jamie.lentin.co.uk

D-Link DNS-320 & DNS-325 NAS: Replacing Firmware

Replacing U-boot means you can boot from kernels stored on a UBIFS filesystem, as well as reliably booting off USB sticks. Whilst a somewhat scary procedure, the kirkwood SoC supports booting via. the serial port, so we cannot brick the device this way.

Running u-boot via Serial ↑

This isn't strictly necessary, but it's definitely worth checking you can boot via. serial first, so you know you can unbrick the NAS. There is mainline DNS-325 support, but DNS-320 isn't accepted just yet. As a result you need to use my branch.

1. Compile u-boot.kwb

```
host:~$ git clone -b dns320_support https://github.
host:~$ cd u-boot
host:u-boot$ export CROSS_COMPILE=arm-linux-gnueab
# Obviously, use dns325_config for a DNS-325
host:u-boot$ make distclean && make dns320_config
```

2. Turn NAS off
3. Close any open terminal window to the NAS
4. Get your PC ready to boot

```
# Replace /dev/ttyUSB1 with your serial port
host:u-boot$ ./tools/kwboot -p -b u-boot.kwb -B115
```

5. Turn NAS on. Be patient, it can take some time to recognise the boot message and sometimes halt with `xmodem: Bad message`. Lowering baud rate does not help.
6. Be ready to interrupt u-boot from booting.

The DNS-325 is a lot better at doing this than the DNS-320. You may have more luck with kwuartboot. kwuartboot cannot patch an image to boot from UART (the -p option). Instead you have to recompile with "BOOT_FROM uart" set in `board/d-link/dnskw/kwbimage.dns320.cfg`

At this point, you should have a u-boot prompt but the default environment. You might want to play around a bit to see what

contact me

you can do, there are notes at the bottom of the page.

Writing a new u-boot image ↑

NB: Newer stock versions of u-boot cannot boot the original D-link kernels. I'm not sure why this is but if this is likely to be an issue for you then don't do this, or at very least keep a backup.

You can either compile the image with the following...

1. Ensure you have "BOOT_FROM nand" set in `board/d-link/dnskw/kwbimage.dns320.cfg`
2. Compile u-boot.kwb

```
host:u-boot$ export CROSS_COMPILE=arm-linux-gnueab
# Obviously, use dns325_config for a DNS-325
host:u-boot$ make distclean && make dns320_config
```

...or download it. Here are 2 kwb images for the DNS-320 and DNS-325 (unlike the kernel, there isn't a dramatic advantage in having the latest and greatest, so thought I'd upload these here).

1. Copy the u-boot.kwb onto a ext2 USB stick, or somwhere TFTP-able.
2. Using u-boot, load it into memory and write it to the NAND.

```
Marvell>> usb reset ; ext2load usb 0:1 0x1000000 /u
Marvell>> nand erase 0x000000 0xe0000
Marvell>> nand write 0x1000000 0x000000 0xe0000
Marvell>> reset
```

3. Cross fingers! Note that the new u-boot will still countdown so be ready to interrupt the process.
4. At this point your u-boot environment should be very minimal. You will want to set at least the following.

```
=> setenv ethaddr 00:50:43:xx:xx:xx [Any MAC addres
=> setenv ipaddr [Our IP address, e.g. 10.150.1.3]
=> setenv serverip [Where TFTP server is, e.g. 10.1
=> setenv mtdparts 'mtdparts=orion_nand:896k(u-boot
Marvell>> saveenv
Saving Environment to NAND...
Erasing Nand...Writing to Nand... done
```

Building a root filesystem image ↑

Follow the the instructions on the keeping original frimware.

```
I have no name!@host:/# cat <<EOF > etc/fstab
/dev/root   /       ubifs   sync,noatime    0 0
EOF
```

If you have replaced u-boot, then you no longer have to append the devicetree file to the kernel image:

```
I have no name!@host:/# cat <<'EOF' > etc/kernel/postins
#!/bin/sh -e
# passing the kernel version is required
version="$1"
[ -z "${version}" ] && exit 0

cp /usr/lib/linux-image-${version}/kirkwood-dns*.dtb /bo

/usr/bin/mkimage -A arm -O linux -T kernel -C none -n ul
                 -a 0x00008000 -e 0x00008000 \
                 -d /boot/vmlinuz-${version} /boot/uIma
ln -sf /boot/uImage-${version} /boot/uImage

/usr/bin/mkimage -A arm -O linux -T ramdisk -C gzip -n u
                 -a 0x00e00000 -e 0x00e00000 \
                 -d /boot/initrd.img-${version} /boot/ul
ln -sf /boot/uInitrd-${version} /boot/uInitrd
EOF
I have no name!@host:/# chmod a+x /etc/kernel/postinst.d
# Recreate the kernel package to create the images to bo
I have no name!@host:/# dpkg-reconfigure $(dpkg --get-se
```

Creating a UBIFS image

UBIFS is a filesystem specifically designed for NAND devices. UBIFS is the filesystem, which lives inside a UBI volume. UBI is a volume manager similar to LVM.

To get it to fit you'll have to to tighten your belt somewhat. Note we loaded a bunch of modules into the initramfs, so our system won't be completely useless:

```
host:# mkdir cruft ; mv \
        rootfs/usr/share/man \
        rootfs/usr/share/doc \
        rootfs/lib/modules \
        cruft/
host:# cat <<EOF > rootfs/etc/dpkg/dpkg.cfg.d/excludes
path-exclude=/usr/share/man/*
path-exclude=/usr/share/doc/*
EOF
```

Once installed you may want to make some of the above available again.

We want to turn the filesystem generated into a UBIFS image. The short version is:

```
host:~$ /usr/sbin/mkfs.ubifs -v -r rootfs -m 2048 -e 12
```

If you want to know more about what is going on here, my friend has a much more detailed page. Although there is no need to prepare an initrd or use ubinize, our new u-boot can make our lives easier.

We now need to get this onto the NAND. Copy it onto your USB stick or somewhere else u-boot can get at it.

Formatting NAND with UBIFS ↑

Like HDDs, NAND chips can be divided into partitions. However, the table itself isn't stored on the device, it's a configuration option. By default D-Link by default divide it into 6, which is what the mainline kernel does also.

We only want 2 partitions, one for u-boot, the other for the UBIFS filesystem. We set this in `mtdparts`, and tell both u-boot and the Linux kernel:

```
=> setenv mtdparts 'mtdparts=orion_nand:896k(u-boot),128
=> setenv bootargs 'console=ttyS0,115200 ${mtdparts}'
=> saveenv
Saving Environment to NAND...
Erasing Nand...Writing to Nand... done
```

We can now format the rest of the NAND with UBI, make a volume within this, and write our UBIFS image into it:

```
# NB: If you already have UBI installed on your NAND, th
# to reinstall it, as the erase counters will not be pre
=> nand erase.part root
=> ubi part root
 . . .
UBI: empty MTD device detected
UBI: create volume table (copy #1)
UBI: create volume table (copy #2)
 . . .
=> ubi create rootfs
No size specified -> Using max size (129153024)
Creating dynamic volume rootfs of size 129153024
=> usb reset ; ext2load usb 0:1 0x1000000 /ubifs.img
 . . .
Loading file "/ubifs.img" from usb device 0:1
60512256 bytes read
=> ubi write 0x1000000 rootfs 0x[60512256 in hex]
60512256 bytes written to volume rootfs
=> ubifsmount rootfs # or ubifsmount ubi:rootfs with new
UBIFS: mounted UBI device 0, volume 0, name "rootfs"
```

Alternatively, if the NAS is already booted, you can use `ubinize` and `ubiformat` to re-flash the filesystem image.

Finally, reboot and modify U-Boot environment so that loads the kernel from our new FS:

```
=> setenv root_flash 'ubi.mtd=root root=ubi0:rootfs root
=> setenv load_net 'tftp 0x0900000 kirkwood-dns320.dtb ;
=> setenv load_ubifs 'ubi part root ; ubifsmount rootfs
# Or, with newer u-boot revisions... "ubifsmount ubi:roo
=> setenv console 'ttyS0,115200'
=> setenv optargs ''
=> setenv bootcmd 'setenv bootargs console=${console} $
=> saveenv
Saving Environment to NAND...
Erasing Nand...Writing to Nand... done
```

Now you should be booting Linux, with no remains of D-Link (well, beyond the massive letters on the side).

Saving space ↑

You've probably noticed that the flash chip is 128MB, and the above squeezes a full-fat Linux distribution onto it. This works, but be wary of what you install. The following also help

- Move `/var/lib/apt`, `/var/lib/dpkg`, `/var/cache/apt`, `/var/cache/debconf` onto a partition on the HDD. They are only needed when running apt-get, and can get pretty big.
- Stop syslogd, log over the network to a different host, or use busybox-syslogd's ring buffer.
- You may want some swap space on the HDDs for when running memory intensive processes, e.g. fsck.

Network logging

Logging onto the NAND will both use up valuable space and cause lots of churn on the device. Turning syslog off also works, but is a bit annoying. If you have another device on the network you can send logs to, then you can keep logs going.

On the NAS, edit `/etc/syslog.conf` so that the only lines are:

```
*.emerg                         *
*.*                             @[log server IP / hostna
```

On the log server, enable receiving logs with `SYSLOGD="-r"` in `/etc/default/syslogd`.

Next Steps ↑

- Userland configuration

Appendix: Using a mainline u-boot ↑

A few tips on how to do things in u-boot.

Reading USB sticks

Firstly run "usb start":

```
=> usb start
(Re)start USB...
USB:   Register 10011 NbrPorts 1
USB EHCI 1.00
scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Dev
=> usb storage
  Device 0: Vendor: pqi      Rev: 1100 Prod: Intelligent
            Type: Removable Hard Disk
            Capacity: 7725.0 MB = 7.5 GB (15820800 x 512
```

Then you can use ls or load commands for your filesystem. e.g. for ext2:

```
=> ext2ls usb 0:1
<DIR>       4096 .
<DIR>       4096 ..
<DIR>      16384 lost+found
         3029296 dlink-nas.img
=> ext2load usb 0:1 0x1000000 dlink-nas.img
```

And `fatls` and `fatload` for FAT-formatted sticks.

Reading UBIFS partitions

Assuming your partitions are labelled as above, this should work:

```
=> ubi part root
Creating 1 MTD partitions on "nand0":
0x000000300000-0x000008000000 : "mtd=3"
. . .
=> ubifsmount rootfs
UBIFS: mounted UBI device 0, volume 0, name "rootfs"
=> ubifsls
<DIR>        5648  Sat Mar 24 14:43:26 2012  bin
```

And you can use ubifsload to fetch files into memory.

Netconsole

u-boot also supports a console over the network instead of the

serial port. To set this up, you need to point it at a particular host on your network then activate. The following points it at the same IP as the TFTP server:

```
=> set ncip ${serverip} ; set stdin nc; set stdout nc; s
```

U-boot has a helper program to allow you to connect to your NAS

```
host:u-boot$ ./tools/netconsole 10.150.1.2
```

Appendix: Compiling a kernel ↑

You don't need to compile a kernel any more, but just in case you want to anyway here are some notes. Any Linux 3.6+ kernel will have support for the NASes built in. You will need an ARM cross compiling toolchain, see instructions here:

```
host:~$ cd ${kernel_source}
# Set up cross-compiler
host:linux-2.6$ alias cross-make='make ARCH=arm CROSS_C
# Configure kernel
host:linux-2.6$ cat arch/arm/configs/kirkwood_defconfig
### Extra options atop kirkwood_defconfig
CONFIG_MACH_DLINK_KIRKWOOD_DT=y
USE_OF=y
CONFIG_SERIAL_OF_PLATFORM=y
CONFIG_HWMON=y
CONFIG_SENSORS_GPIO_FAN=y
CONFIG_MTD_UBI=y
CONFIG_UBIFS_FS=y
EOF
host:linux-2.6$ cross-make menuconfig # If you need to
# Compile
host:linux-2.6$ cross-make -j5 uImage modules kirkwood-
```

Copy a kernel into the filesystem for the NAS:

```
host:linux-2.6$ cp arch/arm/boot/uImage rootfs/boot/uIma
host:linux-2.6$ cp arch/arm/boot/kirkwood-dns320.dtb roo
host:linux-2.6$ cp arch/arm/boot/kirkwood-dns325.dtb roo
host:linux-2.6$ cross-make -j5 modules
host:linux-2.6$ cross-make INSTALL_MOD_PATH=rootfs modul
```

And use symlinks to specify which kernel we want to boot by default:

```
host:linux-2.6$ cd rootfs/boot
host:boot$ ln -s uImage-${version} uImage
host:boot$ ln -s kirkwood-dns320-${version}.dtb kirkwoo
host:boot$ ln -s kirkwood-dns325-${version}.dtb kirkwoo
```