






## iTOP-6818- QtE-WiFi\_mt6620 的移植

本文档介绍 SDIO WIFI 在 6818 开发板，QtE 的移植方法，请注意：移植过程中的部分库和工具，在源码或者镜像中可能已经存在，大家可以使用自己生成的覆盖即可。提供的文件如下图所示，分别为三个源码压缩包、编译好的库位于 lib 文件夹、wpa\_supplicant 工具位于 sbin 文件夹、6620\_launcher 和脚本 make\_mt6620.sh。

 wpa_drivers	文件夹
 wpa_lib	文件夹
 6620_launcher	文件
 make_mt6620.sh	SH 文件
 wpa_supplicant	文件

这里我们所使用的驱动模块对应应在 Android 源码中的

```
device/nexell/s5p6818_drone/mt6620_6818/
```

目录下，用户只要在内核将 WiFi 设置为模块方式(M)，然后编译 Android 源码之后可以在该目录找到可以使用的驱动模块。同样所需驱动我们在 wpa\_driver 文件夹直接提供，以方便使用。

### 1 6620\_launcher

6620\_launcher 工具是作为后台的一个服务程序运行，该服务会配置串口的工作参数，下载固件补丁到 MT6620 中，它位于开发板的 “/usr/bin/” 目录下，在系统中内置，用户可以直接覆盖。

### 2 移植 wpa\_supplicant

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# 1
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# 1
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#

```

## 2.1 移植 OpenSSL

首先进入目录 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# 1
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 pod2htmli.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。

```

SHA1_OBJ_ASM =
RMD160_OBJ_ASM=
CMLL_ENC      =camellia.o cml1_misc.o cml1_cbc.o
MODES_OBJ     =
PADLOCK_OBJ   =
CHACHA_ENC    =chacha_enc.o
POLY1305_OBJ  =
BLAKE2_OBJ    =
PROCESSOR     =
RANLIB        =ranlib
ARFLAGS       =
PERL          =/usr/bin/perl

SIXTY_FOUR_BIT_LONG mode

Configured for linux-x86_64.
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g#

```

no-asm：是在交叉编译过程中不使用汇编代码加速编译过程，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，定位到下图所示位置。

```
DOCDIR=$(INSTALLTOP)/share/doc/$(BASENAME)
HTMLDIR=$(DOCDIR)/html

# 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)\" -DENGESDIR="\"$(ENGES
DIR)\" -Wall -O3 -pthread -m64 -DL_ENDIAN
CFLAGS_Q=-DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STAT
IC_ENGINE -DOPENSSL_PIC -DOPENSSLDIR="\"\\\"$(OPENSSLDIR)\\\"\\\" -DENGESDIR="\"\\
\"$(ENGESDIR)\\\"\\\"
LDFLAGS=
PLIB_LDFLAGS=
EX_LIBS= -ldl
LIB_CFLAGS=-fPIC -DOPENSSL_USE_NODELETE
LIB_LDFLAGS=-Wl,-znodelete -m64
DSO_CFLAGS=-fPIC -DOPENSSL_USE_NODELETE
```

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)\" -DENGESDIR="\"$(ENGES
DIR)\" -Wall -O3 -pthread -DL_ENDIAN
CFLAGS_Q=-DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STAT
IC_ENGINE -DOPENSSL_PIC -DOPENSSLDIR="\"\\\"$(OPENSSLDIR)\\\"\\\" -DENGESDIR="\"\\
\"$(ENGESDIR)\\\"\\\"
LDFLAGS=
PLIB_LDFLAGS=
EX_LIBS= -ldl
LIB_CFLAGS=-fPIC -DOPENSSL_USE_NODELETE
LIB_LDFLAGS=-Wl,-znodelete
DSO_CFLAGS=-fPIC -DOPENSSL_USE_NODELETE
DSO_LDFLAGS=$(LIB_LDFLAGS)
BIN_CFLAGS=
```

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_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STATIC_ENGINE -DOPENSSL_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         pod2htmli.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# ls
bin/  include/  lib/  share/  ssl/
root@ubuntu:/home/frao/fraomt6620/openssl-1.1.0g/___install#

```

include 下文件在编译程序的时候需要指定该 include 的路径。而 lib 下在程序运行时会用到，需要将 lib 下所有文件包括文件夹拷贝到开发板/lib 文件夹中。

## 2.2 移植 libnl

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# ls
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 setting
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 n1-qdisc-delete
LD n1-qdisc-dump
LD n1-route-add
LD n1-route-del
LD n1-route-dump
LD n1-route-get
LD n1-rule-dump
LD n1-tctree-dump
LD n1-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。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#
```

include 目录下文件在编译程序的时候会用到，而 lib 下在程序运行时会用到。故在移植 hostapd 的时候需要指定 include 的路径，需要将 lib 目录下所有文件包括文件夹拷贝到开发板中的/lib 文件夹中。

## 2.3 移植 wpa\_supplicant

wpa\_supplicant 是作为 hostap 的一部分，它位于 hostap 目录中。使用命令 “cd hostap/wpa\_supplicant/” 进入 wpa\_supplicant 目录，如下图所示。

```
root@ubuntu:/home/frao/fraomt6620# 1
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

```
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

ifndef 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/utlis/radiotap.c
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/utlis/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 工具拷贝到开发板上的 “/usr/sbin” 目录即可。

### 3 直接拷贝

跳过编译步骤，可以直接拷贝压缩包中 lib 文件夹中的库文件到开发板的 /lib 目录，以及复制 sbin 文件夹中的文件到开发板的 /usr/sbin 目录，复制 6620\_launcher 到开发板的 /usr/bin 目录下，复制 wpa\_driver 中的所有文件到 /lib/modules/mt6620/ 目录，以及复制脚本 make\_mt6620.sh 到开发板的任意目录，然后进行下一章节的操作。

## 4 开发板连接 WiFi

在开发板执行下面的指令，更新 WiFi 开机启动脚本。

```
cat << EOF > /etc/init.d/mt6620
#!/bin/sh
#support MT6620 WIFI Module
mknod /dev/stpwmt c 190 0
mknod /dev/stpgps c 191 0
mknod /dev/fm c 193 0
mknod /dev/wmtWifi c 194 0

insmod /lib/modules/mt6620/mtk_hif_sdio.ko
insmod /lib/modules/mt6620/mtk_stp_wmt.ko
insmod /lib/modules/mt6620/mtk_stp_uart.ko
insmod /lib/modules/mt6620/mtk_stp_gps.ko
#insmod /lib/modules/mt6620/hci_stp.ko
#insmod /lib/modules/mt6620/mt6620_fm_drv.ko
#insmod /lib/modules/mt6620/mtk_fm_priv.ko
insmod /lib/modules/mt6620/mtk_wmt_wifi.ko WIFI_major=194
insmod /lib/modules/mt6620/wlan_mt6620.ko

chmod 0666 /dev/stpwmt
chmod 0666 /dev/stpgps
chmod 0666 /dev/fm
chmod 0666 /dev/wmtWifi
chmod 0666 /dev/gps
chmod 0660 /dev/ttySAC2

/usr/bin/6620_launcher -m 1 -b 921600 -n /etc/firmware/mt6620_patch_hdr.bin -d /dev/ttySAC2 &
sleep 4
echo 1 > /dev/wmtWifi

wpa_supplicant -iwlan0 -Dnl80211 -c/etc/wpa_supplicant.conf &
sleep 3
udhcpc -i wlan0 >/var/udhcpc_log &
EOF
```

粘贴到开发板然后按回车，如下图所示。



```

> #insmod /lib/modules/mt6620/mtk_fm_priv.ko
> insmod /lib/modules/mt6620/mtk_wmt_wifi.ko WIFI_major=194
> insmod /lib/modules/mt6620/wlan_mt6620.ko
>
>
> chmod 0666 /dev/stpwm
> chmod 0666 /dev/stpgps
> chmod 0666 /dev/fm
> chmod 0666 /dev/wmtWifi
> chmod 0666 /dev/gps
> chmod 0660 /dev/ttySAC2
>
> /usr/bin/6620_launcher -m 1 -b 921600 -n /etc/firmware/mt6620_patch_hdr.bin
> d /dev/ttySAC2 &
> sleep 7
> echo 1 > /dev/wmtWifi
>
> wpa_supplicant -iwlan0 -Dnl80211 -c/etc/wpa_supplicant.conf &
> sleep 3
> udhcpc -i wlan0 >/var/udhcpc_log &
> EOF
~ # echo done
done
~ #
~ #

```

或者在开发板上执行我们提供的 “make\_mt6620.sh” ，起到同样效果。

然后执行以下指令

```
wpa_passphrase XXX "YYY" > /etc/wpa_supplicant.conf
```

其中 XXX 代表 WiF 网络名称, YYY 代表 WPA-PSK 或者 WPA2-PSK 加密的密码。然后执行命令 “ ./etc/init.d/mt6620 ” 即可连接到 WiFi 网络，配置时间大约 30 秒左右。

```

~ # wpa_passphrase xunwei "topeet15" >> /etc/wpa_supplicant.conf
~ # /etc/init.d/mt6620

saaFsmSteps: (SAA STATE) TRANSITION: [SAA_STATE_WAIT_ASSOC2] -> [AA_STATE_IDLE]
secFsmSteps: (RSN STATE)
94:77:2b:a1:b4:c0 TRANSITION: [SEC_STATE_INIT] -> [SEC_STATE_CHECK_OK]

ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
roamingFsmSteps: (ROAMING STATE) wlan0: Associated with 94:77:2b:a1:b4:c0 TRANSITION: [ROAMING_STATE_IDLE] -> [ROAMING_STATE_DECISION]
1:b4:c0
wlan0: CTRL-EVENT-SUBNETTaisFsmSteps: (AIS STATE) -STATUS-UPDATE status=0
TRANSITION: [AIS_STATE_JOIN] -> [AIS_STATE_NORMAL TR]
wlan0: WPA: Key negotiation completed with 94:77:2b:a1:b4:c0 [PTK=CCMP GTK=CCMP]
wlan0: CTRL-EVENT-CONNECTED - Connection to 94:77:2b:a1:b4:c0 completed [id=0 id_str=]
~ # route: SIOCDELRT: No such process

~ #
~ # [STP-PSM] [I]_stp_psm_stp_is_idle: **IDLE is over 60000 msec, go to sleep!!!**
ifconfig
lo          Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            inet6 addr: ::1/128  Scope:Host
            UP LOOPBACK RUNNING  MTU:16436  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0      Link encap:Ethernet  HWaddr 58:12:43:FA:4D:47
            inet addr:192.168.3.22  Bcast:192.168.3.255  Mask:255.255.255.0
            inet6 addr: fe80::5a12:43ff:fefa:4d47/64  Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1143 errors:0 dropped:0 overruns:0 frame:0
            TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:82996 (81.0 KiB)  TX bytes:1498 (1.4 KiB)

~ #

```



## 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-6818 开发板是迅为电子基于三星最新八核处理器 6818 研制的一款实验开发平台，可以通过该产品评估 6818 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-6818 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