



How to Replace a System on Your Hardware

A Journey of Synology RS815RP+ to Custom Linux

Jan Kaisrlik
Linuxdays 2026

About me

Jan Kaisrlik

- Engineer @Intel (Research & POC) & Contractor
- System and Embedded Linux specialist/enthusiast
- Currently running custom Linux on Synology RS815RP+



Why Replace the System?

- Acquired Synology RS815RP+ rackmount NAS
 - No longer maintained system
 - Running old kernel

```
[ 0.000000] Linux version 3.10.102 (root@build3) (gcc version 4.9.3 20150311  
(prerelease) (crosstool-NG 1.20.0) ) #15091 SMP Wed May 17 11:18:57 CST 2017
```

- Original system limitations
- Desire for newer, more flexible system

Why Replace the System?

- Acquired Synology RS815RP+ rackmount NAS
 - No longer maintained system
 - Running old kernel

```
[ 0.000000] Linux version 3.10.102 (root@build3) (gcc version 4.9.3 20150311  
(prerelease) (crosstool-NG 1.20.0) ) #15091 SMP Wed May 17 11:18:57 CST 2017
```

- Original system limitations
- Desire for newer, more flexible system

It is simply a challenge!

Understanding the Challenge

What we may need to know before replacing the system:

1. Platform analysis

- What is platform specification?
- What peripherals are available?
- Do we need any special equipment?
- Are there any existing resources/docs available?

2. Boot process

- Can we log in?
- Can we parse rootfs?
- Can we load own binary?
- Can we intercept any stage?
- How does the device boot?

1. Driver requirements

- What kernel modules are needed?
- Are drivers available?

Keep it mind you may not get answer for all your questions.

Warning

Some steps may have negative impact and could potentially brick the device. Proceed with caution!

Platform Analysis 1/2

- must have: USB-to-TTL/USB-to-RS232, multimeter, soldering station
- good to have: oscilloscope, flash programmers, JTAG

Step 1: Hardware Investigation

- Network interfaces: Check if any ports are open (telnet, ssh, ...)

```
nmap -p- <device-ip> # or try default  
ports: telnet (23), ssh (22)
```

- Is UART/JTAG available on the board?
- What storage/controllers are available?

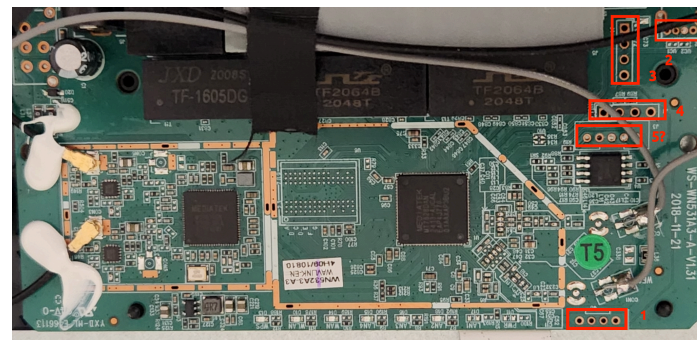


Figure 2: In some cases, we need to debug a bit.

Platform Analysis 1/2

- must have: USB-to-TTL/USB-to-RS232, multimeter, soldering station
- good to have: oscilloscope, flash programmers, JTAG

Step 1: Hardware Investigation

- Network interfaces: Check if any ports are open (telnet, ssh, ...)

```
nmap -p- <device-ip> # or try default  
ports: telnet (23), ssh (22)
```

- Is UART/JTAG available on the board?
- What storage/controllers are available?

Why Serial Console Matters

- Primary debugging interface
- Usually outputs all boot messages

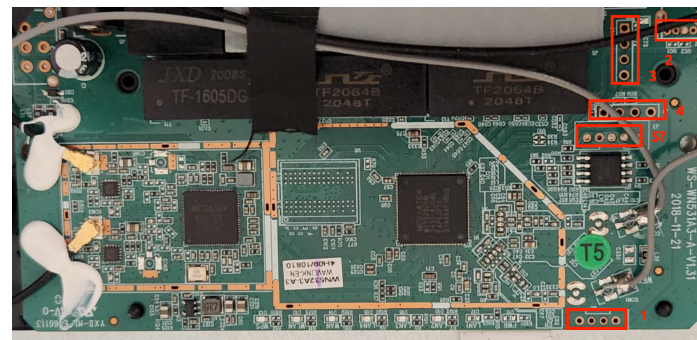


Figure 3: In some cases, we need to debug a bit.

Platform Analysis 1/2

- must have: USB-to-TTL/USB-to-RS232, multimeter, soldering station
- good to have: oscilloscope, flash programmers, JTAG

Step 1: Hardware Investigation

- Network interfaces: Check if any ports are open (telnet, ssh, ...)

```
nmap -p- <device-ip> # or try default  
ports: telnet (23), ssh (22)
```

- Is UART/JTAG available on the board?
- What storage/controllers are available?

Why Serial Console Matters

- Primary debugging interface
- Usually outputs all boot messages

! Warning

Double check if logic is USB-to-TTL or USB-to-RS232 before connecting adapter

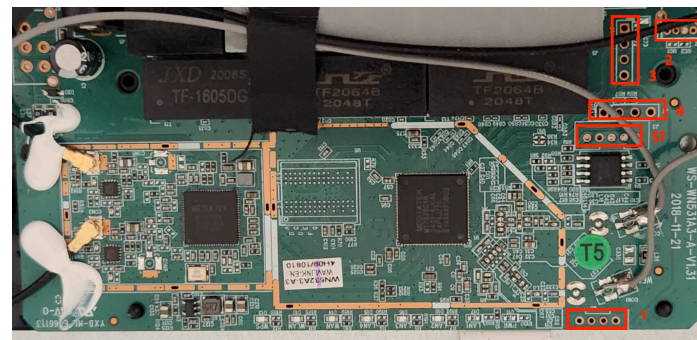


Figure 4: In some cases, we need to debug a bit.

Boot Process

Bootstages: EFI/Bootstrap -> Bootloader(1st stage, 2nd stage) -> Kernel(ramdisk) -> Full system

Step 1: Can we intercept anything

- Can EFI load our stub?
- Can we intercept bootloader (GRUB/Uboot) and modify boot inputs (special boot seq.?)
 - Can we modify kernel bootargs and boot directly to shell?

```
# GRUB legacy - edit entry with 'e' key
kernel /bzImage init=/bin/sh ...
# Uboot modify bootargs
setenv bootargs "${bootargs} init=/bin/sh"
```

- Can we log in to system and modify behavior of any steps above?

Platform Analysis 2/2

Step 2: Can you investigate system?

- Can you log in to current system or find rootfs/initramfs?
 - Commands below do not work for offline investigation e.g. dumping flash memory
 - Check for drivers

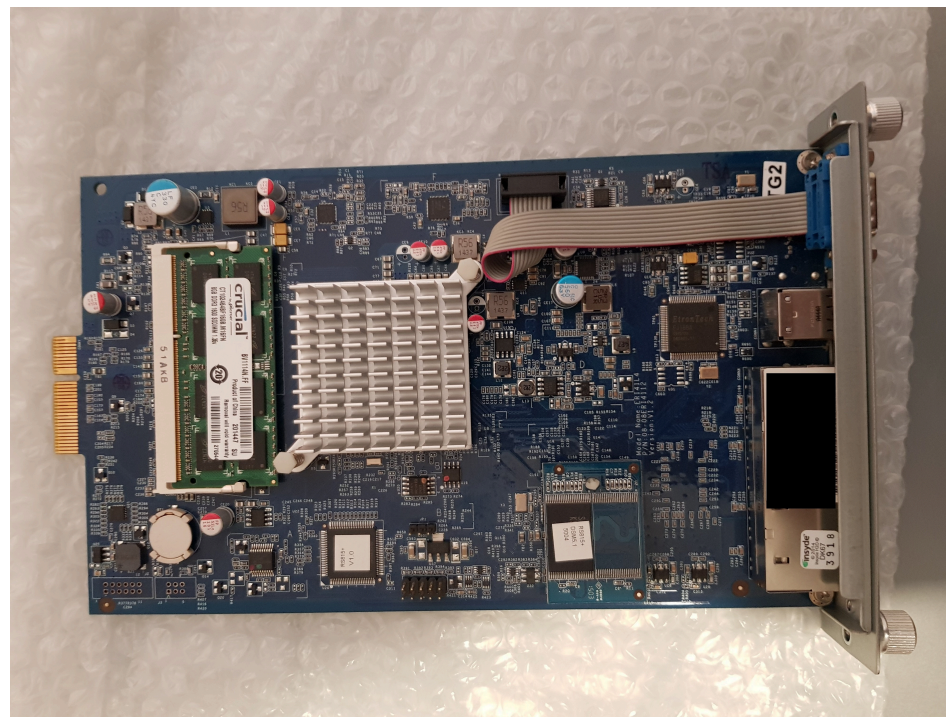
```
# Inspect kernel logs
dmesg
# Find appropriate drivers for eth0
cat /sys/class/net/eth0/device/uevent | grep DRIVER
DRIVER=igb
```

- filesystem format mount, lsblk, fdisk, etc.
- Check where kernel image, bootloader, ramdisk is stored
- Does it use ACPI or Device tree?

```
# Extract ACPI tables
cat /sys/firmware/acpi/tables/DSDT > dsdt.dat
iasl -d dsdt.dat
# Extract DTS
dtc -I dtb -O dts /sys/firmware/fdt -o devicetree.dts
```

Platform analysis of RS815RP+

- CPU Model: Intel Atom C2538
- Chips visible on board: Etron EJ168A (PCIe Gen2 to USB 3.0 controller)
- Altera FPGA: hardware system manager
- Ethernet Controllers: Intel I350 (4x Gigabit ports)
- SPI NOR flash for BIOS (Winbond W25Q64FV - 8MB)
- USB flash - 128MB (Phison PS2251-68-5)
- UART: DB9 (9-pin D-sub connector) - RS-232 logic level



Boot Process on RS815RP+

- Confirmation of UART settings found online: 115200 baud, 8N1, RS-232 logic level
- Can GRUB be stopped to allow me to overwrite boot arguments?

```
Booting SYNOLOGY_2 in 1 seconds...
```

```
GNU GRUB version 0.97 (540K lower / 2074236K upper memory)
+-----+
| SYNOLOGY_1                               |
| SYNOLOGY_2                               |
+-----+
Use the ^ and v keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, 'a' to modify the kernel arguments
before booting, or 'c' for a command-line
```

- Password generator available on the [internet](#)

(so I do not need to modify bootargs to jump to shell directly)

In the Shell of RS815RP+

Start snooping around:

```
RackStation> dmesg
...

RackStation> mount
/tmp on /tmp type tmpfs (0)
none on /dev type devtmpfs (0)
none on /dev/pts type devpts (gid=4,mode=620)
/sys on /sys type sysfs (0)
none on /sys/kernel/debug type debugfs (0)

RackStation> cat /proc/partitions
major      minor    #blocks  name
 135         240     122880  synoboot
 135         241      16033  synoboot1
 135         242      96390  synoboot2
```

synoboot sounds important, let's check what's inside

```
RackStation> ls -la /dev/synoboot1
brw----- 1 root    root      135, 241 Jan  1 00:01 /dev/synoboot1
RackStation> mount -t vfat /dev/synoboot1 /mnt
mount: mounting /dev/synoboot1 on /mnt failed: Invalid argument
# Tried all available filesystems, but can mount extrnal USB flash drive
```

Does Synology prevent loading synoboot files?

Let's build our own mount. - There are a lot of useful tools missing anyway.

Own Tooling on RS815RP+

The easiest way to compile your own toolset is to use Buildroot.

```
git clone https://github.com/buildroot/buildroot.git
make pc_x86_64_defconfig
```

- Create debugging tooling (using the same kernel version) and statically compile all binaries.

Configure following:

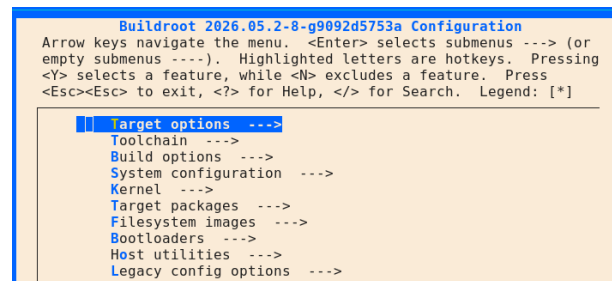
1. Use same kernel version as target system (avoids API incompatibility)
2. Configure Buildroot to use musl (or uClibc)
3. Build static busybox with Buildroot (available only for libc mentioned in previous step)
4. Create minimal initramfs (disable what you don't need for faster compilation, especially when compiling against an old kernel):

- /bin/sh (busybox), mount, hexdump etc.

5. Package into rootfs.cpio

Copy to flash drive and transfer it to the system.

- Help us to configrm /dev/synobootX are FAT32



Driver Discovery

```
RackStation> echo a > /dev/ttyS0
a
RackStation> cat /sys/class/tty/ttyS0/uevent | grep DRIVER
PHYSDEVDRIVER=serial8250
# Find appropriate drivers for eth0
RackStation> cat /sys/class/net/eth0/device/uevent | grep DRIVER
DRIVER=igb
```

Essential drivers for RS815RP+:

- Network: igb (Intel Gigabit Ethernet)
- Storage: AHCI/SATA controllers
- Serial: 8250 UART
- Filesystem: ext4, btrfs, etc.

Boot Partition on RS815RP+

Mounting Boot Partitions

Creating device nodes and mounting:

```
RackStation> my/mount -t vfat /dev/synoboot1 x
mount: mounting /dev/synoboot1 on x failed: Invalid argument
RackStation> cat /proc/mounts | grep /dev
none /dev devtmpfs rw,relatime,size=1013056k,nr_inodes=253264,mode=755 0 0
# Issue: Could there be some hardening in place?
# Create new device node in /tmp
RackStation> mknod -m 660 /tmp/synoboot2 b 135 242
# Mount partition with new node
RackStation> mount -t vfat /tmp/synoboot2 x
```

Boot Partition modification on RS815RP+

Partition layout:

- /dev/synoboot1 (16MB) - First boot partition (FAT)
 - GRUB EFI bootloader
 - GRUB configuration
 - Kernel images (zImage)
 - Initial ramdisk (rd.gz)
- /dev/synoboot2 (96MB) - Second boot partition (FAT)
 - Kernel images (zImage)
 - Initial ramdisk (rd.gz)

```
RackStation> ls
EFI          GRUB_VER  rd.gz      zImage
RackStation> ls EFI/boot/
SynoBootLoader.conf  SynoBootLoader.efi
```

Can We Provide Our Own Kernel/Ramdisk

With the knowledge above, we can try to build our own system. Buildroot is great tool for that.

Building minimal x86 system:

```
# Configure buildroot for x86_64
make pc_x86_64_defconfig
make menuconfig
make linux-menuconfig # You can take existing config, f.e. `zcat /proc/config.gz`
```

Key kernel config changes:

- Disable CONFIG_DRM (DRM stack not needed)
- Disable CONFIG_MODULES (built-in drivers only)
- Enable CONFIG_SERIAL_8250 for console
- Enable network drivers: CONFIG_IGB
- This is still not the final image - keep it minimal for faster boot/build!
- Enable tools that you may need.
- Enable ramdisk

Modification of the Boot Configuration in /dev/synoboot1

- Load kernel to /dev/synoboot2, and add entry to GRUB.
- Append config below to synoboot2/EFI/boot/SynoBootLoader.conf

```
title MYBOOT
  root (hd0,1)
  kernel /bzImage rdinit=/bin/sh console=ttyS0,115200
  initrd /rootfs.cpio
```

Typical mistakes:

- Incorrect console number - no output
- Invalid device ID
- Missing kernel drivers
- Wrong console configuration
- Partition layout confusion
- Boot order problems
- Hardware initialization timing

✓ **Success**

If you see a login prompt or shell, you've won!

Our result:

```
Welcome to Buildroot
buildroot login: root
# _
```

Building final system

Now that we have a working kernel and boot process:

My options for final system:

1. **Buildroot/Poky** - Minimal, embedded-friendly
 - Fast boot, small footprint
2. **NixOS** - Declarative, reproducible
 - Configure entire system in code
 - Easy rollbacks
 - My choice for RS815RP+

Final Boot Entry

Full NixOS configuration:

- <https://github.com/Kaisrlik/nixos-cfg/blob/master/machine/synas/configuration.nix>.

```
{ pkgs, ... }:

let
  myCustomKernel = pkgs.linuxManualConfig {
    version = "6.12.47";
    src = pkgs.fetchurl {
      url = "mirror://kernel/linux/v6.x/linux-6.12.47.tar.xz";
      sha256 = "sha256-6C/kCHF0MEgiaYe9NJ7xBxaLFaq5AUDocsp01HCSLiU=";
    };
    configfile = ./config;
  };
in
pkgs.linuxPackagesFor myCustomKernel
```

```
system.build.ext4Image = pkgs.callPackage
  "${toString modulesPath}/../lib/make-ext4-fs.nix" {
    storePaths = [ config.system.build.toplevel ];
    populateImageCommands = ''
      ln -s ${config.system.build.toplevel}/init ./files/init
      cp ${config.boot.kernelPackages.kernel}/bzImage ./files/boot/bzImage
    '';
  };
};
```

■ Copy to flash drive:

```
# Write to partition (not whole disk)
dd if=result-synas-img of=/dev/sdX1 bs=4M status=progress
```

Copy kernel from flash drive to /dev/synoboot2 and add new entry to bootloader:

```
title NixOS
  root (hd0,1)
  kernel /bzImage root=/dev/sda1 console=ttyS0,115200
```

Final Boot Entry

```
<<< Welcome to NixOS 26.11.20260805.b0db7f4 (x86_64) - ttyS0 >>>
synas login:
[root@synas:~]# cat /etc/os-release
ANSI_COLOR="0;38;2;126;186;228"
BUG_REPORT_URL="https://github.com/NixOS/nixpkgs/issues"
BUILD_ID="26.11.20260805.b0db7f4"
CPE_NAME="cpe:/o:nixos:nixos:26.11"
DEFAULT_HOSTNAME=nixos
HOME_URL="https://nixos.org/"
ID=nixos
LOGO="nix-snowflake"
NAME=NixOS
PRETTY_NAME="NixOS 26.11 (Zokor)"
VENDOR_NAME=NixOS
VERSION="26.11 (Zokor)"
VERSION_CODENAME=zokor
VERSION_ID="26.11"
```

Resources and Tools

- `stern` - Serial terminal
- `dmesg`, `lspci`, `lsblk`, ... - Device analysis
- `dd`, `grep`, `mount`, `fdisk`, ... - Rootfs analysis
- `hexdump` - Firmware/partition type analysis
- `buildroot` - `busybox`, kernel build
- `nix` - Final rootfs

Summary

Process overview:

1. Analyze hardware platform
2. Enable serial console
3. Mount and explore boot partitions
4. Identify required drivers
5. Build custom kernel + system
6. Create boot entry

Wasn't that hard:

- No ROT
- No encrypted partition
- Interactive bootloader
- Open serial
- Able to mount USB-drive

Next steps

- Update configuration with NAS-specific settings
- Enable NixOS ramdisk
- Can I replace grub?
- Can I replace bios?
- Can I replace bitstream?

Steps may differ board to board and you may brick the platform.

Questions?

Thank you!