

MT7601 USB 网卡驱动程序移植教程

宿主机: win7 64 位

虚拟机: VMware Workstation 14、Ubuntu-16.04.3

交叉编译工具链: arm-linux-gnueabi-

目标板: AM335x

内核版本: linux-3.14.26

mt7601 的驱动有两个类别, 分别是 STA 驱动和 AP 驱动, 通俗的说就是 wifi 客户端驱动和 wifi 热点驱动。STA 驱动中主要分为两个大版本 DPO 和 DPA, 两者具体的差异不太明白, 但是 DPA 版本的编译后生成三个 ko 文件, 文件较大; 而 DPO 版本的编译后只有一个 ko 文件, 文件大小较小。为了节省 flash 空间, AM3352 平台选用 DPO 版本的驱动程序。DPO 和 DPA 两个版本的驱动程序在 AM3352 平台均移植成功, 并且完美支持 wpa_supplicant 进行热点管理, 同时修复了多次拔插网卡后连接 wifi 失败的问题。

为了使读者对整个移植的过程有一个全面的把握, 下面将详细介绍 DPO 和 DPA 两个版本驱动程序的移植过程。

DPA 版本驱动程序移植过程详解

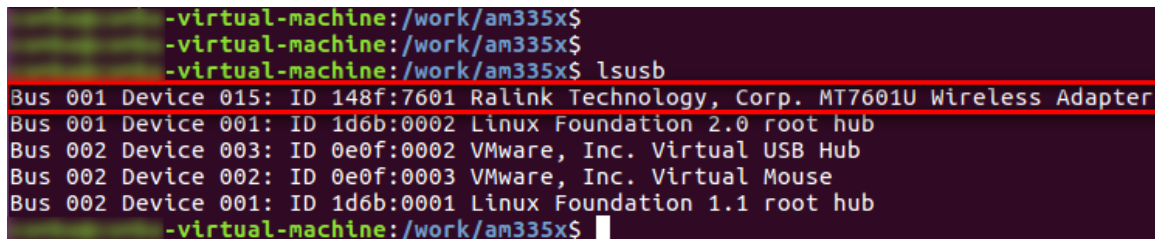
1 下载并解压源码

```
tar -xzf DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221.tar.gz
```

2 在驱动程序里面添加自己所使用的网卡

2.1 查看自己所使用网卡的型号

1) 将网卡插到电脑上, 通过 lsusb 命令查看型号, 如下图:



```
-virtual-machine:/work/am335x$  
-virtual-machine:/work/am335x$  
-virtual-machine:/work/am335x$ lsusb  
Bus 001 Device 015: ID 148f:7601 Ralink Technology, Corp. MT7601U Wireless Adapter  
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub  
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub  
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse  
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub  
-virtual-machine:/work/am335x$
```

图 1 查看网卡型号

如上图所示我们使用的 USB 网卡的 VID 和 PID 分别为 148f 和 7601

2) 修改 DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/NETIF/common/rtnetif_dev_id.c 文件, 添加网卡支持, 修改后的代码如下图所示;

```

34: /* module table */
35: USB_DEVICE_ID rtusb_dev_id[] = {
36: #ifdef RT6570
37:     {USB_DEVICE(0x148f, 0x6570)}, /* Ralink 6570 */
38: #endif /* RT6570 */
39:     {USB_DEVICE(0x148f, 0x7650)}, /* MT7650 */
40: #ifdef MT7601U
41:     {USB_DEVICE(0x148f, 0x6370)}, /* Ralink 6370 */
42:     {USB_DEVICE(0x148f, 0x7601)}, /* MT 6370 */
43: #endif /* MT7601U */
44:     { } /* Terminating entry */
45: };

```

图 2 添加网卡支持

3 设置网卡名称（可以不修改，根据个人习惯）

修改 DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/MODULE/include/rtmp_def.h 文件，设置网卡的名称，修改后的代码如下图所示：

```

1737: #ifdef ANDROID_SUPPORT
1738: #define INF_MAIN_DEV_NAME      "wlan"
1739: #define INF_MBSSID_DEV_NAME    "wlan"
1740: #else
1741: #define INF_MAIN_DEV_NAME      "ra"
1742: #define INF_MBSSID_DEV_NAME    "ra"
1743: #endif /* ANDROID_SUPPORT */
1744: #define INF_WDS_DEV_NAME       "wds"
1745: #define INF_APCLI_DEV_NAME     "apcli"
1746: #define INF_MESH_DEV_NAME      "mesh"
1747: #define INF_P2P_DEV_NAME       "p2p"

```

图 3 设置网卡的名称

4 修改 Makefile.inc

4.1 使网卡工作于 STA 模式

由于我们使用网卡事，网卡作为 wifi 客户端使用，所以需要网卡工作于 STA 模式，如下图所示。

```

1: ifeq ($(WIFI_MODE),)
2: RT28xx_MODE = STA
3: else
4: RT28xx_MODE = $(WIFI_MODE)
5: endif

```

图 4 使网卡工作于 STA 模式

4.2 修改驱动程序所支持的平台

这里没有我们对应的 AM3352 平台，我们选择相近的平台即可，修改后的代码如下图所示。

```

29: #PLATFORM: Target platform
30: #PLATFORM = PC
31: #PLATFORM = 5VT
32: #PLATFORM = IKANOS_V160
33: #PLATFORM = IKANOS_V180
34: #PLATFORM = SIGMA
35: #PLATFORM = SIGMA_8622
36: #PLATFORM = INIC
37: #PLATFORM = STAR
38: #PLATFORM = IXP
39: #PLATFORM = INF_TWINPASS
40: #PLATFORM = INF_DANUBE
41: #PLATFORM = INF_AR9
42: #PLATFORM = INF_VR9
43: #PLATFORM = BCM_6358
44: #PLATFORM = INF_AMAZON_SE
45: #PLATFORM = CAVM_OCTEON
46: #PLATFORM = CMPC
47: #PLATFORM = RALINK_2880
48: #PLATFORM = RALINK_3052
49: #PLATFORM = SMDK
50: #PLATFORM = RMI

```

图 5 修改所支持的平台

4.3 内核路径和交叉编译工链前缀

在编译驱动程序时需要使用到事先已经编译好的内核源码，所以我们在这里需要指定源码路径。由于我们要交叉编译所以要指定交叉编译工具链的路径和前缀。修改好的代码如下图所示。

```

275 ifeq ($(PLATFORM),SMDK)
276 LINUX_SRC = /work/am335x/linux-3.14.26
277 CROSS_COMPILE = /opt/ti-sdk-am335x-evm-08.00.00.00/linux-devkit/sysroots/i686-arago-linux/usr/bin/arm-linux-gnueabi-
278 endif

```

图 6 指定已经编译好的内核路径和交叉编译工具链前缀

5 修改源码消除编译错误

原来的驱动程序是在 2.6 内核的基础上开发的，我们现在所使用的内核是 3.14 版本的，有些函数定义发生了变化，需稍作修改。需要修改的文件为 DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/UTIL/os/linux/rt_linux.c，修改后的代码如下图所示。

```

1178: static inline void __RtmpOSFSInfoChange(OS_FS_INI
1179: {
1180:     if (bSet) {
1181:         /* Save uid and gid used for filesystem access. */
1182:         /* Set user and group to 0 (root) */
1183:         #if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,29)
1184:             pOSFSInfo->fsuid = current->fsuid;
1185:             pOSFSInfo->fsgid = current->fsgid;
1186:             current->fsuid = current->fsgid = 0;
1187:         #else
1188:             pOSFSInfo->fsuid = *(int *)&current_fsuid();
1189:             pOSFSInfo->fsgid = *(int *)&current_fsgid();
1190:         #endif
1191:         pOSFSInfo->fs = get_fs();
1192:         set_fs(KERNEL_DS);
1193:     } else {
1194:         set_fs(pOSFSInfo->fs);
1195:         #if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,29)
1196:             current->fsuid = pOSFSInfo->fsuid;
1197:             current->fsgid = pOSFSInfo->fsgid;
1198:         #endif
1199:     }
1200: } « end __RtmpOSFSInfoChange »

```

图 7 修改源码消除编译错误

6 修改源码 Bug

6.1 官方驱动程序源码 Bug 分析

MT7601 官方驱动程序里面存在 Bug，在插上 USB 网卡时申请了内存，在拔掉网卡时没有将申请的内存全部释放掉。所以在多次拔插网卡后会出现内存申请失败而无法识别网卡的情况，在实际使用中多次拔插网卡后出现的情况如下图所示。

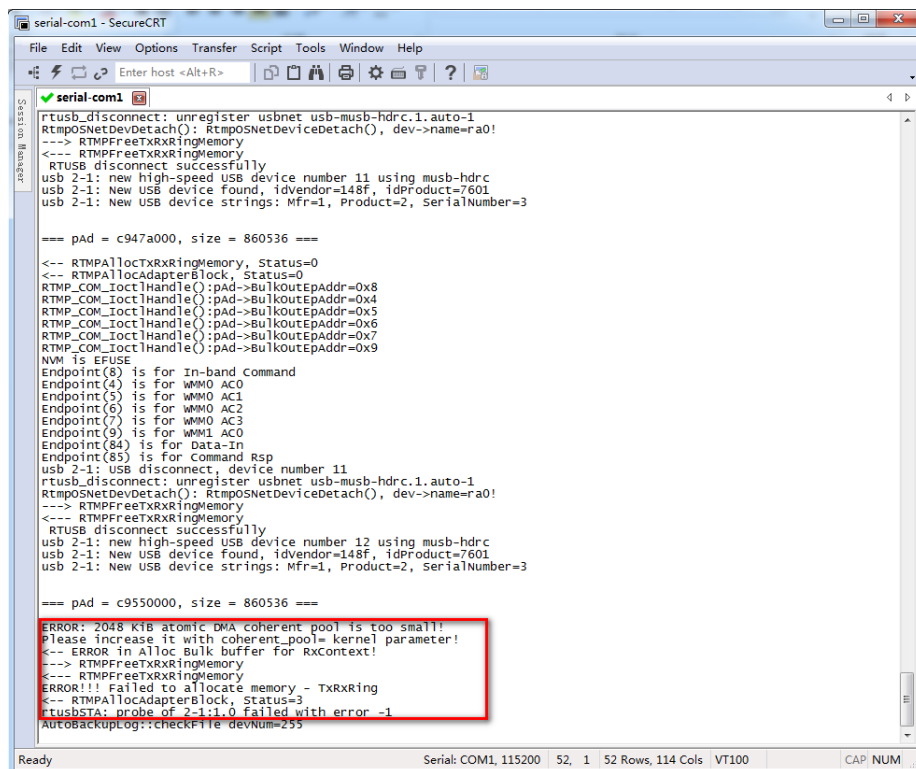


图 8 多次拔插网卡后出错现象

上图的出错现象表明，出错的原因是由于用于 DMA 传输的内存分配失败导致的，所以官方提供的驱动程序里面存在内存泄露的情况在插上网卡时会触发 RTMPAllocTxRxRingMemory 函数调用 RTMPAllocUsbBulkBufStruct 函数申请内存，在拔掉网卡后会触发 RTMPFreeTxRxRingMemory 函数调用 RTMPFreeUsbBulkBufStruct 函数将申请的内存释放。经过分析源码找到了内存泄露的位置，一共有两个位置出现了内存泄露，现将这两种情况分析如下。存在 Bug 的文件为：

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/MODULE/common/cmm_mac_usb.c

定义了 RESOURCE_PRE_ALLOC 的情况下，内存申请和释放的位置图 9 和图 10 所示。

```

545: NDIS_STATUS RTMPAllocTxRxRingMemory(
546: IN PRTMP_ADAPTER pAd)
547: {
548:     NDIS_STATUS Status = NDIS_STATUS_FAILURE;
549:     PTX_CONTEXT pNullContext = &(pAd->NullContext);
550:     PTX_CONTEXT pPsPollContext = &(pAd->PsPollContext);
551:     PCMD_RSP_CONTEXT pCmdRspEventContext = &(pAd->CmdRspEventContext);
552:     INT i, acidx;
553:
554:     DBGPRINT(RT_DEBUG_TRACE, ("--> RTMPAllocTxRxRingMemory\n"));
555:
556:     do
557:     {
558:
559:         /* Init send data structures and related parameters*/
560:
561:         /* TX_RING_SIZE, 4 ACs*/
562:
563:         for(acidx=0; acidx<NUM_OF_TX_RING; acidx++)
564:         {
565:             PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
566:
567:             NdisZeroMemory(pHTTXContext, sizeof(HT_TX_CONTEXT));
568:             /*Allocate URB and bulk buffer*/
569:             Status = RTMPAllocUsbBulkBufStruct(pAd,
570:                 &pHTTXContext->pUrb,
571:                 (PVOID *)&pHTTXContext->TransferBuffer,
572:                 sizeof(HTTX_BUFFER),
573:                 &pHTTXContext->data_dma,
574:                 "HTTxContext");
575:             if (Status != NDIS_STATUS_SUCCESS)
576:                 goto ↓err;
577:         }
578:     }
579:

```

图 9 插上网卡后申请内存的位置

```

273: /* Free Tx frame resource*/
274: for (acidx = 0; acidx < 4; acidx++)
275: {
276:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
277:     if (pHTTXContext)
278:     {
279:         RTMPFreeUsbBulkBufStruct(pAd,
280:             &pHTTXContext->pUrb,
281:             (PUCHAR *)&pHTTXContext->TransferBuffer,
282:             sizeof(HTTX_BUFFER),
283:             pHTTXContext->data_dma);
284:     }
285:
286:     if (pAd->FragFrame.pFragPacket)
287:         RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);
288:
289:     DBGPRINT(RT_DEBUG_ERROR, ("<--- RTMPFreeTxRxRingMemory\n"));
290: } « end RTMPFreeTxRxRingMemory »

```

图 10 拔掉网卡后内存释放的位置

没有定义 RESOURCE_PRE_ALLOC 的情况下，内存申请和释放的位置图 11 和图 12 所示。

```

832: NDIS_STATUS NICInitTransmit(
833:     IN PRTMP_ADAPTER pAd)
834: {
835:     UCHAR i, acidx;
836:     NDIS_STATUS Status = NDIS_STATUS_SUCCESS;
837:     PTX_CONTEXT pNullContext = &(pAd->NullContext[0]);
838:     PTX_CONTEXT pPsPollContext = &(pAd->PsPollContext);
839:     PTX_CONTEXT pMLMEContext = NULL;
840:     POS_COOKIE pObj = (POS_COOKIE) pAd->OS_Cookie;
841:     PVOID RingBaseVa;
842:     RTMP_MGMT_RING *pMgmtRing;
843:
844:     DBGPRINT(RT_DEBUG_TRACE, ("--> NICInitTransmit\n"));
845:     pObj = pObj;
846:
847:     /* Init 4 set of Tx parameters*/
848:     for(acidx = 0; acidx < NUM_OF_TX_RING; acidx++)
849:     {
850:         /* Initialize all Transmit related queues*/
851:         InitializeQueueHeader(&pAd->TxSwQueue[acidx]);
852:
853:         /* Next Local tx ring pointer waiting for buck out*/
854:         pAd->NextBulkOutIndex[acidx] = acidx;
855:         pAd->BulkOutPending[acidx] = FALSE; /* Buck Out control flag */
856:     }
857:
858:
859:     do
860:     {
861:
862:         /* TX_RING_SIZE, 4 ACs*/
863:
864:         for(acidx=0; acidx<NUM_OF_TX_RING; acidx++)
865:         {
866:             PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
867:
868:             NdisZeroMemory(pHTTXContext, sizeof(HT_TX_CONTEXT));
869:             /*Allocate URB*/
870:             Status = RTMPAllocUsbBulkBufStruct(pAd,
871:                 &pHTTXContext->pUrb,
872:                 (PVOID *)&pHTTXContext->TransferBuffer,
873:                 sizeof(HTTX_BUFFER),
874:                 &pHTTXContext->data_dma,
875:                 "HTTxContext");

```

图 11 插上网卡后申请内存的位置

```

1223: /* Free Tx frame resource*/
1224: for (acidx = 0; acidx < 4; acidx++)
1225: {
1226:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
1227:     if (pHTTXContext)
1228:     {
1229:         RTMPFreeUsbBulkBufStruct(pAd,
1230:             &pHTTXContext->pUrb,
1231:             (PUCHAR *)&pHTTXContext->TransferBuffer,
1232:             sizeof(HTTX_BUFFER),
1233:             pHTTXContext->data_dma);
1234:     }
1235:
1236:     /* Free fragmentation frame buffer*/
1237:     if (pAd->FragFrame.pFragPacket)
1238:         RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);

```

图 12 拔掉网卡后内存释放的位置

插上网卡后 RTMPAllocTxRxRingMemory 函数调用 RTMPAllocUsbBulkBufStruct 函数申请内存块的数量是由 NUM_OF_TX_RING 宏确定的，在拔掉网卡后 RTMPFreeTxRxRingMemory 函数调用 RTMPFreeUsbBulkBufStruct 函数释放内存块的数量固定为 4，NUM_OF_TX_RING 宏定义如下图所示。


```

2334: #define PID_MGMT          0x05
2335: #define PID_BEACON        0x0c
2336: #define PID_DATA_NORMALCAST 0x02
2337: #define PID_DATA_AMPDU    0x04
2338: #define PID_DATA_NO_ACK   0x08
2339: #define PID_DATA_NOT_NORM_ACK 0x03
2340: /* value domain of pTxD->HostQId (4-bit: 0~15) */
2341: #define QID_AC_BK          1 /* meet ACI definition in 802.11e */
2342: #define QID_AC_BE          0 /* meet ACI definition in 802.11e */
2343: #define QID_AC_VI          2
2344: #define QID_AC_VO          3
2345: #define QID_HCCA           4
2346: #define NUM_OF_TX_RING     5
2347: #define QID_CTRL           9
2348: #define QID_MGMT           13
2349: #define QID_RX             14
2350: #define QID_OTHER          15

```

图 12 NUM_OF_TX_RING 宏定义代码

经上面的分析可知，网卡插上之后 RTMPAllocTxRxRingMemory 函数申请了 5 块内存，在拔掉网卡后 RTMPFreeTxRxRingMemory 函数只释放了 4 块，存在内存泄露。

6.2 官方驱动源码 Bug 修改

定义了 RESOURCE_PRE_ALLOC 的情况下，修改后的代码如下图。

```

273: /* Free Tx frame resource*/
274: for (acidx = 0; acidx < NUM_OF_TX_RING; acidx++) |
275: {
276:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
277:     if (pHTTXContext)
278:         RTMPFreeUsbBulkBufStruct(pAd,
279:             &pHTTXContext->pUrb,
280:             (PUCHAR *)&pHTTXContext->TransferBuffer,
281:             sizeof(HTTX_BUFFER),
282:             pHTTXContext->data_dma);
283: }
284:
285: if (pAd->FragFrame.pFragPacket)
286:     RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);
287:
288:
289: DBGPRINT(RT_DEBUG_ERROR, ("<--- RTMPFreeTxRxRingMemory\n"));
290: } « end RTMPFreeTxRxRingMemory »

```

图 13 定义了 RESOURCE_PRE_ALLOC 修改后代码

没有定义 RESOURCE_PRE_ALLOC 的情况下，修改后的代码如下图。

```

1223: /* Free Tx frame resource*/
1224: for (acidx = 0; acidx < NUM_OF_TX_RING; acidx++) |
1225: {
1226:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
1227:     if (pHTTXContext)
1228:         RTMPFreeUsbBulkBufStruct(pAd,
1229:             &pHTTXContext->pUrb,
1230:             (PUCHAR *)&pHTTXContext->TransferBuffer,
1231:             sizeof(HTTX_BUFFER),
1232:             pHTTXContext->data_dma);
1233: }
1234:
1235: /* Free fragmentation frame buffer*/
1236: if (pAd->FragFrame.pFragPacket)
1237:     RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);

```

图 14 未定义 RESOURCE_PRE_ALLOC 修改后代码

7 修改源码确定配置文件的存放位置

MT7601 网卡在使用时会依赖一个配置文件，该配置文件的存放位置和文件名称可根据

个人习惯随意指定，一共有 3 个地方需要修改，在这三个地方配置文件的位置和名称要保持一致，三个地方分别为：

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/NETIF/include/os/rt_linux.h 155 行

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/UTIL/include/os/rt_linux.h 155 行

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/MODULE/include/os/rt_linux.h 155 行

示例代码如下图所示：

```
154: #ifdef RTMP_MAC_USB
155: #define STA_PROFILE_PATH          "/etc/Wireless/RT2870STA/RT2870STA.dat"
156: #define STA_DRIVER_VERSION        "3.0.0.4"
157: #ifdef MULTIPLE_CARD_SUPPORT
158: #define CARD_INFO_PATH            "/etc/Wireless/RT2870STA/RT2870STACard.dat"
159: #endif /* MULTIPLE_CARD_SUPPORT */
160: #endif /* RTMP_MAC_USB */
```

图 15 指定配置文件位置及名称

8 添加 wpa_supplicant 支持

我们最终要使用 wpa_supplicant 工具来管理无线热点，所以在源码里要添加 wpa_supplicant 支持，分别修改下列文件的第 28 行和第 33 行，添加 wpa_supplicant 支持。

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/config.mk

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/MODULE/os/linux/config.mk

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/UTIL/os/linux/config.mk

DPA_MT7601U_LinuxSTA_3.0.0.4_rev2_20140221/NETIF/os/linux/config.mk

修改后的代码如下图所示：

```
26: # Support Wpa_Supplicant
27: # i.e. wpa_supplicant -Dralink
28: HAS_WPA_SUPPLICANT=y
29:
30:
31: # Support Native WpaSupplicant for Network Manager
32: # i.e. wpa_supplicant -Dwext
33: HAS_NATIVE_WPA_SUPPLICANT_SUPPORT=y
--
```

图 16 添加 wpa_supplicant 支持

9 编译

DPA 版本驱动程序编译后正常情况下会生成三个 ko 文件，AM3352 平台编译过程如下图所示。


```

1 #!/bin/bash
2 export PATH=$PATH:/opt/ti-sdk-am335x-evm-08.00.00.00/linux-devkit/sysroots/i686-arago-linux/usr/bin/
3 export ARCH=arm
4
5 make clean
6 make
7
8 cp ./NETIF/os/linux/mtnet7601Usta.ko ./
9 cp ./UTIL/os/linux/mtutil7601Usta.ko ./
10 cp ./MODULE/os/linux/mt7601Usta.ko ./
11
12 arm-linux-gnueabi-hf-strip -S mt7601Usta.ko
13 arm-linux-gnueabi-hf-strip -S mtnet7601Usta.ko
14 arm-linux-gnueabi-hf-strip -S mtutil7601Usta.ko

```

图 17 AM3352 平台编译脚本

10 测试

1) 加载驱动程序

加载顺序如下，加载顺序不可改变，卸载时逆向即可

```
insmod mtutil7601Usta.ko
```

```
insmod mt7601Usta.ko
```

```
insmod mtnet7601Usta.ko
```

2) 添加配置文件

将 RT2870STA.dat 文件添加到 /etc/Wireless/RT2870STA/ 目录，并修改 RT2870STA.dat 文件的 SSID 和 WPAESK，SSID 为热点名称，WPAESK 为密码，如下图所示

```

1 #The word of "Default" must not be removed
2 Default
3 CountryRegion=5
4 CountryRegionABand=7
5 CountryCode=
6 ChannelGeography=1
7 SSID=troy xiao
8 NetworkType=Infra
9 WirelessMode=9
10 Channel=0
11 BeaconPeriod=100
12 TxPower=100
13 BGProtection=0
14 TxPreamble=0
15 RTSThreshold=2347
16 FragThreshold=2346
17 TxBurst=1
18 PktAggregate=0
19 WmmCapable=1
20 AckPolicy=0;0;0;0
21 AuthMode=OPEN
22 EncrypType=NONE
23 WPAESK=1234567890
24 DefaultKeyID=1

```

图 17 RT2870STA.dat 文件修改位置

3) 连接网卡、启动网卡、获取 IP

将网卡连接到 USB: `ifconfig -a`

如下图所示，已经识别出网卡

```

~ # ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:27:1D:D2:03:9A
          inet addr:192.168.1.254  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:278 errors:0 dropped:217 overruns:0 frame:0
          TX packets:59 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:38302 (37.4 KiB)  TX bytes:3662 (3.5 KiB)
          Interrupt:56

eth0:1    Link encap:Ethernet  HWaddr 00:27:1D:D2:03:9A
          inet addr:195.60.16.254  Bcast:195.60.16.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          Interrupt:56

eth1      Link encap:Ethernet  HWaddr 00:30:BF:48:59:FF
          inet addr:192.167.101.254  Bcast:192.167.101.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:2176 (2.1 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:407 (407.0 B)  TX bytes:407 (407.0 B)

ra0       Link encap:Ethernet  HWaddr 24:05:0F:16:0F:89
          inet addr:192.168.43.209  Bcast:192.168.43.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4525 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1068777 (1.0 MiB)  TX bytes:6234 (6.0 KiB)

```

图 18 查看是否识别出网卡

自动获取 ip: udhcpc -i ra0

```

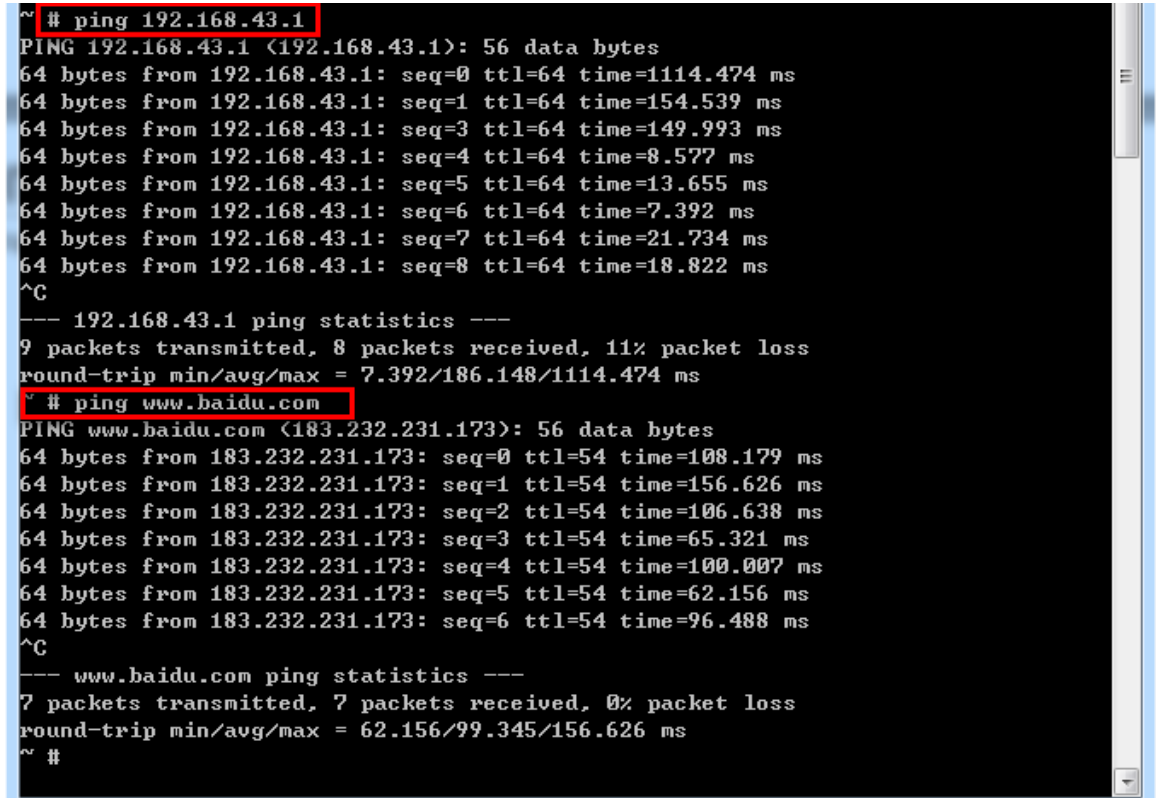
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

~ # udhcpc -i ra0
udhcpc (v1.22.1) started
Setting IP address 0.0.0.0 on ra0
Sending discover...
Sending select for 192.168.43.209...
Lease of 192.168.43.209 obtained, lease time 3600
Setting IP address 192.168.43.209 on ra0
Deleting routers
route: SIOCDELRT: No such process
Adding router 192.168.43.1
Recreating /etc/resolv.conf
Adding DNS server 192.168.43.1
~ #

```

图 19 自动获取 ip

ping 内外网:



```
~ # ping 192.168.43.1
PING 192.168.43.1 (192.168.43.1): 56 data bytes
64 bytes from 192.168.43.1: seq=0 ttl=64 time=1114.474 ms
64 bytes from 192.168.43.1: seq=1 ttl=64 time=154.539 ms
64 bytes from 192.168.43.1: seq=3 ttl=64 time=149.993 ms
64 bytes from 192.168.43.1: seq=4 ttl=64 time=8.577 ms
64 bytes from 192.168.43.1: seq=5 ttl=64 time=13.655 ms
64 bytes from 192.168.43.1: seq=6 ttl=64 time=7.392 ms
64 bytes from 192.168.43.1: seq=7 ttl=64 time=21.734 ms
64 bytes from 192.168.43.1: seq=8 ttl=64 time=18.822 ms
^C
--- 192.168.43.1 ping statistics ---
9 packets transmitted, 8 packets received, 11% packet loss
round-trip min/avg/max = 7.392/186.148/1114.474 ms
~ # ping www.baidu.com
PING www.baidu.com (183.232.231.173): 56 data bytes
64 bytes from 183.232.231.173: seq=0 ttl=54 time=108.179 ms
64 bytes from 183.232.231.173: seq=1 ttl=54 time=156.626 ms
64 bytes from 183.232.231.173: seq=2 ttl=54 time=106.638 ms
64 bytes from 183.232.231.173: seq=3 ttl=54 time=65.321 ms
64 bytes from 183.232.231.173: seq=4 ttl=54 time=100.007 ms
64 bytes from 183.232.231.173: seq=5 ttl=54 time=62.156 ms
64 bytes from 183.232.231.173: seq=6 ttl=54 time=96.488 ms
^C
--- www.baidu.com ping statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max = 62.156/99.345/156.626 ms
~ #
```

图 20 ping 内外网测试

至此 MT7601 网卡 DPA 版本的驱动程序, 已经成功移植到 AM335x 平台, wpa_supplicant 移植过程请参看, 《wpa_supplicant 移植总结》

DPO 版本驱动程序移植过程详解

1 下载并解压源码

```
tar -xzf DPO_MT7601U_LinuxSTA_3.0.0.4_20130913.tar.gz
```

2 在驱动程序里面添加自己所使用的网卡

2.1 查看自己所使用网卡的型号

3) 将网卡插到电脑上，通过 lsusb 命令查看型号，如下图：

```
-virtual-machine:/work/am335x$
-virtual-machine:/work/am335x$
-virtual-machine:/work/am335x$ lsusb
Bus 001 Device 015: ID 148f:7601 Ralink Technology, Corp. MT7601U Wireless Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
-virtual-machine:/work/am335x$
```

图 1 查看网卡型号

如上图所示我们使用的 USB 网卡的 VID 和 PID 分别为 148f 和 7601

4) 修改 DPO_MT7601U_LinuxSTA_3.0.0.4_20130913/common/rtusb_dev_id.c 文件，添加网卡支持，修改后的代码如下图所示：

```
36: /* module table */
37: USB_DEVICE_ID rtusb_dev_id[] = {
38: #ifdef RT6570
39:     {USB_DEVICE(0x148f, 0x6570)}, /* Ralink 6570 */
40: #endif /* RT6570 */
41:     {USB_DEVICE(0x148f, 0x7650)}, /* MT7650 */
42: #ifdef MT7601U
43:     {USB_DEVICE(0x148f, 0x6370)}, /* Ralink 6370 */
44:     {USB_DEVICE(0x148f, 0x7601)}, /* MT 6370 */
45: #endif /* MT7601U */
46:     { } /* Terminating entry */
47: };
```

图 2 添加网卡支持

3 设置网卡名称（可以不修改，根据个人习惯）

修改 DPO_MT7601U_LinuxSTA_3.0.0.4_20130913/include/rtmp_def.h 文件，设置网卡的名字，修改后的代码如下图所示：

```

1600: #ifdef ANDROID_SUPPORT
1601: #define INF_MAIN_DEV_NAME      "wlan"
1602: #define INF_MBSSID_DEV_NAME    "wlan"
1603: #else
1604: #define INF_MAIN_DEV_NAME      "ra"
1605: #define INF_MBSSID_DEV_NAME    "ra"
1606: #endif /* ANDROID_SUPPORT */
1607: #define INF_WDS_DEV_NAME        "wds"
1608: #define INF_APCLI_DEV_NAME      "apcli"
1609: #define INF_MESH_DEV_NAME       "mesh"
1610: #define INF_P2P_DEV_NAME        "p2p"

```

图 3 设置网卡的名字

4 修改 Makefile

4.1 使网卡工作于 STA 模式

由于我们使用网卡事，网卡作为 wifi 客户端使用，所以需要网卡工作于 STA 模式，如下图所示。

```

1: ifeq ($(WIFI_MODE),)
2: RT28xx_MODE = STA
3: else
4: RT28xx_MODE = $(WIFI_MODE)
5: endif
~

```

图 4 使网卡工作于 STA 模式

4.2 修改驱动程序所支持的平台

这里没有我们对应的 AM3352 平台，我们选择相近的平台即可，修改后的代码如下图所示。

```

29: #PLATFORM: Target platform
30: #PLATFORM = PC
31: #PLATFORM = SVT
32: #PLATFORM = IKANOS_V160
33: #PLATFORM = IKANOS_V180
34: #PLATFORM = SIGMA
35: #PLATFORM = SIGMA_8622
36: #PLATFORM = INIC
37: #PLATFORM = STAR
38: #PLATFORM = IXP
39: #PLATFORM = INF_TWINPASS
40: #PLATFORM = INF_DANUBE
41: #PLATFORM = INF_AR9
42: #PLATFORM = INF_VR9
43: #PLATFORM = BRCM_6358
44: #PLATFORM = INF_AMAZON_SE
45: #PLATFORM = CAVM_OCTEON
46: #PLATFORM = CMPC
47: #PLATFORM = RALINK_2880
48: #PLATFORM = RALINK_3052
49: PLATFORM = SMDK
50: #PLATFORM = RMI
51: #PLATFORM = RMI_64

```

图 5 修改所支持的平台

4.3 内核路径和交叉编译工具链前缀

在编译驱动程序时需要使用到事先已经编译好的内核源码，所以我们在这里需要指定源码路径。由于我们要交叉编译所以要指定交叉编译工具链的路径和前缀。修改好的代码如下图所示。

```
275 ifeq ($(PLATFORM),SMDK)
276 LINUX_SRC = /work/am335x/linux-3.14.26
277 CROSS_COMPILE = /opt/ti-sdk-am335x-evm-08.00.00.00/linux-devkit/sysroots/i686-arago-linux/usr/bin/arm-linux-gnueabihf-
278 endif
```

图 6 指定已经编译好的内核路径和交叉编译工具链前缀

5 修改源码消除编译错误

原来的驱动程序是在 2.6 内核的基础上开发的，我们现在所使用的内核是 3.14 版本的，有些函数定义发生了变化，需稍作修改。需要修改的文件为 DPO_MT7601U_LinuxSTA_3.0.0.4_20130913/os/linux/rt_linux.c，修改后的代码如下图所示。

```
1111: static inline void __RtmpOSFSInfoChange(OS_FS_INFO * pOSFSInfo, BOOLEAN bSet
1112: {
1113:     if (bSet) {
1114:         /* Save uid and gid used for filesystem access. */
1115:         /* Set user and group to 0 (root) */
1116:         #if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,29)
1117:             pOSFSInfo->fsuid = current->fsuid;
1118:             pOSFSInfo->fsgid = current->fsgid;
1119:             current->fsuid = current->fsgid = 0;
1120:         #else
1121:             pOSFSInfo->fsuid = *(int *)&current_fsuid();
1122:             pOSFSInfo->fsgid = *(int *)&current_fsgid();
1123:         #endif
1124:         pOSFSInfo->fs = get_fs();
1125:         set_fs(KERNEL_DS);
1126:     } else {
1127:         set_fs(pOSFSInfo->fs);
1128:         #if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,29)
1129:             current->fsuid = pOSFSInfo->fsuid;
1130:             current->fsgid = pOSFSInfo->fsgid;
1131:         #endif
1132:     }
1133: } « end __RtmpOSFSInfoChange »
```

图 7 修改源码消除编译错误

6 修改源码 Bug

6.1 官方驱动程序源码 Bug 分析

MT7601 官方驱动程序里面存在 Bug，在插上 USB 网卡时申请了内存，在拔掉网卡时没有将申请的内存全部释放掉。所以在多次拔插网卡后会出现内存申请失败而无法识别网卡的情况，在实际使用中多次拔插网卡后出现的情况如下图所示。


```

serial-com1 - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Enter host <Alt+R>
serial-com1
rtusb_disconnect: unregister usbnet usb-musb-hdrc.1.auto-1
RtmpoNetDevDetach(): RtmpoNetDeviceDetach(), dev->name=ra0!
--> RTMPFreeTxRxRingMemory
<--- RTMPFreeTxRxRingMemory
RTUSB disconnect successfully
usb 2-1: new high-speed USB device number 11 using musb-hdrc
usb 2-1: New USB device found, idVendor=148f, idProduct=7601
usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3

=== pad = c947a000, size = 860536 ===
<-- RTMPAllocTxRxRingMemory, Status=0
<-- RTMPAllocAdapterBlock, Status=0
RTMP_COM_IoctHandle(): pad->BulkoutEpAddr=0x8
RTMP_COM_IoctHandle(): pad->BulkoutEpAddr=0x4
RTMP_COM_IoctHandle(): pad->BulkoutEpAddr=0x5
RTMP_COM_IoctHandle(): pad->BulkoutEpAddr=0x6
RTMP_COM_IoctHandle(): pad->BulkoutEpAddr=0x7
RTMP_COM_IoctHandle(): pad->BulkoutEpAddr=0x9
NVM is EFUSE
Endpoint(8) is for In-band Command
Endpoint(4) is for WMMQ AC0
Endpoint(5) is for WMMQ AC1
Endpoint(6) is for WMMQ AC2
Endpoint(7) is for WMMQ AC3
Endpoint(9) is for WMM1 AC0
Endpoint(84) is for Data-In
Endpoint(85) is for Command Rsp
usb 2-1: USB disconnect, device number 11
rtusb_disconnect: unregister usbnet usb-musb-hdrc.1.auto-1
RtmpoNetDevDetach(): RtmpoNetDeviceDetach(), dev->name=ra0!
--> RTMPFreeTxRxRingMemory
<--- RTMPFreeTxRxRingMemory
RTUSB disconnect successfully
usb 2-1: new high-speed USB device number 12 using musb-hdrc
usb 2-1: New USB device found, idVendor=148f, idProduct=7601
usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3

=== pad = c9550000, size = 860536 ===
ERROR: 2048 KiB atomic DMA coherent pool is too small!
Please increase it with coherent_pool= kernel parameter!
<-- ERROR in Alloc Bulk buffer for RxContext!
--> RTMPFreeTxRxRingMemory
<--- RTMPFreeTxRxRingMemory
ERROR!!! Failed to allocate memory - TxRxRing
<-- RTMPAllocAdapterBlock, Status=3
rtusbSTA: probe of 2-1:1.0 failed with error -1
AutoBackupLog::checkFile devNum=255

Ready Serial: COM1, 115200 52, 1 52 Rows, 114 Cols VT100 CAP NUM

```

图 8 多次拔插网卡后出错现象

上图的出错现象表明，出错的原因是由于用于 DMA 传输的内存分配失败导致的，所以官方提供的驱动程序里面存在内存泄露的情况在插上网卡时会触发 RTMPAllocTxRxRingMemory 函数调用 RTMPAllocUsbBulkBufStruct 函数申请内存，在拔掉网卡后会触发 RTMPFreeTxRxRingMemory 函数调用 RTMPFreeUsbBulkBufStruct 函数将申请的内存释放。。经过分析源码找到了内存泄露的位置，一共有两个位置出现了内存泄露，现将这两种情况分析如下。存在 Bug 的文件为：

[DPO_MT7601U_LinuxSTA_3.0.0.4_20130913/common/cmm_mac_usb.c](#)

定义了 RESOURCE_PRE_ALLOC 的情况下，内存申请和释放的位置图 9 和图 10 所示。

```

558: NDIS_STATUS RTMPAllocTxRxRingMemory(
559:     IN PRTMP_ADAPTER pAd)
560: {
561:     NDIS_STATUS Status = NDIS_STATUS_FAILURE;
562:     PTX_CONTEXT pNullContext = &(pAd->NullContext);
563:     PTX_CONTEXT pPsPollContext = &(pAd->PsPollContext);
564:     PCMD_RSP_CONTEXT pCmdRspEventContext = &(pAd->CmdRspEventContext);
565:     INT i, acidx;
566:
567:     DBGPRINT(RT_DEBUG_TRACE, ("--> RTMPAllocTxRxRingMemory\n"));
568:
569:     do
570:     {
571:
572:         /* Init send data structures and related parameters*/
573:
574:
575:         /* TX_RING_SIZE, 4 ACs*/
576:
577:         for(acidx=0; acidx<NUM_OF_TX_RING; acidx++)
578:         {
579:             PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
580:
581:             NdisZeroMemory(pHTTXContext, sizeof(HT_TX_CONTEXT));
582:             /*Allocate URB and bulk buffer*/
583:             Status = RTMPAllocUsbBulkBufStruct(pAd,
584:                 &pHTTXContext->pUrb,
585:                 (PVOID *)&pHTTXContext->TransferBuffer,
586:                 sizeof(HTTX_BUFFER),
587:                 &pHTTXContext->data_dma,
588:                 "HTTxContext");
589:             if (Status != NDIS_STATUS_SUCCESS)
590:                 goto ↓err;
591:         }

```

图 9 插上网卡后申请内存的位置

```

286: /* Free Tx frame resource*/
287: for (acidx = 0; acidx < 4; acidx++)
288: {
289:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
290:     if (pHTTXContext)
291:     {
292:         RTMPFreeUsbBulkBufStruct(pAd,
293:             &pHTTXContext->pUrb,
294:             (PUCHAR *)&pHTTXContext->TransferBuffer,
295:             sizeof(HTTX_BUFFER),
296:             pHTTXContext->data_dma);
297:     }
298:     if (pAd->FragFrame.pFragPacket)
299:         RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);
300:
301:
302:     DBGPRINT(RT_DEBUG_ERROR, ("<--- RTMPFreeTxRxRingMemory\n"));
303: } « end RTMPFreeTxRxRingMemory »

```

图 10 拔掉网卡后内存释放的位置

没有定义 RESOURCE_PRE_ALLOC 的情况下，内存申请和释放的位置图 11 和图 12 所示。

```

845: NDIS_STATUS NICInitTransmit(
846:     IN PRTMP_ADAPTER pAd)
847: {
848:     UCHAR i, acidx;
849:     NDIS_STATUS Status = NDIS_STATUS_SUCCESS;
850:     PTX_CONTEXT pNullContext = &(pAd->NullContext[0]);
851:     PTX_CONTEXT pPsPollContext = &(pAd->PsPollContext);
852:     PTX_CONTEXT pMLMEContext = NULL;
853:     POS_COOKIE pObj = (POS_COOKIE) pAd->OS_Cookie;
854:     PVOID RingBaseVa;
855:     RTMP_MGMT_RING *pMgmtRing;
856:
857:     DBGPRINT(RT_DEBUG_TRACE, ("--> NICInitTransmit\n"));
858:     pObj = pObj;
859:
860:     /* Init 4 set of Tx parameters*/
861:     for(acidx = 0; acidx < NUM_OF_TX_RING; acidx++)
862:     {
863:         /* Initialize all Transmit releated queues*/
864:         InitializeQueueHeader(&pAd->TxSwQueue[acidx]);
865:
866:         /* Next Local tx ring pointer waiting for buck out*/
867:         pAd->NextBulkOutIndex[acidx] = acidx;
868:         pAd->BulkOutPending[acidx] = FALSE; /* Buck Out control flag */
869:     }
870:
871:     do
872:     {
873:
874:         /* TX_RING_SIZE, 4 ACs*/
875:
876:         for(acidx=0; acidx<NUM_OF_TX_RING; acidx++)
877:         {
878:             PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
879:
880:             NdisZeroMemory(pHTTXContext, sizeof(HT_TX_CONTEXT));
881:             /*Allocate URB*/
882:             Status = RTMPAllocUsbBulkBufStruct(pAd,
883:                 &pHTTXContext->pUrb,
884:                 (PVOID *)&pHTTXContext->TransferBuffer,
885:                 sizeof(HTTX_BUFFER),
886:                 &pHTTXContext->data_dma,
887:                 "HTTxContext");
888:

```

图 11 插上网卡后申请内存的位置

```

1236: /* Free Tx frame resource*/
1237: for (acidx = 0; acidx < 4; acidx++)
1238: {
1239:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
1240:     if (pHTTXContext)
1241:         RTMPFreeUsbBulkBufStruct(pAd,
1242:             &pHTTXContext->pUrb,
1243:             (PUCHAR *)&pHTTXContext->TransferBuffer,
1244:             sizeof(HTTX_BUFFER),
1245:             pHTTXContext->data_dma);
1246: }
1247:
1248: /* Free fragement frame buffer*/
1249: if (pAd->FragFrame.pFragPacket)
1250:     RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);

```

图 12 拔掉网卡后内存释放的位置

插上网卡后 RTMPAllocTxRxRingMemory 函数调用 RTMPAllocUsbBulkBufStruct 函数申请内存块的数量是由 NUM_OF_TX_RING 宏确定的，在拔掉网卡后 RTMPFreeTxRxRingMemory 函数调用 RTMPFreeUsbBulkBufStruct 函数释放内存块的数量固定为 4，NUM_OF_TX_RING 宏定义如下图所示。

```

2334: #define PID_MGMT          0x05
2335: #define PID_BEACON        0x0c
2336: #define PID_DATA_NORMALCAST 0x02
2337: #define PID_DATA_AMPDU    0x04
2338: #define PID_DATA_NO_ACK   0x08
2339: #define PID_DATA_NOT_NORM_ACK 0x03
2340: /* value domain of pTxD->HostQId (4-bit: 0~15) */
2341: #define QID_AC_BK          1 /* meet ACI definition in 802.11e */
2342: #define QID_AC_BE          0 /* meet ACI definition in 802.11e */
2343: #define QID_AC_VI          2
2344: #define QID_AC_VO          3
2345: #define QID_HCCA           4
2346: #define NUM_OF_TX_RING     5
2347: #define QID_CTRL           9
2348: #define QID_MGMT           13
2349: #define QID_RX             14
2350: #define QID_OTHER          15

```

图 12 NUM_OF_TX_RING 宏定义代码

经上面的分析可知，网卡插上之后 RTMPAllocTxRxRingMemory 函数申请了 5 块内存，在拔掉网卡后 RTMPFreeTxRxRingMemory 函数只释放了 4 块，存在内存泄露。

6.2 官方驱动源码 Bug 修改

定义了 RESOURCE_PRE_ALLOC 的情况下，修改后的代码如下图。

```

286: /* Free Tx frame resource*/
287: for (acidx = 0; acidx < NUM_OF_TX_RING; acidx++) /* miaoguoqiang fix */
288: {
289:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
290:     if (pHTTXContext)
291:     {
292:         RTMPFreeUsbBulkBufStruct(pAd,
293:             &pHTTXContext->pUrb,
294:             (PUCHAR *)&pHTTXContext->TransferBuffer,
295:             sizeof(HTTX_BUFFER),
296:             pHTTXContext->data_dma);
297:     }
298:     if (pAd->FragFrame.pFragPacket)
299:         RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);
300:
301:     DBGPRINT(RT_DEBUG_ERROR, ("<--- RTMPFreeTxRxRingMemory\n"));
302: } // end RTMPFreeTxRxRingMemory

```

图 13 定义了 RESOURCE_PRE_ALLOC 修改后代码

没有定义 RESOURCE_PRE_ALLOC 的情况下，修改后的代码如下图。

```

1236: /* Free Tx frame resource*/
1237: for (acidx = 0; acidx < NUM_OF_TX_RING; acidx++)
1238: {
1239:     PHT_TX_CONTEXT pHTTXContext = &(pAd->TxContext[acidx]);
1240:     if (pHTTXContext)
1241:     {
1242:         RTMPFreeUsbBulkBufStruct(pAd,
1243:             &pHTTXContext->pUrb,
1244:             (PUCHAR *)&pHTTXContext->TransferBuffer,
1245:             sizeof(HTTX_BUFFER),
1246:             pHTTXContext->data_dma);
1247:     }
1248:     /* Free fragment frame buffer*/
1249:     if (pAd->FragFrame.pFragPacket)
1250:         RELEASE_NDIS_PACKET(pAd, pAd->FragFrame.pFragPacket, NDIS_STATUS_SUCCESS);

```

图 14 未定义 RESOURCE_PRE_ALLOC 修改后代码

7 修改源码确定配置文件的存放位置

MT7601 网卡在使用时会依赖一个配置文件，该配置文件的存放位置和文件名称可根据

个人习惯随意指定，一共有 2 个地方需要修改，在这 2 个地方配置文件的位置和名称要保持一致，这 2 个地方分别为：

DPO_MT7601U_LinuxSTA_3.0.0.4_20130913/include/os/rt_linux.h 130 行

DPO_MT7601U_LinuxSTA_3.0.0.4_20130913/include/os/rt_drv.h 59 行

示例代码如下所示：

```
129: #ifdef RTMP_MAC_USB
130: #define STA_PROFILE_PATH "/etc/Wireless/RT2870STA/RT2870STA.dat"
131: #define STA_DRIVER_VERSION "3.0.0.3"
132: #ifdef MULTIPLE_CARD_SUPPORT
133: #define CARD_INFO_PATH "/etc/Wireless/RT2870STA/RT2870STACard.dat"
134: #endif /* MULTIPLE_CARD_SUPPORT */
135: #endif /* RTMP_MAC_USB */

58: #ifdef RTMP_MAC_USB
59: #define STA_PROFILE_PATH "/etc/Wireless/RT2870STA/RT2870STA.dat"
60: #define STA_DRIVER_VERSION "3.0.0.3"
61: #ifdef MULTIPLE_CARD_SUPPORT
62: #define CARD_INFO_PATH "/etc/Wireless/RT2870STA/RT2870STACard.dat"
63: #endif /* MULTIPLE_CARD_SUPPORT */
64: #endif /* RTMP_MAC_USB */
```

图 15 指定配置文件位置及名称

8 添加 wpa_supplicant 支持

我们最终要使用 wpa_supplicant 工具来管理无线热点，所以在源码里要添加 wpa_supplicant 支持，修改 DPO_MT7601U_LinuxSTA_3.0.0.4_20130913/os/linux/config.mk 文件的第 26 行和第 31 行，添加 wpa_supplicant 支持。

修改后的代码如下所示：

```
24: # Support Wpa_Supplicant
25: # i.e. wpa supplicant -Dralink
26: HAS_WPA_SUPPLICANT=y
27:
28:
29: # Support Native WpaSupplicant for Network Manager
30: # i.e. wpa_supplicant -Dwext
31: HAS_NATIVE_WPA_SUPPLICANT_SUPPORT=y
```

图 16 添加 wpa_supplicant 支持

9 编译

DPO 版本驱动程序编译后正常情况下会生成 1 个 ko 文件，AM3352 平台编译过程如下图所示。

```
1 #!/bin/bash
2 export PATH=$PATH:/opt/ti-sdk-am335x-evm-08.00.00.00/linux-devkit/sysroots/i686-arago-linux/usr/bin/
3 export ARCH=arm
4
5 make clean
6 make
7
8 cp os/linux/mt7601Usta.ko ./
9
10 arm-linux-gnueabi-hf-strip -S mt7601Usta.ko
```

图 17 AM3352 平台编译脚本

10 测试

4) 加载驱动程序

```
insmod mt7601Usta.ko
```

5) 添加配置文件

将 RT2870STA.dat 文件添加到 /etc/Wireless/RT2870STA/ 目录, 并修改 RT2870STA.dat 文件的 SSID 和 WPA PSK, SSID 为热点名称, WPA PSK 为密码, 如下图所示

```
1 #The word of "Default" must not be removed
2 Default
3 CountryRegion=5
4 CountryRegionABand=7
5 CountryCode=
6 ChannelGeography=1
7 SSID=troy xiao
8 NetworkType=Infra
9 WirelessMode=9
10 Channel=0
11 BeaconPeriod=100
12 TxPower=100
13 BGProtection=0
14 TxPreamble=0
15 RTSThreshold=2347
16 FragThreshold=2346
17 TxBurst=1
18 PktAggregate=0
19 WmmCapable=1
20 AckPolicy=0;0;0;0
21 AuthMode=OPEN
22 EncrypType=NONE
23 WPA PSK=1234567890
24 DefaultKeyID=1
```

图 17 RT2870STA.dat 文件修改位置

6) 连接网卡、启动网卡、获取 IP

将网卡连接到 USB: `ifconfig -a`

如下图所示, 已经识别出网卡


```

~ # ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:27:1D:D2:03:9A
          inet addr:192.168.1.254  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:278 errors:0 dropped:217 overruns:0 frame:0
          TX packets:59 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:38302 (37.4 KiB)  TX bytes:3662 (3.5 KiB)
          Interrupt:56

eth0:1    Link encap:Ethernet  HWaddr 00:27:1D:D2:03:9A
          inet addr:195.60.16.254  Bcast:195.60.16.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          Interrupt:56

eth1      Link encap:Ethernet  HWaddr 00:30:BF:48:59:FF
          inet addr:192.167.101.254  Bcast:192.167.101.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:2176 (2.1 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:407 (407.0 B)  TX bytes:407 (407.0 B)

ra0       Link encap:Ethernet  HWaddr 24:05:0F:16:0F:89
          inet addr:192.168.43.209  Bcast:192.168.43.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4525 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1068777 (1.0 MiB)  TX bytes:6234 (6.0 KiB)

```

图 18 查看是否识别出网卡

自动获取 ip: udhcpc -i ra0

```

          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

~ # udhcpc -i ra0
udhcpc (v1.22.1) started
Setting IP address 0.0.0.0 on ra0
Sending discover...
Sending select for 192.168.43.209...
Lease of 192.168.43.209 obtained, lease time 3600
Setting IP address 192.168.43.209 on ra0
Deleting routers
route: SIOCDELRT: No such process
Adding router 192.168.43.1
Recreating /etc/resolv.conf
Adding DNS server 192.168.43.1
~ #

```

图 19 自动获取 ip

ping 内外网:

```
~ # ping 192.168.43.1
PING 192.168.43.1 <192.168.43.1>: 56 data bytes
64 bytes from 192.168.43.1: seq=0 ttl=64 time=1114.474 ms
64 bytes from 192.168.43.1: seq=1 ttl=64 time=154.539 ms
64 bytes from 192.168.43.1: seq=3 ttl=64 time=149.993 ms
64 bytes from 192.168.43.1: seq=4 ttl=64 time=8.577 ms
64 bytes from 192.168.43.1: seq=5 ttl=64 time=13.655 ms
64 bytes from 192.168.43.1: seq=6 ttl=64 time=7.392 ms
64 bytes from 192.168.43.1: seq=7 ttl=64 time=21.734 ms
64 bytes from 192.168.43.1: seq=8 ttl=64 time=18.822 ms
^C
--- 192.168.43.1 ping statistics ---
9 packets transmitted, 8 packets received, 11% packet loss
round-trip min/avg/max = 7.392/186.148/1114.474 ms
~ # ping www.baidu.com
PING www.baidu.com <183.232.231.173>: 56 data bytes
64 bytes from 183.232.231.173: seq=0 ttl=54 time=108.179 ms
64 bytes from 183.232.231.173: seq=1 ttl=54 time=156.626 ms
64 bytes from 183.232.231.173: seq=2 ttl=54 time=106.638 ms
64 bytes from 183.232.231.173: seq=3 ttl=54 time=65.321 ms
64 bytes from 183.232.231.173: seq=4 ttl=54 time=100.007 ms
64 bytes from 183.232.231.173: seq=5 ttl=54 time=62.156 ms
64 bytes from 183.232.231.173: seq=6 ttl=54 time=96.488 ms
^C
--- www.baidu.com ping statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max = 62.156/99.345/156.626 ms
~ #
```

图 20 ping 内外网测试

至此 MT7601 网卡 DPO 版本的驱动程序,已经成功移植到 AM3352 平台,wpa_supplicant 移植过程请参看,《wpa_supplicant 移植教程》