Intro to Embedded Linux Part 7 - Enable WiFi Networking with Yocto

By ShawnHymel

In the previous tutorial, we showed you how to build a custom application into a Linux image using the Yocto Project. This time, we enable networking (specifically, WiFi) by including certain kernel modules in our image.

See here if you would like to view this tutorial in video format:

Required Hardware

I will try to explain what is happening at each step in these tutorials so that you can generalize the instructions to almost any single board computer (assuming the board is supported by the build system). However, for the demonstration, I will be using the following platform:

STM32MP157D-DK1

You will also need an SD card. The STM32MP157D-DK1 kit should come with an SD card. In addition, you will need a USB-C power supply capable of supplying 5V, 3A.

You will need a USB WiFi dongle for this tutorial. I am using a TP-Link TL-WN725N. Note that not all adapters will work with Linux--it will likely require some trial and error.

Required Software

You will need Linux for this project, as all of the tools we are using must be run in Linux (on the host computer). I will show steps that work in Ubuntu and Linux Mint (and likely other flavors of Debian), but you can probably get almost any Linux distro to work. LiveCD, dual-booting, Windows Subsystem for Linux (WSL), and pre-made Docker images will also likely work.

I also recommend using a fairly modern computer with at least 4GB of RAM. While you can probably build a Linux image in a Raspberry Pi, expect it to take a very long time.

Enable Kernel Modules

Look up the chipset for your USB WiFi adapter (using a site such as http://linux-wless.passys.nl/). For example, the chipset in my TL-WN725N is an rtl8188eu.

Enable the build environment:

Copy Code

```
cd ~/Projects/yocto
source poky/oe-init-build-env build-mp1
```

Load the kernel configuration tool:

Copy Code

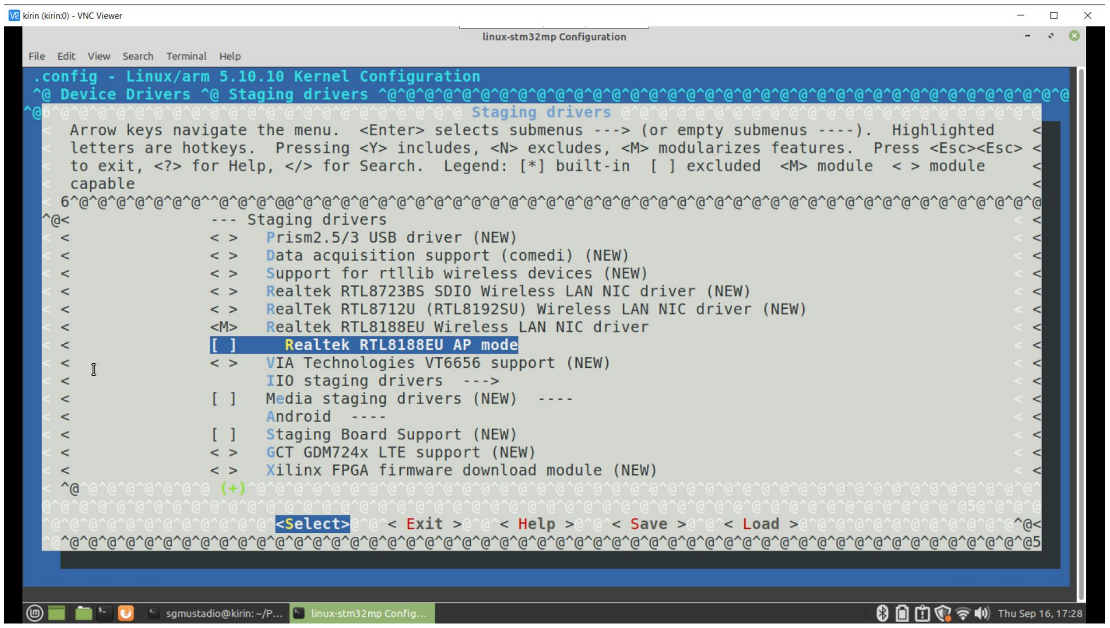
```
bitbake -c menuconfig virtual/kernel
```

Here, enable and disable the following settings:

Copy Code

```
Networking support > Networking options (following should be enabled):
    [*] TCP/IP networking
        [*] IP: kernel level autoconfiguration
            [*] IP: DHCP support
            [*] IP: BOOTP support
            [*] IP: RARP support
        <*> The IPv6 protocol
Networking support > Wireless
    <M> cfg80211 - wireless configuration API
        [*] cfg80211 wireless extensions compatibility
    <M> Generic IEEE 802.11 Networking Stack (mac80211)
    [*] Minstrel (for rate control in 802.11)
Device Drivers> Network device support > Wireless LAN
    [*] Realtek devices
        < > Realtek rtlwifi family of devices
            < > Realtek RTL8192CU/RTL8188CU USB Wireless Network Adapter
            [ ] Debugging output for rtlwifi driver family
        < > RTL8723AU/RTL8188[CR]U/RTL8192[12]CU (mac80211) support
            [ ] Include support for untested Realtek 8xxx USB devices (EXPERIMENTAL)
Device Drivers > USB support, make sure USB Host is enabled:
    <*> Support for Host-side USB
    [*] Enable USB persist by default
Device Drivers > Staging Drivers
    <M> Realtek RTL8188EU Wireless LAN NIC driver
        [ ] Realtek RTL8188EU AP mode (NEW)
[*] Enable loadable module support  --->
    [*]   Module signature verification
        [*]   Require modules to be validly signed
        [*]   Automatically sign all modules
        Which hash algorithm should be signed with? (SHA-256)
```

You can press ‘spacebar’ to enable or disable settings.


kirin (kirin:0) - VNC Viewer
linux-stm32mp Configuration
File Edit View Search Terminal Help
sgmustadio@kirin: ~/P...
linux-stm32mp Config...
Thu Sep 16, 17:28

```
.config - Linux/arm 5.10.10 Kernel Configuration
^@ Device Drivers ^@ Staging drivers ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
                              Staging drivers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc>
  to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module
  capable

              --- Staging drivers
              < >   Prism2.5/3 USB driver (NEW)
              < >   Data acquisition support (comedi) (NEW)
              < >   Support for rtllib wireless devices (NEW)
              < >   Realtek RTL8723BS SDIO Wireless LAN NIC driver (NEW)
              < >   RealTek RTL8712U (RTL8192SU) Wireless LAN NIC driver (NEW)
              <M>   Realtek RTL8188EU Wireless LAN NIC driver
              [ ]     Realtek RTL8188EU AP mode
              < >   VIA Technologies VT6656 support (NEW)
                    IIO staging drivers  --->
              [ ]   Media staging drivers (NEW)  ----
                    Android  ----
              [ ]   Staging Board Support (NEW)
              < >   GCT GDM724x LTE support (NEW)
              < >   Xilinx FPGA firmware download module (NEW)
          (+)

           <Select>    < Exit >    < Help >    < Save >    < Load >
```

Note that these settings only apply to the rtl8188eu chipset! If you are using a different WiFi adapter, the settings will likely be different. It may take some time (days or weeks) to figure out exactly which modules you need to enable to get your WiFi working on your build. Sometimes, the adapter won’t work at all with Linux, so be patient and try several different adapters.

Make Kernel Changes Permanent

Right now, the kernel changes will only be in effect for the current build. You can save the configuration settings to your custom layer so that they become more permanent whenever you build custom-image. To do that, create a default configuration file (defconfig) with the current settings:

Copy Code

```
bitbake -c savedefconfig virtual/kernel
```

Copy the defconfig file to your custom layer:

Copy Code

```
cd ../meta-custom
mkdir -p recipes-kernel/linux/files
cp ../build_mp1/tmp/work/stm32mp1-poky-linux-gnueabi/linux-stm32mp/5.10.10-r0/build/defconfig recip
```

Next, we need to edit the kernel recipe to include our new configuration file:

Copy Code

```
vi recipes-kernel/linux/linux-stm32mp_%.bbappend
```

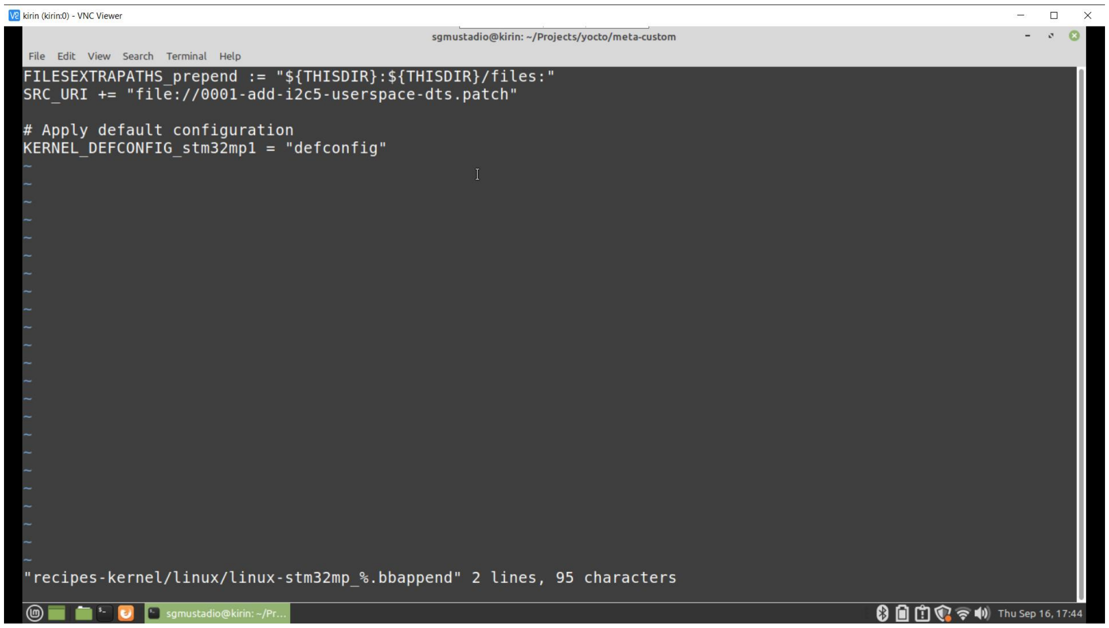
This should have a few lines from when we enabled the I2C port. Change the *FILESEXTRAPATHS_prepend* variable to include the new files directory in the search path. Additionally, add the *KERNEL_DEFCONFIG_<machine>* variable with the name of the defconfig file. The recipe should look like the following:

Copy Code

```
FILESEXTRAPATHS_prepend := "${THISDIR}:${THISDIR}/files:"
SRC_URI += "file://0001-add-i2c5-userspace-dts.patch"

# Apply default configuration
KERNEL_DEFCONFIG_stm32mp1 = "defconfig"
```

Note that the name ("defconfig") and location of the file (*files* directory) are very important! Do not change them--bitbake looks for a very particular filename when using a kernel configuration file from a layer. Additionally, the *KERNEL_DEFCONFIG* variable should have the *MACHINE* name appended to the end (e.g. "_stmp32mp1").


kirin (kirin:0) - VNC Viewer
sgmustadio@kirin: ~/Projects/yocto/meta-custom
File Edit View Search Terminal Help
sgmustadio@kirin: ~/Pr...
Thu Sep 16, 17:44

```
FILESEXTRAPATHS_prepend := "${THISDIR}:${THISDIR}/files:"
SRC_URI += "file://0001-add-i2c5-userspace-dts.patch"

# Apply default configuration
KERNEL_DEFCONFIG_stm32mp1 = "defconfig"
~
"recipes-kernel/linux/linux-stm32mp_%.bbappend" 2 lines, 95 characters
```

Save and exit.

In addition to enabling these modules in the kernel, we also need to include them along with some tools/firmware in our image. To do that, we must update the *IMAGE_INSTALL* variable. Also, we want to autoload the rtl8188eu driver (known as “r8188eu”) whenever Linux boots. This saves us from having to call modprobe after every reboot.

Edit the custom-image.bb recipe:

Copy Code

```
vi recipes-core/images/custom-image.bb
```

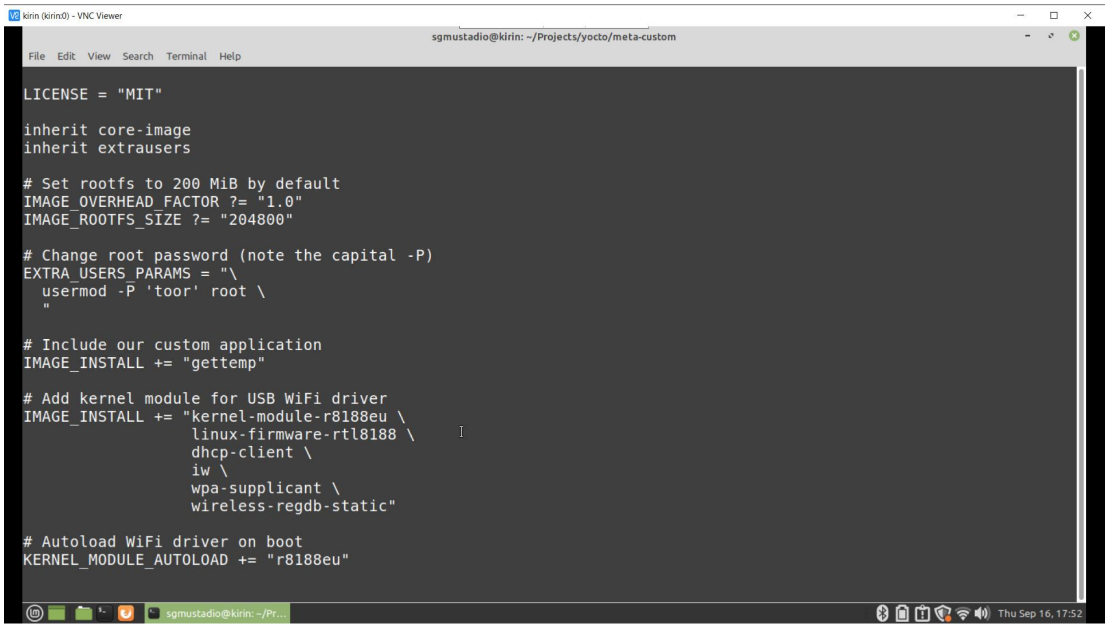
Add the following lines to the end:

Copy Code

```
# Add kernel module for USB WiFi driver
IMAGE_INSTALL += "kernel-module-r8188eu \
                  linux-firmware-rtl8188 \
                  dhcp-client \
                  iw \
                  wpa-supplicant \
                  wireless-regdb-static"

# Autoload WiFi driver on boot
KERNEL_MODULE_AUTOLOAD += “r8188eu”
```

Your recipe should look like the following:


kirin (kirin:0) - VNC Viewer
sgmustadio@kirin: ~/Projects/yocto/meta-custom
File Edit View Search Terminal Help
sgmustadio@kirin: ~/Pr...
Thu Sep 16, 17:52

```
LICENSE = "MIT"

inherit core-image
inherit extrausers

# Set rootfs to 200 MiB by default
IMAGE_OVERHEAD_FACTOR ?= "1.0"
IMAGE_ROOTFS_SIZE ?= "204800"

# Change root password (note the capital -P)
EXTRA_USERS_PARAMS = "\
  usermod -P 'toor' root \
  "

# Include our custom application
IMAGE_INSTALL += "gettemp"

# Add kernel module for USB WiFi driver
IMAGE_INSTALL += "kernel-module-r8188eu \
                  linux-firmware-rtl8188 \
                  dhcp-client \
                  iw \
                  wpa-supplicant \
                  wireless-regdb-static"

# Autoload WiFi driver on boot
KERNEL_MODULE_AUTOLOAD += "r8188eu"
```

Save and exit.

Build and Deploy

Our configuration should be complete, so it’s time to build the image:

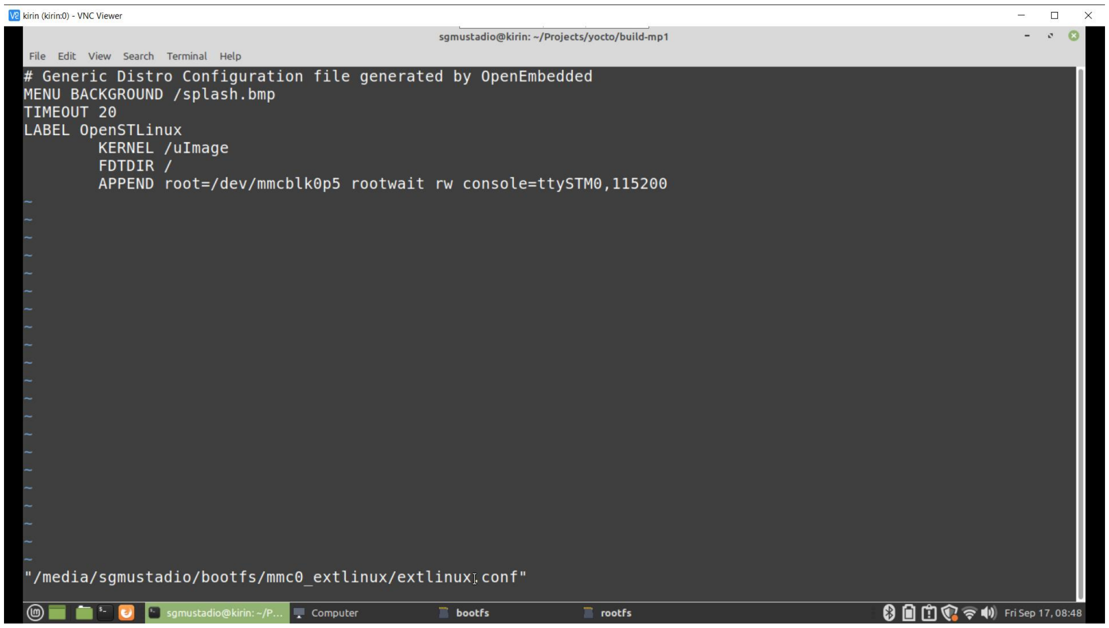
Copy Code

```
cd ../build-mp1
bitbake custom-image
```

This will take some time (1-2 hours likely), so be patient.

When the build process is finished, flash the bootfs and rootfs onto the SD card as shown at the end of Part 4 (we need to flash bootfs as we made changes to the kernel).

Don't forget to change bootfs/mmc0_extlinux/extlinux.conf so that root points to /dev/mmcblk05!



```
# Generic Distro Configuration file generated by OpenEmbedded
MENU BACKGROUND /splash.bmp
TIMEOUT 20
LABEL OpenSTLinux
        KERNEL /uImage
        FDTDIR /
        APPEND root=/dev/mmcblk0p5 rootwait rw console=ttySTM0,115200
"/media/sgmustadio/bootfs/mmc0_extlinux/extlinux.conf"
```

kirin (kirin:0) - VNC Viewer
sgmustadio@kirin: ~/Projects/yocto/build-mp1
File Edit View Search Terminal Help
sgmustadio@kirin: ~/P...
Computer
bootfs
rootfs
Fri Sep 17, 08:48

Test WiFi

Plug the SD card into your STM32MP157D-DK1 and boot it up. Connect to the serial terminal (you may need to change ttyACM0 to some other device file, depending on your particular flavor of Linux and hardware setup):

Copy Code

```
picocom -b 115200 /dev/ttyACM0
```

Log in to the board with "root" and "toor" as the password. List the available network interfaces:

Copy Code

```
ifconfig -a
```

Bring up your wireless interface:

Copy Code

```
ifconfig wlan0 up
```

Scan the available networks (this may or may not work with some chipsets):

Copy Code

```
iw wlan0 scan
```

Add your network credentials to the wpa_supplicant.conf file:

Copy Code

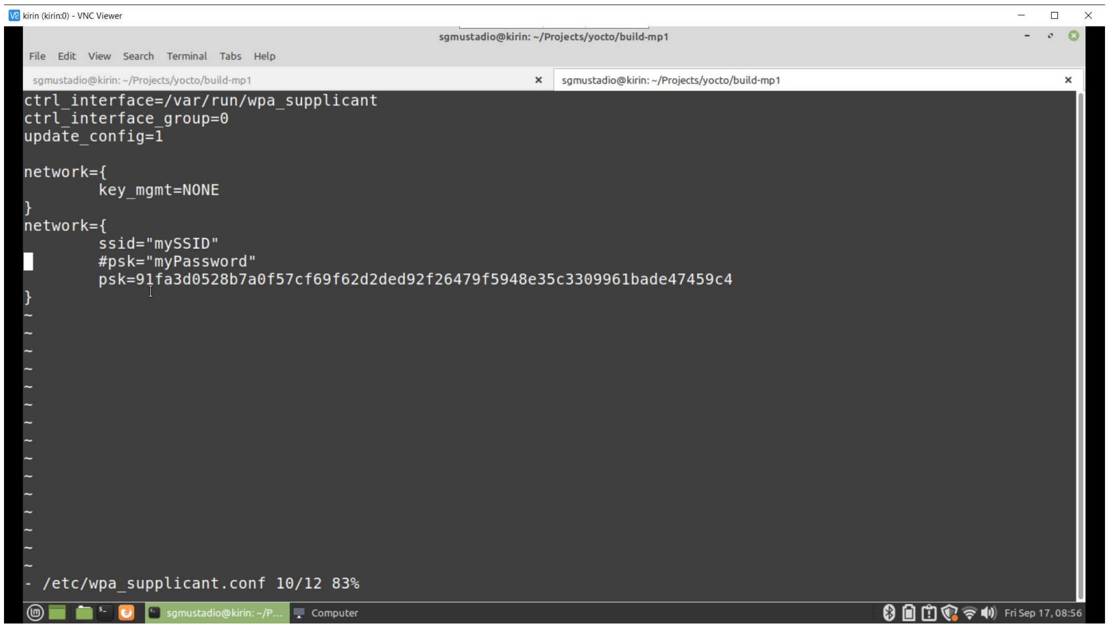
```
wpa_passphrase "<SSID>" "<WiFi Password>" >> /etc/wpa_supplicant.conf
```

I recommend removing the commended out “#psk” variable in the wpa_supplicant.conf file, as it contains your password in plaintext:

Copy Code

```
vi /etc/wpa_supplicant.conf
```

As shown in the screenshot, remove the #psk… line.


kirin (kirin:0) - VNC Viewer
sgmustadio@kirin: ~/Projects/yocto/build-mp1
File Edit View Search Terminal Tabs Help
sgmustadio@kirin: ~/Projects/yocto/build-mp1
sgmustadio@kirin: ~/Projects/yocto/build-mp1
sgmustadio@kirin: ~/P...
Computer
Fri Sep 17, 08:56

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
        key_mgmt=NONE
}
network={
        ssid="mySSID"
        #psk="myPassword"
        psk=91fa3d0528b7a0f57cf69f62d2ded92f26479f5948e35c3309961bade47459c4
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
- /etc/wpa_supplicant.conf 10/12 83%
```

Run the wpa_supplicant tool:

Copy Code

```
wpa_supplicant -B -i wlan0 -D wext -c /etc/wpa_supplicant.conf
```

Check the link status to see if your board is connected to your access point (this may or may not work depending on your particular chipset):

Copy Code

```
iw dev wlan0 link
```

Get an IP address from your DHCP server (likely your router):

Copy Code

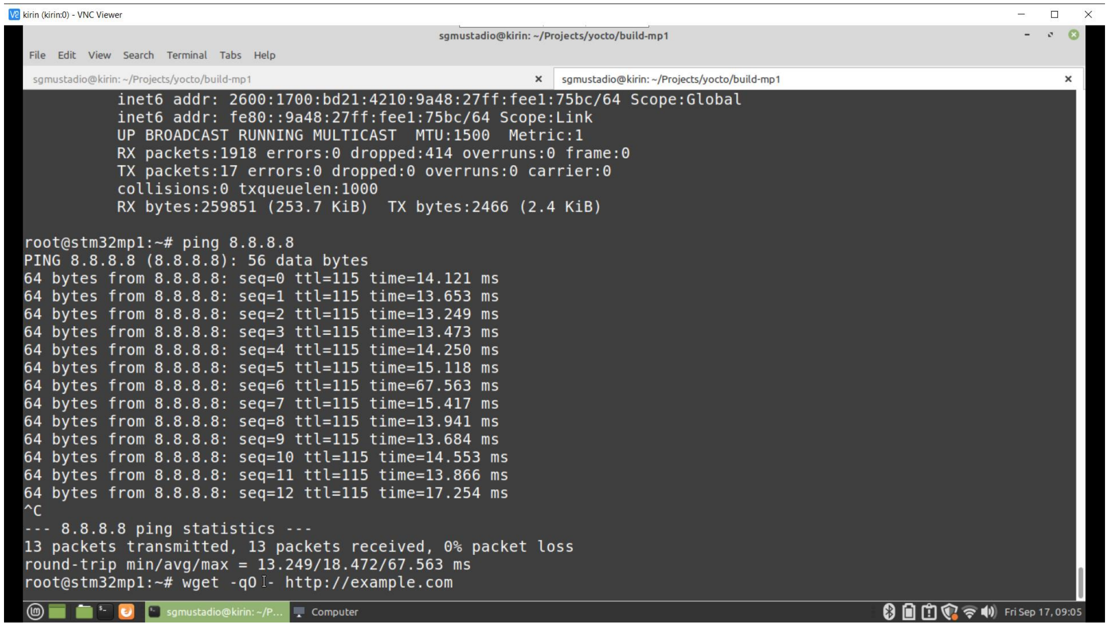
```
dhclient wlan0
```

Do network stuff!

Copy Code

```
ping -c 3 8.8.8.8
wget -qO - http://example.com
```

If all goes well, you should be able to communicate with other devices/servers on the Internet.


kirin (kirin:0) - VNC Viewer
sgmustadio@kirin: ~/Projects/yocto/build-mp1
File Edit View Search Terminal Tabs Help
sgmustadio@kirin: ~/Projects/yocto/build-mp1
sgmustadio@kirin: ~/Projects/yocto/build-mp1
sgmustadio@kirin: ~/P...
Computer
Fri Sep 17, 09:05

```
          inet6 addr: 2600:1700:bd21:4210:9a48:27ff:fee1:75bc/64 Scope:Global
          inet6 addr: fe80::9a48:27ff:fee1:75bc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1918 errors:0 dropped:414 overruns:0 frame:0
          TX packets:17 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:259851 (253.7 KiB)  TX bytes:2466 (2.4 KiB)

root@stm32mp1:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=115 time=14.121 ms
64 bytes from 8.8.8.8: seq=1 ttl=115 time=13.653 ms
64 bytes from 8.8.8.8: seq=2 ttl=115 time=13.249 ms
64 bytes from 8.8.8.8: seq=3 ttl=115 time=13.473 ms
64 bytes from 8.8.8.8: seq=4 ttl=115 time=14.250 ms
64 bytes from 8.8.8.8: seq=5 ttl=115 time=15.118 ms
64 bytes from 8.8.8.8: seq=6 ttl=115 time=67.563 ms
64 bytes from 8.8.8.8: seq=7 ttl=115 time=15.417 ms
64 bytes from 8.8.8.8: seq=8 ttl=115 time=13.941 ms
64 bytes from 8.8.8.8: seq=9 ttl=115 time=13.684 ms
64 bytes from 8.8.8.8: seq=10 ttl=115 time=14.553 ms
64 bytes from 8.8.8.8: seq=11 ttl=115 time=13.866 ms
64 bytes from 8.8.8.8: seq=12 ttl=115 time=17.254 ms
^C
--- 8.8.8.8 ping statistics ---
13 packets transmitted, 13 packets received, 0% packet loss
round-trip min/avg/max = 13.249/18.472/67.563 ms
root@stm32mp1:~# wget -qO - http://example.com
```