目 录

1. 安装编译工具

1.1. 安装编译器

Melis 平台使用的编译开发工具为 ARM 公司的集成开发工具 RealView 2.2(全称为 RealView Develper Suit，简称 RVDS)。关于该工具的安装请参考说明文档《REALVIEW 安装说明》。

1.2. 安装 Cygwin

Cygwin 是一个在 Windows 平台上运行的模拟 Unix 环境的工具，是许多自由软件的集合。Melis 平台使用 Cygwin 工具，主要是借助其功能强大的 make 工具。

Cygwin 工具的安装步骤如下：

1. 运行 Cygwin 的安装文件 cygwin.2.00.exe，选择安装路径，自动安装至完成;
2. 进入 Cygwin 的根目录，以编辑方式打开 cygwin.bat 批处理文件，根据 Cygwin 的安装路径修改路径配置（cygwin 默认安装在 C 盘根目录下）。
   如安装在 D 盘根目录下，则对应的修改为：

```
D:
Chdir D:\cygwin\bin
```

3. 用户可在启动脚本里添加自己的启动设置，比如更改工作目录等。
   如 cygwin 安装在 D 盘根目录下，用户名为 kevin，则启动脚本位置为：

```
D:\cygwin\home\kevin\.bash_profile
```

2. 配置编译环境

2.1. 通用配置

在 Melis 平台上编译代码时，需要先执行硬件平台配置工具“ePDK\includes\cfg.bat”对当前的硬件平台和方案类型进行配置。

Melis 平台对平台相关的编译配置文件为“ePDK\includes\cfgs\CROSSTOOL.CFG”，对编译工具、硬件平台及共用库目录作了一些配置，为所有模块共用。该配置文件中相关变量由平台整合人员统一设置，用户可以使用相关的变量，但是不能对变量作修改。相关变量的含义如下：

$(CROSSTOOL)，定义交叉编译工具。

“ARMRVDS”，定义交叉编译工具为 RVDS；

“ARMGCC”，定义交叉编译工具为 ARMGCC；

$(EPDK_CHIP)，定义硬件平台的类型，定义在脚本“ePDK\includes\cfgs\chip.cfg”中，chip.cfg 由用户执行“ePDK\includes\cfg.bat”选择硬件平台时生成。

“EPDK_CHIP_SUNI”，定义当前硬件平台为 suni 平台；

“EPDK_CHIP_SUNII”，定义当前硬件平台为 sunii 平台；

“EPDK_CHIP_SUNIII”，定义当前硬件平台为 suniii 平台；

$(LIBPATH)，定义 melis 平台的共享库目录。

$(INTERLIBPATH)，定义 melis 平台内部共享库目录。

$(WORKSPACEPATH)，定义 melis 平台目标文件及打包工作路径。

$(ESTUDIOROOT)，定义 melis 平台使用的 PC 工具的路径。

$(CC)，定义 C 语言编译工具。

RVDS 交叉编译工具下为“armcc”；

GCC 交叉编译工具下为“arm-elf-gcc”；

$(CFLAGS)，定义 C 语言编译工具的基本配置参数。

$(AS)，定义汇编器工具。

RVDS 交叉编译工具下为“armasm”；

GCC 交叉编译工具下为“arm-elf-as”；

$(ASFLAGS)，定义汇编器的基本配置参数。

$(LINK)，定义链接工具。

RVDS 交叉编译工具下为“armlink”；

GCC 交叉编译工具下为“arm-elf-ld”；

$(LKFLAGS)，定义链接工具基本配置参数。

$(AR)，定义库打包工具。

RVDS 交叉编译工具下为“armar”；

GCC 交叉编译工具下为“arm-elf-ar”；

$(ARFLAGS)，定义库打包工具的基本配置参数。

$(LOAD)，定义加载器工具。

RVDS 交叉编译工具下为“fromelf”；

GCC 交叉编译工具下为“arm-elf-objcopy”；

$(LDFLAGS)，定义加载器工具的基本配置参数。

2.2. 模块本地配置

CROSSTOOL.CFG 仅对所有模块共用的配置作了定义，而每个模块特有的配置需要针对每个模块来作扩展。因此，针对每个模块的 makefile，都存在一个 make.cfg 文件与之对应，在 CROSSTOOL.CFG 的基础上，对相关的变量作扩展。每个模块的 make.cfg 都不尽相同，需要根据模块的具体情况作修改，但一般都包括下面几个部分：

$(ROOT)，定义当前模块的根目录，一般为“.”。

$(SDKROOT)，定义“ePDK”目录相对于$(ROOT)的相对路径，此变量必须定义，CROSSTOOL.CFG 配置文件中会通过此变量来引用“ePDK”目录的路径。

include $(SDKROOT)/includes/cfgs/CROSSTOOL.CFG，引用编译工具通用配置。

$(INCLUDES)，定义所有需要引用的头文件的路径。

$(LIBS)，定义需要引用的库文件。

$(SRCDIRS)，定义所有需要引用的源文件的路径，一般采用自动扫描的方式来定义，不需要逐项列出。

$(TARGET)，定义需要输送出去的目标文件，一般不包含调试信息。

$(LOCALTARGET)，定义本地生成的目标文件，一般命名为“__image.axf”，包含有完整的调试信息，用作调试。

$(LINK_SCT)，定义链接程序使用的链接脚本。

除此以外，还需要对“CFLAGS”、“ASFLAGS”、“LKFLAGS”、“LDFLAGS”等相关工具配置参数做相应的扩展，以满足模块编译的特定需求。

一个常用的 make.cfg 示例如下：

```
ROOT    = .
SDKROOT = $(ROOT)/../..

#导入交叉编译器通用配置
include $(SDKROOT)/includes/cfgs/CROSSTOOL.CFG

#头文件路径列表
INCLUDES        =   -I$(ROOT)                       \
                    -I$(ROOT)/../include            \
                    -I$(SDKROOT)/includes            \
                    -I$(SDKROOT)/includes/emod

#库文件列表
LIBS            =   $(LIBPATH)/elibs.a

#列出该工程下的所有包含源程序的子目录
SRCDIRS  =  $(shell find . -maxdepth 5 -type d)

INCLUDES        :=  $(INCLUDES)         \
                    $(foreach dir,$(SRCDIRS),-I$(dir))

#定义生成的目标文件(输出/本地)
```

```
TARGET      = $(WORKSPACEPATH)/liveclick/rootfs/apps/anole.axf
LOCALTARGET = __image.axf

ifeq ($(CROSSTOOL), ARMRVDS)

#=================================================================================
#使用 RVDS 编译器
#=================================================================================
#程序链接脚本
LINK_SCT    = --scatter=$(ROOT)/config/anole.sct
#编译器扩展参数
CFLAGS      := $(CFLAGS) -O3 --debug -DEPDK_DEBUG_LEVEL=EPDK_DEBUG_LEVEL_LOG_ALL
CFLAGS      := $(CFLAGS) $(INCLUDES)
#汇编器扩展参数
ASFLAGS     := $(ASFLAGS) --debug  -keep
ASFLAGS     := $(ASFLAGS) $(INCLUDES)
#链接器扩展参数
LKFLAGS     := $(LKFLAGS) $(LINK_SCT)
#加载器扩展参数
LDFLAGS     := $(LDFLAGS) --elf --no_debug  -output

LIBS        := $(LIBS) $(LIBPATH)/lemon.sym

else

#=================================================================================
#使用 GNU-GCC 编译器
#=================================================================================
error:
    $(error GNU Cross-tool is invalid now!!!)

endif
```

2.3. 简单的 makefile

Melis 平台采用 makefile 的隐含规则完成对所有源文件的编译。没有启用完整的依赖规则，只有当源码文件(*.c、*.s)文件发生修改后，才会重新编译该源码文件(未修改的源码文件不会被重新编译)，修改头文件(*.h)不会引发对源码文件的重新编译，因此，当修改了相关的头文件以后，必须先执行 clean，再重新编译。一个基本的 makefile 一般包括以下几个部分：

include　make.cfg，引用 makefile 的配置文件。

$(SRCCS)，通过自动扫描获得的*.c 源文件列表。

$(SRCSS)，通过自动扫描获得的*.s 源文件列表。

$(OBJS)，通过后缀替换规则从$(SRCCS)和$(SRCSS)获得的*.o 文件列表，*.o 文件

通过 makefile 的隐含规则自动编译$(SRCCS)和$(SRCSS)获得。

$(LOCALTARGET):$(OBJS)，链接相关的*.o 和库文件得到本地目标文件。

all:$(LOCALTARGET)，通过本地文件得到输出目标文件，该符号也是 makefile 的默认入口。

clean，清理生成的临时文件。

一个简单的 makefile 示例文件如下：

```
#导入编译器配置
include make.cfg

#从所有子目录中得到源代码的列表
SRCCS=$(foreach dir,$(SRCDIRS),$(wildcard $(dir)/*.c))
SRCSS=$(foreach dir,$(SRCDIRS),$(wildcard $(dir)/*.s))

#得到源代码对应的目标文件的列表
OBJS=$(SRCCS:.c=.o) $(SRCSS:.s=.o)

#生成输出目标文件
all:
    make builders
    make application

buildres:
    $(ESTUDIOROOT)/Softwares/xmlang/xmlang.exe ./res/lang.xml
    $(ESTUDIOROOT)/Softwares/ResCompile/langcompile.exe ./res/lang.cfg
    $(ESTUDIOROOT)/Softwares/ResCompile/themcompile.exe ./res/them.cfg

application:$(LOCALTARGET)
    $(LOAD) $(LDFLAGS) $(TARGET) $(LOCALTARGET)
    @echo ----------------------------
    @echo target make finish
    @echo ----------------------------

#生成本地目标文件
$(LOCALTARGET):$(OBJS)
    $(LINK) $(LKFLAGS) -o $@ $(filter %.o ,$+) $(LIBS)

# 删除生成的中间文件
clean:
    -rm $(OBJS) $(LOCALTARGET)
```

需要注意的是，makefile 的命令前必须使用“TAB”键，不可以用空格。