

OpenWrt Development Guide

Tao Jin taojin@ccs.neu.edu

Wireless Networks Lab, CCIS, NEU

February 13, 2012

Overview of OpenWrt BuildRoot Environment

OpenWrt Buildroot environment is a collection of Makefiles, patches and scripts, which generates the cross-compilation toolchain, downloads Linux kernel, generates a root file system, manages 3rd party packages, etc. The cross-compilation toolchain uses uClibc. With OpenWrt Buildroot, developers can compile the custom firmware image for supported hardware / architectures. Please NOTE that in the OpenWrt Buildroot source tree, there is no Linux kernel or any source code tarballs of the 3rd party packages. The collection of Makefiles determines the version of Linux kernel to download, and the version of the package tarball to download and compiled in to the image. Figure 1 shows the structure of the OpenWrt Buildroot source tree.

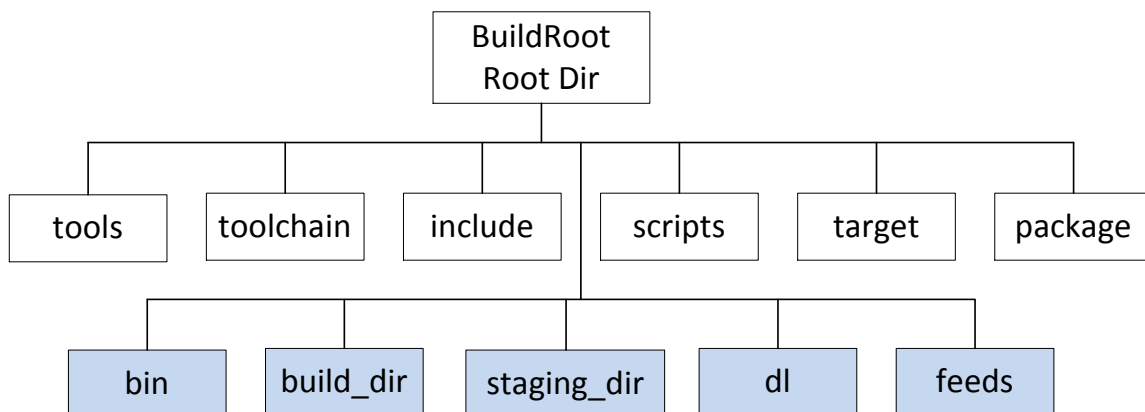


Figure 1. OpenWrt Buildroot Source Tree (The folders in the second lines are generated during compilation).

- **tools** – contains all the build instructions to fetch the image building tools
- **toolchain** - contains all the build instructions to fetch the kernel headers, the C library, the bin-utils, the compiler itself and the debugger. If you add a completely new architecture, you would add a configuration for the C library here.
- **target** - build instruction for firmware image generating process and for the kernel building process; compiles kernel and firmware image utilities, builds firmware image, generate Image Generator (former called Image Builder)
- **package** – the OpenWrt Makefiles and patches for all the main packages. The OpenWrt Makefile has its own syntax, different from the conventional Makefile of Linux make tool.

The OpenWrt Make file defines the meta information of the package, where to download the package, how to compile, where to installed the compiled binaries, etc. See [How to Build OpenWrt Application Package](#) for more detail.

- **include** -
- **scripts** – perl scripts that does the OpenWrt package management
- **dl** - Where the user-space package tarballs will be downloaded
- **build_dir** – where all user-space tools will be cross-compiled
- **staging_dir** – where the cross-compilation tools will be installed
- **feeds** –
- **bin** – where the firmware image will be generated and all the .ipk package files will be generated

Simply speaking, once the OpenWrt buildroot has been properly configured, e.g. the target platform and architecture is specified, user-space packages selected, etc., the OpenWrt Buildroot will do the image building through the following steps (once the configuration is done):

1. Download the cross-compilation tools, kernel headers, etc. and
2. Set up the staging directory (staging_dir /). This is where the cross-compilation toolchain will be installed. If you want to use the same cross-compilation toolchain for other purposes, such as compiling third-party applications, you can find the cross-compiler tools in this directory, and then use arch-linux-gcc to compile your application.
3. Create the download directory (dl/ by default). This is where the tarballs will be downloaded.
4. Create the build directory (build_dir/). This is where all user-space tools while be compiled.
5. Create the target directory (build_dir/target-arch/root by default) and the target filesystem skeleton. This directory will contain the final root filesystem.
6. Install the user-space packages to the root file system and compress the whole root file system with proper format. The result firmware image is generated in bin/

How to Build OpenWRT Firmware (Step-by-Step Tutorial)

NOTE: The router vendors may change their hardware design slightly every once a while, even for the same product. For example, Buffalo WZR-HP-G300NH router has three different models, WZR-HP-G300NH, WZR-HP-G301NH and WZR-HP-G300NH2. OpenWRT trunk code base is frequently updated to add support for the newly identified hardware models. **This step-by-step guide is made based on OpenWRT trunk revision 30368, for Buffalo WZR-HP-G300NH2 model.**

1. Check out OpenWrt Buildroot source tree from the svn server. Revision 30368 is the one that is verified to function properly on this model.
`$ svn co -r 30368 svn://svn.openwrt.org/openwrt/trunk/`

if you want to check out the latest trunk code, run

```
$ svn co svn://svn.openwrt.org/openwrt/trunk/
```

OpenWRT has different branches and trunk code, but at the moment of writing, WZR-HP-G300NH2 model is only supported in trunk. For other branches, check

<https://dev.openwrt.org/wiki/GetSource> for more detail.

2. `$ cd trunk`

3. Update package list by running

```
$ ./scripts/feeds update
```

This command will download the latest package list from OpenWRT server, and build indices locally in `./feeds`. In `./feeds/packages` are included all available packages and their corresponding Makefile's

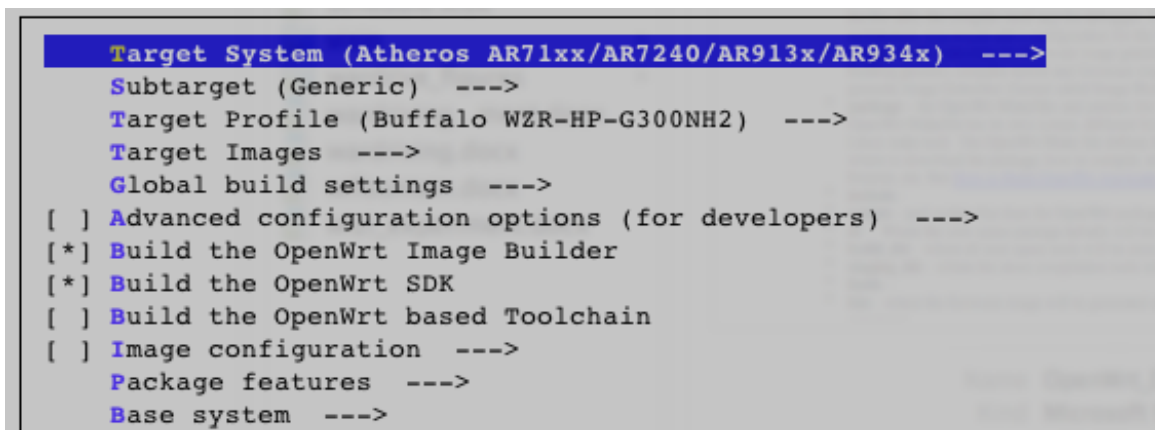
4. Install all package info to config file for later make operation

```
$ make package/symlinks
```

This step will install symbolic links for all packages downloaded in the previous step at `./package/feeds/`. If you check the files in that folder, you will find out they are all symbolic links directing to `./feeds/packages/`. In this way, "make menuconfig" command can find all available packages and show them in the menuconfig interface. NOTE: without this step, you cannot see all available packages in next step

5. Customize your build configuration, also this will check the dependencies and availability of required tools

```
$ make menuconfig
```



```
Target System (Atheros AR71xx/AR7240/AR913x/AR934x) --->
Subtarget (Generic) --->
Target Profile (Buffalo WZR-HP-G300NH2) --->
Target Images --->
Global build settings --->
[ ] Advanced configuration options (for developers) --->
[*] Build the OpenWrt Image Builder
[*] Build the OpenWrt SDK
[ ] Build the OpenWrt based Toolchain
[ ] Image configuration --->
Package features --->
Base system --->
```

Figure 2. OpenWrt make menuconfig window

Several things you have to specify:

- Target System : the chipset architecture of your target hardware. WZR-HP-G300NH uses AR71xx series chipset. See <http://wiki.openwrt.org/toh/buffalo/wzr-hp-g300nh2> for detail about the hardware spec of this router.
- Target Profile: the target hardware model. The image building process may build some hardware-dependent config files into the image. E.g. WZR-HP-G300NH2 requires specific Ethernet switch driver specified in the network config file
- Target Images: Most hardwares need squashfs file system.

```

[ ] ramdisk --->
--- Root filesystem archives
[ ] cpio.gz
[*] tar.gz
--- Root filesystem images
[ ] ext4
[*] jffs2
[*] squashfs
--- Image Options

```

Figure 2. OpenWrt make menuconfig window

6. Start building process

`$ make world`

If you want to see detailed compilation output, add “V=99” at the end of the make command

How to Flash the Image to Buffalo WZR-HP-G300NH2

Different models may require different flashing process, which all depends on the bootloader implementation. We will describe the flashing process for Buffalo WZR-HP-G300NH2. As for the flashing methods for other OpenWRT compatible models, you can probably find that in the OpenWRT official wiki site or the OpenWrt forum (<http://openwrt.org>)

WZR-HP-G300NH2 has a u-boot bootloader, which implements the boot_wait function. When the router is powered on, before the bootloader starts loading the actual firmware image, it waits for 4 second (by default) during which the bootloader has a basic Ethernet network stack and TFTP server initialized and listens for a new firmware image to be pushed via TFTP.

Step 1. Prepare the Firmware Image for WZR-HP-G300NH2

Prepare the firmware image following the above instruction. This would generate a set of bin files under bin/ar71xx/. The images that are compatible with WZR-HP-G300NH2 are

- openwrt-ar71xx-generic-wzr-hp-g300nh2-squashfs-sysupgrade.bin

- openwrt-ar71xx-generic-wzr-hp-g300nh2-squashfs-tftp.bin

The second image is the one that should be used with TFTP method, because this image has a 32-byte header which is required by the u-boot to identify the hardware model. Without proper header, u-boot will not accept the image. For the first time flashing, i.e. flashing the router from the factory firmware to OpenWRT, you have to use the TFTP flashing method. The other one openwrt-ar71xx-generic-wzr-hp-g300nh2-squashfs-sysupgrade.bin is used to flash the router, when the router is already running OpenWRT.

Step 2. Flash WZR-HP-G300NH2

1. Disconnect the power of the router
2. Connect your PC to one of the LAN ports of the router
3. set your PC with static IP 192.168.11.xx, b/c the uboot bootloader use 192.168.11.1 static IP at boot time.
4. set static arp entry on the PC.
`$ arp -s 192.168.11.1 02:aa:bb:cc:dd:1a`
NOTE: the u-boot bootloader uses this fixed MAC address on boot time. After the router is up and running, it will use different MAC addresses.
5. Disable any firewall on your PC. On Linux, it is
`$ iptables -F`
6. Go to the folder that saves the tftp image file, openwrt-ar71xx-generic-wzr-hp-g300nh2-squashfs-tftp.bin. Run the following command:
`$ tftp 192.168.11.1`
`> binary`
`> rexmt 1`
`> timeout 60`
`> trace`
`> put openwrt-ar71xx-generic-wzr-hp-g300nh2-squashfs-tftp.bin`
7. Power on the router
8. wait for 5 seconds, the router will enter boot_wait stage. In the tftp console from step 6, you should see data being transferred to the router via TFTP.
9. wait for another 3 mins for the router to finish the flashing process.
10. After router reboots, the PC should get an IP from the router 192.168.1.x. By default, there is no root password set on the router, and the ssh is disabled. You can telnet to 192.168.1.1 to set the root password. Once root password is set, the ssh is enabled automatically.

NOTE: Once WZR-HP-G300NH2 is flashed properly and runs OpenWRT, we can upgrade the firmware with the sysupgrade tool. Sysupgrade tool is a user space tool built into the OpenWrt firmware. In case the router is bricked, you can use the TFTP method to reflash the router with good image. See <http://wiki.openwrt.org/doc/howto/generic.sysupgrade> for detail about sysupgrade tool.

How to Build OpenWrt User-Space Package (Step-by-Step Tutorial)

Similar to .deb package in Debian system, OpenWrt compresses each user-space package as a .ipk file, which contains the binaries, config files and the instruction of installing the package to the root file system.

OpenWrt buildroot provides a SDK package, which is a stripped-off buildroot, which includes necessary toolchain and header files to cross-compile user-space programs.

There are two options to get OpenWRT SDK:

- Download pre-compiled SDK from <http://downloads.openwrt.org/> . Note that the SDK version should be compatible with the version of the firmware.
- Build your own SDK tar ball by selecting the SDK option in make menuconfig window as shown in Figure 2. The SDK .tar.bz2 file is generated in bin/ar71xx directory

After obtaining the SDK tar ball in either way, uncompress the SDK to any path, and it is ready to use.

Next, we will explain how to create a user-space helloworld package step by step. In this example, we download the precompiled SDK from OpenWRT website.

Overview of OpenWRT Package

We use <BUILDROOT> to denote the top directory of the OpenWRT buildroot source tree checked out with svn. The <BUILDROOT>/package is the subdirectory where all the OpenWRT packages are saved. A typical OpenWRT user space package, e.g. helloworld, looks like this.

```
<BUILDROOT>/helloworld/Makefile  
<BUILDROOT>/helloworld/patches/
```

The Makefile is NOT the term you are familiar with about the GNU Make tool. Instead, the Makefile is an OpenWRT package Makefile, with totally different syntax. The OpenWRT Makefile defines the instructions of building a OpenWRT package, including where to download the source code, how to compile, which path in the firmware should the compiled binary be installed, etc. A typical OpenWRT package directory has NO source folder. Downloading the source tar ball from proper URL is generally the first step of building OpenWRT package. The URL to download the source tar ball is defined in the OpenWRT Makefile. E.g.

```
PKG_SOURCE:=helloworld.tar.gz  
PKG_SOURCE_URL:=http://www.example.com/  
PKG_MD5SUM:=e06c222e186f7cc013fd272d023710cb
```

The above three lines defined in helloworld OpenWRT Makefile means, download helloworld.tar.gz from <http://www.example.com/helloworld.tar.gz> The MD5SUM is used to

verify the integrity of the downloaded package. Generally, the helloworld.tar.gz will be downloaded to <BUILDROOT>/dl/ subdirectory. The tar ball is then uncompressed to <BUILDROOT>/build_dir/helloworld, which actually contains all the source files for compilation.

An alternative simplified approach is to store the source files in the same package directory.

```
<BUILDROOT>/helloworld/Makefile
<BUILDROOT>/helloworld/patches/
<BUILDROOT>/helloworld/src/
<BUILDROOT>/helloworld/src/helloworld.c
<BUILDROOT>/helloworld/src/Makefile
```

In this case, obviously there is no need to download any tar ball from the web, because all the source files are present in the same OpenWRT package directory. NOTE that <BUILDROOT>/helloworld/src/Makefile is the generic GNU Makefile.

The patches subdirectory is optional and typically contains bug fixes or optimizations to reduce the size of the executable.

OpenWRT BuildPackage Makefile Syntax

The OpenWRT BuildPackage Makefile defines a set of variables to instruct the OpenWRT Buildroot or SDK to compile the package. Here are the list of BuildPackage variables and defines.

PKG_* defines the variables related to package information, such as where to download and version information.

- PKG_NAME -The name of the package, as seen via menuconfig and ipkg
- PKG_VERSION -The upstream version number that we're downloading
- PKG_RELEASE -The version of this package Makefile
- PKG_BUILD_DIR -Where to compile the package
- PKG_SOURCE -The filename of the original sources
- PKG_SOURCE_URL -Where to download the sources from
- PKG_MD5SUM -A checksum to validate the download
- PKG_CAT -How to decompress the sources (zcat, bzip, unzip)
- PKG_BUILD_DEPENDS -Packages that need to be built before this package, but are not required at runtime. Uses the same syntax as DEPENDS below.

Package/* describes the package, the menuconfig and ipkg entries.

- SECTION - The type of package (currently unused)
- CATEGORY - Which menu it appears in menuconfig
- TITLE - A short description of the package
- DESCRIPTION - (deprecated) A long description of the package

- URL - Where to find the original software
- MAINTAINER - (optional) Who to contact concerning the package
- DEPENDS - (optional) Which packages must be built/installed before this package
- conffiles - (optional) A list of config files installed by this package, one file per line.

Build/* defines some instructions related to package building

- Build/Prepare - (optional) A set of commands to unpack and patch the sources. You may safely leave this undefined.
- Build/Configure - (optional) 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.
- Build/Compile - (optional) How to compile the source; in most cases you should leave this undefined.
- Package/install - A set of commands to copy files out of the compiled source and into the ipkg, which is represented by the \$(1) directory.
- Package/preinst - The actual text of the script which is to be executed before installation. Dont forget to include the #!/bin/sh. If you need to abort installation have the script return false.
- Package/postinst - The actual text of the script which is to be executed after installation. Dont forget to include the #!/bin/sh.
- Package/prerm - The actual text of the script which is to be executed before removal. Dont forget to include the #!/bin/sh. If you need to abort removal have the script return false.
- Package/postrm - The actual text of the script which is to be executed after removal. Dont forget to include the #!/bin/sh.

Here is a step-by-step guide of building helloworld package with OpenWRT SDK.

You can find the OpenWRT wiki page for SDK here

(<http://wiki.openwrt.org/doc/howto/obtain.firmware.sdk>)

1. Download the SDK from http://downloads.openwrt.org/backfire/10.03.1-rc5/ar71xx/OpenWrt-SDK-ar71xx-for-Linux-i686-gcc-4.3.3+cs_uClibc-0.9.30.1.tar.bz2
NOTE: For some reason, the SDK package built with OpenWRT backfire10.0.1-rc5 is buggy, and cannot function properly. To obtain a working SDK, you can check out the trunk code of buildroot source tree from [svn co svn://svn.openwrt.org/openwrt/trunk/](http://svn.openwrt.org/openwrt/trunk/) and build the SDK following the step-by-step guide in the previous section.
2. Uncompress the package to openwrt-sdk

```
$ tar xvfj OpenWrt-SDK-ar71xx-for-Linux-i686-gcc-4.3.3+cs_uClibc-0.9.30.1.tar.bz2
$ mv OpenWrt-SDK-ar71xx-for-Linux-i686-gcc-4.3.3+cs_uClibc-0.9.30.1 openwrt-sdk
```


3. Create a folder named helloworld in the SDK/package subdirectory
\$ mkdir openwrt-sdk/package/helloworld
4. Edit helloworld.c and save it in openwrt-sdk/package/helloworld/src

```
#include <stdio.h>
int main()
{
    printf("hello world\n")
    return 0;
}
```

5. Edit GNU Makefile and save it in openwrt-sdk/package/helloworld/src

```
CC = gcc
FLAG = -Wall
```

```
helloworld:
```

```
$(CC) $(FLAG) helloworld.c -o helloworld
```

6. Edit the OpenWRT BuildPackage Makefile and save it in openwrt-sdk/package/helloworld/
Here is a sample helloworld Makefile

```
#####
# OpenWrt Makefile for helloworld program
#
#
# Most of the variables used here are defined in
# the include directives below. We just need to
# specify a basic description of the package,
# where to build our program, where to find
# the source files, and where to install the
# compiled program on the router.
#
# Be very careful of spacing in this file.
# Indents should be tabs, not spaces, and
# there should be no trailing whitespace in
# lines that are not commented.
#
#####

include $(TOPDIR)/rules.mk

# Name and release number of this package
PKG_NAME:=helloworld
PKG_RELEASE:=1

# This specifies the directory where we're going to build the program.
```

```

# The root build directory, $(BUILD_DIR), is by default the build_mipsel
# directory in your OpenWrt SDK directory
PKG_BUILD_DIR := $(BUILD_DIR)/$(PKG_NAME)

include $(INCLUDE_DIR)/package.mk

# Specify package information for this program.
# The variables defined here should be self explanatory.
# If you are running Kamikaze, delete the DESCRIPTION
# variable below and uncomment the Kamikaze define
# directive for the description below
define Package/helloworld
    SECTION:=utils
    CATEGORY:=Utilities
    TITLE:=Helloworld Program
endef

# Uncomment portion below for Kamikaze and delete DESCRIPTION variable above
define Package/helloworld/description
    a sample OpenWRT helloworld program
endef

# Specify what needs to be done to prepare for building the package.
# In our case, we need to copy the source files to the build directory.
# This is NOT the default. The default uses the PKG_SOURCE_URL and the
# PKG_SOURCE which is not defined here to download the source from the web.
# In order to just build a simple program that we have just written, it is
# much easier to do it this way.
define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/ # Since we host the source folder in the same
OpenWRT package folder, we copy everything in source folder to $(PKG_BUILD_DIR)/
endef

# Specify where and how to install the program. Since we only have one file,
# the helloworld executable, install it by copying it to the /bin directory on
# the router. The $(1) variable represents the root directory on the router running
# OpenWrt. The $(INSTALL_DIR) variable contains a command to prepare the install
# directory if it does not already exist. Likewise $(INSTALL_BIN) contains the
# command to copy the binary file from its current location (in our case the build
# directory) to the install directory.
define Package/helloworld/install
    $(INSTALL_DIR) $(1)/usr/sbin
    # copy the helloworld binary to /usr/bin of OpenWRT
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/usr/sbin/
endef

```

```
# This line executes the necessary commands to compile our program.
# The above define directives specify all the information needed, but this
# line calls BuildPackage which in turn actually uses this information to
# build a package.
$(eval $(call BuildPackage,helloworld))
```

5. Build helloworld package
\$ make package/helloworld-compile V=99

This cross-compile the helloworld and generates helloworld.ipk in openwrt-sdk/bin/ar71xx

6. Copy the helloworld.ipk to the tmp directory on router
\$ scp helloworld.ipk root@<routerIP>:/tmp/
7. Install helloworld
\$ opkg install /tmp/helloworld.ipk
8. Verify helloworld is properly installed under /usr/bin
\$ which helloworld
/usr/bin/helloworld
9. Try helloworld
\$ helloworld
hello world
\$

For more detailed instruction, please refer to <http://wiki.openwrt.org/doc/howto/obtain.firmware.sdk> and <http://wiki.openwrt.org/doc/devel/packages>

References:

1. OpenWrt Buildroot – Technical Reference (<http://wiki.openwrt.org/doc/techref/buildroot>)
2. OpenWrt Buildroot Documentation (<http://downloads.openwrt.org/docs/buildroot-documentation.html>)
3. WZR-HP-G301NH Hardware Info (<http://wiki.openwrt.org/toh/buffalo/wzr-hp-g300h?s%5B%5D=table&s%5B%5D=hardware>)
4. Creating OpenWrt Packages (<http://wiki.openwrt.org/doc/devel/packages>)
5. Using OpenWrt SDK (<http://wiki.openwrt.org/doc/howto/obtain.firmware.sdk>)