Libmad 音频解码库移植手册

本手册由 UP MCU 工作室编写，若要转载请注明出处

2012-8-12

前言

目前，网络上绝大都数关于 MP3 播放器的例子，大都采用 vs1003 这颗硬件解码芯片或者使用 AT89C51SND1C 这颗含有内部音频解码电路的 NB 单片机，软件音频解码的例子少之又少。UP MCU 工作室的相关人员花了些时间、精力，研究了目前 linux 领域很常用的一个开源音频解码库——libmad 的解码流程，并将其成功移植到裸奔的 stm32 平台上。本着资源共享的奉献精神，本工作室将移植过程整理成手册，发布于网络，希望对大家有用。

本手册移植工作所对应的软硬件平台如下：
操作系统：windows XP
开发环境：MDK V4.23
STM 固件库版本号：V3.5.0
主芯片 ：STM32F103RET6（512K flash 64KRAM）运行于 72M
DA 芯片 ：PCM1770PW

由于本人水平有限，文中若有不对的地方，欢迎拍砖。拍砖地址 447926737wangkai@163.com

1.Libmad 简介

LIBMAD 是一个高质量的音频解码库，MAD 的全称是 MPEG Audio Decoder。LIBMAD 目前支持 MPEG-1、低采样率的 MPEG-2 和 MPEG2.5 格式的 Layer Ⅰ、Layer Ⅱ、LayerⅢ（即 MP3）的解码。

MAD 具有如下特性：

●高精度的 24-BIT PCM 输出；

●100%使用定点运算；

●完全基于 ISO/IEC 标准；

●支持 GNU GPL 协议。

MAD 完全采用 C 语言编写（里面的 IMDCT 部分和乘法运算可根据平台不同使用汇编来加快运算速度），它对 MP3 解码算法做了很多优化，非常适合在没有浮点支持的嵌入式环境下使用。利用 MAD 提供的 API，我们可以很容易的实现音频解码。

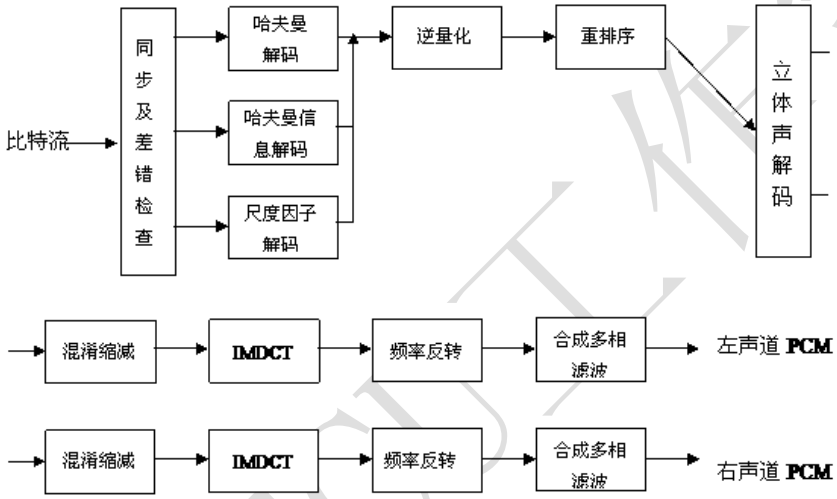
MAD 的源码我们可以在 http://sourceforge.net/projects/mad/files/下载到。

http://www.underbit.com/products/mad 有关于 libmad 的介绍和其他相关的资料，其中 madlld 这个项目是一个 libmad 低层 API 的 demon，我的解码工作就是移植 libmad 和利用 madlld 搭好的框架解析 MP3 数据得到 PCM 格式的数据。

2. Libmad 简单分析

下载 libmad-0.15.1b 解压后除了 msvc++这个文件夹，其他的都是单个文件。其中所有的.C、 .H、.dat 文件都是我们所需要的。minimad.c 这个文件是对 libmad 高层 API 的使用示例，从代码中可以看到它是基于类 unix 平台的。大致瞅瞅，我们可以看到使用 libmad，解码真的很简单！

下图大致展示了 MP3 解码的流程：



其中同步及差错检查包括了头解码模块，在主控模块开始运行后，主控模块将比特流的数据缓冲区交给同步及差错检查模块，此模块包含两个功能，即头信息解码及帧边信息解码，根据它们的信息进行尺度因子解码及哈夫曼解码，得出的结果经过逆量化，立体声解码，混淆缩减，IMDCT，频率反转，合成多相滤波这几个模块之后，得出左右声道的 PCM 码流，再由主控模块将其放入输出缓冲区输出到声音播放设备。

下面我们主要分析一下 libmad 里面用到的一些重要数据结构，这些都定义在 mad.h 文件中。

```
struct mad_stream {
  unsigned char const *buffer;          /* input bitstream buffer */
  unsigned char const *bufend;          /* end of buffer */
  unsigned long skiplen;                /* bytes to skip before next frame */
  int sync;                             /* stream sync found */
  unsigned long freerate;               /* free bitrate (fixed) */
  unsigned char const *this_frame;      /* start of current frame */
  unsigned char const *next_frame;      /* start of next frame */
  struct mad_bitptr ptr;                /* current processing bit pointer */
```

```
  struct mad_bitptr anc_ptr;            /* ancillary bits pointer */
  unsigned int anc_bitlen;              /* number of ancillary bits */
  unsigned char (*main_data)[MAD_BUFFER_MDLEN];
                                       /* Layer III main_data() */
  unsigned int md_len;                  /* bytes in main_data */
  int options;                          /* decoding options (see below) */
  enum mad_error error;                 /* error code (see above) */};
```

此数据结构存放解码前的位流数据。

```
struct mad_synth {
  mad_fixed_t filter[2][2][2][16][8];   /* polyphase filterbank outputs */
                /* [ch][eo][peo][s][v] */
  unsigned int phase;                   /* current processing phase */
  struct mad_pcm pcm;                   /* PCM output */};
```

此数据结构存放解码合成滤波后的 PCM 数据，pcm 域比较重要：

```
struct mad_pcm {
  unsigned int samplerate;              /* sampling frequency (Hz) */
  unsigned short channels;              /* number of channels */
  unsigned short length;                /* number of samples per channel */
  mad_fixed_t samples[2][1152];         /* PCM output samples [ch][sample]
*/};
```

它定义了音频的采样率，声道个数和 PCM 采样数据，我们用这里面的信息来初始化音频设备。libmad 是以桢（frame）为单位对 MP3 进行解码的，当正确的解码完一帧数据可以得到（每声道）1152 个 PCM 数据。那么一帧数据量到底有多少呢？——这个值是变化的，我们可以用下面的公式来计算：

FrameSize = (((MpegVersion == MPEG1 ? 144 : 72) * Bitrate) / SamplingRate) + PaddingBit

例如: Bitrate = 128000, a SamplingRate =44100, and PaddingBit = 1

FrameSize = (144 * 128000) / 44100 + 1 = 417 bytes

也就是说，想解码一个比特率为 128K，采样率为 44.1K 的 MP3 文件，最少一次读入内存 417 bytes 以准备解码，通常我们需要读入的字节数要比一帧的数据量多一些。

```
struct mad_frame {
  struct mad_header header;             /* MPEG audio header */
  int options;                          /* decoding options (from stream) */
  mad_fixed_t sbsample[2][36][32];      /* synthesis subband filter samples */
  mad_fixed_t (*overlap)[2][32][18];    /* Layer III block overlap data */};
```

上面数据结构记录 MPEG 帧解码后 PCM 数据的数据结构，其中 mad_header 这个结构体比较重要，它的内容如下：

```
struct mad_header {
  enum mad_layer layer;                 /* audio layer (1, 2, or 3) */
  enum mad_mode mode;                   /* channel mode (see above) */
  int mode_extension;                   /* additional mode info */
  enum mad_emphasis emphasis;           /* de-emphasis to use (see above) */
  unsigned long bitrate;                /* stream bitrate (bps) */
```

```
  unsigned int samplerate;          /* sampling frequency (Hz) */
  unsigned short crc_check;         /* frame CRC accumulator */
  unsigned short crc_target;        /* final target CRC checksum */
  int flags;                        /* flags (see below) */
  int private_bits;                 /* private bits (see below) */
  mad_timer_t duration;             /* audio playing time of frame */

};
```

我们可以在 layer 这个域中得到音频数据所采的层，在 mode 域中得到音频数据的声道个数，在 birrate 和 samplerete 中得到音频数据的位率（128kbps、384kbps 等等）和采样率（22KHZ、44.1KHZ、48KHZ 等等）

Libmad 解码分同步方式和异步方式两种，所谓同步方式是指解码函数在解码完一帧后才返回并带回出错信息，异步方式是指解码函数在调用后立即返回，通过消息传递解码状态信息。我们采用裸奔的方式，当然只能使用同步解码。

3. Libmad 移植

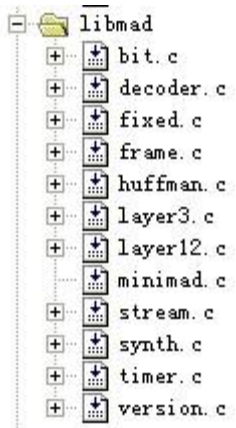
从前面的解码框图中我们知道，使用 libmad 解码，我们只需要把音频文件流读入给 libmad，然后把解码得到的 PCM 数据进行播放就可以。

移植前，我们需要准备好已经能进行文件系统读写的工程模板，相信对文件系统的应用，大家都不陌生，我采用的是 FatfsR0.09，从 SD 卡中读入 MP3 数据给 libmad 解码，最后使用 PCM1770 进行 DA 转换。如果有对文件系统移植不熟悉的童鞋，可以参考我例程中的文件系统移植模板的源码。

Ok，废话少说，下面我们真正的开始移植

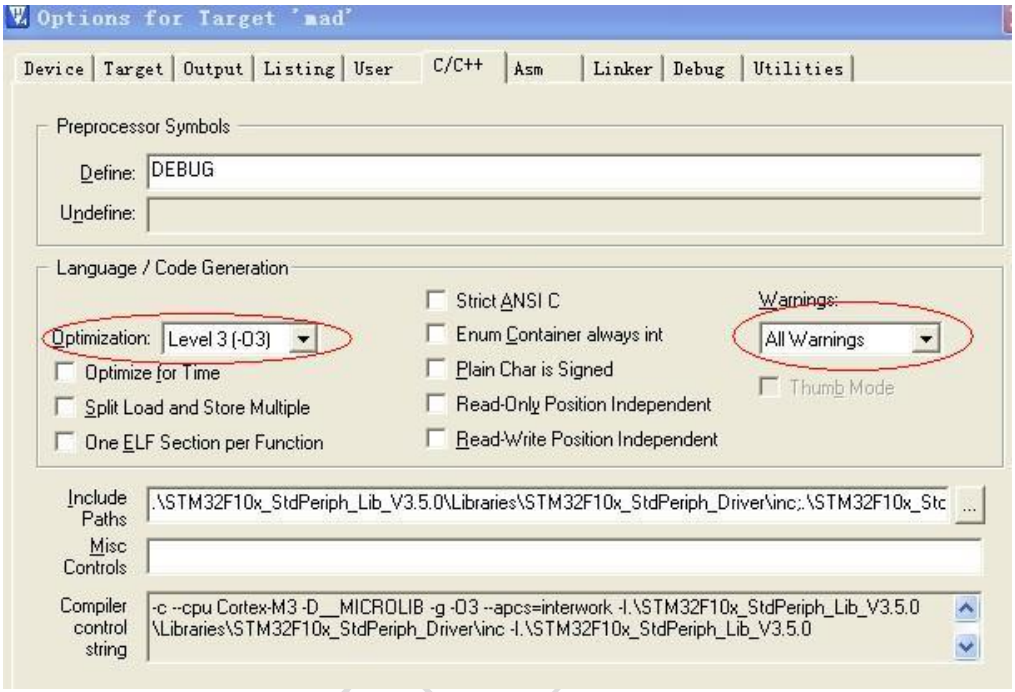
◆ 添加源码到工程

解压源码包内 libmad-0.15.1b.rar，然后将 libmad-0.15.1b 文件夹复制到工程目录，把里面所有.c 文件都加入到 MDK 工程，并设置好包含路径，防止.h 和 .bat 文件在编译时被提示找不到，加入后如下图：



◆ **解开编译警告**

接下来我们先编译一下，看有哪些错误。这里说明下，为得到最彻底的优化，我的编译选项的优化等级选择了等级 3，同时 waring 选择了 all warings，如下图：



首次编译后，我们会看到 3 种“致命”编译警告：

1、 找不到 <unistd.h>

2、 abort 函数声明问题

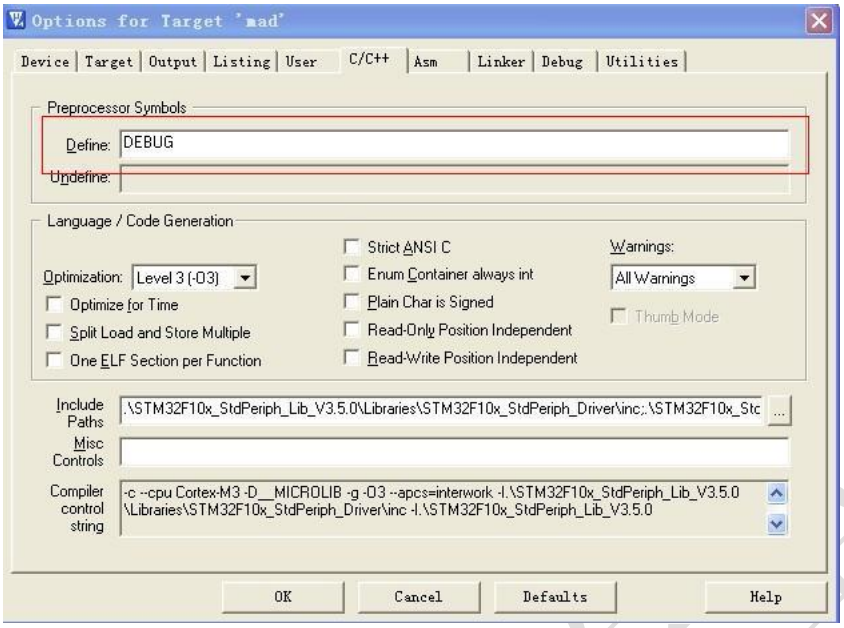
3、 FPM 没选择

先说第一个，前面讲过 minimad.c 这个文件是对 libmad 的高层 API 在 unix 下的应用范例，我们直接把 minimad.c 这个文件从工程中删除或者注释掉里面所有内容即可解决掉第一个警告。

第二个警告是 断言语句声明问题，我们直接在 global.h 的最后那段里把其声明为空就行：

```
# if !defined(HAVE_ASSERT_H)
#   if defined(NDEBUG)
#     define assert(x)              /* nothing */
#   else
#     define assert(x)              /* nothing *///do { if (!(x)) abort(); } while (0)
#   endif
# endif
```

剩下的那个警告需要对 libmad 进行合理的配置。我们知道，让移植性很强的源码在某个确定的平台上运行，都需要针对这个平台进行配置（如 fatfs、ucos），我们可以使用 msvc++ 这个文件夹中的 config.h 文件来进行配置选择（需要把 config.h 包含进各个文件），也可以在 kei 如下界面直接进行宏定义来配置



具体的配置细节我这里不多说，我 copy 下我 config.h 中的内容下来，大家看看应该很容易明白这些配置的意义，其中红色字体是起作用的配置。

```
/* config.h.  Generated by configure.  */
/* config.h.in.  Generated from configure.ac by autoheader.  */

/* Define to enable diagnostic debugging support. */
/* #undef DEBUG */

/* Define to enable experimental code. */
/* #undef EXPERIMENTAL */

/* Define to 1 if you have the <assert.h> header file. */
//#define HAVE_ASSERT_H 1

/* Define to 1 if you have the <dlfcn.h> header file. */
/* #undef HAVE_DLFCN_H */

/* Define to 1 if you have the <errno.h> header file. */
//#define HAVE_ERRNO_H 1

/* Define to 1 if you have the `fcntl' function. */
/* #undef HAVE_FCNTL */

/* Define to 1 if you have the <fcntl.h> header file. */
//#define HAVE_FCNTL_H 1

/* Define to 1 if you have the `fork' function. */
/* #undef HAVE_FORK */
```

```
/* Define to 1 if you have the <inttypes.h> header file. */
//#define HAVE_INTTYPES_H 1

/* Define to 1 if you have the <limits.h> header file. */
//#define HAVE_LIMITS_H 1

/* Define if your MIPS CPU supports a 2-operand MADD16 instruction. */
/* #undef HAVE_MADD16_ASM */

/* Define if your MIPS CPU supports a 2-operand MADD instruction. */
/* #undef HAVE_MADD_ASM */

/* Define to 1 if you have the <memory.h> header file. */
//#define HAVE_MEMORY_H 1

/* Define to 1 if you have the `pipe' function. */
/* #undef HAVE_PIPE */

/* Define to 1 if you have the <stdint.h> header file. */
//#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1

/* Define to 1 if you have the <strings.h> header file. */
#define HAVE_STRINGS_H 1

/* Define to 1 if you have the <string.h> header file. */
#define HAVE_STRING_H 1

/* Define to 1 if you have the <sys/stat.h> header file. */
//#define HAVE_SYS_STAT_H 1

/* Define to 1 if you have the <sys/types.h> header file. */
//#define HAVE_SYS_TYPES_H 1

/* Define to 1 if you have <sys/wait.h> that is POSIX.1 compatible. */
/* #undef HAVE_SYS_WAIT_H */

/* Define to 1 if you have the <unistd.h> header file. */
/* #undef HAVE_UNISTD_H */

/* Define to 1 if you have the `waitpid' function. */
```

```
/* #undef HAVE_WAITPID */

/* Define to disable debugging assertions. */
/* #undef NDEBUG */

/* Define to optimize for accuracy over speed. */
/* #undef OPT_ACCURACY */

/* Define to optimize for speed over accuracy. */
/* #undef OPT_SPEED */

/* Define to enable a fast subband synthesis approximation optimization. */
/* #undef OPT_SSO */

/* Define to influence a strict interpretation of the ISO/IEC standards, even
   if this is in opposition with best accepted practices. */
/* #undef OPT_STRICT */

/* Name of package */
#define PACKAGE "libmad"

/* Define to the address where bug reports for this package should be sent. */
#define PACKAGE_BUGREPORT "support@underbit.com"

/* Define to the full name of this package. */
#define PACKAGE_NAME "MPEG Audio Decoder"

/* Define to the full name and version of this package. */
#define PACKAGE_STRING "MPEG Audio Decoder 0.15.1b"

/* Define to the one symbol short name of this package. */
#define PACKAGE_TARNAME "libmad"

/* Define to the version of this package. */
#define PACKAGE_VERSION "0.15.1b"

/* The size of a `int', as computed by sizeof. */
#define SIZEOF_INT 4

/* The size of a `long', as computed by sizeof. */
#define SIZEOF_LONG 4

/* The size of a `long long', as computed by sizeof. */
#define SIZEOF_LONG_LONG 8
```

```
/* Define to 1 if you have the ANSI C header files. */
#define STDC_HEADERS 1

/* Version number of package */
#define VERSION "0.15.1b"

/* Define to empty if `const' does not conform to ANSI C. */
/* #undef const */

/* Define as `__inline' if that's what the C compiler calls it, or to nothing
    if it is not supported. */
#define inline __inline

#define FPM_DEFAULT          //FPM_ARM  芯片架构选择

#define OPT_SPEED            // 速度优先

//#define OPT_SSO

//#define OPT_ACCURACY    //精度优先

//#define ASO_IMDCT        //使用汇编来 进行 ASO_IMDCT

//#define OPT_STRICT
//#define    ASO_INTERLEAVE1
```

另外 mad.h 第 27 行改为 #define FPM_DEFAULT

在对芯片架构的选择上 我在 MDK 下尝试了 N 种方式来使用 FPM_ARM 这个选项，可惜由于 GNU 汇编和 arm 汇编差异太大，都没整成功，如果能成功的话相信解码速度应该会更快。如果有人尝试 OK 的话，请告诉我。

◆ 动态内存分配修改

Libmad 里有几个地方（decoder.c、layer3.c）使用了动态内存分配，keil 下 stm32 在分配动态内存时会出错，所以我们需要把它们改成静态分配的形式（可搜索关键字 malloc 和 calloc 来确定如何修改），我说说改动的细节：

mad.h 712 行 unsigned char (*main_data)[MAD_BUFFER_MDLEN];
改为 main_data_t *main_data;

mad.h 696 行加上 main_data_t 的定义：
typedef unsigned char main_data_t[MAD_BUFFER_MDLEN];

stream.h 中 76 行 unsigned char (*main_data)[MAD_BUFFER_MDLEN];
改为 main_data_t *main_data;

Stream.h 中 60 行加上 main_data_t 的定义：
typedef unsigned char main_data_t[MAD_BUFFER_MDLEN];

Stream.c 中定义全局变量：main_data_t MainData;

Layer3.h 中声明：extern main_data_t MainData;

Layer3.c 中 2531 行 stream->main_data = malloc(MAD_BUFFER_MDLEN);
改为：stream->main_data = &MainData;

decoder.h 中 mad_decoder 结构体定义（第 52 行）中的

```
struct {
  struct mad_stream stream;
  struct mad_frame frame;
  struct mad_synth synth;
} *sync;
```

用 struct sync_t *sync;替代
并定义 sync_t 如下：

```
struct sync_t {
  struct mad_stream stream;  // definito main_data_t un array di circa 4K
  struct mad_frame frame;
  struct mad_synth synth;
};
```

decoder.c 中定义全局变量 struct sync_t Sync;

decoder.c 中 554 行 decoder->sync = malloc(sizeof(*decoder->sync));
改为：decoder->sync = &Sync; 并注释掉相应的 free 语句。

Layer3.c 中定义全局变量：
unsigned char frame_overlap_buff[2 * 32 * 18 * sizeof(mad_fixed_t)];
同时把 2741 行 frame->overlap = calloc(2 * 32 * 18, sizeof(mad_fixed_t));改为：
frame->overlap = (void*)frame_overlap_buff;

◆ Libmad 源码中 DEGUB 预编译去掉

Libmad 代码中使用 DEBUG 预编译作为调试开关，我们将其屏蔽掉，防止我们自己定义的 DEBUG 宏打开了 libmad 的调试，具体做法我们可以再 libmad 源码目录下搜索 DEBUG 关键字，然后将相关内容删除。

至此 libmad 的解码核心移植完毕（呵呵，很简单吧），然后我们移植 madlld-1.1，madlld 是对 libmad 底层 API 的应用实例，通过 madlld 的移植我们就可以把 libmad 解码核心、音频文件流输入、PCM 输出连接起来，**这是整个移植的关键！**

4. madlld 移植

解压源码包内 madlld-1.1p1.tar，把里面 madlld.c 加入到工程并设置好包含路径。（bstdfile.c 是对文件系统的再一次封装，我们这里直接使用 fatfs 的接口函数，所以不使用 bstdfile.c）

我们先来看看 madlld.c 里面 main 函数的内容，了解下大致流程：

```
/*****************************************************************************
 * Program entry point.                                                      *
 *****************************************************************************/
int main(int argc, char *argv[])
{
    char    *cptr;
    int     Status;

    /* Keep this for error messages. */
    cptr=strrchr(argv[0],'/');
    if(cptr==NULL)
        ProgName=argv[0];
    else
        ProgName=cptr+1;

    /* The command-line arguments are analyzed. */
    if(ParseArgs(argc,argv))
        return(1);

    /* Decode stdin to stdout. */
    Status=MpegAudioDecoder(stdin,stdout);
    if(Status)
        fprintf(stderr,"%s: an error occurred during decoding.\n",ProgName);

    /* All done. */
    return(Status);
}
```

main 函数是典型的带操作系统的主函数格式，带入参数个数和各个参数的内容，MpegAudioDecoder 函数内部完成解码的全部内容，它从标准输入设备（stdin）读入数据，解码完后把结果存入标准输出设备，如果产生错误则错误被写入到标准错误输出。

4.1

从 MpegAudioDecoder 函数为突破口大致浏览一下，我们基本上可以确定有些函数是可以直接删除或注释掉的。我们先删除它们：

1、删除 static int ParseArgs(int argc, char * const argv[]);

2、删除 static void PrintUsage(void);

3、删除 static void ApplyFilter(struct mad_frame *Frame);

4.2

接下来，文件包含我们要做如下修改：

```
/***************************************************
 * Includes
 ***************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h> /* for pow() and log10() */
#include "ff.h"
#include "config.h"
#include "mad.h"
#include "stm32f10x.h"
```

4.3

Madlld.c 中有很多打印函数都是用了 fprintf,我们需要把它们全改为从串口输出，方便我们观察输出信息，这个不用我说，相信大家都会改了。

4.4

Libmad解码出来的数据为24bit的PCM数据，需要用MadFixedToSshort函数进行24bit到16bit的转换 madlld.c 里的 MadFixedToSshort)函数体有点问题，需要使用 minimad.c 中 signed int scale(mad_fixed_t sample)的函数体内容替换。

另外 MpegAudioDecoder 函数中，下面地方

```
else
    ReadSize=INPUT_BUFFER_SIZE,
        ReadStart=InputBuffer,
        Remaining=0;
```

else 少了括号，需要加上。

4.5

对于文件的读取操作，我们需要使用 fatfs 的接口函数替换原有的相关语句，madlld.c 里的一些函数的参数类型为 FILE，我们也给改为 fatfs 里的类型—FIL。具体细节这里不一一列出，大家可以自己发挥或者参考例程包里 libmad 移植模板的内容。

4.6

解码用到的三个主要数据结构：

```
struct mad_stream       Stream;
struct mad_frame        Frame;
struct mad_synth        Synth;
```

总共大概会占 20 多 K 的 RAM 空间，我们把其定义从函数体内拉出来，作为全局变量，防止栈空间不够用。

对解码用到的输入、输出缓冲区，我们要做下修改，把 OutputBuffer 和 InputBuffer 定义成全局变量，且 OutputBuffer 定义为 2 维数组。madlld 使用了单个输出缓冲区，而我们要实现边解码边播放，所以要实现双缓冲机制，即：A 缓冲区的数据通过 DMA 播放的同时，B 缓冲区要接收解码后得到的 PCM 数据，A 缓冲区的数据播放完成后，接着播放 B 缓冲区的数据，而这时 A 缓冲区再去存储解码后的 PCM 数据，如此交替。这里存在一个时间问题，假设解码的歌曲是采样率 44.1K 的双声道歌曲，我们的缓冲区大小设为 1152*2，这 1152*2 个数据播放完使用的时间是：(1152)*(1/44.1)=26 ms

这也就意味着解码得到 1152*2 个数据的时间一定要小于 26ms 否则，一个缓冲区的数据都用完了，另一个缓冲区还没准备好数据，播放就会有停顿。

我的缓冲区设置如下：

```
//此值设大 歌曲解码会不断出错，设小了高码率的（320K）才可勉强解码
#define INPUT_BUFFER_SIZE    1028
//不要小于 1052*2   且要能被 4 整除
#define OUTPUT_BUFFER_SIZE   4096

unsigned char InputBuffer[INPUT_BUFFER_SIZE+MAD_BUFFER_GUARD];
unsigned short OutputBuffer[2][OUTPUT_BUFFER_SIZE]; //双缓冲
```

4.7

前面提到，解码后的数据我们转化为 16bit 的格式，然后存入输出缓冲区，这个动 作

是在下面代码完成的：

```
for(i=0;i<Synth.pcm.length;i++)
{   //解码后的每个PCM数据都是24位的，这里转换为16位数据并按照小端格式存入 输出缓冲区
    signed short    Sample;

    /* Left channel */
    Sample=MadFixedToSshort(Synth.pcm.samples[0][i]);
    *(OutputPtr++)=Sample>>8;
    *(OutputPtr++)=Sample&0xff;

    /* Right channel. If the decoded stream is monophonic then
     * the right output channel is the same as the left one.
     */
    if(MAD_NCHANNELS(&Frame.header)==2)
        Sample=MadFixedToSshort(Synth.pcm.samples[1][i]);
    *(OutputPtr++)=Sample>>8;
    *(OutputPtr++)=Sample&0xff;
```

代码中的

*(OutputPtr++)=Sample>>8;

*(OutputPtr++)=Sample&0xff;

是按照大端格式存放数据，我们需要的是小端格式的数据，因为我们定义 OutputBuffer 为 short 型，所以采用下面形式的语句就可以

Sample=MadFixedToSshort(Synth.pcm.samples[0][i]);

*(OutputPtr++)=Sample;

if(MAD_NCHANNELS(&Frame.header)==2)

Sample=MadFixedToSshort(Synth.pcm.samples[1][i]);

*(OutputPtr++)=Sample;

4.8

最终的 PCM 数据我们用 DMA，通过 IIS 接口传给外部 DA 芯片 PCM1770 进行播放。对 IIS 和 PCM1770 的相关操作，本文不做说明。具体的数据播放用下面代码实现：

```
if(OutputPtr==OutputBufferEnd)
{ //缓冲区填满数据了
    while(curr_buff_done==0);   //缓冲区没播放完成等待

    if (curr_buff_done!=0)//
    {    //播放完成后 IIS中断 中 curr_buff_done会被置1
         curr_buff_done=0;

         P_AudioBuff =  &OutputBuffer[use_buffer][0];  //DMA传输地址设置
         if (SampleRateCopy !=  Synth.pcm.samplerate)
               IIS_Config(Synth.pcm.samplerate); //播放第一帧时设置 IIS的采样速率，后续只要启动DMA
        else
               DMA_Transmit((uint32_t)P_AudioBuff, OUTPUT_BUFFER_SIZE);

         SampleRateCopy =  Synth.pcm.samplerate;
    }
     //切换缓冲区
    use_buffer^=1;
    OutputPtr=&OutputBuffer[use_buffer][0];
    OutputBufferEnd=&OutputBuffer[use_buffer][0]+OUTPUT_BUFFER_SIZE;
}
```

4.9

最后，我们把 madlld.c 中的 main 函数修改成下面形式，在主函数中进行调用，并定义一些需要的变量，配置好 DA 芯片和 IIS 接口，根据实际情况查查编译问题，在 SD 卡的根目

录下放上一首歌曲，就可以看 libmad 的解码效果了。（SD 卡请选择小于 4G 的，测试歌曲最好选择码率 128Kbps 以下的 MP3）

```
/*****************************************************************************
 * Program entry point.
 *****************************************************************************/
int main_decode(void)
{   //没用的内容顺手删掉
    int Status=0;

    fres =f_mount(0, &fs);
    fres = f_open (&F, "hello.mp3", FA_READ);

    Status=MpegAudioDecoder(&F);
    if(Status)
        printf("%s: an error occurred during decoding.\n",ProgName);

    return(Status);
}
```

总结

到这里，libmad 的整个移植完成，当成功移植 libmad 后，再去移植 helix，就驾轻就熟了，所以 helix 的移植我就不再写教材了。

附录

下面是 libmad 解码库在 UP PLAYER V2.0（本工作室推出的音频解码开发板）上运行时的串口输出信息：

附上 IIS 和 PCM1770 相关操作代码，作为参考

IIS 的

```
/*******************************************************************************
  * @file    iis.c
  * @author  UP MCU 工作室
  * @version
  * @date
  */
/* Inicludes ------------------------------------------------------------------*/
#include "stm32f10x.h"
#include "pcm1770.h"
#include "iis.h"

vu16 * /*volatile*/ P_AudioBuff;
vu16 curr_buff_done;  //一个缓冲区数据播放完成的标志
/**
  * @brief  IIS_GPIO_Init
  * @param  None
  * @retval None
  */
void IIS_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStructure;
  /* Enable peripheral clocks ------------------------------------------------*/
  /* GPIOB and AFIO clocks enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC, ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);

  /* Configure SPI2 pins: CK, WS and SD ---------------------------------*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
  /* Configure SPI2 MCK --256fs-----------------------------*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
}
/**
  * @brief  IIS_Config
  * @param  None
  * @retval None
  */
 DMA_InitTypeDef    DMA_InitStructure;
```

```
void IIS_Config(uint32_t freq)
{
  I2S_InitTypeDef I2S_InitStructure;
  /* 复位 SPI2 外设到缺省状态 */
  SPI_I2S_DeInit(SPI2);
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);   //DMA1 通道 5
  /* I2S peripheral configuration */
  I2S_InitStructure.I2S_Standard = I2S_Standard_LSB;//I2S_Standard_Phillips;
  I2S_InitStructure.I2S_DataFormat = I2S_DataFormat_16b;// I2S_DataFormat_24b;
  I2S_InitStructure.I2S_MCLKOutput = I2S_MCLKOutput_Enable;
  I2S_InitStructure.I2S_AudioFreq = freq;//I2S_AudioFreq_16k;
  I2S_InitStructure.I2S_CPOL = I2S_CPOL_Low;
  /* I2S2 Master Transmitter  ----------*/
  /* I2S2 configuration */
  I2S_InitStructure.I2S_Mode = I2S_Mode_MasterTx;
  I2S_Init(SPI2, &I2S_InitStructure);
  /* SPI_MASTER_Tx_DMA_Channel configuration ---------------------------------*/
  DMA_DeInit(DMA1_Channel5);
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SPI2_BASE+0x0c;
  DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)P_AudioBuff;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
  DMA_InitStructure.DMA_BufferSize = OUTPUT_BUFFER_SIZE;//BufferSize;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
  DMA_Init(DMA1_Channel5, &DMA_InitStructure);
  /* Enable SPI_MASTER DMA Tx request */
  SPI_I2S_DMACmd(SPI2, SPI_I2S_DMAReq_Tx, ENABLE);
  I2S_Cmd(SPI2, ENABLE);
  DMA_ITConfig(DMA1_Channel5, DMA_IT_TC, ENABLE);
  DMA_Cmd(DMA1_Channel5, ENABLE);
}
void DMA_Transmit(u32 addr, u32 size)
{
 DMA1_Channel5->CCR &= (uint16_t)(~DMA_CCR1_EN);//DMA_Cmd(DMA1_Channel5, DISABLE);

 DMA_InitStructure.DMA_MemoryBaseAddr = addr;
 DMA_InitStructure.DMA_BufferSize = size;
 DMA_Init(DMA1_Channel5, &DMA_InitStructure);
 DMA_Cmd(DMA1_Channel5, ENABLE);
```

```
}
```

PCM1770 的

```
/**
  ******************************************************************************
  * @file    pcm1770.c
  * @author  UP MCU 工作室
  * @version
  * @date
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32f10x.h"
#include "pcm1770.h"
#include "my_init.h"

#define PCM_PD          // (1 <<  14)//PB14        PD 口接到主 ic reset 脚
#define PCM_PD_SET_L     //GPIOB->ODR&=~(PCM_PD)  //GPIOB->ODR = (GPIOB->ODR & ~PCM_PD) | (x ? PCM_PD : 0);
#define PCM_PD_SET_H      //GPIOB->ODR|=(PCM_PD)

#define PCM_CS               (1 <<  1)//PB1
#define PCM_CS_SET_L        GPIOB->ODR&=~(PCM_CS) //GPIOB->ODR = (GPIOB->ODR & ~PCM_CS) | (x ? PCM_CS : 0);
#define PCM_CS_SET_H        GPIOB->ODR|=(PCM_CS)

#define PCM_CLK              (1 <<  11)//PA11
#define PCM_CLK_SET_L        GPIOA->ODR&=~(PCM_CLK)//     GPIOB->ODR = (GPIOB->ODR & ~PCM_CLK) | (x ?
PCM_CLK : 0);
#define PCM_CLK_SET_H           GPIOA->ODR|=(PCM_CLK)
#define PCM_DAT              (1 <<  12)//PA12
#define PCM_DAT_SET_L        GPIOA->ODR&=~(PCM_DAT)    //GPIOB->ODR = (GPIOB->ODR & ~PCM_DAT) | (x ?
PCM_DAT : 0);
#define PCM_DAT_SET_H           GPIOA->ODR|=(PCM_DAT)

#define C_VOLUME_MAX 0x3F

static vu8 s_Volume;//音量大小
static void Delay(u32 Num);
/**
  * @brief  PCM1770 Init
  * @param  None
  * @retval None
  */
void PCM1770Init(void)
```

```
{
    GPIO_InitTypeDef GPIO_InitStructure;
    //使能控制信号的时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11 | GPIO_Pin_12;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    //硬件复位一下
    PCM_PD_SET_L;
   Delay(100);//
    PCM_PD_SET_H;
    Delay(100);     //

    PCM_CS_SET_H ;
    PCM_CLK_SET_H;
    PCM_DAT_SET_H;
    s_Volume = C_VOLUME_MAX-2;//默认一半音量
    PCM_WriteData(0x01, s_Volume); //右耳机音量设为中间大小
    PCM_WriteData(0x02, s_Volume); //右耳机音量设为中间大小
    PCM_WriteData(0x03, 0x84);     //256fs IIS 格式  stm32 的 iis mclk 规定为 256fs ( 84= 16 right)
    PCM_WriteData(0x04, 0x00);     //
}
/**
  * @brief
  * @param  Reg Index,  Data
  * @retval None
  */
void PCM_WriteData(const u8 Reg, const u8 Data)
{
    vu16 TrasferData, i;
    TrasferData = Data;
    TrasferData |= (Reg<<8)&0xff00;

    PCM_CS_SET_L;//select
    Delay(10);
    for (i = 0; i < 16; i++)
    {//传输时 MSB first
        PCM_CLK_SET_L;
        if (TrasferData&(0x8000>>i))
        {
```

```
                PCM_DAT_SET_H;
            }
            else
            {
                PCM_DAT_SET_L;
            }
            Delay(10);//等数据稳定
            PCM_CLK_SET_H;  //上升沿写入
            Delay(10);//等待从机读数据
        }
        PCM_CLK_SET_H;
        PCM_DAT_SET_H;
        PCM_CS_SET_H;//relase
        Delay(10);
}
/**
  * @brief  Volume_Dec
  * @param  None
  * @retval None
  */
void Volume_Dec(void)
{
        if (s_Volume > 0)
        {
            s_Volume--;
            PCM_WriteData(0x01, s_Volume);//左右耳机静音  ，左耳机音量设为中间大小
            PCM_WriteData(0x02, s_Volume);//右耳机音量设为中间大小
        }
}

/**
  * @brief  Volume_Add
  * @param  None
  * @retval None
  */
void Volume_Add(void)
{
        if (s_Volume < C_VOLUME_MAX)
        {
            s_Volume++;
            PCM_WriteData(0x01, s_Volume);//左右耳机静音  ，左耳机音量设为中间大小
            PCM_WriteData(0x02, s_Volume);//右耳机音量设为中间大小
        }
}
```

```
/**
  * @brief  Volume_Add
  * @param  None
  * @retval None
  */
void PCM1770_VolumeSet(vu8 vol)
{
          s_Volume = vol;
          PCM_WriteData(0x01, s_Volume);//左右耳机静音  ，左耳机音量设为中间大小
          PCM_WriteData(0x02, s_Volume);//右耳机音量设为中间大小
}

/**
  * @brief  Volume_Add
  * @param  None
  * @retval None
  */
void PCM1770_Mute(void)
{
          PCM_WriteData(0x01, 0xc0);//左右耳机静音  ，左耳机音量设为中间大小
          PCM_WriteData(0x02, 0x00);//右耳机音量设为中间大小
}
/**
  * @brief  GetCurrentVolume
  * @param  None
  * @retval Volume
  */
u8 GetCurrentVolume(void)
{
     return    s_Volume;
}
/**
  * @brief  Delay
  * @param  Delay Num
  * @retval None
  */
void Delay(u32 Num)
{
     vu32 Count = Num*4;

     while (--Count);
}
```