

原 FFMpeg源代码简单分析：configure

2015年03月24日 10:22:53 阅读数：32197

=====

FFmpeg的库函数源代码分析文章列表：

【架构图】

[FFmpeg 源代码结构图 - 解码](#)

[FFmpeg 源代码结构图 - 编码](#)

【通用】

[FFmpeg 源代码简单分析：av_register_all\(\)](#)

[FFmpeg 源代码简单分析：avcodec_register_all\(\)](#)

[FFmpeg 源代码简单分析：内存的分配和释放（av_malloc\(\)、av_free\(\)等）](#)

[FFmpeg 源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）](#)

[FFmpeg 源代码简单分析：avio_open2\(\)](#)

[FFmpeg 源代码简单分析：av_find_decoder\(\)和av_find_encoder\(\)](#)

[FFmpeg 源代码简单分析：avcodec_open2\(\)](#)

[FFmpeg 源代码简单分析：avcodec_close\(\)](#)

【解码】

[图解 FFMPEG 打开媒体的函数 avformat_open_input](#)

[FFmpeg 源代码简单分析：avformat_open_input\(\)](#)

[FFmpeg 源代码简单分析：avformat_find_stream_info\(\)](#)

[FFmpeg 源代码简单分析：av_read_frame\(\)](#)

[FFmpeg 源代码简单分析：avcodec_decode_video2\(\)](#)

[FFmpeg 源代码简单分析：avformat_close_input\(\)](#)

【编码】

[FFmpeg 源代码简单分析：avformat_alloc_output_context2\(\)](#)

[FFmpeg 源代码简单分析：avformat_write_header\(\)](#)

[FFmpeg 源代码简单分析：avcodec_encode_video\(\)](#)

[FFmpeg 源代码简单分析：av_write_frame\(\)](#)

[FFmpeg 源代码简单分析：av_write_trailer\(\)](#)

【其它】

[FFmpeg 源代码简单分析：日志输出系统（av_log\(\)等）](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVClass](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVOption](#)

[FFmpeg 源代码简单分析：libswscale 的 sws_getContext\(\)](#)

[FFmpeg 源代码简单分析：libswscale 的 sws_scale\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 avdevice_register_all\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 gdigrab](#)

【脚本】

FFmpeg 源代码简单分析：makefile

FFmpeg 源代码简单分析：configure

【H.264】

FFmpeg 的 H.264 解码器源代码简单分析：概述

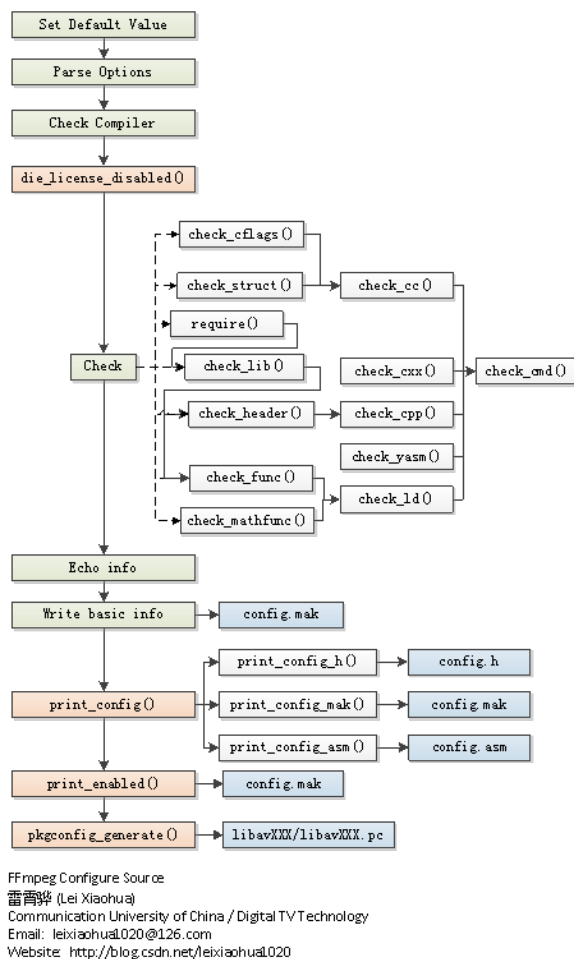
本文记录FFmpeg的Configure脚本的源代码。Configure一方面用于检测FFmpeg的编译环境，另一方面根据用户配置的选项生成config.mak, config.h文件（可能还有config.asm），提供给Makefile使用。由于FFmpeg的configure脚本很复杂（一个4000-5000行的Shell脚本），难以逐行细致的分析，因此本文简单梳理一下它的结构。

PS1：Configure的日志位于config.log文件中。查看该文件有助于分析Configure的过程。

PS2：使用“sh -x script_name.sh”可以调试Shell脚本。

Configure文件的整体流程

Configure文件的整体流程如下所示。



Configure的整体流程可以分成以下几步：

Set Default Value：设置各个变量默认值；

Parse Options：解析输入的选项；

Check Compiler：检查编译器；

die_license_disabled()：检查GPL等协议的设置情况；

Check：检查编译环境（数学函数，第三方类库等）；

Echo info：控制台上打印配置信息；

Write basic info：向config.mak中写入一些基本信息；

print_config()：向config.h、config.mak、config.asm中写入所有配置信息；

print_enabled()：向config.mak写入所有enabled的组件信息；

pkgconfig_generate()：向libavXXX/libavXXX.pc中写入pkgconfig信息（XXX代表avcodec，avformat等）；

下文简单梳理一下这些步骤。

Set Default Value

Set Default Value部分设置一些Configure的默认值。例如下面的代码。

```
[python]
1. # 默认参数 default parameters
2. # 日志
3. logfile="config.log"
4.
5. # 安装路径 installation paths
6. prefix_default="/usr/local"
7. bindir_default='${prefix}/bin'
8. datadir_default='${prefix}/share/ffmpeg'
9. incdir_default='${prefix}/include'
10. libdir_default='${prefix}/lib'
11. mandir_default='${prefix}/share/man'
12. shlibdir_default="shlibdir_default"
13. postproc_version_default="current"
14.
15. # 工具链 toolchain
16. ar_default="ar"
17. cc_default="gcc"
18. cxx_default="g++"
19. cc_version="\unknown\"
20. host_cc_default="gcc"
21. install="install"
22. ln_s="ln -sf"
23. nm_default="nm"
24. objformat="elf"
25. pkg_config_default=pkg-config
26. ranlib="ranlib"
27. strip_default="strip"
28. yasmexe_default="yasm"
29.
30. nm_opts='-g'
31. nogas=":"
32.
33. # 机器 machine
34. arch_default=$(uname -m)
35. cpu="generic"
36.
37. # 操作系统 OS
38. target_os_default=$(tolower $(uname -s))
39. host_os=$target_os_default
40.
41. # alternative libpostproc version
42. ALT_PP_VER_MAJOR=51
43. ALT_PP_VER_MINOR=2
44. ALT_PP_VER_MICRO=101
45. ALT_PP_VER=$ALT_PP_VER_MAJOR.$ALT_PP_VER_MINOR.$ALT_PP_VER_MICRO
46.
47. # 选项 configurable options
48. # PROGRAM_LIST内容是 ffplay ffprobe ffserver ffmpeg
49. enable $PROGRAM_LIST
50.
51. enable avcodec
52. enable avdevice
53. enable avfilter
54. enable avformat
55. enable avutil
56. enable postproc
57. enable stripping
58. enable swresample
59. enable swscale
60.
61. enable asm
62. enable debug
63. enable doc
64. enable fastdiv
65. enable network
66. enable optimizations
67. enable safe_bitstream_reader
68. enable static
69. enable swscale_alpha
70.
71. # 编译选项 build settings
72. SHELL_ARGS="-shared -Wl,-soname,$$(/AF)"
```

```

72.  SH_LIBS= "${SH_LIB} -lc, -l${NAME} ${LIB} "
73.  FFSERVERLDFLAGS=-Wl, -E
74.  # 前缀后缀
75.  LIBPREF="lib"
76.  LIBSUF=".a"
77.  FULLNAME='$(NAME)$(BUILDSUF)'
78.  # 名称
79.  LIBNAME='$(LIBPREF)$(FULLNAME)$(LIBSUF)'
80.  # 动态库前缀后缀
81.  SLIBPREF="lib"
82.  SLIBSUF=".so"
83.  # 名称
84.  SLIBNAME='$(SLIBPREF)$(FULLNAME)$(SLIBSUF)'
85.  SLIBNAME_WITH_VERSION='$(SLIBNAME).$(LIBVERSION)'
86.  SLIBNAME_WITH_MAJOR='$(SLIBNAME).$(LIBMAJOR)'
87.  LIB_INSTALL_EXTRA_CMD='$$RANLIB "$$(LIBDIR)/$(LIBNAME) "'
88.  SLIB_INSTALL_NAME='$(SLIBNAME_WITH_VERSION)'
89.  SLIB_INSTALL_LINKS='$(SLIBNAME_WITH_MAJOR) $(SLIBNAME)'
90.
91.  AS_0='-o $@"
92.  CC_0='-o $@"
93.  CXX_0='-o $@"
94.
95.  host_cflags='-D_ISOC99_SOURCE -O3 -g'
96.  host_libs='-lm'
97.
98.  target_path='$(CURDIR)'

```

需要注意的是，“enable avcodec”，“enable avformat”，“enable avutil”等中的enable()本身是一个函数。enable()的定义如下。

```

[python]
1.  #把所有输入参数的值设置为"yes"
2.  enable(){
3.      set_all yes $*
4.  }

```

可以看出enable()调用了set_all()函数。并且将第1个参数设置为“yes”，并且将调用enable()时候的参数传递给set_all()。set_all()函数的定义如下所示。

```

[python]
1.  #第一个参数为值，后面的参数为变量
2.  set_all(){
3.      value=$1
4.      shift
5.      for var in $*; do
6.          eval $var=$value
7.      done
8.  }

```

可以看出set_all()将传入的参数全部进行赋值。特定于enable()函数来说，就是将所有的输入变量赋值为“yes”。由此可见，“enable avcodec”实际上相当于执行了：

```

[python]
1.  avcodec="yes"

```

Parse Options

Parse Options部分用于解析Configure的附加参数。该部分的代码如下所示。

```
[python]
1. #注意：opt不是参数列表（实际上也没有看见opt变量的定义）
2. #原因是处在for循环中，当你没有为in指定列表时，for会默认取命令行参数列表。
3. #因此“opt”这个名字实际上是可以随便取的
4. for opt do
5. # “#”用于去除特定字符前面的字符串
6. # optval内容为opt去掉“=”以及其前面字符串之后的内容
7. optval="{opt#*=}"
8. case "$opt" in
9. # 不同的选项
10. --extra-ldflags=*) add_ldflags $optval
11. ;;
12. --extra-libs=*) add_extralibs $optval
13. ;;
14. --disable-devices) disable $INDEV_LIST $OUTDEV_LIST
15. ;;
16. --enable-debug=*) debuglevel="$optval"
17. ;;
18. --disable-everything)
19. map 'eval unset \${$(toupper ${v%s})}_LIST)' $COMPONENT_LIST
20. ;;
21. --enable-*=*|--disable-*=*)
22. eval $(echo "${opt%*=}" | sed 's/--/action=/;s/-/ thing=/' )
23. is_in "${thing}s" $COMPONENT_LIST || die_unknown "$opt"
24. eval list=\${$(toupper $thing)_LIST}
25. name=$(echo "${optval}" | sed 's/,/_/${thing}|g')_${thing}
26. $action $(filter "$name" $list)
27. ;;
28. --enable-?|--disable-?*)
29. eval $(echo "$opt" | sed 's/--/action=/;s/-/ option=/;s/_/_/g')
30. if is_in $option $COMPONENT_LIST; then
31. test $action = disable && action=unset
32. eval $action \${$(toupper ${option%s})}_LIST
33. elif is_in $option $CMDLINE_SELECT; then
34. $action $option
35. else
36. die_unknown $opt
37. fi
38. ;;
39. --list-*)
40. NAME="{opt--list-}"
41. is_in $NAME $COMPONENT_LIST || die_unknown $opt
42. NAME=${NAME%s}
43. eval show_list $NAME \${$(toupper $NAME)_LIST}
44. ;;
45. --help|-h) show_help
46. ;;
47. *)
48. #% 就是从右边开始删除符合条件的字符串（符合条件的最短字符串）
49. #%%是删除符合条件的最长的字符串
50.
51. #删除“=”右边的内容
52. optname="{opt%*=}"
53. #删除左边的“--”
54. optname="{optname#--}"
55. optname=$(echo "$optname" | sed 's/_/_/g')
56. #看看是否在opt列表中，不在的话就会返回错误
57. if is_in $optname $CMDLINE_SET; then
58. eval $optname="{optval}"
59. elif is_in $optname $CMDLINE_APPEND; then
60. append $optname "$optval"
61. else
62. die_unknown $opt
63. fi
64. ;;
65. esac
66. done
```

在这里需要注意，取出opt的值一般都是“--extra-ldflags=XXX”的形式，通过“\${opt#*=}”截取获得“=”号后面的内容作为optval，对于“--extra-ldflags=XXX”来说，optval取值为“XXX”。

然后根据opt种类的不同，以及optval取值的不同，分别作不同的处理。

Check Compiler

Check Compiler用于检查编译器。这部分代码还没有细看，暂时不做分析。

die_license_disabled()

die_license_disabled()用于检查是否指定了特定License。像libx264、libfaac这些第三方类库，都需要指定特定的License才可以使用（例如libfaac必须指定nonfree）。开启这些第三方类库后如果没有指定License，Configure会立刻退出。这部分代码如下所示。

```
[python]
1. #检查License
2. #GPL
3. die_license_disabled gpl libcdio
4. die_license_disabled gpl libx264
5. die_license_disabled gpl libxavs
6. die_license_disabled gpl libxvid
7. die_license_disabled gpl x11grab
8. #nonfree
9. die_license_disabled nonfree libaacplus
10. die_license_disabled nonfree libfaac
11. die_license_disabled nonfree openssl
12. #Version3
13. die_license_disabled version3 libopencore_amrnb
14. die_license_disabled version3 libopencore_amrwb
15. die_license_disabled version3 libvo_aacenc
16. die_license_disabled version3 libvo_amrwbenc
```

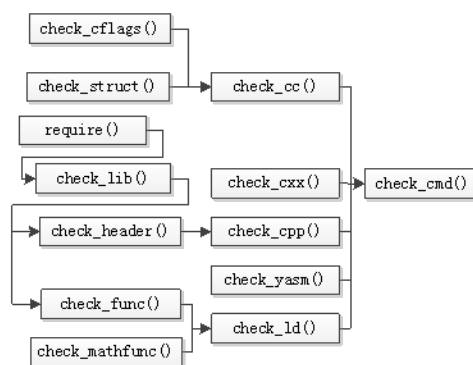
其中涉及到一个函数die_license_disabled(), 它的定义如下所示。

```
[python]
1. #不符合License则立刻结束
2. die_license_disabled() {
3.     enabled $1 || { enabled $2 && die "$2 is $1 and --enable-$1 is not specified."; }
4. }
```

从定义可以看出, die_license_disabled()首先会看第1个参数(对应“gpl”, “nonfree”)对应的组件是否enable, 如果已经enable, 则正常运行完函数; 如果没有enable, 则会检查第2个参数(对应“libx264”, “libfaac”)是否enable, 如果第2个参数enable了, 就会报错退出。

Check

Check部分是Configure中最重要的部分。该部分用于检查编译环境(例如数学函数, 第三方类等)。这一部分涉及到很多的函数。包括check_cflags()、check_struct()、require()、check_lib()、check_header()、check_func()、check_mathfunc()等等。这些函数之间的调用关系如下图所示。



下面简单举例下面几个函数：

check_func()：用于检查函数。

check_header()：用于检查头文件。

check_func_headers()：用于同时检查头文件和函数。

check_mathfunc()：用于检查数学类函数。

require()：检查第三方类库。

check_cflags()：用于检查编译器的cflags标志参数。

下面详细看看这几个函数。

check_func()

check_func()用于检查函数。它的输入参数一个函数名。Configure中与check_func()有关的代码如下所示。

```
[python]
1. check_func isatty
2. check_func localtime_r
3. check_func ${malloc_prefix}memalign      && enable memalign
4. check_func mkstemp
5. check_func mmap
6. check_func ${malloc_prefix}posix_memalign  && enable posix_memalign
7. check_func setrlimit
8. check_func strerror_r
9. check_func strptime
10. check_func sched_getaffinity
11. check_func sysconf
12. check_func sysctl
```

check_func()的定义如下所示。

```
[python]
1. check_func(){
2.     log check_func "$@"
3.     func=$1
4.     shift
5.     disable $func
6.     check_ld "cc" "$@" <<EOF && enable $func
7. extern int $func();
8. int main(void){ $func(); }
9. EOF
10. }
```

从check_func()的定义可以看出，该函数首先将输入的第1个参数赋值给func，然后生成一个下述内容的C语言文件。

```
[python]
1. extern int $func();
2. int main(void){ $func(); }
```

最后调用check_ld()完成编译测试。check_ld()的定义如下所示。

```
[python]
1. check_ld(){
2.     log check_ld "$@"
3.     type=$1
4.     shift 1
5.     flags=''
6.     libs=''
7.     for f; do
8.         test "${f}" = "${f#-l}" && flags="$flags $f" || libs="$libs $f"
9.     done
10.    check_type $(filter_cflags $flags) || return
11.    check_cmd $ld $LDFLAGS $flags -o $TMPE $TMP0 $libs $extralibs
12. }
```

其中check_cmd()是个很简单的函数，可以输出日志，如下所示。

```
[python]
1. check_cmd(){
2.     log "$@"
3.     "$@" >> $logfile 2>&1
4. }
```

例如，“check_func mkstemp”相当于编译了下述代码。

```
[python]
1. extern int mkstemp();
2. int main(void){ mkstemp(); }
```

check_header()

check_header()用于检查头文件。Configure中与check_header()有关的代码如下所示。

```
[python]
1. check_header dlfcn.h
2. check_header dxva2api.h -D_WIN32_WINNT=0x0600
3. check_header libcrystalhd/libcrystalhd_if.h
4. check_header malloc.h
5. check_header poll.h
6. check_header sys/mman.h
7. check_header sys/param.h
8. check_header sys/resource.h
9. check_header sys/select.h
10. check_header termios.h
11. check_header vdpau/vdpau.h
12. check_header vdpau/vdpau_x11.h
13. check_header X11/extensions/XvMClib.h
14. check_header asm/types.h
```

check_header()的定义如下所示。

```
[python]
1. check_header(){
2.     log check_header "$@"
3.     header=$1
4.     shift
5.     disable_safe $header
6.     check_cpp "$@" <<EOF && enable_safe $header
7.     #include <$header>
8.     int x;
9.     EOF
10. }
```

从check_header()的定义可以看出，该函数首先将输入的第1个参数赋值给header，然后生成一个下述内容的C语言文件。

```
[python]
1. #include <$header>
2. int x;
```

最后调用check_cpp()完成编译测试。check_cpp()的定义如下所示。

```
[python]
1. check_cpp(){
2.     log check_cpp "$@"
3.     cat > $TMPC
4.     log_file $TMPC
5.     #-E选项，可以让编译器在预处理后停止，并输出预处理结果。
6.     check_cmd $cc $CPPFLAGS $CFLAGS "$@" -E -o $TMPD $TMPC
7. }
```

例如，“check_header malloc.h”相当于处理以下C语言文件。

```
[python]
1. #include <malloc.h>
2. int x;
```

check_func_headers()

check_func_headers()用于同时检查头文件和函数。Configure中与check_header()有关的代码如下所示。

```
[python]
1. check_func_headers conio.h kbhit
2. check_func_headers windows.h PeekNamedPipe
3. check_func_headers io.h setmode
4. check_func_headers lzo/lzo1x.h lzo1x_999_compress
5. check_func_headers windows.h GetProcessAffinityMask
6. check_func_headers windows.h GetProcessTimes
7. check_func_headers windows.h MapViewOfFile
8. check_func_headers windows.h VirtualAlloc
```

check_func_headers()的定义如下所示。


```
[python]
1. check_func_headers(){
2.     log check_func_headers "$@"
3.     headers=$1
4.     funcs=$2
5.     shift 2
6.     {
7.         for hdr in $headers; do
8.             echo "#include <$hdr>"
9.         done
10.        for func in $funcs; do
11.            echo "long check_$func(void) { return (long) $func; }"
12.        done
13.        echo "int main(void) { return 0; }"
14.    } | check_ld "cc" "$@" && enable $funcs && enable_safe $headers
15. }
```

从check_func_headers()的定义可以看出，该函数首先将输入的第1个参数赋值给header，第2个参数赋值给funcs，然后生成一个下述内容的C语言文件。

```
[python]
1. #include <$hdr>
2. long check_$func (void) { return (long) $func; }
3. int main(void) { return 0; }
```

例如，“check_func_headers windows.h PeekNamedPipe”相当于处理以下C语言文件。

```
[python]
1. #include <windows.h>
2. long check_PeekNamedPipe (void) { return (long) PeekNamedPipe; }
3. int main(void) { return 0; }
```

check_mathfunc()

check_mathfunc()用于检查数学类函数。Configure中与check_mathfunc()有关的代码如下所示。

```
[python]
1. check_mathfunc cbrtf
2. check_mathfunc exp2
3. check_mathfunc exp2f
4. check_mathfunc llrint
5. check_mathfunc llrintf
6. check_mathfunc log2
7. check_mathfunc log2f
8. check_mathfunc lrint
9. check_mathfunc lrintf
10. check_mathfunc round
11. check_mathfunc roundf
12. check_mathfunc trunc
13. check_mathfunc truncf
```

check_mathfunc()的定义如下所示。

```
[python]
1. check_mathfunc(){
2.     log check_mathfunc "$@"
3.     #数学函数名称
4.     func=$1
5.     shift
6.     disable $func
7.     check_ld "cc" "$@" <<EOF && enable $func
8.     #include <math.h>
9.     float foo(float f) { return $func(f); }
10.    int main(void){ return (int) foo; }
11.    EOF
12. }
```

从check_mathfunc()的定义可以看出，该函数首先将输入的第1个参数赋值给func，然后生成一个下述内容的C语言文件。

```
[python]
1. #include <math.h>
2. float foo(float f) { return $func(f); }
3. int main(void){ return (int) foo; }
```

最后调用check_ld()完成编译测试。

例如，“check_mathfunc exp2”相当于编译连接了下面这个C文件。

```
[python]
1. #include <math.h>
2. float foo(float f) { return exp2(f); }
3. int main(void){ return (int) foo; }
```

require()



require()用于检查第三方类库。Configure中与require()有关的代码如下所示。

```
[python]
1. #检查第三方类库
2. # these are off by default, so fail if requested and not available
3. #require()函数参数的规范：(名称, 头文件, 函数名, 附加选项)
4. #require2()函数参数规范类似
5. enabled avisynth && require2 vfw32 "windows.h vfw.h" AVIFileInit -lavifil32
6. enabled frei0r && { check_header frei0r.h || die "ERROR: frei0r.h header not found"; }
7. enabled gnutls && require_pkg_config gnutls gnutls/gnutls.h gnutls_global_init
8. enabled libaacplus && require "libaacplus >= 2.0.0" aacplus.h aacplusEncOpen -laacplus
9. enabled libass && require_pkg_config libass ass/ass.h ass_library_init
10. enabled libcelt && require libcelt celt/celt.h celt_decode -lcelt0 &&
11.     { check_lib celt/celt.h celt_decoder_create_custom -lcelt0 ||
12.       die "ERROR: libcelt version must be >= 0.11.0."; }
13. enabled libdc1394 && require_pkg_config libdc1394-2 dc1394/dc1394.h dc1394_new
14. enabled libdirac && require_pkg_config dirac \
15.     "libdirac_decoder/dirac_parser.h libdirac_encoder/dirac_encoder.h" \
16.     "dirac_decoder_init dirac_encoder_init"
17. #测试libfaac
18. enabled libfaac && require2 libfaac "stdint.h faac.h" faacEncGetVersion -lfaac
19. enabled libfreetype && require_pkg_config freetype2 "ft2build.h freetype/freetype.h" FT_Init_FreeType
20. enabled libgsm && require libgsm gsm/gsm.h gsm_create -lgsm
21. enabled libmodplug && require libmodplug libmodplug/modplug.h ModPlug_Load -lmodplug
22. enabled libmp3lame && require "libmp3lame >= 3.98.3" lame/lame.h lame_set_VBR_quality -lmp3lame
23. enabled libnut && require libnut libnut.h nut_demuxer_init -lnut
24. enabled libopencore_amrnb && require libopencore_amrnb opencore-amrnb/interf_dec.h Decoder_Interface_init -lopencore-amrnb
25. enabled libopencore_amrwb && require libopencore_amrwb opencore-amrwb/dec_if.h D_IF_init -lopencore-amrwb
26. enabled libopencv && require_pkg_config opencv opencv/cvcore.h cvCreateImageHeader
27. enabled libopenjpeg && require libopenjpeg openjpeg.h opj_version -lopenjpeg
28. enabled libpulse && require_pkg_config libpulse-simple pulse/simple.h pa_simple_new
29. enabled librtmp && require_pkg_config librtmp librtmp/rtmp.h RTMP_Socket
30. enabled libschrödinger && require_pkg_config schroedinger-1.0 schroedinger/schro.h schro_init
31. enabled libspeex && require libspeex speex/speex.h speex_decoder_init -lspeex
32. enabled libstagefright_h264 && require_cpp libstagefright_h264 "binder/ProcessState.h media/stagefright/MetaData.h
33.     media/stagefright/MediaBufferGroup.h media/stagefright/MediaDebug.h media/stagefright/MediaDefs.h
34.     media/stagefright/OMXClient.h media/stagefright/OMXCodec.h" android::OMXClient -lstagefright -lmedia -lutils -lbinder
35. enabled libtheora && require libtheora theora/theoraenc.h th_info_init -ltheoraenc -ltheoradec -logg
36. enabled libutvideo && require_cpp utvideo "stdint.h stdlib.h utvideo/utvideo.h utvideo/Codec.h" 'CCodec*' -lutvideo -lstdc++
37. enabled libv4l2 && require_pkg_config libv4l2 libv4l2.h v4l2_ioctl
38. enabled libvo_aacenc && require libvo_aacenc vo-aacenc/voAAC.h voGetAACEncAPI -lvo-aacenc
39. enabled libvo_amrwbenc && require libvo_amrwbenc vo-amrwbenc/enc_if.h E_IF_init -lvo-amrwbenc
40. enabled libvorbis && require libvorbis vorbis/vorbisenc.h vorbis_info_init -lvorbisenc -lvorbis -logg
41. enabled libvpx && {
42.     enabled libvpx_decoder && { check_lib2 "vpx/vpx_decoder.h vpx/vp8dx.h" vpx_codec_dec_init_ver -lvpx ||
43.       die "ERROR: libvpx decoder version must be >=0.9.1"; }
44.     enabled libvpx_encoder && { check_lib2 "vpx/vpx_encoder.h vpx/vp8cx.h" "vpx_codec_enc_init_ver VPX_C0" -lvpx ||
45.       die "ERROR: libvpx encoder version must be >=0.9.6"; } }
46. #测试libx264
47. enabled libx264 && require libx264 x264.h x264_encoder_encode -lx264 &&
48.     { check_cpp_condition x264.h "X264_BUILD >= 118" ||
49.       die "ERROR: libx264 version must be >= 0.118."; }
50. enabled libxavs && require libxavs xavs.h xavs_encoder_encode -lxavs
51. enabled libxvid && require libxvid xvid.h xvid_global -lxvidcore
52. enabled openal && { { for al_libs in "${OPENAL_LIBS}" "-lopenal" "-lOpenAL32"; do
53.     check_lib 'AL/al.h' alGetError "${al_libs}" && break; done } ||
54.     die "ERROR: openal not found"; } &&
55.     { check_cpp_condition "AL/al.h" "defined(AL_VERSION_1_1)" ||
56.       die "ERROR: openal version must be 1.1 or compatible"; }
57. enabled mlib && require mediaLib mlib_types.h mlib_VectorSub_S16_U8_Mod -lmlib
58. enabled openssl && { check_lib openssl/ssl.h SSL_library_init -lssl -lcrypto ||
59.     check_lib openssl/ssl.h SSL_library_init -lssl32 -leay32 ||
60.     check_lib openssl/ssl.h SSL_library_init -lssl -lcrypto -lws_32 -lgdi32 ||
61.     die "ERROR: openssl not found"; }
```



由于上述代码量比较大。在这里我们只选择一个典型的例子——libx264来看一下。require()检测libx264的定义如下所示。

```
[python]
1. #测试libx264
2. require libx264 x264.h x264_encoder_encode -lx264 &&
3.     { check_cpp_condition x264.h "X264_BUILD >= 118" ||
4.       die "ERROR: libx264 version must be >= 0.118."; }
```



从测试libx264的代码可以看出，require()函数的使用方式为：

```
[plain]    
1. require {名称} {头文件} {函数名} {附加选项}
```

require()函数定义如下所示。

```
[python]    
1. #检查依赖项的时候使用  
2. require(){  
3.     name="$1"  
4.     header="$2"  
5.     func="$3"  
6.     shift 3  
7.     check_lib $header $func "$@" || die "ERROR: $name not found"  
8. }
```

从require()的定义可以看出，该函数将第1个参数赋值给name，第2个参数赋值给header，第3个参数赋值给func。最后调用check_lib()函数。check_lib()的定义如下所示。



```
[python]    
1. #检查类库  
2. check_lib(){  
3.     log check_lib "$@"  
4.     header="$1"  
5.     func="$2"  
6.     shift 2  
7.     check_header $header && check_func $func "$@" && add_extralibs "$@"  
8. }
```

可以看出check_lib()调用了check_header()、check_func()等几个函数完成检查工作。这两个函数在前文中已经介绍过，就不再重复了。例如检测libx264的时候调用check_header()会生成以下临时文件：



```
[python]    
1. #include <x264.h>  
2. int x;
```

check_cflags ()

check_cflags()用于检查编译器的cflags标志参数。Configure中与check_cflags()有关的代码如下所示。

```
[python]    
1. #添加一些编译选项  
2. # add some useful compiler flags if supported  
3. check_cflags -Wdeclaration-after-statement  
4. check_cflags -Wall  
5. check_cflags -Wno-parentheses  
6. check_cflags -Wno-switch  
7. check_cflags -Wno-format-zero-length  
8. check_cflags -Wdisabled-optimization  
9. check_cflags -Wpointer-arith  
10. check_cflags -Wredundant-decls  
11. check_cflags -Wno-pointer-sign  
12. check_cflags -Wcast-qual  
13. check_cflags -Wwrite-strings  
14. check_cflags -Wtype-limits  
15. check_cflags -Wundef  
16. check_cflags -Wmissing-prototypes  
17. check_cflags -Wno-pointer-to-int-cast  
18. check_cflags -Wstrict-prototypes  
19. check_cflags()函数的定义如下所示。  
20. check_cflags(){  
21.     log check_cflags "$@"  
22.     set -- $($filter_cflags "$@")  
23.     check_cc "$@" <<EOF && append CFLAGS "$@"  
24. int x;  
25. EOF  
26. }
```

从定义可以看出，check_cflags()调用了check_cc()执行命令。整个代码只有一行：

```
[cpp]    
1. int x;
```

check_cc()的定义如下所示。

```
[python]
1. check_cc(){
2.     log check_cc "$@"
3.     cat > $TMPC
4.     log_file $TMPC
5.     #很多检查都调用了这个check_cmd
6.     #-c 只编译不链接
7.     check_cmd $cc $CPPFLAGS $CFLAGS "$@" -c -o $TMP0 $TMPC
8. }
```

可以看出check_cflags()将输入参数设置到命令行中，并最终调用了check_cmd。
除了上述几个函数之外，还有其他的一些检查编译环境的函数，在这里就不一一列举了。

Echo info

Echo info用于在控制台上打印配置信息。Configure中该部分的代码如下所示。

```
[python]
1. #在控制台输出信息
2. echo "install prefix          $prefix"
3. echo "source path             $source_path"
4. echo "C compiler                $cc"
5. echo "ARCH                      $arch ($cpu)"
6. if test "$build_suffix" != ""; then
7.     echo "build suffix              $build_suffix"
8. fi
9. if test "$progs_suffix" != ""; then
10.    echo "progs suffix              $progs_suffix"
11. fi
12. if test "$extra_version" != ""; then
13.    echo "version string suffix      $extra_version"
14. fi
15. #${}的特异功能：
16. #${file-my.file.txt}假如 $file 为空值，则使用 my.file.txt 作默认值。(保留没设定及非空值)
17. #在这里，如果某个变量为空值，则取默认值为no
18. echo "big-endian                ${bigendian-no}"
19. echo "runtime cpu detection      ${runtime_cpudetect-no}"
20. if enabled x86; then
21.     echo "${yasmexe}          ${yasm-no}"
22.     echo "MMX enabled                ${mmx-no}"
23.     echo "MMX2 enabled                ${mmx2-no}"
24.     echo "3DNow! enabled              ${amd3dnow-no}"
25.     echo "3DNow! extended enabled     ${amd3dnowext-no}"
26.     echo "SSE enabled                 ${sse-no}"
27.     echo "SSSE3 enabled               ${sse3-no}"
28.     echo "AVX enabled                 ${avx-no}"
29.     echo "CMOV enabled                ${cmov-no}"
30.     echo "CMOV is fast                 ${fast_cmov-no}"
31.     echo "EBX available               ${ebx_available-no}"
32.     echo "EBP available               ${ebp_available-no}"
33. fi
34. if enabled arm; then
35.     echo "ARMv5TE enabled             ${armv5te-no}"
36.     echo "ARMv6 enabled               ${armv6-no}"
37.     echo "ARMv6T2 enabled             ${armv6t2-no}"
38.     echo "ARM VFP enabled             ${armvfp-no}"
39.     echo "IWMX enabled                ${iwmxt-no}"
40.     echo "NEON enabled                ${neon-no}"
41. fi
42. if enabled mips; then
43.     echo "MMI enabled                 ${mmi-no}"
44. fi
45. if enabled ppc; then
46.     echo "Altivec enabled             ${altivec-no}"
47.     echo "PPC 4xx optimizations       ${ppc4xx-no}"
48.     echo "dcbzl available              ${dcbzl-no}"
49. fi
50. if enabled sparc; then
51.     echo "VIS enabled                 ${vis-no}"
52. fi
53. echo "debug symbols               ${debug-no}"
54. echo "strip symbols                ${stripping-no}"
55. echo "optimize for size           ${small-no}"
56. echo "optimizations                ${optimizations-no}"
57. echo "static                       ${static-no}"
58. echo "shared                       ${shared-no}"
59. echo "postprocessing support       ${postproc-no}"
60. echo "new filter support           ${avfilter-no}"
61. echo "network support              ${network-no}"
62. echo "threading support            ${thread_type-no}"
63. echo "safe bitstream reader        ${safe_bitstream_reader-no}"
64. echo "SDL support                  ${sdl-no}"
65. echo "Sun medialib support         ${mlib-no}"
66. echo "libdxva2 enabled             ${dxva2-no}"
67. echo "libva enabled                ${vaapi-no}"
```

```

68. echo "libvdpau enabled      ${vdpau-no}"
69. echo "AVISynth enabled      ${avisynth-no}"
70. echo "frei0r enabled         ${frei0r-no}"
71. echo "gnutls enabled         ${gnutls-no}"
72. echo "libaacplus enabled     ${libaacplus-no}"
73. echo "libass enabled         ${libass-no}"
74. echo "libcdio support        ${libcdio-no}"
75. echo "libcelt enabled        ${libcelt-no}"
76. echo "libdc1394 support      ${libdc1394-no}"
77. echo "libdirac enabled       ${libdirac-no}"
78. echo "libfaac enabled        ${libfaac-no}"
79. echo "libgsm enabled         ${libgsm-no}"
80. echo "libmodplug enabled     ${libmodplug-no}"
81. echo "libmp3lame enabled     ${libmp3lame-no}"
82. echo "libnut enabled         ${libnut-no}"
83. echo "libopencore-amrnb support ${libopencore_amrnb-no}"
84. echo "libopencore-amrwb support ${libopencore_amrwb-no}"
85. echo "libopencv support      ${libopencv-no}"
86. echo "libopenjpeg enabled    ${libopenjpeg-no}"
87. echo "libpulse enabled       ${libpulse-no}"
88. echo "librtmp enabled        ${librtmp-no}"
89. echo "libschrödinger enabled  ${libschrödinger-no}"
90. echo "libspeex enabled       ${libspeex-no}"
91. echo "libstagefright-h264 enabled  ${libstagefright_h264-no}"
92. echo "libtheora enabled      ${libtheora-no}"
93. echo "libutvideo enabled     ${libutvideo-no}"
94. echo "libv4l2 enabled        ${libv4l2-no}"
95. echo "libvo-aacenc support    ${libvo_aacenc-no}"
96. echo "libvo-amrwbenc support  ${libvo_amrwbenc-no}"
97. echo "libvorbis enabled      ${libvorbis-no}"
98. echo "libvpx enabled         ${libvpx-no}"
99. echo "libx264 enabled        ${libx264-no}"
100. echo "libxavs enabled        ${libxavs-no}"
101. echo "libxvid enabled        ${libxvid-no}"
102. echo "openal enabled         ${openal-no}"
103. echo "openssl enabled       ${openssl-no}"
104. echo "zlib enabled           ${zlib-no}"
105. echo "bzlib enabled         ${bzlib-no}"
106. echo
107.
108. for type in decoder encoder hwaccel parser demuxer muxer protocol filter bsf indec outdev; do
109.     echo "Enabled ${type}s:"
110.     eval list=\${$(toupper $type)_LIST}
111.     print_enabled '_' $list | sort | pr -r -3 -t
112.     echo
113. done
114.
115. license="LGPL version 2.1 or later"
116. if enabled nonfree; then
117.     license="nonfree and unredistributable"
118. elif enabled gplv3; then
119.     license="GPL version 3 or later"
120. elif enabled lgplv3; then
121.     license="LGPL version 3 or later"
122. elif enabled gpl; then
123.     license="GPL version 2 or later"
124. fi
125.
126. echo "License: $license"

```

有关这段代码，有一个地方需要注意：很多的`\${}`符号中的字符为“XXX-no”，这种格式的意思是如果XXX取值为空，则使用默认值“no”（这个规则比较奇特）。

Write basic info

Write basic info用于向config.mak中写入一些基本信息。Configure中该部分的代码如下所示。

```

[python]
1. #创建config.mak和config.h
2. #根据情况也会创建config.asm
3. echo "Creating config.mak and config.h..."
4.
5. test -e Makefile || $ln_s "$source_path/Makefile" .
6.
7. enabled stripping || strip="echo skipping strip"
8. #重要：需要输出的文件
9. #TMPH是一个临时文件，最终会拷贝给config.h
10. config_files="TMPH config.mak"
11. #写入config.mak文件
12. #首先写入一些基本信息
13. #"<<EOF"表示后续输入作为子命令或子shell的输入，直到遇到"EOF"，再次返回到
14. #主调shell，可将其理解为分界符（delimiter）。
15. #最后的"EOF"必须单独占一行
16. cat > config.mak <<EOF
17. # Automatically generated by configure - do not modify!
18. ifndef FFMPEG_CONFIG_MAK
19. FFMPEG_CONFIG_MAK=1
20. FFMPEG_CONFIGURATION=$FFMPEG_CONFIGURATION
21.

```

```

21. prelink=aprilix
22. LIBDIR=\$(DESTDIR)\libdir
23. SHLIBDIR=\$(DESTDIR)\shlibdir
24. INCDIR=\$(DESTDIR)\incdir
25. BINDIR=\$(DESTDIR)\bindir
26. DATADIR=\$(DESTDIR)\datadir
27. MANDIR=\$(DESTDIR)\mandir
28. SRC_PATH=\$source_path
29. ifndef MAIN_MAKEFILE
30. SRC_PATH:=\$(SRC_PATH:./=.)
31. endif
32. CC_IDENT=\$cc_ident
33. ARCH=\$arch
34. CC=\$cc
35. CXX=\$cxx
36. AS=\$as
37. LD=\$ld
38. DEPCC=\$dep_cc
39. YASM=\$yasmexe
40. YASMDEP=\$yasmexe
41. AR=\$ar
42. RANLIB=\$ranlib
43. CP=cp -p
44. LN_S=\$ln_s
45. STRIP=\$strip
46. CPPFLAGS=\$CPPFLAGS
47. CFLAGS=\$CFLAGS
48. CXXFLAGS=\$CXXFLAGS
49. ASFLAGS=\$ASFLAGS
50. AS_O=\$CC_O
51. CC_O=\$CC_O
52. CXX_O=\$CXX_O
53. LDFLAGS=\$LDFLAGS
54. FFSERVERLDFLAGS=\$FFSERVERLDFLAGS
55. SHFLAGS=\$SHFLAGS
56. YASMFLAGS=\$YASMFLAGS
57. BUILDSUF=\$build_suffix
58. PROGSUF=\$progs_suffix
59. FULLNAME=\$FULLNAME
60. LIBPREF=\$LIBPREF
61. LIBSUF=\$LIBSUF
62. LIBNAME=\$LIBNAME
63. SLIBPREF=\$SLIBPREF
64. SLIBSUF=\$SLIBSUF
65. EXESUF=\$EXESUF
66. EXTRA_VERSION=\$extra_version
67. DEPFLAGS=\$DEPFLAGS
68. CCDEP=\$CCDEP
69. CXXDEP=\$CXXDEP
70. ASDEP=\$ASDEP
71. CC_DEPFLAGS=\$CC_DEPFLAGS
72. AS_DEPFLAGS=\$AS_DEPFLAGS
73. HOSTCC=\$host_cc
74. HOSTCFLAGS=\$host_cflags
75. HOSTEXESUF=\$HOSTEXESUF
76. HOSTLDFLAGS=\$host_ldflags
77. HOSTLIBS=\$host_libs
78. TARGET_EXEC=\$target_exec
79. TARGET_PATH=\$target_path
80. SDL_LIBS=\$sdl_libs
81. SDL_CFLAGS=\$sdl_cflags
82. LIB_INSTALL_EXTRA_CMD=\$LIB_INSTALL_EXTRA_CMD
83. EXTRALIBS=\$extralibs
84. INSTALL=\$install
85. LIBTARGET=\${LIBTARGET}
86. SLIBNAME=\${SLIBNAME}
87. SLIBNAME_WITH_VERSION=\${SLIBNAME_WITH_VERSION}
88. SLIBNAME_WITH_MAJOR=\${SLIBNAME_WITH_MAJOR}
89. SLIB_CREATE_DEF_CMD=\${SLIB_CREATE_DEF_CMD}
90. SLIB_EXTRA_CMD=\${SLIB_EXTRA_CMD}
91. SLIB_INSTALL_NAME=\${SLIB_INSTALL_NAME}
92. SLIB_INSTALL_LINKS=\${SLIB_INSTALL_LINKS}
93. SLIB_INSTALL_EXTRA_LIB=\${SLIB_INSTALL_EXTRA_LIB}
94. SLIB_INSTALL_EXTRA_SHLIB=\${SLIB_INSTALL_EXTRA_SHLIB}
95. SAMPLES=\${samples:-\$(FATE_SAMPLES)}
96. NOREDZONE_FLAGS=\$noredzone_flags
97. EOF
98. #获取版本
99. #主要通过各个类库文件夹中的version.h文件
100. #读取XXX_VERSION (相当于把头文件当成一个文本来读)
101. get_version(){
102.     name=\$1
103.     file=\$source_path/\$2
104.     # This condition will be removed when we stop supporting old libpostproc versions
105.     if ! test "\$name" = LIBPOSTPROC || test "\$postproc_version" = current; then
106.         eval \$(grep "#define \${name}_VERSION_M" "\$file" | awk '{ print \$2="\$3 }')
107.         eval \${name}_VERSION=\${\${name}_VERSION_MAJOR.\${\${name}_VERSION_MINOR.\${\${name}_VERSION_MICRO}
108.     fi
109.     lname=\$(tolower \$name)
110.     eval echo "\${lname}_VERSION=\${\${name}_VERSION}" >> config.mak
111.     eval echo "\${lname}_VERSION_MAJOR=\${\${name}_VERSION_MAJOR}" >> config.mak
112. }

```

```
113. #获取版本
114. get_version LIBAVCODEC libavcodec/version.h
115. get_version LIBAVDEVICE libavdevice/avdevice.h
116. get_version LIBAVFILTER libavfilter/version.h
117. get_version LIBAVFORMAT libavformat/version.h
118. get_version LIBAVUTIL libavutil/avutil.h
119. get_version LIBPOSTPROC libpostproc/postprocess.h
120. get_version LIBSWRESAMPLE libswresample/swresample.h
121. get_version LIBSWSCALE libswscale/swscale.h
```

关于这段代码，有以下几点需要注意：

- (1) “cat > config.mak <<EOF”的作用就是往config.mak中写入文本，当遇到“EOF”的时候写入结束
- (2) get_version()用于获取当前的Ffmpeg源代码中各个类库的版本。通过把各个类库文件夹下的version.h当作文本读取之后，分析字符串并且得到版本号，最终写入config.mak文件。

print_config()

print_config()用于向config.h、config.mak、config.asm中写入所有配置信息。Configure中该部分的代码如下所示。

```
1. #输出所有的配置信息包含3类：
2. #以“ARCH_”开头，包含系统架构信息
3. #以“HAVE_”开头，包含系统特征信息
4. #以“CONFIG_”开头，包含编译配置（数量最多，包含协议、复用器、编解码器等配置，将近1000行）
5. #config_files
6. print_config ARCH_ "$config_files" $ARCH_LIST
7. print_config HAVE_ "$config_files" $HAVE_LIST
8. print_config CONFIG_ "$config_files" $CONFIG_LIST \
9. $CONFIG_EXTRA \
10. $ALL_COMPONENTS \
```

从源代码中可以看出，其中调用了函数print_config()。print_config()的源代码如下所示。

```
1. # 输出文本到config.mak, config.h等文件
2. # 该函数的示例调用方法：print_config CONFIG_ "$config_files" $CONFIG_LIST
3. print_config(){
4. # 前缀
5. pfx=$1
6. # 文件列表
7. files=$2
8. # 位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1
9. shift 2
10. #for循环中，当没有in指定列表时，for会默认取命令行参数列表。
11. #在这里取的就是$CONFIG_LIST 等
12. for cfg; do
13. # toupper()：转换为大写
14. ucname="$(toupper $cfg)"
15. # files= config.h config.mak config.asm
16. # 循环输出
17. for f in $files; do
18. # "x#*"代表去取x的第一个slash之后的所有内容（不包括slash）
19. # "#"代表删除从前往后最小匹配的内容
20. # "f##*."代表去取f的第一个"."之后的所有内容。在这里是"h"、“mak”等
21. # 在这里print_config_h(),print_config_mak(),print_config_asm()
22. "print_config_${f##*}." $cfg ${pfx}${ucname} >>$f
23. done
24. done
25. }
```

可以看出print_config()的第1个参数是写入参数的前缀（例如可以取“ARCH_”、“HAVE_”、“CONFIG_”）；第2个参数是文件列表（例如可以是“config.h config.mak config.asm”）；第3个以后的参数就是需要写入的变量（例如\$ARCH_LIST、\$CONFIG_LIST等）。

print_config()有两层循环：外层循环遍历了所有的变量（例如\$CONFIG_LIST），内层循环遍历了所有文件（例如“config.h config.mak”），其中调用了函数print_config_XXX()，其中“XXX”根据文件后缀的不同可以取不同的值（例如“h”、“mak”）。下面举例看两个函数：print_config_h()和print_config_mak()。

print_config_h()

print_config_h()用于输出配置信息至config.h。该函数的源代码如下所示。

```
[python]
1. #输出config.h的时候使用
2. #调用示例:print_config_h ffplay CONFIG_FFPLAY
3. print_config_h(){
4.     #command1 && command2
5.     #&&左边的命令（命令1）返回真（即返回0，成功被执行）后，&&右边的命令（命令2）才能够被执行
6.     #command1 || command2
7.     #||左边的命令（命令1）未执行成功，那么就执行||右边的命令（命令2）
8.     enabled $1 && v=1 || v=0
9.     #示例:#define CONFIG_FFPLAY 1
10.    echo "#define $2 $v"
11. }
```

从源代码中可以看出，参数1是变量名称，参数2是经过处理后准备写入文件的变量名称（变量名转换成了大写并且添加了前缀）。如果参数1所指向的变量是enabled的，那么v取值为1，那么写入文件的格式就是：

```
[cpp]
1. #define {处理后变量名称} 1
```

如果参数1所指向的变量不是enabled的，那么v取值为0，那么写入文件的格式就是：

```
[python]
1. #define {处理后变量名称} 0
```

print_config_mak()

print_config_mak()用于输出配置信息至config.mak。该函数的源代码如下所示。

```
[python]
1. #输出config.mak的时候使用
2. print_config_mak(){
3.     enabled $1 && v= || v=!
4.     echo "$v$2=yes"
5. }
```

从源代码中可以看出print_config_mak()的原理和print_config_h()是类似的。如果变量是enabled的，那么写入文件的格式就是：

```
[plain]
1. {处理后变量名称}=yes
```

如果变量不是enabled的，那么写入文件的格式就是：

```
[plain]
1. !{处理后变量名称}=yes
```

print_enabled()

print_enabled()用于向config.mak写入所有enabled的组件信息。这方面功能通过print_enabled()函数完成，就不再详细分析了。

pkgconfig_generate()

pkgconfig_generate()用于向libavXXX/libavXXX.pc中写入pkgconfig信息（XXX代表avcodec，avformat等）。这方面的代码还没有细看，以后有机会再进行补充。

源代码（包含注释）

至此，FFmpeg的Configure的流程就大致梳理完毕了，最后附上和Configure有关的config.mak、config.h以及Configure本身的源代码。

config.mak源代码

```
[python]
1. # FFmpeg config.mak
2. #
3. # 注释：雷霄骅
4. # leixiaohua1020@126.com
5. # http://blog.csdn.net/leixiaohua1020
6. #
7. # Configure脚本生成的Makefile，包含了各种配置信息。
8. #
9. # Automatically generated by configure - do not modify!
10. #基本信息
11. ifndef FFMPEG_CONFIG_MAK
```



```

12. FFMPEG_CONFIG_MAK=1
13. FFMPEG_CONFIGURATION=
14. #各种路径=====
15. prefix=/usr/local
16. LIBDIR=$(DESTDIR){prefix}/lib
17. SHLIBDIR=$(DESTDIR){prefix}/bin
18. INCDIR=$(DESTDIR){prefix}/include
19. BINDIR=$(DESTDIR){prefix}/bin
20. DATADIR=$(DESTDIR){prefix}/share/ffmpeg
21. MANDIR=$(DESTDIR){prefix}/share/man
22. #是个相对路径
23. SRC_PATH=.
24. ifndef MAIN_MAKEFILE
25. SRC_PATH:=$(SRC_PATH:.%=%.%)
26. endif
27. #工具集=====
28. CC_IDENT=gcc 4.6.2 (GCC)
29. #架构
30. ARCH=x86
31. #编译器
32. CC=gcc
33. CXX=g++
34. AS=gcc
35. #链接器
36. LD=gcc
37. DEPCC=gcc
38. #汇编器
39. YASM=yasm
40. YASMDPE=yasm
41. #生成静态库.a工具
42. AR=ar
43. RANLIB=ranlib
44. CP=cp -p
45. LN_S=ln -sf
46. STRIP=strip
47. #参数集=====
48. #编译器的参数
49. CPPFLAGS= -D_ISO_C99_SOURCE -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -U_STRICT_ANSI
50. CFLAGS= -std=c99 -fno-common -fomit-frame-pointer -I/include/SDL -D_GNU_SOURCE=1 -Dmain=SDL_main -g -Wdeclaration-after-statement
-Wall -Wno-parentheses -Wno-switch -Wno-format-zero-length -Wdisabled-optimization -Wpointer-arith -Wredundant-decls -Wno-pointer-si
gn -Wcast-qual -Wwrite-strings -Wtype-limits -Wundef -Wmissing-prototypes -Wno-pointer-to-int-cast -Wstrict-prototypes -O3 -fno-math
-errno -fno-signed-zeros -fno-tree-vectorize -Werror=implicit-function-declaration -Werror=missing-prototypes
51. CXXFLAGS= -D__STDC_CONSTANT_MACROS
52. ASFLAGS= -g
53. #目标文件有关的参数
54. AS_0=-o $@
55. CC_0=-o $@
56. CXX_0=-o $@
57. #链接器有关的参数
58. LDFLAGS= -Wl,--as-needed -Wl,--warn-common -Wl,-rpath-
link=libpostproc:libswresample:libswscale:libavfilter:libavdevice:libavformat:libavcodec:libavutil
59. FFSERVERLDFLAGS=-Wl,-E
60. SHFLAGS=-shared -Wl,--output-def,$$(@:$(SLIBSUF)=.def) -Wl,--out-implib,$(SUBDIR)lib$(SLIBNAME:$(SLIBSUF)=.dll.a) -Wl,--enable-runti
me-pseudo-reloc -Wl,--enable-auto-image-base -Wl,-Bsymbolic -Wl,--version-script,$(SUBDIR)lib$(NAME).ver
61. YASMFLAGS=-f win32 -DPREFIX
62. #前缀后缀=====
63. BUILDSUF=
64. PROGSUF=
65. #$(NAME)位于每个liavXXX/Makefile中, 例如avformat
66. FULLNAME=$(NAME)$(BUILDSUF)
67. LIBPREF=lib
68. LIBSUF=.a
69. #例如libavformat.a
70. LIBNAME=$(LIBPREF)$(FULLNAME)$(LIBSUF)
71. SLIBPREF=
72. SLIBSUF=.dll
73. EXESUF=.exe
74. EXTRA_VERSION=
75. DEPFLAGS=$(CPPFLAGS) $(CFLAGS) -MM
76. CCDEP=
77. CXDEP=$(DEPCC) $(DEPFLAGS) $< | sed -e "/^\#.*\/d" -e "s,^[[:space:]]*$(F)\.o,$(@)/$(F).o," > $(@:.o=.d)
78. ASDEP=
79. CC_DEPFLAGS=-MMD -MF $(@:.o=.d) -MT $@
80. AS_DEPFLAGS=-MMD -MF $(@:.o=.d) -MT $@
81. HOSTCC=gcc
82. HOSTCFLAGS=-D_ISO_C99_SOURCE -O3 -g -std=c99 -Wall
83. HOSTEXESUF=.exe
84. HOSTLDFLAGS=
85. HOSTLIBS=-lm
86. TARGET_EXEC=
87. TARGET_PATH=$(CURDIR)
88. #SDL
89. SDL_LIBS=-L/lib -lmingw32 -lSDLmain -lSDL -lwindows
90. SDL_CFLAGS=-I/include/SDL -D_GNU_SOURCE=1 -Dmain=SDL_main
91. LIB_INSTALL_EXTRA_CMD=$(RANLIB) "$(LIBDIR)/$(LIBNAME)"
92. #链接
93. EXTRALIBS=-lavicap32 -lws2_32 -L/lib -lmingw32 -lSDLmain -lSDL -lwindows -lm -lz -lpapi
94. INSTALL=install
95. LIBTARGET=i386
96. #例如libavformat.dll
97. SLIBNAME=$(SLIBPREF)$(FULLNAME)$(SLIBSUF)

```

```

98. #LIBVERSION变量位于library.mak
99. #例如libavformat-53.dll
100. #生成的DLL似乎就是这个版本的
101. SLIBNAME_WITH_VERSION=$(SLIBPREFIX)$(FULLNAME) - $(LIBVERSION)$(SLIBSUF)
102. #例如libavformat-53.31.100.dll
103. SLIBNAME_WITH_MAJOR=$(SLIBPREFIX)$(FULLNAME) - $(LIBMAJOR)$(SLIBSUF)
104. SLIB_CREATE_DEF_CMD=
105. #生成导出库lib, 会调用lib.exe
106. SLIB_EXTRA_CMD=-lib.exe /machine:$(LIBTARGET) /def:$$(@:$(SLIBSUF)=.def) /out:$(SUBDIR)$(SLIBNAME:$(SLIBSUF)=.lib)
107. SLIB_INSTALL_NAME=$(SLIBNAME_WITH_MAJOR)
108. SLIB_INSTALL_LINKS=
109. SLIB_INSTALL_EXTRA_LIB=lib$(SLIBNAME:$(SLIBSUF)=.dll.a) $(SLIBNAME_WITH_MAJOR:$(SLIBSUF)=.def)
110. SLIB_INSTALL_EXTRA_SHLIB=$(SLIBNAME:$(SLIBSUF)=.lib)
111. SAMPLES:=$(FATE_SAMPLES)
112. NOREDZONE_FLAGS=-mno-red-zone
113. #版本信息=====
114. libavcodec_VERSION=53.60.100
115. libavcodec_VERSION_MAJOR=53
116. libavdevice_VERSION=53.4.100
117. libavdevice_VERSION_MAJOR=53
118. libavfilter_VERSION=2.60.100
119. libavfilter_VERSION_MAJOR=2
120. libavformat_VERSION=53.31.100
121. libavformat_VERSION_MAJOR=53
122. libavutil_VERSION=51.34.101
123. libavutil_VERSION_MAJOR=51
124. libpostproc_VERSION=52.0.100
125. libpostproc_VERSION_MAJOR=52
126. libswresample_VERSION=0.6.100
127. libswresample_VERSION_MAJOR=0
128. libswscale_VERSION=2.1.100
129. libswscale_VERSION_MAJOR=2
130. #组件配置=====
131. #ARCH_
132. !ARCH_ALPHA=yes
133. !ARCH_ARM=yes
134. !ARCH_AVR32=yes
135. !ARCH_AVR32_AP=yes
136. !ARCH_AVR32_UC=yes
137. !ARCH_BFIN=yes
138. !ARCH_IA64=yes
139. !ARCH_M68K=yes
140. !ARCH_MIPS=yes
141. !ARCH_MIPS64=yes
142. !ARCH_PARISC=yes
143. !ARCH_PPC=yes
144. !ARCH_PPC64=yes
145. !ARCH_S390=yes
146. !ARCH_SH4=yes
147. !ARCH_SPARC=yes
148. !ARCH_SPARC64=yes
149. !ARCH_TOMI=yes
150. ARCH_X86=yes
151. ARCH_X86_32=yes
152. !ARCH_X86_64=yes
153. #HAVE_
154. !HAVE_ALTIVEC=yes
155. HAVE_AMD3DNOW=yes
156. HAVE_AMD3DNOWEXT=yes
157. !HAVE_ARMV5TE=yes
158. !HAVE_ARMV6=yes
159. !HAVE_ARMV6T2=yes
160. !HAVE_ARMVFP=yes
161. HAVE_AVX=yes
162. !HAVE_IWMX=yes
163. !HAVE_MMI=yes
164. HAVE_MMX=yes
165. HAVE_MMX2=yes
166. //略.....
167. HAVE_YASM=yes
168. #CONFIG_
169. CONFIG_BSFS=yes
170. CONFIG_DECODERS=yes
171. CONFIG_DEMUXERS=yes
172. CONFIG_ENCODERS=yes
173. CONFIG_FILTERS=yes
174. !CONFIG_HWACCELS=yes
175. CONFIG_INDEVS=yes
176. CONFIG_MUXERS=yes
177. CONFIG_OUTDEVS=yes
178. CONFIG_PARSERS=yes
179. CONFIG_PROTOCOLS=yes
180. CONFIG_FFPLAY=yes
181. CONFIG_FFPROBE=yes
182. !CONFIG_FFSERVER=yes
183. CONFIG_FFMPEG=yes
184. !CONFIG_AVPLAY=yes
185. !CONFIG_AVPROBE=yes
186. !CONFIG_AVSERVER=yes
187. CONFIG_AANDCT=yes
188. CONFIG_AC3DSP=yes

```

```
189. CONFIG_AVCODEC=yes
190. CONFIG_AVDEVICE=yes
191. CONFIG_AVFILTER=yes
192. CONFIG_AVFORMAT=yes
193. !CONFIG_AVISYNTH=yes
194. !CONFIG_BZLIB=yes
195. !CONFIG_CRYSTALHD=yes
196. CONFIG_DCT=yes
197. !CONFIG_D0C=yes
198. CONFIG_DWT=yes
199. !CONFIG_DXVA2=yes
200. CONFIG_FASTDIV=yes
201. CONFIG_FFT=yes
202. !CONFIG_FREI0R=yes
203. !CONFIG_GNUTLS=yes
204. CONFIG_GOLOMB=yes
205. !CONFIG_GPL=yes
206. !CONFIG_GRAY=yes
207. CONFIG_H264CHROMA=yes
208. CONFIG_H264DSP=yes
209. CONFIG_H264PRED=yes
210. !CONFIG_HARDCODED_TABLES=yes
211. CONFIG_HUFFMAN=yes
212. !CONFIG_LIBAACPLUS=yes
213. !CONFIG_LIBASS=yes
214. !CONFIG_LIBCDIO=yes
215. !CONFIG_LIBCELT=yes
216. !CONFIG_LIBDC1394=yes
217. !CONFIG_LIBDIRAC=yes
218. !CONFIG_LIBFAAC=yes
219. !CONFIG_LIBFREETYPE=yes
220. !CONFIG_LIBGSM=yes
221. !CONFIG_LIBMODPLUG=yes
222. !CONFIG_LIBMP3LAME=yes
223. !CONFIG_LIBNUT=yes
224. !CONFIG_LIBOPENCORE_AMRNB=yes
225. !CONFIG_LIBOPENCORE_AMRWB=yes
226. !CONFIG_LIBOPENCV=yes
227. !CONFIG_LIBOPENJPEG=yes
228. !CONFIG_LIBPULSE=yes
229. !CONFIG_LIBRTMP=yes
230. !CONFIG_LIBSCHROEDINGER=yes
231. !CONFIG_LIBSPEEX=yes
232. !CONFIG_LIBSTAGEFRIGHT_H264=yes
233. !CONFIG_LIBTHEORA=yes
234. !CONFIG_LIBUTVIDE0=yes
235. !CONFIG_LIBV4L2=yes
236. !CONFIG_LIBVO_AACENC=yes
237. !CONFIG_LIBVO_AMRWBENC=yes
238. !CONFIG_LIBVORBIS=yes
239. !CONFIG_LIBVPX=yes
240. !CONFIG_LIBX264=yes
241. !CONFIG_LIBXAVS=yes
242. !CONFIG_LIBXVID=yes
243. #此处省略将近1000条...
244. CONFIG_RTMP_PROTOCOL=yes
245. CONFIG_RTMP_PROTOCOL=yes
246. CONFIG_RTMP_PROTOCOL=yes
247. CONFIG_RTMP_PROTOCOL=yes
248. CONFIG_RTMP_PROTOCOL=yes
249. CONFIG_RTP_PROTOCOL=yes
250. CONFIG_TCP_PROTOCOL=yes
251. !CONFIG_TLS_PROTOCOL=yes
252. CONFIG_UDP_PROTOCOL=yes
253. #Test
254. ACODEC_TESTS=ac3_fixed adpcm_adx adpcm_ima_qt adpcm_ima_wav adpcm_ms adpcm_swf adpcm_yam alac aref flac g722 g723_1 g726 mp2 pcm_alaw
    pcm_f32be pcm_f32le pcm_f64be pcm_f64le pcm_mulaw pcm_s16be pcm_s16le pcm_s24be pcm_s24daud pcm_s24le pcm_s32be pcm_s32le pcm_s8 pcm_u8
    av1 wavav2
255. VCODEC_TESTS=amv asv1 asv2 cljr dnxhd_1080i dnxhd_720p dnxhd_720p_10bit dnxhd_720p_rd dv dv50 dv_411 error ffv1 flashsv flashsv2 flv
    1 h263 h263p huffyuv jpeg2000 jpeglS jpeg mjpeg mpeg mpeg1b mpeg2 mpeg2_422 mpeg2_idct_int mpeg2_ilace mpeg2_ivlc qprd mpeg2thread m
    2thread ilace mpeg4 mpeg4_adap mpeg4_qpel mpeg4_qprd mpeg4adv mpeg4nr mpeg4thread mpng msmpeg4 msmpeg4v2 msvideo1 prores qtrle qtrleg
    rc rgb roq rv10 rv20 snow snowll svq1 v210 vref wmv1 wmv2 yuv zlib zmbv
256. LAVF_TESTS=aiff alaw asf au avi bmp caf dpx dv_fmt ffm flv_fmt gif gxf jpg mkv mmf mov mpg mulaw mxf mxf_d10 nut ogg pbmpipe pcx pgm
    pipe pixfmt png ppm pppipe rm rso sgi sox swf tga tiff ts voc voc_s16 wav wtv yuv4mpeg
257. LAVFI_TESTS=crop crop_scale crop_scale_vflip crop_vflip null pixdesc pixfmts_copy pixfmts_crop pixfmts_hflip pixfmts_null pixfmts_pad
    xfmts_scale pixfmts_vflip scale200 scale500 vflip vflip_crop vflip_vflip
258. SEEK_TESTS=seek_ac3_rm seek_adpcm_ima_wav seek_adpcm_ms_wav seek_adpcm_qt_aiff seek_adpcm_swf_flv seek_adpcm_yam_wav seek_alac_m4a se
    asv1 avi seek_asv2 avi seek_dnxhd_1080i mov seek_dnxhd_720p_dnxhd seek_dnxhd_720p_rd dnxhd seek_dv411 dv seek_dv50 dv seek_dv dv seek
    ror_mpeg4_adv.avi seek_ffv1.avi seek_flac_flac seek_flashsv_flv seek_flv_flv seek_g726_wav seek_h261.avi seek_h263.avi seek_h263p.avi
    ek_huffyuv.avi seek_image.bmp seek_image.jpg seek_image_pcx seek_image_pgm seek_image_ppm seek_image_sgi seek_image_tga seek_image_ti
    seek_jpegls.avi seek_lavf_aif seek_lavf_al seek_lavf_asf seek_lavf_av seek_lavf.avi seek_lavf_dv seek_lavf_ffm seek_lavf_flv seek_lav
    if seek_lavf_gxf seek_lavf_mkv seek_lavf_mmf seek_lavf_mov seek_lavf_mpg seek_lavf_mxf seek_lavf_mxf_d10 seek_lavf_nut seek_lavf_ogg
    k_lavf_rm seek_lavf_swf seek_lavf_ts seek_lavf_ul seek_lavf_voc seek_lavf_wav seek_lavf_wtv seek_lavf_y4m seek_ljpeg.avi seek_mjpeg_a
    seek_mp2_mp2 seek_mpeg1_mpg seek_mpeg1b_mpg seek_mpeg2_422_mpg seek_mpeg2_idct_int_mpg seek_mpeg2i_mpg seek_mpeg2ivlc_qprd_mpg seek_m
    2reuse_mpg seek_mpeg2thread_mpg seek_mpeg2threadivlc_mpg seek_mpeg4_adv.avi seek_mpeg4_adv.avi seek_mpeg4_nr.avi seek_mpeg4_qprd.avi
    ek_mpeg4_rc.avi seek_mpeg4_thread.avi seek_msmpeg4.avi seek_msmpeg4v2.avi seek_odivx_mp4 seek_pbmpipe_pbm seek_pcm_alaw_wav seek_pcm_
    be_au seek_pcm_f32le_wav seek_pcm_f64be_au seek_pcm_f64le_wav seek_pcm_mulaw_wav seek_pcm_s16be_mov seek_pcm_s16le_wav seek_pcm_s24be
    v seek_pcm_s24daud_302 seek_pcm_s24le_wav seek_pcm_s32be_mov seek_pcm_s32le_wav seek_pcm_s8_mov seek_pcm_u8_wav seek_pgmpipe_pgm seek
    mpipipe_ppm seek_rgb.avi seek_roqav_roq seek_rv10_rm seek_rv20_rm seek_snow53.avi seek_snow.avi seek_svq1_mov seek_wmav1_asf seek_wmav2
    f seek_wmv1.avi seek_wmv2.avi seek_yuv.avi
259. endif # FFMPEG_CONFIG_MAK
```

config.h源代码

```

1.  /**
2.   * Ffmpeg config.h
3.   *
4.   * 注释：雷霄骅
5.   * leixiaohua1020@126.com
6.   * http://blog.csdn.net/leixiaohua1020
7.   *
8.   * Configure脚本生成的config.h, 包含了各种配置信息。
9.   /
10.  /* Automatically generated by configure - do not modify! */
11.  #ifndef FFMPEG_CONFIG_H
12.  #define FFMPEG_CONFIG_H
13.  #define FFMPEG_CONFIGURATION ""
14.  #define FFMPEG_LICENSE "LGPL version 2.1 or later"
15.  #define FFMPEG_DATADIR "/usr/local/share/ffmpeg"
16.  #define AVCONV_DATADIR "/usr/local/share/ffmpeg"
17.  #define CC_TYPE "gcc"
18.  #define CC_VERSION _VERSION_
19.  #define restrict restrict
20.  #define EXTERN_PREFIX " "
21.  #define EXTERN_ASM _
22.  #define SLIBSUF ".dll"
23.  #define ARCH_ALPHA 0
24.  #define ARCH_ARM 0
25.  #define ARCH_AVR32 0
26.  #define ARCH_AVR32_AP 0
27.  #define ARCH_AVR32_UC 0
28.  #define ARCH_BFIN 0
29.  #define ARCH_IA64 0
30.  #define ARCH_M68K 0
31.  #define ARCH_MIPS 0
32.  #define ARCH_MIPS64 0
33.  #define ARCH_PARISC 0
34.  #define ARCH_PPC 0
35.  #define ARCH_PPC64 0
36.  #define ARCH_S390 0
37.  #define ARCH_SH4 0
38.  #define ARCH_SPARC 0
39.  #define ARCH_SPARC64 0
40.  #define ARCH_TOMI 0
41.  #define ARCH_X86 1
42.  #define ARCH_X86_32 1
43.  #define ARCH_X86_64 0
44.  #define HAVE_ALTIVEC 0
45.  #define HAVE_AMD3DNOW 1
46.  #define HAVE_AMD3DNOWEXT 1
47.  #define HAVE_ARMV5TE 0
48.  #define HAVE_ARMV6 0
49.  #define HAVE_ARMV6T2 0
50.  #define HAVE_ARMVFP 0
51.  #define HAVE_AVX 1
52.  #define HAVE_IWMXMT 0
53.  #define HAVE_MMI 0
54.  #define HAVE_MMX 1
55.  #define HAVE_MMX2 1
56.  //略.....
57.  #define HAVE_YASM 1
58.  #define CONFIG_BSFS 1
59.  #define CONFIG_DECODERS 1
60.  #define CONFIG_DEMUXERS 1
61.  #define CONFIG_ENCODERS 1
62.  #define CONFIG_FILTERS 1
63.  #define CONFIG_HWACCELS 0
64.  #define CONFIG_INDEVS 1
65.  #define CONFIG_MUXERS 1
66.  #define CONFIG_OUTDEVS 1
67.  #define CONFIG_PARSERS 1
68.  #define CONFIG_PROTOCOLS 1
69.  #define CONFIG_FFPLAY 1
70.  #define CONFIG_FFPROBE 1
71.  #define CONFIG_FFSERVER 0
72.  #define CONFIG_FFMPEG 1
73.  #define CONFIG_AVPLAY 0
74.  #define CONFIG_AVPROBE 0
75.  #define CONFIG_AVSERVER 0
76.  #define CONFIG_AANDCT 1
77.  #define CONFIG_AC3DSP 1
78.  #define CONFIG_AVCODEC 1
79.  #define CONFIG_AVDEVICE 1
80.  #define CONFIG_AVFILTER 1
81.  #define CONFIG_AVFORMAT 1
82.  #define CONFIG_AVISYNTH 0

```

```

83. #define CONFIG_BZLIB 0
84. #define CONFIG_CRYSTALHD 0
85. #define CONFIG_DCT 1
86. #define CONFIG_DOC 0
87. #define CONFIG_DWT 1
88. #define CONFIG_DXVA2 0
89. #define CONFIG_FASTDIV 1
90. #define CONFIG_FFT 1
91. #define CONFIG_FREI0R 0
92. #define CONFIG_GNUTLS 0
93. #define CONFIG_GOLOMB 1
94. #define CONFIG_GPL 0
95. #define CONFIG_GRAY 0
96. #define CONFIG_H264CHROMA 1
97. #define CONFIG_H264DSP 1
98. #define CONFIG_H264PRED 1
99. #define CONFIG_HARDCODED_TABLES 0
100. #define CONFIG_HUFFMAN 1
101. #define CONFIG_LIBAACPLUS 0
102. #define CONFIG_LIBASS 0
103. #define CONFIG_LIBCDIO 0
104. #define CONFIG_LIBCELT 0
105. #define CONFIG_LIBDC1394 0
106. #define CONFIG_LIBDIRAC 0
107. #define CONFIG_LIBFAAC 0
108. #define CONFIG_LIBFRETYPE 0
109. #define CONFIG_LIBGSM 0
110. #define CONFIG_LIBMODPLUG 0
111. #define CONFIG_LIBMP3LAME 0
112. #define CONFIG_LIBNUT 0
113. #define CONFIG_LIBOPENCORE_AMRNB 0
114. #define CONFIG_LIBOPENCORE_AMRWB 0
115. #define CONFIG_LIBOPENCV 0
116. #define CONFIG_LIBOPENJPEG 0
117. #define CONFIG_LIBPULSE 0
118. #define CONFIG_LIBRTMP 0
119. #define CONFIG_LIBSCHROEDINGER 0
120. #define CONFIG_LIBSPEEX 0
121. #define CONFIG_LIBSTAGEFRIGHT_H264 0
122. #define CONFIG_LIBTHEORA 0
123. #define CONFIG_LIBUTVIDEO 0
124. #define CONFIG_LIBV4L2 0
125. #define CONFIG_LIBVO_AACENC 0
126. #define CONFIG_LIBVO_AMRWBENC 0
127. #define CONFIG_LIBVORBIS 0
128. #define CONFIG_LIBVPX 0
129. #define CONFIG_LIBX264 0
130. //此处省略将近1000条
131. #define CONFIG_RTMP_PROTOCOL 1
132. #define CONFIG_RTMP_T_PROTOCOL 1
133. #define CONFIG_RTMP_E_PROTOCOL 1
134. #define CONFIG_RTMP_T_E_PROTOCOL 1
135. #define CONFIG_RTMP_S_PROTOCOL 1
136. #define CONFIG_RTP_PROTOCOL 1
137. #define CONFIG_TCP_PROTOCOL 1
138. #define CONFIG_TLS_PROTOCOL 0
139. #define CONFIG_UDP_PROTOCOL 1
140. #endif /* FFMPEG_CONFIG_H */

```

Configure的源代码

```

[plain]
1. #!/bin/sh
2. #
3. # FFMpeg configure script
4. #
5. # Copyright (c) 2000-2002 Fabrice Bellard
6. # Copyright (c) 2005-2008 Diego Biurrun
7. # Copyright (c) 2005-2008 Mans Rullgard
8. #
9. # 注释: 雷霄骅
10. # leixiaohua1020@126.com
11. # http://blog.csdn.net/leixiaohua1020
12. #
13. # 添加了注释的FFmpeg的Configure文件
14. #
15. # Prevent locale nonsense from breaking basic text processing.
16. LC_ALL=C
17. export LC_ALL
18.
19. # make sure we are running under a compatible shell
20. # try to make this part work with most shells
21.
22. try_exec(){
23.     echo "Trying shell $1"
24.     type "$1" > /dev/null 2>&1 && exec "$@"
25. }

```

```

26.
27. unset foo
28. (: ${foo%%bar}) 2> /dev/null
29. E1="$?"
30.
31. (: ${foo?}) 2> /dev/null
32. E2="$?"
33.
34. if test "$E1" != 0 || test "$E2" = 0; then
35.     echo "Broken shell detected. Trying alternatives."
36.     export FF_CONF_EXEC
37.     if test "$FF_CONF_EXEC" -lt 1; then
38.         FF_CONF_EXEC=1
39.         try_exec bash "$@" "$@"
40.     fi
41.     if test "$FF_CONF_EXEC" -lt 2; then
42.         FF_CONF_EXEC=2
43.         try_exec ksh "$@" "$@"
44.     fi
45.     if test "$FF_CONF_EXEC" -lt 3; then
46.         FF_CONF_EXEC=3
47.         try_exec /usr/xpg4/bin/sh "$@" "$@"
48.     fi
49.     echo "No compatible shell script interpreter found."
50.     echo "This configure script requires a POSIX-compatible shell"
51.     echo "such as bash or ksh."
52.     echo "THIS IS NOT A BUG IN FFMPEG, DO NOT REPORT IT AS SUCH."
53.     echo "Instead, install a working POSIX-compatible shell."
54.     echo "Disabling this configure test will create a broken Ffmpeg."
55.     if test "$BASH_VERSION" = '2.04.0(1)-release'; then
56.         echo "This bash version ($BASH_VERSION) is broken on your platform."
57.         echo "Upgrade to a later version if available."
58.     fi
59.     exit 1
60. fi
61. #帮助菜单
62. show_help(){
63.     cat <<EOF
64.     Usage: configure [options]
65.     Options: [defaults in brackets after descriptions]
66.
67.     Standard options:
68.     --help                print this message
69.     --logfile=FILE        log tests and output to FILE [config.log]
70.     --disable-logging      do not log configure debug information
71.     --prefix=PREFIX       install in PREFIX [$prefix]
72.     --bindir=DIR           install binaries in DIR [PREFIX/bin]
73.     --datadir=DIR          install data files in DIR [PREFIX/share/ffmpeg]
74.     --libdir=DIR           install libs in DIR [PREFIX/lib]
75.     --shlibdir=DIR         install shared libs in DIR [PREFIX/lib]
76.     --incdir=DIR           install includes in DIR [PREFIX/include]
77.     --mandir=DIR           install man page in DIR [PREFIX/share/man]
78.
79.     Configuration options:
80.     --disable-static       do not build static libraries [no]
81.     --enable-shared        build shared libraries [no]
82.     --enable-gpl           allow use of GPL code, the resulting libs
83.                           and binaries will be under GPL [no]
84.     --enable-version3      upgrade (L)GPL to version 3 [no]
85.     --enable-nonfree       allow use of nonfree code, the resulting libs
86.                           and binaries will be unredistributable [no]
87.     --disable-doc          do not build documentation
88.     --disable-ffmpeg       disable ffmpeg build
89.     --disable-ffplay       disable ffplay build
90.     --disable-ffprobe      disable ffprobe build
91.     --disable-ffserver     disable ffserver build
92.     --disable-avdevice     disable libavdevice build
93.     --disable-avcodec      disable libavcodec build
94.     --disable-avformat     disable libavformat build
95.     --disable-swsample     disable libswresample build
96.     --disable-swscale      disable libswscale build
97.     --disable-postproc     disable libpostproc build
98.     --disable-avfilter     disable video filter support [no]
99.     --disable-pthreads     disable pthreads [auto]
100.    --disable-w32threads    disable Win32 threads [auto]
101.    --disable-os2threads    disable OS/2 threads [auto]
102.    --enable-x11grab        enable X11 grabbing [no]
103.    --disable-network       disable network support [no]
104.    --enable-gray           enable full grayscale support (slower color)
105.    --disable-swscale-alpha disable alpha channel support in swscale
106.    --disable-fastdiv       disable table-based division
107.    --enable-small          optimize for size instead of speed
108.    --disable-aandct        disable AAN DCT code
109.    --disable-dct           disable DCT code
110.    --disable-fft           disable FFT code
111.    --disable-golomb        disable Golomb code
112.    --disable-huffman       disable Huffman code
113.    --disable-lpc           disable LPC code
114.    --disable-mdct          disable MDCT code
115.    --disable-rdft          disable RDFT code
116.    --enable-vaapi          enable VAAPI code [autodetect]
117.    --enable-vdpau          enable VDA code [autodetect]

```

```

117. --enable-vda          enable VDA code [autodetect]
118. --enable-udpau        enable VDPau code [autodetect]
119. --disable-dxva2       disable DXVA2 code
120. --disable-vda         disable VDA code
121. --enable-runtime-cpudetect detect cpu capabilities at runtime (bigger binary)
122. --enable-hardcoded-tables use hardcoded tables instead of runtime generation
123. --disable-safe-bitstream-reader
124.                      disable buffer boundary checking in bitreaders
125.                      (faster, but may crash)
126. --enable-memalign-hack emulate memalign, interferes with memory debuggers
127. --disable-everything  disable all components listed below
128. --disable-encoder=NAME disable encoder NAME
129. --enable-encoder=NAME  enable encoder NAME
130. --disable-encoders     disable all encoders
131. --disable-decoder=NAME disable decoder NAME
132. --enable-decoder=NAME  enable decoder NAME
133. --disable-decoders     disable all decoders
134. --disable-hwaccel=NAME disable hwaccel NAME
135. --enable-hwaccel=NAME  enable hwaccel NAME
136. --disable-hwaccels     disable all hwaccels
137. --disable-muxer=NAME   disable muxer NAME
138. --enable-muxer=NAME    enable muxer NAME
139. --disable-muxers       disable all muxers
140. --disable-demuxer=NAME disable demuxer NAME
141. --enable-demuxer=NAME  enable demuxer NAME
142. --disable-demuxers     disable all demuxers
143. --enable-parser=NAME   enable parser NAME
144. --disable-parser=NAME  disable parser NAME
145. --disable-parsers      disable all parsers
146. --enable-bsf=NAME       enable bitstream filter NAME
147. --disable-bsf=NAME     disable bitstream filter NAME
148. --disable-bsfs         disable all bitstream filters
149. --enable-protocol=NAME enable protocol NAME
150. --disable-protocol=NAME disable protocol NAME
151. --disable-protocols    disable all protocols
152. --disable-indev=NAME   disable input device NAME
153. --disable-outdev=NAME  disable output device NAME
154. --disable-indevs       disable input devices
155. --disable-outdevs      disable output devices
156. --disable-devices      disable all devices
157. --enable-filter=NAME   enable filter NAME
158. --disable-filter=NAME  disable filter NAME
159. --disable-filters      disable all filters
160. --list-decoders         show all available decoders
161. --list-encoders         show all available encoders
162. --list-hwaccels         show all available hardware accelerators
163. --list-muxers           show all available muxers
164. --list-demuxers         show all available demuxers
165. --list-parsers          show all available parsers
166. --list-protocols        show all available protocols
167. --list-bsfs             show all available bitstream filters
168. --list-indevs           show all available input devices
169. --list-outdevs          show all available output devices
170. --list-filters          show all available filters
171.
172. External library support:
173. --enable-avisynth       enable reading of AVISynth script files [no]
174. --enable-bzlib           enable bzlib [autodetect]
175. --enable-frei0r          enable frei0r video filtering
176. --enable-gnutls          enable gnutls [no]
177. --enable-libaacplus      enable AAC+ encoding via libaacplus [no]
178. --enable-libass          enable libass subtitles rendering [no]
179. --enable-libcelt         enable CELT decoding via libcelt [no]
180. --enable-libopencore-amrnb enable AMR-NB de/encoding via libopencore-amrnb [no]
181. --enable-libopencore-amrwb enable AMR-WB decoding via libopencore-amrwb [no]
182. --enable-libopencl       enable video filtering via libopencl [no]
183. --enable-libcdio         enable audio CD grabbing with libcdio
184. --enable-libdc1394       enable IIDC-1394 grabbing using libdc1394
185.                         and libraw1394 [no]
186. --enable-libdirac        enable Dirac support via libdirac [no]
187. --enable-libfaac         enable FAAC support via libfaac [no]
188. --enable-libfreetype     enable libfreetype [no]
189. --enable-libgsm          enable GSM support via libgsm [no]
190. --enable-libmodplug      enable ModPlug via libmodplug [no]
191. --enable-libmp3lame       enable MP3 encoding via libmp3lame [no]
192. --enable-libnut          enable NUT (de)muxing via libnut,
193.                         native (de)muxer exists [no]
194. --enable-libopenjpeg     enable JPEG 2000 encoding/decoding via OpenJPEG [no]
195. --enable-libpulse        enable Pulseaudio input via libpulse [no]
196. --enable-librtmp         enable RTMP[E] support via librtmp [no]
197. --enable-libschrödinger  enable Dirac support via libschrödinger [no]
198. --enable-libspeex        enable Speex support via libspeex [no]
199. --enable-libstagefright-h264 enable H.264 decoding via libstagefright [no]
200. --enable-libtheora       enable Theora encoding via libtheora [no]
201. --enable-libutvideo      enable Ut Video decoding via libutvideo [no]
202. --enable-libv4l2         enable libv4l2/v4l-utils [no]
203. --enable-libvo-aacenc     enable AAC encoding via libvo-aacenc [no]
204. --enable-libvo-amrwbenc   enable AMR-WB encoding via libvo-amrwbenc [no]
205. --enable-libvorbis       enable Vorbis encoding via libvorbis,
206.                         native implementation exists [no]
207. --enable-libvpx          enable VP8 support via libvpx [no]
208. --enable-libx264         enable H.264 encoding via x264 [no]

```

```

200. --enable-libx264      enable H.264 encoding via x264 [no]
209. --enable-libxavs      enable AVS encoding via xavs [no]
210. --enable-libxvid      enable Xvid encoding via xvidcore,
211.                        native MPEG-4/Xvid encoder exists [no]
212. --enable-openal       enable OpenAL 1.1 capture support [no]
213. --enable-mllib        enable Sun medialib [no]
214. --enable-openssl      enable openssl [no]
215. --enable-zlib         enable zlib [autodetect]
216.
217. Advanced options (experts only):
218. --cross-prefix=PREFIX use PREFIX for compilation tools [$cross_prefix]
219. --enable-cross-compile assume a cross-compiler is used
220. --sysroot=PATH         root of cross-build tree
221. --sysinclude=PATH      location of cross-build system headers
222. --target-os=OS         compiler targets OS [$target_os]
223. --target-exec=CMD      command to run executables on target
224. --target-path=DIR      path to view of build directory on target
225. --nm=NM               use nm tool NM [$nm_default]
226. --ar=AR               use archive tool AR [$ar_default]
227. --as=AS               use assembler AS [$as_default]
228. --yasmexe=EXE         use yasm-compatible assembler EXE [$yasmexe_default]
229. --cc=CC               use C compiler CC [$cc_default]
230. --cxx=CXX            use C compiler CXX [$cxx_default]
231. --ld=LD               use linker LD [$ld_default]
232. --host-cc=HOSTCC      use host C compiler HOSTCC
233. --host-cflags=HCFLAGS use HCFLAGS when compiling for host
234. --host-ldflags=HLDFLAGS use HLDFLAGS when linking for host
235. --host-libs=HLIBS     use libs HLIBS when linking for host
236. --extra-cflags=ECFLAGS add ECFLAGS to CFLAGS [$CFLAGS]
237. --extra-cxxflags=ECXFLAGS add ECXFLAGS to CXXFLAGS [$CXXFLAGS]
238. --extra-ldflags=ELDFLAGS add ELDFLAGS to LDFLAGS [$LDFLAGS]
239. --extra-libs=ELIBS    add ELIBS [$ELIBS]
240. --extra-version=STRING version string suffix []
241. --build-suffix=SUFFIX library name suffix []
242. --progs-suffix=SUFFIX program name suffix []
243. --arch=ARCH           select architecture [$arch]
244. --cpu=CPU             select the minimum required CPU (affects
245.                        instruction selection, may crash on older CPUs)
246. --disable-asm         disable all assembler optimizations
247. --disable-altivec     disable AltiVec optimizations
248. --disable-amd3dnow    disable 3DNow! optimizations
249. --disable-amd3dnowext disable 3DNow! extended optimizations
250. --disable-mmx         disable MMX optimizations
251. --disable-mmx2        disable MMX2 optimizations
252. --disable-sse         disable SSE optimizations
253. --disable-ssse3       disable SSSE3 optimizations
254. --disable-avx         disable AVX optimizations
255. --disable-armv5te     disable armv5te optimizations
256. --disable-armv6       disable armv6 optimizations
257. --disable-armv6t2     disable armv6t2 optimizations
258. --disable-armvfp      disable ARM VFP optimizations
259. --disable-iwmmxt      disable iwmmxt optimizations
260. --disable-mmi         disable MMI optimizations
261. --disable-neon        disable NEON optimizations
262. --disable-vis         disable VIS optimizations
263. --disable-yasm        disable use of yasm assembler
264. --enable-pic          build position-independent code
265. --malloc-prefix=PFX   prefix malloc and related names with PFX
266. --enable-sram         allow use of on-chip SRAM
267. --disable-symver      disable symbol versioning
268. --optflags            override optimization-related compiler flags
269. --postproc-version=V  build libpostproc version V.
270.                        Where V can be 'ALT_PP_VER_MAJOR.$ALT_PP_VER_MINOR.$ALT_PP_VER_MICRO' or 'current'. [$postproc_version_de
lt]
271.
272. Developer options (useful when working on FFmpeg itself):
273. --enable-coverage     build with test coverage instrumentation
274. --disable-debug       disable debugging symbols
275. --enable-debug=LEVEL  set the debug level [$debuglevel]
276. --disable-optimizations disable compiler optimizations
277. --enable-extra-warnings enable more compiler warnings
278. --disable-stripping   disable stripping of executables and shared libraries
279. --valgrind=VALGRIND  run "make fate" tests through valgrind to detect memory
280.                        leaks and errors, using the specified valgrind binary.
281.                        Cannot be combined with --target-exec
282. --samples=PATH        location of test samples for FATE, if not set use
283.                        \FATE_SAMPLES at make invocation time.
284.
285. NOTE: Object files are built at the place where configure is launched.
286. EOF
287.     exit 0
288. }
289.
290. quotes='""'
291.
292. #日志config.log
293. log(){
294.     echo "$@" >> $logfile
295. }
296.
297. log_file(){
298.     log BEGIN $1

```



```

299.     pr -n -t $1 >> $logfile
300.     log END $1
301. }
302.
303. echolog(){
304.     log "$@"
305.     echo "$@"
306. }
307.
308. warn(){
309.     log "WARNING: $"
310.     WARNINGS="{WARNINGS}WARNING: $*\n"
311. }
312.
313. #出错了
314. die(){
315.     echolog "$@"
316.     cat <<EOF
317.
318. If you think configure made a mistake, make sure you are using the latest
319. version from Git. If the latest version fails, report the problem to the
320. ffmpeg-user@ffmpeg.org mailing list or IRC #ffmpeg on irc.freenode.net.
321. EOF
322.     if disabled logging; then
323.         cat <<EOF
324. Rerun configure with logging enabled (do not use --disable-logging), and
325. include the log this produces with your report.
326. EOF
327.     else
328.         cat <<EOF
329. Include the log file "$logfile" produced by configure as this will help
330. solving the problem.
331. EOF
332.     fi
333.     exit 1
334. }
335.
336. # Avoid locale weirdness, besides we really just want to translate ASCII.
337. toupper(){
338.     echo "$@" | tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
339. }
340.
341. tolower(){
342.     echo "$@" | tr ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
343. }
344.
345. c_escape(){
346.     echo "$*" | sed 's/[\"\\]/\\0/g'
347. }
348.
349. sh_quote(){
350.     v=$(echo "$1" | sed "s/'/'\\''/g")
351.     test "x$v" = "x${v#[!A-Za-z0-9_./+]} " || v="'$v'"
352.     echo "$v"
353. }
354.
355. cleanws(){
356.     echo "$@" | sed 's/^ *//;s/ */ /g;s/ *$//'
357. }
358.
359. filter(){
360.     pat=$1
361.     shift
362.     for v; do
363.         eval "case $v in $pat) echo $v ;; esac"
364.     done
365. }
366.
367. filter_out(){
368.     pat=$1
369.     shift
370.     for v; do
371.         eval "case $v in $pat) ;; *) echo $v ;; esac"
372.     done
373. }
374.
375. map(){
376.     m=$1
377.     shift
378.     for v; do eval $m; done
379. }
380.
381. #第一个参数为值，后面的参数为变量
382. set_all(){
383.     value=$1
384.     shift
385.     for var in $*; do
386.         eval $var=$value
387.     done
388. }
389.

```

```

390. set_weak(){
391.     value=$1
392.     shift
393.     for var; do
394.         eval : \${$var:=$value}
395.     done
396. }
397.
398. set_safe(){
399.     var=$1
400.     shift
401.     eval $(echo "$var" | sed 's/[^A-Za-z0-9_]/_/g')='$*'
402. }
403.
404. get_safe(){
405.     eval echo \${$(echo "$1" | sed 's/[^A-Za-z0-9_]/_/g')}
406. }
407.
408. pushvar(){
409.     for var in $*; do
410.         eval level=\${${var}_level:=0}
411.         eval ${var}_${level}="\${$var}"
412.         eval ${var}_level=$((level+1))
413.     done
414. }
415.
416. popvar(){
417.     for var in $*; do
418.         eval level=\${${var}_level:-0}
419.         test $level = 0 && continue
420.         eval level=$((level-1))
421.         eval $var="\${${var}_${level}}"
422.         eval ${var}_level=$level
423.         eval unset ${var}_${level}
424.     done
425. }
426. #把所有输入参数的值设置为"yes"
427. enable(){
428.     set_all yes $*
429. }
430. #把所有输入参数的值设置为"no"
431. disable(){
432.     set_all no $*
433. }
434.
435. enable_weak(){
436.     set_weak yes $*
437. }
438.
439. disable_weak(){
440.     set_weak no $*
441. }
442.
443. enable_safe(){
444.     for var; do
445.         enable $(echo "$var" | sed 's/[^A-Za-z0-9_]/_/g')
446.     done
447. }
448.
449. disable_safe(){
450.     for var; do
451.         disable $(echo "$var" | sed 's/[^A-Za-z0-9_]/_/g')
452.     done
453. }
454.
455. do_enable_deep(){
456.     for var; do
457.         enabled $var && continue
458.         eval sel="\${${var}_select}"
459.         eval sgs="\${${var}_suggest}"
460.         pushvar var sgs
461.         enable_deep $sel
462.         popvar sgs
463.         enable_deep_weak $sgs
464.         popvar var
465.     done
466. }
467.
468. enable_deep(){
469.     do_enable_deep $*
470.     enable $*
471. }
472.
473. enable_deep_weak(){
474.     do_enable_deep $*
475.     enable_weak $*
476. }
477. #是否开后该组件
478. #主要通过看该组件的取值为"yes"还是"no"
479. #示例 enabled yasm
480. enabled(){

```

```

481. # "#"代表删除从前往后最小匹配的内容
482. # 在这里，即删除"!", 去掉${1}的"!"?
483. # 这句话的意思可能是，检查去掉"!"之后，${1}是否相等？即检查${1}中是否包含"!"
484. # PS：一般很少有包含"!"的情况吧 == !=
485. # 如果包含"!", op取值为"=", 否则取值"!="
486. test "${1#!}" = "$1" && op== || op!=
487. # 进行比较，看看取值是否为"yes"
488. # 为什么要在两边的前面都加一个"x"?
489. eval test "x\${1#!}" $op "xyes"
490. }
491. #是否关闭该组件
492. #和enabled相对应
493. disabled(){
494.     test "${1#!}" = "$1" && op== || op!=
495.     #看看取值是否为"no"
496.     eval test "x\${1#!}" $op "xno"
497. }
498.
499. enabled_all(){
500.     for opt; do
501.         enabled $opt || return 1
502.     done
503. }
504.
505. disabled_all(){
506.     for opt; do
507.         disabled $opt || return 1
508.     done
509. }
510.
511. enabled_any(){
512.     for opt; do
513.         enabled $opt && return 0
514.     done
515. }
516.
517. disabled_any(){
518.     for opt; do
519.         disabled $opt && return 0
520.     done
521.     return 1
522. }
523. #设置默认值
524. set_default(){
525.     for opt; do
526.         eval : \${sopt:=\${opt}_default}
527.     done
528. }
529.
530. is_in(){
531.     value=$1
532.     shift
533.     for var in $*; do
534.         [ $var = $value ] && return 0
535.     done
536.     return 1
537. }
538.
539. check_deps(){
540.     for cfg; do
541.         cfg="\${cfg#!}"
542.         enabled ${cfg}_checking && die "Circular dependency for $cfg."
543.         disabled ${cfg}_checking && continue
544.         enable ${cfg}_checking
545.
546.         eval dep_all="\${cfg}_deps"
547.         eval dep_any="\${cfg}_deps_any"
548.         eval dep_sel="\${cfg}_select"
549.         eval dep_sgs="\${cfg}_suggest"
550.         eval dep_ifa="\${cfg}_if"
551.         eval dep_ifn="\${cfg}_if_any"
552.
553.         pushvar cfg dep_all dep_any dep_sel dep_sgs dep_ifa dep_ifn
554.         check_deps $dep_all $dep_any $dep_sel $dep_sgs $dep_ifa $dep_ifn
555.         popvar cfg dep_all dep_any dep_sel dep_sgs dep_ifa dep_ifn
556.
557.         [ -n "$dep_ifa" ] && { enabled_all $dep_ifa && enable_weak $cfg; }
558.         [ -n "$dep_ifn" ] && { enabled_any $dep_ifn && enable_weak $cfg; }
559.         enabled_all $dep_all || disable $cfg
560.         enabled any $dep_any || disable $cfg
561.         disabled_any $dep_sel && disable $cfg
562.
563.         if enabled $cfg; then
564.             eval dep_extralibs="\${cfg}_extralibs"
565.             test -n "$dep_extralibs" && add_extralibs $dep_extralibs
566.             enable_deep $dep_sel
567.             enable_deep_weak $dep_sgs
568.         fi
569.
570.         disable ${cfg}_checking
571.     done

```

```

572. }
573. #输出config.h的时候使用
574. #调用示例: print_config_h ffmpeg CONFIG_FFPLAY
575. print_config_h(){
576. #command1 && command2
577. #&&左边的命令 (命令1) 返回真 (即返回0, 成功被执行) 后, &&右边的命令 (命令2) 才能够被执行
578. #command1 || command2
579. #||左边的命令 (命令1) 未执行成功, 那么就执行||右边的命令 (命令2)
580.     enabled $1 && v=1 || v=0
581. #示例: #define CONFIG_FFPLAY 1
582.     echo "#define $2 $v"
583. }
584. #输出config.mak的时候使用
585. print_config_mak(){
586.     enabled $1 && v= || v=!
587.     echo "$v$2=yes"
588. }
589. # 输出config.asm的时候使用
590. print_config_asm(){
591.     enabled $1 && echo "%define $2"
592. }
593. # 输出文本到config.mak, config.h等文件
594. # 该函数的示例调用方法: print_config CONFIG_ "$config_files" $CONFIG_LIST
595. print_config(){
596. # 前缀
597.     pfx=$1
598. # 文件列表
599.     files=$2
600.     # 位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1
601.     shift 2
602.     #for循环中, 当没有in指定列表时, for会默认取命令行参数列表。
603.     #在这里取的就是$CONFIG_LIST 等
604.     for cfg; do
605.         # toupper(): 转换为大写
606.         ucname="$(toupper $cfg)"
607.         # files= config.h config.mak config.asm
608.         # 循环输出
609.         for f in $files; do
610.             # "x#"/"代表去取x的第一个slash之后的所有内容 (不包括slash)
611.             # "#"代表删除从前往后最小匹配的内容
612.             # "f##"."代表去取f的第一个"."之后的所有内容。在这里是"h"、"mak"等
613.             # 在这里print_config_h(), print_config_mak(), print_config_asm()
614.             "print_config_{f##.}" $cfg ${pfx}${ucname} >>$f
615.         done
616.     done
617. }
618.
619. print_enabled(){
620.     test "$1" = -n && end=" " && shift || end="\n"
621.     suf=$1
622.     shift
623.     for v; do
624.         enabled $v && printf "%s$end" "${v}${suf}";
625.     done
626. }
627. # 添加
628. # 示例append config_files "config.asm"
629. append(){
630.     var=$1
631.     shift
632.     # eval命令将会首先扫描命令行进行所有的替换, 然后再执行该命令
633.     # 按照上面的示例, 替换后为 config_files="$config_files config.asm"
634.     eval "$var=\"\${$var} ${var}\""
635. }
636.
637. prepend(){
638.     var=$1
639.     shift
640.     eval "$var=\"${var} ${var}\""
641. }
642.
643. add_cppflags(){
644.     append CPPFLAGS "${filter_cppflags "$@"}
645. }
646.
647. add_cflags(){
648.     append CFLAGS "${filter_cflags "$@"}
649. }
650.
651. add_cxxflags(){
652.     append CXXFLAGS "${filter_cflags "$@"}
653. }
654.
655. add_asflags(){
656.     append ASFLAGS "${filter_asflags "$@"}
657. }
658.
659. add_ldflags(){
660.     append LDFLAGS "$@"
661. }
662.
663. add_ldlibs(){

```

```

663.  add_extralibs(){
664.      prepend extralibs "$@"
665.  }
666.  #2> 代表的是错误输出的重定向
667.  #标准的输入、输出、和错误输出分别表示STDIN STDOUT STDERR 也可以用数字表示 0 1 2
668.  #在这里也就是标准输出STDOUT 和 标准错误输出STDERR 都输入到了$logfile文件
669.  check_cmd(){
670.      log "$@"
671.      "$@" >> $logfile 2>&1
672.  }
673.  #检查CC编译器
674.  check_cc(){
675.      log check_cc "$@"
676.      cat > $TMPC
677.      log_file $TMPC
678.      #很多检查都调用了这个check_cmd
679.      #-c 只编译不链接
680.      check_cmd $cc $CPPFLAGS $CFLAGS "$@" -c -o $TMPO $TMPC
681.  }
682.
683.  check_cxx(){
684.      log check_cxx "$@"
685.      cat > $TMPCPP
686.      log_file $TMPCPP
687.      check_cmd $cxx $CPPFLAGS $CFLAGS $CXXFLAGS "$@" -c -o $TMPO $TMPCPP
688.  }
689.
690.  check_cpp(){
691.      log check_cpp "$@"
692.      cat > $TMPC
693.      log_file $TMPC
694.      #-E选项, 可以让编译器在预处理后停止, 并输出预处理结果。
695.      check_cmd $cc $CPPFLAGS $CFLAGS "$@" -E -o $TMPO $TMPC
696.  }
697.
698.  check_as(){
699.      log check_as "$@"
700.      cat > $TMPC
701.      log_file $TMPC
702.      check_cmd $as $CPPFLAGS $ASFLAGS "$@" -c -o $TMPO $TMPC
703.  }
704.
705.  check_asm(){
706.      log check_asm "$@"
707.      name="$1"
708.      code="$2"
709.      shift 2
710.      disable $name
711.      check as "$@" <<EOF && enable $name
712.  void foo(void){ __asm__ volatile($code); }
713.  EOF
714.  }
715.
716.  check_yasm(){
717.      log check_yasm "$@"
718.      echo "$1" > $TMPS
719.      log_file $TMPS
720.      shift 1
721.      check_cmd $yasmexe $YASMFLAGS "$@" -o $TMPO $TMPS
722.  }
723.
724.  check_ld(){
725.      log check_ld "$@"
726.      type=$1
727.      shift 1
728.      flags=''
729.      libs=''
730.      for f; do
731.          test "${f}" = "${f#-l}" && flags="$flags $f" || libs="$libs $f"
732.      done
733.      check_type $(($filter_cflags $flags)) || return
734.      #编译连接
735.      check_cmd $ld $LDFLAGS $flags -o $TMPE $TMPO $libs $extralibs
736.  }
737.
738.  check_cppflags(){
739.      log check_cppflags "$@"
740.      set -- $(($filter_cppflags "$@"))
741.      check_cc "$@" <<EOF && append CPPFLAGS "$@"
742.  int x;
743.  EOF
744.  }
745.
746.  check_cflags(){
747.      log check_cflags "$@"
748.      set -- $(($filter_cflags "$@"))
749.      check_cc "$@" <<EOF && append CFLAGS "$@"
750.  int x;
751.  EOF
752.  }
753.
754.  check_cxxflags(){

```

```

754. check_cxxflags(){
755.     log check_cxxflags "$@"
756.     set -- $($filter_cflags "$@")
757.     check_cxx "$@" <<EOF && append CXXFLAGS "$@"
758.     int x;
759.     EOF
760. }
761.
762. test_ldflags(){
763.     log test_ldflags "$@"
764.     check_ld "cc" "$@" <<EOF
765.     int main(void){ return 0; }
766.     EOF
767. }
768.
769. check_ldflags(){
770.     log check_ldflags "$@"
771.     test_ldflags "$@" && add_ldflags "$@"
772. }
773. #检查头文件
774. #生成一个简单的源代码文件
775. check_header(){
776.     log check_header "$@"
777.     header=$1
778.     shift
779.     disable_safe $header
780.     check_cpp "$@" <<EOF && enable_safe $header
781.     #include <$header>
782.     int x;
783.     EOF
784. }
785.
786. check_func(){
787.     log check_func "$@"
788.     func=$1
789.     shift
790.     disable $func
791.     check_ld "cc" "$@" <<EOF && enable $func
792.     extern int $func();
793.     int main(void){ $func(); }
794.     EOF
795. }
796. #检查数学函数
797. check_mathfunc(){
798.     log check_mathfunc "$@"
799.     #数学函数名称
800.     func=$1
801.     shift
802.     disable $func
803.     check_ld "cc" "$@" <<EOF && enable $func
804.     #include <math.h>
805.     float foo(float f) { return $func(f); }
806.     int main(void){ return (int) foo; }
807.     EOF
808. }
809.
810. check_func_headers(){
811.     log check_func_headers "$@"
812.     headers=$1
813.     funcs=$2
814.     shift 2
815.     {
816.         for hdr in $headers; do
817.             echo "#include <$hdr>"
818.         done
819.         for func in $funcs; do
820.             echo "long check_$func(void) { return (long) $func; }"
821.         done
822.         echo "int main(void) { return 0; }"
823.     } | check_ld "cc" "$@" && enable $funcs && enable_safe $headers
824. }
825.
826. check_class_headers_cpp(){
827.     log check_class_headers_cpp "$@"
828.     headers=$1
829.     classes=$2
830.     shift 2
831.     {
832.         for hdr in $headers; do
833.             echo "#include <$hdr>"
834.         done
835.         echo "int main(void) { "
836.         i=1
837.         for class in $classes; do
838.             echo "$class obj$i;"
839.             i=$(expr $i + 1)
840.         done
841.         echo "return 0; }"
842.     } | check_ld "cxx" "$@" && enable $funcs && enable_safe $headers
843. }
844.
845. check_cpp_condition(){

```

```

846.     log check_cpp_condition "$@"
847.     header=$1
848.     condition=$2
849.     shift 2
850.     check_cpp "${filter_cppflags}" "$@" <<EOF
851. #include <$header>
852. #if !($condition)
853. #error "unsatisfied condition: $condition"
854. #endif
855. EOF
856. }
857. #检查类库
858. check_lib(){
859.     log check_lib "$@"
860.     header="$1"
861.     func="$2"
862.     shift 2
863.     check_header $header && check_func $func "$@" && add_extralibs "$@"
864. }
865.
866. check_lib2(){
867.     log check_lib2 "$@"
868.     headers="$1"
869.     funcs="$2"
870.     shift 2
871.     check_func_headers "$headers" "$funcs" "$@" && add_extralibs "$@"
872. }
873.
874. check_lib_cpp(){
875.     log check_lib_cpp "$@"
876.     headers="$1"
877.     classes="$2"
878.     shift 2
879.     check_class_headers_cpp "$headers" "$classes" "$@" && add_extralibs "$@"
880. }
881.
882. check_pkg_config(){
883.     log check_pkg_config "$@"
884.     pkg="$1"
885.     headers="$2"
886.     funcs="$3"
887.     shift 3
888.     $pkg_config --exists $pkg 2>/dev/null || return
889.     pkg_cflags=$( $pkg_config --cflags $pkg )
890.     pkg_libs=$( $pkg_config --libs $pkg )
891.     check_func_headers "$headers" "$funcs" $pkg_cflags $pkg_libs "$@" &&
892.         set_safe ${pkg}_cflags $pkg_cflags &&
893.         set_safe ${pkg}_libs $pkg_libs
894. }
895.
896. check_exec(){
897.     check_ld "cc" "$@" && { enabled cross_compile || $TMPE >> $logfile 2>&1; }
898. }
899.
900. check_exec_crash(){
901.     code=$(cat)
902.
903.     # exit() is not async signal safe. _Exit (C99) and _exit (POSIX)
904.     # are safe but may not be available everywhere. Thus we use
905.     # raise(SIGTERM) instead. The check is run in a subshell so we
906.     # can redirect the "Terminated" message from the shell. SIGBUS
907.     # is not defined by standard C so it is used conditionally.
908.
909.     (check_exec "$@" >> $logfile 2>&1 <<EOF
910. #include <signal.h>
911. static void sighandler(int sig){
912.     raise(SIGTERM);
913. }
914. int func(void){
915.     $code
916. }
917. int main(void){
918.     signal(SIGILL, sighandler);
919.     signal(SIGFPE, sighandler);
920.     signal(SIGSEGV, sighandler);
921. #ifdef SIGBUS
922.     signal(SIGBUS, sighandler);
923. #endif
924.     return func();
925. }
926. EOF
927. }
928.
929. check_type(){
930.     log check_type "$@"
931.     headers=$1
932.     type=$2
933.     shift 2
934.     disable_safe "$type"
935.     incs=""
936.     for hdr in $headers; do

```

```

937.         incs="$incs
938. #include <$hdr>"
939.         done
940.         check_cc "$@" <<EOF && enable_safe "$type"
941. $incs
942. $type v;
943. EOF
944. }
945.
946. check_struct(){
947.     log check_type "$@"
948.     headers=$1
949.     struct=$2
950.     member=$3
951.     shift 3
952.     disable_safe "${struct}_${member}"
953.     incs=""
954.     for hdr in $headers; do
955.         incs="$incs
956. #include <$hdr>"
957.         done
958.         check_cc "$@" <<EOF && enable_safe "${struct}_${member}"
959. $incs
960. const void *p = &(($struct *)0)->$member;
961. EOF
962. }
963. #检查依赖项的时候使用
964. require(){
965.     name="$1"
966.     header="$2"
967.     func="$3"
968.     shift 3
969.     check_lib $header $func "$@" || die "ERROR: $name not found"
970. }
971.
972. require2(){
973.     name="$1"
974.     headers="$2"
975.     func="$3"
976.     shift 3
977.     check_lib2 "$headers" $func "$@" || die "ERROR: $name not found"
978. }
979.
980. require_cpp(){
981.     name="$1"
982.     headers="$2"
983.     classes="$3"
984.     shift 3
985.     check_lib_cpp "$headers" "$classes" "$@" || die "ERROR: $name not found"
986. }
987.
988. require_pkg_config(){
989.     pkg="$1"
990.     check_pkg_config "$@" || die "ERROR: $pkg not found"
991.     add_cflags $(get_safe ${pkg}_cflags)
992.     add_extralibs $(get_safe ${pkg}_libs)
993. }
994.
995. check_host_cc(){
996.     log check_host_cc "$@"
997.     cat > $TMPC
998.     log_file $TMPC
999.     check_cmd $host_cc $host_cflags "$@" -c -o $TMPO $TMPC
1000. }
1001.
1002. check_host_cflags(){
1003.     log check_host_cflags "$@"
1004.     check_host_cc "$@" <<EOF && append host_cflags "$@"
1005. int x;
1006. EOF
1007. }
1008.
1009. apply(){
1010.     file=$1
1011.     shift
1012.     "$@" < "$file" > "$file.tmp" && mv "$file.tmp" "$file" || rm "$file.tmp"
1013. }
1014. #比较两个文件${1}和${2}，如果两个文件发生了变化，则将${1}强制覆盖${2}
1015. #该函数主要用于生成config.h
1016. cp_if_changed(){
1017.     #cmp是二进制文件比较命令
1018.     #-s：只返回退出值。值0（真）指示相同的文件；值1（假）指示不同的文件；值 2 指示不可访问的文件或缺少选项。
1019.     cmp -s "$1" "$2" && echo "$2 is unchanged" && return
1020.     mkdir -p "$(dirname $2)"
1021.     cp -f "$1" "$2"
1022. }
1023.
1024. # CONFIG_LIST contains configurable options, while HAVE_LIST is for
1025. # system-dependent things.
1026. #各种List
1027. COMPONENT_LIST="

```



```
1028. bsfs
1029. decoders
1030. demuxers
1031. encoders
1032. filters
1033. hwaccels
1034. indevs
1035. muxers
1036. outdevs
1037. parsers
1038. protocols
1039. "
1040.
1041. PROGRAM_LIST="
1042.     ffplay
1043.     ffprobe
1044.     ffserver
1045.     ffmpeg
1046. "
1047.
1048. CONFIG_LIST="
1049. #组件
1050.     $COMPONENT_LIST
1051. #可执行程序
1052.     $PROGRAM_LIST
1053.     avplay
1054.     avprobe
1055.     avserver
1056.     aandct
1057.     ac3dsp
1058.     avcodec
1059.     avdevice
1060.     avfilter
1061.     avformat
1062.     avisynth
1063.     bzlib
1064.     crystalhd
1065.     dct
1066.     doc
1067.     dwf
1068.     dxva2
1069.     fastdiv
1070.     fft
1071.     frei0r
1072.     gnutls
1073.     golomb
1074.     gpl
1075.     gray
1076.     h264chroma
1077.     h264dsp
1078.     h264pred
1079.     hardcoded_tables
1080.     huffman
1081.     libaacplus
1082.     libass
1083.     libcdio
1084.     libcelt
1085.     libdc1394
1086.     libdirac
1087.     libfaac
1088.     libfreetype
1089.     libgsm
1090.     libmodplug
1091.     libmp3lame
1092.     libnut
1093.     libopencore_amrnb
1094.     libopencore_amrwb
1095.     libopencl
1096.     libopenjpeg
1097.     libpulse
1098.     librtmp
1099.     libschroedinger
1100.     libspeex
1101.     libstagefright_h264
1102.     libtheora
1103.     libutvideo
1104.     libv4l2
1105.     libvo_aacenc
1106.     libvo_amrwbenc
1107.     libvorbis
1108.     libvpx
1109.     libx264
1110.     libxavs
1111.     libxvid
1112.     lpc
1113.     lsp
1114.     mdct
1115.     memalign_hack
1116.     mlib
1117.     mpegauddsp
1118.     network
```

```

1119. nonfree
1120. openal
1121. openssl
1122. pic
1123. postproc
1124. rdtft
1125. rtpdec
1126. runtime_cpudetect
1127. safe_bitstream_reader
1128. shared
1129. sinewin
1130. small
1131. sram
1132. static
1133. swresample
1134. swscale
1135. swscale_alpha
1136. thumb
1137. vaapi
1138. vda
1139. vdpau
1140. version3
1141. x11grab
1142. zlib
1143. "
1144.
1145. THREADS_LIST='
1146. pthreads
1147. w32threads
1148. os2threads
1149. '
1150.
1151. ARCH_LIST='
1152. alpha
1153. arm
1154. avr32
1155. avr32_ap
1156. avr32_uc
1157. bfin
1158. ia64
1159. m68k
1160. mips
1161. mips64
1162. parisc
1163. ppc
1164. ppc64
1165. s390
1166. sh4
1167. sparc
1168. sparc64
1169. tomi
1170. x86
1171. x86_32
1172. x86_64
1173. '
1174.
1175. ARCH_EXT_LIST='
1176. altivec
1177. amd3dnow
1178. amd3dnowext
1179. armv5te
1180. armv6
1181. armv6t2
1182. armvfp
1183. avx
1184. iwmmxt
1185. mmi
1186. mmx
1187. mmx2
1188. neon
1189. ppc4xx
1190. sse
1191. ssse3
1192. vfpv3
1193. vis
1194. '
1195.
1196. HAVE_LIST_PUB='
1197. bigendian
1198. fast_unaligned
1199. '
1200.
1201. HAVE_LIST="
1202. $ARCH_EXT_LIST
1203. $HAVE_LIST_PUB
1204. $THREADS_LIST
1205. aligned_stack
1206. alsa_asoundlib_h
1207. altivec_h
1208. arpa_inet_h
1209. asm_mod_y
1210.

```

1210.	asm_types_n
1211.	attribute_may_alias
1212.	attribute_packed
1213.	cbtft
1214.	closesocket
1215.	cmov
1216.	dcbl
1217.	dev_bktr_ioctl_bt848_h
1218.	dev_bktr_ioctl_meteor_h
1219.	dev_ic_bt8xx_h
1220.	dev_video_bktr_ioctl_bt848_h
1221.	dev_video_meteor_ioctl_meteor_h
1222.	dlfcn_h
1223.	dlopen
1224.	dos_paths
1225.	ebp_available
1226.	ebx_available
1227.	exp2
1228.	exp2f
1229.	fast_64bit
1230.	fast_clz
1231.	fast_cmov
1232.	fcntl
1233.	fork
1234.	getaddrinfo
1235.	gethrtime
1236.	GetProcessAffinityMask
1237.	GetProcessMemoryInfo
1238.	GetProcessTimes
1239.	getrusage
1240.	gnu_asm
1241.	ibm_asm
1242.	inet_aton
1243.	inline_asm
1244.	isatty
1245.	kbhit
1246.	ldbrx
1247.	llrint
1248.	llrintf
1249.	local_aligned_16
1250.	local_aligned_8
1251.	localtime_r
1252.	log2
1253.	log2f
1254.	loongson
1255.	lrint
1256.	lrintf
1257.	lzolx_999_compress
1258.	machine_ioctl_bt848_h
1259.	machine_ioctl_meteor_h
1260.	makeinfo
1261.	malloc_h
1262.	MapViewOfFile
1263.	memalign
1264.	mkstemp
1265.	mmap
1266.	PeekNamedPipe
1267.	poll_h
1268.	posix_memalign
1269.	round
1270.	roundf
1271.	sched_getaffinity
1272.	sdl
1273.	sdl_video_size
1274.	setmode
1275.	setrlimit
1276.	sndio_h
1277.	socklen_t
1278.	soundcard_h
1279.	strerror_r
1280.	strptime
1281.	struct_addrinfo
1282.	struct_ipv6_mreq
1283.	struct_rusage_ru_maxrss
1284.	struct_sockaddr_in6
1285.	struct_sockaddr_sa_len
1286.	struct_sockaddr_storage
1287.	struct_v4l2_frmivalenum_discrete
1288.	symver
1289.	symver_asm_label
1290.	symver_gnu_asm
1291.	sysconf
1292.	sysctl
1293.	sys_mman_h
1294.	sys_param_h
1295.	sys_resource_h
1296.	sys_select_h
1297.	sys_soundcard_h
1298.	sys_videoio_h
1299.	termios_h
1300.	threads
1301.	trunc

```

1302. trunc
1303. vfp_args
1304. VirtualAlloc
1305. winsock2_h
1306. xform_asm
1307. xmm_clobbers
1308. yasm
1309. "
1310.
1311. # options emitted with CONFIG_prefix but not available on command line
1312. CONFIG_EXTRA="
1313.     avutil
1314.     gplv3
1315.     lgplv3
1316. "
1317.
1318. CMDLINE_SELECT="
1319.     $ARCH_EXT_LIST
1320.     $CONFIG_LIST
1321.     $THREADS_LIST
1322.     asm
1323.     coverage
1324.     cross_compile
1325.     debug
1326.     extra_warnings
1327.     logging
1328.     optimizations
1329.     stripping
1330.     symver
1331.     yasm
1332. "
1333.
1334. PATHS_LIST='
1335.     bindir
1336.     datadir
1337.     incdir
1338.     libdir
1339.     mandir
1340.     prefix
1341.     shlibdir
1342. '
1343.
1344. CMDLINE_SET="
1345.     $PATHS_LIST
1346.     ar
1347.     arch
1348.     as
1349.     build_suffix
1350.     progs_suffix
1351.     cc
1352.     cpu
1353.     cross_prefix
1354.     cxx
1355.     dep_cc
1356.     extra_version
1357.     host_cc
1358.     host_cflags
1359.     host_ldflags
1360.     host_libs
1361.     host_os
1362.     install
1363.     ld
1364.     logfile
1365.     malloc_prefix
1366.     nm
1367.     optflags
1368.     pkg_config
1369.     samples
1370.     strip
1371.     sysinclude
1372.     sysroot
1373.     target_exec
1374.     target_os
1375.     target_path
1376.     postproc_version
1377.     valgrind
1378.     yasmexe
1379. "
1380.
1381. CMDLINE_APPEND="
1382.     extra_cflags
1383.     extra_cxxflags
1384. "
1385.
1386. # code dependency declarations
1387.
1388. # architecture extensions
1389.
1390. armv5te_deps="arm"
1391. armv6_deps="arm"
1392. armv6t2_deps="arm"

```

```

1393. armvfp_deps="arm"
1394. iwmmt_deps="arm"
1395. neon_deps="arm"
1396. vfpv3_deps="armvfp"
1397.
1398. mmi_deps="mips"
1399.
1400. altivec_deps="ppc"
1401. ppc4xx_deps="ppc"
1402.
1403. vis_deps="sparc"
1404.
1405. x86_64_suggest="cmov fast_cmov"
1406. amd3dnow_deps="mmx"
1407. amd3dnowext_deps="amd3dnow"
1408. mmx_deps="x86"
1409. mmx2_deps="mmx"
1410. sse_deps="mmx"
1411. ssse3_deps="sse"
1412. avx_deps="ssse3"
1413.
1414. aligned_stack_if_any="ppc x86"
1415. fast_64bit_if_any="alpha ia64 mips64 parisc64 ppc64 sparc64 x86_64"
1416. fast_clz_if_any="alpha armv5te avr32 mips ppc x86"
1417. fast_unaligned_if_any="armv6 ppc x86"
1418.
1419. inline_asm_deps="!tms470"
1420. need_memalign="altivec neon sse"
1421.
1422. symver_if_any="symver_asm_label symver_gnu_asm"
1423.
1424. # subsystems
1425. dct_select="rdft"
1426. mdct_select="fft"
1427. rdft_select="fft"
1428. mpegaudioldsp_select="dct"
1429.
1430. # decoders / encoders / hardware accelerators
1431. aac_decoder_select="mdct sinewin"
1432. aac_encoder_select="mdct sinewin"
1433. aac_latm_decoder_select="aac_decoder aac_latm_parser"
1434. ac3_decoder_select="mdct ac3dsp ac3_parser"
1435. ac3_encoder_select="mdct ac3dsp"
1436. ac3_fixed_encoder_select="mdct ac3dsp"
1437. alac_encoder_select="lpc"
1438. amrnb_decoder_select="lsp"
1439. amrwb_decoder_select="lsp"
1440. atrac1_decoder_select="mdct sinewin"
1441. atrac3_decoder_select="mdct"
1442. binkaudio_dct_decoder_select="mdct rdft dct sinewin"
1443. binkaudio_rdft_decoder_select="mdct rdft sinewin"
1444. cavs_decoder_select="golomb"
1445. cook_decoder_select="mdct sinewin"
1446. cscd_decoder_suggest="zlib"
1447. dca_decoder_select="mdct"
1448. dnxhd_encoder_select="aandct"
1449. dxa_decoder_select="zlib"
1450. eac3_decoder_select="ac3_decoder"
1451. eac3_encoder_select="mdct ac3dsp"
1452. eamad_decoder_select="aandct"
1453. eatgq_decoder_select="aandct"
1454. eatqi_decoder_select="aandct"
1455. ffv1_decoder_select="golomb"
1456. flac_decoder_select="golomb"
1457. flac_encoder_select="golomb lpc"
1458. flashsv_decoder_select="zlib"
1459. flashsv_encoder_select="zlib"
1460. flashsv2_encoder_select="zlib"
1461. flashsv2_decoder_select="zlib"
1462. flv_decoder_select="h263_decoder"
1463. flv_encoder_select="h263_encoder"
1464. fraps_decoder_select="huffman"
1465. h261_encoder_select="aandct"
1466. h263_decoder_select="h263_parser"
1467. h263_encoder_select="aandct"
1468. h263_vaapi_hwaccel_select="vaapi h263_decoder"
1469. h263i_decoder_select="h263_decoder"
1470. h263p_encoder_select="h263_encoder"
1471. h264_crystalhd_decoder_select="crystalhd h264_mp4toannex_bsf h264_parser"
1472. h264_decoder_select="golomb h264chroma h264dsp h264pred"
1473. h264_dxva2_hwaccel_deps="dxva2api h"
1474. h264_dxva2_hwaccel_select="dxva2 h264_decoder"
1475. h264_vaapi_hwaccel_select="vaapi h264_decoder"
1476. h264_vda_hwaccel_deps="VideoDecodeAcceleration_VDADecoder_h pthreads"
1477. h264_vda_hwaccel_select="vda h264_decoder"
1478. h264_vdpau_decoder_select="vdpau h264_decoder"
1479. imc_decoder_select="fft mdct sinewin"
1480. jpegls_decoder_select="golomb"
1481. jpegls_encoder_select="golomb"
1482. ljpeg_encoder_select="aandct"
1483. loco_decoder_select="golomb"

```

```
1484.  mjpeg_encoder_select="aandct"
1485.  mlp_decoder_select="mlp_parser"
1486.  mp1_decoder_select="mpegaudiiodsp"
1487.  mp1float_decoder_select="mpegaudiiodsp"
1488.  mp2_decoder_select="mpegaudiiodsp"
1489.  mp2float_decoder_select="mpegaudiiodsp"
1490.  mp3_decoder_select="mpegaudiiodsp"
1491.  mp3adu_decoder_select="mpegaudiiodsp"
1492.  mp3adufloat_decoder_select="mpegaudiiodsp"
1493.  mp3float_decoder_select="mpegaudiiodsp"
1494.  mp3on4_decoder_select="mpegaudiiodsp"
1495.  mp3on4float_decoder_select="mpegaudiiodsp"
1496.  mpc7_decoder_select="mpegaudiiodsp"
1497.  mpc8_decoder_select="mpegaudiiodsp"
1498.  mpeg_vdpau_decoder_select="vdpau mpegvideo_decoder"
1499.  mpeg_xvmc_decoder_deps="X11_extensions_XvMCLib_h"
1500.  mpeg_xvmc_decoder_select="mpegvideo_decoder"
1501.  mpeg1_vdpau_decoder_select="vdpau mpeg1video_decoder"
1502.  mpeg1_vdpau_hwaccel_select="vdpau mpeg1video_decoder"
1503.  mpeg1video_encoder_select="aandct"
1504.  mpeg2_crystalhd_decoder_select="crystalhd"
1505.  mpeg2_dxva2_hwaccel_deps="dxva2api_h"
1506.  mpeg2_dxva2_hwaccel_select="dxva2 mpeg2video_decoder"
1507.  mpeg2_vdpau_hwaccel_select="vdpau mpeg2video_decoder"
1508.  mpeg2_vaapi_hwaccel_select="vaapi mpeg2video_decoder"
1509.  mpeg2video_encoder_select="aandct"
1510.  mpeg4_crystalhd_decoder_select="crystalhd"
1511.  mpeg4_decoder_select="h263_decoder mpeg4video_parser"
1512.  mpeg4_encoder_select="h263_encoder"
1513.  mpeg4_vaapi_hwaccel_select="vaapi mpeg4_decoder"
1514.  mpeg4_vdpau_decoder_select="vdpau mpeg4_decoder"
1515.  msmpeg4_crystalhd_decoder_select="crystalhd"
1516.  msmpeg4v1_decoder_select="h263_decoder"
1517.  msmpeg4v1_encoder_select="h263_encoder"
1518.  msmpeg4v2_decoder_select="h263_decoder"
1519.  msmpeg4v2_encoder_select="h263_encoder"
1520.  msmpeg4v3_decoder_select="h263_decoder"
1521.  msmpeg4v3_encoder_select="h263_encoder"
1522. nellymoser_decoder_select="mdct sinewin"
1523.  nellymoser_encoder_select="mdct sinewin"
1524.  png_decoder_select="zlib"
1525.  png_encoder_select="zlib"
1526.  qcelp_decoder_select="lsp"
1527.  qdm2_decoder_select="mdct rdft mpegaudiiodsp"
1528.  ra_144_encoder_select="lpc"
1529.  rv10_decoder_select="h263_decoder"
1530.  rv10_encoder_select="h263_encoder"
1531.  rv20_decoder_select="h263_decoder"
1532.  rv20_encoder_select="h263_encoder"
1533.  rv30_decoder_select="golomb h264chroma h264pred"
1534.  rv40_decoder_select="golomb h264chroma h264pred"
1535.  shorten_decoder_select="golomb"
1536.  sipr_decoder_select="lsp"
1537.  snow_decoder_select="dwt"
1538.  snow_encoder_select="aandct dwt"
1539.  sonic_decoder_select="golomb"
1540.  sonic_encoder_select="golomb"
1541.  sonic_ls_encoder_select="golomb"
1542.  svq1_encoder_select="aandct"
1543.  svq3_decoder_select="golomb h264chroma h264dsp h264pred"
1544.  svq3_decoder_suggest="zlib"
1545.  theora_decoder_select="vp3_decoder"
1546.  tiff_decoder_suggest="zlib"
1547.  tiff_encoder_suggest="zlib"
1548.  truehd_decoder_select="mlp_decoder"
1549.  tsccl_decoder_select="zlib"
1550.  twinvq_decoder_select="mdct lsp sinewin"
1551.  vc1_crystalhd_decoder_select="crystalhd"
1552.  vc1_decoder_select="h263_decoder h264chroma"
1553.  vc1_dxva2_hwaccel_deps="dxva2api_h"
1554.  vc1_dxva2_hwaccel_select="dxva2 vc1_decoder"
1555.  vc1_vaapi_hwaccel_select="vaapi vc1_decoder"
1556.  vc1_vdpau_decoder_select="vdpau vc1_decoder"
1557.  vc1image_decoder_select="vc1_decoder"
1558.  vorbis_decoder_select="mdct"
1559.  vorbis_encoder_select="mdct"
1560.  vp6_decoder_select="huffman"
1561.  vp6a_decoder_select="vp6_decoder"
1562.  vp6f_decoder_select="vp6_decoder"
1563.  vp8_decoder_select="h264pred"
1564.  wmapro_decoder_select="mdct sinewin"
1565.  wav1_decoder_select="mdct sinewin"
1566.  wav1_encoder_select="mdct sinewin"
1567.  wav2_decoder_select="mdct sinewin"
1568.  wav2_encoder_select="mdct sinewin"
1569.  wavoice_decoder_select="lsp rdft dct mdct sinewin"
1570.  wmv1_decoder_select="h263_decoder"
1571.  wmv1_encoder_select="h263_encoder"
1572.  wmv2_decoder_select="h263_decoder"
1573.  wmv2_encoder_select="h263_encoder"
1574.  wmv3_decoder_select="vc1_decoder"
```

```
1575. wmv3_crystalhd_decoder_select="crystalhd"
1576. wmv3_dxva2_hwaccel_select="vc1_dxva2_hwaccel"
1577. wmv3_vaapi_hwaccel_select="vc1_vaapi_hwaccel"
1578. wmv3_vdpau_decoder_select="vc1_vdpau_decoder"
1579. wmv3image_decoder_select="wmv3_decoder"
1580. zlib_decoder_select="zlib"
1581. zlib_encoder_select="zlib"
1582. zmbv_decoder_select="zlib"
1583. zmbv_encoder_select="zlib"
1584.
1585. crystalhd_deps="libcrystalhd libcrystalhd_if_h"
1586. vaapi_deps="va_va_h"
1587. vda_deps="VideoDecodeAcceleration_VDADecoder_h pthreads"
1588. vdpau_deps="vdpau_vdpau_h vdpau_vdpau_x11_h"
1589.
1590. # parsers
1591. h264_parser_select="golomb h264chroma h264dsp h264pred"
1592.
1593. # external libraries
1594. libaacplus_encoder_deps="libaacplus"
1595. libcelt_decoder_deps="libcelt"
1596. libdirac_decoder_deps="libdirac !libschrödinger"
1597. libdirac_encoder_deps="libdirac"
1598. libfaac_encoder_deps="libfaac"
1599. libgsm_decoder_deps="libgsm"
1600. libgsm_encoder_deps="libgsm"
1601. libgsm_ms_decoder_deps="libgsm"
1602. libgsm_ms_encoder_deps="libgsm"
1603. libmodplug_demuxer_deps="libmodplug"
1604. libmp3lame_encoder_deps="libmp3lame"
1605. libopencore_amrnb_decoder_deps="libopencore_amrnb"
1606. libopencore_amrnb_encoder_deps="libopencore_amrnb"
1607. libopencore_amrwb_decoder_deps="libopencore_amrwb"
1608. libopenjpeg_decoder_deps="libopenjpeg"
1609. libopenjpeg_encoder_deps="libopenjpeg"
1610. libschroedinger_decoder_deps="libschrödinger"
1611. libschroedinger_encoder_deps="libschrödinger"
1612. libspeex_decoder_deps="libspeex"
1613. libspeex_encoder_deps="libspeex"
1614. libstagefright_h264_decoder_deps="libstagefright_h264"
1615. libtheora_encoder_deps="libtheora"
1616. libvo_aacenc_encoder_deps="libvo_aacenc"
1617. libvo_amrwbenc_encoder_deps="libvo_amrwbenc"
1618. libvorbis_encoder_deps="libvorbis"
1619. libvpx_decoder_deps="libvpx"
1620. libvpx_encoder_deps="libvpx"
1621. libx264_encoder_deps="libx264"
1622. libx264rgb_encoder_deps="libx264"
1623. libxavs_encoder_deps="libxavs"
1624. libxvid_encoder_deps="libxvid"
1625. libutvideo_decoder_deps="libutvideo gpl"
1626.
1627. # demuxers / muxers
1628. ac3_demuxer_select="ac3_parser"
1629. asf_stream_muxer_select="asf_muxer"
1630. avisynth_demuxer_deps="avisynth"
1631. dirac_demuxer_select="dirac_parser"
1632. eac3_demuxer_select="ac3_parser"
1633. flac_demuxer_select="flac_parser"
1634. ipod_muxer_select="mov_muxer"
1635. libnut_demuxer_deps="libnut"
1636. libnut_muxer_deps="libnut"
1637. matroska_audio_muxer_select="matroska_muxer"
1638. matroska_demuxer_suggest="zlib bzlib"
1639. mov_demuxer_suggest="zlib"
1640. mp3_demuxer_select="mpegaudio_parser"
1641. mp4_muxer_select="mov_muxer"
1642. mpegtsraw_demuxer_select="mpegts_demuxer"
1643. mxf_d10_muxer_select="mxf_muxer"
1644. ogg_demuxer_select="golomb"
1645. psp_muxer_select="mov_muxer"
1646. rtp_demuxer_select="sdp_demuxer"
1647. rtpdec_select="asf_demuxer rm_demuxer rtp_protocol mpegts_demuxer mov_demuxer"
1648. rtsp_demuxer_select="http_protocol rtpdec"
1649. rtsp_muxer_select="rtp_muxer http_protocol rtp_protocol"
1650. sap_demuxer_select="sdp_demuxer"
1651. sap_muxer_select="rtp_muxer rtp_protocol"
1652. sdp_demuxer_select="rtpdec"
1653. spdif_muxer_select="aac_parser"
1654. tg2_muxer_select="mov_muxer"
1655. tgp_muxer_select="mov_muxer"
1656. w64_demuxer_deps="wav_demuxer"
1657.
1658. # indevs / outdevs
1659. alsa_indev_deps="alsa_asoundlib_h snd_pcm_htimestamp"
1660. alsa_outdev_deps="alsa_asoundlib_h"
1661. bktr_indev_deps_any="dev_bktr_ioctl_bt848_h machine_ioctl_bt848_h dev_video_bktr_ioctl_bt848_h dev_ic_bt8xx_h"
1662. dshow_indev_deps="IBaseFilter"
1663. dshow_indev_extralibs="-lpsapi -lole32 -lstrmiids -luuid"
1664. dv1394_indev_deps="dv1394 dv_demuxer"
1665. fbdev_indev_deps="linux_fb_h"
```

```

1666. jack_indev_deps="jack_jack_h sem_timedwait"
1667. lavfi_indev_deps="avfilter"
1668. libcdio_indev_deps="libcdio"
1669. libdc1394_indev_deps="libdc1394"
1670. libv4l2_indev_deps="libv4l2"
1671. openal_indev_deps="openal"
1672. oss_indev_deps_any="soundcard_h sys_soundcard_h"
1673. oss_outdev_deps_any="soundcard_h sys_soundcard_h"
1674. pulse_indev_deps="libpulse"
1675. sdl_outdev_deps="sdl"
1676. sndio_indev_deps="sndio_h"
1677. sndio_outdev_deps="sndio_h"
1678. v4l_indev_deps="linux_videodev_h"
1679. v4l2_indev_deps_any="linux_videodev2_h sys_videoio_h"
1680. vfwcap_indev_deps="capCreateCaptureWindow vfwcap_defines"
1681. vfwcap_indev_extralibs="-lavicap32"
1682. x11_grab_device_indev_deps="x11grab XShmCreateImage"
1683. x11_grab_device_indev_extralibs="-lX11 -lXext -lXfixes"
1684.
1685. # protocols
1686. gopher_protocol_deps="network"
1687. httpproxy_protocol_deps="network"
1688. httpproxy_protocol_select="tcp_protocol"
1689. http_protocol_deps="network"
1690. http_protocol_select="tcp_protocol"
1691. https_protocol_select="tls_protocol"
1692. mmsh_protocol_select="http_protocol"
1693. mmst_protocol_deps="network"
1694. rtmp_protocol_select="tcp_protocol"
1695. rtp_protocol_select="udp_protocol"
1696. tcp_protocol_deps="network"
1697. tls_protocol_deps_any="openssl gnutls"
1698. tls_protocol_select="tcp_protocol"
1699. udp_protocol_deps="network"
1700.
1701. # filters
1702. amovie_filter_deps="avcodec avformat"
1703. ass_filter_deps="libass"
1704. blackframe_filter_deps="gpl"
1705. boxblur_filter_deps="gpl"
1706. cropdetect_filter_deps="gpl"
1707. delogo_filter_deps="gpl"
1708. drawtext_filter_deps="libfreetype"
1709. frei0r_filter_deps="frei0r dlopen"
1710. frei0r_src_filter_deps="frei0r dlopen"
1711. hqdn3d_filter_deps="gpl"
1712. movie_filter_deps="avcodec avformat"
1713. mp_filter_deps="gpl avcodec"
1714. mptestsrc_filter_deps="gpl"
1715. negate_filter_deps="lut_filter"
1716. ocv_filter_deps="libopencv"
1717. pan_filter_deps="swresample"
1718. scale_filter_deps="swscale"
1719. tinterlace_filter_deps="gpl"
1720. yadif_filter_deps="gpl"
1721.
1722. # libraries
1723. avdevice_deps="avcodec avformat"
1724. avformat_deps="avcodec"
1725. postproc_deps="gpl"
1726.
1727. # programs
1728. ffplay_deps="avcodec avformat swscale sdl"
1729. ffplay_select="buffersink filter rdft"
1730. ffprobe_deps="avcodec avformat"
1731. ffserver_deps="avformat ffm_muxer fork rtp_protocol rtsp_demuxer"
1732. ffserver_extralibs='$ldl'
1733. ffmpeg_deps="avcodec avformat swscale swresample"
1734. ffmpeg_select="buffersink_filter"
1735.
1736. doc_deps="texi2html"
1737.
1738. # tests
1739.
1740. test_deps(){
1741.     suf1=$1
1742.     suf2=$2
1743.     shift 2
1744.     for v; do
1745.         dep=${v%*=*}
1746.         tests=${v#*=*}
1747.         for name in ${tests}; do
1748.             append ${name}_test_deps ${dep}$suf1 ${dep}$suf2
1749.         done
1750.     done
1751. }
1752.
1753. mxf_d10_test_deps="avfilter"
1754. seek_lavf_mxf_d10_test_deps="mxf_d10_test"
1755.
1756. test_deps_encoder_decoder \
1757.     -data -ss 0

```



```

1757.      adpcm_ima_qt \
1758.      adpcm_ima_wav \
1759.      adpcm_ms \
1760.      adpcm_swf \
1761.      adpcm_yamaha=adpcm_yam \
1762.      alac \
1763.      asv1 \
1764.      asv2 \
1765.      bmp \
1766.      dnxhd="dnxhd_1080i dnxhd_720p dnxhd_720p_rd" \
1767.      dvvideo="dv dv_411 dv50" \
1768.      ffv1 \
1769.      flac \
1770.      flashsv \
1771.      flv \
1772.      adpcm_g726=g726 \
1773.      gif \
1774.      h261 \
1775.      h263="h263 h263p" \
1776.      huffyuv \
1777.      jpegls \
1778.      mjpeg="jpg mjpeg ljpeg" \
1779.      mp2 \
1780.      mpeg1video="mpeg mpeg1b" \
1781.      mpeg2video="mpeg2 mpeg2_422 mpeg2_idct_int mpeg2_ilace mpeg2_ivlc_qprd" \
1782.      mpeg2video="mpeg2thread mpeg2thread_ilace" \
1783.      mpeg4="mpeg4 mpeg4_adap mpeg4_qpel mpeg4_qprd mpeg4adv mpeg4nr" \
1784.      mpeg4="mpeg4thread error rc" \
1785.      msmpeg4v3=msmpeg4 \
1786.      msmpeg4v2 \
1787.      pbm=pbmpipe \
1788.      pcx \
1789.      pgm="pgm pgmpipe" \
1790.      png \
1791.      ppm="ppm ppmpipe" \
1792.      rawvideo="rgb yuv" \
1793.      roq \
1794.      rv10 \
1795.      rv20 \
1796.      sgi \
1797.      snow="snow snowll" \
1798.      svq1 \
1799.      targa=tga \
1800.      tiff \
1801.      wma_v1 \
1802.      wma_v2 \
1803.      wmv1 \
1804.      wmv2 \
1805.
1806.      test_deps_muxer_demuxer \
1807.      aiff \
1808.      pcm_alaw=alaw \
1809.      asf \
1810.      au \
1811.      avi \
1812.      dv=dv_fmt \
1813.      ffm \
1814.      flv=flv_fmt \
1815.      gxf \
1816.      matroska=mkv \
1817.      mmf \
1818.      mov \
1819.      pcm_mulaw=mulaw \
1820.      mxf="mxf mxf_d10" \
1821.      nut \
1822.      ogg \
1823.      rawvideo=pixfmt \
1824.      rm \
1825.      swf \
1826.      mpegts=ts \
1827.      voc \
1828.      wav \
1829.      yuv4mpegpipe=yuv4mpeg \
1830.
1831.      ac3_fixed_test_deps="ac3_fixed_encoder ac3_decoder rm_muxer rm_demuxer"
1832.      mpg_test_deps="mpeg1system_muxer mpegps_demuxer"
1833.
1834.      # 默认参数 default parameters
1835.      # 日志
1836.      logfile="config.log"
1837.
1838.      # 安装路径 installation paths
1839.      prefix_default="/usr/local"
1840.      bindir_default='${prefix}/bin'
1841.      datadir_default='${prefix}/share/ffmpeg'
1842.      incdir_default='${prefix}/include'
1843.      libdir_default='${prefix}/lib'
1844.      mandir_default='${prefix}/share/man'
1845.      shlibdir_default="$libdir_default"
1846.      postproc_version_default="current"
1847.
1848.      # 工具链 toolchain

```

```

1849. ar_default="ar"
1850. cc_default="gcc"
1851. cxx_default="g++"
1852. cc_version="\unknown\"
1853. host_cc_default="gcc"
1854. install="install"
1855. ln_s="ln -sf"
1856. nm_default="nm"
1857. objformat="elf"
1858. pkg_config_default=pkg-config
1859. ranlib="ranlib"
1860. strip_default="strip"
1861. yasmexe_default="yasm"
1862.
1863. nm_opts='-g'
1864. nogas=":"
1865.
1866. # 机器 machine
1867. arch_default=$(uname -m)
1868. cpu="generic"
1869.
1870. # 操作系统 OS
1871. target_os_default=$(tolower $(uname -s))
1872. host_os=$target_os_default
1873.
1874. # alternative libpostproc version
1875. ALT_PP_VER_MAJOR=51
1876. ALT_PP_VER_MINOR=2
1877. ALT_PP_VER_MICRO=101
1878. ALT_PP_VER=$ALT_PP_VER_MAJOR.$ALT_PP_VER_MINOR.$ALT_PP_VER_MICRO
1879.
1880. # 选项 configurable options
1881. # PROGRAM_LIST内容是 ffmpeg ffprobe ffserver ffmpeg
1882. enable $PROGRAM_LIST
1883.
1884. enable avcodec
1885. enable avdevice
1886. enable avfilter
1887. enable avformat
1888. enable avutil
1889. enable postproc
1890. enable stripping
1891. enable swresample
1892. enable swscale
1893.
1894. enable asm
1895. enable debug
1896. enable doc
1897. enable fastdiv
1898. enable network
1899. enable optimizations
1900. enable safe_bitstream_reader
1901. enable static
1902. enable swscale_alpha
1903.
1904. # 编译选项 build settings
1905. SHFLAGS='-shared -Wl,-soname,$(@F)'
1906. FFSERVERLDFLAGS=-Wl,-E
1907. # 前缀后缀
1908. LIBPREF="lib"
1909. LIBSUF=".a"
1910. FULLNAME='$(NAME)$(BUILDSUF)'
1911. # 名称
1912. LIBNAME='$(LIBPREF)$(FULLNAME)$(LIBSUF)'
1913. # 动态库前缀后缀
1914. SLIBPREF="lib"
1915. SLIBSUF=".so"
1916. # 名称
1917. SLIBNAME='$(SLIBPREF)$(FULLNAME)$(SLIBSUF)'
1918. SLIBNAME_WITH_VERSION='$(SLIBNAME).$(LIBVERSION)'
1919. SLIBNAME_WITH_MAJOR='$(SLIBNAME).$(LIBMAJOR)'
1920. LIB_INSTALL_EXTRA_CMD='$(RANLIB) "$(LIBDIR)/$(LIBNAME)'"
1921. SLIB_INSTALL_NAME='$(SLIBNAME_WITH_VERSION)'
1922. SLIB_INSTALL_LINKS='$(SLIBNAME_WITH_MAJOR) $(SLIBNAME)'
1923.
1924. AS_0='-o $@'
1925. CC_0='-o $@'
1926. CXX_0='-o $@'
1927.
1928. host_cflags='-D ISOC99_SOURCE -O3 -g'
1929. host_libs='-lm'
1930.
1931. target_path='$(CURDIR)'
1932.
1933. # since the object filename is not given with the -MM flag, the compiler
1934. # is only able to print the basename, and we must add the path ourselves
1935. DEPEND_CMD='(DEPCC) $(DEPFLAGS) < | sed -e "/^#.*d" -e "s,^[[:space:]]*$(*)\\.o,$(@D)/$(*)\\.o," > $@:.o=.d)'
1936. DEPFLAGS='$(CPPFLAGS) $(CFLAGS) -MM'
1937.
1938. # find source path
1939. # $0就是该bash文件名

```

```

1940. # dirname /home/lxh/test.txt 输出/home/lxh
1941. if test -f configure; then
1942.     source_path=.
1943. else
1944.     source_path=$(cd $(dirname "$0"); pwd)
1945.     echo "$source_path" | grep -q '[:blank:]' &&
1946.     die "Out of tree builds are impossible with whitespace in source path."
1947.     test -e "$source_path/config.h" &&
1948.     die "Out of tree builds are impossible with config.h in source dir."
1949. fi
1950. # 脚本名称叫test.sh
1951. # 入参三个: 1 2 3
1952. # 运行test.sh 1 2 3后
1953. # $*为"1 2 3" (一起被引号包住)
1954. # @$为"1" "2" "3" (分别被包住)
1955. # $#为3 (参数数量)
1956. for v in "$@"; do
1957.     r=${v#*=}
1958.     l=${v%$r}
1959.     r=$(sh_quote "$r")
1960.     FFMPEG_CONFIGURATION="${FFMPEG_CONFIGURATION# } ${l}${r}"
1961. done
1962. # ${数字} 一般是位置参数的用法。
1963. # 如果运行脚本的时候带参数, 那么可以在脚本里通过 $1 获取第一个参数, $2 获取第二个参数
1964. # 例如以ENCODER_LIST为例, $1为"encoder", $2为"ENC", $3为"libavcodec/allcodecs.c"
1965. find_things(){
1966.     thing=$1
1967.     pattern=$2
1968.     file=$source_path/$3
1969.     # 处理一行字符串?挺复杂
1970.     sed -n 's/^[^#]*$pattern.*([^\,]*, *\([^,]*\)\(,.*\)*).*/\1_$thing/p' "$file"
1971. }
1972. #从allcodecs.c等文件中提取编解码器
1973. ENCODER_LIST=$(find_things encoder ENC libavcodec/allcodecs.c)
1974. DECODER_LIST=$(find_things decoder DEC libavcodec/allcodecs.c)
1975. HWACCEL_LIST=$(find_things hwaccel HWACCEL libavcodec/allcodecs.c)
1976. PARSER_LIST=$(find_things parser PARSER libavcodec/allcodecs.c)
1977. BSF_LIST=$(find_things bsf BSF libavcodec/allcodecs.c)
1978. MUXER_LIST=$(find_things muxer _MUX libavformat/allformats.c)
1979. DEMUXER_LIST=$(find_things demuxer DEMUX libavformat/allformats.c)
1980. OUTDEV_LIST=$(find_things outdev OUTDEV libavdevice/alldevices.c)
1981. INDEV_LIST=$(find_things indev _IN libavdevice/alldevices.c)
1982. PROTOCOL_LIST=$(find_things protocol PROTOCOL libavformat/allformats.c)
1983. FILTER_LIST=$(find_things filter FILTER libavfilter/allfilters.c)
1984.
1985. # 所有组件
1986. ALL_COMPONENTS="
1987.     $BSF_LIST
1988.     $DECODER_LIST
1989.     $DEMUXER_LIST
1990.     $ENCODER_LIST
1991.     $FILTER_LIST
1992.     $HWACCEL_LIST
1993.     $INDEV_LIST
1994.     $MUXER_LIST
1995.     $OUTDEV_LIST
1996.     $PARSER_LIST
1997.     $PROTOCOL_LIST
1998. "
1999.
2000. find_tests(){
2001.     map "echo ${2}\${v}_test" $(ls "$source_path"/tests/ref/$1 | grep -v '[-a-z0-9_]')
2002. }
2003.
2004. ACODEC_TESTS=$(find_tests acodec)
2005. VCODEC_TESTS=$(find_tests vsynth1)
2006. LAVF_TESTS=$(find_tests lavf)
2007. LAVFI_TESTS=$(find_tests lavfi)
2008. SEEK_TESTS=$(find_tests seek seek_)
2009.
2010. ALL_TESTS="$ACODEC_TESTS $VCODEC_TESTS $LAVF_TESTS $LAVFI_TESTS $SEEK_TESTS"
2011.
2012. pcm_test_deps=$(map 'echo ${v%_*}_decoder $v' $(filter pcm_* $ENCODER_LIST))
2013.
2014. for n in $COMPONENT_LIST; do
2015.     v=$(toupper ${n%*})_LIST
2016.     eval enable \${v}
2017.     eval ${n}_if_any="\${v}"
2018. done
2019.
2020. enable $ARCH_EXT_LIST $ALL_TESTS
2021.
2022. die_unknown(){
2023.     echo "Unknown option \"${1}\"."
2024.     echo "See $0 --help for available options."
2025.     exit 1
2026. }
2027.
2028. show_list() {
2029.     suffix=_$1
2030.     shift

```

```

2031.     echo $* | sed s/$suffix//g | tr ' ' '\n' | sort | pr -3 -t
2032.     exit 0
2033. }
2034. # 解析各种各样的选项
2035. #
2036. # case分支语句的格式如下：
2037. #   case $变量名 in
2038. #       模式1)
2039. #       命令序列1
2040. #       ;;
2041. #       模式2)
2042. #       命令序列2
2043. #       ;;
2044. #       *)
2045. #       默认执行的命令序列
2046. #       ;;
2047. #   esac
2048. # case语句结构特点如下：
2049. # case行尾必须为单词“in”，每一个模式必须以右括号“)”结束。
2050. # 双分号“;;”表示命令序列结束。
2051. # 最后的“*)”表示默认模式，当使用前面的各种模式均无法匹配该变量时，将执行“*)”后的命令序列。
2052. #
2053. #注意：opt不是参数列表（实际上也没有看见opt变量的定义）
2054. #原因是处在for循环中，当你没有为in指定列表时，for会默认取命令行参数列表。
2055. #因此“opt”这个名字实际上是可以随便取的
2056. for opt do
2057. # “#”用于去除特定字符前面的字符串
2058. # optval内容为opt去掉“=”以及其前面字符串之后的内容
2059.     optval="${opt#*=}"
2060.     case "$opt" in
2061.         # 不同的选项
2062.         --extra-ldflags=*) add_ldflags $optval
2063.         ;;
2064.         --extra-libs=*) add_extralibs $optval
2065.         ;;
2066.         --disable-devices) disable $INDEV_LIST $OUTDEV_LIST
2067.         ;;
2068.         --enable-debug=*) debuglevel="$optval"
2069.         ;;
2070.         --disable-everything)
2071.         map 'eval unset \${$(toupper ${v%s})}_LIST)' $COMPONENT_LIST
2072.         ;;
2073.         --enable-*=*|--disable-*=*)
2074.         eval $(echo "${opt%*=}" | sed 's/--/action=/;s/-/ thing=/' )
2075.         is_in "${thing}s" $COMPONENT_LIST || die_unknown "$opt"
2076.         eval list=\${$(toupper $thing)_LIST}
2077.         name=$(echo "${optval}" | sed "s/_/${thing}|/g")_${thing}
2078.         $action $(filter "$name" $list)
2079.         ;;
2080.         --enable-?*|--disable-?*)
2081.         eval $(echo "$opt" | sed 's/--/action=/;s/-/ option=/;s/_/_/g')
2082.         if is_in $option $COMPONENT_LIST; then
2083.             test $action = disable && action=unset
2084.             eval $action \${$(toupper ${option%s})}_LIST
2085.         elif is_in $option $CMDLINE_SELECT; then
2086.             $action $option
2087.         else
2088.             die_unknown $opt
2089.         fi
2090.         ;;
2091.         --list-*)
2092.             NAME="${opt#--list-}"
2093.             is_in $NAME $COMPONENT_LIST || die_unknown $opt
2094.             NAME=${NAME}s
2095.             eval show_list $NAME \${$(toupper $NAME)_LIST}
2096.             ;;
2097.         --help|-h) show_help
2098.             ;;
2099.         *)
2100.             #% 就是从右边开始删除符合条件的字符串（符合条件的最短字符串）
2101.             #%%是删除符合条件的的最长的字符串
2102.
2103.             #删除“=”右边的内容
2104.             optname="${opt%*=}"
2105.             #删除左边的“--”
2106.             optname="${optname#--}"
2107.             optname=$(echo "$optname" | sed 's/_/_/g')
2108.             #看看是否在opt列表中，不在的话就会返回错误
2109.             if is_in $optname $CMDLINE_SET; then
2110.                 eval $optname='${optval}'
2111.             elif is_in $optname $CMDLINE_APPEND; then
2112.                 append $optname "$optval"
2113.             else
2114.                 die_unknown $opt
2115.             fi
2116.             ;;
2117.     esac
2118. done
2119.
2120. disabled logging && logfile=/dev/null
2121.

```

```

2122. echo "# $0 $FFMPEG_CONFIGURATION" > $logfile
2123. set >> $logfile
2124.
2125. test -n "$cross_prefix" && enable cross_compile
2126.
2127. if enabled cross_compile; then
2128.     test -n "$arch" && test -n "$target_os" ||
2129.         die "Must specify target arch and OS when cross-compiling"
2130. fi
2131.
2132. set_default arch target_os postproc_version
2133.
2134. # Check if we should build alternative libpostproc version instead of current
2135. if test "$postproc_version" = $ALT_PP_VER; then
2136.     LIBPOSTPROC_VERSION=$ALT_PP_VER
2137.     LIBPOSTPROC_VERSION_MAJOR=$ALT_PP_VER_MAJOR
2138.     LIBPOSTPROC_VERSION_MINOR=$ALT_PP_VER_MINOR
2139.     LIBPOSTPROC_VERSION_MICRO=$ALT_PP_VER_MICRO
2140. elif test "$postproc_version" != current; then
2141.     die "Invalid argument to --postproc-version. See --help output."
2142. fi
2143.
2144. ar_default="${cross_prefix}${ar_default}"
2145. cc_default="${cross_prefix}${cc_default}"
2146. cxx_default="${cross_prefix}${cxx_default}"
2147. nm_default="${cross_prefix}${nm_default}"
2148. pkg_config_default="${cross_prefix}${pkg_config_default}"
2149. ranlib="${cross_prefix}${ranlib}"
2150. strip_default="${cross_prefix}${strip_default}"
2151.
2152. sysinclude_default="${sysroot}/usr/include"
2153.
2154. set_default cc cxx nm pkg_config strip sysinclude yasmexe
2155. enabled cross_compile || host_cc_default=$cc
2156. set_default host_cc
2157.
2158. if ! $pkg_config --version >/dev/null 2>&1; then
2159.     warn "$pkg_config not found, library detection may fail."
2160.     pkg_config=false
2161. fi
2162.
2163. exesuf() {
2164.     case $1 in
2165.         mingw32*|cygwin*|*-dos|freedos|opendos|os/2*|symbian) echo .exe ;;
2166.         esac
2167.     }
2168.
2169. EXESUF=$(exesuf $target_os)
2170. HOSTEXESUF=$(exesuf $host_os)
2171.
2172. # set temporary file name
2173. : ${TMPDIR:=TMPDIR}
2174. : ${TMPDIR:=TMP}
2175. : ${TMPDIR:=/tmp}
2176.
2177. if ! check_cmd mktemp -u XXXXXX; then
2178.     # simple replacement for missing mktemp
2179.     # NOT SAFE FOR GENERAL USE
2180.     mktemp(){
2181.         echo "${2%XXX*}.${HOSTNAME}.${UID}.$$"
2182.     }
2183. fi
2184. #生成临时文件
2185. #${2}为该文件的后缀
2186. tmpfile(){
2187.     tmp=$(mktemp -u "${TMPDIR}/ffconf.XXXXXXXX")$2 &&
2188.     (set -C; exec > $tmp) 2>/dev/null ||
2189.         die "Unable to create temporary file in $TMPDIR."
2190.     append TMPFILES $tmp
2191.     eval $1=$tmp
2192. }
2193.
2194. trap 'rm -f -- $TMPFILES' EXIT
2195. #各种临时文件
2196. tmpfile TMPASM .asm
2197. tmpfile TMPC .c
2198. tmpfile TMPCPP .cpp
2199. tmpfile TMPE $EXESUF
2200. tmpfile TMPH .h
2201. tmpfile TMPO .o
2202. tmpfile TMPH .S
2203. tmpfile TMPH .sh
2204. tmpfile TMPV .ver
2205.
2206. unset -f mktemp
2207.
2208. chmod +x $TMPE
2209.
2210. # make sure we can execute files in $TMPDIR
2211. cat > $TMPH 2>> $logfile <<EOF
2212. #! /bin/sh

```

```

2213. EOF
2214. chmod +x $TMPSSH >> $logfile 2>&1
2215. if ! $TMPSSH >> $logfile 2>&1; then
2216.     cat <<EOF
2217. Unable to create and execute files in $TMPDIR. Set the TMPDIR environment
2218. variable to another directory and make sure that it is not mounted noexec.
2219. EOF
2220.     die "Sanity test failed."
2221. fi
2222.
2223. filter_asflags=echo
2224. filter_cflags=echo
2225. filter_cppflags=echo
2226. #检查编译器
2227. if $cc -v 2>&1 | grep -q '^gcc.*LLVM'; then
2228.     cc_type=llvm_gcc
2229.     cc_version=_VERSION_
2230.     gcc_extra_ver=$(expr "$($cc --version | head -n1)" : '\.*\([^\)]*\)\(')
2231.     cc_id="llvm-gcc $($cc --dumpversion) $gcc_extra_ver"
2232.     CC_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@"'
2233.     AS_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@"'
2234.     speed_cflags='-O3'
2235.     size_cflags='-Os'
2236. elif $cc -v 2>&1 | grep -qi ^gcc; then
2237.     cc_type=gcc
2238.     cc_version=_VERSION_
2239.     gcc_version=$($cc --version | head -n1)
2240.     gcc_basever=$(gcc --dumpversion)
2241.     gcc_pkg_ver=$(expr "$gcc_version" : '[^]*\([^\)]*\)\(')
2242.     gcc_ext_ver=$(expr "$gcc_version" : "\.*$gcc_pkg_ver $gcc_basever \\(.*)")
2243.     cc_id=$(cleanws "gcc $gcc_basever $gcc_pkg_ver $gcc_ext_ver")
2244.     if ! $cc --dumpversion | grep -q '^2\.'; then
2245.         CC_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@"'
2246.         AS_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@"'
2247.     fi
2248.     speed_cflags='-O3'
2249.     size_cflags='-Os'
2250. elif $cc --version 2>/dev/null | grep -q Intel; then
2251.     cc_type=icc
2252.     cc_version="AV_STRINGIFY(_INTEL_COMPILER)"
2253.     cc_id=$(gcc --version | head -n1)
2254.     icc_version=$($cc --dumpversion)
2255.     CC_DEPFLAGS='-MMD'
2256.     AS_DEPFLAGS='-MMD'
2257.     speed_cflags='-O3'
2258.     size_cflags='-Os'
2259.     noopt_cflags='-O1'
2260. elif $cc -v 2>&1 | grep -q xlc; then
2261.     cc_type=xlc
2262.     cc_version="AV_STRINGIFY(_IBMC_)"
2263.     cc_id=$(gcc -qversion 2>/dev/null | head -n1)
2264.     speed_cflags='-O5'
2265.     size_cflags='-O5 -qcompact'
2266. elif $cc -V 2>/dev/null | grep -q Compaq; then
2267.     cc_type=cxx
2268.     cc_version="AV_STRINGIFY(_DECC_VER)"
2269.     cc_id=$(gcc -V | head -n1 | cut -d' ' -f1-3)
2270.     DEPFLAGS='$(CPPFLAGS) $(CFLAGS) -M'
2271.     debuglevel=3
2272.     add_ldflags '-Wl,-z,now # calls to libots crash without this'
2273.     speed_cflags='-fast'
2274.     size_cflags='-O1'
2275. elif $cc --vsn 2>/dev/null | grep -q "ARM C/C++ Compiler"; then
2276.     test -d "$sysroot" || die "No valid sysroot specified."
2277.     cc_type=armcc
2278.     cc_version="AV_STRINGIFY(_ARMCC_VERSION)"
2279.     cc_id=$(gcc --vsn | head -n1)
2280.     armcc_conf="$PWD/armcc.conf"
2281.     $cc --arm_linux_configure \
2282.         --arm_linux_config_file="$armcc_conf" \
2283.         --configure_sysroot="$sysroot" \
2284.         --configure_cpp_headers="$sysinclude" >>$logfile 2>&1 ||
2285.         die "Error creating armcc configuration file."
2286.     $cc --vsn | grep -q RVCT && armcc_opt=rvct || armcc_opt=armcc
2287.     cc="$cc --arm_linux_config_file=$armcc_conf --translate_gcc"
2288.     as_default='${cross_prefix}gcc'
2289.     CC_DEPFLAGS='-MMD'
2290.     AS_DEPFLAGS='-MMD'
2291.     speed_cflags='-O3'
2292.     size_cflags='-Os'
2293.     filter_asflags="filter_out -W${armcc_opt}"
2294. elif $cc --version 2>/dev/null | grep -q TMS470; then
2295.     cc_type=tms470
2296.     cc_version="AV_STRINGIFY(_TI_COMPILER_VERSION_)"
2297.     cc_id=$(gcc --version | head -n1 | tr -s ' ')
2298.     cc="$cc --gcc --abi=eabi -eo=-o -mc-me"
2299.     CC_O='-fr=${@}'
2300.     as_default='${cross_prefix}gcc'
2301.     ld_default='${cross_prefix}gcc'
2302.     TMPPO=$(basename $TMPC .c).o
2303.     append TMPFILES $TMPPO
2304.     add_flags "-B ${sysroot}/lib -L ${sysroot}/usr/lib -L ${sysroot}/usr/local/lib -L ${sysroot}/lib64 -L ${sysroot}/usr/local/lib64"

```

```

2304. add_cflags -D __GNUC_VA_LIST=VA_LIST -D __USER_LABEL_PREFIX__ =
2305. CC_DEPFLAGS='-ppa -ppd=$(@:.o=.d)'
2306. AS_DEPFLAGS='-MMD'
2307. speed_cflags='-O3 -mf=5'
2308. size_cflags='-O3 -mf=2'
2309. filter_cflags=tms470_flags
2310. tms470_flags(){
2311.     for flag; do
2312.         case $flag in
2313.             -march=*)
2314.                 case "${flag#*=}" in
2315.                     armv7-a|cortex-a*) echo -mv=7a8 ;;
2316.                     armv7-r|cortex-r*) echo -mv=7r4 ;;
2317.                     armv7-m|cortex-m*) echo -mv=7m3 ;;
2318.                     armv6*|arm11*) echo -mv=6 ;;
2319.                     armv5*|arm[79]*|arm9[24]6*|arm96*|arm102[26])
2320.                         echo -mv=5e ;;
2321.                     armv4*|arm7*|arm9[24]*) echo -mv=4 ;;
2322.                 esac
2323.             ;;
2324.             -mfpu=neon) echo --float_support=vfpv3 --neon ;;
2325.             -mfpu=vfp) echo --float_support=vfpv2 ;;
2326.             -mfpu=vfpv3) echo --float_support=vfpv3 ;;
2327.             -msoft-float) echo --float_support=vfpplib ;;
2328.             -O[0-3]|-mf=*) echo $flag ;;
2329.             -g) echo -g -mn ;;
2330.             -pds=*) echo $flag ;;
2331.         esac
2332.     done
2333. }
2334. elif $cc -v 2>&1 | grep -q clang; then
2335.     cc_type=clang
2336.     $cc -dM -E $TMPC | grep -q __clang_version__ &&
2337.     cc_version=__clang_version__ || cc_version=__VERSION__
2338.     cc_ident=$(($cc --version | head -n1)
2339.     CC_DEPFLAGS='-MMD'
2340.     AS_DEPFLAGS='-MMD'
2341.     speed_cflags='-O3'
2342.     size_cflags='-Os'
2343. elif $cc -V 2>&1 | grep -q Sun; then
2344.     cc_type=suncc
2345.     cc_version="AV_STRINGIFY(__SUNPRO_C)"
2346.     cc_ident=$(($cc -V 2>&1 | head -n1 | cut -d' ' -f 2-)
2347.     DEPEND_CMD='$(DEPCC) $(DEPFLAGS) $< | sed -e "1s,^,*, :$@,;" -e "\$\$!\$, \\\$, " -e "1!s,^,*, :$@,;" -e "1!s,^,*, :$@,;"'
2348.     DEPFLAGS='$(CPPFLAGS) $(CFLAGS) -xM1'
2349.     add_ldflags -xc99
2350.     speed_cflags='-O5'
2351.     size_cflags='-O5 -xspace'
2352.     filter_cflags=suncc_flags
2353.     suncc_flags(){
2354.         for flag; do
2355.             case $flag in
2356.                 -march=*)
2357.                     case "${flag#*=}" in
2358.                         native) echo -xtarget=native ;;
2359.                         v9|niagara) echo -xarch=sparc ;;
2360.                         ultrasparc) echo -xarch=sparcvis ;;
2361.                         ultrasparc3|niagara2) echo -xarch=sparcvis2 ;;
2362.                         i586|pentium) echo -xchip=pentium ;;
2363.                         i686|pentiumpro|pentium2) echo -xtarget=pentium_pro ;;
2364.                         pentium3*|c3-2) echo -xtarget=pentium3 ;;
2365.                         pentium-m) echo -xarch=sse2 -xchip=pentium3 ;;
2366.                         pentium4*) echo -xtarget=pentium4 ;;
2367.                         prescott|nocona) echo -xarch=sse3 -xchip=pentium4 ;;
2368.                         *-sse3) echo -xarch=sse3 ;;
2369.                         core2) echo -xarch=ssse3 -xchip=core2 ;;
2370.                         amdfam10|barcelona) echo -xarch=sse4_1 ;;
2371.                         athlon-4|athlon-[mx]p) echo -xarch=ssea ;;
2372.                         k8|opteron|athlon64|athlon-fx)
2373.                             echo -xarch=sse2a ;;
2374.                         athlon*) echo -xarch=pentium_proa ;;
2375.                     esac
2376.                 ;;
2377.                 -std=c99) echo -xc99 ;;
2378.                 -fomit-frame-pointer) echo -xregs=frameptr ;;
2379.                 -fPIC) echo -KPIC -xcode=pic32 ;;
2380.                 -W*,*) echo $flag ;;
2381.                 -f*-*|-W*) echo $flag ;;
2382.                 *) echo $flag ;;
2383.             esac
2384.         done
2385.     }
2386. elif $cc -v 2>&1 | grep -q 'PathScale\|Path64'; then
2387.     cc_type=pathscales
2388.     cc_version=__PATHSCALE__
2389.     cc_ident=$(($cc -v 2>&1 | head -n1 | tr -d :)
2390.     CC_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2391.     AS_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2392.     speed_cflags='-O2'
2393.     size_cflags='-Os'
2394.     filter_cflags='filter_out -Wdisabled-optimization'
2395. elif $cc -v 2>&1 | grep -q Open64; then

```

```

2396.     cc_type=open64
2397.     cc_version=_OPEN64_
2398.     cc_id=$(($cc -v 2>&1 | head -n1 | tr -d :))
2399.     CC_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2400.     AS_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2401.     speed_cflags='-O2'
2402.     size_cflags='-Os'
2403.     filter_cflags='filter_out -Wdisabled-optimization|-Wtype-limits|-fno-signed-zeros'
2404. fi
2405.
2406. test -n "$cc_type" && enable $cc_type ||
2407.     warn "Unknown C compiler $cc, unable to select optimal CFLAGS"
2408.
2409. : ${as_default:=$cc}
2410. : ${dep_cc_default:=$cc}
2411. : ${ld_default:=$cc}
2412. set_default ar as dep_cc ld
2413.
2414. test -n "$CC_DEPFLAGS" || CCDEP=$DEPEND_CMD
2415. test -n "$CXX_DEPFLAGS" || CXXDEP=$DEPEND_CMD
2416. test -n "$AS_DEPFLAGS" || ASDEP=$DEPEND_CMD
2417.
2418. add_cflags $extra_cflags
2419. add_cxxflags $extra_cxxflags
2420. add_asflags $extra_cflags
2421.
2422. if test -n "$sysroot"; then
2423.     case "$cc_type" in
2424.         gcc|llvm_gcc|clang)
2425.             add_cppflags --sysroot="$sysroot"
2426.             add_ldflags --sysroot="$sysroot"
2427.             ;;
2428.         tms470)
2429.             add_cppflags -I"$sysinclude"
2430.             add_ldflags --sysroot="$sysroot"
2431.             ;;
2432.         esac
2433. fi
2434.
2435. if test "$cpu" = host; then
2436.     enabled cross_compile && die "--cpu=host makes no sense when cross-compiling."
2437.
2438.     case "$cc_type" in
2439.         gcc|llvm_gcc)
2440.             check_native(){
2441.                 $cc $1=native -v -c -o $TMP0 $TMPC >$TMPE 2>&1 || return
2442.                 sed -n "/cc1.*$1=/{
2443.                     s/.*$1=\\([^\ ]*\\).*/\\1/
2444.                     p
2445.                     q
2446.                 }" $TMPE
2447.             }
2448.             cpu=$(check_native -march || check_native -mcpu)
2449.             ;;
2450.         esac
2451.
2452.     test "${cpu:-host}" = host && die "--cpu=host not supported with compiler $cc"
2453. fi
2454.
2455. # Deal with common $arch aliases
2456. case "$arch" in
2457.     arm*|iPod*)
2458.         arch="arm"
2459.         ;;
2460.     mips|mipsel|IP*)
2461.         arch="mips"
2462.         ;;
2463.     mips64*)
2464.         arch="mips"
2465.         subarch="mips64"
2466.         ;;
2467.     parisc|hppa)
2468.         arch="parisc"
2469.         ;;
2470.     parisc64|hppa64)
2471.         arch="parisc"
2472.         subarch="parisc64"
2473.         ;;
2474.     "Power Macintosh"|ppc|powerpc|ppc64|powerpc64)
2475.         arch="ppc"
2476.         ;;
2477.     s390|s390x)
2478.         arch="s390"
2479.         ;;
2480.     sh4|sh)
2481.         arch="sh4"
2482.         ;;
2483.     sun4u|sparc64)
2484.         arch="sparc"
2485.         subarch="sparc64"
2486.         ;;

```



```

2487.     i[3-6]86|i86pc|BePC|x86pc|x86_64|x86_32|amd64)
2488.     arch="x86"
2489.     ;;
2490. esac
2491.
2492. is_in $arch $ARCH_LIST || warn "unknown architecture $arch"
2493. enable $arch
2494.
2495. # Add processor-specific flags
2496. #根据CPU类型的不同, 进行cpuflag的设置
2497. if test "$cpu" = generic; then
2498.     : do nothing
2499. elif enabled ppc; then
2500.
2501.     case $(tolower $cpu) in
2502.         601|ppc601|powerpc601)
2503.             cpuflags="-mcpu=601"
2504.             disable altivec
2505.             ;;
2506.         603*|ppc603*|powerpc603*)
2507.             cpuflags="-mcpu=603"
2508.             disable altivec
2509.             ;;
2510.         604*|ppc604*|powerpc604*)
2511.             cpuflags="-mcpu=604"
2512.             disable altivec
2513.             ;;
2514.         g3|75*|ppc75*|powerpc75*)
2515.             cpuflags="-mcpu=750 -mpowerpc-gfxopt"
2516.             disable altivec
2517.             ;;
2518.         g4|745*|ppc745*|powerpc745*)
2519.             cpuflags="-mcpu=7450 -mpowerpc-gfxopt"
2520.             ;;
2521.         74*|ppc74*|powerpc74*)
2522.             cpuflags="-mcpu=7400 -mpowerpc-gfxopt"
2523.             ;;
2524.         g5|970|ppc970|powerpc970|power4*)
2525.             cpuflags="-mcpu=970 -mpowerpc-gfxopt -mpowerpc64"
2526.             ;;
2527.         cell)
2528.             cpuflags="-mcpu=cell"
2529.             enable ldbrx
2530.             ;;
2531.         e500v2)
2532.             cpuflags="-mcpu=8548 -mhard-float -mfloat-gprs=double"
2533.             disable altivec
2534.             ;;
2535.         e500)
2536.             cpuflags="-mcpu=8540 -mhard-float"
2537.             disable altivec
2538.             ;;
2539.     esac
2540. #X86架构
2541. elif enabled x86; then
2542.
2543.     case $cpu in
2544.         i[345]86|pentium)
2545.             cpuflags="-march=$cpu"
2546.             disable mmx
2547.             ;;
2548.         # targets that do NOT support conditional mov (cmov)
2549.         pentium-mmx|k6|k6-[23]|winchip-c6|winchip2|c3)
2550.             cpuflags="-march=$cpu"
2551.             disable cmov
2552.             ;;
2553.         # targets that do support conditional mov (cmov)
2554.         i686|pentiumpro|pentium[23]|pentium-m|athlon|athlon-tbird|athlon-4|athlon-[mx]p|athlon64*|k8*|opteron*|athlon-fx|core2|amdfam10|barcelona|atom)
2555.             cpuflags="-march=$cpu"
2556.             enable cmov
2557.             enable fast_cmov
2558.             ;;
2559.         # targets that do support conditional mov but on which it's slow
2560.         pentium4|pentium4m|prescott|nocona)
2561.             cpuflags="-march=$cpu"
2562.             enable cmov
2563.             disable fast_cmov
2564.             ;;
2565.     esac
2566.
2567. elif enabled sparc; then
2568.
2569.     case $cpu in
2570.         niagara)
2571.             cpuflags="-mcpu=$cpu"
2572.             disable vis
2573.             ;;
2574.         sparc64)
2575.             cpuflags="-mcpu=v9"
2576.             ;;

```

```

2577.     esac
2578. #ARM架构
2579. elif enabled arm; then
2580.
2581.     case $cpu in
2582.         armv*)
2583.             cpuflags="-march=$cpu"
2584.             subarch=$(echo $cpu | sed 's/[^a-z0-9]//g')
2585.             ;;
2586.         *)
2587.             cpuflags="-mcpu=$cpu"
2588.             case $cpu in
2589.                 cortex-a*)             subarch=armv7a ;;
2590.                 cortex-r*)             subarch=armv7r ;;
2591.                 cortex-m*)             enable_thumb; subarch=armv7m ;;
2592.                 arm11*)                 subarch=armv6 ;;
2593.                 arm[79]*e*|arm9[24]6*|arm96*|arm102[26]) subarch=armv5te ;;
2594.                 armv4*|arm7*|arm9[24]*) subarch=armv4 ;;
2595.             esac
2596.             ;;
2597.     esac
2598.
2599. elif enabled alpha; then
2600.
2601.     enabled ccc && cpuflags="-arch $cpu" || cpuflags="-mcpu=$cpu"
2602.
2603. elif enabled bfin; then
2604.
2605.     cpuflags="-mcpu=$cpu"
2606.
2607. elif enabled mips; then
2608.
2609.     cpuflags="-march=$cpu"
2610.
2611. elif enabled avr32; then
2612.
2613.     case $cpu in
2614.         ap7[02]0[0-2])
2615.             subarch="avr32_ap"
2616.             cpuflags="-mpart=$cpu"
2617.             ;;
2618.         ap)
2619.             subarch="avr32_ap"
2620.             cpuflags="-march=$cpu"
2621.             ;;
2622.         uc3[ab]*)
2623.             subarch="avr32_uc"
2624.             cpuflags="-mcpu=$cpu"
2625.             ;;
2626.         uc)
2627.             subarch="avr32_uc"
2628.             cpuflags="-march=$cpu"
2629.             ;;
2630.     esac
2631.
2632. fi
2633.
2634. add_cflags $cpuflags
2635. add_asflags $cpuflags
2636.
2637. # compiler sanity check
2638. # 用个简单的main()检查能不能用
2639. check_exec <<EOF
2640. int main(void){ return 0; }
2641. EOF
2642. if test "$?" != 0; then
2643.     echo "$cc is unable to create an executable file."
2644.     if test -z "$cross_prefix" && ! enabled cross_compile ; then
2645.         echo "If $cc is a cross-compiler, use the --enable-cross-compile option."
2646.         echo "Only do this if you know what cross compiling means."
2647.     fi
2648.     die "C compiler test failed."
2649. fi
2650.
2651. add_cppflags -D_ISO99_SOURCE
2652. add_cxxflags -D_STDC_CONSTANT_MACROS
2653. check_cflags -std=c99
2654. check_cc -D_FILE_OFFSET_BITS=64 <<EOF && add_cppflags -D_FILE_OFFSET_BITS=64
2655. #include <stdlib.h>
2656. EOF
2657. check_cc -D_LARGEFILE_SOURCE <<EOF && add_cppflags -D_LARGEFILE_SOURCE
2658. #include <stdlib.h>
2659. EOF
2660.
2661. check_host_cflags -std=c99
2662. check_host_cflags -Wall
2663. #32位系统指针变量占用32bit (4Byte) 数据 (32位寻址)
2664. #64位系统指针变量占用64bit (4Byte) 数据 (64位寻址)
2665. case "$sarch" in
2666.     alpha|ia64|mips|parisc|sparc)
2667.         spic=$shared

```

```

2668. ;;
2669. x86)
2670.     subarch="x86_32"
2671.     check_cc <<EOF && subarch="x86_64"
2672.     int test[(int)sizeof(char*) - 7];
2673. EOF
2674.     if test "$subarch" = "x86_64"; then
2675.         spic=$shared
2676.     fi
2677. ;;
2678. ppc)
2679.     check_cc <<EOF && subarch="ppc64"
2680.     int test[(int)sizeof(char*) - 7];
2681. EOF
2682. ;;
2683. esac
2684.
2685. enable $subarch
2686. enabled spic && enable pic
2687.
2688. # 不同的操作系统 OS specific
2689. # target-os参数
2690. case $target_os in
2691.     haiku)
2692.         prefix_default="/boot/common"
2693.         network_extralibs="-lnetwork"
2694.         host_libs=
2695.         ;;
2696.     sunos)
2697.         FFSERVERLDFLAGS=""
2698.         SHFLAGS='-shared -Wl,-h,$$(@F)'
2699.         enabled x86 && SHFLAGS="-mimpure-text $SHFLAGS"
2700.         network_extralibs="-lsocket -lnsl"
2701.         add_cppflags -D_EXTENSIONS_
2702.         # When using suncc to build, the Solaris linker will mark
2703.         # an executable with each instruction set encountered by
2704.         # the Solaris assembler. As our libraries contain their own
2705.         # guards for processor-specific code, instead suppress
2706.         # generation of the HWCAPS ELF section on Solaris x86 only.
2707.         enabled_all suncc x86 && echo "hwcap_1 = OVERRIDE;" > mapfile && add_ldflags -Wl,-M,mapfile
2708.         nm_opts='-P -g'
2709.         ;;
2710.     netbsd)
2711.         disable symver
2712.         oss_indev_extralibs="-lossaudio"
2713.         oss_outdev_extralibs="-lossaudio"
2714.         ;;
2715.     openbsd)
2716.         enable malloc_aligned
2717.         # On OpenBSD 4.5. the compiler does not use PIC unless
2718.         # explicitly using -fPIC. FFmpeg builds fine without PIC,
2719.         # however the generated executable will not do anything
2720.         # (simply quits with exit-code 1, no crash, no output).
2721.         # Thus explicitly enable PIC here.
2722.         enable pic
2723.         disable symver
2724.         SHFLAGS='-shared'
2725.         oss_indev_extralibs="-lossaudio"
2726.         oss_outdev_extralibs="-lossaudio"
2727.         ;;
2728.     dragonfly)
2729.         enable malloc_aligned
2730.         disable symver
2731.         ;;
2732.     freebsd)
2733.         enable malloc_aligned
2734.         ;;
2735.     bsd/os)
2736.         add_extralibs -lpoll -lgnugetopt
2737.         strip="strip -d"
2738.         ;;
2739.     #苹果Mac操作系统
2740.     darwin)
2741.         enable malloc_aligned
2742.         #以前见过这个
2743.         gas="gas-preprocessor.pl $cc"
2744.         enabled ppc && add_asflags -force_cpusubtype_ALL
2745.         SHFLAGS="-dynamiclib -Wl,-single_module -Wl,-install_name,$(SHLIBDIR)/$(SLIBNAME),-current_version,$(LIBVERSION),-compatibil
ity_version,$(LIBMAJOR)"
2746.         enabled x86_32 && append SHFLAGS -Wl,-read_only_relocs,suppress
2747.         strip="{strip} -x"
2748.         add_ldflags -Wl,-dynamic,-search_paths_first
2749.         #Mac下的动态库
2750.         SLIBSUF=".dylib"
2751.         SLIBNAME_WITH_VERSION='$(SLIBPREFIX)$(FULLNAME).$(LIBVERSION)$(SLIBSUF)'
2752.         SLIBNAME_WITH_MAJOR='$(SLIBPREFIX)$(FULLNAME).$(LIBMAJOR)$(SLIBSUF)'
2753.         FFSERVERLDFLAGS=-Wl,-bind_at_load
2754.         #macho目标文件格式
2755.         objformat="macho"
2756.         enabled x86_64 && objformat="macho64"
2757.         enabled_any pic shared ||

```

```

2758.         { check_cflags -mdynamic-no-pic && add_asflags -mdynamic-no-pic; }
2759.         ;;
2760. #MinGW
2761.     mingw32*)
2762.         if test $target_os = "mingw32ce"; then
2763.             disable_network
2764.         else
2765.             target_os=mingw32
2766.         fi
2767.         LIBTARGET=i386
2768.         if enabled x86_64; then
2769.             enable_malloc_aligned
2770.             LIBTARGET=x64
2771.         elif enabled arm; then
2772.             LIBTARGET=arm-wince
2773.         fi
2774.         shlibdir_default="$bindir_default"
2775.         SLIBPREFIX=""
2776.         # Windows下动态库后缀
2777.         SLIBSUF=".dll"
2778.         SLIBNAME_WITH_VERSION='$(SLIBPREFIX)$(FULLNAME)-$(LIBVERSION)$(SLIBSUF)'
2779.         SLIBNAME_WITH_MAJOR='$(SLIBPREFIX)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)'
2780.         # 借助lib.exe生成导出库lib
2781.         SLIB_EXTRA_CMD='-lib.exe /machine:$$(LIBTARGET) /def:$$(@:$(SLIBSUF)=.def) /out:$$(SUBDIR)$(SLIBNAME:$(SLIBSUF)=.lib)'
2782.         SLIB_INSTALL_NAME='$(SLIBNAME_WITH_MAJOR)'
2783.         SLIB_INSTALL_LINKS=
2784.         # 额外的lib导入库
2785.         SLIB_INSTALL_EXTRA_SHLIB='$(SLIBNAME:$(SLIBSUF)=.lib)'
2786.         # 额外的
2787.         SLIB_INSTALL_EXTRA_LIB='lib$(SLIBNAME:$(SLIBSUF)=.dll.a) $(SLIBNAME_WITH_MAJOR:$(SLIBSUF)=.def)'
2788.         SHFLAGS='-shared -Wl,--output-def,$$(@:$(SLIBSUF)=.def) -Wl,--out-implib,$(SUBDIR)lib$(SLIBNAME:$(SLIBSUF)=.dll.a) -Wl,--enable-runtime-pseudo-reloc -Wl,--enable-auto-image-base'
2789.         # windows PE格式
2790.         objformat="win32"
2791.         enable_dos_paths
2792.         check_cflags -fno-common
2793.         check_cpp_condition _mingw.h "defined (__MINGW64_VERSION_MAJOR) || (__MINGW32_MAJOR_VERSION > 3) \
2794.             || (__MINGW32_MAJOR_VERSION == 3 && __MINGW32_MINOR_VERSION >= 15)" ||
2795.             die "ERROR: MinGW runtime version must be >= 3.15."
2796.         add_cppflags -U__STRICT_ANSI__
2797.         ;;
2798. #Cygwin
2799.     cygwin*)
2800.         target_os=cygwin
2801.         shlibdir_default="$bindir_default"
2802.         SLIBPREFIX="cyg"
2803.         SLIBSUF=".dll"
2804.         SLIBNAME_WITH_VERSION='$(SLIBPREFIX)$(FULLNAME)-$(LIBVERSION)$(SLIBSUF)'
2805.         SLIBNAME_WITH_MAJOR='$(SLIBPREFIX)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)'
2806.         SHFLAGS='-shared -Wl,--enable-auto-image-base'
2807.         objformat="win32"
2808.         enable_dos_paths
2809.         check_cflags -fno-common
2810.         add_cppflags -U__STRICT_ANSI__
2811.         ;;
2812. *-dos|freedos|opendos)
2813.         network_extralibs="-lsocket"
2814.         objformat="coff"
2815.         enable_dos_paths
2816.         add_cppflags -U__STRICT_ANSI__
2817.         ;;
2818. #Linux操作系统
2819.     linux)
2820.         add_cppflags -D_POSIX_C_SOURCE=200112 -D_XOPEN_SOURCE=600
2821.         enable_dv1394
2822.         ;;
2823.     irix*)
2824.         target_os=irix
2825.         ranlib="echo ignoring ranlib"
2826.         ;;
2827.     os/2*)
2828.         strip="xlite -CS"
2829.         ln_s="cp -f"
2830.         objformat="aout"
2831.         add_cppflags -D_GNU_SOURCE
2832.         add_ldflags -Zomf -Zbin-files -Zargs-wild -Zmap
2833.         SHFLAGS='$(SUBDIR)$(NAME).def -Zdll -Zomf'
2834.         FFSERVERLDFLAGS=""
2835.         LIBSUF="_s.a"
2836.         SLIBPREFIX=""
2837.         SLIBSUF=".dll"
2838.         SLIBNAME_WITH_VERSION='$(SLIBPREFIX)$(NAME)-$(LIBVERSION)$(SLIBSUF)'
2839.         SLIBNAME_WITH_MAJOR='$(SLIBPREFIX)$(shell echo $(NAME) | cut -c1-6)$(LIBMAJOR)$(SLIBSUF)'
2840.         SLIB_CREATE_DEF_CMD='echo LIBRARY $(SLIBNAME_WITH_MAJOR) INITINSTANCE TERMINSTANCE > $(SUBDIR)$(NAME).def; \
2841.             echo PROTMODE >> $(SUBDIR)$(NAME).def; \
2842.             echo CODE PRELOAD MOVEABLE DISCARDABLE >> $(SUBDIR)$(NAME).def; \
2843.             echo DATA PRELOAD MOVEABLE MULTIPLE NONSHARED >> $(SUBDIR)$(NAME).def; \
2844.             echo EXPORTS >> $(SUBDIR)$(NAME).def; \
2845.             emxexp -o $(OBJS) >> $(SUBDIR)$(NAME).def'
2846.         SLIB_EXTRA_CMD='emximp -o $(SUBDIR)$(LIBPREFIX)$(NAME)_dll.a $(SUBDIR)$(NAME).def; \
2847.             emximp -o $(SUBDIR)$(LIBPREFIX)$(NAME)_dll.lib $(SUBDIR)$(NAME).def;'
2848.         SLIB_INSTALL_EXTRA_LIB='$(SUBDIR)$(LIBPREFIX)$(NAME)_dll.a $(SUBDIR)$(NAME).def;'
2849.         SLIB_INSTALL_EXTRA_SHLIB='$(SUBDIR)$(LIBPREFIX)$(NAME)_dll.lib $(SUBDIR)$(NAME).def;'

```

```

2848.     SLIB_INSTALL_EXTRA_LIB='${LIBPREFIX}${NAME}_dll.a ${LIBPREFIX}${NAME}_dll.lib'
2849.     enable dos_paths
2850.     enable_weak os2threads
2851.     ;;
2852.     gnu/kfreebsd)
2853.         add_cppflags -D_POSIX_C_SOURCE=200112 -D_XOPEN_SOURCE=600 -D_BSD_SOURCE
2854.         ;;
2855.     gnu)
2856.         add_cppflags -D_POSIX_C_SOURCE=200112 -D_XOPEN_SOURCE=600
2857.         ;;
2858.     qnx)
2859.         add_cppflags -D_QNX_SOURCE
2860.         network_extralibs="-lsocket"
2861.         ;;
2862.     symbian)
2863.         SLIBSUF=".dll"
2864.         enable dos_paths
2865.         add_cflags --include=$sysinclude/gcce/gcce.h -fvisibility=default
2866.         add_cppflags -D_GCCE_ -D_SYMBIAN32 -DSYMBIAN_OE_POSIX_SIGNALS
2867.         add_ldflags -Wl,--target1-abs,--no-undefined \
2868.             -Wl,-Ttext,0x80000,-Tdata,0x1000000 -shared \
2869.             -Wl,--entry=_E32Startup -Wl,-u,_E32Startup
2870.         add_extralibs -l:eeexe.lib -l:usrt2_2.lib -l:dfpaeabi.dso \
2871.             -l:drtaeabi.dso -l:scppnwdl.dso -lsupc++ -lgcc \
2872.             -l:libc.dso -l:libm.dso -l:euser.dso -l:libcrt0.lib
2873.         ;;
2874.     none)
2875.         ;;
2876.     *)
2877.         die "Unknown OS '$target_os'."
2878.         ;;
2879. esac
2880.
2881. echo "config:$arch:$subarch:$cpu:$target_os:$cc_ident:$FFMPEG_CONFIGURATION" >config.fate
2882.
2883. check_cpp_condition stdlib.h "defined(__PIC__) || defined(__pic__) || defined(PIC)" && enable pic
2884.
2885. set_default $PATHS_LIST
2886.
2887. # we need to build at least one lib type
2888. if ! enabled_any static shared; then
2889.     cat <<EOF
2890. At least one library type must be built.
2891. Specify --enable-static to build the static libraries or --enable-shared to
2892. build the shared libraries as well. To only build the shared libraries specify
2893. --disable-static in addition to --enable-shared.
2894. EOF
2895.     exit 1;
2896. fi
2897. #不符合License则立刻结束
2898. die_license_disabled() {
2899.     enabled $1 || { enabled $2 && die "$2 is $1 and --enable-$1 is not specified."; }
2900. }
2901. #检查License
2902. #GPL
2903. die_license_disabled gpl libcdio
2904. die_license_disabled gpl libx264
2905. die_license_disabled gpl libxavs
2906. die_license_disabled gpl libxvid
2907. die_license_disabled gpl x11grab
2908. #nonfree
2909. die_license_disabled nonfree libaacplus
2910. die_license_disabled nonfree libfaac
2911. die_license_disabled nonfree openssl
2912. #Version3
2913. die_license_disabled version3 libopencore_amrnb
2914. die_license_disabled version3 libopencore_amrwb
2915. die_license_disabled version3 libvo_aacenc
2916. die_license_disabled version3 libvo_amrwbenc
2917.
2918. enabled version3 && { enabled gpl && enable gplv3 || enable lgplv3; }
2919.
2920. disabled optimizations || check_cflags -fomit-frame-pointer
2921. #添加fPIC
2922. enable_pic() {
2923.     enable pic
2924.     add_cppflags -DPIC
2925.     add_cflags -fPIC
2926.     add_asflags -fPIC
2927. }
2928.
2929. enabled pic && enable_pic
2930.
2931. check_cc <<EOF || die "Symbol mangling check failed."
2932. int ff_extern;
2933. EOF
2934. sym=${nm $nm_opts $TMP0 | awk '/ff_extern/{ print substr($0, match($0, /^[^ \t]*ff_extern/)) }'}
2935. extern_prefix=${sym%%ff_extern*}
2936.
2937. check_cc <<EOF && enable inline_asm
2938. void foo(void) { __asm__ volatile (""; ); }
2939. EOF

```

```

2940.
2941. _restrict=
2942. for restrict_keyword in restrict __restrict__ __restrict; do
2943.     check_cc <<EOF && _restrict=$restrict_keyword && break
2944. void foo(char * $restrict_keyword p);
2945. EOF
2946. done
2947.
2948. check_cc <<EOF && enable attribute_packed
2949. struct { int x; } __attribute__((packed)) x;
2950. EOF
2951.
2952. check_cc <<EOF && enable attribute_may_alias
2953. union { int x; } __attribute__((may_alias)) x;
2954. EOF
2955.
2956. check_cc <<EOF || die "endian test failed"
2957. unsigned int endian = 'B' << 24 | 'I' << 16 | 'G' << 8 | 'E';
2958. EOF
2959. od -t x1 $TMP0 | grep -q '42 *49 *47 *45' && enable bigendian
2960.
2961. if enabled alpha; then
2962.
2963.     check_cflags -mieee
2964.
2965. elif enabled arm; then
2966.
2967.     enabled thumb && check_cflags -mthumb || check_cflags -marm
2968.     nogas=die
2969.
2970.     if check_cpp_condition stddef.h "defined __ARM_PCS_VFP"; then
2971.         enable vfp_args
2972.     elif ! check_cpp_condition stddef.h "defined __ARM_PCS || defined __SOFTFP__"; then
2973.         case "${cross_prefix:-$cc}" in
2974.             *hardfloat*) enable vfp_args; fpabi=vfp ;;
2975.             *) check_ld "cc" <<EOF && enable vfp_args && fpabi=vfp || fpabi=soft ;;
2976. __asm__ (".eabi_attribute 28, 1");
2977. int main(void) { return 0; }
2978. EOF
2979.     esac
2980.     warn "Compiler does not indicate floating-point ABI, guessing $fpabi."
2981. fi
2982.
2983. enabled armv5te && check_asm armv5te '"qadd r0, r0, r0"'
2984. enabled armv6 && check_asm armv6 '"sadd16 r0, r0, r0"'
2985. enabled armv6t2 && check_asm armv6t2 '"movt r0, #0"'
2986. enabled armvfp && check_asm armvfp '"fadds s0, s0, s0"'
2987. enabled iwmmxt && check_asm iwmmxt '"wunpckelub wr6, wr4"'
2988. enabled neon && check_asm neon '"vadd.i16 q0, q0, q0"'
2989. enabled vfpv3 && check_asm vfpv3 '"vmov.f32 s0, #1.0"'
2990.
2991. check_asm asm_mod_y '"vmul.i32 d0, d0, %y0" :: "x"(0)'
2992.
2993. enabled_all armv6t2 shared !pic && enable_pic
2994.
2995. elif enabled mips; then
2996.
2997.     check_asm loongson '"dmult.g $1, $2, $3"'
2998.     enabled mmi && check_asm mmi '"lq $2, 0($2)'"
2999.
3000. elif enabled ppc; then
3001.
3002.     enable local_aligned_8 local_aligned_16
3003.
3004.     check_asm dcbzl '"dcbzl 0, %0" :: "r"(0)'
3005.     check_asm ibm_asm '"add 0, 0, 0"'
3006.     check_asm ppc4xx '"macrlhw r10, r11, r12"'
3007.     check_asm xform_asm '"lwzx %1, %y0" :: "Z"(*(int*)0), "r"(0)'
3008.
3009.     # AltiVec flags: The FSF version of GCC differs from the Apple version
3010.     if enabled altivec; then
3011.         nogas=warn
3012.         check_cflags -maltivec -mabi=altivec &&
3013.         { check_header altivec.h && inc_altivec_h="#include <altivec.h>" ; } ||
3014.         check_cflags -faltivec
3015.
3016.         # check if our compiler supports Motorola AltiVec C API
3017.         check_cc <<EOF || disable altivec
3018. $inc_altivec_h
3019. int main(void) {
3020.     vector signed int v1, v2, v3;
3021.     v1 = vec_add(v2,v3);
3022.     return 0;
3023. }
3024. EOF
3025.
3026.     # check if our compiler supports braces for vector declarations
3027.     check_cc <<EOF || die "You need a compiler that supports {} in AltiVec vector declarations."
3028. $inc_altivec_h
3029. int main (void) { (vector int) {1}; return 0; }
3030. EOF

```

```

3031.     fi
3032.
3033. elif enabled sparc; then
3034.
3035.     enabled vis && check_asm vis "'pdist %f0, %f0, %f0'" -mcpu=ultrasparc &&
3036.         add_cflags -mcpu=ultrasparc -mtune=ultrasparc
3037.
3038. elif enabled x86; then
3039.
3040.     enable local_aligned_8 local_aligned_16
3041.
3042.     # check whether EBP is available on x86
3043.     # As 'i' is stored on the stack, this program will crash
3044.     # if the base pointer is used to access it because the
3045.     # base pointer is cleared in the inline assembly code.
3046.     check_exec_crash <<EOF && enable ebp_available
3047.     volatile int i=0;
3048.     __asm__ volatile (
3049.         "xorl %%ebp, %%ebp"
3050.         ::: "%ebp");
3051.     return i;
3052. EOF
3053.
3054.     # check whether EBX is available on x86
3055.     check_asm ebx_available '""::"b"(0)' &&
3056.         check_asm ebx_available '""::"%ebx"'
3057.
3058.     # check whether xmm clobbers are supported
3059.     check_asm xmm_clobbers '""::"%xmm0"'
3060.
3061.     # check whether binutils is new enough to compile SSE3/MMX2
3062.     enabled sse3 && check_asm sse3 "'pabsw %xmm0, %xmm0'"
3063.     enabled mmx2 && check_asm mmx2 "'pmaxub %mm0, %mm1'"
3064.
3065.     if ! disabled_any asm mmx yasm; then
3066.         if check_cmd $yasmexe --version; then
3067.             enabled x86_64 && yasm_extra="-m amd64"
3068.             yasm_debug="-g dwarf2"
3069.         elif check_cmd nasm -v; then
3070.             yasmexe=nasm
3071.             yasm_debug="-g -F dwarf"
3072.             enabled x86_64 && test "$objformat" = elf && objformat=elf64
3073.         fi
3074.
3075.         YASMFLAGS="-f $objformat $yasm_extra"
3076.         enabled pic && append YASMFLAGS "-DPIC"
3077.         test -n "$extern_prefix" && append YASMFLAGS "-DPREFIX"
3078.         case "$objformat" in
3079.             elf*) enabled debug && append YASMFLAGS $yasm_debug ;;
3080.             esac
3081.
3082.         check_yasm "pextrd [eax], xmm0, 1" && enable yasm ||
3083.             die "yasm not found, use --disable-yasm for a crippled build"
3084.         check_yasm "vextractf128 xmm0, ymm0, 0" || disable avx
3085.     fi
3086.
3087.     case "$cpu" in
3088.         athlon*|opteron*|k8*|pentium|pentium-mmx|prescott|nocona|atom|geode)
3089.             disable fast_clz
3090.             ;;
3091.         esac
3092.
3093. fi
3094.
3095. if enabled asm; then
3096.     as=${gas:=$as}
3097.     check_asm gnu_as "'macro m n\n\n: int 0\n.endm\nm x'" ||
3098.         $nogas "GNU assembler not found, install gas-preprocessor"
3099. fi
3100.
3101. check_ldflags -WL,--as-needed
3102.
3103. if check_func dlopen; then
3104.     ldld=
3105. elif check_func dlopen -ldl; then
3106.     ldld=-ldl
3107. fi
3108. #网络socket
3109. if enabled network; then
3110.     check_type "sys/types.h sys/socket.h" socklen_t
3111.     check_type netdb.h "struct addrinfo"
3112.     check_type netinet/in.h "struct ipv6_mreq" -D_DARWIN_C_SOURCE
3113.     check_type netinet/in.h "struct sockaddr_in6"
3114.     check_type "sys/types.h sys/socket.h" "struct sockaddr_storage"
3115.     check_struct "sys/types.h sys/socket.h" "struct sockaddr" sa_len
3116.     # Prefer arpa/inet.h over winsock2
3117.     if check_header arpa/inet.h ; then
3118.         check_func closesocket
3119.     elif check_header winsock2.h ; then
3120.         check_func_headers winsock2.h closesocket -lws2 && \
3121.             network_extralibs="-lws2" || \

```

```

3122.     { check_func_headers winsock2.h closesocket -lws2_32 && \
3123.         network_extralibs="-lws2_32"; }
3124.     check_type ws2tcpip.h socklen_t
3125.     check_type ws2tcpip.h "struct addrinfo"
3126.     check_type ws2tcpip.h "struct ipv6_mreq"
3127.     check_type ws2tcpip.h "struct sockaddr_in6"
3128.     check_type ws2tcpip.h "struct sockaddr_storage"
3129.     check_struct winsock2.h "struct sockaddr" sa_len
3130. else
3131.     disable network
3132. fi
3133.
3134.
3135. # Solaris has nanosleep in -lrt, OpenSolaris no longer needs that
3136. check_func nanosleep || { check_func nanosleep -lrt && add_extralibs -lrt; }
3137. #检查函数
3138. check_func fcntl
3139. check_func fork
3140. check_func getaddrinfo $network_extralibs
3141. check_func gethrtime
3142. check_func getrusage
3143. check_struct "sys/time.h sys/resource.h" "struct rusage" ru_maxrss
3144. check_func inet_aton $network_extralibs
3145. check_func isatty
3146. check_func localtime_r
3147. check_func ${malloc_prefix}memalign && enable memalign
3148. check_func mkstemp
3149. check_func mmap
3150. check_func ${malloc_prefix}posix_memalign && enable posix_memalign
3151. check_func setrlimit
3152. check_func strerror_r
3153. check_func strptime
3154. check_func sched_getaffinity
3155. check_func sysconf
3156. check_func sysctl
3157. check_func_headers conio.h kbhit
3158. check_func_headers windows.h PeekNamedPipe
3159. check_func_headers io.h setmode
3160. check_func_headers lzo/lzo1x.h lzo1x_999_compress
3161. check_lib2 "windows.h psapi.h" GetProcessMemoryInfo -lpsapi
3162. check_func_headers windows.h GetProcessAffinityMask
3163. check_func_headers windows.h GetProcessTimes
3164. check_func_headers windows.h MapViewOfFile
3165. check_func_headers windows.h VirtualAlloc
3166. #检查头文件
3167. check_header dlfcn.h
3168. check_header dxva2api.h -D_WIN32_WINNT=0x0600
3169. check_header libcrystalhd/libcrystalhd_if.h
3170. check_header malloc.h
3171. check_header poll.h
3172. check_header sys/mman.h
3173. check_header sys/param.h
3174. check_header sys/resource.h
3175. check_header sys/select.h
3176. check_header termios.h
3177. check_header vdpau/vdpau.h
3178. check_header vdpau/vdpau_x11.h
3179. check_header X11/extensions/XvMClib.h
3180. check_header asm/types.h
3181.
3182. disabled zlib || check_lib zlib.h zlibVersion -lz || disable zlib
3183. disabled bzip || check_lib2 bzip.h BZ2_bzipVersion -lbz2 || disable bzip
3184.
3185. # check for VDA header
3186. if ! disabled vda; then
3187.     if check_header VideoDecodeAcceleration/VDADecoder.h; then
3188.         enable vda
3189.         add_extralibs -framework CoreFoundation -framework VideoDecodeAcceleration -framework QuartzCore
3190.     fi
3191. fi
3192.
3193. if ! disabled w32threads && ! enabled pthreads; then
3194.     check_func _beginthreadex && enable w32threads
3195. fi
3196.
3197. # check for some common methods of building with pthread support
3198. # do this before the optional library checks as some of them require pthreads
3199. if ! disabled pthreads && ! enabled w32threads && ! enabled os2threads; then
3200.     enable pthreads
3201.     if check_func pthread_create; then
3202.         :
3203.     elif check_func pthread_create -pthread; then
3204.         add_cflags -pthread
3205.         add_extralibs -pthread
3206.     elif check_func pthread_create -pthreads; then
3207.         add_cflags -pthreads
3208.         add_extralibs -pthreads
3209.     elif check_func pthread_create -lpthreadGC2; then
3210.         add_extralibs -lpthreadGC2
3211.     elif ! check_lib pthread.h pthread_create -lpthread; then
3212.         disable pthreads

```



```

3213.     fi
3214. fi
3215.
3216. for thread in $THREADS_LIST; do
3217.     if enabled $thread; then
3218.         test -n "$thread_type" &&
3219.             die "ERROR: Only one thread type must be selected." ||
3220.                 thread_type="$thread"
3221.     fi
3222. done
3223.
3224. check_lib math.h sin -lm && LIBM="-lm"
3225. disabled crystalhd || check_lib libcrystalhd/libcrystalhd_if.h DtsCrystalHDVersion -lcrystalhd || disable crystalhd
3226. enabled vaapi && require vaapi va/va.h vaInitialize -lva
3227. #检查数学函数
3228. check_mathfunc cbrtf
3229. check_mathfunc exp2
3230. check_mathfunc exp2f
3231. check_mathfunc llrint
3232. check_mathfunc llrintf
3233. check_mathfunc log2
3234. check_mathfunc log2f
3235. check_mathfunc lrint
3236. check_mathfunc lrintf
3237. check_mathfunc round
3238. check_mathfunc roundf
3239. check_mathfunc trunc
3240. check_mathfunc truncf
3241.
3242. #检查第三方类库
3243. # these are off by default, so fail if requested and not available
3244. #require()函数参数的规范：(名称, 头文件, 函数名, 附加选项)
3245. #require2()函数参数规范类似
3246. enabled avisynth && require2 vfw32 "windows.h vfw.h" AVIFileInit -lavifil32
3247. enabled frei0r && { check_header frei0r.h || die "ERROR: frei0r.h header not found"; }
3248. enabled gnutls && require_pkg_config gnutls gnutls/gnutls.h gnutls_global_init
3249. enabled libaacplus && require "libaacplus >= 2.0.0" aacplus.h aacplusEncOpen -laacplus
3250. enabled libass && require_pkg_config libass ass/ass.h ass_library_init
3251. enabled libcelt && require libcelt celt/celt.h celt_decode -lcelt0 &&
3252.     { check_lib celt/celt.h celt_decoder_create_custom -lcelt0 ||
3253.         die "ERROR: libcelt version must be >= 0.11.0."; }
3254. enabled libdc1394 && require_pkg_config libdc1394-2 dc1394/dc1394.h dc1394_new
3255. enabled libdirac && require_pkg_config dirac \
3256.     "libdirac_decoder/dirac_parser.h libdirac_encoder/dirac_encoder.h" \
3257.     "dirac_decoder_init dirac_encoder_init"
3258. #测试libfaac
3259. enabled libfaac && require2 libfaac "stdint.h faac.h" faacEncGetVersion -lfaac
3260. enabled libfreetype && require_pkg_config freetype2 "ft2build.h freetype/freetype.h" FT_Init_FreeType
3261. enabled libgsm && require libgsm gsm/gsm.h gsm_create -lgsm
3262. enabled libmodplug && require libmodplug libmodplug/modplug.h ModPlug_Load -lmodplug
3263. enabled libmp3lame && require "libmp3lame >= 3.98.3" lame/lame.h lame_set_VBR_quality -lmp3lame
3264. enabled libnut && require libnut libnut.h nut_demuxer_init -lnut
3265. enabled libopencore_amrnb && require libopencore_amrnb opencore-amrnb/interf_dec.h Decoder_Interface_init -lopencore-amrnb
3266. enabled libopencore_amrwb && require libopencore_amrwb opencore-amrwb/dec_if.h D_IF_init -lopencore-amrwb
3267. enabled libopencv && require_pkg_config opencv opencv/cvcore.h cvCreateImageHeader
3268. enabled libopenjpeg && require libopenjpeg openjpeg.h opj_version -lopenjpeg
3269. enabled libpulse && require_pkg_config libpulse-simple pulse/simple.h pa_simple_new
3270. enabled librtmp && require_pkg_config librtmp librtmp/rtmp.h RTMP_Socket
3271. enabled libschrödinger && require_pkg_config schroedinger-1.0 schroedinger/schro.h schro_init
3272. enabled libspeex && require libspeex speex/speex.h speex_decoder_init -lspeex
3273. enabled libstagefright_h264 && require_cpp libstagefright_h264 "binder/ProcessState.h media/stagefright/MetaData.h
3274.     media/stagefright/MediaBufferGroup.h media/stagefright/MediaDebug.h media/stagefright/MediaDefs.h
3275.     media/stagefright/OMXClient.h media/stagefright/OMXCodec.h" android::OMXClient -lstagefright -lmedia -lutils -lbinder
3276. enabled libtheora && require libtheora theora/theoraenc.h th_info_init -ltheoraenc -ltheoradec -logg
3277. enabled libutvideo && require_cpp utvideo "stdint.h stdlib.h utvideo/utvideo.h utvideo/Codec.h" 'CCodec*' -lutvideo -lstdc++
3278. enabled libv4l2 && require_pkg_config libv4l2 libv4l2.h v4l2_ioctl
3279. enabled libvo_aacenc && require libvo_aacenc vo-aacenc/voAAC.h voGetAACEncAPI -lvo-aacenc
3280. enabled libvo_amrwbenc && require libvo_amrwbenc vo-amrwbenc/enc_if.h E_IF_init -lvo-amrwbenc
3281. enabled libvorbis && require libvorbis vorbis/vorbisenc.h vorbis_info_init -lvorbisenc -lvorbis -logg
3282. enabled libvpx && {
3283.     enabled libvpx_decoder && { check_lib2 "vpx/vpx_decoder.h vpx/vp8dx.h" vpx_codec_dec_init_ver -lvpx ||
3284.         die "ERROR: libvpx decoder version must be >=0.9.1"; }
3285.     enabled libvpx_encoder && { check_lib2 "vpx/vpx_encoder.h vpx/vp8cx.h" "vpx_codec_enc_init_ver VPX_CQ" -lvpx ||
3286.         die "ERROR: libvpx encoder version must be >=0.9.6"; } }
3287. #测试libx264
3288. enabled libx264 && require libx264 x264.h x264_encoder_encode -lx264 &&
3289.     { check_cpp_condition x264.h "X264_BUILD >= 118" ||
3290.         die "ERROR: libx264 version must be >= 0.118."; }
3291. enabled libxavs && require libxavs xavs.h xavs_encoder_encode -lxavs
3292. enabled libxvid && require libxvid xvid.h xvid_global -lxvidcore
3293. enabled openal && { { for al_libs in "${OPENAL_LIBS}" "-lopenal" "-lopenAL32"; do
3294.         check_lib 'AL/al.h' alGetError "${al_libs}" && break; done } ||
3295.         die "ERROR: openal not found"; } &&
3296.     { check_cpp_condition "AL/al.h" "defined(AL_VERSION_1_1)" ||
3297.         die "ERROR: openal version must be 1.1 or compatible"; }
3298. enabled mlib && require mediaLib mlib_types.h mlib_VectorSub_S16_U8_Mod -lmlib
3299. enabled openssl && { check_lib openssl/ssl.h SSL_library_init -lssl -lcrypto ||
3300.     check_lib openssl/ssl.h SSL_library_init -lssl32 -leay32 ||
3301.     check_lib openssl/ssl.h SSL_library_init -lssl -lcrypto -lws2_32 -lgdi32 ||
3302.     die "ERROR: openssl not found"; }
3303. #检查SDL

```

```

3304. SDL_CONFIG="${cross_prefix}sdl-config"
3305. if check_pkg_config sdl SDL_version.h SDL_Linked_Version; then
3306.     check_cpp_condition SDL.h "(SDL_MAJOR_VERSION<=16 | SDL_MINOR_VERSION<=8 | SDL_PATCHLEVEL) >= 0x010201" $sdl_cflags &&
3307.     enable sdl &&
3308.     check_struct SDL.h SDL_VideoInfo current_w $sdl_cflags && enable sdl_video_size
3309. else
3310.     if "${SDL_CONFIG}" --version > /dev/null 2>&1; then
3311.         sdl_cflags=$(("${SDL_CONFIG}" --cflags)
3312.         sdl_libs=$(("${SDL_CONFIG}" --libs)
3313.         check_func_headers SDL_version.h SDL_Linked_Version $sdl_cflags $sdl_libs &&
3314.         check_cpp_condition SDL.h "(SDL_MAJOR_VERSION<=16 | SDL_MINOR_VERSION<=8 | SDL_PATCHLEVEL) >= 0x010201" $sdl_cflags &&
3315.         enable sdl &&
3316.         check_struct SDL.h SDL_VideoInfo current_w $sdl_cflags && enable sdl_video_size
3317.     fi
3318. fi
3319. enabled sdl && add_cflags $sdl_cflags && add_extralibs $sdl_libs
3320.
3321. texi2html -version > /dev/null 2>&1 && enable texi2html || disable texi2html
3322. makeinfo --version > /dev/null 2>&1 && enable makeinfo || disable makeinfo
3323. #检查头文件
3324. check_header linux/fb.h
3325. check_header linux/videodev.h
3326. check_header linux/videodev2.h
3327. check_struct linux/videodev2.h "struct v4l2_frmlenenum" discrete
3328.
3329. check_header sys/videoio.h
3330.
3331. check_func_headers "windows.h vfw.h" capCreateCaptureWindow "$vfwcap_indev_extralibs"
3332. # check that WM_CAP_DRIVER_CONNECT is defined to the proper value
3333. # w32api 3.12 had it defined wrong
3334. check_cpp_condition vfw.h "WM_CAP_DRIVER_CONNECT > WM_USER" && enable vfwcap_defines
3335.
3336. check_type "dshow.h" IBaseFilter
3337.
3338. # check for ioctl_meteor.h, ioctl_bt848.h and alternatives
3339. { check_header dev/bktr/ioctl_meteor.h &&
3340.   check_header dev/bktr/ioctl_bt848.h; } ||
3341. { check_header machine/ioctl_meteor.h &&
3342.   check_header machine/ioctl_bt848.h; } ||
3343. { check_header dev/video/meteor/ioctl_meteor.h &&
3344.   check_header dev/video/bktr/ioctl_bt848.h; } ||
3345. check_header dev/ic/bt8xx.h
3346.
3347. check_header sndio.h
3348. if check_struct sys/soundcard.h audio_buf_info bytes; then
3349.     enable_safe sys/soundcard.h
3350. else
3351.     check_cc -D_BSD_VISIBLE -D_XSI_VISIBLE <<EOF && add_cppflags -D_BSD_VISIBLE -D_XSI_VISIBLE && enable_safe sys/soundcard.h
3352.     #include <sys/soundcard.h>
3353.     audio_buf_info abc;
3354. EOF
3355. fi
3356. check_header soundcard.h
3357.
3358. enabled_any alsa_indev alsa_outdev && check_lib2 alsa/asoundlib.h snd_pcm_htimestamp -lasound
3359.
3360. enabled jack_indev && check_lib2 jack/jack.h jack_client_open -ljack && check_func sem_timedwait
3361.
3362. enabled_any sndio_indev sndio_outdev && check_lib2 sndio.h sio_open -lsndio
3363.
3364. enabled libcdio &&
3365.     check_lib2 "cdio/cdda.h cdio/paranoia.h" cdio_cddap_open "-lcdio_paranoia -lcdio_cdda -lcdio"
3366.
3367. enabled x11grab &&
3368. check_header X11/Xlib.h &&
3369. check_header X11/extensions/XShm.h &&
3370. check_header X11/extensions/Xfixes.h &&
3371. check_func XOpenDisplay -lX11 &&
3372. check_func XShmCreateImage -lX11 -lXext &&
3373. check_func XFixesGetCursorImage -lX11 -lXext -lXfixes
3374.
3375. if ! disabled vaapi; then
3376.     check_lib va/va.h vaInitialize -lva && {
3377.         check_cpp_condition va/va_version.h "VA_CHECK_VERSION(0,32,0)" ||
3378.         warn "Please upgrade to VA-API >= 0.32 if you would like full VA-API support.";
3379.     } || disable vaapi
3380. fi
3381.
3382. if ! disabled vdpau && enabled vdpau_vdpau_h; then
3383.     check_cpp_condition \
3384.         vdpau/vdpau.h "defined VDP_DECODER_PROFILE_MPEG4_PART2_ASP" ||
3385.         { echolog "Please upgrade to libvdpau >= 0.2 if you would like vdpau support." &&
3386.           disable vdpau; }
3387. fi
3388.
3389. enabled debug && add_cflags -g"$debuglevel" && add_asflags -g"$debuglevel"
3390. enabled coverage && add_cflags "-fprofile-arcs -ftest-coverage" && add_ldflags "-fprofile-arcs -ftest-coverage"
3391. test -n "$valgrind" && target_exec="$valgrind --error-exitcode=1 --malloc-fill=0x2a --track-origins=yes --leak-check=full --gen-supp
revisions=all --suppressions=$source_path/tests/fate-valgrind.supp"
3392. #添加一些编译选项
3393. # add some useful compiler flags if supported
3394. check_cflags "x86asm -x86asm" && add_cflags -x86asm

```

```

3394. check_cflags -Wdeclaration-atter-statement
3395. check_cflags -Wall
3396. check_cflags -Wno-parentheses
3397. check_cflags -Wno-switch
3398. check_cflags -Wno-format-zero-length
3399. check_cflags -Wdisabled-optimization
3400. check_cflags -Wpointer-arith
3401. check_cflags -Wredundant-decls
3402. check_cflags -Wno-pointer-sign
3403. check_cflags -Wcast-qual
3404. check_cflags -Wwrite-strings
3405. check_cflags -Wtype-limits
3406. check_cflags -Wundef
3407. check_cflags -Wmissing-prototypes
3408. check_cflags -Wno-pointer-to-int-cast
3409. check_cflags -Wstrict-prototypes
3410. enabled extra_warnings && check_cflags -Winline
3411.
3412. # add some linker flags
3413. check_ldflags -Wl,--warn-common
3414. check_ldflags -Wl,-rpath-link=libpostproc:libswresample:libswscale:libavfilter:libavdevice:libavformat:libavcodec:libavutil
3415. test_ldflags -Wl,-Bsymbolic && append SHFLAGS -Wl,-Bsymbolic
3416.
3417. echo "X{};" > $TMPV
3418. if test_ldflags -Wl,--version-script,$TMPV; then
3419.     append SHFLAGS '-Wl,--version-script,\$(SUBDIR)lib\$(NAME).ver'
3420.     check_cc <<EOF && enable symver_asm_label
3421. void ff_foo(void) __asm__ ("av_foo@VERSION");
3422. void ff_foo(void) { ${inline_asm+__asm__ ($quotes);} }
3423. EOF
3424.     check_cc <<EOF && enable symver_gnu_asm
3425. __asm__ (".symver ff_foo,av_foo@VERSION");
3426. void ff_foo(void) {}
3427. EOF
3428. fi
3429.
3430. if [ -n "$optflags" ]; then
3431.     add_cflags $optflags
3432. elif enabled small; then
3433.     add_cflags $size_cflags
3434. elif enabled optimizations; then
3435.     add_cflags $speed_cflags
3436. else
3437.     add_cflags $noopt_cflags
3438. fi
3439. check_cflags -fno-math-errno
3440. check_cflags -fno-signed-zeros
3441. check_cc -mno-red-zone <<EOF && noredzone_flags="-mno-red-zone"
3442. int x;
3443. EOF
3444.
3445.
3446. if enabled icc; then
3447.     # Just warnings, no remarks
3448.     check_cflags -w1
3449.     # -wd: Disable following warnings
3450.     # 144, 167, 556: -Wno-pointer-sign
3451.     # 1292: attribute "foo" ignored
3452.     # 10006: ignoring unknown option -fno-signed-zeros
3453.     # 10148: ignoring unknown option -Wno-parentheses
3454.     # 10156: ignoring option '-W'; no argument required
3455.     check_cflags -wd144,167,556,1292,10006,10148,10156
3456.     # 11030: Warning unknown option --as-needed
3457.     # 10156: ignoring option '-export'; no argument required
3458.     check_ldflags -wd10156,11030
3459.     # Allow to compile with optimizations
3460.     check_ldflags -march=$cpu
3461.     # icc 11.0 and 11.1 work with ebp_available, but don't pass the test
3462.     enable ebp_available
3463.     if enabled x86_32; then
3464.         test ${icc_version%. *} -ge 11 && \
3465.             check_cflags -falign-stack=maintain-16-byte || \
3466.             disable aligned_stack
3467.     fi
3468. elif enabled ccc; then
3469.     # disable some annoying warnings
3470.     add_cflags -msg_disable cvtu32to64
3471.     add_cflags -msg_disable embedcomment
3472.     add_cflags -msg_disable needconstext
3473.     add_cflags -msg_disable nomainiee
3474.     add_cflags -msg_disable ptrmismatch1
3475.     add_cflags -msg_disable unreachable
3476. elif enabled gcc; then
3477.     check_cflags -fno-tree-vectorize
3478.     check_cflags -Werror=implicit-function-declaration
3479.     check_cflags -Werror=missing-prototypes
3480. elif enabled llvm_gcc; then
3481.     check_cflags -mllvm -stack-alignment=16
3482. elif enabled clang; then
3483.     check_cflags -mllvm -stack-alignment=16
3484.     check_cflags -Qunused-arguments
3485. elif enabled armcc; then

```

```

3486.     # 2523: use of inline assembler is deprecated
3487.     add_cflags -W${armcc_opt},--diag_suppress=2523
3488.     add_cflags -W${armcc_opt},--diag_suppress=1207
3489.     add_cflags -W${armcc_opt},--diag_suppress=1293 # assignment in condition
3490.     add_cflags -W${armcc_opt},--diag_suppress=3343 # hardfp compat
3491.     add_cflags -W${armcc_opt},--diag_suppress=167  # pointer sign
3492.     add_cflags -W${armcc_opt},--diag_suppress=513  # pointer sign
3493. elif enabled tms470; then
3494.     add_cflags -pds=824 -pds=837
3495. elif enabled pathscale; then
3496.     add_cflags -fstrict-overflow -OPT:wrap_around_unsafe_opt=OFF
3497. fi
3498.
3499. enabled_any $THREADS_LIST      && enable_threads
3500.
3501. check_deps $CONFIG_LIST        \
3502.            $CONFIG_EXTRA       \
3503.            $HAVE_LIST          \
3504.            $ALL_COMPONENTS     \
3505.            $ALL_TESTS          \
3506.
3507. enabled asm || { arch=c; disable $ARCH_LIST $ARCH_EXT_LIST; }
3508.
3509. if test $target_os = "haiku"; then
3510.     disable memalign
3511.     disable posix_memalign
3512. fi
3513.
3514. ! enabled_any memalign posix_memalign malloc_aligned &&
3515.     enabled_any $need_memalign && enable memalign_hack
3516.
3517. #在控制台输出信息
3518. echo "install prefix          $prefix"
3519. echo "source path              $source_path"
3520. echo "C compiler                 $cc"
3521. echo "ARCH                       $arch ($cpu)"
3522. if test "$build_suffix" != ""; then
3523.     echo "build suffix                $build_suffix"
3524. fi
3525. if test "$progs_suffix" != ""; then
3526.     echo "progs suffix                 $progs_suffix"
3527. fi
3528. if test "$extra_version" != ""; then
3529.     echo "version string suffix       $extra_version"
3530. fi
3531. #${} 的特异功能：
3532. #${file-my.file.txt}假如 $file 为空值，则使用 my.file.txt 作默认值。(保留没设定及非空值)
3533. #在这里，如果某个变量为空值，则取默认值为no
3534. echo "big-endian                 ${bigendian-no}"
3535. echo "runtime cpu detection       ${runtime_cpudetect-no}"
3536. if enabled x86; then
3537.     echo "${yasmexe}             ${yasm-no}"
3538.     echo "MMX enabled                 ${mmx-no}"
3539.     echo "MMX2 enabled                 ${mmx2-no}"
3540.     echo "3DNow! enabled               ${amd3dnow-no}"
3541.     echo "3DNow! extended enabled      ${amd3dnowext-no}"
3542.     echo "SSE enabled                  ${sse-no}"
3543.     echo "SSSE3 enabled               ${ssse3-no}"
3544.     echo "AVX enabled                  ${avx-no}"
3545.     echo "CMOV enabled                 ${cmov-no}"
3546.     echo "CMOV is fast                 ${fast_cmov-no}"
3547.     echo "EBX available                ${ebx_available-no}"
3548.     echo "EBP available                ${ebp_available-no}"
3549. fi
3550. if enabled arm; then
3551.     echo "ARMv5TE enabled              ${armv5te-no}"
3552.     echo "ARMv6 enabled                ${armv6-no}"
3553.     echo "ARMv6T2 enabled              ${armv6t2-no}"
3554.     echo "ARM VFP enabled              ${armvfp-no}"
3555.     echo "IWMXTE enabled               ${iwmxt-no}"
3556.     echo "NEON enabled                 ${neon-no}"
3557. fi
3558. if enabled mips; then
3559.     echo "MMI enabled                  ${mmi-no}"
3560. fi
3561. if enabled ppc; then
3562.     echo "Altivec enabled              ${altivec-no}"
3563.     echo "PPC 4xx optimizations        ${ppc4xx-no}"
3564.     echo "dcbzl available              ${dcbzl-no}"
3565. fi
3566. if enabled sparc; then
3567.     echo "VIS enabled                  ${vis-no}"
3568. fi
3569. echo "debug symbols                ${debug-no}"
3570. echo "strip symbols                 ${stripping-no}"
3571. echo "optimize for size             ${small-no}"
3572. echo "optimizations                 ${optimizations-no}"
3573. echo "static                       ${static-no}"
3574. echo "shared                       ${shared-no}"
3575. echo "postprocessing support        ${postproc-no}"
3576. echo "new filter support            ${avfilter-no}"

```

```

3577. echo "network support" "${network-no}"
3578. echo "threading support" "${thread_type-no}"
3579. echo "safe bitstream reader" "${safe_bitstream_reader-no}"
3580. echo "SDL support" "${sdl-no}"
3581. echo "Sun medialib support" "${mlib-no}"
3582. echo "libdxva2 enabled" "${dxva2-no}"
3583. echo "libva enabled" "${vaapi-no}"
3584. echo "libvdpau enabled" "${vdpau-no}"
3585. echo "AVISynth enabled" "${avisynth-no}"
3586. echo "frei0r enabled" "${frei0r-no}"
3587. echo "gnutls enabled" "${gnutls-no}"
3588. echo "libaacplus enabled" "${libaacplus-no}"
3589. echo "libass enabled" "${libass-no}"
3590. echo "libcdio support" "${libcdio-no}"
3591. echo "libcelt enabled" "${libcelt-no}"
3592. echo "libdc1394 support" "${libdc1394-no}"
3593. echo "libdirac enabled" "${libdirac-no}"
3594. echo "libfaac enabled" "${libfaac-no}"
3595. echo "libgsm enabled" "${libgsm-no}"
3596. echo "libmodplug enabled" "${libmodplug-no}"
3597. echo "libmp3lame enabled" "${libmp3lame-no}"
3598. echo "libnut enabled" "${libnut-no}"
3599. echo "libopencore-amrnb support" "${libopencore_amrnb-no}"
3600. echo "libopencore-amrwb support" "${libopencore_amrwb-no}"
3601. echo "libopencv support" "${libopencv-no}"
3602. echo "libopenjpeg enabled" "${libopenjpeg-no}"
3603. echo "libpulse enabled" "${libpulse-no}"
3604. echo "librtmp enabled" "${librtmp-no}"
3605. echo "libschrödinger enabled" "${libschrödinger-no}"
3606. echo "libspeex enabled" "${libspeex-no}"
3607. echo "libstagefright-h264 enabled" "${libstagefright_h264-no}"
3608. echo "libtheora enabled" "${libtheora-no}"
3609. echo "libutvideo enabled" "${libutvideo-no}"
3610. echo "libv4l2 enabled" "${libv4l2-no}"
3611. echo "libvo-aacenc support" "${libvo_aacenc-no}"
3612. echo "libvo-amrwbenc support" "${libvo_amrwbenc-no}"
3613. echo "libvorbis enabled" "${libvorbis-no}"
3614. echo "libvpx enabled" "${libvpx-no}"
3615. echo "libx264 enabled" "${libx264-no}"
3616. echo "libxavs enabled" "${libxavs-no}"
3617. echo "libxvid enabled" "${libxvid-no}"
3618. echo "openal enabled" "${openal-no}"
3619. echo "openssl enabled" "${openssl-no}"
3620. echo "zlib enabled" "${zlib-no}"
3621. echo "bzlib enabled" "${bzlib-no}"
3622. echo
3623.
3624. for type in decoder encoder hwaccel parser demuxer muxer protocol filter bsf indec outdev; do
3625.     echo "Enabled ${type}s:"
3626.     eval list=\${(toupper $type)_LIST}
3627.     print_enabled '$_*' $list | sort | pr -r -3 -t
3628.     echo
3629. done
3630.
3631. license="GPL version 2.1 or later"
3632. if enabled nonfree; then
3633.     license="nonfree and unredistributable"
3634. elif enabled gplv3; then
3635.     license="GPL version 3 or later"
3636. elif enabled lgplv3; then
3637.     license="GPL version 3 or later"
3638. elif enabled gpl; then
3639.     license="GPL version 2 or later"
3640. fi
3641.
3642. echo "License: $license"
3643. #创建config.mak和config.h
3644. #根据情况也会创建config.asm
3645. echo "Creating config.mak and config.h..."
3646.
3647. test -e Makefile || ln_s "$source_path/Makefile" .
3648.
3649. enabled stripping || strip="echo skipping strip"
3650. #重要：需要输出的文件
3651. #TMPH是一个临时文件，最终会拷贝给config.h
3652. config_files="$TMPH config.mak"
3653. #写入config.mak文件
3654. #首先写入一些基本信息
3655. #"<<EOF"表示后续的输入作为子命令或子shell的输入，直到遇到"EOF"，再次返回到
3656. #主调shell，可将其理解为分界符（delimiter）。
3657. #最后的"EOF"必须单独占一行
3658. cat > config.mak <<EOF
3659. # Automatically generated by configure - do not modify!
3660. ifndef FFMPEG_CONFIG_MAK
3661. FFMPEG_CONFIG_MAK=1
3662. FFMPEG_CONFIGURATION=FFMPEG_CONFIGURATION
3663. prefix=$prefix
3664. LIBDIR=\$(DESTDIR)$libdir
3665. SHLIBDIR=\$(DESTDIR)$shlibdir
3666. INCDIR=\$(DESTDIR)$incdir
3667. BINDIR=\$(DESTDIR)$bindir

```

```

3668. DATADIR=\$(DESTDIR)$datadir
3669. MANDIR=\$(DESTDIR)$mandir
3670. SRC_PATH=$source_path
3671. ifndef MAIN_MAKEFILE
3672. SRC_PATH:=\$(SRC_PATH:.%=%.%)
3673. endif
3674. CC_IDENT=$cc_ident
3675. ARCH=$arch
3676. CC=$cc
3677. CXX=$cxx
3678. AS=$as
3679. LD=$ld
3680. DEPCC=$dep_cc
3681. YASM=$yasmexe
3682. YASMDEP=$yasmexe
3683. AR=$ar
3684. RANLIB=$ranlib
3685. CP=cp -p
3686. LN S=$ln_s
3687. STRIP=$strip
3688. CPPFLAGS=$CPPFLAGS
3689. CFLAGS=$CFLAGS
3690. CXXFLAGS=$CXXFLAGS
3691. ASFLAGS=$ASFLAGS
3692. AS_0=$CC_0
3693. CC_0=$CC_0
3694. CXX_0=$CXX_0
3695. LDFLAGS=$LDFLAGS
3696. FFSERVERLDFLAGS=$FFSERVERLDFLAGS
3697. SHFLAGS=$SHFLAGS
3698. YASMFLAGS=$YASMFLAGS
3699. BUILDSUF=$build_suffix
3700. PROGSUF=$progs_suffix
3701. FULLNAME=$FULLNAME
3702. LIBPREF=$LIBPREF
3703. LIBSUF=$LIBSUF
3704. LIBNAME=$LIBNAME
3705. SLIBPREF=$SLIBPREF
3706. SLIBSUF=$SLIBSUF
3707. EXESUF=$EXESUF
3708. EXTRA_VERSION=$extra_version
3709. DEPFLAGS=$DEPFLAGS
3710. CCDEP=$CCDEP
3711. CXXDEP=$CXXDEP
3712. ASDEP=$ASDEP
3713. CC_DEPFLAGS=$CC_DEPFLAGS
3714. AS_DEPFLAGS=$AS_DEPFLAGS
3715. HOSTCC=$host_cc
3716. HOSTCFLAGS=$host_cflags
3717. HOSTEXESUF=$HOSTEXESUF
3718. HOSTLDFLAGS=$host_ldflags
3719. HOSTLIBS=$host_libs
3720. TARGET_EXEC=$target_exec
3721. TARGET_PATH=$target_path
3722. SDL_LIBS=$sdl_libs
3723. SDL_CFLAGS=$sdl_cflags
3724. LIB_INSTALL_EXTRA_CMD=$LIB_INSTALL_EXTRA_CMD
3725. EXTRALIBS=$extralibs
3726. INSTALL=$install
3727. LIBTARGET=${LIBTARGET}
3728. SLIBNAME=${SLIBNAME}
3729. SLIBNAME_WITH_VERSION=${SLIBNAME_WITH_VERSION}
3730. SLIBNAME_WITH_MAJOR=${SLIBNAME_WITH_MAJOR}
3731. SLIB_CREATE_DEF_CMD=${SLIB_CREATE_DEF_CMD}
3732. SLIB_EXTRA_CMD=${SLIB_EXTRA_CMD}
3733. SLIB_INSTALL_NAME=${SLIB_INSTALL_NAME}
3734. SLIB_INSTALL_LINKS=${SLIB_INSTALL_LINKS}
3735. SLIB_INSTALL_EXTRA_LIB=${SLIB_INSTALL_EXTRA_LIB}
3736. SLIB_INSTALL_EXTRA_SHLIB=${SLIB_INSTALL_EXTRA_SHLIB}
3737. SAMPLES:=${samples:-\$(FATE_SAMPLES)}
3738. NOREDZONE_FLAGS=$noredzone_flags
3739. EOF
3740. #获取版本
3741. #主要通过各个类库文件夹中的version.h文件
3742. #读取XXX_VERSION (相当于把头文件当成一个文本来读)
3743. get_version(){
3744.     name=$1
3745.     file=$source_path/$2
3746.     # This condition will be removed when we stop supporting old libpostproc versions
3747.     if ! test "$name" = LIBPOSTPROC || test "$postproc_version" = current; then
3748.         eval $(grep "#define ${name}_VERSION_M" "$file" | awk '{ print $2="$3 }')
3749.         eval ${name}_VERSION=${${name}_VERSION_MAJOR}.${${name}_VERSION_MINOR}.${${name}_VERSION_MICRO}
3750.     fi
3751.     lname=$(tolower $name)
3752.     eval echo "${lname}_VERSION=${${name}_VERSION}" >> config.mak
3753.     eval echo "${lname}_VERSION_MAJOR=${${name}_VERSION_MAJOR}" >> config.mak
3754. }
3755. #获取版本
3756. get_version LIBAVCODEC libavcodec/version.h
3757. get_version LIBAVDEVICE libavdevice/avdevice.h
3758. get_version LIBAVFILTER libavfilter/version.h

```

```

3759. get_version LIBAVFORMAT libavformat/version.h
3760. get_version LIBAVUTIL libavutil/avutil.h
3761. get_version LIBPOSTPROC libpostproc/postprocess.h
3762. get_version LIBSWRESAMPLE libswresample/swresample.h
3763. get_version LIBSWSCALE libswscale/swscale.h
3764. #config.h前面需要添加的一些内容（TMPH是一个临时文件，最终会拷贝给config.h）
3765. cat > $TMPH <<EOF
3766. /* Automatically generated by configure - do not modify! */
3767. #ifndef FFMPEG_CONFIG_H
3768. #define FFMPEG_CONFIG_H
3769. #define FFMPEG_CONFIGURATION "$(c_escape $FFMPEG_CONFIGURATION)"
3770. #define FFMPEG_LICENSE "$(c_escape $license)"
3771. #define FFMPEG_DATADIR "$(eval c_escape $datadir)"
3772. #define AVCONV_DATADIR "$(eval c_escape $datadir)"
3773. #define CC_TYPE "$cc_type"
3774. #define CC_VERSION $cc_version
3775. #define restrict $restrict
3776. #define EXTERN_PREFIX "${extern_prefix}"
3777. #define EXTERN_ASM ${extern_prefix}
3778. #define SLIBSUF "$SLIBSUF"
3779. EOF
3780.
3781. test -n "$malloc_prefix" &&
3782.     echo "#define MALLOC_PREFIX $malloc_prefix" >>$TMPH
3783.
3784. if enabled small || disabled optimizations; then
3785.     echo "#undef av_always_inline" >> $TMPH
3786.     if enabled small; then
3787.         echo "#define av_always_inline inline" >> $TMPH
3788.     else
3789.         echo "#define av_always_inline av_unused" >> $TMPH
3790.     fi
3791. fi
3792. #包含yasm
3793. if enabled yasm; then
3794.     append config_files $TMPASM
3795.     printf '' >$TMPASM
3796. fi
3797. #输出所有的配置信息包含3类：
3798. #以“ARCH_”开头，包含系统架构信息
3799. #以“HAVE_”开头，包含系统特征信息
3800. #以“CONFIG_”开头，包含编译配置（数量最多，包含协议、复用器、编解码器等配置，将近1000行）
3801. #config_files
3802. print_config ARCH_ "$config_files" $ARCH_LIST
3803. print_config HAVE_ "$config_files" $HAVE_LIST
3804. print_config CONFIG_ "$config_files" $CONFIG_LIST \
3805.                                     $CONFIG_EXTRA \
3806.                                     $ALL_COMPONENTS \
3807.
3808. #经过测试的组件？
3809. cat >>config.mak <<EOF
3810. ACODEC_TESTS=$(print_enabled -n _test $ACODEC_TESTS)
3811. VCODEC_TESTS=$(print_enabled -n _test $VCODEC_TESTS)
3812. LAVF_TESTS=$(print_enabled -n _test $LAVF_TESTS)
3813. LAVFI_TESTS=$(print_enabled -n _test $LAVFI_TESTS)
3814. SEEK_TESTS=$(print_enabled -n _test $SEEK_TESTS)
3815. EOF
3816.
3817. echo "#endif /* FFMPEG_CONFIG_H */" >> $TMPH
3818. #结束了
3819. echo "#endif # FFMPEG_CONFIG_MAK" >> config.mak
3820.
3821. # 关键：临时文件拷贝至config.h
3822. # Do not overwrite an unchanged config.h to avoid superfluous rebuilds.
3823. # 配置没有变化的时候，不重新生成config.h（重新生成config.h会导致大量文件需要重新编译）
3824. #
3825. cp_if_changed $TMPH config.h
3826. # touch fileA
3827. # 如果fileA不存在，touch指令会在当前目录下新建一个空白文件fileA。
3828. touch .config
3829.
3830. enabled yasm && cp_if_changed $TMPASM config.asm
3831.
3832. cat > $TMPH <<EOF
3833. /* Generated by ffconf */
3834. #ifndef AVUTIL_AVCONFIG_H
3835. #define AVUTIL_AVCONFIG_H
3836. EOF
3837.
3838. test "$postproc_version" != current && cat >> $TMPH <<EOF
3839. #define LIBPOSTPROC_VERSION_MAJOR $LIBPOSTPROC_VERSION_MAJOR
3840. #define LIBPOSTPROC_VERSION_MINOR $LIBPOSTPROC_VERSION_MINOR
3841. #define LIBPOSTPROC_VERSION_MICRO $LIBPOSTPROC_VERSION_MICRO
3842. EOF
3843.
3844. print_config AV_HAVE_ $TMPH $HAVE_LIST_PUB
3845.
3846. echo "#endif /* AVUTIL_AVCONFIG_H */" >> $TMPH
3847.
3848. cp_if_changed $TMPH libavutil/avconfig.h
3849.
3850. test -n "$WARNINGS" && printf "\n$WARNINGS"

```



```

3850.
3851. # build pkg-config files
3852.
3853. pkgconfig_generate(){
3854.     name=$1
3855.     shortname=${name#lib}${build_suffix}
3856.     comment=$2
3857.     version=$3
3858.     libs=$4
3859.     requires=$5
3860.     enabled ${name#lib} || return 0
3861.     mkdir -p $name
3862.     cat <<EOF > $name/$name.pc
3863.     prefix=$prefix
3864.     exec_prefix=${prefix}
3865.     libdir=$libdir
3866.     includedir=$incdir
3867.
3868.     Name: $name
3869.     Description: $comment
3870.     Version: $version
3871.     Requires: $(enabled shared || echo $requires)
3872.     Requires.private: $(enabled shared && echo $requires)
3873.     Conflicts:
3874.     Libs: -L\${libdir} -l\${shortname} $(enabled shared || echo $libs)
3875.     Libs.private: $(enabled shared && echo $libs)
3876.     Cflags: -I\${includedir}
3877.     EOF
3878.     cat <<EOF > $name/$name-uninstalled.pc
3879.     prefix=
3880.     exec_prefix=
3881.     libdir=\${pcfiledir}
3882.     includedir=\${source_path}
3883.
3884.     Name: $name
3885.     Description: $comment
3886.     Version: $version
3887.     Requires: $requires
3888.     Conflicts:
3889.     Libs: \${libdir}/${LIBPREFIX}${shortname}${LIBSUF} $libs
3890.     Cflags: -I\${includedir}
3891.     EOF
3892. }
3893.
3894. pkgconfig_generate libavutil "FFmpeg utility library" "$LIBAVUTIL_VERSION" "$LIBM"
3895. pkgconfig_generate libavcodec "FFmpeg codec library" "$LIBAVCODEC_VERSION" "$extralibs" "libavutil = $LIBAVUTIL_VERSION"
3896. pkgconfig_generate libavformat "FFmpeg container format library" "$LIBAVFORMAT_VERSION" "$extralibs" "libavcodec = $LIBAVCODEC_VERSION"
3897. pkgconfig_generate libavdevice "FFmpeg device handling library" "$LIBAVDEVICE_VERSION" "$extralibs" "libavformat = $LIBAVFORMAT_VERSION"
3898. pkgconfig_generate libavfilter "FFmpeg video filtering library" "$LIBAVFILTER_VERSION" "$extralibs"
3899. pkgconfig_generate libpostproc "FFmpeg postprocessing library" "$LIBPOSTPROC_VERSION" "" "libavutil = $LIBAVUTIL_VERSION"
3900. pkgconfig_generate libswscale "FFmpeg image rescaling library" "$LIBSWSCALE_VERSION" "$LIBM" "libavutil = $LIBAVUTIL_VERSION"
3901. pkgconfig_generate libswresample "FFmpeg audio rescaling library" "$LIBSWRESAMPLE_VERSION" "$LIBM" "libavutil = $LIBAVUTIL_VERSION"

```

雷霄骅

leixiaohua1020@126.com

<http://blog.csdn.net/leixiaohua1020>

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/44587465>

文章标签： [FFmpeg](#) [Configure](#) [Shell](#) [源代码](#) [Make](#)

个人分类： [FFMPEG](#)

所属专栏： [FFmpeg](#)

此PDF由spyyg生成, 请尊重原作者版权!!!

我的邮箱: liushidc@163.com