Overview

Get the kernel source

The kernel source should be downloaded from the Odroidw-3.12.y branch of Linux section on GitHub of hardkernel

Create our working directory. (for example, /home/$USER/linux)

```
mkdir -p /home/$USER/linux
cd /home/$USER/linux
```

you can download the source directly using Git. for the 3.12 branch:

```
git init
git clone --depth 1 git://github.com/hardkernel/linux.git
```

And for the other stable code branch, change the numbers in the following to suit:

```
git init
git fetch git://github.com/hardkernel/linux.git
odroidw-3.12.y:refs/remotes/origin/odroidw-3.12.y
git checkout odroidw-3.12.y
```

Or you can download a tarball from the same website:

Get a compiler

Next, you will need to get a version of GCC in order to build the kernel.

On the Odroid W (Native Compile)

Raspbian and Pi Bang

```
apt-get update
apt-get -y
dist-upgrade apt-get -y install gcc make bc screen ncurses-dev
```

Cross compiling from Linux

Please note that when cross-compiling, your compiler may not target the correct ARM processor by default. This will at best reduce performance, or worse, compile for a much newer processor resulting in illegal instructions in your code. The pre-built compiler or a custom-built compiler are recommended because of this. (For example, the latest GCC Linaro binary targets armv7-a by default, whereas the Raspberry Pi requires armv6kz). It is possible to add extra compiler options to the HOSTCFLAGS line in Makefile. The correct flags are shown on the software page - note that you may also need to add -marm if your compiler produces Thumb code by default.

Use the provided compiler

Download the pre-built bmc2708 compiler from the Raspberry Pi tools section on GitHub.

```
git clone git://github.com/raspberrypi/tools.git --depth 1
```

Or you can download a tarball from the website using this link.

Custom-built Linaro GCC

See Linaro GCC Compilation.

Ubuntu

```
apt-get install gcc-arm-linux-gnueabi make ncurses-dev
```

Perform the compilation

Firstly, ensure your build directory is clean:

```
make mrproper
```

Next, in all cases, you will want to get a working kernel configuration to start from. You can get the one running on the Odroid W by typing the following (on the Odroid W):

```
zcat /proc/config.gz > .config
```

Then copy .config into your build directory.

Alternatively, the default configuration is available in the downloaded kernel source in

arch/arm/configs/odroidw_defconfig. Just copy this to .config in the build directory.

From this point on, if you are cross-compiling, set an environment variable CCPREFIX that points to the prefix of your compiler binary as each compiler will be named slightly differently.

```
export CCPREFIX=/path/to/your/compiler/binary/prefix-of-binary-
```

If you are building on the Odroid W, remove ARCH=arm CROSS_COMPILE=${CCPREFIX} from each command.

Ensure that your configuration file is up-to-date:

```
make ARCH=arm CROSS_COMPILE=${CCPREFIX} oldconfig
```

If any configuration options have been added, you will be asked what set each option to. If you don't know the answer, just press enter to accept the default.

Optionally, if you want to make changes to the configuration, run this next:

```
make ARCH=arm CROSS_COMPILE=${CCPREFIX} menuconfig
```

Now you are ready to build: (On the Odroid W, type 'screen' to open a virtual screen. If you use it you can disconnect from the Odroid W and compile overnight...)

```
make ARCH=arm CROSS_COMPILE=${CCPREFIX}
```

If you are on a multi-core system, you can make the build faster by appending -j<N> where <N> is the number of cores on your system plus one (that is, -j3 for two cores).

Find something else to get on with while the compilation takes place. On an average PC with the default configuration, this should take about 15 minutes.

The modules will be build with the following command.

```
make ARCH=arm CROSS_COMPILE=${CCPREFIX} modules
```

Transfer the build

The fully built kernel will be arch/arm/boot/zImage. Copy your new kernel file(kernel.img) into the Odroid W boot partition, though preferably as a new file (such as kernel_new.img) just in case it doesn't work. If you're building on the Odroid W, just copy the file to /boot. If you use a different filename, edit config.txt change the kernel line:

```
kernel=kernel_new.img
#kernel=kernel.img
```

Now you need to transfer the modules. Set an environment variable that points to a temporary module path.

```
export MODULES_TEMP=~/modules
```

In the build directory, run the following command:

```
make ARCH=arm CROSS_COMPILE=${CCPREFIX} INSTALL_MOD_PATH=${MODULES_TEMP}
modules_install
```

The contents of this directory, a single

```
lib
```

directory, should then be copied into the Odroid W root directory, merging or overwriting

```
/lib
```

NOTE: If you have rebuilt the new kernel with exactly the same version as the one that's running, you'll need to remove the old modules first. Ideally this should be done offline by mounting the uSD card on another system.

NOTE: The lib directory will have symlinks back to the kernel sources (lib/modules/<kernel-version>/source and lib/modules/<kernel-version>/build). If you have limited space on the uSD card and don't intend to compile modules on the Odroid W itself, you will probably want to remove those links before you transfer the lib directory. The size difference can be many hundreds of MB.

Your Odroid W should now be ready to boot the new kernel.