QNAP Homebrew

17/01/24

Overview

The QNAPHomebrew project is aimed to reuse an old TS412 (and similar) as a simple basic NAS, goals include:

- Reuse of old (not current hardware)
- Minimal use of (electrical) power
- Reusing older (mismatched) disks
- Support for SMB (Samba), NFSv3 and DLNA
- Near silent operation
- Support for Modern Linux Kernel

For background the author owns a TS412 and a TS451+ . The latter is a more modern Intel based system however recent “upgrades” have left the system unusable

- Printer support dropped
- Recent changes make photo storage unworkable
- Failing processes like LDAP backup cannot be disabled
- Various debug options and similar settings means the HDD are constantly “busy”
- The busy disk leads to higher temperatures and power consumption

Having spent many man-days dealing with QNAP support trying to get these resolved the author finally decided a better route was to switch back to ARM based TS412 but running a more modern Linux Kernel and corresponding utilities.

The QNAPHomebrew project has three parts, each of which can be used in isolation and may prove useful on other systems.

- An alternative Flash setup and system of booting
- A modified version of **qcontrol(1)** and the fan hysteresis settings
- A set to scripts to setup and run the system as a NAS

For each of these there are choice:

Flash & Booting

The original QTS partitioning and booting system cannot support recent default Debian kernels, so some change is required here. Possibilities include:

1. Produce a smaller kernel (and initrd)
2. Develop a different manner of booting (e.g via network or HDD)
3. Reconfigure/repurpose the FLASH layout

The solution offered by QNAPHomebrew offers options for 2 & 3.

qcontrol and hysteresis settings

There is a modified version of the **qcontrol(8)** command which exposes the internal data, such as temperature an fan speed. The hoped for use of this was to display these data on the LCD (the TS412 lacks an LCD and the author failed to add one, others may be more successful). The hysteresis settings allowed the system to run at a slightly higher temperature, while allowing the fan to stop completely.

NAS scripts

This is a relatively small set to scripts which allow the system to support NFSv3, SAMBA and DLNA protocols. This is pretty common and minimal set needed for a NAS. The setup supports the system normally “booting” from a small external SSD and able to “boot” from ANY single HDD. So if the system loses 3 of the 4 HDD and the external SSD it will still boot, albeit with not all the data being available.

A different flavour of NAS

The authors TS412 runs all the parts of the QNAPHomebrew, the “flavour” is not the same as the QTS system, or indeed things like TrueNAS or MediaTomb. One thing to consider is using some parts of this software, like qcontrol & flash config with another NAS implementation like TrueNAS. The author’s systems:

- Has 4 HDD (internal) and an external SSD plugged into one of the eSATA ports
  - Tray1=SAMSUNG HD103UJ (1TB)
  - Tray2= SAMSUNG HD154UI (1.5TB)
  - Tray3= HGST HDN724040ALE640 (4TB)
  - Tray4= SAMSUNG HD154UI (1.5TB)
  - eSATA1=KINGSTON SA400S3 (500GB)
- Many of the same filesystems as QTS exist “homes” is on a RAID filesystem, (across all 4 disks) but other filesystems are **non-RAID**.
- Typically all the disks spin down an hour after booting
- The fan is either stopped or “silent” (very slow mode)
- Power consumption is 13W
- The system usually boots via BOOTP and does not access Flash memory
- The Flash memory is only infrequently updated, most updates are made to the BOOTP server.
- The Flash uses the M11 layout

The system uses old hardware, old HDD, typically removed from another NAS, for example one of the Samsung disks has SMART errors and reports an unrecoverable sector. The system is intended to be used for low volume data, such that it can be left on 24X7 while using very little power.

Flash & Booting

When the TS412 is powered on (and the RESET PIN is NOT depressed) the system loads and executes UBOOT code (from flash) and reads the UBOOT config. This determines how UBOOT performs. In the original QTS setup it loads the kernel (from mtd1) into RAM at address 0x800000 and initrd (from mtd2) then transfers control to the kernel. This boots QTS, which then reads NAS Config (mtd5) to set various QTS variables (e.g. the MAC address of interfaces). One issue with this layout is that the partition called Kernel (mtd1) is only 2mb in size, which is too small to hold the default kernel for recent kernel (Bullseye onward).

Various solutions exist for the FLASH layout (be aware it is possible to boot without using the flash at all, thus the layout is irrelevant) in overview the solutions are:

Name	Kernel Size	Initrd size	PiXE recovery	Support Bullseye
Original	2MB	9MB	Yes	No
Saboteur	3MB	9MB	Yes	Yes
Mouiche	3MB	12MB	No	Yes
M11	3MB	11MB	Yes	Yes

Full Memory Map (Flash & RAM) original

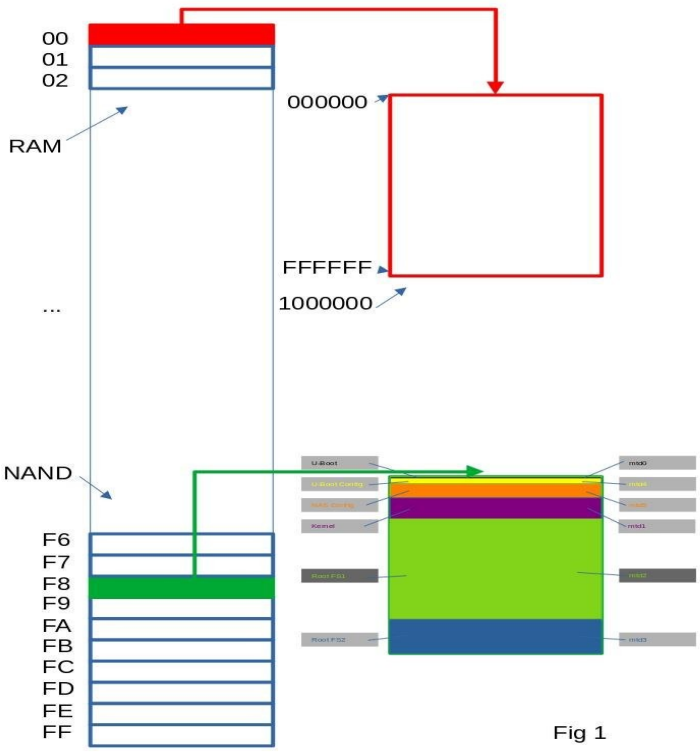


Fig 1

The **Saboteur layout** simply stores (and loads) the kernel in mtd3 (RootFS2) instead of mtd1 (Kernel). It is described in .https://forum.qnap.com/viewtopic.php?p=847122 This is probably the most straightforward solution for most people.

Original Partition names and sizes:

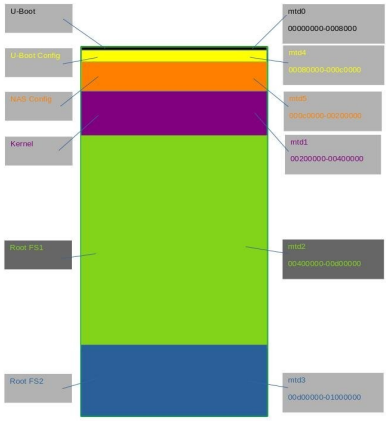


Fig 2

Relocation of kernel & initrd using **Saboteur layout**

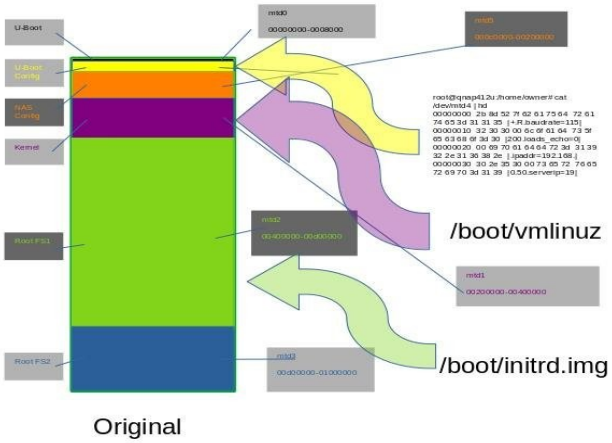


Fig 3

The next solution is described https://github.com/amouiche/qnap_mtd_resize_for_bullseye . This creates the biggest initrd. However in achieving this it move the kernel stored in flash back 1MB. to 0xF8100000 rather than 0xF8200000, a side effect of this is that **it is no longer possible to use PiXE recovery** (https://forum.qnap.com/viewtopic.php?t=171411). If you are able to use PiXE recovery then losing this ability is a major shortcoming of this layout. The Mouiche layout does however come with a script to do all the heavy lifting.

The Mouiche layout verses the original layout.

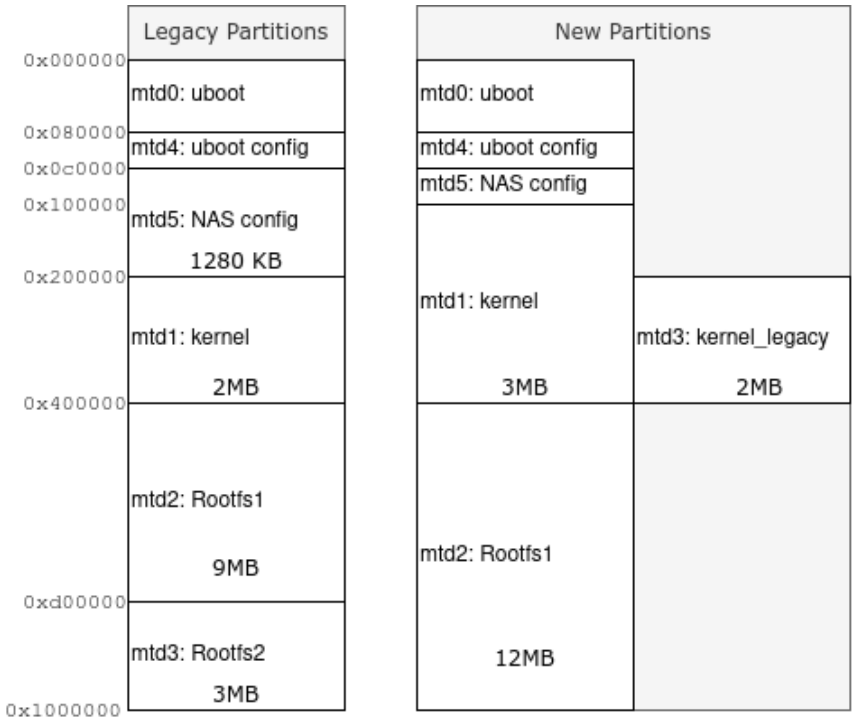
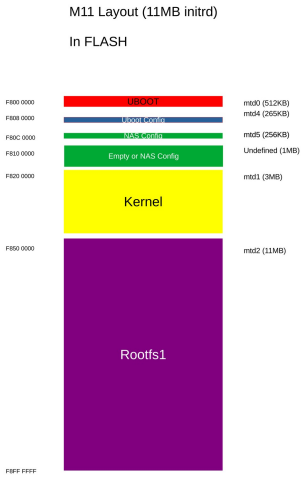


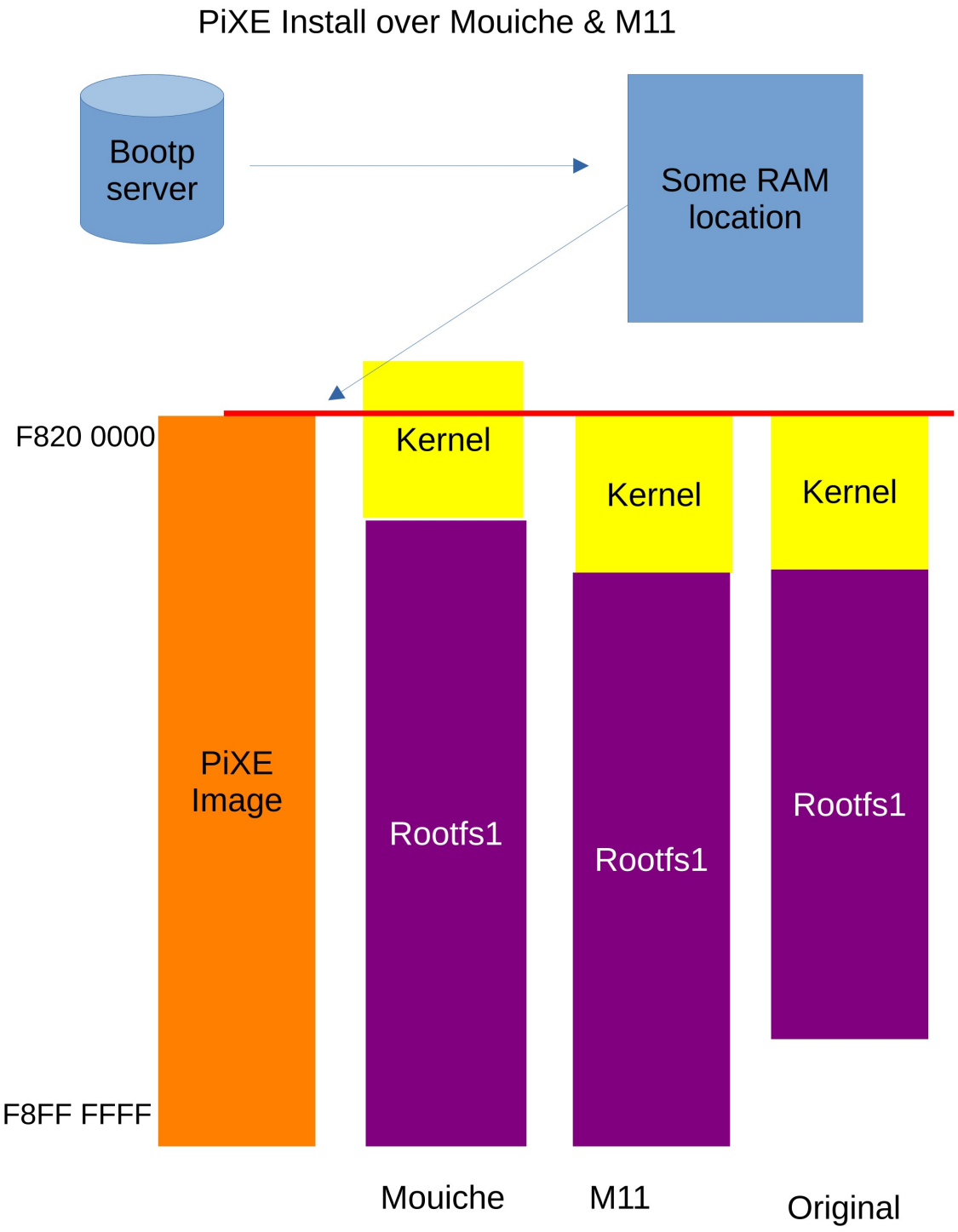
Figure 4

The final layout is called M11 because it has an initrd (mtd2/Rootfs1) sized at 11MB. It leaves most partitions in place and simply joins Rootfs1 & Rootfs2 into a single partition and moves it down 1MB to allow for the bigger Kernel.



The main reason to switch from the Mouiche layout to the M11 is that fact that PiXE recovery is still possible.

Since the Mouiche layout has some data (start of Kernel) before 0xF8200000 so this would not get



Then you need something like screen(1) or mincom(1) to use the device.

To use BOOTP (or PiXE recovery) you need a DHCP server and a TFTP server. Fortunately dnsmasq(8) can fulfil both these roles (at least as far as this usecase goes). The author uses a Debian bullseye system running dnsmasq 2.85. The file /etc/dnsmaq.conf is empty and there are 2 files in /etc/dnsmasq.d:

- 01local.conf
- 02PXE

As is common in many home systems 01local.conf defines all the devices (phone, *Alexas*, doorbells etc) including the NAS. The 02PXE contains just the extra bits needed here for BOOTP and PiXE. An alternative arrangement might be to define everything about the NAS in a single file say 03NAS. So **part** of 01local,conf is:

01local.conf

```
# Original Motherboard
dhcp-host=00:08:9b:cb:8d:b4,set:TS412,          10.117.1.251,QNAP412U,24h
dhcp-host=00:08:9b:cb:8d:b5,set:TS412,          10.117.1.252,QNAP412,24h

# Replacement Motherboard
dhcp-host=00:08:9b:cc:cb:ca,set:TS412,          10.117.1.232,TS412,24h
dhcp-host=00:08:9b:cc:cb:cb,set:TS412,          10.117.1.233,TS412B,24h
# Hardcoded MACs inside uboot config
dhcp-host=00:50:43:3c:3b:5d,set:TS412,          10.117.1.234,TS412C,24h
dhcp-host=00:11:88:0f:62:81,set:TS412,          10.117.1.235,TS412D,24h
dhcp-host=00:00:00:EE:51:81,set:TS412,          10.117.1.236,TS412E,24h


dhcp-host=24:5E:BE:2D:98:4D,set:TS451P,         10.117.1.161,QNAP451PLUS,24h
dhcp-host=24:5E:BE:2D:98:4E,set:TS451P,         10.117.1.162,QNAP451PLUS2,24h
```

The key part here is the "**set TS412**" which defines a "tag" essentially saying "this is a TS412, this is user later in 02PXE (this is complete file):

02PXE

```
enable-tftp
tftp-root=/srv/tftp

#
# Need to create:
#
#  /srv/tftp/F_TS-412-QTS
#  /srv/tftp/F_TS-412-stretch
#  /srv/tftp/F_TS-412-SABOTEUR-STRETCH
#  /srv/tftp/F_TS-412-SABOTEUR-BULLSEYE
#


# Any deviced tagged as a TS412 (we only have one) gets told to ask for this image (only used when RESET pin
activated)
#
# Note for the MOUICHE layout, PiXE recovery is NOT possible (because it starts 010000 and The PiXE is fixed at
020000)
# but if you plan to use the SABOTER laybout you need to change bootargs and bootcmd
#
#dhcp-boot=tag:TS412,F_TS-412-SABOTEUR-BULLSEYE
#dhcp-boot=tag:TS412,F_TS-412-MOUICHE_BULLSEYE
#dhcp-boot=tag:TS412,F_TS-412-MOUICHE_BOOKWORM
dhcp-boot=tag:TS412,F_TS-412-M11_BOOKWORM
```

So 02PXE enables the TFTP server in dnsmasq and defines /srv/tftp as the place files will be served from. The dhcp-boot line matches requests which are tagged as TS412 (see 01local.conf) and for them offers the boot filename of F_TS-412-M11_BOOKWORM. Looking in /srv/ftp we see:

```
  /srv/tftp:
  total used in directory 237740 available 21.9 GiB
  drwxr-xr-x 2 root   root         4096 Dec  8 00:07 .
  lrwxrwxrwx 1 root   root           61 Dec  8 00:07 dl-F_TS-412-M11_BOOKWORM -> /srv/tftp/dl-F_TS-412-M11_bookworm_6.1.0-13-
marvell
  lrwxrwxrwx 1 root   root           48 Dec  8 00:06 F_TS-412-M11_BOOKWORM -> /srv/tftp/F_TS-412-M11_bookworm_6.1.0-13-marvell
  -rw------- 1 graeme everyone 16777216 Dec  8 00:04 F_TS-412-M11_bookworm_6.1.0-13-marvell
  -rw-r--r-- 1 graeme everyone 14680064 Dec  7 23:48 dl-F_TS-412-M11_bookworm_6.1.0-13-marvell-sans-LVM2
  lrwxrwxrwx 1 root   root           54 Sep 10 18:49 F_TS-412-SABOTEUR-BULLSEYE -> /srv/tftp/F_TS-412-SABOTEUR_bullseye_5.10.0-
23-marvell
  -rw-r--r-- 1 root   root     16777216 Sep 10 18:47 F_TS-412-SABOTEUR_bullseye_5.10.0-23-marvell
  drwxr-xr-x 3 root   root         4096 May 20  2023 ..
  -r--r--r-- 1 root   root     16777216 May 19  2023 F_TS-412-SABOTEUR-BULLSEYE_older_version
  -r--r--r-- 1 root   root     16777216 Mar 17  2023 F_TS-412-SABOTEUR-STRETCH
  -r--r--r-- 1 root   root     16777216 Mar 16  2023 F_TS-412-stretch
  -r--r--r-- 1 root   root     16777216 Mar  9  2023 F_TS-412-QTS
```

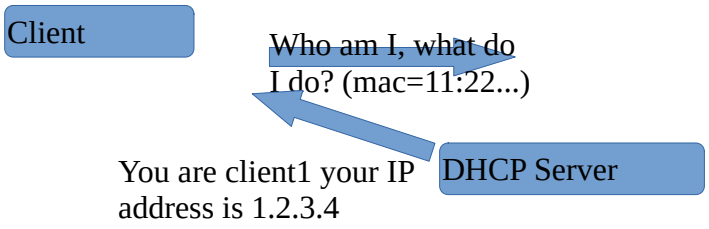
/srv/tftp

The file **F_TS-412-M11_BOOKWORM** is a symbolic link to **F_TS-412-M11_bookworm_6.1.0-13-marvell.** This convention allows simple “upgrades” from say 6.1.0-13- to 6.1.0-14 without needed to edit 02PXE. You just need to point at the version you want.

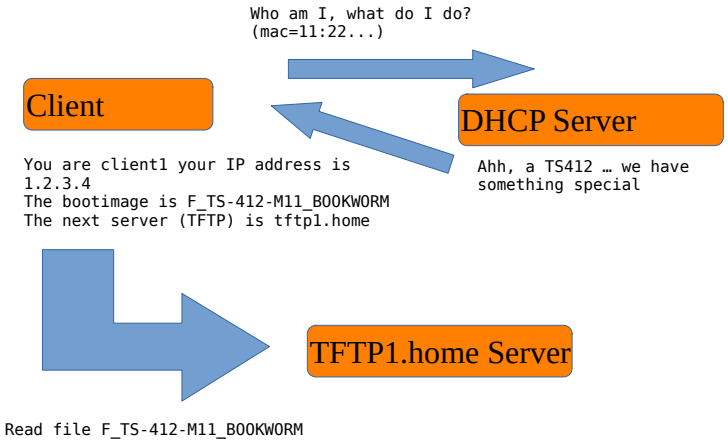
For full details of how this consult a BOOT/PiXE guide:

- https://www.eventhelix.com/networking/Bootp.pdf
- https://en.wikipedia.org/wiki/Bootstrap_Protocol
- https://en.wikipedia.org/wiki/Preboot_Execution_Environment

But paraphrased into simple language:



Or



You will note there are 2 similar names:

```
-rw------- 1 graeme everyone 16777216 Dec  8 00:04 F_TS-412-M11_bookworm_6.1.0-13-marvell
-rw-r--r-- 1 graeme everyone 14680064 Dec  7 23:48 dl-F_TS-412-M11_bookworm_6.1.0-13-marvell
```

The file **dl-*** is 14Mb and the one without the **dl-** prefix is 16MB.

The 16MB is a PiXE image suitable for the **QNAP recovery**. The 1st 2MB are ignored (but sadly still need to be present, to get the offsets correct) the 14MB is used to do a diskless (flashless :-)) boot.

The way this works is; the *builtin recovery* in the QNAP will take the offered filename and request that named file from the given TFTP server. It will then install that image into FLASH memory (skipping the first 2MB) (@ 0xF8200000) then load the kernel into RAM (@0x00800000) and jump to to.

In the *non recovery situation*, it will use the UBOOT setting we define below, the causes the file **dl-***filename* to be loaded directly into RAM at 0x00800000 and executed.

So this brings us to “what are the uboot settings”?

UBOOT Settings:

M11 Layout (11MB rootfs)

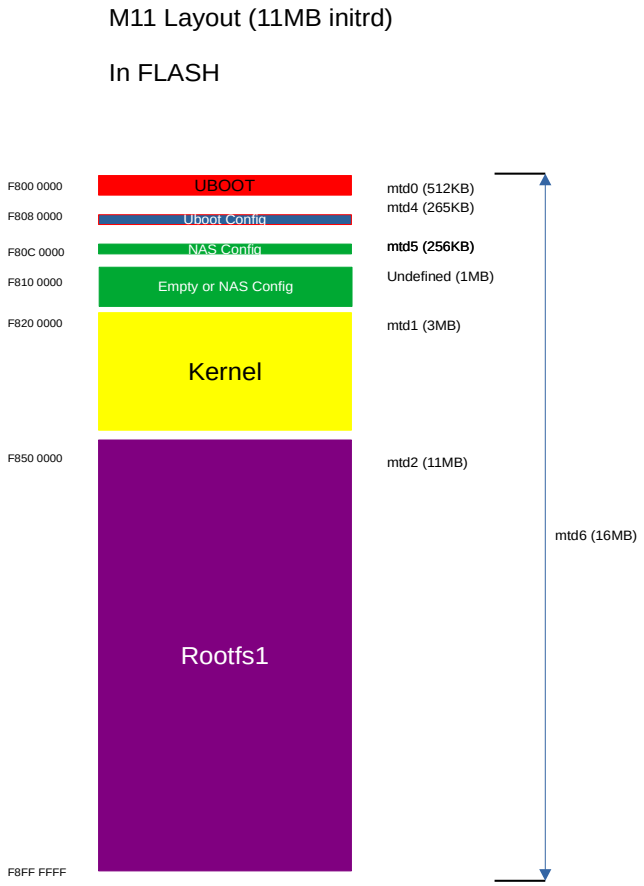
```
setenv mtdparts spi0.0:512k@0(uboot)ro,3M@0x200000(Kernel),11M@0x500000(RootFS1),2M@0x200000(Kernel_legacy),256k@0x80000(U-Boot_Config),1280k@0xc0000(NAS_Config),16M@0( )ro
setenv bootargs console=ttyS0,115200 root=/dev/ram initrd=0xb00000,0xB00000 ramdisk=34816 cmdlinepart.mtdparts=${mtdparts} mtdparts=${mtdparts}

setenv fbootcmd echo flash\;cp.l 0xf8200000 0x800000 0xc0000\;cp.l 0xf8500000 0xb00000 0x2C0000\;bootm 0x800000
setenv nbootcmd echo net\;dhcp\;dhcp\;tftpboot 0x800000 dl-\${bootfile}\;bootm 0x800000
setenv bootcmd  uart1 0x68\;${nbootcmd}\;${fbootcmd}

setenv bootdelay=5

setenv wlanaddr 00:08:9b:cc:xx:xx
setenv ethaddr    00:08:9b:cc:xx:yy
```

**NOTE. If you use cut&paste to set these values be aware the width of the terminal has a large effect (line truncation).

mtdparts defines the various partitions:

Partition	Name	Size	offset
mtd0	uboot	512KB	0x000000
mtd1	kernel	2MB	0x200000
mtd2	RootFS1	11MB	0x500000
mtd3	Kernel_legacy	256KB	0x80000
mtd4	Boot_Config	2MB	0x200000
mtd5	NAS_Config	1280KB	0xc0000
mtd6	all	16MB	0x0

These are the M11 values, a different layout needs different values. Also note these are not in address order.



That's one variable! The two variables **fbootcmd** and **nbootcmd**, "pronounced" flash boot command and network boot command.

- **fbootcmd** echos the word "flash" to the serial console, then copies the flash contents of the kernel and initrd into their correct RAM locations, then calls **bootm 0x800000.**
- **nbootcmd** echos the word "net" to the serial console, then issues **two** dhcp commands, then a **tftpboot 0x800000 dl-\${bootfile},** finally also does a **bootm 0x800000.**

There are a few complexities here:

1. The sizes are in words not bytes (this seems to be the convention used) so some arithmetic is needed.
2. TWO **DHCP**s are needed for a curious reason. When the NAS is powered on (or reset) the NIC hardware is in an unstable state (initialising) in this situation the first DHCP fails.

However it fails by timing out, this long delay means the initialisation is complete in time for the second DHCP. If the commands are run “manually” the 1st DHCP will succeed (so it will go much faster)

3. The **tftpboot** command requests the file **dl-${bootfile}** where bootfile is the name returned by the dhcp request (as defined on 02PXE above). This is the mechanism that separates the PiXE boot from the diskless boot.

Bootcmd, this just attempts **nbootcmd** followed by **fbootcmd**, if nbootcmd succeeds then the fbootcmd is not executed. The effect is, it tries a *network boot*, if it fails it does a *flash boot*. The initial initial **uart1** command on the authors system just reports an error (so is a NOP) but on newer versions of UBOOT it probably enables a hardware watchdog. This means, if you have the BOOTP server setup and plugged in you can boot diskless, then by unplugging the Ethernet cable during boot, the system will boot from flash.

bootargs these are the kernel arguments. The major part of this two keyword arguments **cmdlinepart.mtdparts** and **mtdparts**. Strictly only *mtparts* should be used however a bug means the syntax needed to be *cmdlinepart.mtdparts*. Since we don’t know if the kernel being booted has that bug we set both versions.

bootdelay sets a delay before uboot attempts an automated boot, typically when there is no serial console.

ethaddr defines the MAC address of ETH0

wlanaddr[1] defines the MAC address of ETH1

The actual MAC addresses used must be different and depend on your hardware (the XX in the above are placeholder and will not work) . The easiest way it probably to note the address used initially by QTS. Be aware there are some “fallback addresses” (e.g. 00:00:00:00:05:09, 00:11:88:0f:62:81, 00:50:43:3c:3b:5d or 00:00:00:00:00:00) used by QTS, you should NOT use these. If it’s too late for QTS, then you may be able to read the settings from NAS-Config (quite complex , its a filesystem). Worst case you can just make one up.

The method of setting *quite complex* uboot variables is to substitute “immediately” smaller strings of arguments. The length of the generated strings is quite close to uboot limits. There is a mechanism in uboot to allow *deferred substitution* but the author had no success with this, which could relate to the uboot version.

So if you want to switch to the M11 layout, probably the easiest way is to get the system booting via bootp (network boot) then with the **bootargs** active using the above mtdparts once you have successfully booted (diskless) you can simply use the **original** flash-kernel(8) command (or update-initramfs(8). Since the correct mtd devices are defined it should write the kernel an initrd into the

1 The reason name **wlanaddr** defines ETH1 is historic and possibly relates to the old dreamplug device.

correct locations. Be leery of re-flashing too often as the NOR memory has a limited number of read/write cycles

So the big question left is *how to create the BOOTP image* and PiXE recovery image. The PiXE images is relatively simple , if you are already running from a flash image.

```
modprobe mtdblock
cat /dev/mtd0 > mtd0
cat /dev/mtd1 > mtd1
cat /dev/mtd2 > mtd2


cat /dev/mtd4 > mtd4
cat /dev/mtd5 > mtd5


. /etc/os-release
echo $VERSION_CODENAME
KERNEL=$(uname -r)
NAME="F_TS-412-MOUICHE_${VERSION_CODENAME}_${KERNEL}"


#cat mtd0 mtd4 mtd5 mtd1 mtd2 mtd3 > ${NAME}  # This is layout on Original & SABOTEUR
cat mtd0 mtd4 mtd5 mtd1 mtd2       > ${NAME}  # This is layout with MOUICHE
```

grab-all-mdt-with-mouiche.sh

This example grabs the bits of flash on the assumption they are in the `mouiche` layout. The enclosed comment shows the corresponding line for the `SABOTEUR` layout. For the M11 layout it is much simpler as it is just:

- cat mtd6 > ${name}

Since this covers the entire 16MB.

Making the BOOTP image is more complex and involves using the commands:

- `action_flash_kernel`
- `dl_flash_kernel`
- `/usr/local/lib/QNAPhomebrew`
- `fake_flash_kernel.sh`
- `construct_dl_image`
- `fake_flash_kernel`

A good starting point would be **`fake_flash_kernel(8)`**. Read the man pages to see how to use these.

Once created the bootp image and the PiXE images can be copied to the TFTP server in the locations (and with the names) defined in 10 above.

qcontrol and hysteresis settings

As part of this project the command qcontrol(1) has been modified, tentatively called version 0.5.9~gpvsmh. This saves key data in sysV shared memory and an example command currently called qnap_info(1) (might might get renamed qinfo(1)) . Example output is:

```
graeme@ts412:/etc/qcontrol$ qnap-info
At 2024-01-20 17:19:19 the temp value was 32
At 2024-01-20 17:08:21 the fan state was N
At 2024-01-20 17:08:22 the fan speed was FAN_STOP
```

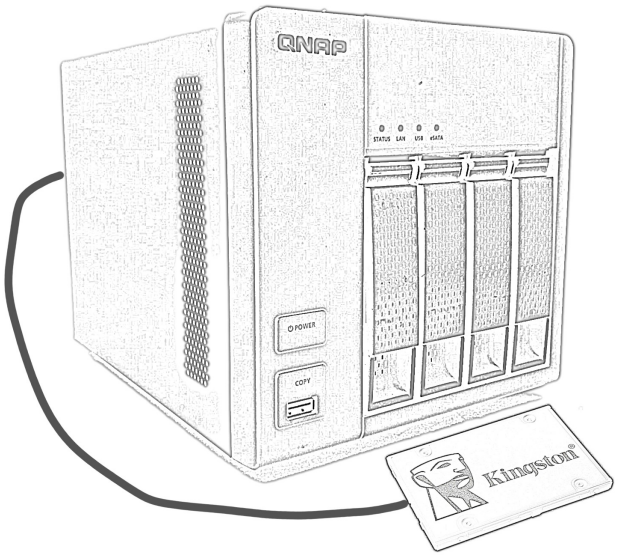
This says the enclosure is at 32 degrees C, the fan is “Normal” (not error, i.e. broken) and the fan is stopped. The effect of the NAS Scripts would also allow the disks to spin down making the enclosure cooler and hopefully totally silent. The hysteresis config file is /etc/qcontrol.d/ts412fan.conf:

```
-- GPV setup to shut fan totally off below 35 degrees
-- hysteresis implementation:
--
--   > 37------+ >40 --------+ > 45 -----+ >50 -------+ > 65 ----+
--             |             |           |            |          |
--               v             v           v            v          v
--  off --    silence --    low    --  medium   --   high    -- full
--    ^           ^           ^           ^             ^         |
--    |           |           |           |             |         v
--  < +------ <35 +------ <37 +------ <40 +-------- <50 +------- <60


function temp( temp )
          logtemp(temp)
          if last_fan_setting == "full" then
          if temp < 60 then
                    setfan(temp, "high")
          end
          elseif last_fan_setting == "high" then
          if temp > 65 then
                    setfan(temp, "full")
          elseif temp < 50 then
                    setfan(temp, "medium")
          end
          elseif last_fan_setting == "medium" then
          if temp > 50 then
                    setfan(temp, "high")
          elseif temp < 40 then
                    setfan(temp, "low")
          end
          elseif last_fan_setting == "low" then
          if temp > 45 then
                    setfan(temp, "medium")
          elseif temp < 37 then
                    setfan(temp, "silence")
          end
          elseif last_fan_setting == "silence" then
          if temp > 40 then
                    setfan(temp, "low")
          elseif temp < 35 then
                    setfan(temp, "stop")
          end
          elseif last_fan_setting == "stop" then
          if temp > 37 then
                    setfan(temp, "silence")
          end
          else
          setfan(temp, "high")
          end
end

--
-- Local variables:
-- mode: lua
-- indent-level: 8
-- End:
```

The only change to the config is to allow the temperate to rise to 37°C with the fan stopped. 37 is body temperature it seems reasonable to allow it to remain 37°C at without needing to cool it.

NAS scripts

The bulk of the work in QNAPHomebrew is in the form of these scripts. Each script has a man page for details but before that it’s necessary to get an overview:



The authors system has 4 HDD in trays 1-4 and an SSD plugged into eSATA1 (there are 2) as described above

The system can boot from either the LAN or Flash (see details above). In either case it will load an initrd which includes the script choose-root. This script looks at the labels on all the filesystems looking for ones that look like “root?” e.g root1 . An example of the script in operation can be seen here below:

```
[   12.382494] FYI incoming ROOT=/dev/disk/by-label/root5 is a valid block device.
[   12.391621] choice=/dev/disk/by-label/root1
[   12.396709] Found /dev/disk/by-label/root1 is valid block device, candidate for ROOT
[   16.411874] choice=/dev/disk/by-label/root2
[   16.416555] Found /dev/disk/by-label/root2 is valid block device, candidate for ROOT
[   20.430242] choice=/dev/disk/by-label/root3
[   20.434829] Found /dev/disk/by-label/root3 is valid block device, candidate for ROOT
[   24.448384] choice=/dev/disk/by-label/root4
[   24.452964] Found /dev/disk/by-label/root4 is valid block device, candidate for ROOT
[   28.466514] choice=/dev/disk/by-label/root5
[   28.471095] Found /dev/disk/by-label/root5 is valid block device, candidate for ROOT
[   32.484607] NEWROOT=/dev/disk/by-label/root5
[   32.489264] Selected /dev/disk/by-label/root5 as root
[   32.494420] write ROOT="/dev/disk/by-label/root5" to /conf/param.conf
[   32.501136] here is /conf/param.conf
[   32.509951] ROOT="/dev/disk/by-label/root5"
[   32.839302] EXT4-fs (sde2): mounted filesystem with ordered data mode. Quota mode: none.
```

In this case all five disks are available

```
[    8.759726] xor: using function: arm4regs (1115 MB/sec)
[    8.963589] FYI incoming ROOT=/dev/disk/by-label/root5 is NOT a valid block device, unless we find an
alternate, boot will fail
[    8.976565] choice=/dev/disk/by-label/root1
[    8.981777] Found /dev/disk/by-label/root1 is valid block device, candidate for ROOT
[   12.997222] NEWROOT=/dev/disk/by-label/root1
[   13.001953] Selected /dev/disk/by-label/root1 as root
[   13.007111] write ROOT="/dev/disk/by-label/root1" to /conf/param.conf
[   13.013824] here is /conf/param.conf
[   13.022937] ROOT="/dev/disk/by-label/root1"
[   13.435596] EXT4-fs (sda2): mounted filesystem with ordered data mode. Quota mode: none.
[   14.652541] systemd[1]: Inserted module 'autofs4'
```

In this case the SSD (eSATA) unplugged and only Tray1 populated.

It finds root1, root2,root3,root4 and finally root5. The filesystem labelled root5 is on the SSD, this will be mounted as / (root filesystem) and the system startup will be defined by it's contents.

If the SSD were unplugged, choose-root would have chosen root4 (the HDD in tray No4) and used that to complete the boot. Thus is can be seen that the system can come up with any **one** disk available., a higher level of redundancy that to original QNAP. However depending on how the data are distributed you will have limited user data. As an aside in the sample setup there is also home1, home2 ...home5 and they behave just like root1..5 so small key data could be stored here (e.g. security keys) there would be 5 copies and any one could be used.

This redundancy is the key to how QNAPHomebrew works. Many of the scripts are designed to setup and maintain this structure. In addition to the system filesystems, the remainder of the the disks are used for user data, this can be configured any many ways with differing degrees of redundancy. One key difference to the QTS approach is not all data is considered equal, for example pgpkeys needed to work with outside systems need a really high level of redundancy, email and scans of important letters needs at least RAID5/6 protection but movies stored on the NAS for use

by DNLA server are also on multiple backups and a single copy here suffices (we can live without these until a replacement disk arrives in the post).

Now it is possible to have, say root1 in tray3 and root5 in tray1 but for the sake of this example we will assume there are 4 HDD in trays 1-4 and an single SSD on eSATA1. What you can't have are say two or more root1 filesystems. If you remove HDD it is NOT necessary that they go back in the same slot but you will probably confuse yourself if you move things around.

So the first step is to commission the drives. This is done with the command **QNAP_commission_disk(1)**. This command requires the disk is "empty" meaning it has an empty gpt partition table on it; this is to avoid you mistakenly overwriting an "in use" disk. You would normally do this in another system but with some effort it can be done if you are already running Debian on the QNAP. So here we assume you commission 4 disks (trays 1-4) and are already running Debian from the the first HDD (Tray1, /dev/sda).

Be aware in what follows that the author developed these scripts with a single TS412 available, so has never been though this "clean install", it has proceeded in step with the script development.

So we start by commissioning the SSD (disk5).[2]

In a linux computer (it need not be the QNAP) put a GPT partition table on the drive (gparted(1),parted(1),fdisk(1),sfdisk(1) ...many choices)

Plug the disk into the QNAP ...note which device it is recognised. In a fully populated TS412 is will probably be /dev/sde, but check syslog (you don't want to wipe the wrong disk..fortunately it will only commission an empty drive.

QNAP_comission -b5 -v /dev/sde

This will (among other things) put root5 on the SSD but it lacks any files, so will cause a boot to fail if choose-root is in use (just unplug the drive at boot time to fix this)

Now we need to get a usable system onto /dev/sde. Assuming you have booted a reasonable Debian system (from /dev/sda) and have the common filesystems and directories. You may be able to use the QNAP_clone_disk(1) utility. To avoid it objecting, add the label "root1" to the root filesystem on /devsda (e2label root1 /dev/sda).

2 You may want to look @ QNAPinstalldepends before starting work

Then you can just run

```
QNAP_clone_disk -v -b5
```

Note there are no device names given. It will clone the current (running) system onto the disk that holds the label “root5” , which we created above.

Now /dev/sde contains a valid “system” but will **not** currently be used at boot time. To do this you need to install the **choose-root** script. This needs to be copied into **/etc/initramfs-tools/scripts/local-top/choose-root.** Then recreate the initrd. How to do do this depends on what boot method you have chosen (see:above). With the original flash boot model it would be to use the command **update-initramfs(8)**.

Before builiding a new initrd, you may want to review QNAPinstalldepends.sh. This removes the package plymouth which makes the initrd considerably smaller (see: https://unix.stackexchange.com/questions/759516/eliminate-inclusion-of-x11-libs-in-initrd) (also https://forum.qnap.com/viewtopic.php?t=172322).

Now if you reboot with, at least, /dev/sda and /dev/sde in place it should boot from /dev/sde (the SSD) . If you remove the SSD and reboot (not necessarily in that order) it should come up from /dev/sda.

Now, at your leisure you can commission disks 2-4 and clone the running system onto these.

Once base system is running to your satisfaction, you can look to setup the disk for data as you might wish**:

- ((1) reassigns data1 – data4 from 4 normal EXT4 filesystems into a single RAID array
- QNAP_recreate_raid(1) needs to be used if you have problems with creating the raid array and need to restart. It is harder to use than QNAP_create_raid(1) as you need to know device names.
- QNAPmount(1) creates directories similar to QTS and exports them via NFSv3. There is a systemd service which name be used to cause the export to happen automagically at boot time. Note the exports are NOT done via /etc/fstab as a missing disk would cause boot to fail. Here there will simply be some missing filesystems.
- QNAPgensamba(1) creates a Samba config sharing out filesystems quite similar to the original QTS names.

**These scripts will probably require local editing before use, especially QNAPmount. Decide if you want RAID or not, work out which are you biggest disk etc, assign directories to disks as appropriate.

Typical disk layout.

Running QNAP_commission_disk on a disk creates eight filesystems.

1. /boot
2. / (root)
3. /var
4. SWAP
5. /tmp
6. /home
7. /data (label = data1, data2 etc)
8. /rest (label = rest1, rest2 etc)

The initial setting in the script makes /data 512GiB and /rest uses “the rest of the space” , so could be big or small depending of the size of your disks.

If you then opt to create a RAID array, then all the /data filesystems are lost and you just get a single raid array (e.g. with 4 *512GiB devices) so it might be “big”.

So in assigning your data directories to filesystems you may have data1...data4 and rest1..rest4 or you might have raid0 + rest1..rest4 , the initial **QNAPmount** assumes the latter. The initial allocation is:

- rest1 is unused
- rest1 holds just one directory “Public”
- rest3 holds “Multimedia” (on the authors system it is quite large, and not RAID)
- rest4 holds multiple directories svn git Web USBUploads Recordings InternalAdmin Download
- raid0 holds “homes” (NB not the same as /home)

The it also creates /share/Public, /share/Multimedia etc as symbolic links, so that users need not be concerned with where exports are physically located. These “shares” and also used in the SAMBA share configuration.

Ongoing admin:

From time to time you will install updates ro Linux and various packages running on the QNAP. You may need to do some of the following:

- Rebuild initrd , this may happen automaticity when some packages are updated.
- Rebuild the BOOTP image and copy to the TFTP server (the build may be automated, iff the various flash scripts are installed)
- Rebuild and install the actual FLASH image, not too frequently because of of memory fatigue but needed to keep the flash booting working.
- Copy of PiXE recovery image back to TFTP server
- re-cloning of disks. The utility QNAP_clone_alldisk(1) exists and simply needs to be run from time to time as root. The big question is “when”
    - Right after an “upgrade” ...carries the risk that a bad upgrade get duplicated and breaks all disks.
    - Never, means the “alternate” boots are possibly too old to work with the current kerel & initrd
    - Once a month, possible ...but risk being too often or too infrequent depending on you use.

Sample output

Normally Tray1 hold /dev/sda with label “root1”, tray2 /dev/sdb etc with label “root2”. In some of these examples disks have been “popped” and replaced, making the mapping “non-standard” it will revert to the normal mapping on the next reboot. However it is interesting to see the warnings generated in this case. Ideally we would like to be able to flash the LED over the correct drive, but the necessary codes to be sent to the PIC are unknown.

QNAP_clone_disk

```
root@ts412:/home/graeme/src/QNAPhomebrew# time bash ./QNAP_clone_disk -b 1
WARNING Clone is going to ../../sdb2 , this is as unexpected (you may have chosen irregular mappings)
Last chance to bail out --- press enter to continue


real    3m18.182s
user    0m28.884s
sys     0m51.578s
```

QNAP_clone_alldisk

```
root@ts412:/home/graeme/src/QNAPhomebrew# time bash ./QNAP_clone_alldisk

We are booted from 5
The list of drives we will clone to is:  1 2 3 4

Doing drive 1
WARNING Clone is going to ../../sdb2 , this is as unexpected (you may have chosen irregular mappings)
Quiet mode enabled ... going ahead with drive 1
That clone (drive 1) took 68 seconds

Doing drive 2
WARNING Clone is going to ../../sdc2 , this is as unexpected (you may have chosen irregular mappings)
Quiet mode enabled ... going ahead with drive 2
That clone (drive 2) took 108 seconds

Doing drive 3
WARNING Clone is going to ../../sde2 , this is as unexpected (you may have chosen irregular mappings)
Quiet mode enabled ... going ahead with drive 3
That clone (drive 3) took 97 seconds

Doing drive 4
Clone is going to ../../sdd2 , this is as expected
Quiet mode enabled ... going ahead with drive 4
That clone (drive 4) took 111 seconds

Total clone took 384 seconds (started at Tue 28 Nov 13:19:58 GMT 2023, ended at Tue 28 Nov 13:26:22 GMT 2023)

real    6m23.640s
user    1m42.457s
sys     3m17.672s
```

QNAPmount

```
root@ts412:/home/graeme/src/QNAPhomebrew# QNAPmount


root@ts412:/home/graeme/src/QNAPhomebrew# mount | grep QNAP
/dev/sda10 on /QNAP/mounts/rest1 type ext4 (rw,relatime)
/dev/sdb10 on /QNAP/mounts/rest2 type ext4 (rw,relatime)
/dev/sdc10 on /QNAP/mounts/rest3 type ext4 (rw,relatime)
/dev/sdd10 on /QNAP/mounts/rest4 type ext4 (rw,relatime)
/dev/md0 on /QNAP/mounts/md0 type ext4 (rw,relatime,stripe=384)
root@ts412:/home/graeme/src/QNAPhomebrew# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4] [linear] [multipath] [raid0] [raid1] [raid10]
md0 : active raid5 sdd9[4] sdc9[2] sdb9[1] sda9[0]
      1610216448 blocks super 1.2 level 5, 512k chunk, algorithm 2 [4/3] [UUU_]
      [====>.............]  recovery = 23.9% (128497404/536738816) finish=434.6min speed=15653K/sec
      bitmap: 0/4 pages [0KB], 65536KB chunk

unused devices: <none>
```