[openwrt开发文档中文版](http://blog.csdn.net/ruiyiin/article/details/24782997)

分类： [openwrt](http://blog.csdn.net/ruiyiin/article/category/1366186)2014-04-30 12:53 1130人阅读 [评论](http://blog.csdn.net/ruiyiin/article/details/24782997#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ruiyiin/article/details/24782997#report)

[openwrt](http://www.csdn.net/tag/openwrt)[路由](http://www.csdn.net/tag/%e8%b7%af%e7%94%b1)

目录[(?)[+]](http://blog.csdn.net/ruiyiin/article/details/24782997)

内容

**1****[路由器](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-20001" \t "_blank)**   
1.1 [开始](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-30001.1" \t "_blank)   
1.1.1 [安装](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-40001.1.1" \t "_blank)   
1.1.2 [初始配置](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-50001.1.2" \t "_blank)   
1.1.3 [安全模式](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-60001.1.3" \t "_blank)   
1.2 [配置OpenWrt](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-70001.2" \t "_blank)   
1.2.1 [网络](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-80001.2.1" \t "_blank)   
1.2.2 [无线](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-120001.2.2" \t "_blank)   
1.3 [高级配置](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-310001.3" \t "_blank)   
1.3.1 [热插拔](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-340001.3.1" \t "_blank)   
1.3.2 [初始化脚本](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-350001.3.2" \t "_blank)   
1.3.3 [网络脚本](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-360001.3.3" \t "_blank)   
**2****[开发需知](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-390002" \t "_blank)**   
2.1 [构建系统](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-400002.1" \t "_blank)   
2.1.1 [构建一个镜像](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-410002.1.1" \t "_blank)   
2.1.2 [创建包](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-460002.1.2" \t "_blank)   
2.1.3 [创建内核模块包](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-470002.1.3" \t "_blank)   
2.1.4 [Conventions](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-480002.1.4" \t "_blank)   
2.1.5 [Troubleshooting](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-490002.1.5" \t "_blank)   
2.1.6 [Using build environments](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-500002.1.6" \t "_blank)   
2.2 [Extra tools](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-510002.2" \t "_blank)   
2.2.1 [Image Builder](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-520002.2.1" \t "_blank)   
2.2.2 [SDK](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-530002.2.2" \t "_blank)   
2.3 [Adding platform support](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-540002.3" \t "_blank)   
2.3.1 [Which Operating System does this device run?](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-550002.3.1" \t "_blank)   
2.3.2 [Finding and using the manufacturer SDK](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-630002.3.2" \t "_blank)   
2.4 [Debugging and debricking](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-710002.4" \t "_blank)   
2.4.1 [Adding a serial port](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-720002.4.1" \t "_blank)   
2.4.2 [JTAG](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-730002.4.2" \t "_blank)   
2.5 [Reporting bugs](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-740002.5" \t "_blank)   
2.5.1 [Using the Trac ticket system](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-750002.5.1" \t "_blank)   
2.6 [Submitting patches](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-780002.6" \t "_blank)   
2.6.1 [How to contribute](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-790002.6.1" \t "_blank)   
2.6.2 [Where to listen and talk](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-800002.6.2" \t "_blank)   
2.6.3 [Patch Submission Process](file:///C:\\Program%20Files\\Zoundry%20Blog%20Writer\\users\\Administrator\\tmp\\entry.tmp.htm" \l "x1-810002.6.3" \t "_blank)

**第一节****路由器**

**1.1****开始**

**1.1.1****安装**

**1.1.2****初始化配置**

**1.1.3****Failsafe mode**

**1.2****配置OpenWrt**

**1.2.1****网络**

Kamikaze 中的网络配置信息储存在 /etc/config/network 文件中，并且各个接口都有自己的配置。每个接口的配置直接关联到一个eth或者wifi接口(eth0, wl0, ..) ，或者桥接到多个接口。像下面这样：

config interface "lan"   
option ifname "eth0"   
option proto "static"   
option ipaddr "192.168.1.1"   
option netmask "255.255.255.0"   
option gateway "192.168.1.254"   
option dns "192.168.1.254"

ifname定义了特殊的Linux接口。如果你想桥接到一个或者多个接口，设置 ifname 为一个接口列表，并添加：

option type "bridge"

可以通过简单的加入VLAN ID号标记来使用VLAN，例如eth0.1。这样就可以了。

这里对eth0.proto有一个简单的静态配置来描述该接口所使用的协议，默认的Image中通常会提供'none' 'static', 'dhcp'和'pppoe'方式。其他方式，可以通过加载包来安装其他协议。

像例子中这样使用'static'方法时，ipaddr和netmask是强制的，gateway和dns是可选的。你可以指定不止一个的DNS server，用空格分开。

DHCP目前只可以设置ipaddr（希望从server请求的IP地址）和hostname（客户端主机名标签），两者都是可选的。PPP协议族接受如下选项：

基于PPP 的协议(pppoe, pptp, ...) 接受下列选项：

* username   
  The PPP username (usually with PAP authentication)
* password   
  The PPP password
* keepalive   
  Ping PPP server（使用LCP）。这个选项定义了重连前的失败ping最大个数。这个ping间隔默认为5，但是可以通过附加"," 来修改keepalive 值。Ping the PPP server (using LCP). The value of this option specifies the maximum number of failed pings before reconnecting. The ping interval defaults to 5, but can be changed by appending "," to the keepalive value
* demand   
  Use Dial on Demand (value specifies the maximum idle time.
* server: (pptp)   
  远程pptp server IP The remote pptp server IP

对于所有类型的协议，都可以通过设置mtu选项来设置MTU。

**设置静态路由****Setting up static routes**

你可以给特定的接口设置静态路由，它将在该接口被配置以后显示出效果。

像下面这样简单的加入：

config route foo   
option interface lan   
option target 1.1.1.0   
option netmask 255.255.255.0   
option gateway 192.168.1.1

route段中的name是可选的，interface，target和gateway选项是强制的。不使用netmask选项将会使该route设置成host route。

The name for the route section is 可选的, the interface, targetand gateway options are mandatory. Leaving out the netmaskoption will turn the route into a host route.

**设置交换（目前仅支持broadcom）****Setting up the switch (currently broadcom only)**

交换设置需要加入'switch'设置段。例如：

The switch configuration is set by adding a 'switch'config section. 例子:

config switch "eth0"   
option vlan0 "1 2 3 4 5\*"   
option vlan1 "0 5"

在Broadcom的硬件上，段名称需要是eth0，所以交换驱动不能识别其他物理交换设备。每个vlan选项都要有个名字vlan，是交换驱动中使用的VLAN序号。这些值可以取如下后缀：

VLANOn Broadcom hardware the section name needs to be eth0, as the switch driver does not detect the switch on any other physical device. Every vlan option needs to have the name vlan where is the VLAN number as used in the switch driver. As value it takes a list of ports with these 可选的 suffixes:

* '\*': 将当前VLAN设置成该端口的默认VLANSet the default VLAN (PVID) of the Port to the current VLAN
* 'u': 强制该端口为非标记的Force the port to be untagged
* 't': 强制该端口为标记的Force the port to be tagged

CPU的端口默认是标记的，其他所有端口为非标记。在Broadcom硬件上，CPU端口永远是5。其他端口可能会因不同的硬件而异。

The CPU port defaults to tagged, all other ports to untagged. On Broadcom hardware the CPU port is always 5. The other ports may vary with different hardware.

例如，如果你希望有3个vlan，3个端口用于交换，1个端口是DMZ（隔离区），另外一个端口是WAN口，可以使用如下配置：

For instance, if you wish to have 3 vlans, like one 3-port switch, 1 port in a DMZ, and another one as your WAN interface, use the following configuration :

config switch "eth0"   
option vlan0 "1 2 3 5\*"   
option vlan1 "0 5"   
option vlan2 "4 5"

交换层将会自动建立3个接口：eth0.0(vlan0), eth0.1(vlan1) and eth0.2(vlan2)。

Three interfaces will be automatically created using this switch layout : eth0.0(vlan0), eth0.1(vlan1) andeth0.2(vlan2). You can then assign those interfaces to a custom network configuration name like lan, wanordmzfor instance.

**Setting up IPv6 connectivity**

OpenWrt supports IPv6 connectivity using PPP, Tunnel brokers or static assignment.

If you use PPP, IPv6 will be setup using IP6CP and there is nothing to configure.

To setup an IPv6 tunnel to a tunnel broker, you can install the 6scriptspackage and edit the/etc/config/6tunnelfile and change the settings accordingly :

config 6tunnel   
option tnlifname 'sixbone'   
option remoteip4 '1.0.0.1'   
option localip4 '1.0.0.2'   
option localip6 '2001::DEAD::BEEF::1'   
option prefix '/64'

* 'tnlifname': Set the interface name of the IPv6 in IPv4 tunnel
* 'remoteip4': IP address of the remote end to establish the 6in4 tunnel. This address is given by the tunnel broker
* 'localip4': IP address of your router to establish the 6in4 tunnel. It will usually match your WAN IP address.
* 'localip6': IPv6 address to setup on your tunnel side This address is given by the tunnel broker
* 'prefix': IPv6 prefix to setup on the LAN.

Using the same package you can also setup an IPv6 bridged connection :

config 6bridge   
option bridge 'br6'

By default the script bridges the WAN interface with the LAN interface and uses ebtables to filter anything that is not IPv6 on the bridge.

IPv6 static addressing is also supported using a similar setup as IPv4 but with the ip6prefixing (when applicable).

config interface "lan"   
option ifname "eth0"   
option proto "static"   
option ip6addr "fe80::200:ff:fe00:0/64"   
option ip6gw "2001::DEAF:BEE:1"

**1.2.2 无线网络****Wireless**

WiFi设置储存在/etc/config/wireless中（目前支持Broadcom, Atheros 和mac80211）。当第一次启动路由器时，将会自动检测你的卡并且创建简单的配置文件。默认情况下 'option network lan' 是被注释掉的。它提供了一个非安全的无线网络共享。

The WiFi settings are configured in the file /etc/config/wireless(currently supported on Broadcom, Atheros and mac80211). When booting the router for the first time it should detect your card and create a sample configuration file. By default 'option network lan' is commented. This prevents unsecured sharing of the network over the wireless interface.

每一个无线驱动都有自己的配置脚本/lib/wifi/driver\_name.sh ，用来处理驱动选项和配置。这个脚本也调用特殊二进制，例如Broadcom的w1c，或者atheros的hostapd 和wpa\_supplicant 。

Each wireless driver has its own configuration script in /lib/wifi/driver\_name.sh which handles driver specific options and configurations. This script is also calling driver specific binaries like wlc for Broadcom, or hostapd and wpa\_supplicant for atheros.

使用不同的架构配置是因为不同的驱动配置。

The reason for using such architecture, is that it abstracts the driver configuration.

**标准的Broadcom无线配置：****Generic Broadcom wireless config:**

config wifi-device "wl0"   
option type "broadcom"   
option channel "5"   
  
config wifi-iface   
option device "wl0"   
# option network lan   
option mode "ap"   
option ssid "OpenWrt"   
option hidden "0"   
option encryption "none"

**标准的Atheros无线配置：****Generic Atheros wireless config:**

config wifi-device "wifi0"   
option type "atheros"   
option channel "5"   
option hwmode "11g"   
  
config wifi-iface   
option device "wifi0"   
# option network lan   
option mode "ap"   
option ssid "OpenWrt"   
option hidden "0"   
option encryption "none"

**标准的mac80211无线配置：****Generic mac80211 wireless config:**

config wifi-device "wifi0"   
option type "mac80211"   
option channel "5"   
  
config wifi-iface   
option device "wlan0"   
# option network lan   
option mode "ap"   
option ssid "OpenWrt"   
option hidden "0"   
option encryption "none"

**标准的多播Atheros无线配置：****Generic multi-radio Atheros wireless config:**

config wifi-device wifi0   
option type atheros   
option channel 1   
  
config wifi-iface   
option device wifi0   
# option network lan   
option mode ap   
option ssid OpenWrt\_private   
option hidden 0   
option encryption none   
  
config wifi-device wifi1   
option type atheros   
option channel 11   
  
config wifi-iface   
option device wifi1   
# option network lan   
option mode ap   
option ssid OpenWrt\_public   
option hidden 1   
option encryption none

这个文件里有两个类型的配置段。wifi-device针对物理wifi接口，wifi-iface配置了一个在wifi-device之上的虚拟接口（需要被驱动支持）。

There are two types of config sections in this file. The 'wifi-device' refers to the physical wifi interface and 'wifi-iface' configures a virtual interface on top of that (if supported by the driver).

无线配置全概括：A full outline of the wireless configuration file with description of each field:

config wifi-device wifi device name   
option type broadcom, atheros, mac80211   
option country us, uk, fr, de, etc.   
option channel 1-14   
option maxassoc 1-128 (broadcom only)   
option distance 1-n   
option hwmode 11b, 11g, 11a, 11bg (atheros, mac80211)   
option rxantenna 0,1,2 (atheros, broadcom)   
option txantenna 0,1,2 (atheros, broadcom)   
option txpower transmission power in dBm   
  
config wifi-iface   
option network the interface you want wifi to bridge with   
option device wifi0, wifi1, wifi2, wifiN   
option mode ap, sta, adhoc, monitor, or wds   
option txpower (deprecated) transmission power in dBm   
option ssid ssid name   
option bssid bssid address   
option encryption none, wep, psk, psk2, wpa, wpa2   
option key encryption key   
option key1 key 1   
option key2 key 2   
option key3 key 3   
option key4 key 4   
option server ip address   
option port port   
option hidden 0,1   
option isolate 0,1

**wifi-device的配置项：****Options for the wifi-device :**

* type   
  该接口使用的驱动。The driver to use for this interface.
* country   
  国家代码用来指名规定设置。The country code used to determine the regulatory settings.
* channel   
  wifi频道（例如1-14，取决于country设置）The wifi channel (e.g. 1-14, depending on your country setting).
* maxassoc   
  可选：最大的可连接客户端。这个特性只在broadcom芯片组上可选。可选的: Maximum number of associated clients. This feature is supported only on the broadcom chipset.
* distance   
  可选的: Distance between the ap and the furthest client in meters. This feature is supported only on the atheros chipset.
* mode   
  The frequency band (b, g, bg, a). This feature is only supported on the atheros chipset.
* diversity   
  可选的: Enable diversity for the Wi-Fi device. This feature is supported only on the atheros chipset.
* rxantenna   
  可选的: Antenna identifier (0, 1 or 2) for reception. This feature is supported by atheros and some broadcom chipsets.
* txantenna   
  可选的: Antenna identifier (0, 1 or 2) for emission. This feature is supported by atheros and some broadcom chipsets.
* txpowerSet the transmission power to be used. The amount is specified in dBm.

**wifi-iface的配置：****Options for the wifi-iface :**

* network   
  选择/etc/config/network中的接口段 Selects the interface section from /etc/config/networkto be used with this interface
* device   
  设置设备名称Set the wifi device name.
* mode   
  可选模式：Operating mode:
  + ap   
    接入点模式Access point mode
  + sta   
    客户端模式Client mode
  + adhoc   
    Ad-Hoc mode
  + monitor   
    监控模式Monitor mode
  + wds   
    WDS点对点连接 WDS point-to-point link
* ssid 设置wifi设备使用的SSID Set the SSID to be used on the wifi device.
* bssid设置BSSID 地址用来设置wds的mac地址。Set the BSSID address to be used for wds to set the mac address of the other wds unit.
* txpower(Deprecated, set in wifi-device) Set the transmission power to be used. The amount is specified in dBm.
* encryption   
  加密设置。可设置为以下值：Encryption setting. Accepts the following values:
  + none
  + wep
  + psk, psk2   
    WPA(2) Pre-shared Key
  + wpa, wpa2   
    WPA(2) RADIUS
* key, key1, key2, key3, key4(wep, wpa and psk)   
  WEP key, WPA key (PSK mode) or the RADIUS shared secret (WPA RADIUS mode)
* server(wpa)   
  The RADIUS server ip address
* port(wpa)   
  The RADIUS server port (defaults to 1812)
* hidden   
  0 broadcasts the ssid; 1 disables broadcasting of the ssid
* isolate   
  可选的: Isolation is a mode usually set on hotspots that limits the clients to communicate only with the AP and not with other wireless clients. 0 disables ap isolation (default); 1 enables ap isolation.

**WDS（ Wireless Distribution System ）是非标准模式，用来互联两个Broadcom设备，而不能连接一个Broadcom和一个Atheros设备。**

**Wireless Distribution System WDS is a non-standard mode which will be working between two Broadcom devices for instance but not between a Broadcom and Atheros device.**

**非加密WDS连接。这个配置例程展示了你如何去设置一个非加密的WD连接。我们假定节点已经配置了BSSID ca:fe:ba:be:00:01 和远程WDS 节点ca:fe:ba:be:00:02 (option bssid field****Unencrypted WDS） connections**

**This configuration 例子 shows you how to setup unencrypted WDS connections. We assume that the peer configured as below as the BSSID ca:fe:ba:be:00:01 and the remote WDS endpoint ca:fe:ba:be:00:02 (option bssid field).**

config wifi-device "wl0"   
option type "broadcom"   
option channel "5"   
  
config wifi-iface   
option device "wl0"   
option network lan   
option mode "ap"   
option ssid "OpenWrt"   
option hidden "0"   
option encryption "none"   
  
config wifi-iface   
option device "wl0"   
option network lan   
option mode wds   
option ssid "OpenWrt WDS"   
option bssid "ca:fe:ba:be:00:02"

**加密的WDS连接。支持加密的WDS连接，支持psk, psk2and psk+psk2模式。下面是使用AES算法的Pre-Shared-Keys 的例子。**

**Encrypted WDS connections It is also possible to encrypt WDS connections. psk, psk2andpsk+psk2modes are supported. Configuration below is an 例子 configuration using Pre-Shared-Keys with AES algorithm.**

config wifi-device wl0   
option type broadcom   
option channel 5   
  
config wifi-iface   
option device "wl0"   
option network lan   
option mode ap   
option ssid "OpenWrt"   
option encryption psk2   
option key "<key< span=""> for clients>"   
  
config wifi-iface   
option device "wl0"   
option network lan   
option mode wds   
option bssid ca:fe:ba:be:00:02   
option ssid "OpenWrt WDS"   
option encryption psk2   
option key "<psk< span=""> for WDS>"

**802.1x configurations OpenWrt supports both 802.1x client and Access Point configurations. 802.1x client is only working with Atheros or mac80211 drivers. Configuration only supports EAP types TLS, TTLS or PEAP.**

**EAP-TLS**

config wifi-iface   
option device "ath0"   
option network lan   
option ssid OpenWrt   
option eap\_type tls   
option ca\_cert "/etc/config/certs/ca.crt"   
option priv\_key "/etc/config/certs/priv.crt"   
option priv\_key\_pwd "PKCS#12 passphrase"

**EAP-PEAP**

config wifi-iface   
option device "ath0"   
option network lan   
option ssid OpenWrt   
option eap\_type peap   
option ca\_cert "/etc/config/certs/ca.crt"   
option auth MSCHAPV2   
option identity username   
option password password

**限制：在选择模式的时候有一些限制。只支持如下的模式组合：****Limitations: There are certain limitations when combining modes. Only the following mode combinations are supported:**

* **Broadcom**:
  + 1x sta, 0-3x ap
  + 1-4x ap
  + 1x adhoc
  + 1x monitor

WDS links can only be used in pure AP mode and cannot use WEP (except when sharing the settings with the master interface, which is done automatically).

* **Atheros**:
  + 1x sta, 0-Nx ap
  + 1-Nx ap
  + 1x adhoc

N is the maximum number of VAPs that the module allows, it defaults to 4, but can be changed by loading the module with the maxvaps=N parameter.

**加入新的驱动配置。因为我们目前只支持Broadcom, Atheros and mac80211三种不同的无线驱动，你可能对加入其它驱动感兴趣，比如Ralink RT2x00, Texas Instruments ACX100/111。**

**Adding a new driver configuration Since we currently only support thread different wireless drivers : Broadcom, Atheros and mac80211, you might be interested in adding support for another driver like Ralink RT2x00, Texas Instruments ACX100/111.**

这些驱动的特定脚本要放在/lib/wifi/.脚本必须包括以下几个方法：

The driver specific script should be placed in /lib/wifi/.shand has to include several functions providing :

* 检测驱动是否存在detection of the driver presence
* 使能/禁用wifi接口enabling/disabling the wifi interface(s)
* 配置读取和设置configuration reading and setting
* 第三方程序的调用(nas, supplicant) third-party programs calling (nas, supplicant)

每个驱动都应该附加到全局变量DRIVERS 中：

Each driver script should append the driver to a global DRIVERS variable :

append DRIVERS "driver name"

**scan\_ ，这个方法将会分析/etc/config/wireless并且保证没有不兼容的配置，例如在ad-hoc模式下隐藏了SSIDS。如果你的设备支持很多配置选项，将是一件非常麻烦的事。 scan\_****This function will parse the /etc/config/wireless and make sure there are no configuration incompatibilities, like enabling hidden SSIDS with ad-hoc mode for instance. This can be more complex if your driver supports a lof of configuration options. It does not change the state of the interface.**

例子:

scan\_dummy() {   
local device="$1"   
  
config\_get vifs "$device" vifs   
for vif in $vifs; do   
# check config consistency for wifi-iface sections   
done   
# check mode combination   
}

**enable\_ ，这个方法将会启动wifi设备，并且可以建立特定的配置文件，例如针对WPA鉴别者或者提供者。**

**enable\_****This function will bring up the wifi device and 可选的ly create application specific configuration files, e.g. for the WPA authenticator or supplicant.**

例子:

enable\_dummy() {   
local device="$1"   
  
config\_get vifs "$device" vifs   
for vif in $vifs; do   
# bring up virtual interface belonging to   
# the wifi-device "$device"   
done   
}

**disable\_ ，这个方法将会关闭wifi设备和它的所有虚拟接口。**

**disable\_****This function will bring down the wifi device and all its virtual interfaces (if supported).**

例子:

disable\_dummy() {   
local device="$1"   
  
# bring down virtual interfaces belonging to   
# "$device" regardless of whether they are   
# configured or not. Don't rely on the vifs   
# variable at this point   
}

**detect\_ ，这个方法寻找驱动可用的接口。为新设备编写的配置模版应该输出到stdout。还必须在建立新模板时，检查是否已经存在属于该接口的配置段。**

**detect\_****This function looks for interfaces that are usable with the driver. Template config sections for new devices should be written to stdout. Must check for already existing config sections belonging to the interfaces before creating new templates.**

例子:

detect\_dummy() {   
[ wifi-device = "$(config\_get dummydev type)" ] && return 0   
cat <<eof< span="">   
config wifi-device dummydev   
option type dummy   
# REMOVE THIS LINE TO ENABLE WIFI:   
option disabled 1   
  
config wifi-iface   
option device dummydev   
option mode ap   
option ssid OpenWrt   
EOF   
}

**1.3 高级配置****Advanced configuration**

**配置文件结构****Structure of the configuration files**

这个配置文件被分为段和选项/值两部分。

The config files are divided into sections and options/values.

每一个段都有一个类型，但是不需要都有名字。每个选项都有一个名字和一个值并且分配到一个段中。

Every section has a type, but does not necessarily have a name. Every option has a name and a value and is assigned to the section it was written under.

语法 Syntax:

config [""] # Section   
option "" # Option

每个参数都必须是一个单独的字符串，并且被格式化为shell方法的参数。

Every parameter needs to be a single string and is formatted exactly like a parameter for a shell function. The same rules for Quoting and special characters also apply, as it is parsed by the shell.

**自定义解析配置文件的脚本****Parsing configuration files in custom scripts**

为了能够加载配置文件，你需要include通用方法：

To be able to load configuration files, you need to include the common functions with:

. /etc/functions.sh

这样你就可以使用config\_load *来加载配置文件。这个函数首先检查的文件名，然后从 /etc/config中调用它。*Then you can use config\_load to load config files. The function first checks for as absolute filename and falls back to loading it from /etc/config (which is the most common way of using it).

如果你希望使用特殊的回调函数来调用段或者选项，在使用config\_load前你需要首先定义你的shell方法（在including /etc/functions.sh之后），如下。

If you want to use special callbacks for sections and/or options, you need to define the following shell functions before running config\_load(after including /etc/functions.sh):

config\_cb() {   
local type="$1"   
local name="$2"   
# commands to be run for every section   
}   
  
option\_cb() {   
# commands to be run for every option   
}

你还可以从You can also alter option\_cbfrom config\_cbbased on the section type. This allows you to process every single config section based on its type individually.

config\_cb在每个新段开始后都会运行（在选项被处理之前）。可以通过CONFIG\_SECTION变量查看最后的配置段。在config\_load完成之后也会调用config\_cb（不需要新的配置段）。这就可以再所有选项被处理之前或者之后处理配置段。

config\_cbis run every time a new section starts (before options are being processed). You can access the last section through the CONFIG\_SECTIONvariable. Also an extra call to config\_cb(without a new section) is generated after config\_loadis done. That allows you to process sections both before and after all options were processed.

Another way of iterating on config sections is using the config\_foreach command.

语法Syntax:

config\_foreach <function< span="">name> [] []

This command will run the supplied function for every single config section in the currently loaded config. The section name will be passed to the function as argument 1. If the section type is added to the command line, the function will only be called for sections of the given type.

You can access already processed options with the config\_getcommand Syntax:

# print the value of the option   
config\_get  
  
# store the value inside the variable   
config\_get

In busybox ash the three-option config\_getis faster, because it does not result in an extra fork, so it is the preferred way.

Additionally you can also modify or add options to sections by using the config\_set command.

Syntax:

config\_set

If a config section is unnamed, an automatically generated name will be assigned internally, e.g. cfg1, cfg2, ...

While it is possible, using unnamed sections through these autogenerated names is strongly discouraged. Use callbacks orconfig\_foreachinstead.

**1.3.1****热插拔**

**1.3.2 初始化脚本****Init scripts**

因为OpenWRT使用它自己的初始化脚本系统，所有其他初始化脚本必须安装在/etc/init.d/ *name，使用/etc/rc.common作为封装。*

Because OpenWrt uses its own init script system, all init scripts must be installed as /etc/init.d/ *name*use/etc/rc.commonas a wrapper.

例子: /etc/init.d/httpd

#!/bin/sh /etc/rc.common   
# Copyright (C) 2006 OpenWrt.org   
  
START=50   
start() {   
[ -d /www ] && httpd -p 80 -h /www -r OpenWrt   
}   
  
stop() {   
killall httpd   
}

正如你所看到的，这个脚本并不实际分析命令行参数。这些都是/etc/rc.common来完成的。

as you can see, the script does not actually parse the command line arguments itself. This is done by the wrapper script/etc/rc.common.

start()and stop()是基本的方法，几乎所有的初始化脚本都需要提供。start()将会在用户执行/etc/init.d/httpd start或者在系统启动的时候（如果该脚本使能了并且没有）执行。

start()and stop()are the basic functions, which almost any init script should provide. start()is called when the user runs/etc/init.d/httpd startor (if the script is enabled and does not override this behavior) at system boot time.

使能和禁用初始化脚本是通过/etc/init.d/ *name* enableor /etc/init.d/ *name* disable。实际上是在/etc/rc.d中建立或撤销连接，它在启动的时候由/etc/init.d/rcS来处理。

Enabling and disabling init scripts is done by running /etc/init.d/ *name* enableor /etc/init.d/ *name* disable. This creates or removes symbolic links to the init script in /etc/rc.d, which is processed by /etc/init.d/rcSat boot time.

这些脚本的默认启动顺序在初始化脚本的START变量中。改变它需要重新该脚本的运行/etc/init.d/ *name*enable 。

The order in which these scripts are run is defined in the variable STARTin the init script. Changing it requires running/etc/init.d/ *name* enable again.

你还可以重写这些标准的初始化脚本函数：You can also override these standard init script functions:

* boot()   
  Commands to be run at boot time. Defaults to start()
* restart()   
  Restart your service. Defaults to stop(); start()
* reload()   
  Reload the configuration files for your service. Defaults to restart()

你也可以通过建立适当的函数并且在EXTRA\_COMMANDS变量里声明他们来自定义命令。帮助文档加在EXTRA\_HELP里。

You can also add custom commands by creating the appropriate functions and referencing them in theEXTRA\_COMMANDSvariable. Helptext is added in EXTRA\_HELP.

例子:

status() {   
# print the status info   
}   
  
EXTRA\_COMMANDS="status"   
EXTRA\_HELP=" status Print the status of the service"

**1.3.3 网络脚本****Network scripts**

**使用网络脚本****Using the network scripts**

为了能够访问网络功能，你需要通过执行如下命令来include一些必须得shell脚本。

To be able to access the network functions, you need to include the necessary shell scripts by running:

. /etc/functions.sh # common functions   
include /lib/network # include /lib/network/\*.sh   
scan\_interfaces # 读取并且分析网络配置

一些协议，如PPP可能在运行的时候改变接口名称（例如eth0=> ppp0for PPPoE）。这就是为什么你要运行scan\_interfaces，而不仅仅是从配置中直接读取变量。在运行scan\_interfaces之后，ifname选项将始终包含有效地接口名称（用来IP通信），并且如果物理设备名字和他不同，物理设备名称将被储存在device选项中。这就意味着在scan\_interfaces之后执行config\_get lan ifname可能得到与反序执行不同的返回值。

Some protocols, such as PPP might change the configured interface names at run time (e.g. eth0=> ppp0for PPPoE). That's why you have to run scan\_interfacesinstead of reading the values from the config directly. After running scan\_interfaces, the'ifname'option will always contain the effective interface name (which is used for IP traffic) and if the physical device name differs from it, it will be stored in the 'device'option. That means that running config\_get lan ifnameafterscan\_interfacesmight not return the same result as running it before.

在执行scan\_interfaces之后，如下的方法是有效的：

After running scan\_interfaces, the following functions are available:

* find\_config *interface*   
  寻找一个带有特定网络接口的配置项looks for a network configuration that includes the specified network interface.
* setup\_interface *interface [config] [protocol]*   
  将会配置指定的接口，也可以重写网络配置名称或者它使用的协议。will set up the specified interface, 可选的ly overriding the network configuration name or the protocol that it uses.

**编写协议句柄****Writing protocol handlers**

你可以加入自己的协议处理句柄，通过在/lib/network下增加脚本。提供了如下的两个脚本函数：You can add custom protocol handlers by adding shell scripts to /lib/network. They provide the following two shell functions:

scan\_() {   
local config="$1"   
# change the interface names if necessary   
}   
  
setup\_interface\_() {   
local interface="$1"   
local config="$2"   
# set up the interface   
}

scan\_ *protocolname是可选的并且只在你的协议需要使用一个自定义设备时。例如管道或者一个ppp设备。*

scan\_ *protocolname*is 可选的 and only necessary if your protocol uses a custom device, e.g. a tunnel or a PPP device.

**第二节****Development issues**

**2.1 构造系统****The build system**

开始开发嵌入式设备的一个最大挑战就是，你无法只安装一个 linux拷贝就想要能够编译一个固件。即使你真的记得安装一个编译链和所有需要的开发工具，你还是没有为产出一个固件image而做些基本配置。嵌入式 设备是一个全新的硬件平台，大多数时候这些硬件和你的开发平台不一致，所以在交叉编译这一步你需要制作一个新的编译器来编译你的嵌入式平台上的代码。并且 使用它来编译一个基本的linux发行版来在你的设备上运行。

One of the biggest challenges to getting started with embedded devices is that you cannot just install a copy of Linux and expect to be able to compile a firmware. Even if you did remember to install a compiler and every development tool offered, you still would not have the basic set of tools needed to produce a firmware image. The embedded device represents an entirely new hardware platform, which is most of the time incompatible with the hardware on your development machine, so in a process called cross compiling you need to produce a new compiler capable of generating code for your embedded platform, and then use it to compile a basic Linux distribution to run on your device.

建立交叉编译链的过程可能是很棘手的，它不是一件尝试一下就搞定的事，因为这里面有很多诡异和黑暗魔法的集合。

The process of creating a cross compiler can be tricky, it is not something that is regularly attempted and so there is a certain amount of mystery and black magic associated with it. In many cases when you are dealing with embedded devices you will be provided with a binary copy of a compiler and basic libraries rather than instructions for creating your own - it is a time saving step but at the same time often means you will be using a rather dated set of tools. Likewise, it is also common to be provided with a patched copy of the Linux kernel from the board or chip vendor, but this is also dated and it can be difficult to spot exactly what has been modified to make the kernel run on the embedded platform.

**2.1.1 构造一个镜像****Building an image**

OpenWrt有很多方法来构造一个固件；下载，打补丁并 且从头开始编译每一个东西，包括交叉编译链。为了简化过程，OpenWrt不包含任何的可执行程序甚至是源码，它是一个自动下载源码的系统，针对给定平台 打补丁并且正确编译他们。意思就是：通过改变模版，就能改变过程中的任何一步。

OpenWrt takes a different approach to building a firmware; downloading, patching and compiling everything from scratch, including the cross compiler. To put it in simpler terms, OpenWrt does not contain any executables or even sources, it is an automated system for downloading the sources, patching them to work with the given platform and compiling them correctly for that platform. What this means is that just by changing the template, you can change any step in the process.

举个例子，如果新的内核发布了，对一个Makefile的 简单修改就能下载最新的内核，对它打上特定嵌入式平台的补丁并且建立一个新的固件镜像--there is no work to be done trying to track down an unmodified copy of the existing kernel to see what changes had been made，补丁已经提供了并且程序也都完全正确了。这些不仅仅针对内核，OpenWrt中的所有东西都实现了--这个概念可以让你理解OpenWrt是如 何保持最新的编译链，最新的内核，和最新的应用程序的。

As an 例子, if a new kernel is released, a simple change to one of the Makefiles will download the latest kernel, patch it to run on the embedded platform and produce a new firmware image - there is no work to be done trying to track down an unmodified copy of the existing kernel to see what changes had been made, the patches are already provided and the process ends up almost completely transparent. This does not just apply to the kernel, but to anything included with OpenWrt - It is this one simple understated concept which is what allows OpenWrt to stay on the bleeding edge with the latest compilers, latest kernels and latest applications.

好了，让我们看看OpenWrt是如何工作的。

So let's take a look at OpenWrt and see how this all works.

**下载OpenWrt****Download OpenWrt**

这里提到的是OpenWrt的一个分支"Kamikaze"，可以通过使用下面的命令下载它。

This article refers to the "Kamikaze" branch of OpenWrt, which can be downloaded via subversion using the following command:

$ svn checkout https://svn.openwrt.org/openwrt/trunk kamikaze

另外，有一个trac界面，可以用来查看svn的注释和浏览代码。

Additionally, there is a trac interface on <https://dev.openwrt.org/> which can be used to monitor svn commits and browse the source repository.

**目录结构****The directory structure**

有四个关键的目录 There are four key directories in the base:

* tools
* toolchain
* package
* target

tools和toolchain

tool和toolchain里包括了用来建立固件镜像的基本工具，包括编译器和C库。将会产生三个新文件夹，build\_dir/host是一个用来存放与目标无关的工具的临时目录，build\_dir/toolchain- \*用来建立针对不同体系结构的目录，staging\_dir/toolchain- \*是得到的编译链的安装位置。你不需要修改这个编译链目录，除非你确定要加入一个新版本的组件。

toolsand toolchainrefer to common tools which will be used to build the firmware image, the compiler, and the C library. The result of this is three new directories, build\_dir/host, which is a temporary directory for building the target independent tools, build\_dir/toolchain- \*which is used for building the toolchain for a specific architecture, andstaging\_dir/toolchain- \*where the resulting toolchain is installed. You will not need to do anything with the toolchain directory unless you intend to add a new version of one of the components above.

* build\_dir/host
* build\_dir/toolchain- \*

package 就像它的名字一样。在OpenWrt里面几乎所有组件都是.ipk，可以在需要的时候加入到固件并提供新的特性，或者在不需要的时候移除来节省空间。请注意这些包是在主线之外维护的，你可以通过subversion获取系统来得到这些包：

packageis for exactly that - packages. In an OpenWrt firmware, almost everything is an .ipk, a software package which can be added to the firmware to provide new features or removed to save space. Note that packages are also maintained outside of the main trunk and can be obtained from subversion using the package feeds system:

$ ./scripts/feeds update

这些包可以用来扩展系统功能，并且需要被连接进主代码树。一旦你这样做了，这些包将会在菜单里出现他们的配置。对于kamikaze 你需要这样做：

Those packages can be used to extend the functionality of the build system and need to be symlinked into the main trunk. Once you do that, the packages will show up in the menu for configuration. From kamikaze you would do something like this:

$ ./scripts/feeds search nmap   
Search results in feed 'packages':   
nmap Network exploration and/or security auditing utility   
  
$ ./scripts/feeds install nmap

如果要安装所有的包，可以按照下面的命令执行：To include all packages, issue the following command:

$ make package/symlinks

target是关于嵌入式平台的。它包括了一些针对特殊平台的项目。应该重点关注的是"target/linux"目录，它按照平台*分开，并且包含了针对内核，配置等的补丁。还有"target/image"（译者注：现在为"target/imagebuilder"）目录描述如何对特定平台打包固件。*

targetrefers to the embedded platform, this contains items which are specific to a specific embedded platform. Of particular interest here is the "target/linux" directory which is broken down by platform and contains the patches to the kernel, profile config, for a particular platform. There's also the "target/image" directory which describes how to package a firmware for a specific platform.

target 和package 步骤会用到 "build\_dir/ "目录作为编译时的临时目录。另外，toolchain, target or package 下载的任何东西都将会放到"dl" 下。

Both the target and package steps will use the directory "build\_dir/ " as a temporary directory for compiling. Additionally, anything downloaded by the toolchain, target or package steps will be placed in the "dl" directory.

* build\_dir/
* dl

**构建OpenWrt****Building OpenWrt**

虽然大多数时候OpenWrt构建环境是被开发者使用，但是它也简单到终端用户可以建立他（或她）自己的定制固件。

While the OpenWrt build environment was intended mostly for developers, it also has to be simple enough that an inexperienced end user can easily build his or her own customized firmware.

执行"make menuconfig"将会打开OpenWrt的配置屏幕，通过这个菜单你可以选择你的目标平台，你要使用的编译器版本和你要安装到固件镜像里的包。请注意，它还会检查你时候有正确运行他们的基本依赖。如果没有，你需要安装一些工具在你的本机上。

Running the command "make menuconfig" will bring up OpenWrt's configuration menu screen, through this menu you can select which platform you're targeting, which versions of the toolchain you want to use to build and what packages you want to install into the firmware image. Note that it will also check to make sure you have the basic dependencies for it to run correctly. If that fails, you will need to install some more tools in your local environment before you can begin.

类似于linux内核，几乎所有的选项都有三个选择y/m/n：

Similar to the linux kernel config, almost every option has three choices, y/m/nwhich are represented as follows:

* <\*>(pressing y)   
  This will be included in the firmware image
* (pressing m)   
  This will be compiled but not included (for later install)
* < >(pressing n)   
  This will not be compiled

在你完成了这些菜单选择之后，退出，并在有提示的时候保存你的配置。

After you've finished with the menu configuration, exit and when prompted, save your configuration changes.

如果你希望，你也可以针对指定目标系统修改内核配置。运行"make kernel\_menuconfig"，构建系统就会解包内核源码（如果需要），执行内核里的menuconfig，并且拷贝内核配置到target/linux/ /config，以防在"make clean"时被清除。

If you want, you can also modify the kernel config for the selected target system. simply run "make kernel\_menuconfig" and the build system will unpack the kernel sources (if necessary), run menuconfig inside of the kernel tree, and then copy the kernel config to target/linux/ /configso that it is preserved over "make clean" calls.

键入 "make".可以开始编译固件。默认情况下，OpenWrt只显示高级别的编译过程，而不是每个命令。

To begin compiling the firmware, type "make". By default OpenWrt will only display a high level overview of the compile process and not each individual command.

**例子:**

make[2] toolchain/install   
make[3] -C toolchain install   
make[2] target/compile   
make[3] -C target compile   
make[4] -C target/utils prepare   
  
[...]

这个make过程可以很容易的监测编译实际进行到哪一步了，并且组织了编译时输出的很多噪音。如果想看到所有的输出，可以执行"make V=99"。

This makes it easier to monitor which step it's actually compiling and reduces the amount of noise caused by the compile output. To see the full output, run the command "make V=99".

在编译的过程中，buildroot 将会下载所有的源代码到"dl"目录，并开始打补丁，在"build\_dir/ " 目录里编译他们。完成以后，得到的固件将在bin目录中，包将在"bin/packages"目录中。

During the build process, buildroot will download all sources to the "dl" directory and will start patching and compiling them in the "build\_dir/ " directory. When finished, the resulting firmware will be in the "bin" directory and packages will be in the "bin/packages" directory.

**2.1.2 创建包****Creating packages**

OpenWrt的模版系统的其中一项功能就是，我们尝试着让移植软件到OpenWrt非常的容易。你可以在标准的package目录下找到两个东西：

One of the things that we've attempted to do with OpenWrt's template system is make it incredibly easy to port software to OpenWrt. If you look at a typical package directory in OpenWrt you'll find two things:

* package/ /Makefile
* package/ /patches
* package/ /files

patches目录是可选的，并通常包括bug修改或者执行尺寸方面的优化。包的makefile是很关键的项目，它提供了必须下载和编译的包。

The patches directory is 可选的 and typically contains bug fixes or optimizations to reduce the size of the executable. The package makefile is the important item, provides the steps actually needed to download and compile the package.

files目录也是可选的，通常包括包的特殊启动脚本或者默认配置文件，这些文件在OpenWrt之外也可以使用。

The files directory is also and typicall contains package specific startup scripts or default configuration files that can be used out of the box with OpenWrt.

看一个包里面的makefile，你很难认出他是一个makefile来。Through what can only be described as blatant disregard and abuse of the traditional make format, the makefile has been transformed into an object oriented template which simplifies the entire ordeal.

Looking at one of the package makefiles, you'd hardly recognize it as a makefile. Through what can only be described as blatant disregard and abuse of the traditional make format, the makefile has been transformed into an object oriented template which simplifies the entire ordeal.

例子, is package/bridge/Makefile:

1 # $Id: Makefile 5624 2006-11-23 00:29:07Z nbd $   
2   
3 include $(TOPDIR)/rules.mk   
4   
5 PKG\_NAME:=bridge   
6 PKG\_VERSION:=1.0.6   
7 PKG\_RELEASE:=1   
8   
9 PKG\_SOURCE:=bridge-utils-$(PKG\_VERSION).tar.gz   
10 PKG\_SOURCE\_URL:=@SF/bridge   
11 PKG\_MD5SUM:=9b7dc52656f5cbec846a7ba3299f73bd   
12 PKG\_CAT:=zcat   
13   
14 PKG\_BUILD\_DIR:=$(BUILD\_DIR)/bridge-utils-$(PKG\_VERSION)   
15   
16 include $(INCLUDE\_DIR)/package.mk   
17   
18 define Package/bridge   
19 SECTION:=net   
20 CATEGORY:=Base system   
21 TITLE:=Ethernet bridging configuration utility   
22 URL:=http://bridge.sourceforge.net/   
23 endef   
24   
25 define Package/bridge/description   
26 Manage ethernet bridging:   
27 a way to connect networks together to form a larger network.   
28 endef   
29   
30 define Build/Configure   
31 $(call Build/Configure/Default, /   
32 --with-linux-headers="$(LINUX\_DIR)" /   
33 )   
34 endef   
35   
36 define Package/bridge/install   
37 $(INSTALL\_DIR) $(1)/usr/sbin   
38 $(INSTALL\_BIN) $(PKG\_BUILD\_DIR)/brctl/brctl $(1)/usr/sbin/   
39 endef   
40   
41 $(eval $(call BuildPackage,bridge))

正如你所看到的，不需要做太多的工作。每件事都被隐藏在其他的makefile里并且只在这里抽象为几个特定的变量。

As you can see, there's not much work to be done; everything is hidden in other makefiles and abstracted to the point where you only need to specify a few variables.

* PKG\_NAME   
  包的名字，通过menuconfig和ipkg看到。The name of the package, as seen via menuconfig and ipkg
* PKG\_VERSION   
  版本号The upstream version number that we are downloading
* PKG\_RELEASE   
  Makefile的版本The version of this package Makefile
* PKG\_SOURCE   
  源代码的文件名The filename of the original sources
* PKG\_SOURCE\_URL   
  下载地址，可以使用多个下载源，他们之间使用回车分隔Where to download the sources from (no trailing slash), you can add multiple download sources by separating them with a and a carriage return.
* PKG\_MD5SUM   
  用来检验下载程序正确性的校验和A checksum to validate the download
* PKG\_CAT   
  如何解压代码How to decompress the sources (zcat, bzcat, unzip)
* PKG\_BUILD\_DIR   
  在哪编译这个包Where to compile the package

PKG\_开头的这些变量定义了道哪里去下载这些包，@SF是一个特殊关键字，意思是从sourceforge下载。还有其他的关键字，比如@GNU是在GNU的源码发布处下载。如果所有的下载源都失败，将会使用OpenWrt的镜像网站。

The PKG\_\*variables define where to download the package from; @SFis a special keyword for downloading packages from sourceforge. There is also another keyword of @GNUfor grabbing GNU source releases. If any of the above mentionned download source fails, the OpenWrt mirrors will be used as source.

md5sum（目前）用来校验包的正确性，并且PKG\_BUILD\_DIR定义了在源码被解压到$(BUILD\_DIR)后到那里去找包。

The md5sum (if present) is used to verify the package was downloaded correctly and PKG\_BUILD\_DIRdefines where to find the package after the sources are uncompressed into $(BUILD\_DIR).

在文件的底部是最美妙的地方，"BuildPackage"是在前面的段落里被定义的一个宏。"BuildPackage"只接受一个参数--要建立的包名字，在这里例子里是"bridge"。其他的信息在默认块里面定义。This is a way of providing a level of verbosity, it's inherently clear what the contents of the descriptiontemplate in Package/bridgeis, which wouldn't be the case if we passed this information directly as the Nth argument to BuildPackage.

At the bottom of the file is where the real magic happens, "BuildPackage" is a macro set up by the earlier include statements. BuildPackage only takes one argument directly - the name of the package to be built, in this case "bridge". All other information is taken from the define blocks. This is a way of providing a level of verbosity, it's inherently clear what the contents of the descriptiontemplate in Package/bridgeis, which wouldn't be the case if we passed this information directly as the Nth argument to BuildPackage.

BuildPackage 使用如下的定义：

Package/ **:**

*匹配传递给buildroot的参数，它描述了包、menuconfig 、和ipkg 的入口。和Package/ 相同，你可以定义如下的变量：*   
matches the argument passed to buildroot, this describes the package the menuconfig and ipkg entries. WithinPackage/ you can define the following variables:

* SECTION   
  The type of package (currently unused)
* CATEGORY   
  Which menu it appears in menuconfig: Network, Sound, Utilities, Multimedia ...
* TITLE   
  A short description of the package
* URL   
  Where to find the original software
* MAINTAINER(可选的)   
  Who to contact concerning the package
* DEPENDS(可选的)   
  Which packages must be built/installed before this package. To reference a dependency defined in the same Makefile, use . If defined as an external package, use *+*. For a kernel version dependency use:*@LINUX\_2\_*
* BUILDONLY(可选的)   
  Set this option to 1 if you do NOT want your package to appear in menuconfig. This is useful for packages which are only used as build dependencies.

Package//conffiles **(可选的):**   
这个安装包的配置文件，每行一个文件。A list of config files installed by this package, one file per line.

Build/Prepare **(可选的):**   
A set of commands to unpack and patch the sources. You may safely leave this undefined.

Build/Configure **(可选的):**   
如果源码不需要配置或者使用一般的配置脚本，你可以不定义它。否则在这里写入你自己的命令或者使用"$(callBuild/Configure/Default, )"，像上面这样传递更多的参数给标准配置脚本。参数的第一个列表将会传递给配置脚本，形如-arg 1 -arg 2。参数的第二个列表定义了在执行配置脚本之前应该定义的变量，如autoconf or compiler的变量。

You can leave this undefined if the source doesn't use configure or has a normal config script, otherwise you can put your own commands here or use "$(call Build/Configure/Default, )" as above to pass in additional arguments for a standard configure script. The first list of arguments will be passed to the configure script like that: -arg 1 -arg 2. The second list contains arguments that should be defined before running the configure script such as autoconf or compiler specific variables.

为了能够更容易的修改配置命令行，你可以扩展甚至重写下面的变量：

To make it easier to modify the configure command line, you can either extend or completely override the following variables:

* CONFIGURE\_ARGS   
  Contains all command line arguments (format: -arg 1 -arg 2)
* CONFIGURE\_VARS   
  Contains all environment variables that are passed to ./configure (format: NAME="value")

Build/Compile **(可选的):**   
How to compile the source; in most cases you should leave this undefined.

As with Build/Configurethere are two variables that allow you to override the make command line environment variables and flags:

* MAKE\_FLAGS   
  Contains all command line arguments (typically variable overrides like NAME="value"
* MAKE\_VARS   
  Contains all environment variables that are passed to the make command

Build/InstallDev **(可选的):**   
If your package provides a library that needs to be made available to other packages, you can use theBuild/InstallDevtemplate to copy it into the staging directory which is used to collect all files that other packages might depend on at build time. When it is called by the build system, two parameters are passed to it. $(1)points to the regular staging dir, typically staging\_dir/ *ARCH*, while $(2)points to staging\_dir/host. The host staging dir is only used for binaries, which are to be executed or linked against on the host and itsbin/subdirectory is included in the PATHwhich is passed down to the build system processes. Please use$(1)and $(2)here instead of the build system variables $(STAGING\_DIR)and $(STAGING\_DIR\_HOST), because the build system behavior when staging libraries might change in the future to include automatic uninstallation.

Package//install **:**   
这个设置用来将文件从编译好的源码中拷贝到ipkg里，这个目录在$(1)里描述。注意有4个预定义的安装宏。

A set of commands to copy files out of the compiled source and into the ipkg which is represented by the$(1)directory. Note that there are currently 4 defined install macros:

* INSTALL\_DIR   
  install -d -m0755
* INSTALL\_BIN   
  install -m0755
* INSTALL\_DATA   
  install -m0644
* INSTALL\_CONF   
  install -m0600

The reason that some of the defines are prefixed by "Package/ " and others are simply "Build" is because of the possibility of generating multiple packages from a single source. OpenWrt works under the assumption of one source per package Makefile, but you can split that source into as many packages as desired. Since you only need to compile the sources once, there's one global set of "Build" defines, but you can add as many "Package/" defines as you want by adding extra calls to BuildPackage- see the dropbear package for an 例子.

在你建立的你自己的package/ /Makefile之后，在你下次运行"make menuconfig" 的时候新的包将会自动在menu里面显示，并且如果被选定在下次运行make时也会自动被建立。

After you have created your package/ /Makefile, the new package will automatically show in the menu the next time you run "make menuconfig" and if selected will be built automatically the next time "make" is run.

**2.1.3 建立内核模块包****Creating kernel modules packages**

OpenWrt分发版针对两种内核模块都有很好的解决方法，不管是在内核开发主线上的，还是其他一些可用的独立项目。之后我们将会看到一个两者都可用的基本的模版。

The OpenWrt distribution makes the distinction between two kind of kernel modules, those coming along with the mainline kernel, and the others available as a separate project. We will see later that a common template is used for both of them.

对于内核模块来讲，它是内核代码开发主线上的一部分，makefiles在*package/kernel/modules/\*.mk 并且他们在"Kernel modules" 下显示。*

For kernel modules that are part of the mainline kernel source, the makefiles are located in*package/kernel/modules/\*.mk*and they appear under the section "Kernel modules"

对于外部内核模块来说，你可以像是加入软件包一样将他们添加进构造系统中，通过在包的makefile中定义KernelPackage 段。

For external kernel modules, you can add them to the build system just like if they were software packages by defining a KernelPackage section in the package makefile.

这是一个I2C子系统内核模块的makefile例子：

Here for instance the Makefile for the I2C subsytem kernel modules :

1 # $Id $   
2   
3 I2CMENU:=I2C Bus   
4   
5 define KernelPackage/i2c-core   
6 TITLE:=I2C support   
7 DESCRIPTION:=Kernel modules for i2c support   
8 SUBMENU:=$(I2CMENU)   
9 KCONFIG:=CONFIG\_I2C\_CORE CONFIG\_I2C\_DEV   
10 FILES:=$(MODULES\_DIR)/kernel/drivers/i2c/\*.$(LINUX\_KMOD\_SUFFIX)   
11 AUTOLOAD:=$(call AutoLoad,50,i2c-core i2c-dev)   
12 endef   
13 $(eval $(call KernelPackage,i2c-core))

为了在menuconfig下使用一个通用的描述，你可能要定义一个*MENU*variable 在内核模块makefile的顶部。

To group kernel modules under a common description in menuconfig, you might want to define a*MENU*variable on top of the kernel modules makefile.

* TITLE   
  The name of the module as seen via menuconfig
* DESCRIPTION   
  The description as seen via help in menuconfig
* SUBMENU   
  The sub menu under which this package will be seen
* KCONFIG   
  Kernel configuration option dependency. For external modules, remove it.
* FILES   
  Files you want to inlude to this kernel module package, separate with spaces.
* AUTOLOAD   
  Modules that will be loaded automatically on boot, the order you write them is the order they would be loaded.

在你建立好package/kernel/modules/ .mk之后，新的内核模块包将会自动出现菜单中，在你下次执行"make menuconfig"的时候，并且如果被选定在下次运行make时也会自动被建立。

After you have created your package/kernel/modules/ .mk, the new kernel modules package will automatically show in the menu under "Kernel modules" next time you run "make menuconfig" and if selected will be built automatically the next time "make" is run.

**2.1.4****Conventions**

这里有针对包的两个约定：

There are a couple conventions to follow regarding packages:

* files
  1. 配置文件要遵循如下configuration files follow the convention   
     .conf
  2. 初始化文件要遵循如下init files follow the convention   
     .init
* patches
  1. patches are numerically prefixed and named related to what they do

**2.1.5 故障解决****Troubleshooting**

如果你发现你的包没有在menuconfig中显示出来，尝试如下的命令来看看是否得到了正确的信息：

If you find your package doesn't show up in menuconfig, try the following command to see if you get the correct description:

TOPDIR=$PWD make -C package/ DUMP=1 V=99

如果你在编译你的包时遇到了麻烦，这有几个简便的方法可以使用。你可以执行下面命令中的一个，而不用等待编译你的包。

If you're just having trouble getting your package to compile, there's a few shortcuts you can take. Instead of waiting for make to get to your package, you can run one of the following:

* make package/ /clean V=99
* make package/ /install V=99

另一个很不错的技巧是，如果源码在build\_dir/ *下 比package目录新，, it won't clobber it by unpacking the sources again. If you were working on a patch you could simply edit the sources under thebuild\_dir/ / directory and run the install command above, when satisfied, copy the patched sources elsewhere and diff them with the unpatched sources. A warning though - if you go modify anything under package/ it will remove the old sources and unpack a fresh copy.*

Another nice trick is that if the source directory under build\_dir/ is newer than the package directory, it won't clobber it by unpacking the sources again. If you were working on a patch you could simply edit the sources under thebuild\_dir/ / directory and run the install command above, when satisfied, copy the patched sources elsewhere and diff them with the unpatched sources. A warning though - if you go modify anything underpackage/ it will remove the old sources and unpack a fresh copy.

其他有用的东东:Other useful targets include:

* make package/ /prepare V=99
* make package/ /compile V=99
* make package/ /configure V=99

**2.1.6 使用构建环境****Using build environments**

OpenWrt可以使用多个配置OpenWrt provides a means of building images for multiple configurations which can use multiple targets in one single checkout.这些环境储存了一个make menuconfig产生的.config和./files目录下的内容。./scripts/env用来管理这些环境变量，它使用git（需要在你的系统上安装）作为后端的版本管理工具。

OpenWrt provides a means of building images for multiple configurations which can use multiple targets in one single checkout. These *environments*store a copy of the .config file generated by make menuconfigand the contents of the ./filesfolder. The script ./scripts/envis used to manage these environments, it usesgit(which needs to be installed on your system) as backend for version control.

命令 The command

./scripts/env help

提供一个带有短注释的命令清单。produces a short help text with a list of commands.

想建立一个叫做current的新的环境变量，可以执行如下的命令

To create a new environment named current, run the following command

./scripts/env new current

这将会移动你的.config文件和./files（如果存在）目录到/env/子目录并且在基本目录下建立连接。

This will move your .configfile and ./files(if it exists) to the env/subdirectory and create symlinks in the base folder.

在执行menuconfig或者改变files/目录下的东西后，你的当前状态将会和以前存储的状态不同。想要显示这些改变，可以执行：

After running make menuconfig or changing things in files/, your current state will differ from what has been saved before. To show these changes, use:

./scripts/env diff

If you want to save these changes, run:

./scripts/env save

If you want to revert your changes to the previously saved copy, run:

./scripts/env revert

如果需要，你可以通过new命令建立一个第二环境。它将会询问你是clone当前环境（例如监视改变）还是你要开始一个全新的版本（例如针对一个新的目标）。

If you want, you can now create a second environment using the newcommand. It will ask you whether you want to make it a clone of the current environment (e.g. for minor changes) or if you want to start with a clean version (e.g. for selecting a new target).

To switch to a different environment (e.g. test1), use:

./scripts/env switch test1

To rename the current branch to a new name (e.g. test2), use:

./scripts/env rename test2

If you want to get rid of environment switching and keep everything in the base directory again, use:

./scripts/env clear

**2.2 外部工具****Extra tools**

**2.2.1 镜像构建器****Image Builder**

**2.2.2****SDK**

**2.3 增加平台支持****Adding platform support**

Linux is now one of the most widespread operating system for embedded devices due to its openess as well as the wide variety of platforms it can run on. Many manufacturer actually use it in firmware you can find on many devices: DVB-T decoders, routers, print servers, DVD players ... Most of the time the stock firmware is not really open to the consumer, even if it uses open source software.

You might be interested in running a Linux based firmware for your router for various reasons: extending the use of a network protocol (such as IPv6), having new features, new piece of software inside, or for security reasons. A fully open-source firmware is de-facto needed for such applications, since you want to be free to use this or that version of a particular reason, be able to correct a particular bug. Few manufacturers do ship their routers with a Sample Development Kit, that would allow you to create your own and custom firmware and most of the time, when they do, you will most likely not be able to complete the firmware creation process.

This is one of the reasons why OpenWrt and other firmware exists: providing a version independent, and tools independent firmware, that can be run on various platforms, known to be running Linux originally.

**2.3.1****Which Operating System does this device run?**

There is a lot of methods to ensure your device is running Linux. Some of them do need your router to be unscrewed and open, some can be done by probing the device using its external network interfaces.

**Operating System fingerprinting and port scanning**

A large bunch of tools over the Internet exists in order to let you do OS fingerprinting, we will show here an 例子 using **nmap**:

nmap -P0 -O <ip< span="">address>   
Starting Nmap 4.20 ( http://insecure.org ) at 2007-01-08 11:05 CET   
Interesting ports on 192.168.2.1:   
Not shown: 1693 closed ports   
PORT STATE SERVICE   
22/tcp open ssh   
23/tcp open telnet   
53/tcp open domain   
80/tcp open http   
MAC Address: 00:13:xx:xx:xx:xx (Cisco-Linksys)   
Device type: broadband router   
Running: Linksys embedded   
OS details: Linksys WRT54GS v4 running OpenWrt w/Linux kernel 2.4.30   
Network Distance: 1 hop

nmap is able to report whether your device uses a Linux TCP/IP stack, and if so, will show you which Linux kernel version is probably runs. This report is quite reliable and it can make the distinction between BSD and Linux TCP/IP stacks and others.

Using the same tool, you can also do port scanning and service version discovery. For instance, the following command will report which IP-based services are running on the device, and which version of the service is being used:

|  |
| --- |
| nmap -P0 -sV  Starting Nmap 4.20 ( http://insecure.org ) at 2007-01-08 11:06 CET  Interesting ports on 192.168.2.1:  Not shown: 1693 closed ports  PORT STATE SERVICE VERSION  22/tcp open ssh Dropbear sshd 0.48 (protocol 2.0)  23/tcp open telnet Busybox telnetd  53/tcp open domain ISC Bind dnsmasq-2.35  80/tcp open http OpenWrt BusyBox httpd  MAC Address: 00:13:xx:xx:xx:xx (Cisco-Linksys)  Service Info: Device: WAP |

The web server version, if identified, can be determining in knowing the Operating System. For instance, the**BOA**web server is typical from devices running an open-source Unix or Unix-like.

**Wireless Communications Fingerprinting**

Although this method is not really known and widespread, using a wireless scanner to discover which OS your router or Access Point run can be used. We do not have a clear 例子 of how this could be achieved, but you will have to monitor raw 802.11 frames and compare them to a very similar device running a Linux based firmware.

**Web server security exploits**

The Linksys WRT54G was originally hacked by using a "ping bug" discovered in the web interface. This tip has not been fixed for months by Linksys, allowing people to enable the "boot\_wait" helper process via the web interface. Many web servers used in firmwares are open source web server, thus allowing the code to be audited to find an exploit. Once you know the web server version that runs on your device, by using **nmap -sV**or so, you might be interested in using exploits to reach shell access on your device.

**Native Telnet/SSH access**

Some firmwares might have restricted or unrestricted Telnet/SSH access, if so, try to log in with the web interface login/password and see if you can type in some commands. This is actually the case for some Broadcom BCM963xx based firmwares such as the one in Neuf/Cegetel ISP routers, Club-Internet ISP CI-Box and many others. Some commands, like **cat**might be left here and be used to determine the Linux kernel version.

**Analysing a binary firmware image**

You are very likely to find a firmware binary image on the manufacturer website, even if your device runs a proprietary operating system. If so, you can download it and use an hexadecimal editor to find printable words such as **vmlinux**, **linux**,**ramdisk**, **mtd**and others.

Some Unix tools like **hexdump**or **strings**can be used to analyse the firmware. Below there is an 例子 with a binary firmware found other the Internet:

|  |
| --- |
| hexdump -C | less (more)  00000000 46 49 52 45 32 2e 35 2e 30 00 00 00 00 00 00 00 |FIRE2.5.0.......|  00000010 00 00 00 00 31 2e 30 2e 30 00 00 00 00 00 00 00 |....1.0.0.......|  00000020 00 00 00 00 00 00 00 38 00 43 36 29 00 0a e6 dc |.......8.C6)..??|  00000030 54 49 44 45 92 89 54 66 1f 8b 08 08 f8 10 68 42 |TIDE..Tf....?.hB|  00000040 02 03 72 61 6d 64 69 73 6b 00 ec 7d 09 bc d5 d3 |..ramdisk.?}.???|  00000050 da ff f3 9b f7 39 7b ef 73 f6 19 3b 53 67 ea 44 |???.?9{?s?.;Sg?D| |

Scroll over the firmware to find printable words that can be significant.

**Amount of flash memory**

Linux can hardly fit in a 2MB flash device, once you have opened the device and located the flash chip, try to find its characteristics on the Internet. If your flash chip is a 2MB or less device, your device is most likely to run a proprietary OS such as WindRiver VxWorks, or a custom manufacturer OS like Zyxel ZynOS.

OpenWrt does not currently run on devices which have 2MB or less of flash memory. This limitation will probably not be worked around since those devices are most of the time micro-routers, or Wireless Access Points, which are not the main OpenWrt target.

**Pluging a serial port**

By using a serial port and a level shifter, you may reach the console that is being shown by the device for debugging or flashing purposes. By analysing the output of this device, you can easily notice if the device uses a Linux kernel or something different.

**2.3.2****Finding and using the manufacturer SDK**

Once you are sure your device run a Linux based firmware, you will be able to start hacking on it. If the manufacturer respected the GPL, it will have released a Sample Development Kit with the device.

**GPL violations**

Some manufacturers do release a Linux based binary firmware, with no sources at all. The first step before doing anything is to read the license coming with your device, then write them about this lack of Open Source code. If the manufacturer answers you they do not have to release a SDK containing Open Source software, then we recommend you get in touch with the gpl-violations.org community.

You will find below a sample letter that can be sent to the manufacturer:

Miss, Mister,

I am using a , and I cannot find neither on your website nor on the CD-ROM the open source software used to build or modify the firmware.

In conformance to the GPL license, you have to release the following sources:

* complete toolchain that made the kernel and applications be compiled (gcc, binutils, libc)
* tools to build a custom firmware (mksquashfs, mkcramfs ...)
* kernel sources with patches to make it run on this specific hardware, this does not include binary drivers

Thank you very much in advance for your answer.

Best regards,

**Using the SDK**

Once the SDK is available, you are most likely not to be able to build a complete or functional firmware using it, but parts of it, like only the kernel, or only the root filesystem. Most manufacturers do not really care releasing a tool that do work every time you uncompress and use it.

You should anyway be able to use the following components:

* kernel sources with more or less functional patches for your hardware
* binary drivers linked or to be linked with the shipped kernel version
* packages of the toolchain used to compile the whole firmware: gcc, binutils, libc or uClibc
* binary tools to create a valid firmware image

Your work can be divided into the following tasks:

* create a clean patch of the hardware specific part of the linux kernel
* spot potential kernel GPL violations especially on netfilter and USB stack stuff
* make the binary drivers work, until there are open source drivers
* use standard a GNU toolchain to make working executables
* understand and write open source tools to generate a valid firmware image

**Creating a hardware specific kernel patch**

Most of the time, the kernel source that comes along with the SDK is not really clean, and is not a standard Linux version, it also has architecture specific fixes backported from the **CVS**or the **git**repository of the kernel development trees. Anyway, some parts can be easily isolated and used as a good start to make a vanilla kernel work your hardware.

Some directories are very likely to have local modifications needed to make your hardware be recognized and used under Linux. First of all, you need to find out the linux kernel version that is used by your hardware, this can be found by editing the **linux/Makefile**file.

|  |
| --- |
| head -5 linux-2.x.x/Makefile  VERSION = 2  PATCHLEVEL = x  SUBLEVEL = y  EXTRAVERSION = z  NAME=A fancy name |

So now, you know that you have to download a standard kernel tarball at **kernel.org** that matches the version being used by your hardware.

Then you can create a **diff**file between the two trees, especially for the following directories:

|  |
| --- |
| diff -urN linux-2.x.x/arch/linux-2.x.x-modified/arch/ > 01-architecture.patch  diff -urN linux-2.x.x/include/ linux-2.x.x-modified/include > 02-includes.patch  diff -urN linux-2.x.x/drivers/ linux-2.x.x-modified/drivers > 03-drivers.patch |

This will constitute a basic set of three patches that are very likely to contain any needed modifications that has been made to the stock Linux kernel to run on your specific device. Of course, the content produced by the **diff -urN**may not always be relevant, so that you have to clean up those patches to only let the "must have" code into them.

The first patch will contain all the code that is needed by the board to be initialized at startup, as well as processor detection and other boot time specific fixes.

The second patch will contain all useful definitions for that board: addresses, kernel granularity, redefinitions, processor family and features ...

The third patch may contain drivers for: serial console, ethernet NIC, wireless NIC, USB NIC ... Most of the time this patch contains nothing else than "glue" code that has been added to make the binary driver work with the Linux kernel. This code might not be useful if you plan on writing drivers from scratch for this hardware.

**Using the device bootloader**

The bootloader is the first program that is started right after your device has been powered on. This program, can be more or less sophisticated, some do let you do network booting, USB mass storage booting ... The bootloader is device and architecture specific, some bootloaders were designed to be universal such as RedBoot or U-Boot so that you can meet those loaders on totally different platforms and expect them to behave the same way.

If your device runs a proprietary operating system, you are very likely to deal with a proprietary boot loader as well. This may not always be a limitation, some proprietary bootloaders can even have source code available (i.e : Broadcom CFE).

According to the bootloader features, hacking on the device will be more or less easier. It is very probable that the bootloader, even exotic and rare, has a documentation somewhere over the Internet. In order to know what will be possible with your bootloader and the way you are going to hack the device, look over the following features :

* does the bootloader allow net booting via bootp/DHCP/NFS or tftp
* does the bootloader accept loading ELF binaries ?
* does the bootloader have a kernel/firmware size limitation ?
* does the bootloader expect a firmware format to be loaded with ?
* are the loaded files executed from RAM or flash ?

Net booting is something very convenient, because you will only have to set up network booting servers on your development station, and keep the original firmware on the device till you are sure you can replace it. This also prevents your device from being flashed, and potentially bricked every time you want to test a modification on the kernel/filesystem.

If your device needs to be flashed every time you load a firmware, the bootlader might only accept a specific firmware format to be loaded, so that you will have to understand the firmware format as well.

**Making binary drivers work**

As we have explained before, manufacturers do release binary drivers in their GPL tarball. When those drivers are statically linked into the kernel, they become GPL as well, fortunately or unfortunately, most of the drivers are not statically linked. This anyway lets you a chance to dynamically link the driver with the current kernel version, and try to make them work together.

This is one of the most tricky and grey part of the fully open source projects. Some drivers require few modifications to be working with your custom kernel, because they worked with an earlier kernel, and few modifications have been made to the kernel in-between those versions. This is for instance the case with the binary driver of the Broadcom BCM43xx Wireless Chipsets, where only few differences were made to the network interface structures.

Some general principles can be applied no matter which kernel version is used in order to make binary drivers work with your custom kernel:

* turn on kernel debugging features such as:
  + CONFIG\_DEBUG\_KERNEL
  + CONFIG\_DETECT\_SOFTLOCKUP
  + CONFIG\_DEBUG\_KOBJECT
  + CONFIG\_KALLSYMS
  + CONFIG\_KALLSYMS\_ALL
* link binary drivers when possible to the current kernel version
* try to load those binary drivers
* catch the lockups and understand them

Most of the time, loading binary drivers will fail, and generate a kernel oops. You can know the last symbol the binary drivers attempted to use, and see in the kernel headers file, if you do not have to move some structures field before or after that symbol in order to keep compatibily with both the binary driver and the stock kernel drivers.

**Understanding the firmware format**

You might want to understand the firmware format, even if you are not yet capable of running a custom firmware on your device, because this is sometimes a blocking part of the flashing process.

A firmware format is most of the time composed of the following fields:

* header, containing a firmware version and additional fields: Vendor, Hardware version ...
* CRC32 checksum on either the whole file or just part of it
* Binary and/or compressed kernel image
* Binary and/or compressed root filesystem image
* potential garbage

Once you have figured out how the firmware format is partitioned, you will have to write your own tool that produces valid firmware binaries. One thing to be very careful here is the endianness of either the machine that produces the binary firmware and the device that will be flashed using this binary firmware.

**Writing a flash map driver**

The flash map driver has an important role in making your custom firmware work because it is responsible of mapping the correct flash regions and associated rights to specific parts of the system such as: bootloader, kernel, user filesystem.

Writing your own flash map driver is not really a hard task once you know how your firmware image and flash is structured. You will find below a commented 例子 that covers the case of the device where the bootloader can pass to the kernel its partition plan.

First of all, you need to make your flash map driver be visible in the kernel configuration options, this can be done by editing the file **linux/drivers/mtd/maps/Kconfig**:

|  |
| --- |
| config MTD\_DEVICE\_FLASH  tristate "Device Flash device"  depends on ARCHITECTURE && DEVICE  help  Flash memory access on DEVICE boards. Currently only works with  Bootloader Foo and Bootloader Bar. |

Then add your source file to the **linux/drivers/mtd/maps/Makefile**, so that it will be compiled along with the kernel.

|  |
| --- |
| obj-/$(CONFIG\_MTD\_DEVICE\_FLASH) += device-flash.o |

You can then write the kernel driver itself, by creating a **linux/drivers/mtd/maps/device-flash.c**C source file.

|  |
| --- |
| // Includes that are required for the flash map driver to know of the prototypes:  #include  #include  #include  #include  #include  #include  #include   // Put some flash map definitions here:  #define WINDOW\_ADDR 0x1FC00000 /\* Real address of the flash \*/  #define WINDOW\_SIZE 0x400000 /\* Size of flash \*/  #define BUSWIDTH 2 /\* Buswidth \*/   static void \_\_exit device\_mtd\_cleanup(void);   static struct mtd\_info \*device\_mtd\_info;   static struct map\_info devicd\_map = {  .name = "device",  .size = WINDOW\_SIZE,  .bankwidth = BUSWIDTH,  .phys = WINDOW\_ADDR,  };   static int \_\_init device\_mtd\_init(void)  {  // Display that we found a flash map device  printk("device: 0x/%08x at 0x/%08x/n", WINDOW\_SIZE, WINDOW\_ADDR);  // Remap the device address to a kernel address  device\_map.virt = ioremap(WINDOW\_ADDR, WINDOW\_SIZE);   // If impossible to remap, exit with the EIO error  if (!device\_map.virt) {  printk("device: Failed to ioremap/n");  return -EIO;  }   // Initialize the device map  simple\_map\_init(&device\_map);   /\* MTD informations are closely linked to the flash map device  you might also use "jedec\_probe" "amd\_probe" or "intel\_probe" \*/  device\_mtd\_info = do\_map\_probe("cfi\_probe", &device\_map);   if (device\_mtd\_info) {  device\_mtd\_info->owner = THIS\_MODULE;   int parsed\_nr\_parts = 0;   // We try here to use the partition schema provided by the bootloader specific code  if (parsed\_nr\_parts == 0) {  int ret = parse\_bootloader\_partitions(device\_mtd\_info, &parsed\_parts, 0);  if (ret > 0) {  part\_type = "BootLoader";  parsed\_nr\_parts = ret;  }  }   add\_mtd\_partitions(devicd\_mtd\_info, parsed\_parts, parsed\_nr\_parts);   return 0;  }  iounmap(device\_map.virt);   return -ENXIO;  }   // This function will make the driver clean up the MTD device mapping  static void \_\_exit device\_mtd\_cleanup(void)  {  // If we found a MTD device before  if (device\_mtd\_info) {  // Delete every partitions  del\_mtd\_partitions(device\_mtd\_info);  // Delete the associated map  map\_destroy(device\_mtd\_info);  }   // If the virtual address is already in use  if (device\_map.virt) {  // Unmap the physical address to a kernel space address  iounmap(device\_map.virt);  // Reset the structure field  device\_map.virt = 0;  }  }    // Macros that indicate which function is called on loading/unloading the module  module\_init(device\_mtd\_init);  module\_exit(device\_mtd\_cleanup);    // Macros defining license and author, parameters can be defined here too.  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Me, myself and I<memyselfandi@domain.tld");< div=""> |

**2.4 除错****Debugging and debricking**

**2.4.1 加入串口****Adding a serial port**

**2.4.2****JTAG**

**2.5****Reporting bugs**

**2.5.1****Using the Trac ticket system**

OpenWrt as an open source software opens its development to the community by having a publicly browseable subversion repository. The Trac software which comes along with a Subversion frontend, a Wiki and a ticket reporting system is used as an interface between developers, users and contributors in order to make the whole development process much easier and efficient.

We make distinction between two kinds of people within the Trac system:

* developers, able to report, close and fix tickets
* reporters, able to add a comment, patch, or request ticket status

**Opening a ticket**

A reporter might want to open a ticket for the following reasons:

* a bug affects a specific hardware and/or software and needs to be fixed
* a specific software package would be seen as part of the official OpenWrt repository
* a feature should be added or removed from OpenWrt

Regarding the kind of ticket that is open, a patch is welcome in those cases:

* new package to be included in OpenWrt
* fix for a bug that works for the reporter and has no known side effect
* new features that can be added by modifying existing OpenWrt files

Once the ticket is open, a developer will take care of it, if so, the ticket is marked as "accepted" with the developer name. You can add comments at any time to the ticket, even when it is closed.

**Closing a ticket**

A ticket might be closed by a developer because:

* the problem is already fixed (wontfix)
* the problem described is not judged as valid, and comes along with an explanation why (invalid)
* the developers know that this bug will be fixed upstream (wontfix)
* the problem is very similar to something that has already been reported (duplicate)
* the problem cannot be reproduced by the developers (worksforme)

At the same time, the reporter may want to get the ticket closed since he is not longer able to trigger the bug, or found it invalid by himself.

When a ticket is closed by a developer and marked as "fixed", the comment contains the subversion changeset which corrects the bug.

**2.6 发送补丁****Submitting patches**

**2.6.1****How to contribute**

OpenWrt is constantly being improved. We'd like as many people to contribute to this as we can get. If you find a change useful, by all means try to get it incorporated into the project. This should improve OpenWrt and it should help carry your changes forward into future versions

This section tries to lay out a procedure to enable people to submit patches in a way that is most effective for all concerned.

It is important to do all these steps repeatedly:

* *listen*to what other people think.
* *talk*explaining what problem you are addressing and your proposed solution.
* *do*write useful patches including documentation.
* *test. test. test.*

**2.6.2 在哪里讨论****Where to listen and talk**

* google to find things related to your problem
* Mailing lists: <http://lists.openwrt.org/>
* Wiki: check the wiki: <http://wiki.openwrt.org/OpenWrtDocs>
* Forum: <http://forum.openwrt.org/>
* IRC: irc.freenode.net, channels #openwrtand #openwrt-devel
* TRAC: <https://dev.openwrt.org/> the issue/bug/change tracking system

It is often best to document what you are doing before you do it. The process of documentation often exposes possible improvements. Keep your documentation up to date.

**2.6.3 补丁发布流程****Patch Submission Process**

1. Use git or svn to create a patch. Creating patches manually with *diff -urN* also works, but is usually unnecessary.
2. Send a mail to openwrt-devel@lists.openwrt.org with the following contents:
   1. [PATCH] in the Subject, followed by:
   2. (可选的) a longer description of your patch in the message body
   3. Signed-off-by: Your name
   4. Your actual patch, inline, not word wrapped or whitespace mangled.
3. Please read <http://kerneltrap.org/Linux/Email_Clients_and_Patches> to find out how to make sure your email client doesn't destroy your patch.
4. Please use your real name and email address in the Signed-off-byline, following the same guidelines as in the [Linux Kernel patch submission guidelines](http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/SubmittingPatches;h=681e2b36195c98ea5271b76383b3a574b190b04f;hb=HEAD)
5. 例子 of a properly formatted patch submission:   
   <http://lists.openwrt.org/pipermail/openwrt-devel/2007-November/001334.html>