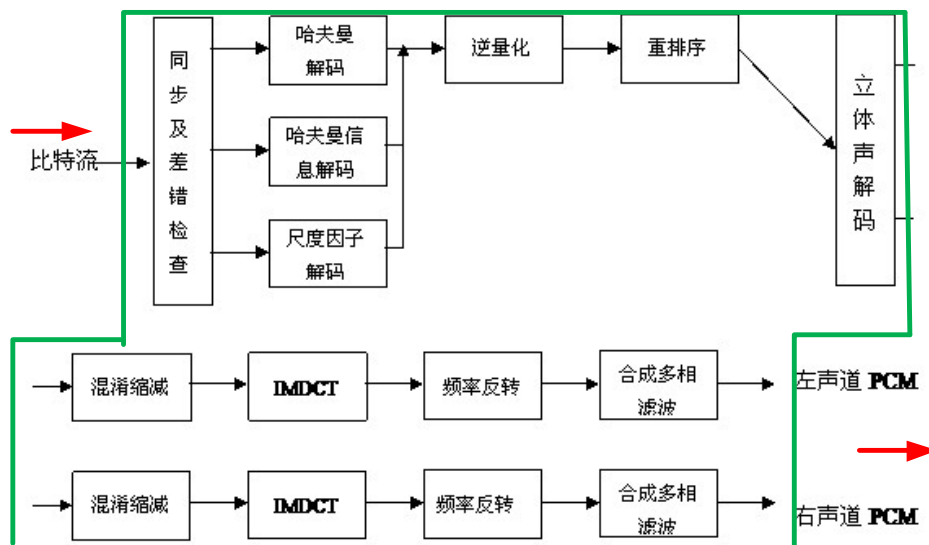
MP3 是一种音频压缩技术，其全称是 MPEG Layer-3 (Moving Picture Experts Group Audio Layer III)，简称为 MP3。它被设计用来大幅度地降低音频数据量，利用 MPEG Audio Layer 3 的技术，将音乐以 1:10 甚至 1:12 的压缩率，压缩成容量较小的文件，而对于大多数用户来说重放的音质与最初的不压缩音频相比没有明显的下降。

MP3 对音频信号采用的是有损压缩方式，为了降低声音失真度，MP3 采取了“感官编码技术”，即编码时先对文件进行频谱分析，然后用过滤器滤掉噪音电平，接着通过量化的

方式将剩下的每一位打散排列（Huffman 无损压缩编码），最后形成具有较高压缩比的 mp3 文件，并使压缩后的文件，在回放时能够达到比较接近原声音的效果。

### MP3 解码库介绍

MP3 的解码流程图如下图：



其中绿色框中的各种计算和变换是解码的关键所在，不过这些“高大上”的东东完全由解码库去完成，我们这里的 MP3 解码例程就是演示如何用解码库完成 MP3 解码和播放。

笔者知道的 MP3 解码库有两个，一个是 libmad，一个是 helix。Libmad 在 linux 下常用的音频解码库，笔者之前有做过裸奔的 libmad 解码应用（见 配套资料《基于 stm32 的软件音频解码库 libmad 移植手册.pdf》），网络收音机例程中用到的解码库就是 helix。虽然这两个解码库都可以完成 MP3 的解码工作，但是它们还是有些差异的，这里列举一下供大家参考：

- 1、libmad 的输出为 24bit PCM 音频数据，helix 的输出为 16bit PCM 数据，即输出精度上有差异；
- 2、libmad 占用 ram 较多，helix 占用 ram 较少（大约 40K）；

### MP3 文件结构

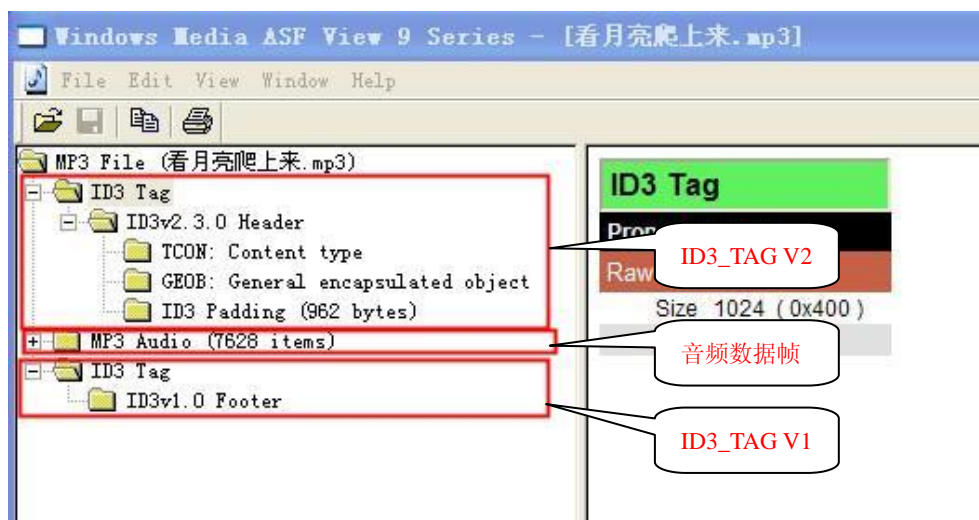
MP3 按照码率是否固定分为 CBR（固定码率）和 VBR（可变码率）两种。不管是 CBR 还是 VBR 其文件结构都大体分为 3 部分：TAGV2、Frame、TAGV1

<b>ID3V2</b>	标签信息，包含了作者、作曲、专辑等，长度不固定，扩展了 ID3V1 的信息量
<b>Frame</b>	一系列的帧，个数由文件大小和帧长决定； CBR 文件每个帧的长度固定，VBR 文件每个帧帧长不固定； 每个帧中包含帧头和数据实体两部分； 帧头记录了 mp3 的位率、采用率等信息，每个帧之间相互独立；
<b>ID3V1</b>	标签信息，包含了作者、作曲、专辑等信息，长度为固定的 128Byte

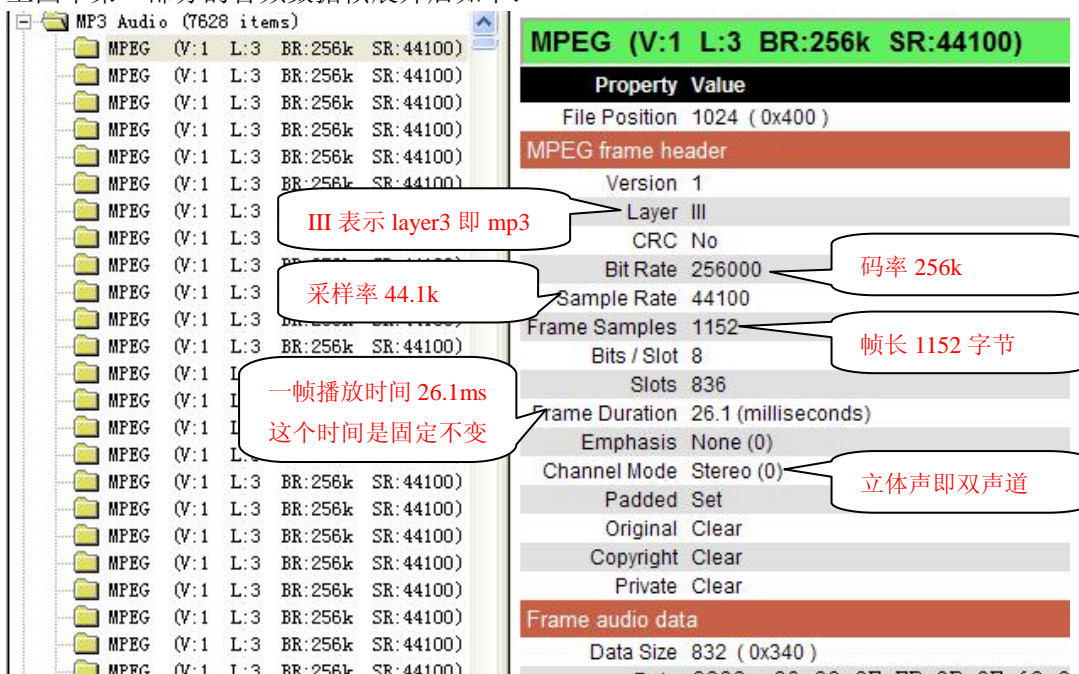
对于 VBR 文件来说，第一个数据帧中并不包含有效地音频数据，而是记录一些特殊的信息，帮助解码。

用户可以使用《配套资料\工具软件\音频相关工具\mp3 信息查看器》中的 PC 软件对 mp3 文件进行分析，加深理解，这里列出一个实际的 mp3 文件分析：





上图中第二部分的音频数据帧展开后如下：



需要用户知道的是，并不是每一个 mp3 文件都会包含 ID3 Tag 信息

### MP3 解码例程工作流程

Libmad 和 helix 两个解码库例程基本流程相同：

- 1、上电扫描 根目录下 music 文件夹中文件，如果是 mp3 文件，则保存文件名供后续播放使用，这些文件名保存在根目录 filelist.txt 文件中；
- 2、如果上一步发现有可播放文件，则先调用 ID3\_tag.c 中的 MP3\_GetInfo()函数读取文件信息（采样率、码率、播放总时间、ID3V2 标签的长度等）；
- 3、我们知道 ID3V2 标签中并不存在实际的音频数据，所以上一步中得到了 ID3V2 标签的长度，是为了在真正播放时，跳过这些标签信息直接到 Frame 部分开始解码
- 4、解码时读取到每帧数据后，要先找到帧头的位置，剩下的才是实际的音频数据，解码就是针对这些实际的音频数据进行运算处理；
- 5、解码数据帧得到的 PCM 数据通过 DMA 方式送到 I2S 从设备用于播放；

6、播放过程中左右键用于换歌，上下键用于调整音量。

#### 4、flac、ape、ogg 音频解码例程

##### Flac 、ape 、ogg 格式音频简介

FLAC 与 MP3 相仿，都是音频压缩编码，但 FLAC 是**无损压缩**，也就是说音频以 FLAC 编码压缩后不会丢失任何信息，将 FLAC 文件还原为 WAV 文件后，与压缩前的 WAV 文件内容相同。FLAC 格式音频分别 LEVEL0—LEVEL8 这 9 个压缩等级，魔笛 F1 开发板对这 9 个压缩等级都可以顺利解码。

Ape 格式也是一种**无损的音频压缩**格式，和 FLAC 格式相同，都可以不失真的还原源文件。Ape 格式音频分三个压缩等级：fast、normal、high，魔笛 F1 开发板可勉强解码 fast 压缩等级的 ape 音频，normal 格式则需要 cpu 超频到约 120M 才可以顺利解码，而 high 格式的解码所做的运算量更大，stm32f1 系列就无能为力了。

Ogg 全称应该是 OGG Vorbis，是一种新的音频压缩格式，Vorbis 是这种音频压缩机制的名字。MP3 是有损压缩格式，因此压缩后的数据与标准的 CD 音乐相比是有损失的，VORBIS 也是有损压缩，但通过使用更加先进的声学模型去减少损失，因此，同样位速率(Bit Rate)编码的 OGG 与 MP3 相比听起来更好一些。介于 stm32f1 的 cpu 处理速度，在解码 Ogg 格式音频时，断断续续，Ogg 解码例程的意义在于提供了一个可用的解码库，让我们可以在更高性能的 CPU 上去进行 Ogg 解码，比如 stm4 系列。

这三种格式音频文件都可以用《配套资料\工具软件\音频相关工具\Foobar2000》来进行转码和播放。

##### 解码例程工作流程

- 1、上电扫描 根目录下 music 文件夹中文件，如果有相应格式的音频文件(flac 或 ape 或 ogg) 文件，则保存文件名供后续播放使用，这些文件名保存在根目录 filelist.txt 文件中；
- 2、如果上一步发现有可播放文件，则先进行解码初始化，分析得出文件信息（采样率、码率、播放总时间等）；
- 3、解码数据帧，完成音频解码，得到 PCM 数据；
- 4、将得到的 PCM 数据通过 DMA 方式送到 I2S 从设备用于播放；
- 5、播放过程中左右键用于换歌，上下键用于调整音量。

## 文件系统例程

### 1、 Romfs

RT-Thread 提供了 romfs 文件系统支持，romfs 是一种存在于 rom 中的只读文件系统，其文件访问方式和传统的 DFS 访问文件方式一样，不过只能只读，其优势是占用 ram 很少，几乎不占用。

相信大家对 fatfs 都不陌生，比如 sd 卡可以被电脑格式化为 fat32 格式，然后我们就可以去使用这个文件系统，而 Romfs 存在于 rom 中，我们如何去建立这个文件系统呢？

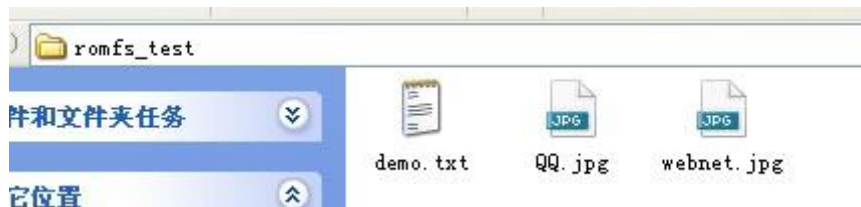
Ok，我们先来分析分析：

既然 romfs 存在于 rom 中，那么我们笨想一下，其存在方式无非就是 const 型的数组格式，只不过这些数组要按照一定的格式存放。

Ok，那么如何产生这些按照一定格式存放的数组呢？

RT-Thread 早给我们想好方法了，我这里来说一下如何做：

- 1、安装开发板配套工具中的 python-2.7.4-win32.msi 软件（在 scons 相关工具目录下），并将 python 加入到环境变量中（具体方法见连载教程 scons 环境的搭建章节）；
- 2、在电脑上建立一个文件夹，为方便描述，我们就将其命名为 romfs\_test；
- 3、给 romfs\_test 文件夹中放几个文件，这些文件将最终在我们的 romfs 文件系统中出现。注意文件不要太多、太大，因为要放入 rom 中，我们板子的 rom 总共才 512k，我放的 3 个文件如下：



- 4、将这个文件夹拷贝到 rt-thread 源码目录的 romfs 文件夹中，具体路径如下：  
RT-Thread\components\dfsfilesystems\romfs
- 5、在 RT-Thread\components\dfsfilesystems\romfs 目录下新建一个.txt 文件，写入 start cmd.exe，然后保存关闭，并将这个.txt 重命名为 startcmd.bat。其实我们这里就是建立一个启动命令行的脚本。
- 6、双击 startcmd.bat，并运行 python mkromfs.py romfs\_test romfs.c 命令：

```
D:\gitbase\stm32-nes-rtt\rtt文件系统-romfs\RT-Thread\components\dfs\filesystems\romfs>mkromfs.py romfs_test romfs.c
```

第二个参数 romfs\_test 是目录名，romfs.c 是生成的文件名。它会自动生成 romfs\_test 目录下所有的文件，并保持相应的目录结构。

- 7、至此完毕。我们可以打开 romfs.c 看看，里面就是各种数组。  
Romfs 的装载使用如下的方式：

```
#if defined(RT_USING_DFS_ROMFS)
if (dfs_mount(RT_NULL, "/romdisk", "rom", 0, DFS_ROMFS_ROOT) == 0)
{
    rt_kprintf("ROM File System initialized!\n");
}
else
    rt_kprintf("ROM File System initialization failed!\n");
#endif
```

为了验证 romfs 功能，我们可以用刚才产生的 romfs.c 文件替换《文件系统例程/romfs 例程》中原有的 romfs.c 然后编译、下载、运行，用 ls("/romdisk")命令来查看结果。

## 2、Fatfs

RT-Thread 的 fatfs 基于 elmfat，相信对于 fatfs 大家都很熟悉了。《文件系统例程/fatfs 例程》例程将魔笛开发板上的 2M spi flash 挂载到文件系统根目录 "/"，将 sd 卡挂载到根目录下的 "/SD" 目录，为了支持长文件名和中文文件名，我们在 rt\_config.h 中设置如下：

```
#define RT_DFS_ELM_USE_LFN 3
// <integer name="RT_DFS_ELM_CODE_PAGE" description="OEM code page"
default="936">
```

```
#define RT_DFS_ELM_CODE_PAGE    936
// <bool name="RT_DFS_ELM_CODE_PAGE_FILE" description="Using OEM code
page file" default="false" />
#define RT_DFS_ELM_CODE_PAGE_FILE
```

因为需要支持中文文件名，需要有一个 unicode 和国标库的转换，我们可以将转换表固化在 rom 中，也可以将转换表存放在外部 flash 或 sd 卡中，一般都放在外部以节省空间。上面的

`#define RT_DFS_ELM_CODE_PAGE_FILE` 就是说要 将转换表放在外部存储器。源码下的 `gbk2uni.tbl` 和 `uni2gbk.tbl` 就是我们需要的转换表，这两个文件的默认存放路径在 `ccfile.c` 中定义：

```
#ifndef GBK2UNI_FILE
#define GBK2UNI_FILE "/resource/gbk2uni.tbl"
#endif

#ifndef UNI2GBK_FILE
#define UNI2GBK_FILE "/resource/uni2gbk.tbl"
#endif
```

按照上面定义，我们需要将两个转换表，放入开发板的 spi flash 中的 resource 文件夹中，（当然我们也可以重新定义这个路径）

例程第一次运行时，由于 spi flash 中无任何文件（之所以发货前不在 spi flash 中放入东西，是想在这里讲解一下如何放入文件到 spi flash），我们可能看到的运行输出如下：

```
\ | /
- RT -   Thread Operating System
/ | \   1.2.0 build May  3 2014
2006 - 2013 Copyright by rt-thread team
Unable to open GBK to Unicode look up table.
Unable to open Unicode to GBK look up table.
rtc is not configured
please configure with set_date and set_time
rtc configure fail...
w25q16bv or w25q16cl or w25q16cv detection
sdcard init failed
finsh />flash0 mount to / failed!, fatmat and try ag
flash0 mount to / OK
Unable to open GBK to Unicode look up table.
Unable to open Unicode to GBK look up table.
sd0 mount to /SD failed !
```

无转换表

无 Sd 目录, 挂载失败

上面的信息表示两个查找表文件打开失败，并且挂载 sd 卡到/SD 目录失败，我们需要将所需的文件补齐：

- 1、resource 文件夹目录下所有文件复制到 tf 卡根目录，然后将 tf 卡插入开发板备用；
- 2、通过复位键，重启开发板；
- 3、Finsh 下执行 `mkdir("/SD")` 命令创建 SD 目录；（删除目录用 `rm ("SD")`）
- 4、Finsh 下执行 `mkdir("/resource ")` 命令创建 resource 目录，然后重启开发板，这样 sd 卡将能正确挂载了，运行结果如下：



```

\ | /
- RT -      Thread Operating System
/ | \      1.2.0 build May  3 2014
2006 - 2013 Copyright by rt-thread team
unable to open GBK to Unicode look up table.
unable to open Unicode to GBK look up table.
rtc is not configured
please configure with set_date and set_time
rtc configure fail...
w25q16bv or w25q16cl or w25q16cv detection
finsh />flash0 mount to / OK
unable to open GBK to Unicode look up table.
unable to open Unicode to GBK look up table.
sd0 mount to /SD OK !

```

挂载 ok

- 5、上面在 spi flash 中建立了 SD 和 resource 文件夹，我们接下来可以将 tf 卡中的文件拷贝过来了

Finsh 下执行 `copy("/SD/resource/uni2gbk.tbl", "/resource/uni2gbk.tbl")`

Finsh 下执行 `copy("/SD/resource/gbk2uni.tbl", "/resource/gbk2uni.tbl")`

- 6、重启开发板再看输出结果，就 ok 了：

```

\ | /
- RT -      Thread Operating System
/ | \      1.2.0 build May  3 2014
2006 - 2013 Copyright by rt-thread team
unable to open GBK to Unicode look up table.
unable to open Unicode to GBK look up table.
rtc is not configured
please configure with set_date and set_time
rtc configure fail...
w25q16bv or w25q16cl or w25q16cv detection
finsh />flash0 mount to / OK
sd0 mount to /SD OK !

```

- 7、到这里，文件系统就算搞定了。例程中加入了文件系统的测试程序，我们再来测试一下，在 finsh 中运行命令 `fs_test(3)`，将开始文件系统的读写测试，运行结果大概是这样的：

```

finsh />fs_test(3)
arg is : 0x03  0, 0x00000000
finsh />thread fswr2 round 1 rd:150501byte/s,wr:59445byte/s
thread fswr1 round 1 rd:71748byte/s,wr:47313byte/s
thread fswr2 round 2 rd:106007byte/s,wr:76857byte/s
thread fswr2 round 3 rd:324909byte/s,wr:51993byte/s
thread fswr1 round 2 rd:95904byte/s,wr:42477byte/s
thread fswr2 round 4 rd:100783byte/s,wr:63246byte/s
thread fswr1 round 3 rd:72837byte/s,wr:52117byte/s

```

### 3、Nfs 网络文件系统

NFS 是 Network File System 的简写，即网络文件系统。NFS 允许一个系统在网络上与他人共享目录和文件。通过使用 NFS，用户和程序可以像访问本地文件一样访问远端系统上的文件。

NFS 需要两个主要部分：一台服务器和一台（或者更多）客户机。客户机远程访问存放在服务器上的数据。

PC 上做 NFS server，运行 RT-Thread 的嵌入式系统做 NFS Client，将 PC 上的目录挂载到 RT-Thread 中

《文件系统例程\例程 4-nfs》例子中，为了使用 NFS，我们需要在配置文件 `rt_thread.h` 中

加入下面的宏：

```
#define RT_USING_DFS_NFS
#define RT_NFS_HOST_EXPORT "192.168.2.3:/" // (这个需要跟主机的 IP 保持一致)
```

上面宏指明了我们要用 NFS，并且设置好了主机端的地址。

例程用下面语句完成 NFS 文件系统的挂载：

```
rt_kprintf("begin init NFSv3 File System ...\n");

if (dfs_mount(RT_NULL, "/", "nfs", 0, RT_NFS_HOST_EXPORT) == 0)
    rt_kprintf("NFSv3 File System initialized!\n");
else
    rt_kprintf("NFSv3 File System initialization failed!\n");
```

下面我们来验证 NFS 功能：

- 1、双击运行开发板配套工具中的 FreeNFS 软件，作为 NFS Server 端，这个软件运行后会出现在系统托盘，右键选择 setting，可以看到主机的路径为：

C:\Documents and Settings\Administrator\My Documents\FreeNFS

这个路径下的文件到时就会出现在我们开发板的根目录下，为方便测试，我们可以给上述路径下放一些文件。

- 2、开发板联网，并上电，即可看到挂接 NFS 成功的运行信息：

```
mem test pass!!

\ | /
- RT -   Thread operating system
/ | \   1.2.0 build May  3 2014
2006 - 2013 Copyright by rt-thread team
dm9000 id: 0x90000a46
finsh />operating at 100M full duplex mode
lwIP-1.4.1 initialized!
Unable to open GBK to Unicode look up table.
Unable to open Unicode to GBK look up table.
rtc is not configured
please configure with set_date and set_time
found part[0], begin: 127488, size: 971.899MB
link beginning
link ok
network interface: e0 (default)
MTU: 1500
MAC: 00 60 6e 11 22 33
FLAGS: UP LINK_UP DHCP ETHARP
ip address: 192.168.2.3
gw address: 192.168.2.1
net mask  : 255.255.255.0

dns server #0: 202.96.134.33
dns server #1: 202.96.128.86
begin init NFSv3 File System ...
NFSv3 File System initialized!
```

挂载成功

- 3、这样我们就可以运行 ls("/")命令来查看 NFS 路径下的文件了。
- 4、如愿以偿看到了电脑端的文件，说明 NFS 系统已经正常运转了。

注：为了使用 NFS，例程使用了外扩 ram

## 网络例程

## 1、Network-ping

Ping 命令是一个因特网包探索器，用于测试网络联通性，属于 TCP/IP 的网络层。Ping 命令给目标 IP 发送一个 ICMP(Internet Control Messages Protocol 即因特网信报控制协议)数据包，再要求对方返回一个同样大小的数据包来确定两台网络机器是否连接相通，时延是多少。

《网络例程\例程 1-network-ping》可以用两种方式来测试网络连通性：

### 1、开发板和电脑网线直连

a、取消 lwip 的 DHCP（注释掉 rt\_config.h 中的#define RT\_LWIP\_DHCP 后编译烧录程序）

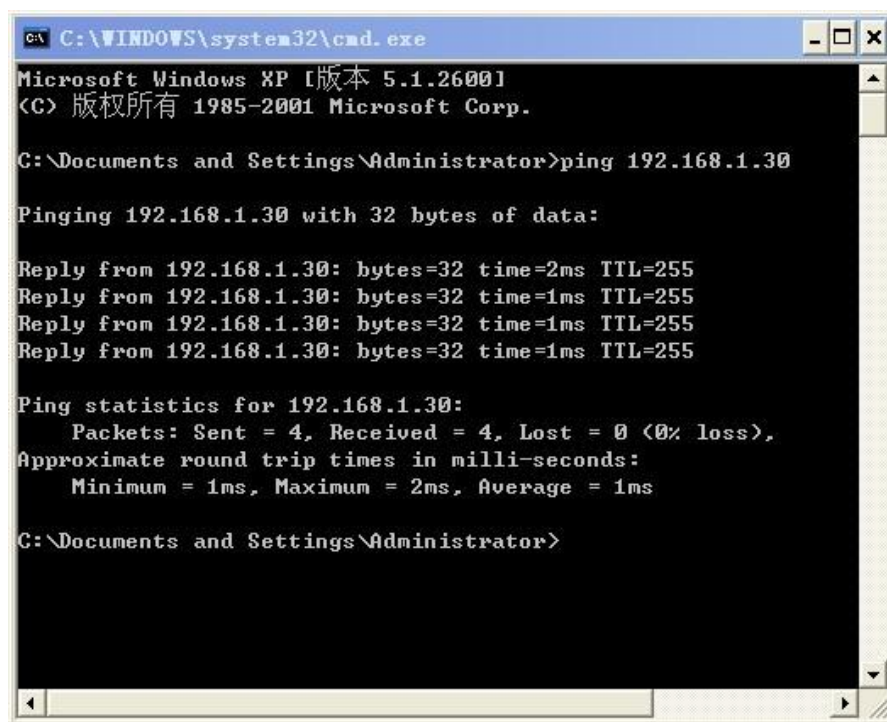
b、此时设置电脑 IP 和开发板 IP 在同一个网段

开发板默认的 IP 是 192.168.1.30

c、打开电脑命令行：开始-运行-输入 cmd-回车

d、运行 ping 192.168.1.30

e、观察输出如下：



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ping 192.168.1.30

Pinging 192.168.1.30 with 32 bytes of data:

Reply from 192.168.1.30: bytes=32 time=2ms TTL=255
Reply from 192.168.1.30: bytes=32 time=1ms TTL=255
Reply from 192.168.1.30: bytes=32 time=1ms TTL=255
Reply from 192.168.1.30: bytes=32 time=1ms TTL=255

Ping statistics for 192.168.1.30:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\Documents and Settings\Administrator>
```

### 2、开发板和电脑通过路由器连接在同一个局域网内

a、打开 lwip 的 DHCP（使能 rt\_config.h 中的#define RT\_LWIP\_DHCP 后编译烧录程序）

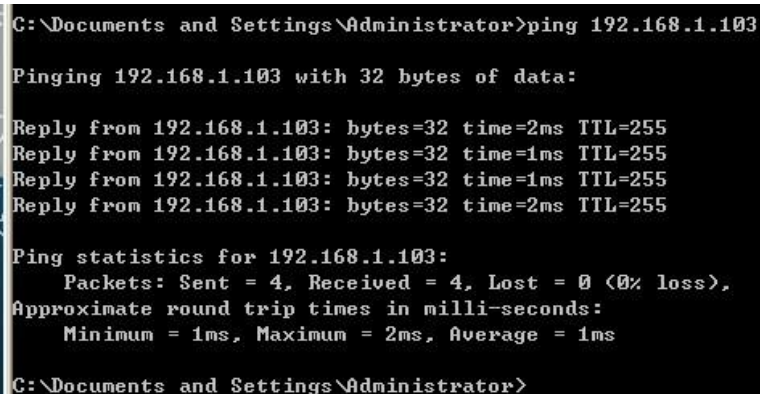
b、电脑连接路由器；开发板接连路由器。

b、通过在开发板上运行 list\_if()命令或通过查看路由器客户端列表 确定开发板的 IP 地址（笔者的 IP 是 192.168.1.103）

d、打开电脑命令行：开始-运行-输入 cmd-回车

e、运行 ping 192.168.1.XXX （笔者运行 192.168.1.103）

f、观察输出如下：



```
C:\Documents and Settings\Administrator>ping 192.168.1.103

Pinging 192.168.1.103 with 32 bytes of data:

Reply from 192.168.1.103: bytes=32 time=2ms TTL=255
Reply from 192.168.1.103: bytes=32 time=1ms TTL=255
Reply from 192.168.1.103: bytes=32 time=1ms TTL=255
Reply from 192.168.1.103: bytes=32 time=2ms TTL=255

Ping statistics for 192.168.1.103:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\Documents and Settings\Administrator>
```

以上实验结果说明 Lwip 正常工作，开发板的网络部分（硬件和驱动）一切 OK。

## 2、网络基础例程-telnet 远程命令行交互

telnet 协议是 TCP/IP 协议族中的一员，是 Internet 远程登陆服务的标准协议和主要方式。它为用户提供了在本地计算机上完成远程主机工作的能力。在这里，我们利用 telnet 的方式访问 finsh，使得 finsh 操作可通过网络方式进行，这是 RT-Thread 又一个强大而实用的功能。

我们在初始化线程中调用 telnet\_srv（）启动了 telnet 服务端（默认在 23 端口上监听），这个函数建立了一个 telnet 线程，此线程向系统注册了一个“telnet 设备”并通过如下语句，将 finsh 和这个“telnet”设备关联了起来（具体细节，参见 telnet.c）：

```
rt_console_set_device("telnet");
/* set finsh device */
finsh_set_device("telnet");
```

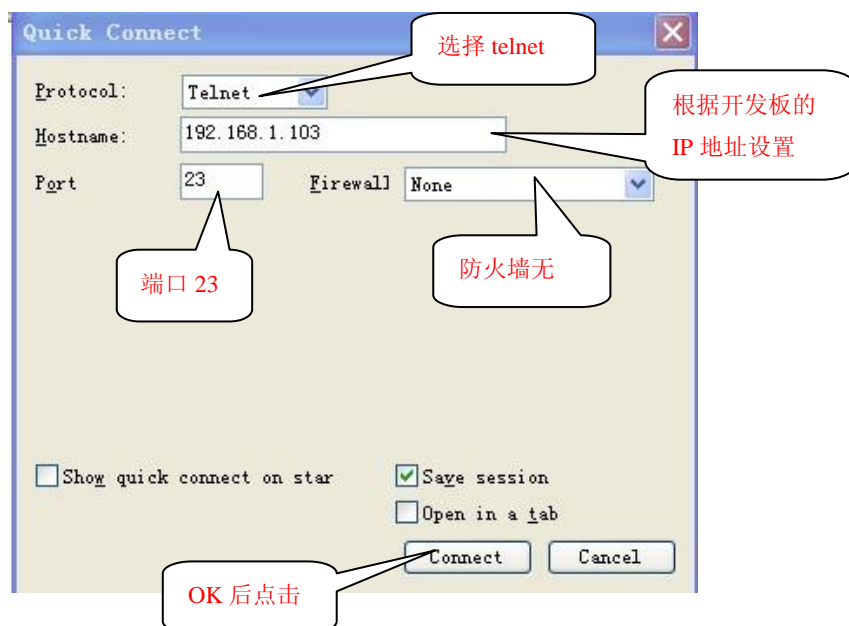
这个例程的测试我们使用电脑和开发板通过路由器连接的方式：

- a、电脑连接路由器，开发板连接路由器；
- b、电脑运行 telnet 客户端程序：

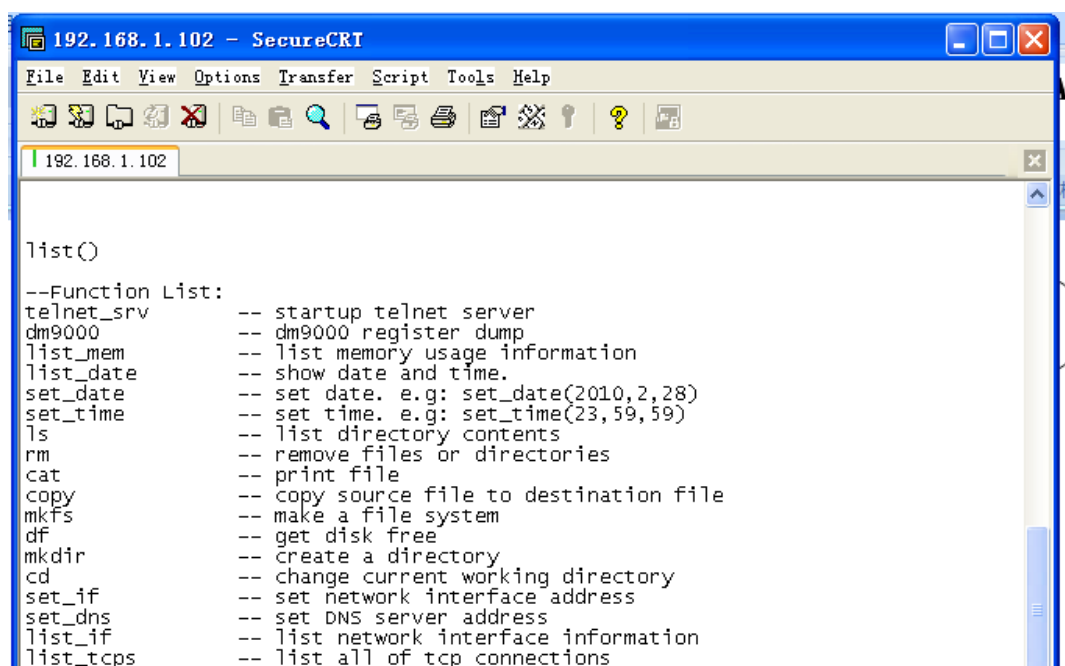
我们在 PC 机上打开一个 telnet 客户端：运行魔笛配套资料的工具文件夹中的 SecureCRT 软件，选择 File->Quick Connect，按下图设置：

（其中 hostname 的 IP 地址指的是开发板从路由器上获得的 IP 地址，可以通过查看路由器客户端列表得到）





c、连接上之后，运行 list () 命令来测试，结果如下图：



### 3、TCP 、UDP 服务器 客户端

Tcp sever、Tcp client、Udp sever、Udp client 例子详情见《一起来学 rtthread 连载教程-19 篇》，这里不累述

### 4、Netio 网络测速

《网络例程\例程 2-network-netio》例程用于测试开发板的网络数据收发速度。我们在开发板上运行 netio 服务端程序，在 PC 端运行 netio 客户端程序，通过客户端程序和服务端程序的数据交互来完成网络通信和速度的测试。该例程主要是测试 TCP 的网络连接速度，对于 UDP 则没有进行测试。

烧录此程序后测速方法如下：

- a、开发板连接路由器，上电；
- b、从路由器客户端列表或串口的输出确定开发板获得的 IP 地址（我这里获得的 ip 是 192.168.1.102）
- c、解压开发板资料中附赠的工具软件-netio 测试工具，从 PC 端 CMD 命令行使用 cd 命令进入到 netio 工具软件的 bin 目录
- d、运行命令 win32-i386 -t 192.168.1.102，等待测试完成，就可以看到开发板的网络速度数据了，如下图：

```
C:\Documents and Settings\Administrator\桌面\netio132\bin>win32-i386 -t 192.168.1.102

NETIO - Network Throughput Benchmark, Version 1.32
(C) 1997-2012 Kai Uwe Ronnel

TCP connection established.
Packet size 1k bytes: 357.93 KByte/s Tx, 4968 Byte/s Rx.
Packet size 2k bytes: 369.88 KByte/s Tx, 9.42 KByte/s Rx.
Packet size 4k bytes: 367.88 KByte/s Tx, 6708 Byte/s Rx.
Packet size 8k bytes: 386.23 KByte/s Tx, 6719 Byte/s Rx.
Packet size 16k bytes: 385.54 KByte/s Tx, 6.54 KByte/s Rx.
Packet size 32k bytes: 368.64 KByte/s Tx, 6.55 KByte/s Rx.
Done.
```

从这个数据来看，我的这个网络环境下速度还是较慢的，因为我电脑有线网口换了，不能使用电脑和 pc 直连的方式来测试，只能通过路由器中转，并且电脑还是通过 wifi 和路由器连接的，所以测试速度自然高不到哪里去。

另外为了获得较佳的速度，请多尝试几个

RT\_LWIP\_TCP\_SND\_BUF、RT\_LWIP\_TCP\_WND 值

## 5、Tftp 文件传输

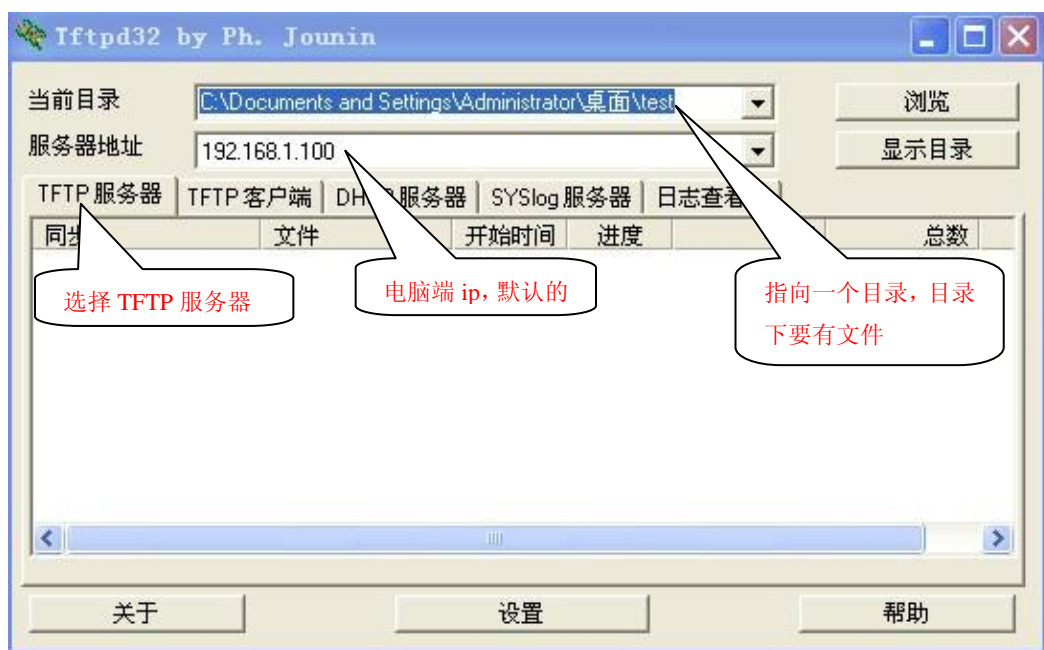
TFTP（Trivial File Transfer Protocol,简单文件传输协议）是 TCP/IP 协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。TFTP 服务端口号为 69，基于 UDP 协议而实现，协议设计的时候主要用于进行小文件传输。

《网络例程\例程 8-network-tftplient》例程演示了 TFTP 的功能，为了要进行文件传输，例程实现了 fatfs 文件系统，将 sd 卡作为文件系统根目录。

例程中的 TFTP 功能为我们提供了 tftp\_get 和 tftp\_put 的两个命令，前者用于从服务器端获得文件，后者用户将本地文件传到服务器上。

下面来讲解如何进行功能验证：

- 1、打开开发板配套工具软件中的 tftpd32 软件，并进行相应设置：



- 2、开发板连接网线并上电，设置好 SecureCRT 软件，用于运行 finsh 命令
- 3、运行 `tftp_get ("192.168.1.100", "/", "1.txt")` 命令即可将电脑端的 1.txt 文件传到开发板 tf 卡根目录（1.txt 文件不要太大）

```

tftp_get          -- get file from tftp server
finsh />tftp_get("192.168.1.100", "/", "1.txt")
#done
          0, 0x00000000
finsh />

```

- 4、运行 `tftp_put ("192.168.1.100", "/", "setup.ini")` 命令即可将开发板 tf 卡根目录下的 setup.ini 文件传到电脑端

```

finsh />tftp_put("192.168.1.100", "/", "setup.ini")
#done
          0, 0x00000000
finsh />

```

## 6、简单的 web 服务器

《网络例程\例程 9-network-simple-webserver》例程设计了一个简单的 web 服务器应用，它由单一线程组成，负责接收来自网络的连接，响应 HTTP 请求，以及关闭连接。

这个简单的 web 服务器只响应 HTTP GET 对文件 "/" 的请求，并且检测到请求就会发出响应。这里，我们既需要发送针对 HTML 数据的 HTTP 头，还要发送 HTML 数据，所以对 `netconn_write()` 函数调用了两次。因为我们不需要修改 HTTP 头和 HTML 数据，所以我们将 `netconn_write()` 函数的 `flags` 参数值设为 `NETCONN_NOCOPY` 以避免复制。最后，连接被关闭并且 `process_connection()` 函数返回。连接结构也会在这个调用后被删除。

开发板连接上网络后，我们从 PC 端打开 IE 输入 192.168.1.102(开发板获取到的 IP)既看到如下图的信息：



## 7、Web 服务器-webnet

WebNet 是 RT-Thread 上的新一代 web server，贴身为小型嵌入式设备订做，具有低资源占用，扩展性好的特点。在 RT-Thread 实时操作系统的移植中，对它进行了部分优化，在保留它全部功能的同时，把它的常规内存占用减少到了 10kB RAM 以下。

webnet 功能概要

- ✓ 支持 HTTP 1.0/1.1 规范；
- ✓ 支持 basic authentication
- ✓ 支持 CGI
- ✓ 支持 ASP 变量替换
- ✓ 支持路径 alias
- ✓ 支持目录 index

在验证例程《网络例程\例程 11-network-webnet》前，需要将代码目录中的 resource 和 webnet 文件夹考入到 tf 卡，保证电脑和开发板在同一个局域网中，联网上电后，打开电脑端 ie 浏览器，输入开发板的 IP 地址（可以通过 finsh 下查看），即可。





## 8、Wifi 模块使用

具体见《wifi 例程/wifi 模块使用说明.pdf》

### Zmodem 协议传输文件例程

用串口来传输文件，有这么 3 种协议：Xmodem、Ymodem、Zmodem，简单介绍一下

(1) Xmodem: 这种古老的传输协议速度较慢，但由于使用了 CRC 错误侦测方法，传输的准确率可高达 99.6%。

(2) Ymodem: 这是 Xmodem 的改良版，使用了 1024 位区段传送，速度比 Xmodem 要快。

(3) Zmodem: Zmodem 采用了串流式 (streaming) 传输方式，传输速度较快，而且还具有自动改变区段大小和断点续传、快速错误侦测等功能。这是目前最流行的文件传输协议。

RT-Thread 提供了和后两种传输协议的支持。

我们先来看看 Zmodem 协议如何来传输文件：

- 1、下载《Y-Zmodem 协议例程\例程 2-zmodem》例程到开发板，并运行。
- 2、打开具有 zmodem 协议的串口终端，我们这里使用电脑自带的超级终端，连接时选择相应的 com 口



确认后设置好串口参数，如下：

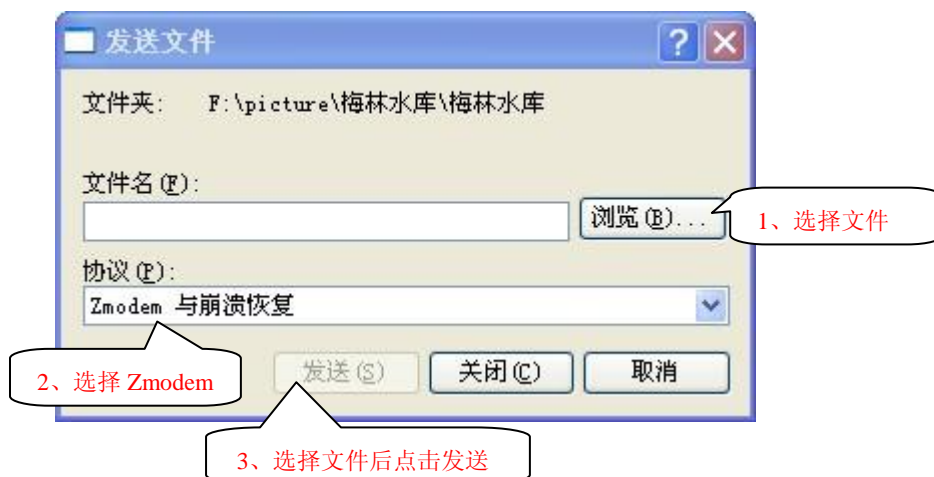


然后点击确定就好了。

- 3、按一下开发板上复位键，来检查一下是否能正常串口输出信息。
- 4、先来测试从电脑端发送数据到开发板的文件系统，超级终端里输入 rz("/music") 回车，可看到如下信息：

```
finsh />rz("/music")
0, 0x00000000
finsh />
rz: ready...
```

其中的 rz:ready 表示开发板已经准备好接收文件了，(最后文件会被保存在 tf 卡的 music 文件夹下)，这时我们点击超级终端的传送--发送文件，并选择要发送的文件：



点击发送后，出现下面的界面，表示文件在传输中：



传输完成后，串口输出如下：

```
size: 545768 bytes  
receive completed.  
finsh />
```

传输完成后，我们可以拿下 tf 卡，插在电脑上检查接收到的文件。

#### 5、测试从开发板发送文件到电脑：

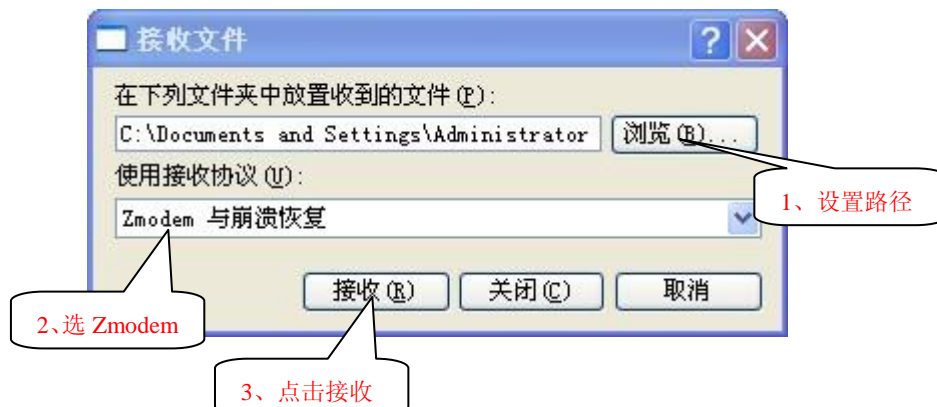
再把 tf 卡插回开发板，复位下开发板，finsh 里输入 sz ( " /music/1.jpg" ) 并回车。

(其中 /music/1.jpg 是笔者 tf 卡中的文件路径，读者需要按照自己 tf 卡中文件设置)

当出现如下图中的 sz: ready 时表示开发板已经准备好像电脑发送文件了

```
sz("/music/1.jpg")  
0, 0x00000000  
finsh />  
sz: ready...
```

我们再点击超级终端中的传送—接收文件，并设置好存放路径：



开始接收后将出现下面界面：



接收完成后，串口输出如下：

```
file: 1.jpg  
size: 692255 bytes  
send completed.  
finsh />
```

完成后我们可以在电脑端查看刚接收到的文件（如果是文本文件也可以直接在 finsh 下使用 cat 命令查看）。

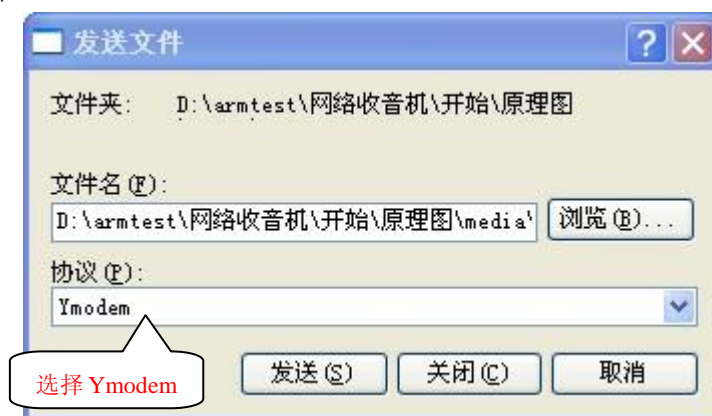
### Ymodem 协议传输文件例程

《Y-Zmodem 协议例程\例程 1-ymodem》例程实现了使用 Ymodem 协议从电脑端发送文件到开发板的功能，从开发板发送文件到电脑的功能暂时还没有，具体可以看这个帖子：

<http://www.rt-thread.org/phpBB3/viewtopic.php?t=2514>

验证方法和 Zmodem 传输相似，不同的是 Zmodem 使用 rz 命令，Ymodem 使用 ry 命令。

- 1、按照上例设置好超级终端；
- 2、运行 ry ( “/music/” ) 命令 ；(这个路径可自行设置)
- 3、串口会每隔 3 秒输出字符 ‘C’ ，提示我们尽快传输，连续输出 10 次 ‘C’ 字符后将结束，我们需要在这 10 个字符输出完之前设置好发送，设置也和上例相似，只是需要选择 Ymodem：



- 4、选择完成后点击发送，开始传输：





- 5、传输完成后，拿下 tf 卡，插入电脑查看收到的文件（如果是文本文件也可以直接在 finsh 下使用 cat 命令查看）。

我们在实际应用中可以将上数两种协议加入到工程，这样往开发板上拷贝资料就很方便了

## 五、 常见问题汇总

### 1、收音机例程我自己编译后下载，播放歌曲没有声音、并且容易死机

这个问题，很多用户都有遇到过，已经证实是编译器版本问题，请升级 MDK 最新版，推荐使用 MDK4.54。

### 2、编译收音机例程时，遇到关于“;”的编译 error

请直接注释或删除 error 所在行。