Quick Start Guide for Driver Compilation and Installation

Contents

Introduction

In this document, we introduce two ways to compile and install our Wi-Fi driver: 1) Using install.sh script for PC-Linux and 2) Step by step manually. The former targets for end users who are not familiar with Linux system, while the later for engineers who want to port our Wi-Fi driver onto different platforms.

1. Using install.sh Script for PC-Linux

For driver compilation and installation in PC-Linux, we provide an install.sh script to do the duties automatically. If you want to use our Wi-Fi solutions to access network on PC-Linux, you can just run install.sh script and then control Wi-Fi with utilities such as Network Manager. For further information about Wi-Fi station mode, please refer to:

document/Quick_Start_Guide_for_Station_Mode.pdf.

If you want to apply our Wi-Fi solutions on other embedded platforms, you should read and check the following paragraphs.

2. Decompress the driver source tar ball

The driver source tar ball is located in the driver folder of our software package. For example, to decompress rtl8188C_8192C_8192D_usb_linux_v3.3.0_2920.20111123.tar.gz:

```
root@driver/# tar zxvf rtl8188C_8192C_8192D_usb_linux_v3.3.0_2920.20111123.tar.gz
```

3. Selecting Chip Type with make_drv Script (for compound release)

Our driver source release has two types: 1) single release, which can build out driver only for single chip type, and 2) compound release, which can build out drivers for multiple chip types separately.

For compound release driver, you will see make_drv script after you decompress the driver tar ball located in driver folder. Before compiling driver source, executing the make_drv to select the target chip type to compile. For example:

```
root@rtl8188C_8192C_8192D_usb_linux_v3.3.0_2920.20111123# ./make_drv
Please select chip type(1/2):
1) RTL8192cu
2) RTL8192du
#? 1
You have selected RTL8192cu
```

4. Compilation Settings in Makefile

4.1. Adding or Selecting Target Platform

The default target platform is PC-Linux, if you do not want to compile driver for other platforms you can skip this section.

To add or select target platform for compilation, we provide two sections in Makefile: 1) platform selection section and 2) platform setting section. First, you should look at the platform selection section of Makefile:

```
CONFIG_PLATFORM_I386_PC                 =       y
CONFIG_PLATFORM_ANDROID_X86             =       n
CONFIG_PLATFORM_ARM_S3C2K4              =       n
CONFIG_PLATFORM_ARM_PXA2XX              =       n
CONFIG_PLATFORM_ARM_S3C6K4              =       n
CONFIG_PLATFORM_MIPS_RMI                =       n
CONFIG_PLATFORM_RTD2880B                =       n
CONFIG_PLATFORM_MIPS_AR9132             =       n
CONFIG_PLATFORM_MT53XX                  =       n
CONFIG_PLATFORM_RTK_DMP                 =       n
```

The platform selection section consists of entries with ‘CONFIG_PLATFORM_’ prefix. Only one entry is allowed to be set with value ‘y’ and others with ‘n’. The

‘CONFIG_PLATFORM_I386_PC’ is selected by default.

We can select an existing entry or add a new entry for your target platform. For example, to add and select a new entry, ‘CONFIG_PLATFORM_NEW’:

```
CONFIG_PLATFORM_I386_PC        =    n
CONFIG_PLATFORM_NEW            =    y
```

Second, you should create and/or modify the corresponding entry inside platform setting section. For example, adding the following entry in platform setting section for ‘CONFIG_PLATFORM_NEW’ we just add:

```
ifeq ($(CONFIG_PLATFORM_NEW), y)
EXTRA_CFLAGS += -DCONFIG_LITTLE_ENDIAN
ARCH := arm
CROSS_COMPILE := /opt/ new/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
KSRC := /opt /new/kernel
endif
```

4.2. Platform Setting Section in Detail

I EXTRA_CFLAGS

The EXTRA_CFLAGS is usually used to carry some additional settings at compilation time through macro definitions.

Macro	Effect
CONFIG_BIG_ENDIAN	Define some internal data structure as big endian.
CONFIG_LITTLE_ENDIAN	Define some internal data structure as little endian.
CONFIG_MINIMAL_MEMORY_USAGE	For better performance in powerful platform, we allocate large physical continuous memory as TX/RX IO buffers. In some embedded platform, there is chance to fail to allocate memory. Define this macro to prevent this situation.
CONFIG_PLATFORM_ANDROID	Older Android kernel do not has CONFIG_ANDROID defined. Define this macro to force the Android corresponding code inside our driver to be compiled. For newer Android kernel, it has no need to define this macro, otherwise, warning message about redefinition will show up

- **ARCH**

The ARCH is used to specify the architecture of the target platform CPU, such as: arm, mips, i386, etc.

- **CROSS_COMPILE**

The CROSS_COMPILE is used to specify the toolchain prefix used for driver compilation.

- **KSRC**

The KSRC is used to specify the path of kernel source used for driver compilation

- **MODULE_NAME**

Different module name is assigned to drivers for different chips:

Chip type	Default module name
RTL8192CU-series	8192cu
RTL8192CE-series	8192ce
RTL8192DU-series	8192du
RTL8192DE-series	8192de
RTL8723AS-series	8723as
RTL8723AU-series	8723au
RTL8189ES-series	8189es
RTL8188EU-series	8188eu
RTL8723BS-series	8723bs
RTL8723BU-series	8723bu

If you want to change the module name, you can set value of MODULE_NAME here. For example, setting module name as ‘wlan’:

```
ifeq ($(CONFIG_PLATFORM_NEW), y)
EXTRA_CFLAGS += -DCONFIG_LITTLE_ENDIAN
ARCH := arm
CROSS_COMPILE := /opt/ new/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
KSRC := /opt /new/kernel
MODULE_NAME := wlan
endif
```

4.3. Other Compilation Settings

We still have some compilation settings could be applied. For settings and further information about power saving mode, please refer to:

document/HowTo_enable_the_power_saving_functionality.pdf.

If you know what the macro means in the autoconf file, you could modify the configuration by yourself. See the following table for the autoconf file you should modify for a specific chip type:

Chip type	Autoconf file to modify
RTL8192CU-series	autoconf_rtl8192c_usb_linux.h
RTL8192CE-series	autoconf_rtl8192c_pci_linux.h
RTL8192DU-series	autoconf_rtl8192d_usb_linux.h
RTL8192DE-series	autoconf_rtl8192d_pci_linux.h
RTL8723AS-series	autoconf_rtl8723a_sdio_linux.h
RTL8723AU-series	autoconf_rtl8723a_usb_linux.h
RTL8189ES-series	autoconf_rtl8189e_sdio_linux.h
RTL8188EU-series	autoconf_rtl8188e_usb_linux.h
RTL8723BS-series	autoconf_rtl8723b_sdio_linux.h
RTL8723BU-series	autoconf_rtl8723b_usb_linux.h

5. Integrating Driver Source into Linux Kernel Tree

This paragraph is for integrating our driver source into Linux kernel tree and building system. If you have no need to do this, simply skip this paragraph.

For compound release driver source, make_drv should be execute to select chip type for the driver source. Please refer to:

“3. Selecting Chip Type with make_drv Script (for compound release)”.

For different chip types, we have different suggestions for <compile_flag> and <folder_name> to use for the integration process:

Chip type	<compile_flag>	<folder_name>
RTL8192CU-series	CONFIG_RTL8192CU	rtl8192cu
RTL8192CE-series	CONFIG_RTL8192CE	rtl8192du
RTL8192DU-series	CONFIG_RTL8192DU	rtl8192du
RTL8192DE-series	CONFIG_RTL8192DE	rtl8192de
RTL8723AS-series	CONFIG_RTL8723AS	rtl8723as
RTL8723AU-series	CONFIG_RTL8723AU	rtl8723au
RTL8189ES-series	CONFIG_RTL8189ES	rtl8189es
RTL8188EU-series	CONFIG_RTL8188EU	rtl8188eu
RTL8723BS-series	CONFIG_RTL8723BS	rtl8723bs
RTL8723BU-series	CONFIG_RTL8723BU	rtl8723bu

Assuming the driver source is for RTL8192CU-series, to integrate driver source into kernel building system, go through the following steps:

1). Copy the driver source folder into drivers/net/wireless/ and rename it as <folder_name>, rtl8192cu.

2). Add the following line into drivers/net/wireless/Makefile, CONFIG_RTL8192CU is for <compile_flag>, rtl8192cu is for <folder_name>:

```
obj-$(CONFIG_RTL8192CU)    += rtl8192cu/
```

3). Add the following line into drivers/net/wireless/Kconfig, rtl8192cu is for <folder_name>:

```
source "drivers/net/wireless/rtl8192cu/Kconfig"
```

4). Config kernel, for example, with ‘make menuconfig’ command to select ‘y’ or ‘m’ for our driver.

5). Now, you can build kernel with ‘make’ command.

6. Compiling Driver

6.1. Compiling Driver in Driver Source Folder

For compiling driver in the original driver source folder, simply cd into the driver source folder and start build driver with ‘make’ command.

```
root@rtl8188C_8192C_8192D_usb_linux_v3.3.0_2920.20111123# ./make
```

If everything goes well, it will produce a *MODULE_NAME*.ko file. The MODULE_NAME is specified in Makefile. Please refer to:

"MODULE_NAME" in "4.2. Platform Setting Section in Detail".

6.2. Compiling Driver under Kernel Tree

For compiling driver under kernel tree, please refer to:

"5. Integrating Driver Source into Linux Kernel Tree".

7. Driver Installation

If you have compiled Wi-Fi driver as kernel module and produced a .ko file such as 8192cu.ko, you should insert driver module with 'insmod' command:

```
root@rtl8188C_8192C_8192D_usb_linux_v3.3.0_2920.20111123# insmod 8192cu.ko
```

As for driver compiled in kernel, it has no need to do 'insmod' command.