Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

1 NanoPi

Youtube: https://youtu.be/lbny3zOalls

Hes·so// FRIBOURG FREIBURG
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

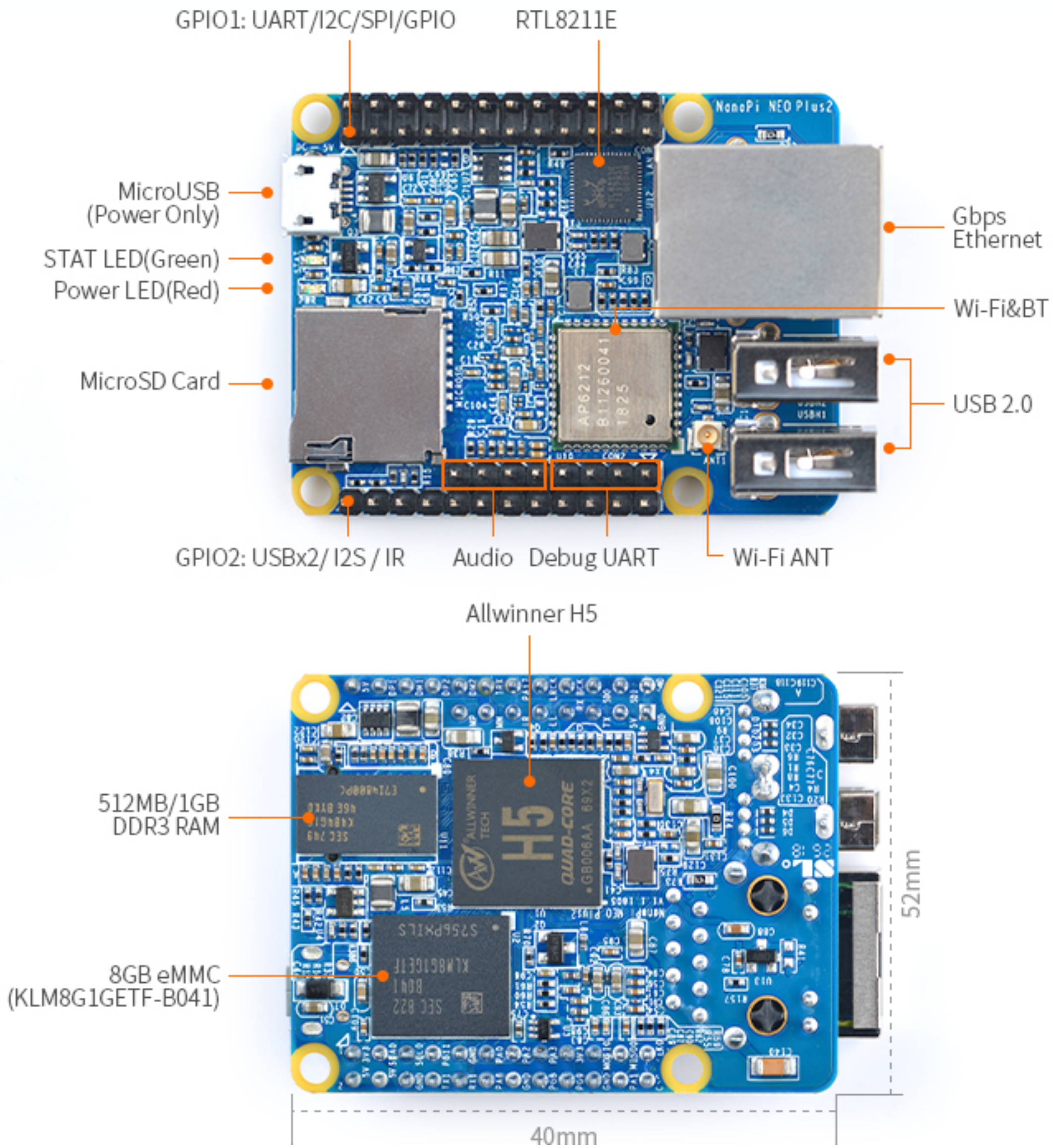
References

[1]: wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO_Plus2

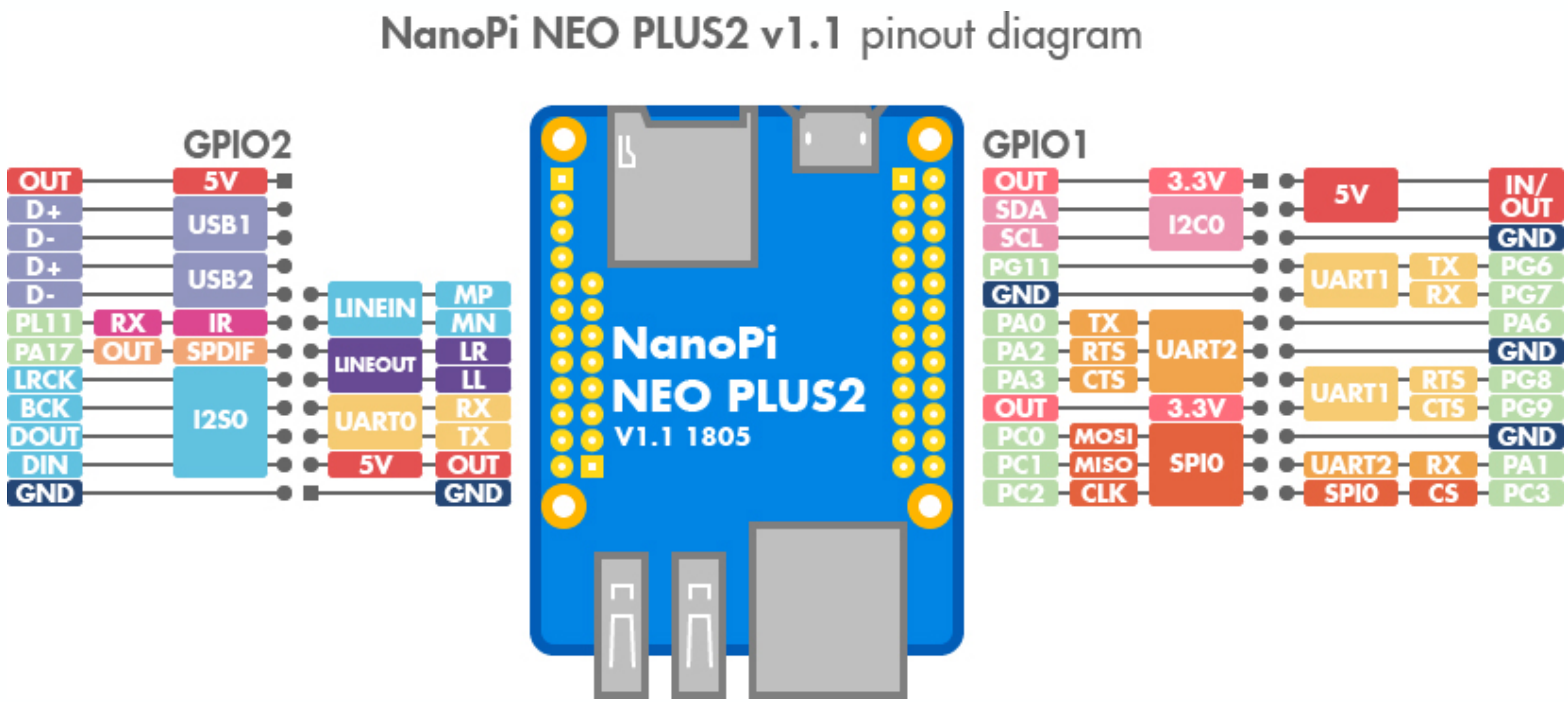
[2]: buildroot.uclibc.org

[3]: https://gitlab.forge.hefr.ch/embsys/linuxenv.git

Nano Pi NEO Plus2 [1]

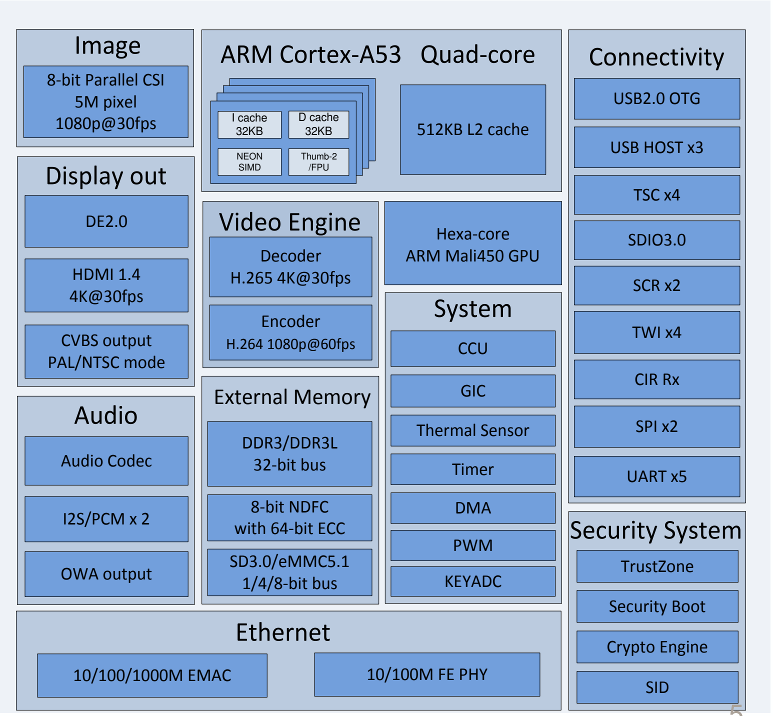


Nano Pi NEO Plus2 [1]

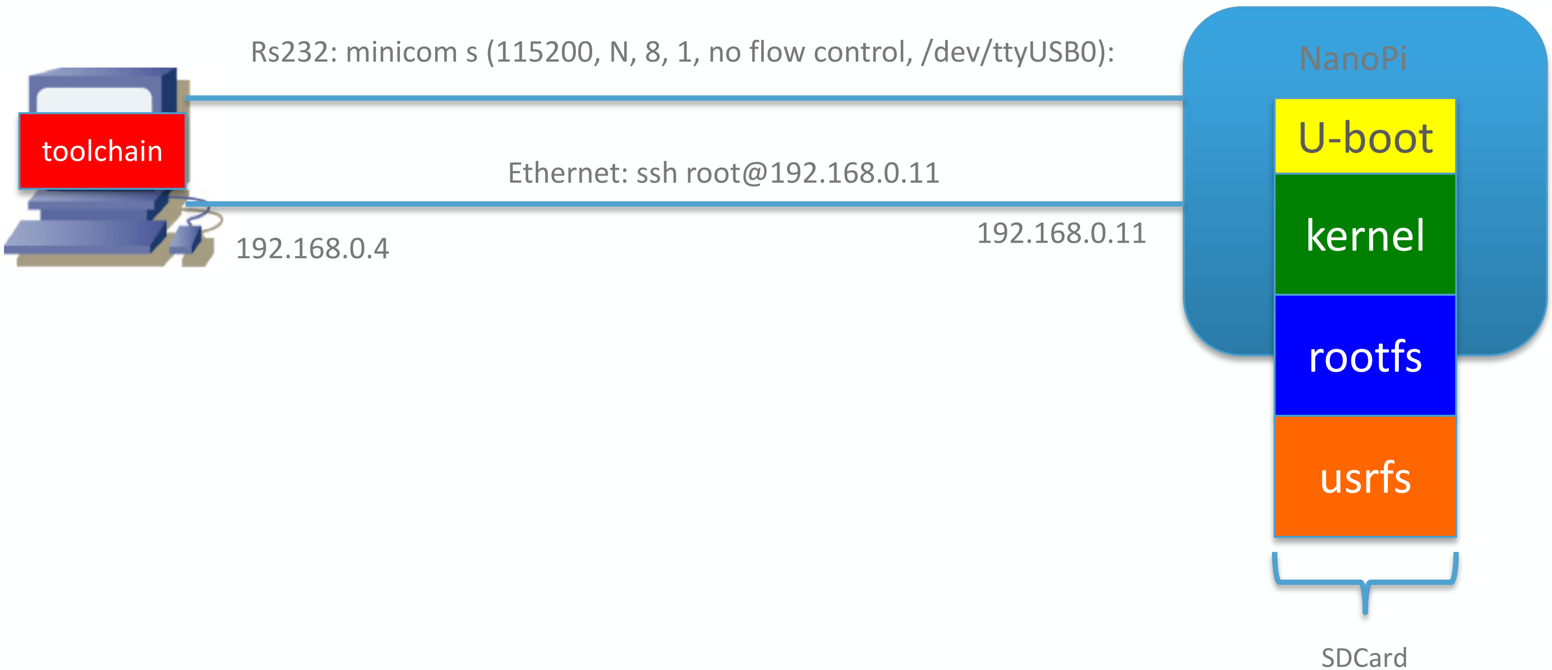


Nano Pi NEO Plus2 [1]

- SoC: Allwinner H5, Quad-core 64-bit high-performance Cortex A53
- DDR3 RAM:1GB
- Storage: 8GB eMMC
- Network: 10/100/1000M Ethernet based on RTL8211E-VB-CG
- WiFi: 802.11b/g/n
- Bluetooth: 4.0 dual mode
- MicroSD: 1 x slot supporting system booting
- Audio Input/Output: 5 pins
- MicroUSB: power input
- Debug Serial: 4 pins
- GPIO1: 24 pins (UART, SPI, I2C and IO)
- GPIO2: 12 pins (USB, IR receiver, I2S and IO)
- Power Supply: DC 5V/2A
- PCB Dimension: 40 x 52mm
- PCB Layer: 6-Layer



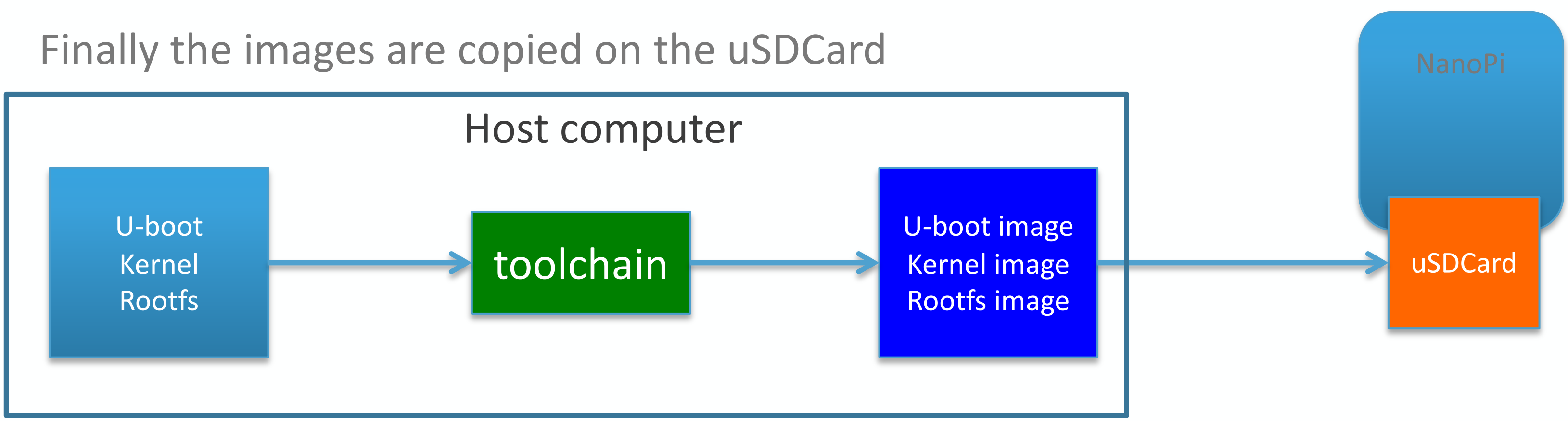
NanoPi: development environment



- Toolchain: All source codes and tools used to generate image running on the embedded system
- Kernel: Linux kernel with the u-boot format
- Rootfs: Root filesystem, with all directories and tools used by Linux
- Usrfs: User filesystem for the specific applications of the embedded system (smartphone, wifi router, …

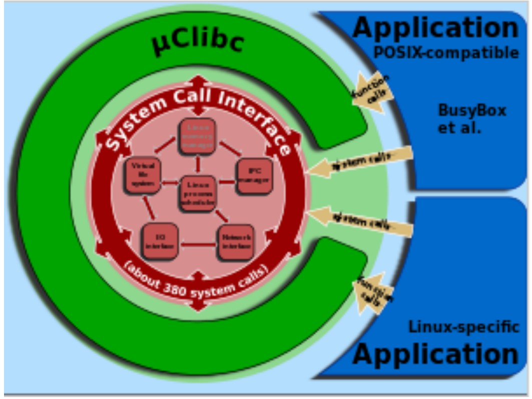
NanoPi: Toolchain

- NanoPi uses an ARM processor.
- Generally the development is doing on a x86/i64 processor and the code generated is downloaded to the NanoPi.
- In order to build the u-boot, Linux kernel, rootfs and customized application, a build ARM-toolchain is needed.
- The build ARM-toolchain includes a gcc cross-toolchain (x86/i64 cross compiler which generates code for NanoPi
- Buildroot [2] is the build ARM-toolchain used
- When this toolchain is ready, it is necessary to compile the bootloader, the kernel and the rootfs
- Finally the images are copied on the uSDCard



Buildroot [2]

- Buildroot is a set of Makefile and patches that simplifies and automates the process of building a complete Linux system for an embedded system
- In order to achieve this, Buildroot is able to build a required cross-compilation toolchain, create a root filesystem, compile a Linux kernel image, and generate a boot loader for the targeted embedded system.
- Buildroot uses uClibc which is much smaller than the glibc, the C library normally used with Linux distributions.
- uClibc runs on standard and MMU-less Linux systems.

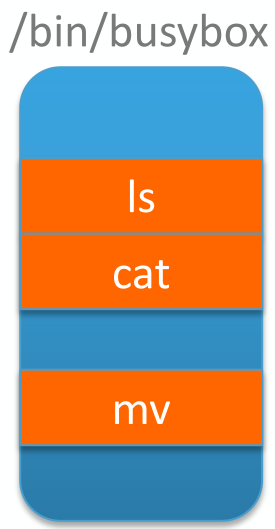


Busybox [http://en.wikipedia.org/wiki/BusyBox]

- BusyBox is a single binary, which is a conglomerate of many applications
- The desired command names are linked to the BusyBox executable

```
ln -s /bin/busybox /bin/ls
```

- After `ls` is used.



Summary buildroot, rootfs, Linux, u-boot installation (1) [3]

In order to

- install the buildroot environment,
- generate the rootfs,
- Compile the linux kernel
- Compile the u-boot

Summary of the script installation:

```
cd ~
mkdir –p ~/workspace
cd workspace
git clone https://gitlab.forge.hefr.ch/embsys/linuxenv.git
```

Summary buildroot, rootfs, Linux, u-boot installation (2) [4]

Scripts are copied in the directory: ~/workspace/linuxenv.

```
cd ~/workspace/linuxenv
```

You can execute this script:

```
./linuxenv/nano-env
```

Or executes these commands (`nano-env` script main commands) :

```
mkdir -p ~/workspace/nano
cd ~/workspace/nano
git clone git://git.buildroot.net/buildroot
cd buildroot
git checkout -b nano 2020.05.1
patch -p1 < ~/workspace/linuxenv/config/buildroot_nano.patch
make friendlyarm_nanopi_neo_plus2_defconfig
```

Compile the nanoPi development environment:

```
cd ~/workspace/nano/buildroot
make      // Make command lasts a long time (> 1h)
```

Directories installation [2]

~/workspace → working space
/nano → working space for NanoPi
/buildroot → space for tools, kernel, rootfs generation
/board/friendlyarm/nanopi-neo-plus2→ genimage.cfg, boot.cmd
/dl → downloaded « tared » packets: e.g. busybox-1.30.1.tar.bz2
/system/skeleton → Rootfs skeleton
/output
/build → source codes and compiled packets, e.g.: linux-5.1.16
/images → Image, nanopi-neo-plus2.dtb, rootfs.ext4, u-boot.itb, boot.scr, sunxi-spl.bin
/target → rootfs not "tared"
/host/usr/bin → cross-compiler: aarch64-linux-gnu-gcc, …

Files u-boot.itb, sunxi-spl.bin, Image, nanopi-neo-plus2.dtb, rootfs.ext4, boot.scr will be copied to the uSD card.

In order to cross-compile, link, … , add the
PATH=$PATH:/home/<USER>/workspace/nano/buildroot/output/host/usr/bin

Remark: it is better to indicate an absolute path (not a relative path with ~/workspace/…)

Buildroot: Rebuild packages (1)

See: https://buildroot.org/downloads/manual/manual.html#configure

How it is possible to rebuild a given package or how to remove a package without rebuilding everything from scratch.

Package sources are in this directory:
`~/workspace/nano/buildroot/output/build`

- Re-install a package: The easiest way to rebuild a single package from scratch is to remove its build directory in output/build. Buildroot will then re-extract, re-configure, re-compile and re-install this package from scratch, example: re-install u-boot:

```
cd ~/workspace/nano/buildroot/
rm –rf output/build/uboot-2020.07
make
```

Buildroot: Rebuild packages (2)

- Rebuild a package, possibility 1: run this command: `make <package>-rebuild`
  The package name: see the directory: `output/build/<package>-<version>`
  ```
  Example:      cd ~/workspace/nano/buildroot/
                make uboot-rebuild
  ```

- Rebuild a package, possibility 2: buildroot uses .stamp-<step> files to indicate different stages. Delete the .stamp_built, example: uboot
  ```
  cd ~/workspace/nano/buildroot/
  rm output/build/uboot-2020.07/.stamp-built
  make
  ```

Buildroot: Add file to rootfs

Overlay directory is used to add some files or directories to the rootfs. By default overlay directory is : `$HOME/workspace/nano/buildroot/board/friendlyarm/nanopi-neo-plus2/rootfs_overlay`

Configure Buildroot:

```
cd ~/workspace/nano/buildroot
make menuconfig
   System configuration → Root filesystem overlay directories
     board/friendlyarm/nanopi-neo-plus2/rootfs_overlay
```

If not exist, create this directory:

```
mkdir ~/workspace/nano/buildrootboard/friendlyarm/nanopi-neo-plus2/rootfs_overlay
```

Put inside the overlay, directories and files added to rootfs, example the init script `S45iptables` is added to directory /etc/init.d in rootfs

```
$HOME/workspace/nano/buildroot/board/friendlyarm/nanopi-neo-plus2/rootfs_overlay/etc/init.d/S45iptables
```

Configure and compile buildroot

Compile

```
cd ~/workspace/nano/buildroot
make help                         // Different commands

make menuconfig                   // to configure buildroot
```

Compile

```
cd ~/workspace/nano/buildroot
make                             // Compile all
```

```
Other commands
make friendlyarm_nanopi_neo_plus2_defconfig  // Only the first time
make clean                        // Be careful: delete output
                                  // directory and the .o files
                                  // → compile all files
make V=1                          // print commands
```

Configure and compile u-boot

Configure:

```
cd ~/workspace/nano/buildroot
make uboot-menuconfig
```

Compile: 2 possibilities:

```
1)
cd ~/workspace/nano/buildroot
make uboot-rebuild

2)
cd ~/workspace/nano/buildroot/
rm output/build/uboot-2019.01/.stamp-built
make
```

Configure and compile linux

Configure:

```
cd ~/workspace/nano/buildroot
make linux-menuconfig or      // to configure the linux kernel
make linux-xconfig            // to configure the linux kernel
```

Compile

```
cd ~/workspace/nano/buildroot
make linux-rebuild
```

Configure and compile busybox

Configure

```
cd ~/workspace/nano/buildroot
make busybox-menuconfig
```

Compile

```
cd ~/workspace/nano/buildroot
make busybox-rebuild
```

Rootfs generation

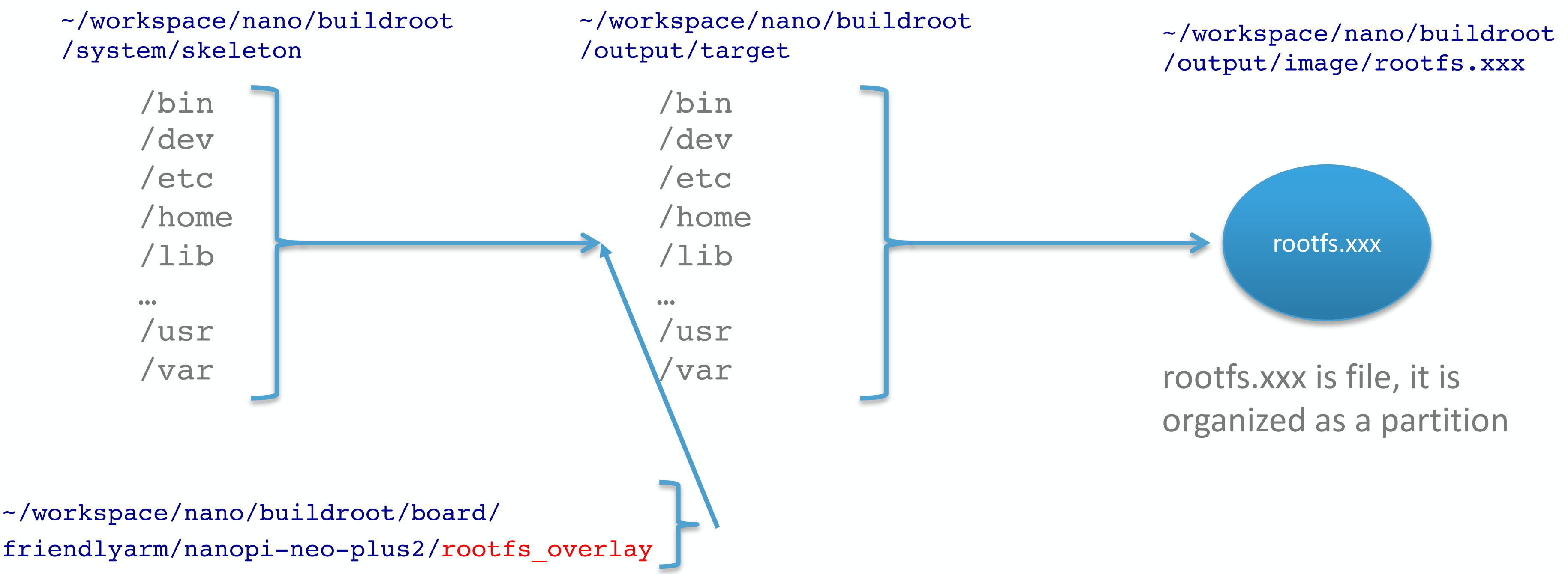
- The rootfs configuration is made by Buildroot
- cd ~/workspace/nano/buildroot
- make menuconfig → Filesystem Images →

```
[ ] cloop root filesystem for the target device
[ ] cpio the root filesystem (for use as an initial RAM filesyste
[ ] cramfs root filesystem
[*] ext2/3/4 root filesystem
      ext2/3/4 variant (ext4)  --->
(0)   size in blocks (leave at 0 for auto calculation)
(0)   inodes (leave at 0 for auto calculation)
(0)   reserved blocks percentage
      Compression method (no compression)  --->
[ ] initial RAM filesystem linked into linux kernel
↓(+)
```

Rootfs generation

- A rootfs skeleton is in the directory `~/workspace/nano/buildroot/system/skeleton`
- This skeleton is copied to the pseudo rootfs directory `~/workspace/nano/buildroot/output/target`
- After make command the pseudo rootfs is populated and copied to one file in this directory

  `~/workspace/nano/buildroot/output/image/rootfs.xxx (xxx can be ext4, squashfs, …)`



Read rootfs

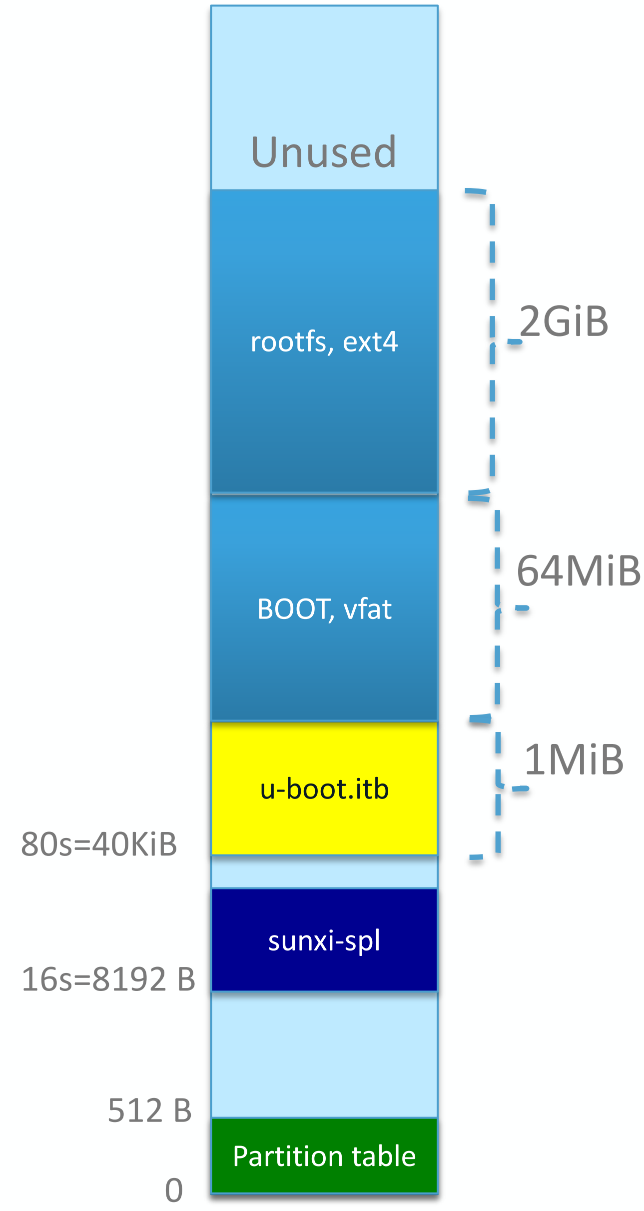
- With the mount loop option, it is possible to mount the rootfs.xxx file to /mnt
- Example: rootfs.ext4

```
# sudo mount –t ext4 –o loop rootfs.ext4 /mnt
# ls /mnt
    bin  etc   lib   linuxrc     media opt  root sbin tmp var
    dev  home  lib32 lost+found  mnt   proc run  sys  usr
```

SDCard Mapping

Sector size = 512 Bytes

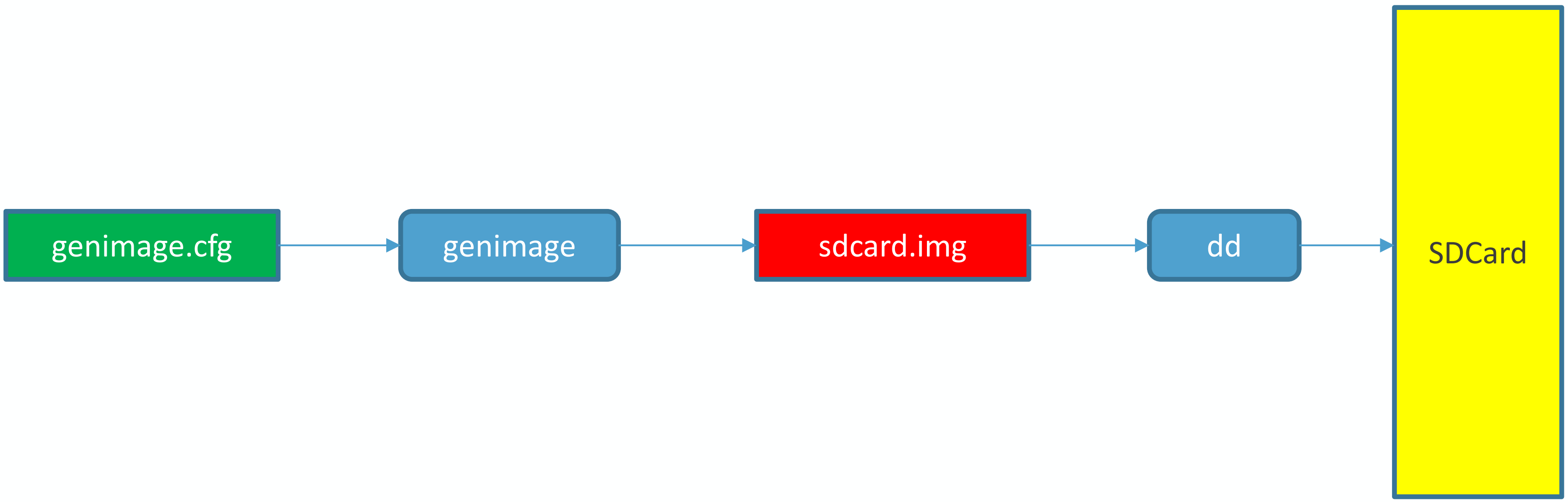
Area Name	From (Sector #)	To (Sector #)	Size
rootfs, ext4			**2GiB**
BOOT, vfat			**64MiB**
U-Boot	**80**		**1MiB**
Sunxi-spl.bin	**16**	**79**	**32KiB**
MBR (partition table)	**0**	**15**	**512B**



Initialize SDCard with genimage (1)

Genimage initializes the SDCard

Genimage reads the file genimage.cfg and creates the file sdcard.img. This file is copied to SDCard with the standard command dd




```
genimage:     ~/workspace/nano/buildroot/output/host/bin/genimage
genimage.cfg: ~/workspace/nano/buildroot/board/friendlyarm/nanopi-neo-plus2/genimage.cfg
sdcard.img:   ~/workspace/nano/buildroot/output/images/sdcard.img
```

Initialize SDCard with genimage (2)

`cat ~/workspace/nano/buildroot/board/friendlyarm/nanopi-neo-plus2/genimage.cfg`

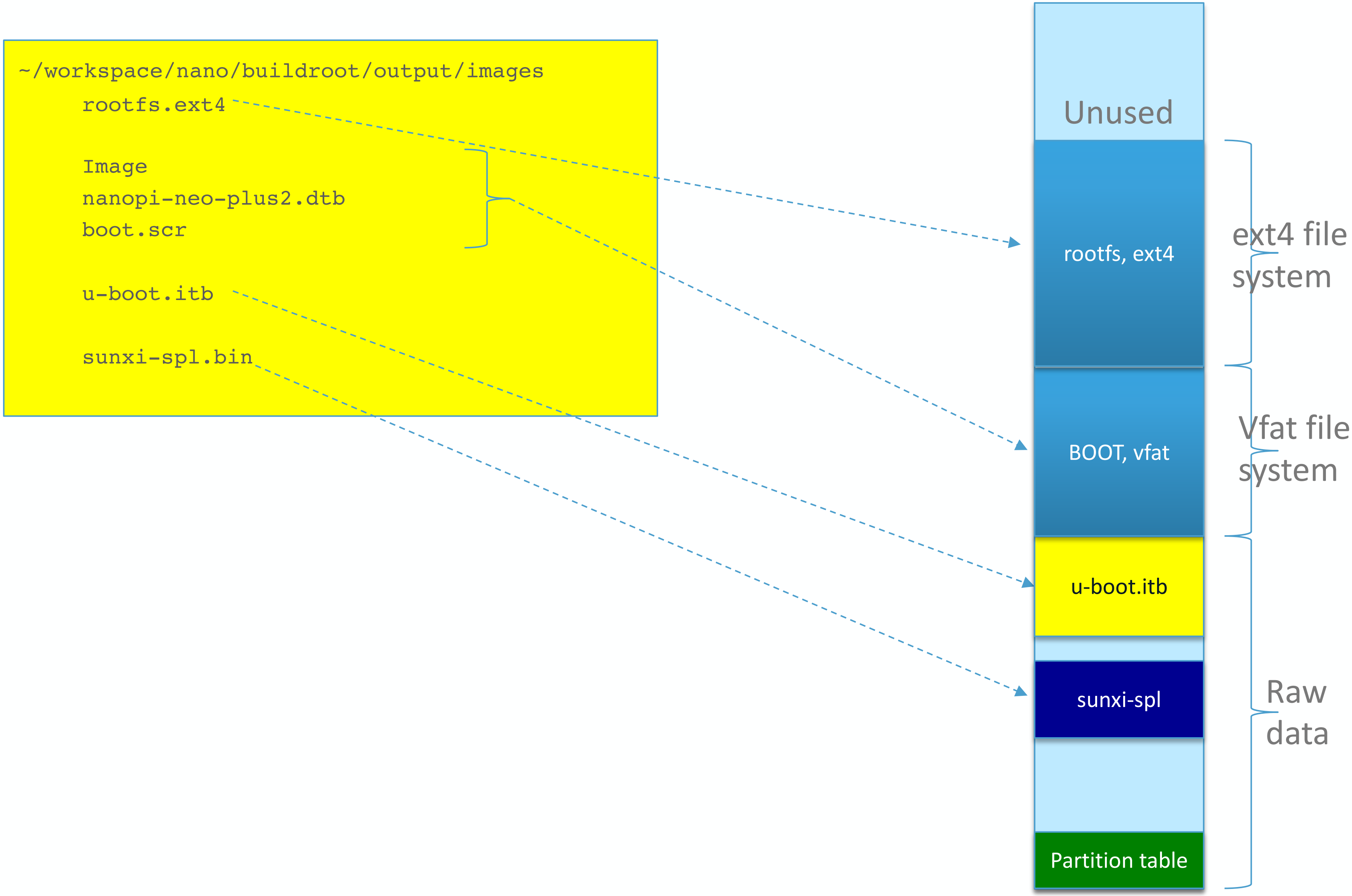
```
image boot.vfat {
    vfat {
         files = {
            "Image",
            "nanopi-neo-plus2.dtb",
            "boot.scr"
         }
    }
    size = 64M
}

image sdcard.img {
    hdimage {
    }

    partition spl {
        in-partition-table = "no"
        image  = "sunxi-spl.bin"
        offset = 8192
    }

    partition u-boot {
        in-partition-table = "no"
        image  = "u-boot.itb"
        offset = 40K
        size   = 1M # 1MB - 40K
    }

    partition boot {
        partition-type = 0xC
        bootable = "true"
        image = "boot.vfat"
    }

    partition rootfs {
        partition-type = 0x83
        image = "rootfs.ext4"
        size = 2G
    }
}
```

Initialize SDCard with genimage (3)

The SDCard is initialized with different files which are in this directory:

```
~/workspace/nano/buildroot/output/images
    rootfs.ext4                         // root files system
    Image                               // Linux kernel
    nanopi-neo-plus2.dtb                // Flattened device tree
    boot.scr                            // compiled boot commands used by u-boot
    boot.vfat                           // BOOT partition, contains Image,
                                           nanopi-neo-plus2.dtb, boot.scr
    u-boot.itb                          // u-boot: boot loader
    sunxi-spl.bin                       // Secondary Program Loader
```

Initialize SDCard with genimage (4)



boot.scr file

The boot.scr file is used by uboot in order to load Linux kernel

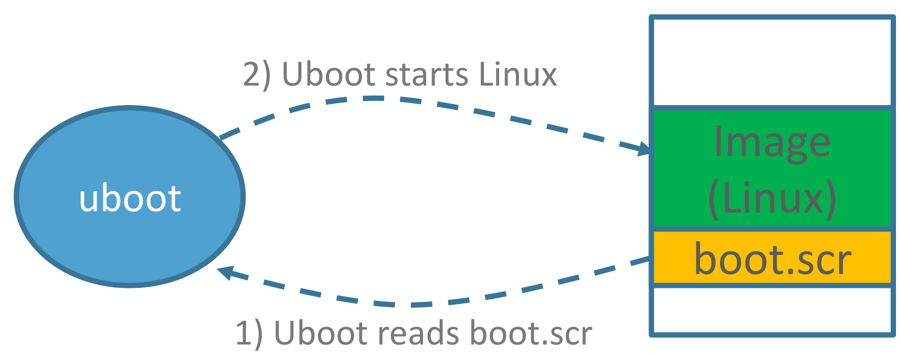
Boot.scr is created by this command:

```
~/workspace/nano/buildroot/output/host/bin/mkimage -C none -A arm -T script –d
            ~/workspace/nano/buildroot/board/friendlyarm/nanopi-neo-plus2/boot.cmd
            ~/workspace/nano/buildroot/output/image/boot.scr
```

Remark: -C option must be before –A option

cat boot.cmd

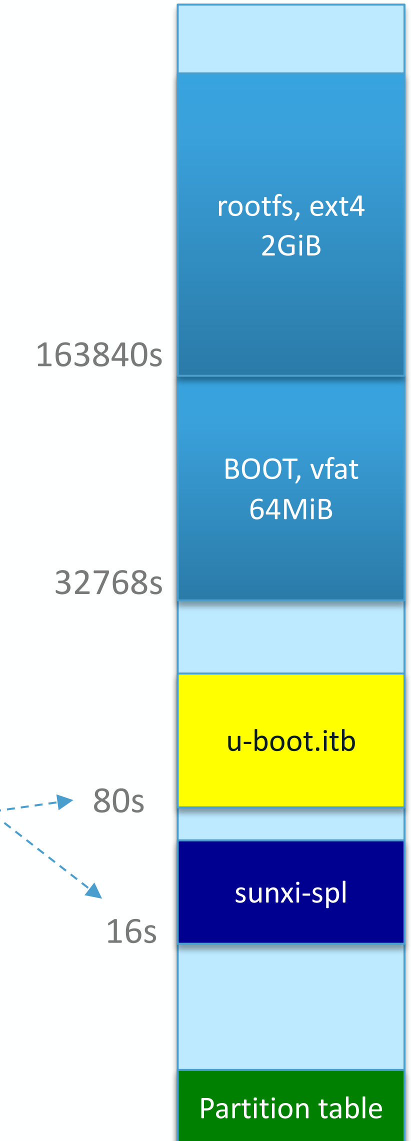
```
setenv bootargs console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 rootwait

fatload mmc 0 $kernel_addr_r Image
fatload mmc 0 $fdt_addr_r nanopi-neo-plus2.dtb

booti $kernel_addr_r - $fdt_addr_r
```



Manually initialize SDCard (1)

The offset option in the `~/workspace/nano/buildroot/board/friendlyarm/nanopi-neo-plus2/genimage.cfg` file indicates in bytes where data is loaded

```
partition spl {
        in-partition-table = "no"
        image  = "sunxi-spl.bin"
        offset = 8192 → in sector: 8192/512 = 16
    }

    partition u-boot {
        in-partition-table = "no"
        image  = "u-boot.itb"
        offset = 40K → in sector: 40*1024/512 = 80
        size   = 1M # 1MB - 40K
    }
```



0

Manually initialize SDCard (1)

```
umount /dev/sdbX
```
// X=1, 2, 3, …, be careful /dev/sda, /dev/sdb, /dev/sdc

#initialize 400MiB to 0

```
sudo dd if=/dev/zero of=/dev/sdb bs=4k count=100000
sync
```

First sector: msdos

```
sudo parted /dev/sdb mklabel msdos
```

#copy sunxi-spl.bin binaries

```
sudo dd if=~/workspace/nano/buildroot/output/images/sunxi-spl.bin of=/dev/sdb
bs=512 seek=16
```

#copy u-boot

```
sudo dd if=~/workspace/nano/buildroot/output/images/u-boot.itb of=/dev/sdb
bs=512 seek=80
```

Manually initialize SDCard (2)

1st partition: 64MiB: (163840-32768)*512/1024 = 64MiB

```
sudo parted /dev/sdb mkpart primary fat32 32768s  163839s
```

2nd partition: 1GiB: 4358144-163840)*512/1024 = 1GiB

```
sudo parted /dev/sdb mkpart primary ext4 163840s 4358143s

sudo mkfs.vfat /dev/sdb1
sudo mkfs.ext4 /dev/sdb2 -L rootfs
sync
```

#copy kernel, flattened device tree, boot.scr

```
sudo mount /dev/sdb1 /run/media/schuler/
sudo cp ~/workspace/nano/buildroot/output/images/Image /run/media/schuler
sudo cp ~/workspace/nano/buildroot/output/images/nanopi-neo-plus2.dtb
/run/media/schuler
sudo cp ~/workspace/nano/buildroot/output/images/boot.scr /run/media/schuler
sync
```

Manually initialize SDCard (3)

#Rename 1st partition tom BOOT

```
sudo umount /dev/sdb1
sudo fatlabel /dev/sdb1 BOOT
```

#copy rootfs

```
sudo dd if=~/workspace/nano/buildroot/output/images/rootfs.ext4 of=/dev/sdb2
```

Resize and rename 2nd partition to rootfs

check if the partition must be mounted

```
sudo e2fsck -f /dev/sdb2
sudo resize2fs /dev/sdb2
sudo e2label /dev/sdb2 rootfs
```

Boot sequence [Cours CSEL, D. Gachet]

- **Le démarrage du NanoPi NEO Plus2 se décompose en 6 phases:**
  - Lorsque le µP est mis sous tension, le code stocké dans son BROM va charger dans ses 32KiB de SRAM interne le firmware « sunxi-spl » stocké dans le secteur nº 16 de la carte SD / eMMC et l'exécuter.
  - Le firmware « sunxi-spl » (Secondary Program Loader) initialise les couches basses du µP, puis charge l'U-Boot dans la RAM du µP avant de le lancer.
  - L'U-Boot va effectuer les initialisations hardware nécessaires (horloges, contrôleurs, …) avant de charger l'image non compressées du noyau Linux dans la RAM, le fichier «Image», ainsi que le fichier de configuration FDT (flattened device tree).
  - L'U-Boot lancera le noyau Linux en lui passant les arguments de boot (bootargs).
  - Le noyau Linux procédera à son initialisation sur la base des bootargs et des éléments de configuration contenus dans le fichier FDT (sun50i-h5-nanopi-neo-plus2.dtb).
  - Le noyau Linux attachera les systèmes de fichiers (rootfs, tmpfs, usrfs, …) et poursuivra son exécution.

Boot sequence [Cours CSEL, D. Gachet]