E169F 基线需求定制总结

1、 项目基本配置信息

1.1 项目配置信息

（1） 在 wind/config/下面添加 CONFIG_E169F_XXX.mk 文件，文件内容如下：

```
WIND_PORDUCT_NAME=P635F33
WIND_DEVICE_NAME=ZTE_T620
WIND_PRODUCT_BRAND=ZTE
WIND_PRODUCT_MODEL=ZTE T620
WIND_PHONE_NAME=ZTE Blade X3
WIND_PRODUCT_MANUFACTURER=ZTE
WIND_PORDUCT_BOARD=MT6735P
WIND_PORDUCT_TIMEZONE=Europe/Moscow
WIND_PORDUCT_NOTIFICATION_SOUND=Glow.ogg 默认通知音铃声
WIND_PORDUCT_ALARM_ALERT=
WIND_PORDUCT_RINGTONE=The_party.ogg 默认来电铃声
WIND_PORDUCT_DATEFORMAT=yyyy-MM-dd 默认日期格式
WIND_CUS_MIN_MATCH=7
WIND_CUS_EMAIL_SIGNATURE=wind-mobi 邮件签名
WIND_APN_CONF=apns-conf-E169L_Asia.xml
WIND_SPN_CONF=spn-conf-E169L.xml
WIND_VOICEMAIL_CONF=voicemail-conf-E169L.xml
WIND_MTP_NAME=ZTE T620
```

（2）在 wind/custom_files/device/ginreen/E169F/下添加 versionXXX 文件，内容如下：

```
INVER=DIS_IT_P635F33V1.0.0B01
OUTVER=DIS_IT_A452V1.0.0
#PROVINCE=
#OPERATOR=
```

1.2 默认 WiFi 热点名称、蓝牙名称

在 wind/custom_files/device/ginreen/E169F/custom.conf_XXX 中(没有 custom.conf 的话就从别的项目中拷贝一份过来)

修改：

```
wlan.SSID = ZTE Blade X3
bluetooth.HostName = ZTE Blade X3
```

1.3 双卡或单卡的配置

L 版本默认就是支持双 SIM 卡，可查看相关宏控：

device/ginreen/E169F/ProjectConfig.mk 中

```
GEMINI = yes
MTK_GEMINI_ENHANCEMENT = yes
MTK_DISABLE_CAPABILITY_SWITCH = no
MTK_SHARE_MODEM_SUPPORT = 2
MTK_SHARE_MODEM_CURRENT = 2
```

如果是单 SIM 卡，宏控如下：

GEMINI = no

MTK_GEMINI_ENHANCEMENT = no

MTK_DISABLE_CAPABILITY_SWITCH = yes

MTK_SHARE_MODEM_SUPPORT = 2

MTK_SHARE_MODEM_CURRENT = 1

2、 UI 类客制化

2.1 开机 LOGO，bmp 格式

开机 **logo** 分为 **uboot logo** 和 **kernel logo**，也就是开机时显示的第一屏， 开机时通过 **ProjectConfig.mk** 中的 **BOOT_LOGO=XXX** 来定义用哪个的目录的开机第一屏图片。 我们新增一个开机第一屏定制目录：**BOOT_LOGO= e169f_asi_hd720**，并且在 **bootable/bootloader/lk/project/E169F.mk** 文件这个编译 **lk** 的 **mk** 文件中将 **BOOT_LOGO** 的值也设置为 **e169f_asi_hd720**，

在 **bootable/bootloader/lk/dev/logo** 目录增加 **e169f_asi_hd720** 目录，注意图片的分辨率要和 **ProjectConfig.mk** 中的宽高值一致，不然开机第一屏不会显示。

$ cp cmcc_lte_hd720 e169f_asi_hd720 -r

将 **e169f_asi_hd720** 目录中所有文件前缀修改为 **e169f_asi_hd720**，替换开机第一屏图片。 另需要在

/bootable/bootloader/lk/target/D260/include/target/cust_display.h

文件找到相应的分辨率的位置，在此行的后面添加 **|| defined(**E169F_ASI_HD720**)**。 注意：文件夹名称后面的后缀，例如：**e169f_asi_hd720** 的 **hd720** 是跟手机的分辨率一致 **cust_display.h** 添加的**|| defined(**E169F_ASI_HD720**)**中的 E169F_ASI_HD720 与 **e169f_asi_hd720** 目录名一致

2.2 开机动画、关机动画

客制化动画、铃声我们先在 wind/custom_files/frameworks/base/data/sounds/目录下新建 E169F 目录，把我们项目的动画铃声放在其中。

我们需要修改 frameworks/base/data/sounds/AllAudio.mk 文件中的下面位置：
使编译时使用我们自己创建的 E169FASIAudio.mk 文件，然后我们再在我们自己创建的文件中进行客制化。创建 E169FASIAudio.mk 文件以后要修改资源文件路径

```
ifeq ($(TARGET_PRODUCT),full_E169F)
  ifeq ($(strip $(WIND_PROJECT_NAME_CUSTOM)),E169F_ASI)
  $(call inherit-product, frameworks/base/data/sounds/E169FASIAudio.mk)
  else
  $(call inherit-product, frameworks/base/data/sounds/E169FAudio.mk)
  endif
else
```

LOCAL_PATH:= frameworks/base/data/sounds/E169F/ASI

注意:**AllAudio** 文件中包含了所有的铃声，修改是最好是只修改对应的要修改的铃声不要全部删 除。

开机时播放的动画，若定制，资源要求:动画帧，不超过 50 张，png 格式，位深 24，分辨率请参 照产品定义

我们客制化开机动画的路径：frameworks/base/data/sounds 目录
在 ASI 目录下面创建 bootanimation 和 shutanimation 的文件夹，之后将制作的开机动画和关机动画放在相应的目录下即可。 编译脚本：
frameworks/base/data/sounds/E169FASIAudio.mk 文件
关机动画需通过 mk 脚本编译到相应的位置，在这里我们添加 E169FASIAudio.mk 进行编译，

```
LOCAL_PATH:= frameworks/base/data/sounds/E169F/ASI

#bootanimation
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/bootanimation/bootanimation.zip:system/media/bootanimation.zip

#bootaudio
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/bootaudio/bootaudio.mp3:system/media/bootaudio.mp3

#shutanimation
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/shutanimation/shutanimation.zip:system/media/shutanimation.zip
```

开关机动画的制作:

分别创建名为“**part0**”和“**part1**”的文件夹以及一个名为“**desc.txt**”文件。“**part0**”中存储动画的第一阶段的资源图片，“**part1**”存储第二阶段的资源图片，注意图片为 **png** 格式。 播放控制由“**desc.txt**”指定，内容如下:

480 854 12
part0 p 0 0
part1

各参数功能如下：（注意:**desc.txt** 文本内容必须用单个空格隔开，且不能有多余空行。）

480	854	12	
宽	高	每秒播放帧数	
p	1	0	part0
标志符	循环次数	阶段切换间隔时间	对应目录名
p	0	0	part1
标志符	循环次数	阶段切换间隔时间	对应目录名

最后，将这三个组件通过存储压缩的方式压缩为 **bootanimation.zip** 文件即制作完成。
关机动画的制作也是如此，将这三个组件通过存储压缩的方式压缩为 **shutanimation.zip** 文件，注意:在压缩之后不能产生目录结构，也就是在压缩文件中不能有文件夹存在。

2.3 开机铃声、关机铃声

伴随开机动画的铃声，若定制，资源要求:ogg 格式或 mp3 格式，铃声长度不超过 5s

编译脚本：frameworks/base/data/sounds/E169FASIAudio.mk 文件

```
#bootaudio
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/bootaudio/bootaudio.mp3:system/media/bootaudio.mp3
```

修改对应的铃声还需要修改编译的语句：

例如：添加闹铃要把放在 frameworks/base/data/sounds/E169F 目录下的 Alarm_Beep_01.ogg 添加进去就要在 E169FASIAudio.mk 增加下面语句

```
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/Alarm_Beep_01.ogg:system/media/audio/alarms/Alarm_Beep_01.ogg \
```

2.4 桌面布局

android5.1 overlay 目录：**device/ginreen/E169F /**overlay

2.4.1 桌面图标定制化

主桌面布局，集成壁纸均需要在 overlay 中修改。

SourceFile: packages/apps/Launcher2/res/xml/default_workspace.xml 【4.2，注意 overlay】
SourceFile: packages/apps/Launcher3/res/xml/default_workspace.xml 【4.4，注意 overlay】
SourceFile: packages/apps/Launcher3/res/xml/default_workspace_4X4.xml 【5.1，注意 overlay】

在此文件夹中可以选择是定制什么类型的图标。

可以通过创建 folder 的方式添加 Google 应用的集合

```
//default_workspace.xml中，支持的标签有:
favorite:应用程序快捷方式。
shortcut:链接，如网址，本地磁盘路径等。
search:搜索框。
clock:桌面上的钟表Widget

//支持的属性有:
launcher:title:图标下面的文字，目前只支持引用，不能直接书写字符串;
launcher:icon:图标引用;
launcher:uri:链接地址,链接网址用的，使用shortcut标签就可以定义一个超链接，打开某个网址。
launcher:packageName:应用程序的包名;
launcher:className:应用程序的启动类名;
launcher:screen:图标所在的屏幕编号;
launcher:x:图标在横向排列上的序号;
launcher:y:图标在纵向排列上的序号;
```

2.4.2 壁纸的定制与默认壁纸

壁纸资源放置在 xx/resource_overlay/xx/packages/apps/Launcher3/res/drawable-hdpi 修改 xx/resource_overlay/xx/packages/apps/Launcher3/res/values-nodpi/wallpapers.xml

默认壁纸放置在 xx/resource_overlay/xx/frameworks/base/core/res/res/drawable-nodpi 目录，命名为 default_wallpaper.jpg。【Android4.4 默认壁纸不需要放到壁纸资源目录，也不需要在 wallpapers.xml
中集成，但是 Android4.2 需要】

添加壁纸：

在 Launcher3/WallpaperPicker/res/drawable-xxx 的文件夹下增加 wallpaer 的图片，每个 wallpaper 需要两种图片一张原图一张缩略图，如下形式

wallpaper_01.jpg

wallpaper_01_small.jpg

wallpaper_02.jpg

wallpaper_02_small.jpg

缩略图的文件名必须原图"文件名"+"_small"

在 Launcher3/WallpaperPicker/res/values-nodpi 的 wallpapers.xml 中修改如下：

<resources>

<string-array name="wallpapers"
translatable="false"> <item>wallpaper_01</item>
<item>wallpaper_02</item>

</string-array>
< /resources>

注意：以上的修改都在 overlay 中进行！

2.5 来电铃声的定制以及默认铃声

支持多首闹钟铃声、来电铃声、短信铃声以及系统铃声的定制，若定制，资源要求：ogg/mp3 格式；请选择各种铃声的默认铃声，在相应的<默认 XX>栏填上它的名称即可

编译脚本：修改 frameworks/base/data/sounds/E169FASIAudio.mk 文件
来电铃声 copy 到 system/media/audio/ringtones

通知铃声 copy 到 system/media/audio/notifications

闹钟铃声 copy 到 system/media/audio/alarms

默认铃声配置：在 wind/config/CONFIG_E169F_XXX.mk 修改的是
WIND_PORDUCT_NOTIFICATION_SOUND=Glow.ogg
WIND_PORDUCT_ALARM_ALERT=
WIND_PORDUCT_RINGTONE=The_party.ogg

3、 紧急号码定制

L 版本紧急号码 Customer 的部分改成了在 XML 文件中来配置，文件的路径：

Vendor/mediatek/proprietary/external/EccList

EccList 文件夹中会包含 ecc_list.xml，以及与运营商有关的 ecc_list_OP01.xml、ecc_list_OPXX.xml 等对

应文件，此外还包括一个 EccList.mk 的 Makefile。实际运行中会根据 Makefile 文件中的定义匹配对应的 XML 文件作为判断是否是紧急号码的来源。

我们可以添加 ecc_list_E169F.xml 文件来定制化紧急拨号项目，在 EccList.mk 文件中编译的时候添加判断为 E169F 项目的时候讲此文件编译进去：

下面是 ecc_list.xml 文件中的内容：

```
<?xml version="1.0" encoding="utf-8"?>
<EccTable>
    <!--
        The attribute definition for tag EccEntry:
        - Ecc: the emergnecy number
        - Category: the service category
        - Condition: there are following values:
            - 0: ecc only when no sim
            - 1: ecc always
            - 2: MMI will show ecc but send to nw as normal call
    -->
    <EccEntry Ecc="911" Category="0" Condition="1" />
    <EccEntry Ecc="000" Category="0" Condition="1" />
    <EccEntry Ecc="08" Category="0" Condition="1" />
    <EccEntry Ecc="110" Category="0" Condition="1" />
    <EccEntry Ecc="118" Category="0" Condition="1" />
    <EccEntry Ecc="119" Category="0" Condition="1" />
    <EccEntry Ecc="999" Category="0" Condition="1" />
    <!--<EccEntry Ecc="100" Category="0" Condition="2" />
    <EccEntry Ecc="101" Category="0" Condition="2" />
    <EccEntry Ecc="102" Category="0" Condition="2" />-->
    <EccEntry Ecc="112" Category="0" Condition="1" />
</EccTable>
```

*说明：

一 、添加号码请注意 Condition 的配置，根据需求来选择对应的值。

0:表示在无卡的时候当紧急号码；

1:表示始终当紧急号码；

2:表示界面上显示成紧急拨号，但实际以普通方式拨出。

二 、Category 属性的设置于语音台选择有关，只有在实际拨打紧急号码的时候会将此号码配置的 Category 属性发送到 Modem。国内默认都是‘0’，国外根据实际情况选择。

4、 语言和输入法定制

4.1 语言和默认语言

修改 **device/ginreen/E169F/full_E169F.mk** 文件中的 **PRODUCT_LOCALES** 配置，添加所需要的语言，将默认语言配置成第一个。

4.2 默认输入法

修改文件 **wind/custom_files/device/ginreen/E169F/overlay/XXX/frameworks/base/packages/SettingsProvider/res/values/ defaults.xml**

将 **def_input_method** 的值设置为所需的输入法；

比如 **android** 默认输入法，则设为

<string name="def_input_method">"com.android.inputmethod.latin/.LatinIME"</string>

4.3 增加支持的输入法

当然还可以在 Settings 中做修改， 添加多个输入法， 系统默认的输入法是由字段 DEFAULT_INPUT_METHOD 来控制，而可使用输入法列表是由 ENABLE_INPUT_METHOD 来确定目前手机可支持的输入法，此时，先判断属性 ro. mtk_default_ime 的值，然后根据需求添加：

```
String enabledMethod = SystemProperties.get("ro.mtk_enabled_method","null");
if(!"null".equals(enabledMethod)){
    loadSetting(stmt, Settings.Secure.ENABLED_INPUT_METHODS, "com.android.inputmethod.latin/.LatinIME:"
    + enabledMethod + ":com.android.inputmethod.latin/LatinIME:
    com.google.android.googlequicksearchbox/com.google.android.voicesearch.ime.VoiceInputMethodService");
    loadSetting(stmt, Settings.Secure.DEFAULT_INPUT_METHOD,SystemProperties.get("ro.mtk_default_ime",
    "com.android.inputmethod.latin/LatinIME"));
}
```

添加 ENABLE_INPUT_METHOD 时，要注意，不同的输入法之间使用“:”隔开。

5、 UA Agent 定制

某些国家或地区的运营商或某些网站(通常，海外项目请务必为手机配置 UA)会检查手机 UA，如发现 UA 未注册或 UA 提供的手机能力不适合某网站，网络可能会禁止提供或提供“非期望”的服务， 如:某些网页不能登陆、网页内某些链接不能进入、不能下载某些网络资源等。 如果有客制化的需求，具体配置在 device/ginreen/E169F/custom.conf 中，配置方法是将所有的 Agent 粘贴在文档下面即可。

```
browser.UserAgent = Athens15_TD/V2 Linux/3.0.13 Android/4.0 Release/02.15.2012 Browse
browser.UAProfileURL = http://218.249.47.94/Xianghe/MTK_LTE_Phone_L_UAprofile.xml
mms.UserAgent = Android-Mms/2.0
mms.UAProfileURL = http://www.google.com/oha/rdf/ua-profile-kila.xml
```

如果没有客制化的需求，则不需要添加，系统有默认值，存放在 vendor/mediatek/proprietary/frameworks/base/custom/custom.conf。

6、 APN 定制

Apn 的配置路径是:/mediatek/proprietary/frameworks/base/telephony/etc/apn-config.xml 可根据需要添加或修改。Apn 添加时可根据项目的需求在当前目录 etc 目录下新建一个文件 apn-config-E169F.xml，里面配置所需要的 apn 选项，然后在 CONFIG_E169F_XXX.mk 中将这个文件编译到指定位置

system/etc/apn-config.xml，如下：编译路径：wind/config/CONFIG_E169F_XXX.mk
WIND_APN_CONF=apns-conf-E169L_Asia.xml

验证方法：

编译完成，将版本 down 进手机，使用 DDMS 将 data/data/com.android.providers.telephony 里面的数据库 telephony.db Push 出来，查看里面有个 carrier 的表，这里面会存放当前手机中保存的所有 APN 的信息。

手机第一次开机的时候会将 anps-conf.xml 里所有的 apn 都读到 这个表里。若是添加成功即可以看到该表里面有新添加的项

7、 浏览器和搜索引擎的定制

7.1 浏览器

修改 homepage 和 bookmarks Sourcefile:

device/ginreen/E169F/overlay/packages/apps/Browser/res/values/strings.xml
(修改 homepage_base 和紧接着下面的 bookmarks)

Sourcefile:
device/ginreen/E169F/overlay/packages/apps/Browser/res/values/mtk_strings.xml [此文件需要观察 homepage_base_site_navigation 是否符合要求以作修改]

7.2 搜索引擎

搜索引擎的配置放在 overlay 中路径是：
device/ginreen/E169F/overlay/vendor/mediatek/framework/proprietary/frameworks/base/core/res/res/values/donottranslate-new-search_engines.xml 中。可根据不同的语言进入不同的 values 目录下配置不同的搜索引擎。
譬如在中文中的搜素引擎默认配置成百度：

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
  <string-array name="new_search_engines" translatable="false">
    <item>--</item>
    <item>baidu_English--Baidu--baidu.com--search_engine_baidu--http://m.baidu.com/s?ie=utf8&amp;
        word={searchTerms}--UTF-8--http://suggestion.baidu.com/su?wd={searchTerms}</item>
    </string-array>
</resources>
```

在<item>--</item>后添加的原因是，searchEngineManager 会拿第一个作为默认搜索引擎。

frameworks/base/services/core/java/com/mediatek/search/SearchEngineManagerService.java 中：

```
if (mDefaultSearchEngine == null) {
    mDefaultSearchEngine = mSearchEngineInfos.get(0);
}
```

8、 日期和时区的定制

8.1 默认日期格式

日期格式若无定制，则不需修改，保持默认即可。若有定制，一般修改
wind/config/CONFIG_E169F_XXX.mk
WIND_PORDUCT_DATEFORMAT=yyyy-MM-dd

8.2 时区

若客户定制，需明确不插卡时默认时区，配置在 CONFIG_E169F_XXX.mk
WIND_PORDUCT_TIMEZONE=Europe/Moscow

8.3 默认时间格式

24 小时制：WIND_DATEFORMAT_IS_24= yes
12 小时制：WIND_DATEFORMAT_IS_12= yes

9、 号码匹配的定制

修改 PhoneNumberExt.java
(vendor/mediatek/proprietary/frameworks/base/packages/FwkPlugin/src/com/mediatek/op/telephony)中的 getMinMatch() 返回值

```
public int getMinMatch() {
    //shengbotao 20150728 add start
    String minMatchLen = SystemProperties.get("ro.product.minmatch");
    if (!minMatchLen.isEmpty()) {
        return Integer.parseInt(minMatchLen);
    }
    //shengbotao 20150728 add end
    return 7;
}
```

修改 CONFIG_E169F_XXX.mk 中的：
WIND_CUS_MIN_MATCH=7

10、 APK 预装说明

10.1 预装无源码 APK

10.1.1 可卸载且恢复出厂设置可恢复

（1）在 vendor\mediatek\proprietary\binary\3rd-party\free 下面以需要预置的 APK 名字创建文件夹，以预置一个名为 Test 的 APK 为例
（2）将 Test.apk 放入 vendor\mediatek\proprietary\binary\3rd-party\free\Test 下面
（3）在 vendor\mediatek\proprietary\binary\3rd-party\free\Test 下面创建文 件 Android.mk，文件内容如下(红色字体为备注)：
LOCAL_PATH := $(call my-dir) 设置当前模块的编译路径为当前文件夹路径。即当前 Android.mk 所在 的目录

include $(CLEAR_VARS) 清除变量
Module name should match apk name to be installed
LOCAL_MODULE := Test 要和 apk 的名字一样
LOCAL_MODULE_TAGS := optional 指该模块在所有版本下都编译
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk 指定源 apk
LOCAL_MODULE_CLASS := APPS 指定文件类型是 apk 文件，并会检查是否是 apk 文件
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX) module 的后缀，=.apk
LOCAL_CERTIFICATE := PRESIGNED 表示这个 apk 已经签过名了，不需要再签名
LOCAL_MODULE_PATH := $(TARGET_OUT)/vendor/operator/app 指定最后的目标安装路径
include $(BUILD_PREBUILT)

（4）在 device/ginreen/E169F/device.mk 中 添加：PRODUCT_PACKAGES += Test

10.1.2 可卸载且恢复出厂设置不可恢复

（1）在 packages/apps 下面以需要预置的 APK 名字创建文件夹，以预置 一个名为 Test 的 APK 为例
（2）将 Test.apk 放到 packages/apps/Test 下面
（3）在 packages/apps/Test 下面创建文件 Android.mk，文件内容如下：

```
LOCAL_PATH := $(call my-dir) include
$(CLEAR_VARS)

# Module name should match apk name to be installed
LOCAL_MODULE := Test
LOCAL_MODULE_TAGS := optional LOCAL_SRC_FILES :=
$(LOCAL_MODULE).apk

LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := PRESIGNED

LOCAL_MODULE_PATH := $(TARGET_OUT_DATA_APPS) include
$(BUILD_PREBUILT)
```

（10） 在 device/ginreen/E169F/device.mk 中 添加：PRODUCT_PACKAGES += Test

10.1.3 不可卸载

（1）在 packages/apps 下面以需要预置的 APK 名字创建文件夹，以预置 一个名为 Test 的 APK 为例

（2）将 Test.apk 放到 packages/apps/Test 下面

（3） 在 packages/apps/Test 下面创建文件 Android.mk，文件内容如下：

```
LOCAL_PATH := $(call my-dir) include $(CLEAR_VARS)
# Module name should match apk name to be installed LOCAL_MODULE := Test
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := PRESIGNED
LOCAL_PRIVILEGED_MODULE := true
```

有这一句 表示是安装在 system/priv-app 下面；如果没有，表示是安 装在 system/app 下面

```
#LOCAL_PREBUILT_JNI_LIBS:= \
#@lib/armeabi/libtest.so \
#@lib/armeabi/libtest2.so
include $(BUILD_PREBUILT)
```

注：

解压 Test.apk 文件，并且查看。

若无 so，注释或删除 LOCAL_PREBUILT_JNI_LIBS 及其相关部分

若有 so，使用 LOCAL_PREBUILT_JNI_LIBS 列出所有 so 的路径，不要忘 记使用@。@ 标识符会将 apk 中的 so 抽离出来 build 进 system/lib 或者 system/lib64 中

若 apk 支持不同 cpu 类型的 so，针对 so 的部分的处理:

```
ifeq ($(TARGET_ARCH),arm)
LOCAL_PREBUILT_JNI_LIBS := \
@lib/armeabi-v7a/xxx.so\
@ lib/armeabi-v7a/xxxx.so
else ifeq ($(TARGET_ARCH),x86)
LOCAL_PREBUILT_JNI_LIBS := \
@lib/x86/xxx.so
else ifeq ($(TARGET_ARCH),arm64)
LOCAL_PREBUILT_JNI_LIBS := \
@lib/armeabi-v8a/xxx.so
endif
```

即将和 TARGET_ARCH 对应的 so 抽离出来

（4） 在 device/ginreen/E169F/device.mk 中 添加：PRODUCT_PACKAGES += Test

10.2 预装有源码 APK

在 packages/apps 下面以需要预置的 APK 的 名字创建一个新文件夹，以预置 一个名为 Test 的 APK 为例

（1） 将 Test APK 的 Source code 拷贝到 Test 文件夹下，删除/bin 和/gen 目录

（2） 在 Test 目录下创建一个名为 Android.mk 的文件，内容如下：

```
LOCAL_PATH:= $(call my-dir) include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
```

LOCAL_SRC_FILES := $(call all-subdir-java-files)

LOCAL_PACKAGE_NAME := Test include $(BUILD_PACKAGE)

在 device/ginreen/E169F/device.mk 中 添加：PRODUCT_PACKAGES += Test

10.3 预装注意说明

若需要 apk 作为 32bit 的 apk 运行，则需要在 Android.mk 中定义

LOCAL_MULTILIB :=32

因为 L 版本默认的 LOCAL_MULTILIB :=64

但是为了保险起见，也可以设置 **LOCAL_MULTILIB :=both**

另外，关于 **LOCAL_CERTIFICATE**:

1.系统中所有使用 android.uid.system 作为共享 UID 的 APK，都会首先在 manifest 节点中增加 android:sharedUserId="android.uid.system"，然后在 Android.mk 中增加 LOCAL_CERTIFICATE:= platform。

2. 系统中所有使用 android.uid.shared 作为共享 UID 的 APK，都会在 manifest 节点中增加 android:sharedUserId="android.uid.shared"，然后在 Android.mk 中增加 LOCAL_CERTIFICATE:= shared。

3.系统中所有使用 android.media 作为共享 UID 的 APK，都会在 manifest 节点中增加 android:sharedUserId="android.media"，然后在 Android.mk 中增加 LOCAL_CERTIFICATE:= media

11、 硬件版本号定义

(1) device/ginreen/E169F/ProjectConfig.mk 中，修改:

MTK_CHIP_VER = MBV1.0

(2) device/ginreen/E169F/device.mk 中，修改:

PRODUCT_PROPERTY_OVERRIDES += ro.mediatek.chip_ver=MBV1.0

12、 项目分区，默认共享分区

共享分区有一个宏开关控制在 ProjectConfig.java 中

```
MTK_SHARED_SDCARD = yes
```

只需要将这个宏开关的值设为 yes 就行了

13、 关机充电图标

Android5.1 默认使用的是开机 logo

14、 照片--详情中 **Maker** 和 **Model** 默认信息

照片详情中的信息在如下路径修改:
vendor/mediatek/proprietary/hardware/mtkcam/exif/camera/CamExif.cpp

```
{
    char make[PROPERTY_VALUE_MAX] = {'\0'};
    char model[PROPERTY_VALUE_MAX] = {'\0'};
    property_get("ro.product.manufacturer", make, "0");
    property_get("ro.product.model", model, "0");
    MY_LOGI("property: make(%s), model(%s)", make, model);
    // [Make]
    if ( ::strcmp(make, "0") != 0 ) {
        ::memset(pexifApp1Info->strMake, 0, 32);
        ::strncpy((char*)pexifApp1Info->strMake, (const char*)make, 32);
    }
    // [Model]
    if ( ::strcmp(model, "0") != 0 ) {
        ::memset(pexifApp1Info->strModel, 0, 32);
        ::strncpy((char*)pexifApp1Info->strModel, (const char*)model, 32);
    }
}
```

这里可以看到通过读取上面两个属性 ro.product.manufacturer 和 ro.product.model 的方式来 设置 Maker 和 Model 的值。
若有定制，修改 wind/config/CONFIG_E169F_XXX.mk 中的：
WIND_PRODUCT_CAMERA_MANUFACTURER=ZTE
WIND_PRODUCT_CAMERA_MODEL=ZTE Blade A452

15、 网络切换

平台默认卡槽 2 只支持 2G 网络，需软件开宏使卡槽 2 可以通过软件自动切换为 3G 网络或 4G 网络 路径:
device/ginreen/E169F/ProjectConfig.mk

```
MTK_DISABLE_CAPABILITY_SWITCH = no
```

确保 MTK_DISABLE_CAPABILITY_SWITCH 的值为 no，如果为 yes，网络将不能进行切换。

16、 关闭移动联通定制的 OP01、OP02 的宏

修改路径 device/ginreen/E169F/ProjectConfig.mk 文件中的

```
OPTR_SPEC_SEG_DEF = NONE
```

OPTR_SPEC_SEG_DEF 宏，注意将这个宏的值修改成 NONE，不能修改成 no 和其他