iTOP-4412-驱动-usb 文档 11-usbWIFI 驱动移植

USB 的 WIFI 移植，主要分为以下几个步骤：

1 找到 WIFI 的驱动源码；

2 找到产品识别码（Product ID）和供应商 ID（Vendor ID）将其添加到 USB 的 WIFI 源码中；

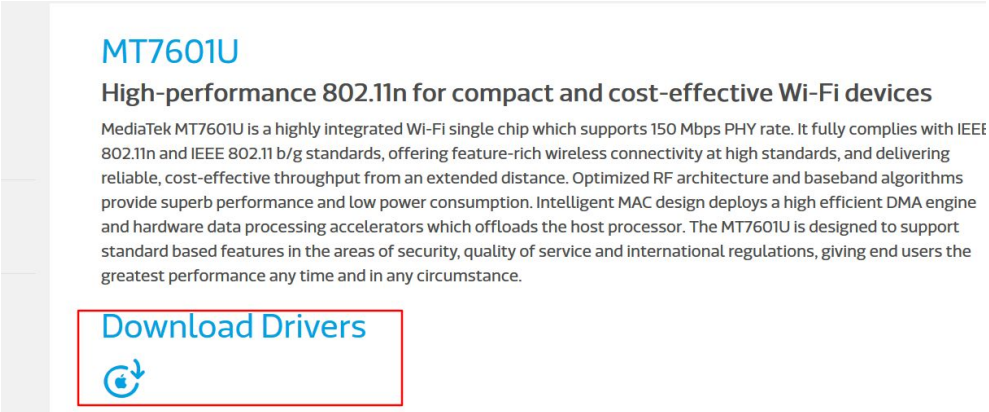
3 设置编译器，内核路径，将源码编译通过；

4 移植开源 wpa 工具。

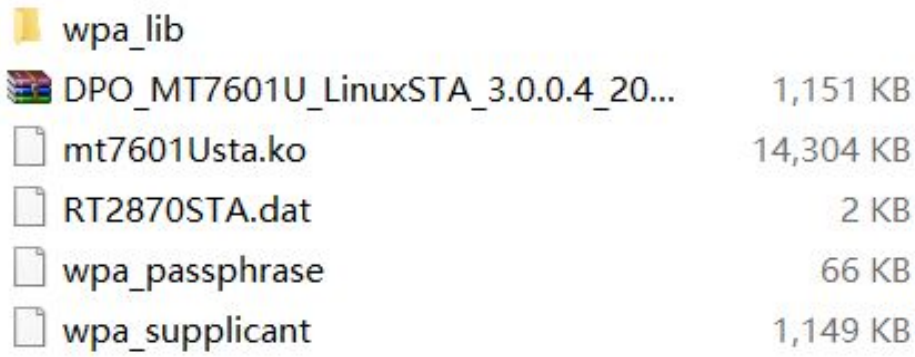
一般情况下，USB 的 WIFI 驱动源码除了 ID 部分，其它都不需要修改，wpa 工具有可能因为版本问题需要尝试一到两个版本。移植的 USBwifi 是 360 二代 wifi，请大家在购买硬件的时候要注意硬件版本。

1 驱动源码下载

360WiFi2 代使用的是 MT7601U 芯片，要移植首先要找到厂商提供的驱动源码，这里我们去 www.mediatek.com 官网搜索 MT7601，可以得到相关的驱动源码，如下图所示。

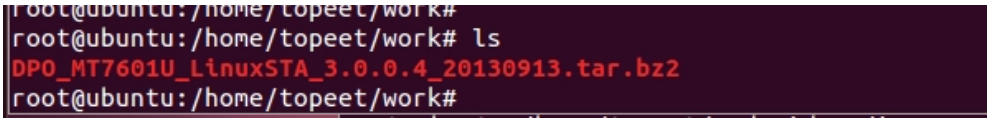


下载得到“DPO_MT7601U_LinuxSTA_3.0.0.4_20130913.tar.gz”，该驱动在压缩包中也有提供，如下图所示。

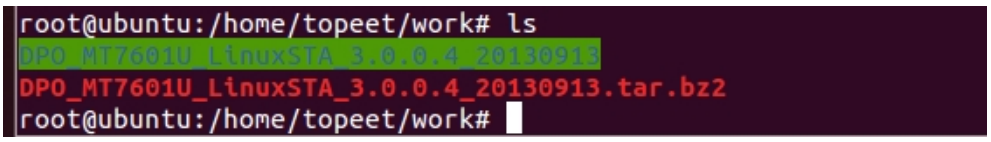


如上图所示，在 wpa_lib 文件夹是两个 wpa 工具的支持库，所有文件的用法在本文档后面部分都有详细描述。

将驱动源码拷贝到工作目录，如下图。



解压完成得到驱动的代码，如下图所示。



2 驱动的 ID 修改

为了让驱动识别 USB 设备，需要得到 360WiFi 的 PID 和 VID，即产品识别码（Product ID）和供应商 ID（Vendor ID）。先将 360WiFi 插到 Ubuntu 上，运行命令“lsusb”来得到，或者查看厂商的设备手册也可以得到，运行 lsusb 如下图所示。

可以看到 ID 一栏后面得到 VID 为 148f，PID 为 760b。

进入驱动源码的目录“/DPO_MT7601U_LinuxSTA_3.0.0.4_20130913”，打开文件“common/rtusb_dev_id.c”，在第 42 行下面加上 360WiFi 的 ID 号 {USB_DEVICE(0x148f,0x760b)}, /* 360 Wifi */ 。

```
/* module table */
USB_DEVICE_ID rtusb_dev_id[] = {
#ifdef RT6570
        {USB_DEVICE(0x148f,0x6570)}, /* Ralink 6570 */
#endif /* RT6570 */
        {USB_DEVICE(0x148f, 0x7650)}, /* MT7650 */
#ifdef MT7601U
        {USB_DEVICE(0x148f,0x760b)},/*360wifi*/
        {USB_DEVICE(0x148f,0x6370)}, /* Ralink 6370 */
        {USB_DEVICE(0x148f,0x7601)}, /* MT 6370 */
#endif /* MT7601U */
        { }/* Terminating entry */
};
```

3 驱动移植和编译

驱动源码目录“/DPO_MT7601U_LinuxSTA_3.0.0.4_20130913”下的 Makefile 编译文件需要修改。其实，在任何一个大点的驱动或者应用，都会提供 Makefile 文件，一般都需要修改编译器、目标平台和目标内核等部分。

打开 Makefile 文件，在第 30 行，可以看到默认平台为 PC。

```
#PLATFORM: Target platform
PLATFORM = PC
#PLATFORM = 5VT
#PLATFORM = IKANOS_V160
#PLATFORM = IKANOS_V180
```

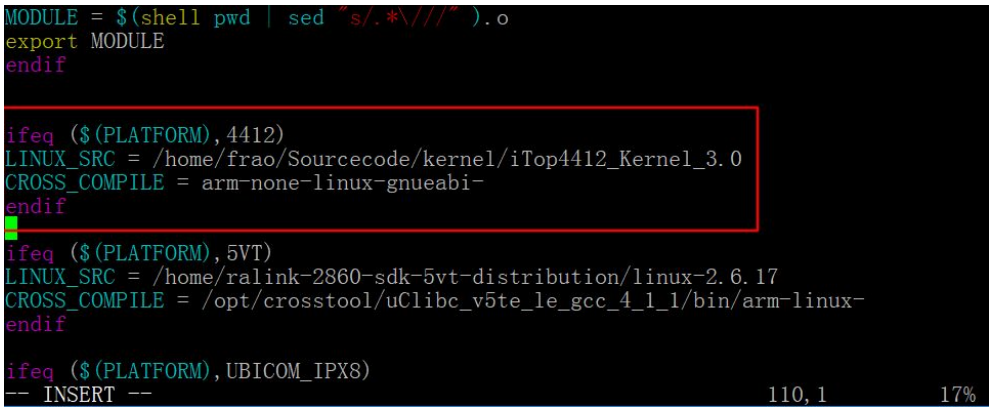
将 PC 注释掉，然后在这里定义一个自己的平台 4412，如下图所示。

```
#PLATFORM: Target platform
#PLATFORM = PC
PLATFORM = 4412
#PLATFORM = 5VT
#PLATFORM = IKANOS_V160
#PLATFORM = IKANOS_V180
#PLATFORM = SIGMA
```

然后针对自定义平台，在第 105 行开始，添加交叉编译链和内核路径。红色部分要和自己 Ubuntu 中的内核源码目录对应，而且内核源码一定要先编译通过，这一点在前面都强调过的。

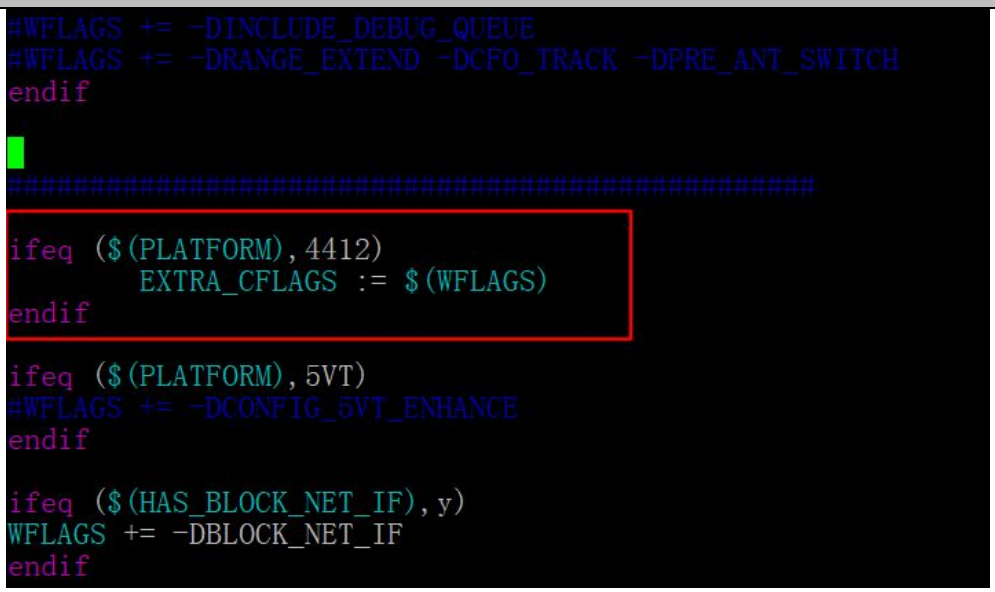
```
ifeq ($(PLATFORM),4412)
LINUX_SRC = /home/frao/Sourcecode/kernel/iTop4412_Kernel_3.0
CROSS_COMPILE = arm-none-linux-gnueabi-
endif
```

这里的 LINUX_SRC 为要运行平台的内核源码，此时为 4412 的内核源码位置。CROSS_COMPILE 为要使用的交叉编译器，可以是绝对路径或者环境变量。添加完成如下图所示，Makefile 文件就修改完成。



接着修改配置文件“os/linux/config.mk”，从 854 行起，添加以下内容。

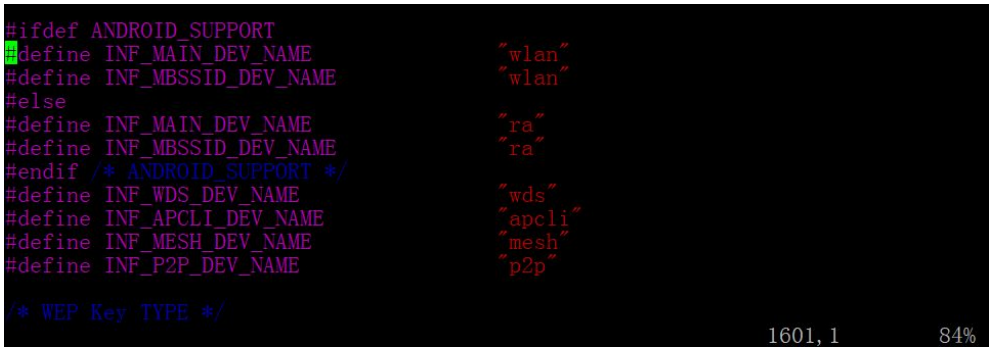
```
ifeq ($(PLATFORM),4412)
        EXTRA_CFLAGS := $(WFLAGS)
endif
```



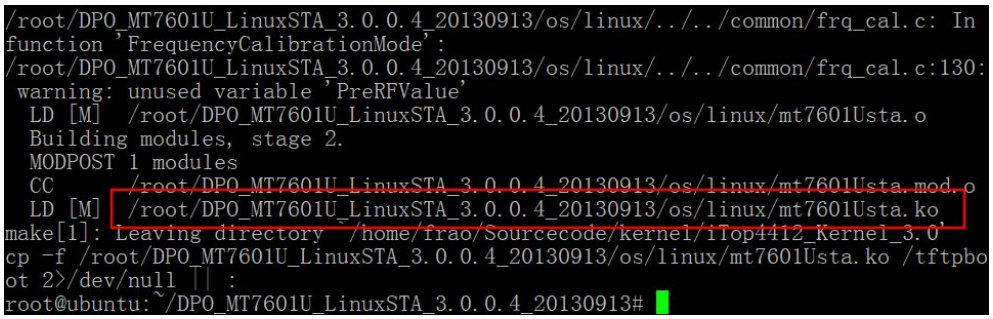
如果在 android 平台使用，则应确保该文件中下面两项为 y。

在“include/rtmp_def.h”的第 1601 行，该预编译定义了分别在 Android 和嵌入式 Linux 平台上的设备名称，即在 Android 平台显示 wlan 设备名，在 Linux 显示 ra 设备名，

如下图所示，由于一般在 Android 下的 HAL 和脚本中的 wifi 设备结点名称用的 wlan0，嵌入 Linux 设备中 wifi 设备结点名称为 ra0，所以这里我们保持不变。

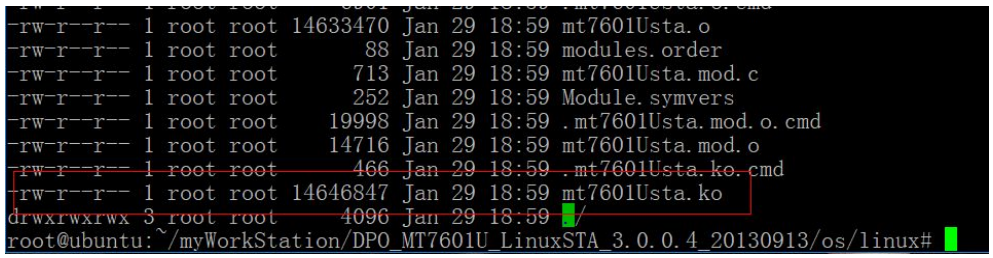


在 DPO_MT7601U_LinuxSTA_3.0.0.4_20130913 目录下执行“make”命令。编译成功后如下图所示。



按照上图中提示，编译生成驱动模块 mt7601Usta.ko，后面可以动态加载到内核中。

源码目录下的 RT2870STA.dat 需要拷贝到开发板的“/etc/Wireless/RT2870STA/”下，这个目录默认没有，需要手动创建。



至此，驱动编译完成。

4 wpa 的移植

wpa 是一款开源工具，移植这个工具需要做以下工作：

1 编译器环境设置；

2 OpenSSL 库的移植；

3 libnl 库的移植；

4 移植 wpa_supplicant 工具。

需要用到的库和工具的源码在 wpa_supplicant.zip 压缩包中，解压之后得到 hostap.tar.gz、libnl-1.1.4.tar.gz 和 openssl-1.1.0g.tar.gz 三个源码。

4.1 编译器设置

在进行编译之前要先修改编译器为 4.3.2 版本，如何设置编译器参见手册 7.1 章节“Qt/E4.7.1 编译器的安装”。另外为了避免使用环境变量设置编译器而可能出现的问题，文档中大部分编译是使用编译器的绝对路径，用户也应先找到自己编译器的绝对路径待用。下图是本次编译使用的编译器以及编译器压缩包。

```
root@ubuntu:/usr/local/arm# l
4.3.2/
arm-2009q3/
arm-2009q3.tar.bz2
arm-2014.05/
arm-2014.05-29-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
arm-linux-gcc-4.3.2.tar.gz*
root@ubuntu:/usr/local/arm#
```

由上图可知该编译器的绝对路径为“/usr/local/arm/4.3.2/bin/ arm-none-linux-gnueabi-gcc”。

用户需要将提供的源码压缩包拷贝到 Ubuntu 的工作目录，分别解压 ，如下图所示。

```
root@ubuntu:/home/frao/fraomt6620# l
hostap/        libnl-1.1.4/        openssl-1.1.0g/
hostap.tar.gz  libnl-1.1.4.tar.gz  openssl-1.1.0g.tar.gz
root@ubuntu:/home/frao/fraomt6620#
```

4.2 移植 OpenSSL

解压 OpenSSL 压缩包“openssl-1.1.0g.tar.gz”。

进入目录 openssl-1.1.0g，内容如下图所示。

```
root@ubuntu:/home/frao/fraomt6620# cd openssl-1.1.0g/
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g# l
ACKNOWLEDGEMENTS  crypto/     libcrypto.a        NEWS           README.ENGINE
apps/             crypto.map  libcrypto.pc       NOTES.DJGPP    README.FIPS
appveyor.yml      demos/      libcrypto.so@      NOTES.PERL     ssl/
AUTHORS           doc/        libcrypto.so.1.1*  NOTES.UNIX     ssl.map
build.info        engines/    libssl.a           NOTES.VMS      test/
CHANGES           e_os.h      libssl.pc          NOTES.WIN      tools/
config*           external/   libssl.so@         openssl.pc     util/
config.com        FAQ         libssl.so.1.1*     os-dep/        VMS/
configdata.pm     fuzz/       LICENSE            pod2htmd.tmp
Configurations/   include/    Makefile           pod2htmi.tmp
Configure*        __install/  Makefile.shared    README
CONTRIBUTING      INSTALL     ms/                README.ECC
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g#
```

执行下面指令，做相应配置。

```
./config no-asm shared --prefix=$(pwd)/__install
```

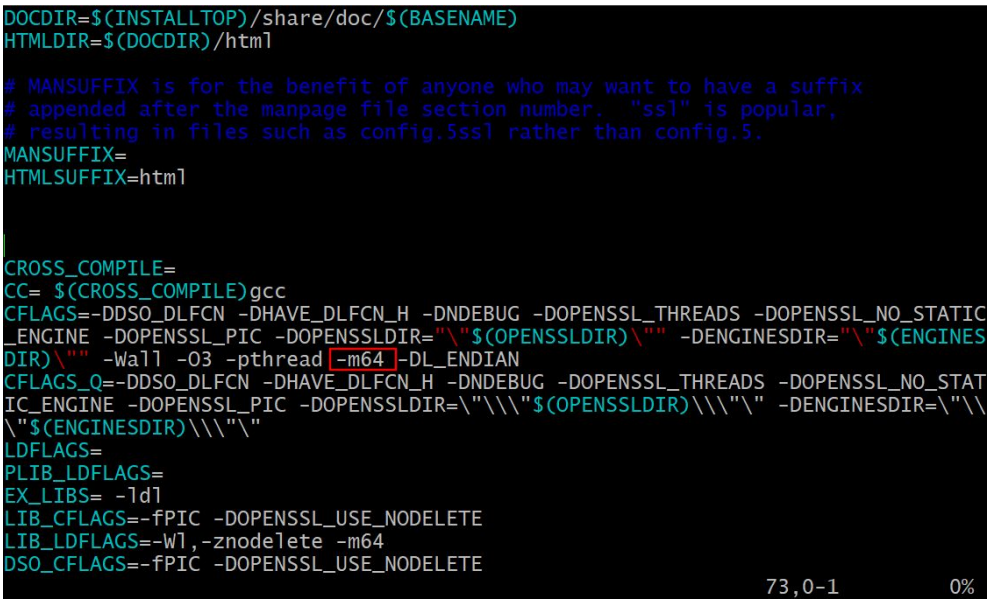
执行完成后如下图所示。

no-asm： 是在交叉编译过程中不使用汇编代码代码加速编译过程，原因是它的汇编代码是不支持 arm 格式。

shared ：生成动态连接库。

--prefix ：指定 make install 后生成目录的路径，不修改此项则默认为 OPENSSLDIR 目录(/usr/local/ssl)。

使用命令“vim Makefile”打开 Makefile，搜索 CFLAG，定位到下图中所示位置。



删除上图中红框中的“-m64”，完成后 CFLAG 应如下图所示。

```
# MANSUFFIX is for the benefit of anyone who may want to have a suffix
# appended after the manpage file section number.  "ssl" is popular,
# resulting in files such as config.5ssl rather than config.5.
MANSUFFIX=
HTMLSUFFIX=html



CROSS_COMPILE=
CC= $(CROSS_COMPILE)gcc
CFLAGS=-DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STATIC
_ENGINE -DOPENSSL_PIC -DOPENSSLDIR="\"$(OPENSSLDIR)\"" -DENGINESDIR="\"$(ENGINES
DIR)\"" -Wall -O3 -pthread -DL_ENDIAN
CFLAGS_Q=-DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STAT
IC_ENGINE -DOPENSSL_PIC -DOPENSSLDIR=\"\\\"$(OPENSSLDIR)\\\"\" -DENGINESDIR=\"\\
\"$(ENGINESDIR)\\\"\"
LDFLAGS=
PLIB_LDFLAGS=
EX_LIBS= -ldl
LIB_CFLAGS=-fPIC -DOPENSSL_USE_NODELETE
LIB_LDFLAGS=-Wl,-znodelete -m64
DSO_CFLAGS=-fPIC -DOPENSSL_USE_NODELETE
DSO_LDFLAGS=$(LIB_LDFLAGS)
BIN_CFLAGS=
                                                              76,198        0%
```

执行以下命令，编译 OpenSSL 库，注意这里使用的是交叉编译器的绝对路径。

```
make CROSS_COMPILE=/usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-
```

编译完成后如下图所示。

```
make[2]: Entering directory `/home/frao/fraomt6620/openssl-1.1.0g'
LD_LIBRARY_PATH=.: /usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-gcc -DDSO_DLF
CN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STATIC_ENGINE -DOPENSS
L_PIC -DOPENSSLDIR="/home/frao/fraomt6620/openssl-1.1.0g/__install/ssl" -DENGINE
SDIR="/home/frao/fraomt6620/openssl-1.1.0g/__install/lib/engines-1.1" -Wall -O3
-pthread -DL_ENDIAN -o test/x509aux test/x509aux.o -L. -lcrypto -ldl
make[2]: Leaving directory `/home/frao/fraomt6620/openssl-1.1.0g'
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
            "-oMakefile" apps/CA.pl.in > "apps/CA.pl"
chmod a+x apps/CA.pl
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
            "-oMakefile" apps/tsget.in > "apps/tsget"
chmod a+x apps/tsget
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
            "-oMakefile" tools/c_rehash.in > "tools/c_rehash"
chmod a+x tools/c_rehash
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
            "-oMakefile" util/shlib_wrap.sh.in > "util/shlib_wrap.sh"
chmod a+x util/shlib_wrap.sh
make[1]: Leaving directory `/home/frao/fraomt6620/openssl-1.1.0g'
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g#
```

执行下面命令，将编译好的库文件拷贝到第一步指定的目录。

```
make install
```

如下图所示在当前目录下的__install 目录下生成了头文件和库文件：

```
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g# ls
ACKNOWLEDGEMENTS  crypto       libcrypto.a       NEWS          README.ENGINE
apps              crypto.map   libcrypto.pc      NOTES.DJGPP   README.FIPS
appveyor.yml      demos        libcrypto.so      NOTES.PERL    ssl
AUTHORS           doc          libcrypto.so.1.1  NOTES.UNIX    ssl.map
build.info        engines      libssl.a          NOTES.VMS     test
CHANGES           e_os.h       libssl.pc         NOTES.WIN     tools
config            external     libssl.so         openssl.pc    util
config.com        FAQ          libssl.so.1.1     os-dep        VMS
configdata.pm     fuzz         LICENSE           pod2htmd.tmp
Configurations    include      Makefile          pod2htmi.tmp
Configure         __install    Makefile.shared   README
CONTRIBUTING      INSTALL      ms                README.ECC
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g# cd __install/
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g/__install# l
bin/  include/  lib/  share/  ssl/
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g/__install#
```

在编译 wpa 工具的时候需要用到该“__install/include”下的头文件和“__install/lib”下的库，在编译 wpa 的时候，会提醒大家设置这个路径。

而在 wpa 工具运行时需要用到“__install/lib”下的库，所以也要将 lib 下所有文件拷贝到开发板/lib 文件夹中。

4.3 移植 libnl

解压 libnl 压缩包“libnl-1.1.4.tar.gz”。

libnl 是为了方便应用程序使用 netlink 接口而开发的一个库。这个库为原始 netlink 消息传递以及不同的 netlink family 专用接口提供了一个统一的接口。

进入目录“libnl-1.1.4/”，如下图所示。

```
root@ubuntu:/home/frao/mt6620# cd libnl-1.1.4/
root@ubuntu:/home/frao/mt6620/libnl-1.1.4# l
aclocal.m4  configure.in  include/     libnl-1.pc     Makefile.opts.in  src/
ChangeLog   COPYING       install-sh*  libnl-1.pc.in  Makefile.rules    tests/
configure*  doc/          lib/         Makefile       README
root@ubuntu:/home/frao/mt6620/libnl-1.1.4#
```

执行下面的指令，配置编译架构。

```
./configure --prefix=$(pwd)/__install --enable-shared --enable-static
```

其中--prefix=$(pwd)/__install 指定了编译出来的库存放的路径，一般将其放在当前目录下的__install 目录下，执行结果如下图所示。

```
checking for pthread_mutex_lock in -lpthread... yes
configure: creating ./config.status
config.status: creating Makefile.opts
config.status: WARNING:  'Makefile.opts.in' seems to ignore the --datarootdir se
tting
config.status: creating libnl-1.pc
config.status: creating doc/Doxyfile
config.status: creating lib/defs.h
config.status: lib/defs.h is unchanged
configure: WARNING: unrecognized options: --enable-shared, --enable-static

------------------------------------------------------------------------
SUMMARY:

Included in Compilation:
  libnl:   Yes  -lm -lpthread

Dependencies:
  libm             Yes       (required)
root@ubuntu:/home/frao/fraomt6620/libnl-1.1.4#
```

执行下面的命令，编译库。

```
make CC=/usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-gcc
```

完成后如下图所示。

```
  LD nl-qdisc-dump
  LD nl-route-add
  LD nl-route-del
  LD nl-route-dump
  LD nl-route-get
  LD nl-rule-dump
  LD nl-tctree-dump
  LD nl-util-addr
  LD genl-ctrl-dump
  LD genl-ctrl-get
  LD nf-ct-dump
  LD nf-log
  LD nf-monitor
Entering tests
  LD test-cache-mngr
  LD test-genl
  LD test-nf-cache-mngr
  LD test-socket-creation
root@ubuntu:/home/frao/fraomt6620/libnl-1.1.4#
```

使用命令“make install”，将编译好的库文件拷贝到指定目录“__install”下。在当前目录下的__install 目录下生成了头文件和库文件 ，如下图所示。

```
nfig/
root@ubuntu:/home/frao/fraomt6620/libnl-1.1.4# ls
aclocal.m4      configure      include      libnl-1.pc      Makefile.opts.in  tests
ChangeLog       configure.in   __install    libnl-1.pc.in   Makefile.rules
config.log      COPYING        install-sh   Makefile        README
config.status   doc            lib          Makefile.opts   src
root@ubuntu:/home/frao/fraomt6620/libnl-1.1.4# cd __install/
root@ubuntu:/home/frao/fraomt6620/libnl-1.1.4/__install# ls
include  lib
root@ubuntu:/home/frao/fraomt6620/libnl-1.1.4/__install#
```

在编译 wpa 工具的时候需要用到“__install/include”下的头文件和“__install/lib”下的库，我们在编译 wpa 的时候，会提醒大家设置这个路径。

而在 wpa 程序运行时需要用到“__install/lib”下的库，所以同样要将 lib 下所有文件拷贝到开发板/lib 文件夹中。

4.4 移植 wpa_supplicant

wpa_supplicant 是作为 hostap 的一部分，它的源码在 hostap 目录中。

解压 wpa 压缩包“hostap.tar.gz”。

使用命令“cd hostap/wpa_supplicant/”进入 wpa_supplicant 目录，如下图所示。

```
root@ubuntu:/home/frao/fraomt6620# l
hostap/         libnl-1.1.4/         openssl-1.1.0g/
hostap.tar.gz   libnl-1.1.4.tar.gz   openssl-1.1.0g.tar.gz
root@ubuntu:/home/frao/fraomt6620# cd hostap/wpa_supplicant/
root@ubuntu:/home/frao/fraomt6620/hostap/wpa_supplicant#
```

使用命令“cp defconfig .config”复制一份默认的配置文件。然后使用命令“vim Makefile”修改 Makefile，如下图所示。

```
root@ubuntu:/home/frao/fraomt6620/hostap/wpa_supplicant# cp defconfig .config
root@ubuntu:/home/frao/fraomt6620/hostap/wpa_supplicant# vim Makefile
```

将

```
ifndef CC
CC=gcc
endif
```

修改为如下所示，注意下面的黑体部分，这里是设置库 libnl 和 openssl 的头文件和库文件，用户一定要根据实际路径来设置，否则将会无法编译。

```
CFLAGS += -I../../libnl-1.1.4/__install/include/
CFLAGS += -I../../openssl-1.1.0g/__install/include/

LIBS += -L../../libnl-1.1.4/__install/lib/
LIBS += -L../../openssl-1.1.0g/__install/lib/

#ifndef CC
CC=/usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-gcc
#endif
```

注意，CC 路径为自己环境中的交叉工具链路径。

修改完成后 Makefile 如下图所示。

```
CFLAGS += -I../../libnl-1.1.4/__install/include/
CFLAGS += -I../../openssl-1.1.0g/__install/include/

LIBS += -L../../libnl-1.1.4/__install/lib/
LIBS += -L../../openssl-1.1.0g/__install/lib/

#ifndef CC
CC=/usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-gcc
#endif

ifndef CFLAGS
CFLAGS = -MMD -O2 -Wall -g
endif

ifdef LIBS
# If LIBS is set with some global build system defaults, clone those for
# LIBS_c and LIBS_p to cover wpa_passphrase and wpa_cli as well.
ifndef LIBS_c
```

接下来使用命令“make”编译，结果如下图所示。

```
  CC  ../src/drivers/drivers.c
  CC  ../src/l2_packet/l2_packet_linux.c
  LD  wpa_supplicant
  CC  wpa_cli.c
  CC  ../src/common/wpa_ctrl.c
  CC  ../src/common/cli.c
  CC  ../src/utils/edit_simple.c
  LD  wpa_cli
  CC  wpa_passphrase.c
  LD  wpa_passphrase
  sed systemd/wpa_supplicant.service.in
  sed systemd/wpa_supplicant.service.arg.in
  sed systemd/wpa_supplicant-nl80211.service.arg.in
  sed systemd/wpa_supplicant-wired.service.arg.in
  sed dbus/fi.epitest.hostap.WPASupplicant.service.in
  sed dbus/fi.w1.wpa_supplicant1.service.in
root@ubuntu:/home/frao/fraomt6620/hostap/wpa_supplicant#
```

此时在当前目录下出现了“wpa_supplicant”工具，如下图所示。

```
config.h                  mesh_mpm.c              wpa_gui-qt4
config_none.c             mesh_mpm.h              wpa_passphrase
config.o                  mesh_rsn.c              wpa_passphrase.c
config_ssid.h             mesh_rsn.h              wpa_passphrase.o
config_winreg.c           nfc_pw_token.c          wpa_priv.c
ctrl_iface.c              nmake.mak               wpas_glue.c
ctrl_iface.h              notify.c                wpas_glue.h
ctrl_iface_named_pipe.c   notify.h                wpas_glue.o
ctrl_iface.o              notify.o                wpas_kay.c
ctrl_iface_udp.c          offchannel.c            wpas_kay.h
ctrl_iface_unix.c         offchannel.h            wpas_module_tests.c
ctrl_iface_unix.o         op_classes.c            wpa_supplicant
dbus                      op_classes.o            wpa_supplicant.c
defconfig                 p2p_supplicant.c        wpa_supplicant.conf
doc                       p2p_supplicant.h        wpa_supplicant_conf.mk
dpp_supplicant.c          p2p_supplicant_sd.c     wpa_supplicant_conf.sh
dpp_supplicant.h          preauth_test.c          wpa_supplicant_i.h
driver_i.h                README                  wpa_supplicant.o
eapol_test.c              README-HS20             wpa_supplicant_template.conf
eapol_test.py             README-P2P              wps_supplicant.c
eap_proxy_dummy.mak       README-Windows.txt      wps_supplicant.h
eap_proxy_dummy.mk        README-WPS
eap_register.c            rrm.c
root@ubuntu:/home/frao/fraomt6620/hostap/wpa_supplicant#
```

将编译好的 wpa_supplicant 工具以及 wpa_supplicant 拷贝到开发板上的“/usr/sbin”目录下。

至此，wpa 部分完成。

5 测试

在测试前，需要拷贝的内容如下：

拷贝“RT2870STA.dat”到开发板中的“/etc/Wireless/RT2870STA/”目录下；作者编译好的文件，在压缩包中也有提供。

拷贝 libnl 和 openssl 编译出来的“__install/lib”目录下的库文件到开发板的/lib 目录下。作者提供编译好的两个库，都在压缩包中的 wpa_lib 目录下。

拷贝 wpa_supplicant 工具以及 wpa_passphrase 到开发板的“/usr/sbin”目录下；作者编译好的两个工具在压缩包中也有提供。

作者编译好的驱动文件“mt7601Usta.ko”在压缩包中也有提供。

将提供的“default.script”拷贝到开发板的“/usr/share/udhcpc”目录下，这个目录默认不存在，需要手动创建，拷贝完如图所示。注意：如果是 qt 系统则不需要拷贝，如果是根文件系统则需要拷贝这个文件。

```
[root@iTOP-4412]# ls /usr/share/udhcpc
default.script
```

接着我们可以按照文档中的步骤来操作测试。

将 360Wifi 插到开发板的 USB 接口上，可以在端口上看到如下打印信息。

```
[root@iTOP-4412]# [  226.388761] usb 1-3.1: USB disconnect, device number 3
[  228.125100] usb 1-3.1: new high speed USB device number 6 using s5p-ehci
[  228.246232] usb 1-3.1: New USB device found, idVendor=148f, idProduct=760b, b
cdDevice=0000
[  228.253032] usb 1-3.1: New USB device strings: Mfr=1, Product=2, SerialNumber
=3
[  228.260339] usb 1-3.1: New USB device Class: Class=0, SubClass=0, Protocol=0
[  228.267354] usb 1-3.1: Product: 802.11 n WLAN
[  228.271691] usb 1-3.1: Manufacturer: MediaTek
[  228.276030] usb 1-3.1: SerialNumber: 1.0
```

拷贝驱动程序到开发，加载驱动程序“mt7601Usta.ko”，如下图。

```
[root@iTOP-4412]#
[root@iTOP-4412]# insmod /mnt/3604412/mt7601Usta.ko
[  282.528546] rtusb init rt2870 --->
[  282.545672] ===>rt2870_probe()!
[  282.547354] --> RTMPAllocAdapterBlock
[  282.552110]
[  282.552120]
[  282.552128] === pAd = f0cfa000, size = 843016 ===
[  282.552139]
[  282.580053] --> RTMPAllocTxRxRingMemory
```

配置 ra0 网络，输入“ifconfig ra0 up”，如下图所示，因为作者前面没有将 ra 修改为 wlan，所以这里使用命令“ifconfig ra0 up”，如果修改了，则需要使用“ifconfig wlan0 up”命令。

```
~ #
~ # ifconfig ra0 up
[  949.539066] WlanFunCtrl.word = 0xff200003
[  949.542483] MACVersion = 0x76010500
[  949.545710] Allocate 8192 memory for BA reordering
[  949.550609] MAC[Ver:Rev=0x76010500]
[  949.553363] USBLoadFirmwareToAndes
[  949.563767] FW Version:0.1.00 Build:7640
[  949.566249] Build Time:201302052146____
[  949.570110] ILM Length = 45380(bytes)
[  949.573703] DLM Length = 0(bytes)
[  949.584288] Loading FW....
[  949.655430] #
[  949.660925] USBLoadFirmwareToAndes: COM_REG0(0x730) = 0x1
[  949.664914] --> NICInitRecv
[  949.667976] <-- NICInitRecv()
[  949.670699] --> NICInitTransmit
[  949.673867] MGMT Ring: total 32 entry allocated
[  949.678314] <-- NICInitTransmit(Status=0)
```

使用命令“wpa_passphrase SSID 密码 > /etc/ wpa_supplicant.conf”配置 WiFi，如下图所示。

```
~ # wpa_passphrase xunwei2701 topeet2015 > /etc/wpa_supplicant.conf
~ #
```

执行命令“wpa_supplicant -B -i ra0 -D wext -c /etc/wpa_supplicant.conf”。

```
~ #
~ # wpa_supplicant -B -i ra0 -D wext -c /etc/wpa_supplicant.conf
[ 1033.467732] ===> rt_ioctl_siwpmksa
[ 1033.469698] rt_ioctl_siwpmksa - IW_PMKSA_FLUSH
[ 1033.474243] ===>Set_NetworkType_Proc::(INFRA)
[ 1033.478527] Set_NetworkType_Proc::(NetworkType=1)
[ 1033.483506] ===>rt_ioctl_giwrange
[ 1033.495179] rt_ioctl_siwauth::IW_AUTH_WPA_ENABLED - Driver supports WPA!(para
m->value = 1)
[ 1033.502333] RtmpIoctl_rt_ioctl_siwauth::IW_AUTH_WPA_ENABLED - Driver supports
 WPA!(param->value = 1)
[ 1033.511908] AsicUpdateWcidAttributeEntry : WCID #1, KeyIndex #0, Alg=none
[ 1033.517965]          WCIDAttri = 0x1
[ 1033.521322] AsicRemovePairwiseKeyEntry : Wcid #1
```

执行命令“udhcpc -i ra0”获取动态 IP 以及网关 DNS。

```
~ #
~ # udhcpc -i ra0
udhcpc (v1.21.1) started
Sending discover...
[   72.475272] RTMP_TimerListAdd: add timer obj f0e3607c!
[   72.478975] RTMPInitTimer: f0e3607c
[   72.490147] BA Ori Session Timeout(1) : Send ADD BA again
[   72.494455] BA - Send ADDBA request. StartSeq = 3,  FrameLen = 33. BufSize =
64
[   72.501795] PeerAddBARspAction ==> Wcid(1)
[   72.505546]           StatusCode = 0
[   72.508439] ba>WinSize=63, MaxSize=21, MaxPeerRxSize=39
[   72.513767] ba> reassign max win size from 63 to 21
[   72.518745] BAOriSessionAdd():TXBAbitmap=1, BAWinSize=21, TimeOut=0
Sending select for 192.168.3.87...
Lease of 192.168.3.87 obtained, lease time 604800
deleting routers
route: SIOCDELRT: No such process
adding dns 192.168.3.1
~ #
```

到此，就可以使用 360WiFi 上网了，测试完毕，如下图所示。

```
/mnt # ping www.baidu.com &
/mnt # PING www.baidu.com (180.149.132.151): 56 data bytes
64 bytes from 180.149.132.151: seq=0 ttl=54 time=30.697 ms
64 bytes from 180.149.132.151: seq=1 ttl=54 time=17.067 ms
```

小结

大家看了 USB WIFI 驱动的移植，可能会感到很困惑，为什么前面的 USB 学习文档介绍了那么多，移植驱动的时候反而就是这么简单的几个步骤。实际上，USB 驱动框架，是非常复杂的，但是核心的内部驱动部分，甚至外部驱动都是不用驱动工程师写的，我们给大家示范的叫移植，希望通过本文档能够感受到什么叫“移植”。

大家可能还有疑惑，这些驱动是谁写的？首先是 USB 内部驱动，它们是 Linus 同学带着一帮全世界最聪明最厉害的程序员，免费做的 linux 核心部分代码，并且免费开源给大家用；另外一部分是外部驱动，例如这个 360WIFI 的芯片，它的驱动当然是 MTK（芯片厂商）的工程师来做的，这帮工程师是由芯片厂商供养，他们也是在前人的基础上一步一步的做的，也不会一触而就的从零开始做一个驱动，而且他们一般是一个团队，专门做 USB 部分的代码。假如大家将来有机会进入芯片厂商工作，只需要有良好的基本功就没什么问题的。

大家可能还有疑惑，如果想支持一款 USB 的 WIFI，找不到驱动源码怎么办？当然是重新选型，换个方案。原厂不开源，那就是不想让我们用，那我们还舔着脸一定要用它的方案？在选型的时候，就一定要找到对应的驱动，否则一颗芯片都不要买。任何一个产品设计中，都不可能，也没必要让我们去从零开始写 USB 驱动的！切记!!!先找驱动代码！

大家可能还有疑惑，怎么连 datasheet 都不提供，寄存器怎么配置呢？其实像这样大的驱动，寄存器配置都会提供一个脚本或者一个二进制文件（叫芯片固件应该更合适），有的会提供专门的小程序将配置文件在驱动加载前就烧写到芯片中，类似 MT6620。如果要修改一些参数，直接通过上层工具就可以实现。

如果大家还有疑惑，可以先看下一篇关于 USB 3G 的移植，3G 的移植，我们甚至连内核驱动都不用找了，内核自带，只需要移植个工具就成。

联系方式

北京迅为电子有限公司致力于嵌入式软硬件设计，是高端开发平台以及移动设备方案提供商；基于多年的技术积累，在工控、仪表、教育、医疗、车载等领域通过 OEM/ODM 方式为客户创造价值。

iTOP-4412 开发板是迅为电子基于三星最新四核处理器 Exynos4412 研制的一款实验开发平台，可以通过该产品评估 Exynos 4412 处理器相关性能，并以此为基础开发出用户需要的特定产品。

本手册主要介绍 iTOP-4412 开发板的使用方法，旨在帮助用户快速掌握该产品的应用特点，通过对开发板进行后续软硬件开发，衍生出符合特定需求的应用系统。

如需平板电脑案支持，请访问迅为平板方案网“http://www.topeet.com”，我司将有能力为您提供全方位的技术服务，保证您产品设计无忧！

本手册将持续更新，并通过多种方式发布给新老用户，希望迅为电子的努力能给您的学习和开发带来帮助。

迅为电子

2018 年 2 月