

Armbian documentation

Linux for ARM development boards

Armbian documentation team

2020 by Armbian

Table of contents

| | |
|---|----|
| 1. Welcome to the Armbian Documentation! | 4 |
| 2. What is Armbian? | 4 |
| 3. What is supported? | 6 |
| 4. Get Involved! | 6 |
| 5. User Guide | 7 |
| 5.1 Prerequisites for new users | 7 |
| 5.2 What is Armbian Linux? | 15 |
| 5.3 Challenges | 15 |
| 5.4 Benefits | 15 |
| 5.5 Hardware troubleshooting guide | 17 |
| 5.6 How to set wireless access point? | 23 |
| 5.7 How to connect IR remote? | 23 |
| 5.8 How to customize keyboard, time zone? | 25 |
| 5.9 Armbian configuration utility | 28 |
| 5.10 Device Tree overlays | 31 |
| 5.11 Migration from Bananian to Armbian | 36 |
| 6. Hardware Notes | 39 |
| 6.1 Enable Hardware Features | 39 |
| 6.2 Generic howto for Allwinner devices | 39 |
| 6.3 H3 based Orange Pi, legacy kernel | 41 |
| 6.4 Allwinner A10 & A20 boards | 45 |
| 6.5 Allwinner H3 boards | 48 |
| 6.6 Allwinner H5 and A64 boards | 51 |
| 6.7 Allwinner H6 | 52 |
| 6.8 Cubox and Hummingboard boards | 53 |
| 6.9 GPIO | 54 |
| 6.10 Udoo Quad | 55 |
| 6.11 Bugs | 55 |
| 6.12 Udoo Neo | 55 |
| 6.13 Helios4 | 56 |
| 7. Developer Guide | 57 |
| 7.1 What do I need? | 57 |
| 7.2 How to start? | 57 |
| 7.3 Quick Start with Vagrant | 61 |
| 7.4 Officially supported and tested method for building with Docker | 63 |

| | |
|--|----|
| 7.5 Creating and running Docker container manually | 63 |
| 7.6 FEL/NFS boot explanation | 73 |
| 8. Contributor Process | 76 |
| 8.1 Collaborate on the project | 76 |
| 8.2 Help with donations | 76 |
| 8.3 Merge Policy | 77 |
| 8.4 Armbian documentation hosted on Github pages | 81 |
| 9. Release management | 82 |
| 9.1 Release model | 82 |
| 9.2 Release Branching, Versioning and Tags | 83 |
| 9.3 Release Naming | 83 |
| 9.4 Release Planning | 83 |
| 9.5 Release Coordinating | 84 |
| 9.6 Release Testing | 85 |
| 9.7 Reflection on Prior Releases | 86 |
| 9.8 | 86 |
| 10. Community | 88 |

armbian

Linux for ARM development boards

1. Welcome to the Armbian Documentation!

If you are **new to Armbian**, the [Getting Started](#) section provides a tutorial for everything you need to get Armbian running, and answers many **Frequently Asked Questions**. It then continues on to more advanced topics.

If you **need help**, and have read through *Getting Started*, check out [Troubleshooting](#).

If you still cannot find what you need here, visit the [Armbian forum](#), where your input can help improve this documentation.

2. What is Armbian?

Armbian is a base operating system platform for single board computers (SBCs) that other projects can trust to build upon.

- Lightweight Debian or Ubuntu based linux distribution specialized for ARM development boards
- Each system is compiled, assembled and optimized by [Armbian Build Tools](#)
- It has powerful build and software development tools to make [custom builds](#)
- A vibrant community
- What is the [difference between](#) Armbian and Debian Linux

2.0.1 Common features

- Armbian Linux is available as Debian and Ubuntu based images, compiled from scratch
- Images are reduced to actual data size and automatically resized at first boot
- Root password is `1234`. You are forced to change this password and (optional) create a normal user at first login
- Ethernet adapter with DHCP and SSH server ready on default port (22)
- Wireless adapter with DHCP ready (if present) but disabled. You can use `armbian-config` to connect to your router or create an AP
- NAND, SATA, eMMC and USB install script is included (`nand-sata-install`)
- Upgrades are done via standard `apt upgrade` method
- Login script shows: board name with large text, distribution base, kernel version, system load, uptime, memory usage, IP address, CPU and drive temperature, ambient temperature from Temper if exists, SD card usage, battery conditions and number of updates to install

2.0.2 Performance tweaks

- `/var/log` is mounted as compressed device (zram, lzo), log2ram service saves logs to disk daily and on shutdown
- Half of memory is allocated/extended for/with compressed swap
- `/tmp` is mounted as `tmpfs` (optionally compressed)
- Browser profile memory caching
- Optimized IO scheduler (check `/etc/init.d/armhwinfo`)
- Journal data writeback enabled. (`/etc/fstab`)
- `commit=600` to flush data to the disk every 10 minutes (`/etc/fstab`)
- Optimized CPU frequency scaling with `interactive` governor (`/etc/init.d/cpufrequtils`)
 - 480-1010Mhz @Allwinner A10/A20
 - 480-1260Mhz @Allwinner H3
 - 392-996Mhz @Freescale imx
 - 600-2000Mhz @Exynos & S905
- eth0 interrupts are using dedicated core (Allwinner based boards)

3. What is supported?

“Supported” is not a guarantee. “Supported” implies a particular SBC is at a high level of software maturity, but has no intention to support all possible SBC functions. Supported boards do receive preferential treatment to bugfix, improve, or add additional functionality based on any of the following, **non-exclusive** criteria:

1. The discretion of the “Armbian Development Team”
2. The availability of the “Armbian Development Team”
3. The availability of sample boards and ease of testing
4. The mainline kernel maturity for the particular SoC or SBC platform
5. Paid engagements, long-term sponsorship to the Armbian Project or volunteer developers
6. Vendor or 3rd party has a **designated** resource providing support for a SBC or platform ON BEHALF OF THE COMMUNITY and is contributing to the project

3.0.1 Supported chips

- Allwinner A10, A20, A31, H2+, H3, H5, H6, A64
- Amlogic S805 and S905 (Odroid boards), S802/S812, S805, S905, S905X and S912 (fork by [@balbes150](#))
- Actionsemi S500
- Freescale / NXP iMx6
- Marvell Armada A380
- Rockchip RK3288
- Samsung Exynos 5422

3.0.2 Supported boards

Check [download page](#) for recently supported list.

4. Get Involved!

- [Contribute](#)
- [Community](#)
- [Contact](#)

Our IRC channel is [#armbian](#) on [freenode](#).

5. User Guide

5.1 Prerequisites for new users

Please, make sure you have:

- a proper power supply according to the board manufacturer requirements (basic usage example: 5V/2A with DC Jack barrel OR **thick** USB cable)
- a reliable SD card (see below “How to prepare a SD card?”)

What to download?

The download for each image consists of three separate files: An **xz-compressed image file**, a **sha file** for download verification and an **asc file** for image authentication.

For each board we usually provide:

- one CLI Debian **and** one CLI Ubuntu based server image,
- one desktop Ubuntu Bionic **or** Debian Buster

Other unsupported builds may also be available (like Debian Stretch/Bullseye Ubuntu Disco/Edoan).

Some boards have different options due to their hardware specialities - router or IoT boards.

Legacy or current?

Only *current* kernel branch is considered fully supported and can bring up video acceleration for example. NAND support is there but is still experimental.

The level of kernel support does depend on the board family. If in your specific case something does not work well, you are always free to try an image with *legacy* kernel included.

What are testing images?

- made from stable branches
- not very well tested
- for end users

What are experimental/dev images?

- made from unstable branches
- untested
- for experienced users only

Do not use testing or dev images in a productive environment. We do appreciate your constructive [feedback to developers](#).

How to check download authenticity?

All our images are digitally signed and therefore it is possible to check their authenticity. You need to issue these commands (Linux/macOS, you might need to install dependencies first, eg. `apt-get install gnupg` on Debian/Ubuntu or `brew install gnupg` on macOS. on windows install the current simple gnupg [Gnupg](#):

```
# download public key from the database gpg --keyserver ha.pool.sks-keyservers.net --recv-key DF00FAF1C577104B50BF1D0093D6889F9F0E78D5 # perform verification
```

It is safe to ignore the message `WARNING: This key is not certified with a trusted signature!`.

How to check download integrity?

Since it might happen that your download got somehow corrupted we integrate a checksum/hash for the image. You can compare the image's SHA-256 hash with the one contained in the `sha256sum.sha` file.

On Windows, you can download and use the [QuickHash GUI](#) and follow the instructions in the gui.

while on Linux/macOS, in the directory in which you have downloaded the files ,you would do this



```
shasum -a 256 -c Armbian_*.img.sha Armbian_*.img.xz #good response Armbian_5.35_Clearfogpro_Debian_stretch_next_4.13.16.img: OK
```


How to prepare a SD card?

Important note: Make sure you use a **good, reliable and fast** SD card. If you encounter boot or stability troubles in over 95 percent of the time it is either insufficient power supply or related to SD card (bad card, bad card reader, something went wrong when burning the image, card too slow to boot – ‘Class 10’ highly recommended!). Armbian can simply not run on unreliable hardware so checking your SD card with either [F3](#) or [H2testw](#) is mandatory if you run in problems. Since [counterfeit SD cards](#) are still an issue checking with F3/H2testw directly after purchase is **highly recommended**.

Write the xz compressed image with [USBImager](#) or [Etcher](#) on all platforms since unlike other tools, either can validate burning results **saving you from corrupted SD card contents**.

Also important: Most SD cards are only optimised for sequential reads/writes as it is common with digital cameras. This is what the *speed class* is about. The SD Association defined [Application Performance Class](#) as a standard for random IO performance.

| Application Performance Class | Pictograph | Minimum Random Read | Minimum Random Write | Minimum Sustained (Seq. Write) |
|-------------------------------|---|---------------------|----------------------|--------------------------------|
| Class 1 (A1) |  | 1500 4k IOPS | 500 4k IOPS | 10MBytes/sec |
| Class 2 (A2) |  | 4000 4k IOPS | 2000 4k IOPS | 10MBytes/sec |

At the time of this writing A1 and A2 cards are only widely available from SanDisk. Armbian recommends A1 rated SD-Cards **only** now ([A2 rated cards need yet lacking driver support and therefore show lower overall and especially random IO performance](#)). For example:



In case you chose an SD card that was already in use before please consider resetting it back to 'factory default' performance with [SD Formatter](#) before burning Armbian to it ([explanation in the forum](#)). Detailed information regarding 'factory default' SD card performance.

How to boot?

Insert SD card into a slot and power the board. (First) boot (with DHCP) takes up to two minutes with a class 10 SD card and cheapest board.

How to login?

Login as **root** on console (HDMI / serial) or via SSH and use password **1234**. You will be prompted to change this password at first login. You will then be asked to create a normal user account that is sudo enabled (beware of default QWERTY keyboard settings at this stage). Please use [this tool](#), to find your board IP address.

Desktop images start into desktop without asking for password. To change this add some display manager:

```
apt-get install lightdm
```

... or edit the contents of file:

```
/etc/default/nodm
```

and change the autologin user.

How to update?

```
apt update apt upgrade
```

Update process can take hours in case of using cheap SD card and/or under heavy load.

If the kernel was upgraded during this process you will be prompted to reboot at next login.

How to update u-boot?

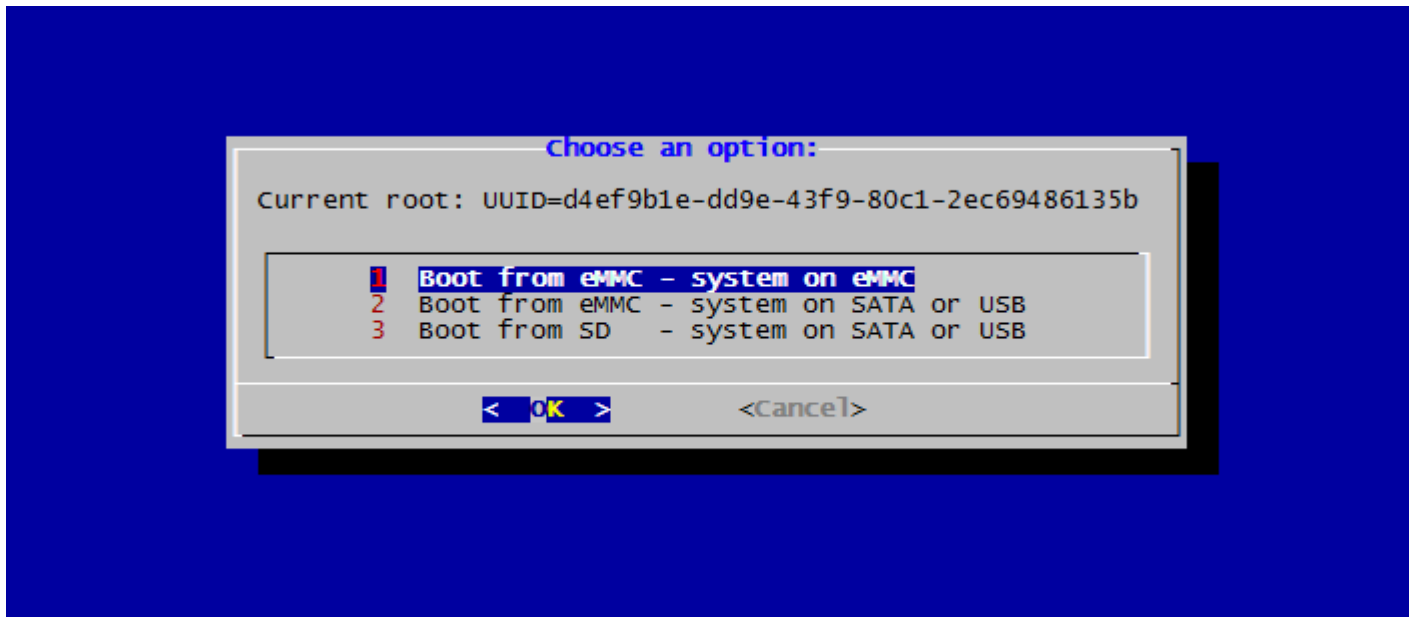
First you need to update packages described in a previous “How to update” step. Then run armbian-config utility, go to system settings and proceed to:

“Install” “Install to/update boot loader” -> Install/Update the bootloader on SD/eMMC

How to adjust hardware features?

Use the Armbian configuration utility `armbian-config`

How to install to eMMC, NAND, SATA & USB?



Required condition:

NAND:

- kernel 3.4.x and NAND storage
- pre-installed system on NAND (stock Android or other Linux)

eMMC/SATA/USB:

- any kernel
- onboard eMMC storage
- attached SATA or USB storage

Start the install script:

```
nand-sata-install
```

and follow the guide. You can create up to three scenarios:

- boot from SD, system on SATA / USB
- boot from eMMC / NAND, system on eMMC/NAND
- boot from eMMC / NAND, system on SATA / USB

and you can choose the following file system options:

- ext2,3,4
- btrfs

On Allwinner devices after switching to boot from NAND or eMMC clearing the boot loader signature on the SD card is recommended:

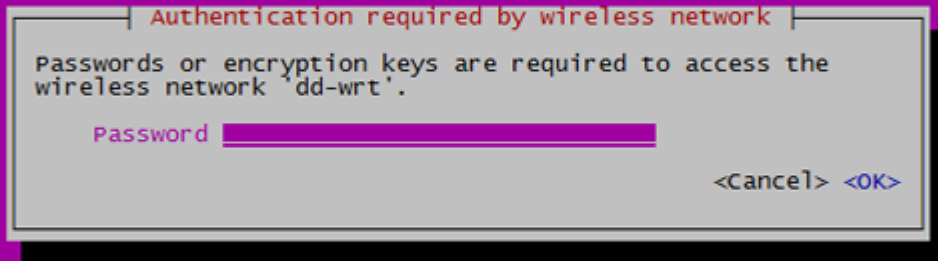
`dd if=/dev/zero of=/dev/mmcblkN bs=1024 seek=8 count=1` (replace `/dev/mmcblkN` with the correct device node – in case you run this directly after `nand-sata-install` without a reboot in between then it's `/dev/mmcblk0`). When booting from eMMC to get SD cards auto-detected on Allwinner legacy images please consider changing `mmc0`'s `sdcdetmode` from 3 to 1 in the board's fex file (see [here](#) for details).

How to connect to wireless?

Required condition: a board with onboard or supported 3rd party wireless adapter on USB

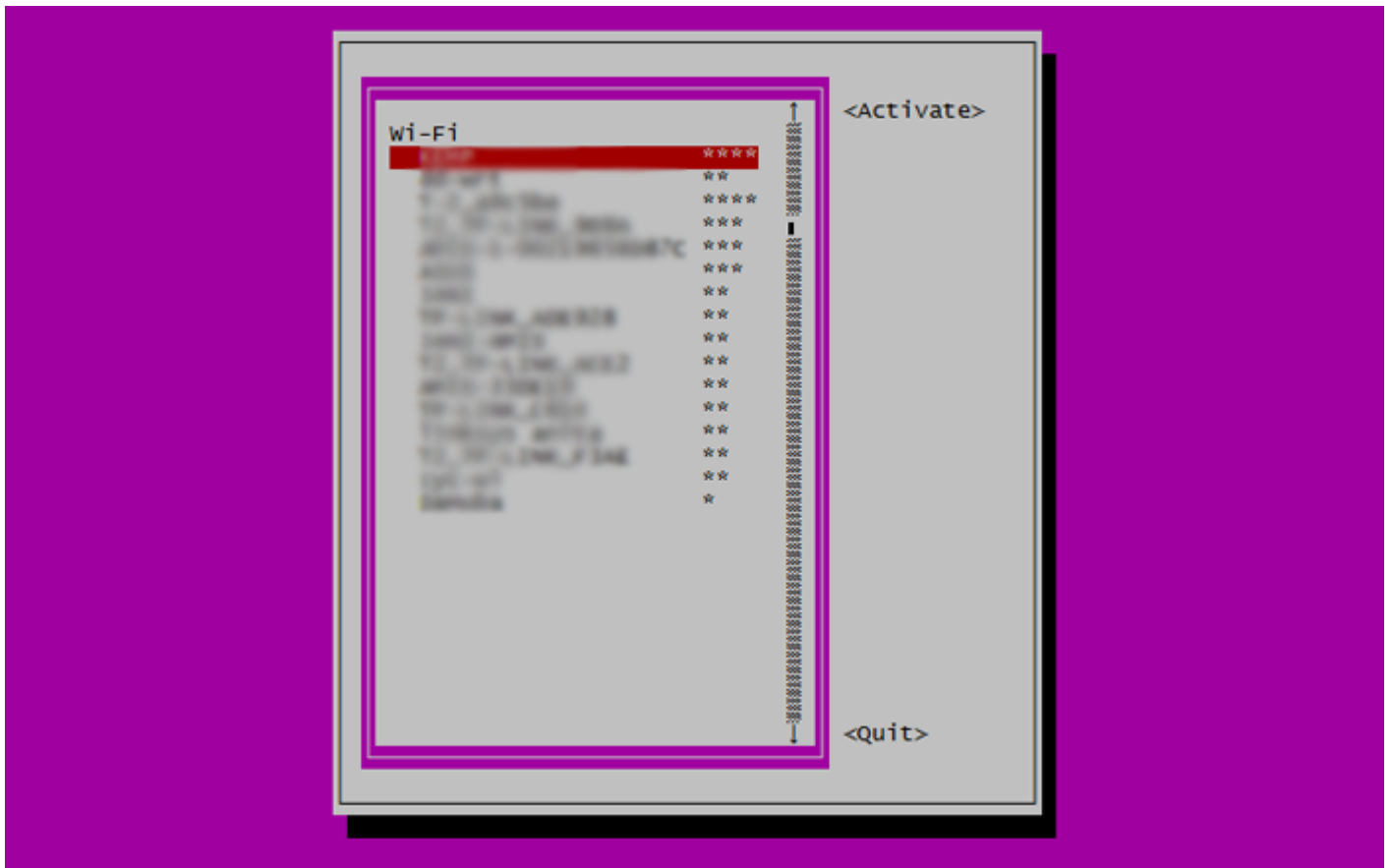
If you know what is your wireless SSID:

```
nmtui-connect SSID
```



If you do not know, you can browse and then connect

```
nmtui-connect
```



How to set fixed IP?

By default your main network adapter's IP is assigned by your router DHCP server and all network interfaces are managed by **NetworkManager**:

```
user@boardname:~$ nmcli con show NAME UUID TYPE DEVICE Wired connection 1 xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx 802-3-ethernet eth0
```

The connection can now be edited with the following:

```
nmcli con mod "Wired connection 1" ipv4.addresses "HOST_IP_ADDRESS" ipv4.gateway "IP_GATEWAY" ipv4.dns "DNS_SERVER(S)" ipv4.dns-search "DOMAIN_NAME"
```

The same changes can also be done with NetworkManagers text user interface:

```
sudo nmtui
```

5.2 What is Armbian Linux?

Armbian Linux provides optimized Debian and Ubuntu Linux images for ARM-based SBCs. There is an incredible ecosystem of small computing platforms that are powerful alternatives to the Raspberry Pi. Armbian's mission is to provide a uniform system offering that is trustworthy to run on any of the dozens of OS-neglected ARM single board computers.

5.3 Challenges

Armbian is the opposite of Raspbian

Raspbian has dozens of contributors to focus on a single SBC platform. Armbian has a dozen contributors to focus on 100+ SBCs spread over 30 platforms.

Balancing Development and Support

Given the point above, resources are thin. Armbian developers have to focus on the core mission of maintaining the [Armbian Build Platform](#). We heavily rely on other members of the community to support each other. Although Armbian does provide a lot of [user friendly features](#), the reality is that Armbian is for more advanced users. If you are really struggling with your SBC, you may want to consider first getting more comfortable with Raspbian Linux on the Raspberry Pi.

5.3.1 More SBCs continuously coming to market

SBC and TV Box manufacturers love to design and ship new products. Unfortunately they do not like to spend time on software and instead rely on community projects such as Armbian to fill in the gaps.

5.4 Benefits

Simple

BASH shell, standard Debian/Ubuntu utilities. Common and specific features can be with minimalistic menu-driven utility. Login is possible via serial, HDMI/VGA or SSH.

Light

No bloatware or spyware. Special utilities are completely optional. Suitable for newcomers and professionals.

Optimized

A distributed image is compacted to real data size and starts at around of 1G. Size is optimized for SD card usage. Bigger is better. Installing applications later severely reduces the life of your SD card. They were not designed for this type of usage.

Fast

Boards are optimized on kernel and userspace level. DVFS optimization, memory log caching, browser profile memory caching, swap usage tuning, garbage commit delay. Our system runs almost read-only and is one of the the fastest Linux for many development boards in just about every case.

Secure

Security level is on a stock Debian/Ubuntu level and can be hardened with the configuration utility. It provides a good starting point for industrial or home usage. The system is regularly inspected by professionals within the community. Each official stable build is thoroughly tested. Images are a direct base for all 3rd party builders.

Supported

Providing long term updates, security fixes, documentation, user support.

Smart

Deep understanding how boards work, how operating system work and how hardware should be designed to run better. Involved in board design. Experience in Linux since early 90'. Specialized in ARM development boards since 2013.

Open

Open source build script and kernel development, maintenance and distribution for more than [30 different ARM and ARM64 Linux kernels](#). Powerful build and software development tools. Can run in fully parallel mode. Can run under Docker.

5.5 Hardware troubleshooting guide

If you are experiencing at least one of these problems:

- board does not boot
- board freezes, crashes or reboots randomly or when connecting USB devices
- plugged in USB devices are not detected (not listed in `lsusb` output)
- error changing the root password at first boot (Authentication token manipulation error)
- error installing or updating packages due to read-only file system

and you are using a stable Armbian image, then most likely you have one of two common problems - **powering issue** or **SD card issue**.

Note that

- *“I know that my power supply is good”, “it worked yesterday”, “it works with a different device”, etc.* are **not objective reasons** to skip powering related diagnostics
- *“I know that my SD card is good”, “it worked yesterday”, “it works with a different device”, etc.* are **not objective reasons** to skip storage related diagnostics
- undervoltage can cause symptoms related to SD card problems such as filesystem corruptions and data loss, so powering has to be checked first

Powering notes

- Most boards, even ones fitted with PMIC (power management integrated circuit) do not have any measures to react to undervoltage that could prevent instability
- It does not matter what voltage your power supply outputs, it matters what voltage will reach the onboard voltage regulators
- Peak power consumption of popular boards can vary from 0.9A at 5V (H3 based Orange Pi PC) to 1.7A at 5V (RK3288 based Tinkerboard), both without any attached peripherals like USB devices
- Due to the Ohm’s law voltage drop due to cable and connector resistance will be proportional to the electric current, so most of the time problems will be experienced at current spikes caused by CPU load or peripherals like spinning up HDDs

Power supply

- Cheap phone chargers may not provide the current listed on their label, especially for long time periods
- Some cheap phone chargers don't have proper feedback based stabilization, so output voltage may change depending on load
- Power supplies will degrade over time (especially when working 24/7)
- Some problems like degraded output filtering capacitors cannot be diagnosed even with a multimeter because of the non-linear voltage form

Cable

- The longer and thinner the cable - the higher its resistance - the greater the voltage drop will be under load
- Even thick looking cable can have thin wires inside, so do not trust the outside cable diameter

Connector

- MicroUSB connector is rated for the maximum current of 1.8A, but even this number cannot be guaranteed. Trying to pass larger current (even momentarily) may result in a voltage dropping below USB specifications
- Most of the boards can also be powered through GPIO pins. This can be used to bypass the microUSB connector and thus to improve stability

SD card notes

- A SD card is a complex storage device with an embedded controller that processes read, erase and write operations, wear leveling, error and corruption detection, but it does not provide any diagnostic protocols like S.M.A.R.T.
- SD cards will degrade over time and may fail in the end in different ways - become completely or partially read-only or cause a silent data corruption

SD card brand

- Based on current prices and performance tests done by Armbian users Samsung Evo, Samsung Evo Plus and Sandisk Ultra cards are recommended
- Other good alternatives may be added to this page in the future

SD card size and speed class

- SD card speed class and size does not influence the reliability directly, but larger size means larger amount of lifetime data written, even if you are using 10-20% of the cards space

Writing images to the SD card

- If you wrote an image to the card it does not mean that it was written successfully without any errors
- so always verify images after write using some tools like *balenaEtcher* which is currently the only popular and cross-platform tool that does mandatory verify on write (more lightweight alternatives may be added to this page in the future)
- “Check for bad blocks” function available in some tools is mostly useless when dealing with SD cards
- Note that *balenaEtcher* verifies only 1-2GB that are occupied by the initial unresized image, it does not verify the whole SD card

How to switch kernels?

Check [this](#) for more info.

How to troubleshoot?

Important: If you came here since you cannot get Armbian running on your board please keep in mind that in 95 percent of all cases it is either a faulty/fraud/counterfeit SD card or an insufficient power supply that is causing these sorts of *does not work* issues!

If you broke the system you can try to get in this way. You have to get to u-boot command prompt, using either a serial adapter or monitor and usb keyboard (USB support in u-boot currently not enabled on all H3 boards).

After switching power on or rebooting, when u-boot loads up, press some key on the keyboard (or send some key presses via terminal) to abort default boot sequence and get to the command prompt:

```
U-Boot SPL 2015.07-dirty (Oct 01 2015 - 15:05:21) ... Hit any key to stop autoboot: 0 sunxi#
```

Enter these commands, replacing root device path if necessary. Select setenv line with ttyS0 for serial, tty1 for keyboard+monitor (these are for booting with mainline kernel, check boot.cmd for your device for commands related to legacy kernel):

```
setenv bootargs init=/bin/bash root=/dev/mmcblk0p1 rootwait console=ttyS0,115200 # or setenv bootargs init=/bin/bash root=/dev/mmcblk0p1 rootwait c
```

System should eventually boot to bash shell:

```
root@(none):/#
```

Now you can try to fix your broken system.

How to unbrick the system? (outdated)

When something goes terribly wrong and you are not able to boot the system, this is the way to proceed. You need some Linux machine where you can mount the failed SD card. With this procedure you will reinstall the u-boot, kernel and hardware settings. In most cases this should be enough to unbrick the board. It is recommended to issue a filesystem check before mounting:

```
fsck /dev/sdX -f
```

Then mount the SD card and download those files (This example is only for Banana R1):

```
http://apt.armbian.com/pool/main/l/linux-trusty-root-current-lamobo-r1/linux-trusty-root-current-lamobo-r1_4.5_armhf.deb http://apt.armbian.com/pool/main/l/linux-trusty-root-current-lamobo-r1/linux-trusty-root-current-lamobo-r1_4.5_armhf.deb
```

This is just an example for: **Ubuntu Trusty, Lamobo R1, mainline kernel** (next).
Alter packages naming according to [this](#).

Mount SD card and extract all those deb files to it's mount point.

```
dpkg -x DEB_FILE /mnt
```

Go to /mnt/boot and link (or copy) **vmlinux-4.x.x-sunxi** kernel file to **zImage**.

If you upgrade from some very old build, you might need to update your boot script.
Example goes for Allwinner boards:

```
cd /mnt/boot wget https://raw.githubusercontent.com/armbian/build/master/config/bootscripts/boot-sunxi.cmd mv boot-sunxi.cmd boot.cmd mkimage -C no
```

Unmount SD card, move it to the board and power on.

How to build a wireless driver?

Install and recreate kernel headers scripts (optional)

```
armbian-config -> install kernel headers exit cd /usr/src/linux-headers-$(uname -r) make scripts
```

Go back to root directory and fetch sources (working example, use ARCH=arm64 on 64bit system)

```
cd git clone https://github.com/pvaret/rtl8192cu-fixes.git cd rtl8192cu-fixes make ARCH=arm
```

Load driver for test

```
insmod 8192cu.ko
```

Check dmesg and the last entry will be:

```
usbcore: registered new interface driver rtl8192cu
```

Plug the USB wireless adaptor and issue a command:

```
iwconfig wlan0
```

You should see this:

```
wlan0 unassociated Nickname:"<WIFI@REALTEK>" Mode:Auto Frequency=2.412 GHz Access Point: Not-Associated Sensitivity:0/0 Retry:off RTS thr:off Fragm
```

Check which wireless stations / routers are in range

```
iwlist wlan0 scan | grep ESSID
```

How to freeze your filesystem? (outdated)

In certain situations it is desirable to have a virtual read-only root filesystem. This prevents any changes from occurring on the root filesystem that may alter system behavior and it allows a simple reboot to restore a system to its clean state.

You need an ODROID XU4 or Allwinner A10, A20 or H3 board with legacy kernel where we added support for overlays. Works only on Ubuntu Xenial. Login as root and execute:

```
apt-get install overlayroot echo 'overlayroot="tmpfs"' >> /etc/overlayroot.conf reboot
```

After your system boots up it will always remain as is. If you want to make any permanent changes, you need to run:

```
overlayroot-chroot
```

Changes inside this will be preserved.

How to run Docker? (outdated)

Preinstallation requirements:

- Armbian 5.1 or newer with Kernel 3.10 or higher
- Debian Jessie (might work elsewhere with some modifications)
- root access

Execute this as root:

```
curl https://get.docker.com | sh
```

Test if Docker works correctly:

```
docker run -d -p 80:80 hypriot/rpi-busybox-httpd
```

... and point the browser of any device in the same network to `http://<IP OF YOUR DEVICE>/`

[More info in this forum topic](#)

5.6 How to set wireless access point?

There are two different HostAP daemons. One is **default** and the other one is for some **Realtek** wifi cards. Both have their own basic configurations and both are patched to gain maximum performances.

Sources: <https://github.com/igorpecovnik/hostapd>

Default binary and configuration location:

```
/usr/sbin/hostapd /etc/hostapd.conf
```

Realtek binary and configuration location:

```
/usr/sbin/hostapd-rt /etc/hostapd.conf-rt
```

Since its hard to define when to use which you always try both combinations in case of troubles. To start AP automatically:

1. Edit /etc/init.d/hostapd and add/alter location of your conf file **DAEMON_CONF=/etc/hostapd.conf** and binary **DAEMON_SBIN=/usr/sbin/hostapd**
2. Copy **/etc/network/interfaces.hostapd** to **/etc/network/interfaces**
3. Reboot
4. Predefined network name: "BOARD NAME" password: 12345678
5. To change parameters, edit /etc/hostapd.conf BTW: You can get WPAPSK the long blob from wpa_passphrase YOURNAME YOURPASS

5.7 How to connect IR remote?

Required conditions:

- IR hardware
- loaded driver

Get your [remote configuration](#) (lircd.conf) or [learn](#). You are going to need the list of all possible commands which you can map to your IR remote keys:

```
irrecord --list-namespace
```

To start with learning process you need to delete old config:

```
rm /etc/lircd.conf
```

Then start the process with:

```
irrecord --driver=default --device=/dev/lirc0 /etc/lircd.conf
```

And finally start your service when done with learning:

```
service lirc start
```

Test your remote:

```
irw /dev/lircd
```

5.8 How to customize keyboard, time zone?

5.8.1 Attention:

The preferred method to change most of this stuff is by using the interactive *armbian-config* tool which is shipped with all Armbian images.

Keyboard:

```
dpkg-reconfigure keyboard-configuration
```

System language:

```
# Debian --> https://wiki.debian.org/ChangeLanguage dpkg-reconfigure locales # Ubuntu --> https://help.ubuntu.com/community/Locale update-locale LA
```

Console font, codepage:

```
dpkg-reconfigure console-setup
```

Time zone:

```
dpkg-reconfigure tzdata
```

Screen resolution on other boards:

```
nano /boot/boot.cmd # example: # change example from # disp.screen0_output_mode=1920x1080p60 # to # disp.screen0_output_mode=1280x720p60 mkimage -C
```

Screen resolution within Xorg [Thx @maxlinux2000](#)

```
Find matching HDMI output: xrandr --listmonitors Calculate VESA CVT mode lines (example for 1440x900) cvt 1440 900 Sample output: 1440x900 59.89 Hz
```

How to alter CPU frequency?

Some boards allow to adjust CPU speed

```
nano /etc/default/cpufrequtils
```

Alter **min_speed** or **max_speed** variable.

```
service cpufrequtils restart
```

How to downgrade a package via apt?

This is useful when you need to fall back to previous kernel version.

```
apt install linux-image-sun8i=5.13
```

This example is for H3 legacy kernel. Check [this page](#) for others.

How to toggle boot output?

Edit and change [boot parameters](#) in `/boot/boot.cmd` (not recommended) or variables in `/boot/armbianEnv.txt`:

```
- console=both + console=serial
```

Recompile boot.cmd to boot.scr if it was changed:

```
mkimage -C none -A arm -T script -d /boot/boot.cmd /boot/boot.scr
```

Reboot.

Serial console on imx6 boards are ttymxc0 (Hummingboard, Cubox-i) or ttymxc1 (Udoo).

How to toggle verbose boot?

Using Armbian 5.05 to 5.20 you would need to touch/rm `/boot/.force-verbose` to increase boot verbosity. With more recent Armbian builds you would have to alter the `verbosity=` line in `/boot/armbianEnv.txt` (defaults to 1 which means less verbose, maximum value is 7).

How to provide boot logs for inspection?

When your SBC behaves strange first step is to check power supply and integrity of boot media (`armbianmonitor -c "$HOME"`). Then look into your kernel logs. We made a tool that grabs info and pastes it to an online pasteboard service. Please increase boot verbosity as shown above (`verbosity=7`), reboot and then run

```
sudo armbianmonitor -u
```

Copy and past URL of your log to the forum, mail, ...

How to change network configuration?

To get Wi-Fi working simply use `nmtui`, a simple console based UI for network-manager (an example how to set up an AP with network-manager can be found [here](#)). To deal with different Ethernet/Wi-Fi combinations there are six predefined configurations available, you can find them in those files:

```
/etc/network/interfaces.bonding /etc/network/interfaces.default /etc/network/interfaces.hostapd /etc/network/interfaces.network-manager /etc/network
```

By default **/etc/network/interfaces** is a copy of **/etc/network/interfaces.default**

1. BONDING: your network adapters are bonded in fail safe / “notebook” way.
2. DEFAULT: your network adapters are connected classical way.
3. HOSTAPD: your network adapters are bridged together and bridge is connected to the network. This allows you to have your AP connected directly to your router.
4. All interfaces are handled by network-manager (`nmtui` / `nmcli` or using the GUI)
5. Router configuration for Lamobo R1 / Banana R1.
6. Switch configuration for Lamobo R1 / Banana R1.

You can switch configuration with copying.

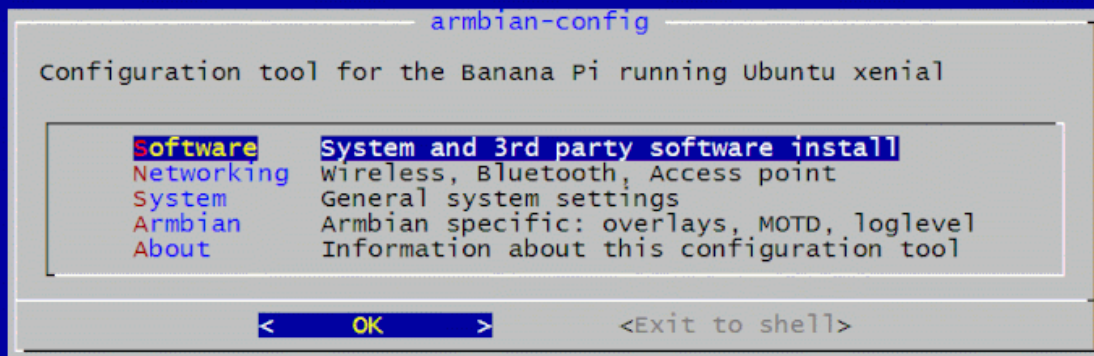
```
cd /etc/network cp interfaces.x interfaces
```

(x = default,hostapd,bonding,r1)

Then check / alter your interfaces:

```
nano /etc/network/interfaces
```

5.9 Armbian configuration utility



Is a base utility for configuring your board, divided into four main sections:

- **S**ystem - system and security settings,
- **N**etwork - wired, wireless, Bluetooth, access point,
- **P**ersonal - timezone, language, hostname,
- **S**oftware - system and 3rd party software install.

The tool needs root privileges to work and can be launched by entering `sudo armbian-config` at the terminal prompt or by clicking to the armbian-config menu item on desktop images.

5.9.1 System

- **Install** - installs to SATA, eMMC, NAND or USB. It gives you an option to install the system to more resilient and faster internal or external media. You can also change filesystem type to ext2,3,4 or BTRFS (if supported),
- **Freeze** - freeze or unfreeze kernel and board support packages, to avoid upgrading,
- **Nightly** - switch between nightly automated beta and stable builds,
- **Bootenv** - edit boot environment and alter kernel boot parameters,
- **Hardware** - toggle board low level functions: UART, I2C, SPI, ...
- **Switch** - switch to/between alternative kernels: legacy, current, dev
- **SSH** - reconfigure SSH daemon. Permit root login, toggle ssh key and mobile phone authentication,
- **Firmware** - execute apt update and upgrade to update your system,
- **Zshell** - toggle stock BASH and ZSH with [Oh My ZSH](#) and [tmux](#)
- **Enable** - toggle desktop on and off (on desktop images)
- **Lightdm** - change login managers from none to lightdm (on desktop images)
- **RDP** - toggle remote desktop from Windows (on desktop images)
- **Overlayroot** - toggle overlayroot (Ubuntu images)
- **Minimal** - install minimal Armbian XFCE powered desktop,
- **Default** - install Armbian XFCE powered desktop with web browser and extras.

5.9.2 Network

- **IP** - choose to select dynamic or edit static IP address,
- **Hotspot** - create or manage wireless access point. If your wireless adapter is recognized by a kernel, then armbian-config utility auto selects best mode on the selected device. It can detect 802.11n, 802.11a and 802.11ac. It also knows how to handle some special Realtek adapters,
- **IPV6** - toggle IPV6 for apt and system,
- **Iperf3** - toggle network throughput tests daemon,
- **LTE** - 3G/4G LTE modem management
- **WiFi** - manage wireless networking. Connect with Wifi network. You can create multiple wireless connections at the same time. They are managed by Network Manager,
- **BT install** - pair Bluetooth devices without PIN code,
- **Advanced** - edit network config manually,
- **Forget** - disconnects and clear all wireless connections.

5.9.3 Personal settings

- **Timezone** - change timezone,
- **Locales** - reconfigure language and character set,
- **Keyboard** - change console keyboard settings,
- **Hostname** - change hostname,
- **Mirror** - change to backup APT repository mirror in case of troubles,
- **Welcome** - toggle welcome screen items.

5.9.4 Software

Software installation menu provides automated install of the following packages.

- **softy**
 - [TV headend](#) (*IPTV server*)
 - [Syncthing](#) (*personal cloud*)
 - [SoftEther VPN server](#) (*VPN server*)
 - [Plex](#) (*Plex media server*)
 - [Radarr](#) (*Movie downloading server*)
 - [Sonarr](#) (*TV shows downloading server*)
 - [Transmission](#) (*torrent server*)
 - [ISPConfig](#) (*WEB & MAIL server*)
 - [NCP](#) (*Nextcloud personal cloud*)
 - [Openmediavault NAS](#) (*NAS server*)
 - [PI hole](#) (*ad blocker*)
 - [UrBackup](#) (*client/server backup system*)
 - [Docker](#) (*Docker CE engine*)
 - [Mayan EDMS](#) (*Document management system within Docker*)
 - [MiniDLNA](#) (*media sharing*)
- **Monitor** = simple CLI monitoring
- **Diagnostics** = create a summary of logs and upload them to paste.bin
- **Toggle** kernel headers, RDP service, Thunderbird and Libreoffice (desktop builds)

5.9.5 Sources

<https://github.com/armbian/config>

5.10 Device Tree overlays

Most in-circuit and GPIO based interfaces (SPI, I2C, I2S, UART, ...) don't have a mechanism for detecting and identifying devices connected to the bus, so Linux kernel has to be told explicitly about the device and its configuration details.

While Device Tree is a way of describing hardware configuration to the kernel, Device Tree overlays are a way for modifying the DT in order to provide the kernel and kernel drivers with details about external devices or to activate interfaces disabled by default.

Note: from the Linux kernel maintainer perspective all unused in-circuit type interfaces that use GPIO pins should be disabled by default and all pins on pin headers or soldering pads will be configured as standard GPIOs.

Note: from the Linux kernel maintainer perspective all dedicated interfaces like USB, Ethernet or analog audio that are wired to soldering pads or a pin headers instead of specialized sockets (like USB socket, Ethernet socket or 3.5mm jack) will be left disabled by default.

Armbian specific notes

- DT overlays are a Work-in-Progress (WIP) feature, present **only in fresh images starting with 5.30**, nightly and user made images
- For older images (even upgraded to 5.30 or later) manual update of the u-boot and the boot script is required
- Currently implemented only for sunxi based devices that use mainline u-boot and kernel

Please note that different SoCs will have different sets of available overlays.

Quick start

1. Check the `README.<soc-id>-overlays` in `/boot/dtb/overlay/` (32-bit SoCs) or `/boot/dtb/allwinner/overlay/` (64-bit SoCs) for a list of provided overlays, their required and optional parameters
2. Add names of overlays you want to activate to `overlays=` line in `/boot/armbianEnv.txt`, separated with spaces
3. Add required parameters with their values to `/boot/armbianEnv.txt`, one per line
4. Add optional parameters with their values to `/boot/armbianEnv.txt` if you want to change the default value, one per line
5. If you didn't find the required overlay or want to change one of provided overlays, refer to "Using custom overlays" section
6. Reboot

Using custom overlays

1. Check [here](#) for some example overlays
2. Copy or create your overlay file (with `.dts` extension) on the device
3. Change I2C or SPI bus number, GPIO and pinctrl pins, `compatible` string to match your SoC if necessary
4. Compile and activate the overlay by running `armbian-add-overlay <overlay_file.dts>` as root, i.e.

```
sudo armbian-add-overlay sht15.dts
```

5. Reboot

armbianEnv.txt entries reference

- `overlay_prefix` - prefix for the DT and overlay file names, set at OS image creation time
- `overlays` - list of overlays to activate from kernel directory
- `user_overlays` - list of overlays to activate from `/boot/overlay-user/` directory
- `param_*` - overlay parameters

Kernel provided vs user provided overlays

Overlays can be loaded from 2 locations:

- `/boot/dtb/overlay/` (`/boot/dtb/allwinner/overlay/` for 64-bit SoCs) - kernel provided overlays
- `/boot/overlay-user/` - user provided overlays

Main differences between these locations:

- Kernel provided overlays are updated with the kernel packages, any changes to this directory (including new files) will be lost on kernel upgrade
- Kernel provided directory may contain overlays for different SoCs, so overlay file name pattern will be `<prefix>-<name>`, for example `sun8i-h3-i2c0.dtbo`, where `sun8i-h3` is the prefix and `i2c0` is the name
- Kernel provided overlays are activated by the overlay name (i.e. `i2c0`), and the prefix is set at OS image creation time
- User provided overlays directory is empty by default and is meant for storing and using user created overlays that are not present in the kernel packages or modified stock overlays
- User provided overlays are activated by the file name (excluding the extension), i.e. for file `adafruit13m.dtbo` overlay name would be `adafruit13m`

Activation

DT overlays are activated by editing u-boot environment file `/boot/armbianEnv.txt`

- Kernel provided overlays are activated by adding a name to the `overlays` variable
- User provided overlays are activated by adding a name to the `user_overlays` variable
- No more than one `overlays` line and one `user_overlays` line can be present in the environment file
- Multiple names should be separated by space
- If activated overlays have parameters marked as “Required”, those parameters have to be set to proper values
- Reboot is required for any changes to take effect

Overlay parameters

Some overlays have additional parameters that can be set.

Parameters marked as “Required” have to be set if overlay with these parameters is activated, other parameters are not mandatory if default value is suitable.

Parameters are set by adding their name and value to `/boot/armbianEnv.txt`, each parameter should be added on a new line.

Please refer to `README.<SoC_prefix>-overlays` files in `/boot/dtb/overlay/` (`/boot/dtb/allwinner/overlay/` for 64-bit SoCs) directory for supported parameters, i.e. `README.sun8i-h3-overlays` for H3 based boards.

Parameters of type `pin` require special format:

- Value consists of a letter `P`, a letter that signifies the pin bank and a number of the pin in the bank
- Letters should be upper case
- Numbers should not contain leading zeros
- Examples: good - `PA9`, `PG12`; bad - `pa2`, `PG08`

Overlay bus selection

SoCs may contain multiple bus controllers of the same type, i.e. Allwinner H3 contains 2 SPI controllers and Allwinner A20 contains 4 SPI controllers.

Please refer to your board documentation and schematic to determine what pins are wired to the pin headers and thus what bus number should be used in each case.

Overlay pinmux conflicts

Some controllers may share the SoC pins in some configurations. For example on Allwinner H3 UART 3 and SPI 1 use the same pins - `PA13`, `PA14`, `PA15`, `PA16`. In this case activating both UART 3 and SPI 1 would result in a conflict, and on practice only one interface (either SPI or UART) will be accessible on these pins.

Please check the SoC specific README, board schematic, SoC datasheet or other documentation if you are unsure about possible conflicts if activating multiple overlays for the controllers that use shared (muxed) pins.

Overlay device endpoint conflicts

Overlays for devices that use addresses or similar mechanisms (i.e. SPI chip selects) can't be activated simultaneously if addresses (chip selects) are identical.

For example A20 SPI controller 1 has only one hardware chip select, so `spi-spidev` and `spi-jedec-nor` overlays cannot be activated both if they would use the same bus number and chip select.

Overlay compatibility

Device Tree overlays for different platforms and SoCs are not directly compatible. This, for example, means that overlays for H3 may need some changes to work on A20, and that Raspberry Pi overlays will need adjustments in order to be used on Allwinner based boards.

Rework may include changing labels, references (phandles) and pinconf bindings.

Notes regarding SPI and I2S overlays

Activating a device on SPI or I2S bus may require more than one overlay. In case a bus overlay like `spi0` or `i2s0` exist for the target SoC they need to be activated in addition to a slave device overlay (provided or custom/user-made). Please note that these overlays (`spi0`, `i2s0`) do not enable any slave devices (like spidev or I2S codec).

Debugging

As overlays and overlay parameters are applied by the u-boot, it is impossible to get any debugging information (such as error messages) from the OS.

Serial console on UART 0 is **required** to debug DT overlay related problems.

Example `/boot/armbianEnv.txt` contents:

```
verbosity=1 console=serial overlay_prefix=sun8i-h3 rootdev=UUID=bd0ded76-1188-4b52-a20a-64f326c1f193 rootfstype=ext4 overlays=w1-gpio uart1 i2c0 sp
```

Example of serial console log when using several overlays:

```
## Executing script at 43100000 U-boot loaded from SD Boot script loaded from mmc 265 bytes read in 182 ms (1000 Bytes/s) 5074230 bytes read in 532
```

5.11 Migration from Bananian to Armbian

While technically possible to do an **in-place** upgrade/crossgrade from latest Bananian release (or similar SBC distros) to Armbian currently there exists no tool helping with this and most probably will never exist. At the bottom is explained where to find resources that help with a manual in-place upgrade but we start with outlining the problems and our recommendations:

5.11.1 The challenges:

SD cards wear out after a certain amount of data being written to

Only reasonable base for an migration to Armbian would be an updated Bananian installation (Bananian 16.04, already Debian Jessie based, Nico's 4.4 kernel). In case Bananian users are still on version 15.04 or earlier they need to upgrade to a more recent Bananian version anyway to move Bananian's base to Jessie. Such `apt-get dist-upgrade` tasks come with heavy write activity. Especially when planning a dist-upgrade to Stretch later this amount of random write activity on older/smaller SD cards might kill them. If the SD card is not brand new it's highly recommended to create a clone/backup first prior to every upgrade step. If the SD card is really old please be prepared that it might not survive an `apt-get dist-upgrade`.

All hardware will die eventually

A lot of Bananian installations today have been running 24/7 for 3 years or even longer. While these little SBC are well suited for light-weight server tasks, the hardware can't exactly be called 'server grade'. Please keep this in mind if you're about to spend some time on a manual migration attempt that once you're done maybe your hardware will stop working few weeks/months later if it already runs +24 months.

Hardware up to the task?

The vast majority of [boards Bananian runs on](#) is based on Allwinner's dual core A20 SoC which was a nice improvement over the first single-core Raspberry Pis few years ago but is pretty slow by today's standards. An awful lot of users (us Armbians **all** included) were excited by A20's 'native SATA' capabilities few years ago just to realize after purchase when using SATA attached storage that it's awfully slow and most

probably the slowest ‘native’ SATA implementation existing (please wake up if in doubt and educate yourself [here](#), [here](#) or [here](#). Important: combining Allwinner’s crappy SATA implementation with port multipliers [is always wrong](#)).

At the time of this writing (Oct 2017) Armbian supports +25 other ARM boards that show between 2 and 6 times better CPU performance than A20 devices. +20 boards we support show better network performance (A20 Gigabit Ethernet is not fully capable of 940 Mbits/sec in both directions). +15 boards support 2GB DRAM (a few even more just recently). And if you don’t need Gigabit Ethernet you can get a new and fully supported board still better suited for light-weight server tasks than any Banana Pi for as less as \$11 shipping included (check [this overview](#) please).

While this diversity of ARM species might be confusing the good news is: When Armbian is running on them they all behave the same.

5.11.2 Alternatives to an in-place migration:

Continue on same hardware but prevent SD card hassles

Especially if you run since years off the same SD card please be prepared that it might not survive an `apt-get dist-upgrade` and similar upgrade/crossgrade tasks. It’s **strongly** recommended to clone/backup your card prior to every necessary upgrade step. Since this is time consuming and just a measure to prepare for what will happen in the future anyway (your SD card failing eventually – if you’re lucky immediately, if you’re out of luck it will corrupt a lot of data dying slowly) a great idea is to buy a new one **now**. Please see [our community’s collection of SD card performance tests](#) and especially the 3 links at the top dealing with reliability concerns.

Once you bought a new, fast and hopefully reliable SD card, you should [test it according to our documentation](#), then burn a fresh Armbian image on it and manually transfer data and settings from your Bananian installation. This way you preserve your current settings/data on the old Bananian SD card saving you also a lot of time/efforts to clone/backup stuff.

Important note: if you’re interested in NAS use cases you could also choose an OMV image from official [download location](#) (all the ARM board images are now based on Armbian, funnily even the ones for Raspberry Pi)

Replacing the hardware

If your Bananian installation has been running for years, you better think about evaluating new hardware now. As explained above, A20's SATA implementation is **awfully slow** compared to good SATA implementations (Espressobin, Clearfog, Helios4) or even recent USB3 solutions, also Banana Pis can not saturate Gigabit Ethernet. It's almost 2018 now and we can choose from a variety of energy efficient boards more suited for the job.

My personal strategy was turning the various A20 servers into backup devices now receiving btrfs snapshots from better suited ARM servers in the meantime. New installation on new board, carefully migrating settings from Bananas, Cubietrucks or Lime boards to new server, testing, testing, testing, new installation on A20 device, setting up btrfs send|receive, testing, testing, testing, done.

5.11.3 In-place migration tips:

Since there is no easy migration tool you can only rely on contents collected below <https://github.com/armbian/build/issues/648> – if you read carefully through we had some hope experienced Bananian users would be volunteering in providing an in-place upgrade tool from Bananian to Armbian but unfortunately to no avail. So 6 months after the problem came to our attention we're now providing this document to help those affected taking the right decisions. Still no need to hurry, Bananian receives bug and security fixes for another 6 months so take your time and evaluate carefully which way to choose.

Trivia: Anyone understanding german **will** enjoy Nico's refreshing [Rise and Fall of Bananian Linux](#) talk.

6. Hardware Notes

6.1 Enable Hardware Features

Some boards require some manual configuration to turn on/off certain features

In some cases, the procedure is “less than obvious”, so we document some basic examples here.

6.2 Generic howto for Allwinner devices

6.2.1 Legacy or current kernel ?

Many Armbian images come in two flavours : *Legacy* (using an older kernel version) and *current* (up-to-date LTS kernel). Depending on kernel version, the procedure to enable/disable features is not the same.

- *Legacy* kernel (4.19.x): DT (Device Tree) overlays
- *Current* kernel (5.4.x) : DT (Device Tree) overlays

Note: Support for older kernel versions (like 3.4.x or 3.10.x) has been dropped.

6.2.2 How to reconfigure video output?

This affect *current* kernel only.

U-Boot supports HDMI and LCD output on Allwinner sunxi SoCs, LCD output requires the `CONFIG_VIDEO_LCD_MODE` Kconfig value to be set.

The sunxi U-Boot driver supports the following video-mode options:

- `monitor=[none|dvi|hdmilcd|vga|composite-*]` - Select the video output to use
- `none`: Disable video output.
- `dvi/hdmi`: Selects output over the hdmi connector with dvi resp. hdmi output format, if edid is used the format is automatically selected.
- `lcd`: Selects video output to a LCD screen.
- `vga`: Selects video output over the VGA connector.
- `composite-pal/composite-ntsc/composite-pal-m/composite-pal-nc`: Selects composite video output, note the specified resolution is ignored with composite video output.
- Defaults to `monitor=dvi`.
- `hpd=[0|1]` - Enable use of the HDMI HotPlug Detect feature 0: Disabled. Configure DVI/HDMI output even if no cable is detected 1: Enabled. Fallback to the LCD / VGA / none in that order (if available) Defaults to `hpd=1`.
- `hpd_delay=<int>` - How long to wait for the HDMI HPD signal in milliseconds When the monitor and the board power up at the same time, it may take some time for the monitor to assert the HPD signal. This configures how long to wait for the HPD signal before assuming no cable is connected. Defaults to `hpd_delay=500`.
- `edid=[0|1]` - Enable use of DDC + EDID to get monitor info 0: Disabled. 1: Enabled. If valid EDID info was read from the monitor the EDID info will overrides the xres, yres and refresh from the video-mode env. variable. Defaults to `edid=1`.
- `overscan_x/overscan_y=<int>` - Set x/y overscan value This configures a black border on the left and right resp. top and bottom to deal with overscanning displays. Defaults to `overscan_x=32` and `overscan_y=20` for composite monitors, 0 for other monitors.

For example to always use the HDMI connector, even if no cable is inserted, using edid info when available and otherwise initializing it at 1024x768@60Hz, use:

```
setenv video-mode sunxi:1024x768-24@60,monitor=dvi,hpd=0,edid=0.
```

Parameters regarding video must be saved into U-Boot environment file since they must be read before reading boot script. You can do this by adding `saveenv` command at the end of boot script (boot.cmd). Remember to recompile boot.cmd to boot.scr and note that changes will come into action after second boot.

6.2.3 What flavour am I using ?

Best way to know is by checking your kernel version :


```
root@bananapipro:~# uname -a Linux bananapipro 4.5.2-sunxi #11 SMP Thu Apr 28 21:53:25 CEST 2016 armv7l GNU/Linux
```

In this example the kernel version is 4.5.2 so you can use DT to tweak some settings. If you get a kernel version 3.X then you'll be certainly using FEX like on an Orange Pi Plus 2E :

```
root@orangeplus2e:~# uname -a Linux orangeplus2e 3.4.112-sun8i #10 SMP PREEMPT Wed Jun 1 19:43:08 CEST 2016 armv7l GNU/Linux
```

6.2.4 FEX (outdated/unsupported, informational only)

Which file should I edit

Armbian embed a lot of BIN files, but a symlink point to the one in use :

```
root@orangeplus2e:~# ls -la /boot/script.bin lrwxrwxrwx 1 root root 22 Jun 1 20:30 /boot/script.bin -> bin/orangeplus2e.bin
```

Updating a FEX

You may need to use `sudo` with all the following commands.

The whole process won't overwrite any of your files. If you're paranoid, you can make a proper backup of your BIN file :

```
cp /boot/script.bin /boot/bin/script.bin.backup
```

Then you can decompile your BIN into a FEX :

```
bin2fex /boot/script.bin /tmp/custom.fex
```

Finally you can edit your FEX file with your favorite text editor and compile it back to a BIN :

```
fex2bin /tmp/custom.fex /boot/bin/custom.bin
```

The last step is to change the symlink to use your custom BIN :

```
ln -sf /boot/bin/custom.bin /boot/script.bin
```

6.3 H3 based Orange Pi, legacy kernel

6.3.1 Enable serial /dev/ttyS3 on pins 8 and 10 of the 40 pin header

Update the FEX configuration (which is compiled into a .bin) located at /boot/script.bin

Decompile .bin to .fex

```
cd /boot bin2fex script.bin > custom.fex rm script.bin # only removes symbolic link
```

Edit .fex file

```
[uart3] uart_used = 1 ; Change from 0 to 1 uart_port = 3 uart_type = 2 ; In this case we have a 2 pin UART uart_tx = port:PA13<3><1><default><default>
```

Compile .fex to .bin

```
fex2bin custom.fex > script.bin
```

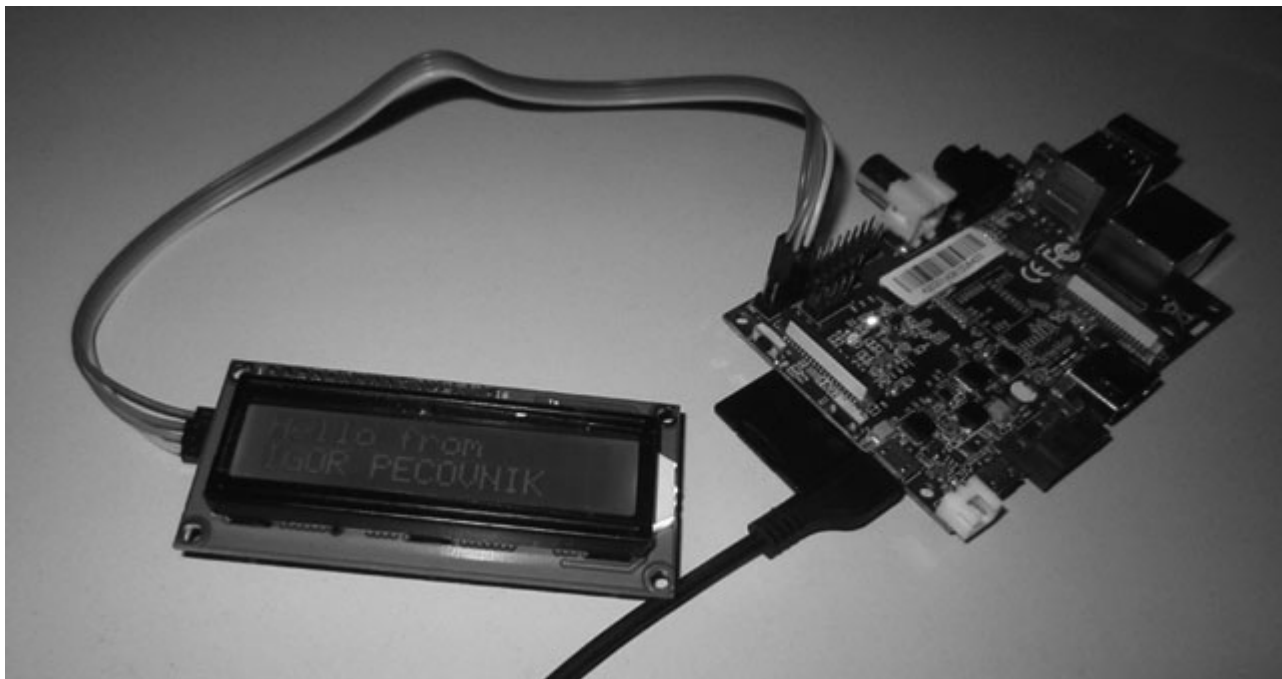
Reboot

Notice that /dev/ttyS3 appears. That is your new UART device.

6.3.2 Connect your LCD display

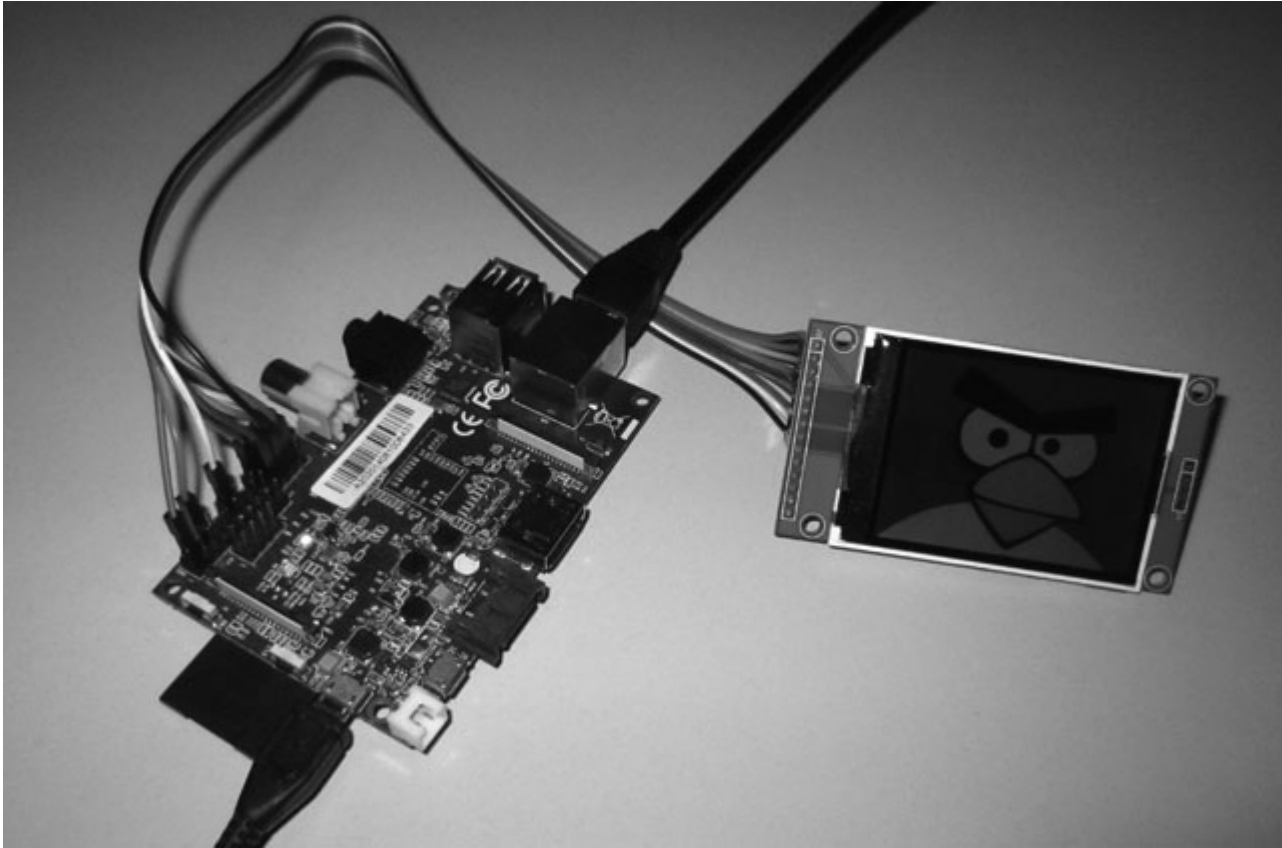
I tried three different display connection types: I2C, (4bit) parallel and SPI. All of them are working perfectly with my image. I didn't took a picture of the third one. It's a standard Hitachi HD44780 based 20×4 LCD, wired and tested [according to Wiring\(B\)PI example](#).

I2C



I am using [this code](#) for mainline kernel and with [changed line](#): /dev/i2c-%u = /dev/i2c-2 for Legacy kernel.

SPI



- I am using [2.4" 240×320 SPI TFT LCD Serial Port Module+5/3.3V Pbc Adapter Micro SD ILI9341](#)
- Wire according to [this map](#).
- You have to use Armbian 1.5 or newer. Currently working only under Legacy kernel.
- Add this to your /etc/modules: `fbtft_device name=adafruit22a rotate=90 speed=48000000 fps=50 gpios=reset:25,led:19,dc:24`
- Reboot
- Test – display some picture on the screen: `fbi -d /dev/fb2 -T 1 -noverbose -a yourimage.jpg`
- [Troubleshooting and settings for other displays LVDS](#)

Currently working only under Legacy kernel.

Image has pre-loaded settings for two LVDS display.

To enable 7 inch.

```
ln -sf /boot/bin/bananapilcd7.bin /boot/script.bin
```

To enable 5 inch.

```
ln -sf /boot/bin/bananapilcd5.bin /boot/script.bin
```

If you need touch screen support, add this module to your `/etc/modules`

```
ft5x_ts
```

6.4 Allwinner A10 & A20 boards

6.4.1 Overview

Both kernels are stable and production ready, but you should use them for different purposes since their basic support differ:

- legacy: video acceleration, NAND support, connecting displays
- mainline: headless server, office desktop operations (not multimedia oriented)

6.4.2 Legacy

System images with legacy kernel

Please note that upstream support for kernel 3.4.x has ended in 2017 so this kernel will not receive security updates in the future.

- Kernel [3.4.x](#) with large hardware support, headers and some firmware included
- Enabled audio devices: analog, 8 channel HDMI, spdif and I2S (if wired and enabled in HW configuration)
- Bluetooth ready (working with supported external keys)
- [Enabled overlayfs](#)
- [I2C](#) ready and tested with small 16×2 LCD. Basic i2c tools included.
- SPI ready and tested with ILI9341 based 2.4" TFT LCD display.
- [Drivers for small TFT LCD](#) display modules.
- [Clustering / stacking](#)
- Onboard LED attached to SD card activity (script.bin)

Bugs or limitation

- NAND install sometime fails. Workaround: install [Lubuntu to NAND](#) with [Phoenix tools](#) and run install again.
- Shutdown results into reboot under certain conditions.

6.4.3 Mainline

System images with mainline kernel

- [Mainline](#) with large hardware support, headers and some firmware included
- [Docker ready](#)
- Enabled audio devices: analog, SPDIF (if available) & USB
- [USB / UAS](#) - more efficient disk access over USB (A20 and H3)
- [CAN bus](#) - Controller Area Network
- [USB OTG connector](#) - OTG or host mode
- Bluetooth ready (working with supported external keys)
- [I2C](#) ready and tested with small 16×2 LCD. Basic i2c tools included.
- Onboard LED attached to SD card activity (not enabled on all boards yet)

Bugs or limitation

- No HW acceleration for desktop and video decoding
- NAND is not supported yet
- Screen output from kernel is set to HDMI by default. Boot loader can detect and switch, kernel not.
- HDMI audio is not supported yet
- SATA port multiplier support is disabled by default, can be enabled by adding kernel parameter

```
ahci_sunxi.enable_pmp=1
```

6.4.4 Desktop

YOUTUBE

- HW accelerated video playback (legacy kernel only)
- MALI Open GLES (legacy kernel only)
- Pre-installed: Firefox, LibreOffice Writer, Thunderbird
- Lightweight XFCE desktop
- Autologin, when normal user is created - no login manager (/etc/default/nodm)

6.4.5 Notes

Setting non-standard monitor settings for A10, A20 and A31 based boards in u-boot

Following commands (example) needs to be executed in u-boot command prompt:

```
setenv video-mode sunxi:1024x768-24@60,monitor=dvi,hpd=0,edid=0,overscan_x=1,overscan_y=2 saveenv
```

Since environment is reset after flashing u-boot, you need to do this after every u-boot upgrade or put this to u-boot script

6.4.6 Resources

[Armbian packages repository](#)

6.5 Allwinner H3 boards

Overview

The H3 SoC from Allwinner is meant for OTT boxes and therefore its reference design is *not* accompanied by a separate PMIC (power management IC) unlike *A series* Allwinner SoCs (like A10, A20, A64, ...). No PMIC means also that there is no battery charging/monitoring implemented so H3 is not that much suited for mobile devices. On the other hand some pretty cheap H3 boards were released that can be driven with rather low consumption and therefore combining H3 devices with a battery became a real use case with boards like [Orange Pi One/Lite](#), NanoPi [NEO](#) and [Neo AIR](#).

As usual [SoC](#) and [device information](#) can be found in Linux-sunxi wiki. Same applies to status of [mainlining kernel efforts](#). Adding to the usual SoC feature set (I2C, SPI, PWM, UART, SDIO, GPIO and so on) H3 has one USB OTG port, 3 real USB host ports (not exposed on all devices), Fast- and Gigabit Ethernet capabilities (board specific), a Mali400MP2 GPU and Allwinner's video encoding/decoding engine.

When CPU or GPU cores are fully utilized H3 tends to overheat over time like any other popular ARM SoC released within the last 2-3 years. With Armbian we provide sane dvfs (dynamic voltage frequency scaling) settings that help a lot with throttling. In case you plan to operate your H3 device constantly under high load please check Armbian forums first since boards behave differently (related to voltage regulation and PCB size and design – some use copper layers to spread the heat away from the SoC). Also consider applying a heatsink to the SoC (a fan should not be necessary unless you want to do number crunching on your board and then you obviously chose the wrong device).

You find some [differentiation criteria regarding supported H3 devices as well as an overview/history of H3 software support in our forums](#) or use Jean-Luc's [nice comparison table](#) (both slightly outdated since more H3 devices have been released in the meantime).

Kernel support

Almost all features of the H3 SoC are supported on Armbian's *current* branch. Please refer to the [Linux sunxi support sheet](#).

Default settings

- CPU frequency settings are 240-912 MHz on NanoPi NEO, 240-1200 MHz on BPi M2+, NanoPi M1 and Beelink X2, 480-1200 MHz on OPi One/Lite and 480-1296 MHz on the other boards (cpufreq governor is *interactive* therefore the boards only increase CPU speed and consumption when needed). The differences are due to different voltage regulators and heat dissipation behaviour.
- Armbian unlike older/other H3 OS images uses the green led as 'power on' indicator (blinking means 'ready to login' or 'shutting down'), the red led (blue on NanoPis) can be used for your own purpose.

Tips and tricks (general)

- An insufficient power supply **is the root cause of many weird symptoms/problems**. Never trust in ratings written on the PSU since they might be wrong, the PSU might be old/dying and cable/contact resistance adds to problems. In other words: Before you blame Armbian for strange behaviour please try at least one second power supply (this applies to both PSU and cable between PSU and board if this is separate – especially USB cables really suck due to high resistance leading to severe voltage drops).
- In case you experience instabilities check your SD card using `armbianmonitor -c $HOME` and think about installing [RPi-Monitor for H3](#) to get an idea whether you suffer from overheating (`sudo armbianmonitor -r` will install everything needed).
- Especially for desktop images the speed of your SD card matters. If possible try to use our *nand-sata-install* script to move the rootfs away from SD card. The script also works with USB disks flawlessly ([some background information](#)).

Tips and tricks (H3 specific / lowering consumption) (outdated)

Recent research showed that H3 boards operated as wired IoT nodes need way less power compared to Raspberry Pis in the same situation (ethernet active). If you want to use your H3 device headless (server/IoT) and care about power consumption then there

exist a couple of tweaks to get your board being more energy efficient when using in the meantime unsupported 3.x kernel (no tests done yet with up-to-date *legacy/current* kernel):

- Disabling HDMI/GPU saves ~200mW.
- Allowing to temporarily only negotiate a Fast Ethernet connection on GbE capable boards saves +350 mW.
- Adjusting DRAM clockspeed is surprisingly another way to control consumption (on NanoPi NEO for example changing DRAM clockspeed between 132 MHz and 672 MHz results in consumption differences of 470mW).
- Limiting maximum CPU clockspeed will help with lowering maximum consumption (think about scripts running amok or something going terribly wrong), the same applies to limiting the count of active CPU cores.
- Choosing a board with Fast instead of Gigabit Ethernet or disabling GbE on the latter using `ethtool` or `ifconfig` saves at least 150 mW (board specific).

As an example: We chose default Armbian settings for NanoPi NEO to ensure this board is not able to exceed 2W consumption when running with no peripherals connected. This resulted in CPU and DRAM clockspeed of just 480/408 MHz while *booting* (the first ~20 seconds). In normal operation we limit maximum CPU clockspeed to 912 MHz to stay below the 2W consumption barrier even in worst case scenarios.

In case you want to have a few more percent maximum CPU performance you would need to set maximum cpufreq to 1200 MHz instead of 'just' 912 MHz maximum CPU clock using our new [h3consumption tool](#). Be warned: This will both heavily increase consumption and SoC temperature since exceeding 912 MHz CPU clockspeed means feeding the SoC with 1.3V instead of 1.1V core voltage (most smaller H3 devices use a voltage regulator only switching between two voltages to feed the SoC based on load).

Walking this route in the other direction is more interesting: In case you want to use an H3 device as IoT node you might want to limit both idle and maximum consumption. That should involve disabling stuff not needed (eg. HDMI/GPU since this saves 200mW) or limiting resource consumption: Lowering maximum clockspeeds for both CPU and DRAM or even disabling CPU cores (which helps *not* with idle consumption since ARM cores enter low-power modes if not needed but can help lowering maximum consumption requirements).

Since all of this stuff is based on recent research and being still WiP please consider reading through relevant threads in Armbian forums **and** join development/research/discussions: [SBC consumption/performance comparisons](#) and [Default settings for NanoPi NEO/Air](#).

6.6 Allwinner H5 and A64 boards

Overview

[See the generic Allwinner page](#)

Warning

Using the board without cooling in conjunction with the stable release of Armbian using kernel 4.19.y there is a risk of the board getting **permanently damaged** due to overheating. If you decide to try the board without cooling, you can use `sudo armbianmonitor -r` to keep an eye on the temperatures.

6.7 Allwinner H6

CPU frequency

[See the generic Allwinner page](#)

~~The H6 CPU frequency has been soft capped at 1,48 GHz to avoid thermal throttling too fast. This limit can be lifted by editing `/etc/default/cpufrequtils` and set `MAX_SPEED` to `1810000`.~~

With the release of Armbian 20.05 “Kagu” new thermal zones have been added making this limitation obsolete and therefore has been removed. All H6 boards now clocking at the highest possible value OOB.

Warning Adding proper cooling is highly recommended.

PCIe (un-)supported

Some H6 SoC based boards (like Pine H64 Model a, discontinued) are shipped with a PCIe slot. This slot **cannot** work out of the box as it has to be considered as broken by design. [Linux-Sunxi](#) writes about this:

Allwinner H6 has a quirky PCIe controller that doesn't map the PCIe address space properly (only 64k accessible at one time) to CPU, and accessing the PCIe config space, I/O space or memory space will need to be wrapped. As Linux doesn't wrap PCIe memory space access, it's not possible to do a proper PCIe controller driver for H6. The BSP kernel modifies the driver to wrap the access, so it's also not generic, and only devices with modified driver will work.

Icenowy is working on a wrapper to make PCIe work. Check [forums](#).

6.8 Cubox and Hummingboard boards

6.8.1 Legacy

System images with legacy kernel

- Kernel [3.14.x](#) with large hardware support, headers and some firmware included
- [Docker ready](#) - [what is Docker?](#)
- PCI-E operational (Hummingboard Pro, Gate & Edge)
- mSATA / m2 operational (Hummingboard Pro & Edge)
- Enabled audio devices: HDMI, spdif, analogue
- [Bluetooth ready](#) (working with Cubox-i/HB PRO on-board device or external key)
- [I2C](#) ready and tested with small 16×2 LCD. Basic i2c tools included.
- SPI ready and tested with ILI9341 based 2.4" TFT LCD display.
- [Drivers for small TFT LCD](#) display modules.
- [USB redirector](#) - for sharing USB over TCP/IP (disabled by default /etc/init.d/rc.usbsrvd)

Bugs or limitation

- Gigabit ethernet transfer rate is around 50% of its theoretical max rate (internal chip bus limitation)

6.8.2 Mainline

System images with mainline kernel

- [Mainline](#) with large hardware support, headers and some firmware included
- [Docker ready](#) - [what is Docker?](#)
- PCI-E operational (Hummingboard Pro, Gate & Edge)
- mSATA / m2 operational (Hummingboard Pro & Edge)
- Enabled audio devices
- Bluetooth ready (working with supported external keys)

Bugs or limitation

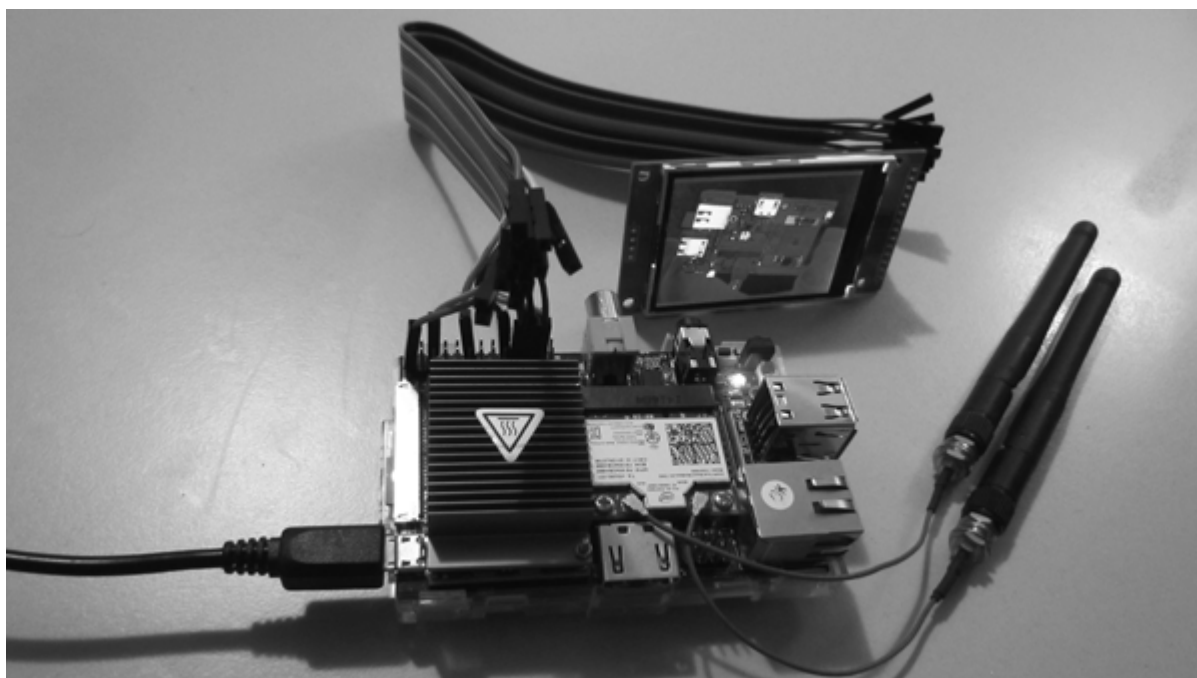
- Gigabit ethernet transfer rate is around 50% of its theoretical max rate (internal chip bus limitation)

6.8.3 Desktop

- Pre-installed: Firefox, LibreOffice Writer, Thunderbird
- Lightweight XFCE desktop
- Autologin, when normal user is created – no login manager (/etc/default/nodm)

6.8.4 Connect your LCD display

I tried two different display connection types: I2C and SPI. Both are working perfectly with my image 2.6 or higher.



- I am using [2.4" 240×320 SPI TFT LCD Serial Port Module+5/3.3V Pbc Adapter Micro SD ILI9341](#)
- Wire according to [this map](#).
- You have to use Armbian 1.5 or newer. Currently working only under Legacy kernel.
- Add this to your /etc/modules:

```
fbtft_device name=adafruit22a rotate=90 speed=48000000 fps=50 gpios=reset:67,led:72,dc:195 busnum=1
```

- Reboot
- Test – display some picture on the screen:

```
fbi -d /dev/fb2 -T 1 -noverbose -a yourimage.jpg
```
- [Troubleshooting and settings for other displays LVDS](#)

6.9 GPIO

[How to control HummingBoard GPIO from kernel space?](#)

6.10 Udoo Quad

- [Kernel 3.14.x](#) and [4.4.x](#) with some hardware support, headers and some firmware included
- [Docker ready](#) - [what is Docker?](#)
- Wireless adapter with DHCP ready but disabled (/etc/network/interfaces, WPA2: normal connect, bonding / notebook or AP mode). It can handle between 40-70Mbit/s.
- SATA operational
- Enabled analogue (VT1613) and HDMI audio device

6.11 Bugs

SATA & USB install not working on legacy kernel

6.12 Udoo Neo

- [Kernel 3.14.x](#) with some hardware support, headers and some firmware included
- Wireless adapter with DHCP ready but disabled

6.13 Helios4

6.13.1 Overview

All builds provide 100% hardware support for Helios4.

6.13.2 Build Version Status

Default

- U-Boot : 2018.11
- Linux Kernel : Mainline 4.14.y
- OS : Debian 9 Stretch

Next

- U-Boot : Mainline 2019.04
- Linux Kernel : Mainline 4.19.y
- OS : Debian 10 Buster or Ubuntu Bionic

Known Limitations

- SDcard High Speed timing have compatibility issue with some brands.

7. Developer Guide

7.1 What do I need?

- x86/x64 machine running any OS; at least 4G RAM, SSD, quad core (recommended),
- [VirtualBox](#) or similar virtualization software (**highly recommended with a minimum of 25GB hard disk space for the virtual disk image**),
- Setting up VirtualBox and compile environment is easy following our [Vagrant tutorial](#),
- [Docker](#) environment is also supported for building kernels and full OS images,
- **The officially supported** compilation environment is [Ubuntu Focal 20.04 x64 only!](#) (Support for Ubuntu 18.04 will be there until either we run into issues we do not want to waste time on or upstream support ends),
- installed basic system, OpenSSH and Samba (optional),
- no spaces in full path to the build script location allowed,
- superuser rights (configured `sudo` or root shell).

Please note that system requirements (both hardware and OS/software) may differ depending on the build environment (Vagrant, Docker, Virtualbox, native).

7.2 How to start?

Native and Virtualbox environments:

Login as root and run:

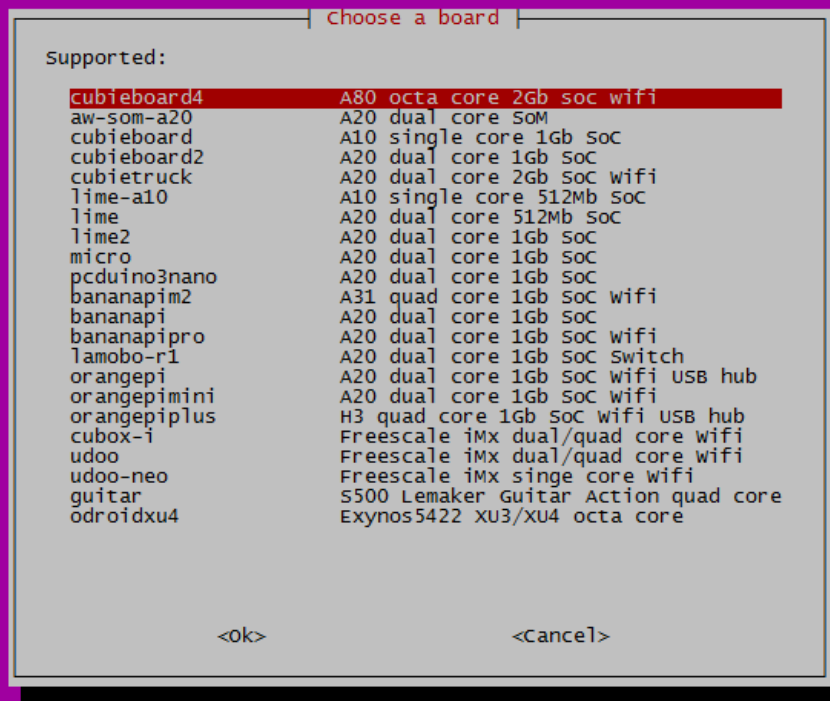
```
apt-get -y -qq install git && git clone --depth 1 https://github.com/armbian/build && cd build
```

Run the script

```
./compile.sh
```

Make sure that full path to the build script **does not contain spaces**.

Armbian building script, <http://www.armbian.com> | Author: Igor Pecovnik



Providing build configuration

After the first run of `compile.sh` a new configuration file `config-example.conf` and symlink `config-default.conf` will be created. You may edit it to your needs or create different configuration files using it as a template.

Alternatively you can supply options as command line parameters to `compile.sh`. Example:

```
./compile.sh BOARD=cubietruck BRANCH=current KERNEL_ONLY=yes RELEASE=bionic
```

Note: Option `BUILD_ALL` cannot be set to “yes” via command line parameter.

7.2.1 Base and descendant configuration

You can create one base configuration (`config-base.conf`) and use this in descendant config (`config-dev.conf`). Three parameters (BRANCH, RELEASE, COMPRESS_OUTPUTIMAGE) will be overwritten.

```
./config-base.conf BRANCH="dev" RELEASE="buster" COMPRESS_OUTPUTIMAGE="sha,gz"
```

Using our automated build system

If you do not own the proper equipment to build images on your own, you can make use of the automated build system. Packages are recompiled every night (starting at 00:01 CEST) and a few testing images are produced. These images are accessible on the [download server](#) under board folder, subfolder “Nightly”. Packages, when successfully built, are published in the beta repository. You can switch to beta repository in [armbian-config](#) or by changing apt.armbian.com to beta.armbian.com in /etc/apt/sources.list.d/armbian.list.

Board beta images are defined in board configuration files which are located [here](#). This is a typical board configuration:

```
# A20 dual core 1Gb SoC BOARD_NAME="Banana Pi" LINUXFAMILY="sun7i" BOOTCONFIG="Bananapi_defconfig" MODULES="hci_uart gpio_sunxi rfcomm hidp sunxi-i
```

You can find more information about those variables [here](#).

If you want that our automated system start making images for this particular board, you need to alter parameters `CLI_BETA_TARGET` and `DESKTOP_BETA_TARGET`. Variants are dependend from `KERNEL_TARGET` definitions and supported userlands: `buster`, `bionic`, `stretch`. To edit those parameters you need to push changes to the build script. You need to [fork a project and create a pull request](#) and after it is imported by one of the administrators, images will start to show up in appropriate folder.

If you want to enable Debian buster desktop image with *current* kernel choose the following:

```
DESKTOP_BETA_TARGET="buster:current"
```

or for command line interfaces Ubuntu Bionic based images with legacy kernel 4.19.x

```
CLI_BETA_TARGET="bionic:legacy"
```

or for image with latest upstream development kernel.

```
DESKTOP_BETA_TARGET="buster:dev"
```

Using alternate armbian builder repos and branches

By default, armbian-builder assumes working from `master` of <https://github.com/armbian/build.git>. If you are working from your own repo / branch, `touch .ignore_changes` will cause armbian-builder to not attempt a repo checkout.

Executing any bash statement

Currently, invoking `compile.sh` will run a monotonous task of building all the components into a final image.

In some situation, especially when developing with Kernel or U-Boot, it is handy to run a portion of that great task like:

```
# using default profile ./compile.sh 'fetch_from_repo "$BOOTSOURCE" "$BOOTDIR" "$BOOTBRANCH" "yes"' ./compile.sh 'compile_uboot'
```

You can also dump the variable:

```
# using profile of `userpatches/config-my.conf` ./compile.sh my 'echo $SRC/cache/sources/$BOOTSOURCE$DIR'
```

NOTE: please use single quotes to keep the `$VAR` from early expansion in the command line shell.

7.3 Quick Start with Vagrant

7.3.1 Vagrant HOST Steps

The following steps are performed on the *host* that runs Vagrant.

Installing Vagrant and Downloading Armbian

Virtualbox Version

WARNING: We'll be using [Virtualbox](#) as a virtualization provider for Vagrant.

Virtualbox has [documented issues running Xenial under heavy disk IO](#). Please make sure your version of Virtualbox is $\geq 5.1.12$ where the issue, "[Storage: fixed a problem with the LsiLogic SCSI controller where requests could be lost with SMP guests](#)", appears to have been resolved.

First, you'll need to [install Vagrant](#) on your host box. Next, you'll need to install a plugin that will enable us to resize the primary storage device. Without it, the default Vagrant images are too small to build Armbian.

```
vagrant plugin install vagrant-disksize
```

Now we'll need to [install git](#) and clone the Armbian repo. While this might seem obvious, we rely on it being there when we use Vagrant to bring up our guest-build box.

```
# Clone the project. git clone --depth 1 https://github.com/armbian/build # Make the Vagrant box available. This might take a while but only needs
```

Armbian Directory Structure

Before we bring up the box, take note of the [directory structure](#) used by the Armbian build tool. When you read the Vagrantfile (which is in the build/config/templates directory) you'll see that Vagrant will mount local *output* and *userpatches* directories. This is helpful as it enables you to easily retrieve your images from the host once built, and [customize the build process](#).

Creating the Vagrant Guest Box Used to Build

Let's bring the box up. This might take a minute or two depending on your bandwidth and hardware.

```
# We have to be in the same directory as the Vagrant file, which is in the build/config/templates directory. cd build/config/templates # Note that
```

7.3.2 Important note

It is strongly recommended to halt and restart the Vagrant box after building an image. Check [this](#) issue for details.

7.3.3 Vagrant GUEST Steps

The following steps are all run on the *guest* Vagrant created for us.

Once it's finally up and you're logged in, it works much like any of the other install methods (NOTE: again, these commands are run on the *guest* box).

```
# Let's get building! cd armbian sudo ./compile.sh
```

7.3.4 More Vagrant HOST Steps

Wrap up your vagrant box when no longer needed (log out of the guest before running these commands on the *host* system):

```
# Shutdown, but leave the box around for more building at a later time: vagrant halt # Trash the box and remove all the related storage devices. va
```

7.4 Officially supported and tested method for building with Docker

This method works for building u-boot and kernel packages as well as building full OS images.

Building additional packages (`EXTERNAL_NEW`) is not supported.

7.4.1 Requirements

- x86/x64 Linux host that supports running a recent Docker daemon. Refer to [Docker documentation](#) for details.
- Docker version 17.06 CE or newer. Installation on Ubuntu Bionic:

```
apt-key adv --keyserver pool.sks-keyservers.net --recv-keys 0EBFCD88 echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic st
```

- Enough free disk space on the storage used for Docker containers and named volumes. Named volumes path can be changed using standard Docker utilites, refer to Docker documentation for details.

7.4.2 Details

There are 2 options to start build process:

1. By passing configuration file name (`config-<conf_name>.conf`), stored in `userpatches` directory, as an argument:

```
# ./compile.sh docker <conf_name>
```

2. By passing addtional line arguments to `compile.sh` after `docker`:

```
# ./compile.sh docker KERNEL_ONLY=yes BOARD=cubietruck BRANCH=current KERNEL_CONFIGURE=yes
```

The process creates and runs a named Docker container `armbian` with 2 named volumes `armbian-cache` and `armbian-ccache`, and mount local directories `output` and `userpatches`.

7.5 Creating and running Docker container manually

NOTE: These methods are not supported by Armbian developers. Use them at your own risk.

Example: Building Armbian using Red Hat or CentOS

Tested by [@rfrht](#)

First of all, it is important to notice that you will be able to build `kernel` and `u-boot` packages. The container method is not suitable for building full Armbian images (the full SD card image containing the userland packages).

This setup procedure was validated to work with Red Hat Enterprise Linux 7.

Preparing your build host

In order to be able to run Docker containers, if you have not done so, just install the Docker package:

```
# yum install -y docker
```

By default, the `docker` service is not started upon system reboot. If you wish to do so:

```
# systemctl enable docker
```

Ensure that you have the `docker` service running:

```
# systemctl start docker`
```

Next step, chdir to a directory where you will be checking out the Armbian `build` repository. I use `/usr/src`. And then, check out using git (with shallow tree, using `--depth 1`, in order to speed up the process):

```
# cd /usr/src # git clone --depth 1 https://github.com/armbian/build
```

And in order to not mistake the newly created `build` directory, I rename it to `build-armbian`. `cd` to the directory:

```
# mv build build-armbian # cd build-armbian
```

Preparing the Container

Our Build toolchain provides a scripted way to create a container and run the container. Run:

```
# ./compile.sh docker
```

Give it some minutes, as it downloads a non-neglectible amount of data.

After your image is created (named `armbian`), it will automatically spawn the Armbian build container.

NOTICE: In some cases, it is possible that SELinux might block your access to `/root/armbian/cache` temporary build directory. You can fix it by either adding the correct SELinux context to your **host** cache directory, or, disabling SELinux.

Get acquainted with the Build system.

If you want to get a shell in the container, skipping the compile script, you can also run:

```
# docker run -dit --entrypoint=/bin/bash -v /mnt:/root/armbian/cache armbian_dev
```

The above command will start the container with a shell. To get the shell session:

```
# docker attach <UUID of your container, returned in the above command>
```

If you want to run SSH in your container, log in and install the `ssh` package:

```
# apt-get install -y ssh
```

Now, define a password and prepare the settings so you `sshd` can run and you can log in as root:

```
# passwd # sed -i -e 's/PermitRootLogin.*/PermitRootLogin yes/' /etc/ssh/sshd_config # mkdir /var/run/sshd # chmod 0755 /var/run/sshd
```

And finally start `sshd`:

```
# /usr/sbin/sshd
```

Do NOT type `exit` - that will stop your container. To leave your container running after starting `sshd`, just type `<Ctrl-P>` and `<Ctrl-Q>`. Now you can ssh to your container.

- **KERNEL_ONLY** (yes|no):
 - leave empty to display selection dialog each time
 - set to “yes” to compile only kernel, u-boot and other packages for installing on existing Armbian system
 - set to “no” to build complete OS image for writing to SD card
- **KERNEL_CONFIGURE** (yes|no):
 - leave empty to display selection dialog each time
 - set to “yes” to configure kernel (add or remove modules or features). Kernel configuration menu will be brought up before compilation
 - set to “no” to compile kernel without changing default or custom provided configuration
- **CLEAN_LEVEL** (comma-separated list): defines what should be cleaned. Default value is `"make,debs"` - clean sources and remove all packages. Changing this option can be useful when rebuilding images or building more than one image
 - “make” = execute `make clean` for selected kernel and u-boot sources,
 - “images” = delete `output/images` (complete OS images),
 - “debs” = delete packages in `output/debs` for current branch and device family,
 - “alldebs” = delete all packages in `output/debs`,
 - “cache” = delete `cache/rootfs` (rootfs cache),
 - “oldcache” = remove old `cache/rootfs` except for the newest 8 files,
 - “sources” = delete `cache/sources` (all downloaded sources),
 - “extras” = delete additional packages for current release in `output/debs/extra`
- **REPOSITORY_INSTALL** (comma-separated list): list of core packages which will be installed from repository
 - “u-boot”, “kernel”, “bsp”, “armbian-config”, “armbian-firmware”
 - “” = packages will be built from sources or use the one from local cache
- **KERNEL_KEEP_CONFIG** (yes|no):
 - set to “yes” to use kernel config file from previous compilation for the same branch, device family and version
 - set to “no” to use default or user-provided config file
- **BUILD_MINIMAL** (yes|no):
 - set to “yes” to build bare CLI image suitable for application deployment. This option is not compatible with `BUILD_DESKTOP=“yes”` and `BUILD_EXTERNAL=“yes”`
- **BUILD_DESKTOP** (yes|no):
 - set to “yes” to build image with minimal desktop environment
 - set to “no” to build image with console interface only
- **EXTERNAL** (yes|no):
 - set to “yes” to compile and install extra applications and firmware
- **BSPFREEZE** (no|yes): freeze (from update) armbian packages when building images (u-boot, kernel, dtb)
- **INSTALL_HEADERS** (no|yes): install kernel headers package

- **EXTERNAL_NEW** (no|prebuilt|compile):
 - set to “prebuilt” to install extra applications from repository
 - set to “compile” to compile extra applications in chroot
- **CREATE_PATCHES** (yes|no):
 - set to “yes” will prompt you right before the compilation starts to make changes to the source code. Separate for u-boot and kernel. It will also create a patch out of this. If you want that this patch is included in the normal run, you need to copy it to appropriate directory
 - set to “no” compilation will run uninterrupted
- **BUILD_ALL** (yes|no|demo): cycle through all available board and kernel configurations and make images for all combinations
- **LIB_TAG** (master|“branchname”):
 - set to “master” to compile from the master branch (default)
 - set to “branchname” to compile from any other branch available (“next” & “second” are deprecated and **not** recommended to use).
- **CARD_DEVICE** (/dev/sdx) set to the device of your SD card. The image will be burned and verified using Etcher for CLI.
- **CRYPTROOT_ENABLE** (yes|no): set to enable LUKS encrypted rootfs. You must also provide unlock password CRYPTROOT_PASSPHRASE=“MYSECRECTPASS” and optional CRYPTROOT_SSH_UNLOCK=yes CRYPTROOT_SSH_UNLOCK_PORT=2222 CRYPTROOT_PARAMETERS=“custom cryptsetup options” Function might not work well with all distributions. Debian Buster and Stretch were tested. For building under the Docker you have to use privilege mode which can be enable in userpatches/config-docker. **Warning:** This feature was added as community contribution and mostly functional. Under some circumstances though the prompt will not be shown. Therefore it should be considered experimental.

More info:

[1] <https://github.com/armbian/build/commit/681e58b6689acda6a957e325f12e7b748faa8330>

[2] <https://github.com/armbian/build/issues/1183>

Hidden options to minimize user input for build automation:

- **BOARD** (string): you can set name of board manually to skip dialog prompt
- **BRANCH** (legacy|current|dev): you can set kernel and u-boot branch manually to skip dialog prompt; some options may not be available for all devices
- **RELEASE** (stretch|buster|bionic|focal): you can set OS release manually to skip dialog prompt; use this option with `KERNEL_ONLY=yes` to create board support package

Hidden options for advanced users (default values are marked bold):

- **EXPERT** (yes|no): Show development features in interactive mode
- **USERPATCHES_PATH** (userpatches/): set alternate path for location of `userpatches` folder
- **USE_CCACHE** (yes|no): use a C compiler cache to speed up the build process
- **PRIVATE_CCACHE** (yes|no) use `$DEST/ccache` as ccache home directory
- **PROGRESS_DISPLAY** (none|plain|dialog): way to display output of verbose processes - compilation, packaging, debootstrap
- **PROGRESS_LOG_TO_FILE** (yes|no): duplicate output, affected by previous option, to log files `output/debug/*.log`
- **USE_MAINLINE_GOOGLE_MIRROR** (yes|no): use `googlesource.com` mirror for downloading mainline kernel sources, may be faster than `git.kernel.org` depending on your location
- **USE_GITHUB_UBOOT_MIRROR** (yes|no): use unofficial Github mirror for downloading mainline u-boot sources, may be faster than `git.denx.de` depending on your location
- **OFFLINE_WORK** (yes|no): skip downloading and updating sources as well as time and host check. Set to “yes” and you can collect packages without accessing the internet.
- **FORCE_USE_RAMDISK** (yes|no): overrides autodetect for using tmpfs in new debootstrap and image creation process
- **FIXED_IMAGE_SIZE** (integer): create image file of this size (in megabytes) instead of minimal
- **COMPRESS_OUTPUTIMAGE** (comma-separated list): create compressed archive with image file and GPG signature for redistribution
 - *sha* - generate SHA256 hash for image,
 - *gpg* - sign image using gpg,
 - *7z* - compress image, hash and signature to 7z archive,
 - *gz* - compress image only using gz format,
 - *yes* - compatibility shortcut for sha,gpg,7z.
- **SEVENZIP** (yes|no): create .7z archive with extreme compression ratio instead of .zip
- **BUILD_KSRC** (yes|no): create kernel source packages
- **ROOTFS_TYPE** (ext4|f2fs|btrfs|nfs|fel): create image with different root filesystems instead of default ext4. Requires setting `FIXED_IMAGE_SIZE` to something smaller than the size of your SD card for F2FS
- **BTRFS_COMPRESSION** (lzo|zlib:3|zstd) select btrfs filesystem compression method and compression level. By default the compression is `lzo`, user must ensure kernel version is above `4.14` when selecting `zstd` or setting zlib compression level(`zlib:[1-9]`). Both the host and the target kernel version must above `5.1` when selecting zstd compression level (`zstd:[1-15]`), since kernel start supporting zstd compression ratio from `5.1`. The script does not check the legality of input variable(compression ratio), input like `zlib:1234` is legal to script, but illegal to kernel. When using microsd card, `zstd` is preferred because of the poor 4k I/O performance of microsd card.
- **FORCE_BOOTSCRIPT_UPDATE** (yes|no): set to “yes” to force bootscript to get updated during bsp package upgrade

- **NAMESERVER** (ipv4 address): the DNS resolver used inside the build chroot. Does not affect the final image.
Default: 1.0.0.1
- **DOWNLOAD_MIRROR** select download mirror for `toolchain` and `debian/ubuntu packages`.
 - set to `china` to use `mirrors.tuna.tsinghua.edu.cn`, it will be very fast thanks to tsinghua university.
 - leave it empty to use official source.
- **MAINLINE_MIRROR** select mainline mirror of `linux-stable.git`
 - set to `google` to use mirror provided by Google, the same as `USE_MAINLINE_GOOGLE_MIRROR=yes`.
 - set to `tuna` to use mirror provided by tsinghua university.
 - leave it empty to use official `git.kernel.org`, it may be very slow for mainland china users.
- **USE_TORRENT** (yes|no): use torrent to download toolchains and rootfs
- **ROOT_FS_CREATE_ONLY** set to `FORCE` to skip rootfs download and create locally

7.5.1 User provided patches

You can add your own patches outside build script. Place your patches inside appropriate directory, for kernel or u-boot. There are no limitations except all patches must have file name extension `.patch`. User patches directory structure mirrors directory structure of `patch`. Look for the hint at the beginning of patching process to select proper directory for patches. Example:

```
[ o.k. ] Started patching process for [ kernel sunxi-dev 4.4.0-rc6 ] [ o.k. ] Looking for user patches in [ userpatches/kernel/sunxi-dev ]
```

Patch with same file name in `userpatches` directory tree substitutes one in `patch`. To *replace* a patch provided by Armbian maintainers, copy it from `patch` to corresponding directory in `userpatches` and edit it to your needs. To *disable* a patch, create empty file in corresponding directory in `userpatches`.

7.5.2 User provided configuration

If file `userpatches/lib.config` exists, it will be called and can override the particular kernel and u-boot versions. It can also add additional packages to be installed, by adding to `PACKAGE_LIST_ADDITIONAL`. For a comprehensive list of available variables, look through `lib/configuration.sh`. Some examples of what you can change:

```
PACKAGE_LIST_ADDITIONAL="$PACKAGE_LIST_ADDITIONAL python-serial python" # additional packages [[ $LINUXFAMILY == sunxi64 && $BRANCH == dev ]] && BO
```

7.5.3 User provided kernel config

If file `userpatches/linux-$KERNELFAMILY-$KERNELBRANCH.config` exists, it will be used instead of default one from `config`. Look for the hint at the beginning of kernel compilation process to select proper config file name. Example:

```
[ o.k. ] Compiling dev kernel [ @host ] [ o.k. ] Using kernel config file [ config/linux-sunxi-dev.config ]
```

7.5.4 User provided sources config overrides

If file `userpatches/sources/$LINUXFAMILY.conf` exists, it will be used in addition to the default one from `config/sources`. Look for the hint at the beginning of compilation process to select proper config file name. Please note that there are some exceptions for LINUXFAMILY like `sunxi` (32-bit mainline sunxi) and `sunxi64` (64-bit mainline sunxi)

Example:

```
[ o.k. ] Adding user provided sunxi64 overrides
```

7.5.5 User provided image customization script

You can run additional commands to customize created image. Edit file:

`userpatches/customize-image.sh`

and place your code here. You may test values of variables noted in the file to use different commands for different configurations. Those commands will be executed in a chroot environment just before closing image.

To add files to image easily, put them in `userpatches/overlay` and access them in

`/tmp/overlay` from `customize-image.sh`

7.5.6 Partitioning of the SD card

In case you define `$FIXED_IMAGE_SIZE` at build time the partition containing the rootfs will be made of this size. Default behaviour when this is not defined is to shrink the partition to minimum size at build time and expand it to the card's maximum capacity at boot time (leaving an unpartitioned spare area of ~5% when the size is 4GB or less to help the SD card's controller with wear leveling and garbage collection on old/slow cards).

You can prevent the partition expansion from within `customize-image.sh` by a `touch /root/.no_rootfs_resize` or configure the resize operation by either a percentage or a sector count using `/root/.rootfs_resize` (`50%` will use only half of the card's size if the image size doesn't exceed this or `3887103s` for example will use sector 3887103 as partition end. Values without either `%` or `s` will be ignored)

7.6 FEL/NFS boot explanation

What is FEL/NFS boot?

FEL/NFS boot mode is a possibility to test freshly created Armbian distribution without using SD card. It is implemented by loading u-boot, kernel, initrd, boot script and .bin/.dtb file via [USB FEL mode](#) and providing root filesystem via NFS share.

NOTE: this mode is designed only for testing. To use root on NFS permanently, use `ROOTFS_TYPE=nfs` option. NOTE: “hot” switching between kernel branches (default <-> dev/next) is not supported

Requirements

- Allwinner device that supports FEL mode. Check [wiki](#) to find out how to enter FEL mode with your device
- USB connection between build host and board OTG port (VM USB passthrough or USB over IP may work too)
- Network connection between build host and board. For target board **wired** Ethernet connection is required (either via onboard Ethernet or via USB ethernet adapter that has required kernel modules built-in)
- NFS ports on build host should be reachable from board perspective (you may need to open ports in firewall or change network configuration of your VM)
- Selected kernel should have built-in support for DHCP and NFS root filesystem
- `CLEAN_LEVEL="make,debs"` to always update u-boot configuration

Additional requirements (recommended)

- DHCP server in local network
- UART console connected to target board

Build script options

- `KERNEL_ONLY=no`
- `ROOTFS_TYPE=fel`

Example:

```
./compile.sh KERNEL_ONLY=no BOARD=cubietruck BRANCH=current PROGRESS_DISPLAY=plain RELEASE=jessie BUILD_DESKTOP=no ROOTFS_TYPE=fel
```

Shutdown and reboot

Once you start FEL boot, you will see this prompt:

```
[ o.k. ] Press any key to boot again, <q> to finish [ FEL ]
```

Pressing `q` deletes current rootfs and finishes build process, so you need to shut down or reboot your board to avoid possible problems unmounting/deleting temporary rootfs. All changes to root filesystem will persist until you exit FEL mode.

To reboot again into testing system, switch your board into FEL mode and press any key other than `q`.

Because kernel and `.bin/.dtb` file are loaded from rootfs each time, it's possible to update kernel or its configuration (via `apt-get`, `dtc`, `fex2bin`/`bin2fex`) from within running system.

Advanced configuration

If you don't have DHCP server in your local network or if you need to alter kernel command line, use `lib/scripts/fel-boot.cmd.template` as a template and save modified script as `userpatches/fel-boot.cmd`. Check [this](#) for configuring static IP for NFS root

Set `FEL_DTB_FILE` to relative path to `.dtb` or `.bin` file if it can't be obtained from u-boot config (mainline kernel) or `boot/script.bin` (legacy kernel)

You may need to set these additional options (it's a good idea to put them in `userpatches/lib.config`):

Set `FEL_NET_IFNAME` to name of your network interface if you have more than one non-loopback interface with assigned IPv4 address on your build host

Set `FEL_LOCAL_IP` to IP address that can be used to reach NFS server on your build host if it can't be obtained from ifconfig (i.e. port forwarding to VM guest)

Set `FEL_AUTO=yes` to skip prompt before trying FEL load

Customization

You can even create `userpatches/fel-hooks.sh` and define there 2 functions: `fel_post_prepare` and `fel_pre_load`. All normal build variables like `$BOARD`, `$BRANCH` and so on can be used in these functions to define specific actions.

`fel_post_prepare` is executed once after setting up u-boot script and NFS share, you can use it to add extra stuff to boot.scr (like `gpio set` or `setenv machid`) based on device name.

`fel_pre_load` is executed before calling sunxi-fel, you can use it to implement logic to select one of multiple connected boards; to pass additional arguments to `sunxi-fel` you can use `FEL_EXTRA_ARGS` variable.

An example is provided as `scripts/fel-hooks.sh.example`.

8. Contributor Process

8.1 Collaborate on the project

8.1.1 How?

1. [Fork](#) the project
2. Make one or more well commented and clean commits to the repository.
3. Perform a [pull request](#) in github's web interface.

If it is a new feature request, don't start the coding first. Remember to [open an issue](#) to discuss the new feature.

If you are struggling, check [WEB](#) or [CLI](#) step by step guide on contributing.

8.1.2 Where are sources?

Build script:

<https://github.com/armbian/build>

Documentation:

<https://github.com/armbian/documentation>

Armbian-config tool:

<https://github.com/armbian/config>

8.2 Help with donations

If you find our project useful, then we'd really appreciate it if you'd consider contributing to the project however you can. Donating is the easiest way to help us – you can use PayPal and Bitcoin or you can buy us something from our Amazon.de wish list.

<http://www.armbian.com/donate/>

Thanks!

8.3 Merge Policy

8.3.1 Overview

Note: This document is a Work In Progress and is subject to change. Definitions may be relocated to a seperate document in the future.

This policy is targeted for Maintainers for Lead Maintainers who have commit access to `master` branch. This document describes the tags needed for different mege types. See Definitions.

The types of code maintained fall into the following categories:

- Kernel
- U-Boot
- Build Scripts

Kernel and U-Boot maintainers are usually grouped by SoC Architecture.

Supported boards will have the most scrunity with code review.

8.3.2 U-Boot Patches

- Standard Contributors provide *tested-by* submitter (`armbianmonitor -u` with a fresh build)
- SoC maintainer maybe submit a PR with only a reviewed by of the lead SoC maintainer

8.3.3 Kernel Related Patches

default and next branches

- DT changes reviewed by at least one person familiar with this SoC (lead maintainer or deputy), *tested-by* the contributor who sends it (armbianmonitor)..
- trivial module activation doesn't matter

dev branches

Constraints are at the discretion of the SoC mantainer. This builds are not expected to be stable.

8.3.4 Armbian Build Scripts

This pertains to code used to build system images, OS configuration, and supporting packages. (Basically Anything not U-Boot or Kernel source)

lib scripts

- Requires at least one *Reviewed-by*

configuration

board promotion

Boards have different levels of commitment / support. EOL, CSC, WIP, Supported. To promote a board from WIP to Supported an Aced-by from a Lead Maintainer is required.

kernel config

- Changes in default & next kernel config should provide at least *tested-by* with `armbianmonitor -u`
- Changes in dev are at the discretion of maintainer. No constraints

kernel sources

Change kernel source repos, branches, versions can be very disruptive to stable builds. Sufficient communication should occur stable changes.

- u-boot and kernel version bump for default and next requires *tested-by* from Maintainers and/or testers on at least two different boards for the impacted platform.
- kernelsources switch (next default) needs a discussion on forum or github documented in PR and *Aced-by* Lead Maintainer. These changes are risky and things can go terrible wrong here...
- dev source changes are at the discretion of the Lead Maintainer
- Boardfamily tweaks require at least *reviewed-by*

packages

- minimum require *Aced-by*

8.3.5 Definitions

Code Review Terms

[click here for attribution to terms used below](#)

Signed-off-by

Certifies that you wrote it or otherwise have the right to pass it on as a open-source patch.

Acked-by

If a person was not directly involved in the preparation or handling of a patch but wishes to signify and record their approval of it then they can arrange to have an Acked-by: line. Acked-by: does not necessarily indicate acknowledgement of the entire patch.

Tested-by

A Tested-by: tag indicates that the patch has been successfully tested (in some environment) by the person named. This tag informs maintainers that some testing has been performed, provides a means to locate testers for future patches, and ensures credit for the testers.

Reviewed-by

A Reviewed-by tag is a statement of opinion that the patch is an appropriate modification of the kernel without any remaining serious technical issues. Any interested reviewer (who has done the work) can offer a Reviewed-by tag for a patch.

Other

Maintainer

An Individual designated to moderate, support and make decisions for a codebase or component. There can be multiple maintainers assigned to any given thing.

Lead Maintainer

A more experienced maintainer that will decide on high-impact and strategic changes and have final say in disputes. A lead maintainer may share or designate responsibility to some or all components within their domain of responsibility.

SoC

System on-a-Chip.

8.4 Armbian documentation hosted on Github pages

On PUSH documentation will be rebuilt and pushed to <https://github.com/EvilOlaf/docutest>

Web version is published here: <https://evilolaf.github.io/docutest/>

PDF version is published here: <https://github.com/EvilOlaf/docutest/blob/master/pdf/document.pdf>

9. Release management

9.1 Release model

Release Dates

Armbian runs “train” based releases. Whatever is ready to board the train, does so. Whatever isn’t ready, has to wait for the next train. This enables us to have a predictable release cycles making it easy to plan. It also puts the responsibility on developers to make sure they have features ready on time.

Armbian releases quarterly at the end of **February, May, August, November**. Offset is because we all know that nothing happens for half of December. At the beginning of a release cycle, we have a planning meeting and **two weeks before the end of the release we freeze integration of new features**.

Release Cycle

Releases last 3 months. Each release starts with a planning meeting. After planning, developers and development teams build their deliverable using whatever methods (scrum, kanban, waterfall, ...) they want but shall commit their code frequently, leading **up to the last 2 weeks**. The project does not accept “dumps” of code at the end. **Commit early and often** on master. Two weeks before the release date, we halt feature integration and only allow bug fixes. At some point during those two weeks, we start the release candidate process. This process starts by pulling a branch off master that will become the release branch. That frees up master for development on the next release. On the release candidate branch we work on bug fixes, and choose “release candidate”, RC, tags. The software at that tag is a candidate for release, and it is submitted to automated and manual tests on real hardware. If automated tests are passing, we can officially tag it as the release. If it doesn’t, we enter another bug fix cycle and create a new release candidate. We iterate until we have a candidate that can be the formal release. Usually, this takes 2-3 cycles and 1-3 weeks of time.

Development epics, stories and bugs for each release are tracked through [Jira](#).

9.2 Release Branching, Versioning and Tags

Branches in Armbian follow this convention:

- **Master branch (master):** Main development will happen on the master branch. This is the latest and greatest branch, but is always “stable” and “deployable”. All tests always pass on this branch.
- **Maintenance branch (support):** This is the long-term maintenance branch per release.
- **Development branch (dev):** This is a branch created for lengthy and/or involved feature development that could destabilize master.

Each Armbian release will have the following version format:

Format: `<major>.<minor>.<revision>`

`<major>` and `<minor>` version will be incremented at the end of the release cycles while `<revision>` is incremented for a fix (or set of fixes)

Tags are used in ad-hoc manner.

9.3 Release Naming

| version | codename | release month | work |
|---------|----------|---------------|---------|
| 19.11 | Vaquita | November | done |
| 20.02 | Chiru | February | done |
| 20.05 | Kagu | May | done |
| 20.08 | Caple | August | planned |
| 20.11 | Tamandua | November | |

by <http://www.codenamegenerator.com/>

9.4 Release Planning

A release planning starts with an public IRC meeting where developers and interested users come together in **first Saturday, one month before the release month.**

Dates for **2020**:

- [April 4th](#)
- July 4th
- October 3rd

Agenda:

- check meeting attendees (if nick is not self explanatory, add your forum/github handle. Just say hi or something)
- choose upcoming release officer (so far it was me and Lane)
- present tasks, bugs or project you are working on (open discussion if there will not be much people, otherwise meeting officer call people out). Jira should be open in not already.
- cycle Jira backlog:
- discuss task / bug (one at a time)
- assign to person / release / tag
- re-prioritise
- cycle open issues and PR on build engine
- board status update on download pages and build engine (wip, supported, eol)
- change (build) branch protection rule to “Require pull request reviews before merging”
- decide upon best meetings hours
- misc / open discussion

Tips:

- when you got a voice, be concise (1-2 min) and make it clear when you stop. (“No more, I’m done”)
- channel is recorded so a summary and adjustments to Jira can made afterwards, ideally along with the meeting

Meeting location is IRC channel [#armbian](#) on [freenode](#). Meeting starts **at 2pm GMT**.

9.5 Release Coordinating

9.5.1 Summary

A release starts as a RC Branch cut from master at freeze time. Once a RC Branch is cut, master can be unfrozen and development can continue. RC Branches accept bug fixes. The bug fixes should be cherry-picked back to master branch. Once the RC Is stable. A Final release is made. A release is *never* merged to master. Once a release is complete, it only should be updated for severe bugs and severe security vulnerabilities. A release is only maintained until the next release.

9.5.2 1. Forum Communication

- Create a new thread in the [Armbian Build Framework Subforum](#)
- Ex topic name: `Ambian 20.02 (Chiru) Release Thread`
- Tag the post with release, release version, and codename
- Use the following template to begin the body of the release thread:

`Release Candidate Code Freeze Date: YYYY-MM-DD Release Date: YYYY-MM-DD Release Candidate Branch Link: URL Release Changelog: URL Release Coordinat`

- Before Code Freeze – Make note in the thread the incomplete jira issues tagged for the release [example](#)
- After test images are procuded, engage in community for assistants wih testing.. forums, twitter, etc. [share this tool](#)

9.5.3 2. Release Candidate Branch Management

- For code freeze – create a RC branch in the format `(vYY.MM-rcX)` ex: `v20.02-rc1`
- If Possible, create Jira tickets for major changes in github that were not tracked in Jira
- Begin Testing Process. See [Release Testing](#)
- Do not modify branch directy. Only accept PRs
- Only accept PRs for Bugfixes. No features
- Update master branch version to the NEXT release version with `-trunk` ex. If RC is v20.02.0-rc1 Master bacomes v20.05.0-trunk
- FIXME ? Coordinate with Igor or other Admins generate test build from branch ?
- Repeat build, test, and bugfix process until release is stable
- Cherry-pick bugfixes back into master
- Create Final Release branch from RC

9.5.4 3. Release

- In Github create a Release from final release branch
- Copy Release notes generated by Jira Release into Github form
- Add other appropriate information into release github release notes
- Point Armbian build system to new release
- Update armbian documentation to reflect current release
- Celebrate

9.6 Release Testing

See [Opportunties for improvement](#)

9.7 Reflection on Prior Releases

9.7.1 Opportunities for Improvement

9.7.2 wireless driver testing

- wireless is a particularly sensitive issue. We need to test, fix, or at least be able to inform others of what is broken

Bug Tracking

Testing

Image Downloads

9.7.3 Positive Observations

- Good response from community for testing assistance
- Release was on time

9.8

20.05.6 / 19.06.2020

- [\[AR-324\]](#) - Add Rockchip RK322X SoC support

20.05.4 / 16.06.2020

- [\[AR-311\]](#) - Initrd on Focal can get corrupted followup fix

20.05.3 / 10.06.2020

- [\[AR-300\]](#) - Enable HDMI audio for OrangePi 4
- [\[AR-305\]](#) - K-worker creates load on Allwinner devices
- [\[AR-282\]](#) - Rockpi 4B 1Gb doesn't boot modern kernel / u-boot

20.05.2 / 05.06.2020

- [\[AR-294\]](#) - Initrd on Focal can get corrupted

10. Community

wgawegawgeawgawgaweg