## OpenWRT中wlan的工具(hostapd/wpa\_supplicant)的编译使用

由于hostapd官网(<http://w1.fi/hostapd/>)上的hostapd和wpa\_supplicant功能很全，编译出来比较大，在IOT平台占用的资源太多，所以我们尝试使用openwrt项目精简过的hostapd工具。

Openwrt是一般无线路由器的基于Linux的OS，带的工具基本功能是完整的，可以满足我们的需要。

验证分为两步。

1. 验证openwrt中的wpad-mini是否功能完备。验证方法是编译一个可以在virtualbox虚拟机上跑的openwrt版本，然后使用虚拟的Wifi驱动mac80211\_hwsim来测试ap和sta 的功能。测试通过说明基本功能是完善的。这个时候需要编译openwrt为x86版本，才能在虚拟机上跑
2. 查看wpad-mini的code size和data size，我这里看的是x64的binary，编译成ARM可能更准确点，但RAM大小应该是一样, code size有点区别

# 编译OpenWRT

1. 下载代码

git clone <https://git.openwrt.org/openwrt/openwrt.git>

1. 编译设置

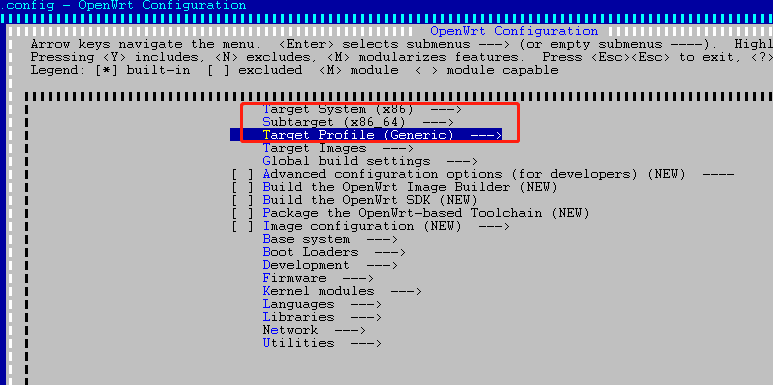
sudo apt-get install gawk

cd openwrt

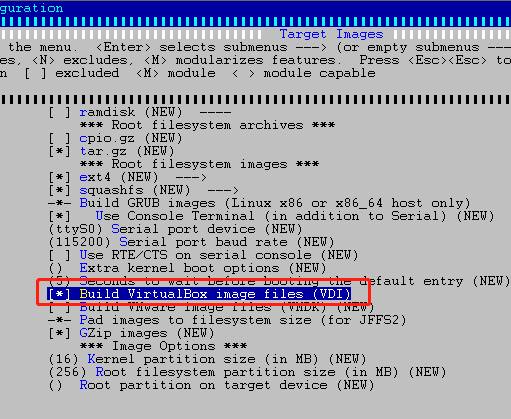
make menuconfig

> Target System

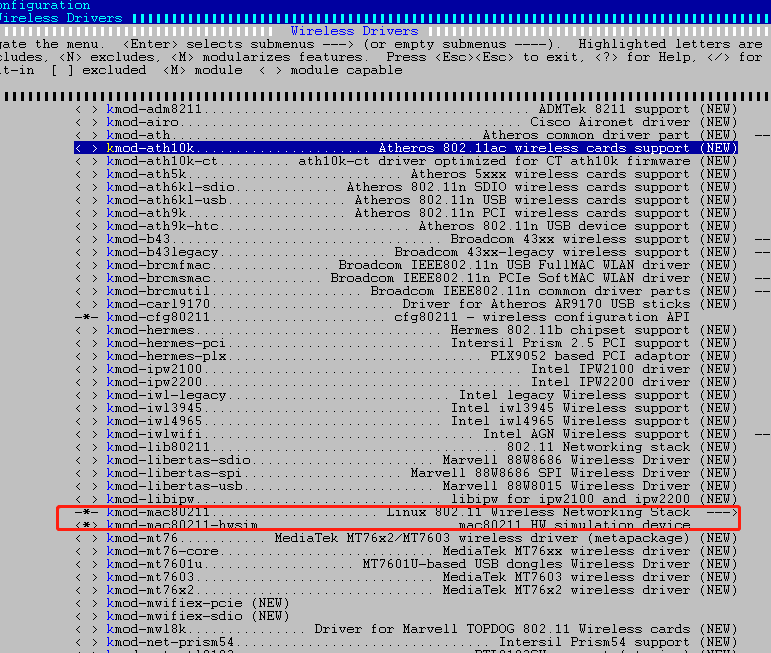
> SubTarget



> Target Images



> Kernel modules -> Wireless Drivers



> Network



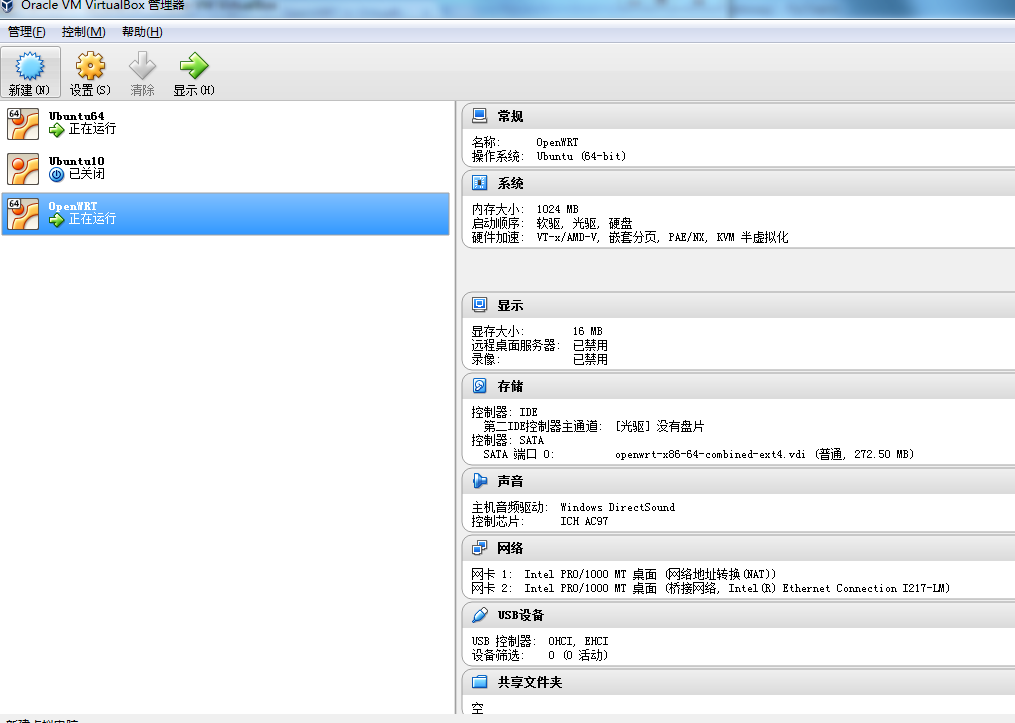
> Exit -> Save

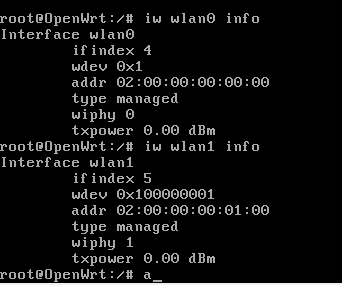
> make V=s

可以需要编译几个小时, 因为需要从网上下载软件包

编译完成后

**openwrt/bin/targets/x86/64/openwrt-x86-64-combined-ext4.vdi**

可以使用virtualbox 加载这个VDI在虚拟机上启动OpenWRT



启动后可以看见有两个虚拟的WIFI设备, 这个时候mac80211\_hwsim.ko生成

这个时候可以用wpad来启动AP和STA

启动AP: wpad hostapd hostapd.conf

启动STA: wpad wpa\_supplicant hostapd.conf

可以用iw命令AP和STA可以连上，说明wpad-mini的功能是完备的

# 查看生成的wpad-mini大小

#./staging\_dir/toolchain-x86\_64\_gcc-5.5.0\_musl/bin/x86\_64-openwrt-linux-musl-strip ./staging\_dir/target-x86\_64\_musl/root-x86/usr/sbin/wpad

#./staging\_dir/toolchain-x86\_64\_gcc-5.5.0\_musl/bin/x86\_64-openwrt-linux-musl-objdump -hww ./staging\_dir/target-x86\_64\_musl/root-x86/usr/sbin/wpad

./staging\_dir/target-x86\_64\_musl/root-x86/usr/sbin/wpad: file format elf64-x86-64

Sections:

Idx Name Size VMA LMA File off Algn Flags

0 .interp 00000019 0000000000400200 0000000000400200 00000200 2\*\*0 CONTENTS, ALLOC, LOAD, READONLY, DATA

1 .hash 000004a0 0000000000400220 0000000000400220 00000220 2\*\*3 CONTENTS, ALLOC, LOAD, READONLY, DATA

2 .dynsym 00000f48 00000000004006c0 00000000004006c0 000006c0 2\*\*3 CONTENTS, ALLOC, LOAD, READONLY, DATA

3 .dynstr 00000641 0000000000401608 0000000000401608 00001608 2\*\*0 CONTENTS, ALLOC, LOAD, READONLY, DATA

4 .gnu.version 00000146 0000000000401c4a 0000000000401c4a 00001c4a 2\*\*1 CONTENTS, ALLOC, LOAD, READONLY, DATA

5 .gnu.version\_r 00000020 0000000000401d90 0000000000401d90 00001d90 2\*\*3 CONTENTS, ALLOC, LOAD, READONLY, DATA

6 .rela.dyn 00000e88 0000000000401db0 0000000000401db0 00001db0 2\*\*3 CONTENTS, ALLOC, LOAD, READONLY, DATA

7 .rela.plt 00000030 0000000000402c38 0000000000402c38 00002c38 2\*\*3 CONTENTS, ALLOC, LOAD, READONLY, DATA

8 .init 0000000d 0000000000402c68 0000000000402c68 00002c68 2\*\*0 CONTENTS, ALLOC, LOAD, READONLY, CODE

9 .plt 00000030 0000000000402c80 0000000000402c80 00002c80 2\*\*4 CONTENTS, ALLOC, LOAD, READONLY, CODE

10 .plt.got 000004b8 0000000000402cb0 0000000000402cb0 00002cb0 2\*\*3 CONTENTS, ALLOC, LOAD, READONLY, CODE

11 .text 00069098 0000000000403168 0000000000403168 00003168 2\*\*0 CONTENTS, ALLOC, LOAD, READONLY, CODE

12 .fini 00000008 000000000046c200 000000000046c200 0006c200 2\*\*0 CONTENTS, ALLOC, LOAD, READONLY, CODE

13 .rodata 00016dc1 000000000046c220 000000000046c220 0006c220 2\*\*5 CONTENTS, ALLOC, LOAD, READONLY, DATA

14 .eh\_frame\_hdr 00003394 0000000000482fe4 0000000000482fe4 00082fe4 2\*\*2 CONTENTS, ALLOC, LOAD, READONLY, DATA

15 .eh\_frame 000143f4 0000000000486378 0000000000486378 00086378 2\*\*3 CONTENTS, ALLOC, LOAD, READONLY, DATA

16 .ctors 00000010 000000000069a900 000000000069a900 0009a900 2\*\*3 CONTENTS, ALLOC, LOAD, DATA

17 .dtors 00000010 000000000069a910 000000000069a910 0009a910 2\*\*3 CONTENTS, ALLOC, LOAD, DATA

18 .data.rel.ro 00000010 000000000069a920 000000000069a920 0009a920 2\*\*3 CONTENTS, ALLOC, LOAD, DATA

19 .dynamic 000001f0 000000000069a930 000000000069a930 0009a930 2\*\*3 CONTENTS, ALLOC, LOAD, DATA

20 .got 000004e0 000000000069ab20 000000000069ab20 0009ab20 2\*\*3 CONTENTS, ALLOC, LOAD, DATA

21 .data 000007e4 000000000069b000 000000000069b000 0009b000 2\*\*5 CONTENTS, ALLOC, LOAD, DATA

22 .bss 00000238 000000000069b800 000000000069b800 0009b7e4 2\*\*5 ALLOC

23 .comment 0000002d 0000000000000000 0000000000000000 0009b7e4 2\*\*0 CONTENTS, READONLY

图中绿色加量的部分为代码段，size=0x69098+0x9+0x16dc1+0x10+0x10= 523906 bytes, 这部分最终要放到Flash上面

黄色高亮的为数据段， size=0x7e4+0x238=2588 bytes, 这部分需要使用RAM。因为一直到RTOS上面的时候大部分动态分配的内存应该都是操作系统用户空间和Kernel空间来回调用的时候用到的，大部分应该用不到。还有很多分配可以在移植的过程中优化，所以预估10K RAM应该可以达到要求。