Alan Pearce

Running NixOS on a NanoPi R5S

2023-07-30

I managed to get NixOS running on my NanoPi R5S (FriendlyElec Wiki).

Firstly, I flashed a pre-built stock Debian image from inindev to an SD card. This can be used as a rescue system later on.

From that SD card, I then flashed the same system onto the internal eMMC Storage. I only really needed to this to ensure UBoot was correctly installed; I think there will be an easier way to do it.

I had nix already installed on the NVMe SSD along with a home directory. I bind-mounted `/nix` and `/home` following the fstab I had previously set up:

```
UUID=replaceme  /mnt    ext4    relatime,lazytime   0 2
/mnt/nix        /nix    none    defaults,bind       0 0
/mnt/srv        /srv    none    defaults,bind       0 0
/mnt/home       /home   none    defaults,bind       0 0
```

I then created a user for myself using that home directory, I had full access to nix in the new Debian environment. This meant I had access to `nixos-install`.

I wanted to use the extlinux support in UBoot, so I made `/mnt/boot` point to `/boot` on the eMMC:

```
mkdir /mnt/{emmc,boot}
mount LABEL=rootfs /mnt/emmc
mount --bind /mnt/emmc /mnt/boot
```

One could *probably* delete everything else on the eMMC and move the contents of `/mnt/emmc/boot` to `/mnt/emmc`, thus obviating the need to bind-mount `/boot`

I ran `nixos-generate-config` as usual, which set up the mount points in `hardware-configuration.nix` correctly. `configuration.nix` needed a bit of tweaking. My first booting configuration was something like this, mostly borrowed from Artem Boldariev's comment:

```
{ config
, pkgs
, lib
, ...
}:
let
  fsTypes = [ "f2fs" "ext" "exfat" "vfat" ];
in
{
  imports = [ ./hardware-configuration.nix ];
  boot = {
    kernelPackages = pkgs.linuxKernel.packages.linux_6_4;

    # partial Rockchip related changes from Debian 12 kernel version 6.1
    # Also, see here:
    # https://discourse.nixos.org/t/how-to-provide-missing-headers-to-a-
kernel-build/11422/3
    kernelPatches = [
      {
        name = "rockchip-config.patch";
        patch = null;
        extraConfig = ''
          PHY_ROCKCHIP_PCIE Y
          PCIE_ROCKCHIP_EP y
          PCIE_ROCKCHIP_DW_HOST y
          ROCKCHIP_VOP2 y
        '';
      }
      {
        name = "status-leds.patch";
        patch = null;
        # old:
        # LEDS_TRIGGER_NETDEV y
        extraConfig = ''
          LED_TRIGGER_PHY y
          USB_LED_TRIG y
          LEDS_BRIGHTNESS_HW_CHANGED y
```

```
            LEDS_TRIGGER_MTD y
          '';
        }
      ];

      supportedFilesystems = fsTypes;
      initrd.supportedFilesystems = fsTypes;

      initrd.availableKernelModules = [
        ## Rockchip
        ## Storage
        "sdhci_of_dwcmshc"
        "dw_mmc_rockchip"

        "analogix_dp"
        "io-domain"
        "rockchip_saradc"
        "rockchip_thermal"
        "rockchipdrm"
        "rockchip-rga"
        "pcie_rockchip_host"
        "phy-rockchip-pcie"
        "phy_rockchip_snps_pcie3"
        "phy_rockchip_naneng_combphy"
        "phy_rockchip_inno_usb2"
        "dwmac_rk"
        "dw_wdt"
        "dw_hdmi"
        "dw_hdmi_cec"
        "dw_hdmi_i2s_audio"
        "dw_mipi_dsi"
      ];
      loader = {
        timeout = 3;
        grub.enable = false;
        generic-extlinux-compatible = {
          enable = true;
          useGenerationDeviceTree = true;
        };
      };
    };
    # this file is from debian and should be in /boot/
```

```
    hardware.deviceTree.name = "../../rk3568-nanopi-r5s.dtb";
    # Most Rockchip CPUs (especially with hybrid cores) work best with
  "schedutil"
    powerManagement.cpuFreqGovernor = "schedutil";

    boot.kernelParams = [
      "console=tty1"
      "console=ttyS2,1500000"
      "earlycon=uart8250,mmio32,0xfe660000"
    ];
    # Let's blacklist the Rockchips RTC module so that the
    # battery-powered HYM8563 (rtc_hym8563 kernel module) will be used
    # by default
    boot.blacklistedKernelModules = [ "rtc_rk808" ];

    # ... typical config omitted for brevity
  }
```

Due to the custom kernel configuration, building takes a while. I set up a distributed build to speed things up, using a Hetzner Cloud CAX21 ARM64 instance (although I could have used an x86_64 system with one of the methods mentioned on the NixOS on ARM NixOS wiki page). This made for a very long `nixos-install` command line:

```
sudo env PATH=$PATH =nixos-install --root /mnt --no-channel-copy --channel
https://nixos.org/channels/nixos-23.05 --option builders'ssh://my-host
aarch64-linux /root/.ssh/id_pappel_nixpkgs 4 2 big-parallel' --option
builders-use-substitutes true --max-jobs 0
```

I added `setenv bootmeths "extlinux"` to `/boot/boot.txt` and ran `/boot/mkscr.sh` as root to ensure that UBoot would search for the `extlinux.conf` file

Tags: #nixos #home-networking #infrastructure