



OLIMAGE **GUIDE**

Linux for Olimex Allwinner and STM32MP1 boards

Document revision F, April 2025

Table of Contents

1. What is this?	4
2. Quickstart	5
2.1. Get the archive with the image	6
2.2. Extract the image	7
2.3. Download the image to an SD card	7
2.4. Connect debug interface and all peripherals	8
2.5. Start the serial monitor software	9
2.6. Powering the board	10
(Optional) 2.7. Update the image	11
(Optional) 2.8. Explore the configuration scripts	11
2.9. Turning off the board	11
3. About the Olimage Linux image	12
3.1. Supported products	12
3.2. What works and what doesn't work in Olimage	12
4. Olimage ready images	13
5. Build information and sources	14
6. Accessing a board running Olimage	15
6.1. USB-serial adapter	16
6.2. HDMI cable	16
6.3. USB cable	17
6.4. LCD cable	18
6.5. Ethernet cable	19
7. The boot process	20
7.1 U-boot tools	20
8. First steps with Olimage	21
8.1. Test the Ethernet	21
8.2. Test the scripts	22
8.3. Main chip and power management chip sensor support	22
8.4. How to compile C files	23
8.5. How to install Python and packages	24
8.6. How to toggle a GPIO pin via sysfs	25
8.7. I2C	29
8.8. SPI	34
8.9. CAN	35
9. What other images are available	39
10. Frequently asked question	40
11. Instructions on rebuilding DTS and kernel	41
11.1. How to build Device Tree Source (DTS) file for Olimage	41
11.2. How to build Linux Kernel for Olimage	43

12. How to send feedback or report a problem.....45

13. Document revision.....45

1. What is this?

This guide provides the reader with information about the basic features and the usage of the Olimage Linux images released and maintained by Olimex. All resources used to build Olimage Linux can be found at our GitHub repository.

Olimage is made to run on Olimex-made boards with Allwinner or STM32MP1 main chip. It supports almost all Olimex boards with the following processors – A10, A13, A20, A64, STM32P15x.

Olimage is a free open-source software effort, using it or testing it requires no purchase.

The Olimage Linux image is made to be used with OLIMEX-made boards with Allwinner A10, A13, A20, A64, and STM32P15x series processors and all their variants (albeit some hardware features might not be supported). The only exceptions at the moment of writing this document are A33-OLinuXino board, and A13-SOM-256 board, and A13-OLinuXino-MICRO board – they are not supported by Olimage.

The Linux image support some of the extension hardware that Olimex manufactures (like LCD displays, USB extensions, UEXT extensions).

Olimage is the recommended image for almost all Olimex-made boards with Allwinner-made or STM-made main chip. The exceptions are the A13-SOM-256, A13-OLinuXino-MICRO board, and the A33-OLinuXino board which are currently not supported by Olimage.

The Olimage Linux image is not suitable for the Olimex Linux boards with RK3328 or iMX8MP chip. RK3328 and iMX8MP boards run modified Armbian-based image, you can refer to official Armbian documentation for these boards. The boards iMX8 chip also come with buildroot setup.

This guide is intended for absolute beginners. It does not attempt to be complete nor it aims to cover all possible hardware and software scenarios. More experienced users would find this document redundant.

This guide starts with quickstart instructions and more detailed explanation follows.

2. Quickstart

The guide assumes that the user has access to:

- a computer with access to the Internet;
- a micro SD card reader and suitable micro SD card;
- a USB-serial adapter with 3.3V levels;
- a supported by Olimage Olimex-made Allwinner-based or STM32MP1-based board;
- a fitting power source to power the board.

The general algorithm for a first-time boot is:

- (1) Download the archive with the Olimage image;
- (2) Extract the archive and write it to a micro SD card;
- (3) Place the micro SD card in your board and attach the debug medium;
- (4) Power the board;
- (5) Login with username "root" and password "olimex";
- (6) Perform a software reboot by typing command "reboot now".

Each of these steps is further detailed in subsequent chapters.

You can usually save some trouble and time and requirements by getting a pre-loaded with Linux micro SD card suitable for your board (for example, [A20-XXX-DEBIAN-SD](#) or [A13-OLinuXino-DEBIAN-SD](#)) but such is not required, a blank SD card is completely sufficient. Even if you have a ready-to-boot card is good idea to still have a micro SD reader-writer, in case the files on the card get corrupted somehow.

You can find affordable options for some of the required hardware at our web-site:

- USB-serial converter with free leads:

<https://www.olimex.com/Products/Breadboarding/BB-CH340T/>

or

<https://www.olimex.com/Products/Components/Cables/USB-Serial-Cable/USB-SERIAL-F/>

- Blank micro SD card:

<https://www.olimex.com/Products/Components/Storage/MICRO-SD-CLASS10/>

- Card reader and writer:

<https://www.olimex.com/Products/USB-Modules/USB-CARD-READER/>

- Power supply options:

<https://www.olimex.com/Products/Power/>

2.1. Get the archive with the image

You can skip straight to chapter “2.4. Connect debug interface and all peripherals” if you have purchased a card with Olimage Linux.

Navigate to this web page:

<https://images.olimex.com/release/>

Get into the sub-folder with the name of the main chip of your Olimex board and download the newest Debian 7z archive. The images are archived to reduce the size.

For example, if you have A20-OLinuxino-LIME2-e16Gs16M, navigate to folder “A20” and click on the highlighted link shown in the screenshot below:

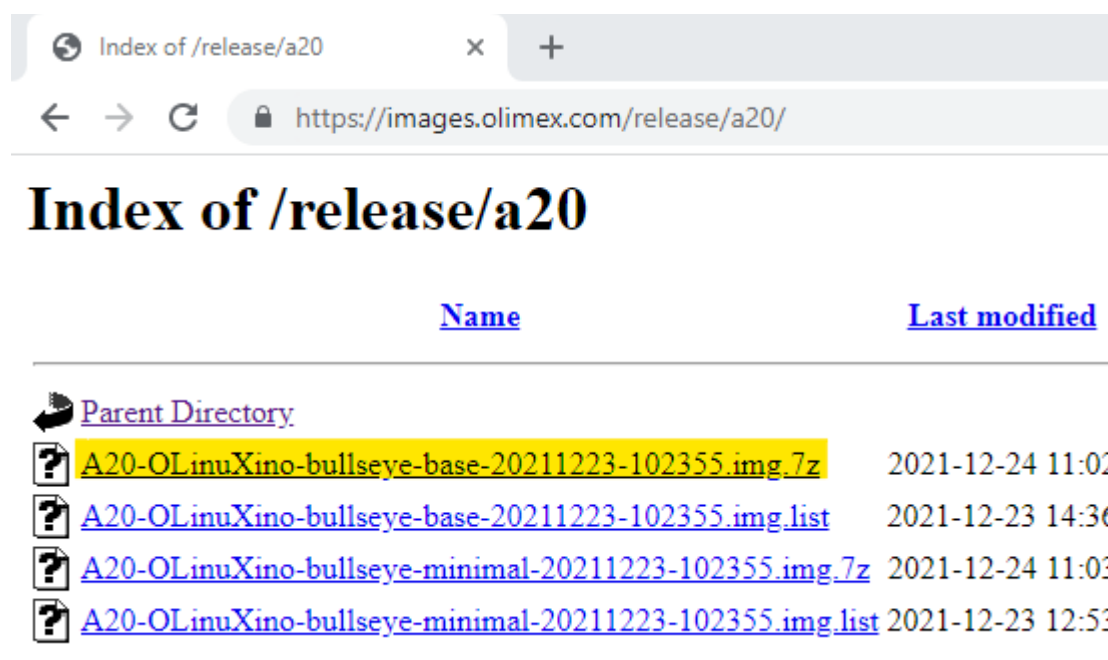


Illustration 1: Web-page with A20 official images

Select a location to save the file and wait for the download to finish.

IMPORTANT! If you have A10, A13, A64, or STM32 board you need to navigate to the respective folder and download suitable for the on-board chip image! Notice that there are separate folders for the OLinuxino and SOM variants of the STM32MP1 series boards.

2.2. Extract the image

Once the 7z archive is downloaded, extract it. If you don't have software suitable for extracting 7z file format already then find free options online. The best place to get such software is usually this web-site:

<https://www.7-zip.org/>

After 7-zip software had been successfully installed, right-click over the archive folder and navigate to "7-Zip" option and then select "Extract Here".

Extracting the archive would result in a file with file extension "img" that has to be downloaded to the card and a md5 file that contains a check sum – this checksum can be used to verify the integrity of the image.

2.3. Download the image to an SD card

You need a micro SD card. It should be at least 4GB of size and there is no maximum size limitation for the card. However, there is speed recommendation. **For best experience use a micro SD card with speed rating between class 10 and A1.** Using slower than class 10 card might severely degrade the user experience. Beware that A2 or faster cards might lack software support in Linux.

It is recommended to test the SD card beforehand. Most of the problems that appear in a first time boot are related to either the SD card or the powering. There are a lot of counterfeit SD cards out there and also cards tend to get damaged easily. Good software for testing the SD cards is either [f3 \(fight flash fraud\)](#) or H2testW.

Connect the micro SD card to a computer, if there is no built-in card reader you would need an external SD card reader and writer hardware.

At this point you need to use software to download the image to the card. We recommend using [BalenaEtcher](#) or [USBImager](#) software – they are open source source, free, have releases for Windows/macOS/Linux and using them minimizes the chances of a mistake.

I used BalenaEtcher, got it from the official site and installed. Then I start it, click on "Select image" and point to the Linux image file that I extracted. Make sure the SD card drive is selected and click "Flash!".

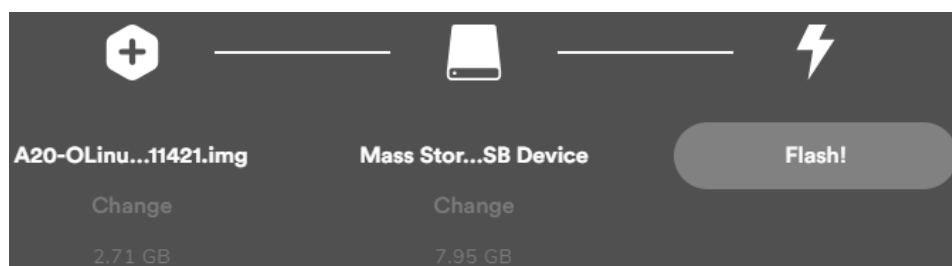


Illustration 2: Etcher ready to burn the image to the micro SD card.

Waited until it completed, removed the card from the PC and placed the it in the board.

2.4. Connect debug interface and all peripherals

Plug the prepared micro SD card in the board and also attach your debug medium. **It is recommended to use USB-serial converter cable attached to UART debug pins of your board.** This approach provides access to only the command line interface but it is the most resilient medium when it comes to software configuration problems. It also allows reliable access to the board even before the Linux kernel starts. Access to u-boot is quite useful since it allows configuring the EEPROM of the board via Olimex-made set of tools. A typical hardware setup looks like the picture shown below:

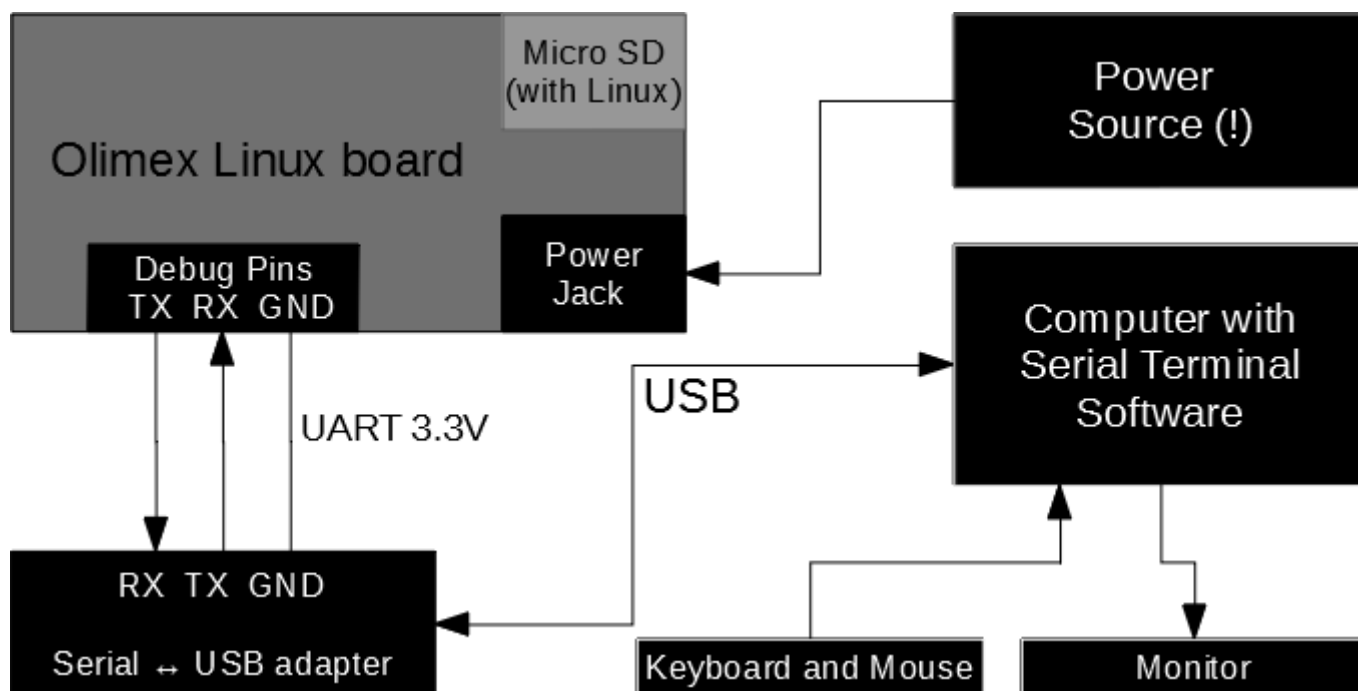


Illustration 3: Recommended hardware setup with serial ↔ USB adapter

Leave the power supply unconnected until you have set everything else together.

Important! Most boards allow other means of accessing them in case you don't currently have a serial-USB converter cable:

- (1) via HDMI connector and monitor;
- (2) via the mini USB or micro USB connector (virtual serial port);
- (3) via LCD connector and display;
- (4) via Ethernet connector.

Using some of these methods might not allow accessing the u-boot configuration (since the software required to get these methods running gets loaded at a later boot stage). These methods are described further below.

We usually use Olimex BB-CH340T USB-serial converter and jumper wires to connect to most Olimex Allwinner boards. These are available in our shop.

If you use USB-SERIAL-CABLE-F then the blue wire is GND, green wire is RX, red wire is TX.

2.5. Start the serial monitor software

Find what is the serial port assigned by your operating system to the USB-serial cable.

For example, I use BB-CH340T and the port it gets assigned in Windows can be found in “Windows Device Manager” in the “Ports (COM & LPT)” section, as show in the picture below:

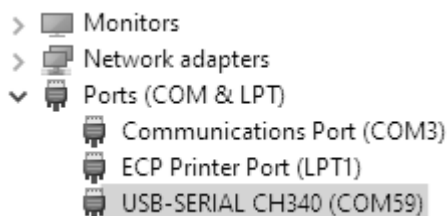


Illustration 4: COM # 59 assigned

Once you know the number start your serial terminal software on the port assigned. If you use another serial-USB converter, the name and driver listed would be different!

In our example configuration, we would use PuTTY under Windows and it looks like that:

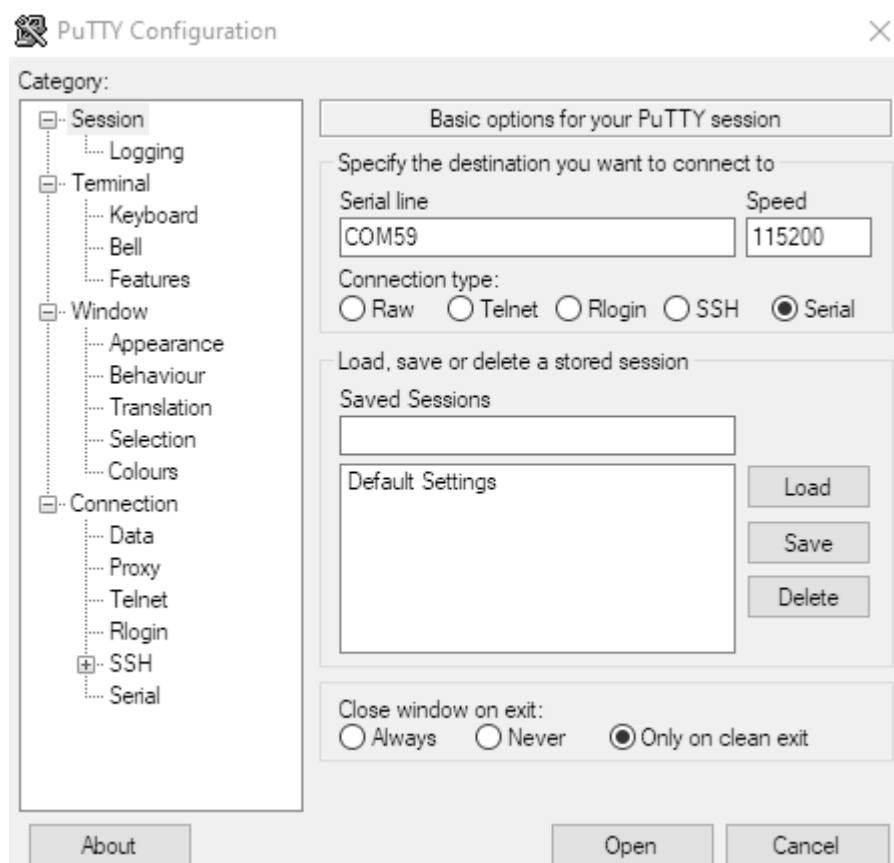


Illustration 5: Starting PuTTY session

The serial settings depend on the adapter used. I used pretty much the default ones (115200 baud, data bits 8, stop bits 1, parity set to none, flow control disabled). For Linux you can use picocom or minicom or similar tool. Open the serial connection and power the board.

2.6. Powering the board

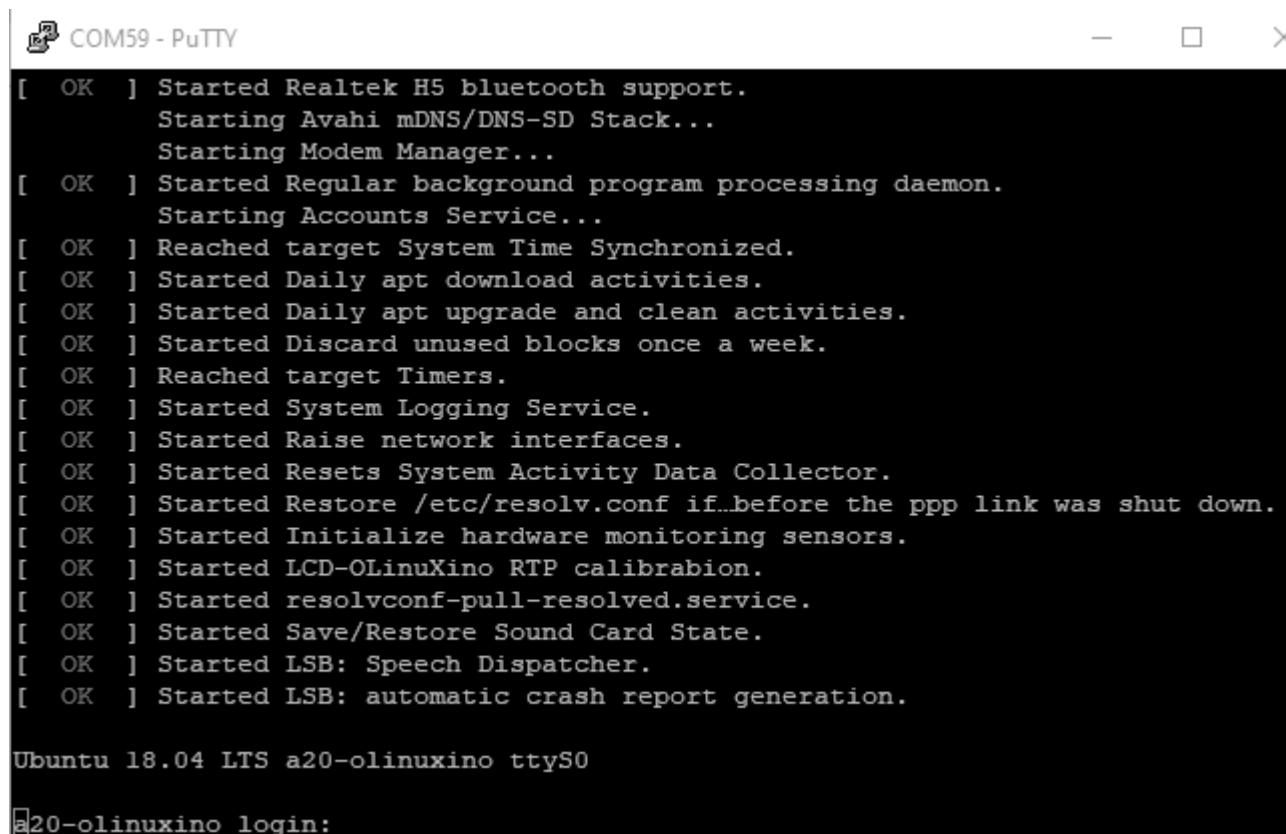
Most Olimex Allwinner boards require external power supply applied to the DC barrel jack! Make sure to double check voltage needed by your board – most boards require 5V. If you provide 12V of voltage to a 5V-only board, you would damage it. If the boot stops abruptly without any warning in the logs – the culprit is usually the powering!

(!) Make sure that you have a fitting external power supply according to the specification of your board. Some boards require 5V DC power supply meaning that any higher voltage might damage them. Other boards require 12V DC!

(!) Most Olimex boards have a standard power supply jack with positive inner pin. The corresponding power supply plug (or male jack) that fits the DC barrel has 2.1mm inner diameter and 5.5mm outer diameter. The length of the male plug should be between 10mm and 14mm. It is recommended to use a power supply capable of providing at least 2A of current (e.g. 2A x 5V).

Once the board is powered you would see boot information printed over the serial terminal software. (!) If you encounter boot problems make sure the board is properly recognized and listed at the start of the boot. For boards with improperly configured EEPROM further configuration is required, refer to chapter “U-boot scripts”.

A successful boot concludes with a similar screen:



```
COM59 - PuTTY
[ OK ] Started Realtek H5 bluetooth support.
Starting Avahi mDNS/DNS-SD Stack...
Starting Modem Manager...
[ OK ] Started Regular background program processing daemon.
Starting Accounts Service...
[ OK ] Reached target System Time Synchronized.
[ OK ] Started Daily apt download activities.
[ OK ] Started Daily apt upgrade and clean activities.
[ OK ] Started Discard unused blocks once a week.
[ OK ] Reached target Timers.
[ OK ] Started System Logging Service.
[ OK ] Started Raise network interfaces.
[ OK ] Started Resets System Activity Data Collector.
[ OK ] Started Restore /etc/resolv.conf if before the ppp link was shut down.
[ OK ] Started Initialize hardware monitoring sensors.
[ OK ] Started LCD-OLinuxino RTP calibration.
[ OK ] Started resolvconf-pull-resolved.service.
[ OK ] Started Save/Restore Sound Card State.
[ OK ] Started LSB: Speech Dispatcher.
[ OK ] Started LSB: automatic crash report generation.

Ubuntu 18.04 LTS a20-olinuxino ttyS0
a20-olinuxino login:
```

Illustration 6: Reaching the login prompt

The default username is **root** and the default password is **olimex**. Once you login it is highly recommended to change the default password using the **~#passwd** command.

(Optional) 2.7. Update the image

If you have access to the internet you should update the image. This ensures that the images is the latest and greatest. Execute:

~#apt update

after it completes run:

~#apt upgrade

Agree to upgrade the image.

In some cases you might also need to copy the u-boot with:

~#u-boot-install

(Optional) 2.8. Explore the configuration scripts

There are a number of kernel scripts worth exploring:

~#olinuxino-display – allows setting different video output medium;

~#olinuxino-overlay – provides easy way to enable or disable different hardware assignments – different I2C, SPI, UART, CAN, PWM, etc;

~#olinuxino-sd-to-emmc – allows booting from eMMC; transfers the Linux file system from the card to the eMMC memory (if your board has one); boot is placed in SPI flash memory;

~#olinuxino-sd-to-sata – allows booting from the SATA disk; transfers the Linux file system from the card to external SATA disk; boot is placed in SPI flash memory;

~#olinuxino-reset – resets the stored configuration in the Linux image, this is useful if you have multiple boards and only one SD card.

2.9. Turning off the board

It is recommended to perform software shut down first. Use commands like “poweroff” and “shutdown now”. Removing the external power supply before performing a software shutdown might lead to software corruption on your SD card. This might require re-writing the Linux image and losing your progress.

If you expect power glitches or sudden power outages consider using a 3.7V Li-Po battery as a backup.

3. About the Olimage Linux image

Olimage Linux image allows Olimex boards to work at their full potential. The image is based on mainline u-boot and mainline kernel. We apply certain patches to the original sources to allow the specific hardware of each board to work properly. These patches do not break the open source nature of the projects underneath. All sources and patches are published at our GitHub pages so anyone can access them and study them. We also try to provide build scripts and instructions on how anyone can build the Olimage images.

There is a single repository for Olimage but each different chip architecture has its own build. Some more details and reasons to make this image can be found online in the original announcement [here](#):

[Olimage Wordpress announcement](#)

3.1. Supported products

The Olimage Linux images can be used with OLIMEX-made boards with Allwinner A10, A13, A20, A64, and STM32MP1 series processors. All variants of boards with these chips should work fine using the Olimage Linux, however some board features might not be supported. **The only exceptions are the A13-SOM-256, A13-OLinuXino-MICRO, and A33-OLinuXino boards which are currently not supported at all (but we still provide non-Olimage Linux images for them).**

(!) NAND flash memories are NOT supported in Olimage nor any mainline releases! If you need NAND support, please use the Olimex “legacy” images. These are older 3.4.x sunxi-based Linux images. Alternatively, switch to a board variant that uses eMMC flash memory instead of NAND flash memory – eMMC is well-supported within mainline.

All Olimex-made displays and touchscreens are supported in the Olimage! However, some older displays might not work out-of-the-box and would require software configuration via another debug medium. Some other boards like [LIME2-SHIELD](#) are also supported.

3.2. What works and what doesn't work in Olimage

Almost everything should work! There are two features currently missing in mainline kernel which are also missing in the Olimage image – NAND flash memory support and full hardware video acceleration. If you need one of those two features consider using the older kernel 3.4.x sunxi-based images.

If you intend to use your board for heavy video decoding, and you are not satisfied with the performance of the Olimage, you can try using the older 3.4.x kernel sunxi-based Linux images which contain such acceleration enabled – **(!) Sunxi-based images contain some binary blobs, which break the open-software nature of the images, double check if that is ok for your project! Sunxi-based images use older kernel (3.4.x) and also might have some issues with the eMMC support.**

If you have A20-OLinuXino-MICRO or A20-OLinuXino-LIME2 and you want a media center you can also try the LibreELEC KODI images available at [this location](#).

4. Olimage ready images

The recommended images can be found and downloaded for free and without sign up at the following location:

<https://images.olimex.com/>

There is a “*release*” folder and inside there are folders for each chip supported. Navigate inside the folder with the chip of your board.

For example, if you have any variant of A20-OLinuXino-LIME2, then navigate into the “a20” sub-folder.

Inside this folder one should find a number of “7z” archives and “list” files. The “7z” archives contain different images that are suitable for the A20 boards. The “list” files contain text detailing what packages are present in the image. The name of each release describes what you can expect from it. The major difference is that there are images with more packages and extras enabled called “base” and images with less things enabled called “minimal”. **Currently we provide only Debian-based Olimage images.**

If you don’t know what to pick, **it is recommended to start with the latest base build of Debian.**

Find how you can rebuild the images and where you can find links to the sources in chapter “5 Build information and sources”.

5. Build information and sources

In order to rebuild the images you need to run a build script. If you wish to tinker or study the image, start by exploring the build scripts here:

<https://github.com/OLIMEX/olimage/blob/master/build.sh>

For example to build the Debian Bullseye Base image for A20-OLinuxino-LIME2 you would need to run:

```
~#bash run.sh -v image A20-OLinuxino bullseye base A20-OLinuxino-bullseye-base-$(date +%Y%m%d-%H%M%S).img
```

Warning! If you run only build.sh without additional parameters all possible images would be built.

[Detailed instructions on building DTS can be found in chapter 11 near the end of this document.](#)

[Detailed step-by-step instructions on re-building and modifying kernel can be found in chapter 11 near the end of this document.](#)

The u-boot here:

<https://github.com/OLIMEX/u-boot-olinuxino>

The kernel here:

<https://github.com/OLIMEX/linux-olimex>

The Olimex overlays can be found here:

<https://github.com/OLIMEX/olinuxino-overlays/>

Few tools made by Olimex here:

<https://github.com/OLIMEX/olinuxino-tools>

6. Accessing a board running Olimage

Couple of items are always needed:

- a power supply unit for the board

This depends on the specific board – either 5V or 12V. Notice that powering a 5V-only board with 12V power supply would damage it permanently.

(!) Most Olimex-made Linux boards CAN NOT be powered from USB cable even when using externally powered USB.

- input and output peripherals (keyboard and mouse, monitor or display, etc)

Depending on the debug method chosen the required peripherals would vary.

- a debug interface

A computer with **USB-serial converter** cable to attached to the UART debug pins of the target board and serial terminal software **is HIGHLY recommended**. Yes, the boards can be accessed via the HDMI connector and HDMI monitor, via the Ethernet connector and SSH, via the USB connector and serial emulator so on, but these mediums might not allow full control over the board. Some of these mediums do not allow access to the board before kernel starts which might be a problem! Furthermore, the settings for these mediums can easily be corrupted leading to unavailability access the board. It is harder to corrupt the direct connection to the UART.

The board can be accessed in a number of ways:

- (1) with personal computer via USB-serial adapter;
- (2) with monitor via HDMI cable;

The methods below do not allow interrupting u-boot, since they become available at a later stages of the boot:

- (3) with personal computer but using USB cable (creating virtual serial port);
- (4) with LCD connector and display;
- (5) with personal computer over the network using the Ethernet connector (SSH).

Each of the connections would be described below.

6.1. USB-serial adapter

This is the recommended way of accessing the boards by beginners. The USB-serial adapter connection is described in the quickstart chapter of this guide. Refer to chapter “2. Quickstart”.

6.2. HDMI cable

This debug method is available only for Olimex Linux boards with on-board HDMI connector.

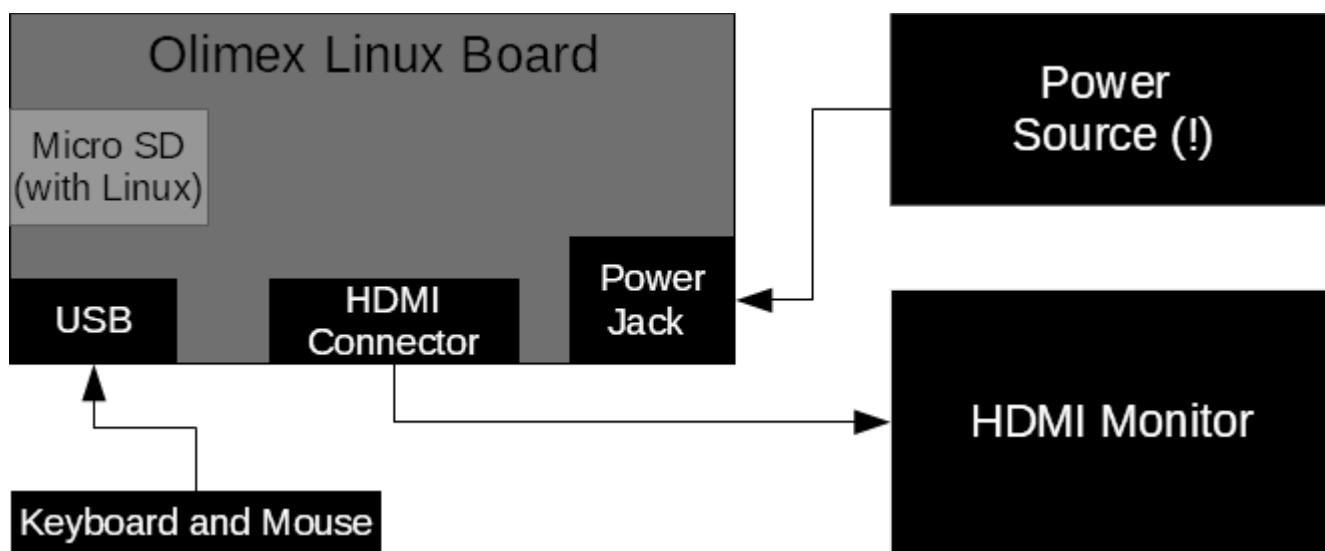


Illustration 7: hardware setup with HDMI monitor

1) Insert the card with the Olimage image. Connect the Olimex board to the monitor using an HDMI cable. You would need to connect at least keyboard and mouse to be able to utilize the GUI that would appear on the monitor after power up. If the board has only one USB port available, you might need to use external USB hub to split it. When ready apply power to the board.

Make sure to double check voltage needed by your board – either 5V or 12V! If you provide 12V of voltage to a 5V-only board, you would damage it.

Wait a few moments after power up, if there is no image shown on the HDMI, reset the board and again wait a few minutes. Make sure your HDMI monitor’s input is properly set.

Login with user **root** and password **olimex**.

Important! If you need HDMI hot-plug, and it is not working properly, you might need to disable the LCD video output from the overlays. Execute “`olinuxino-display`” script and select “DISABLE” option. Reboot the board. The hot-plugging should now work.

If there is no image on the HDMI – first double check the there is a chance that either the board is not properly recognized in u-boot (if your board lacks EEPROM memory or if the EEPROM memory was erased or got corrupted) or the board’s video output is not properly set for HDMI operation – if that is the case use another method of accessing the board to change the video output medium.

6.3. USB cable

The method is quite similar to the USB-serial converter method, but can be accomplished without USB-serial adapter – just a fitting USB cable.

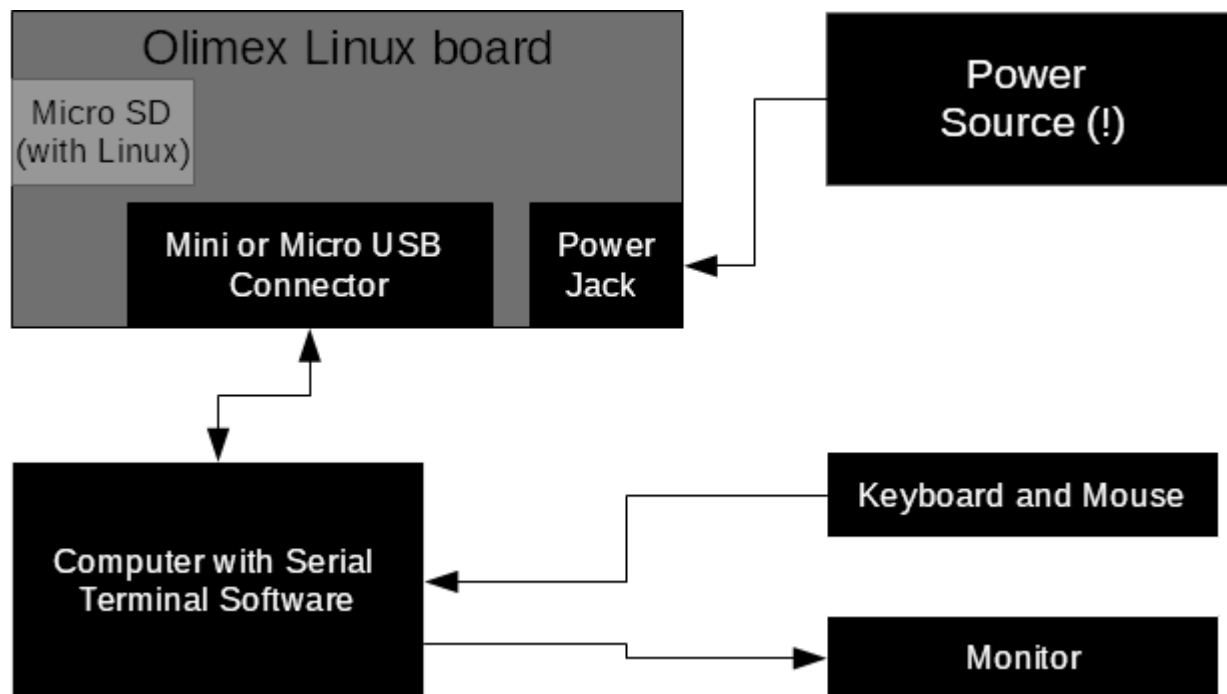


Illustration 8: USB cable setup

- (1) Insert the micro SD card with Olimage Linux in your board;
- (2) Place the USB cable between the board and the computer; depending on the Olimex board, the on-board connector with virtual serial port enabled is either mini USB mini or micro USB;
- (3) Apply the power source to power jack of the board;
- (4) Wait for the Linux to boot and you will see new serial device appear in your computer, it should be called **"PI USB to Serial"** and your operating system will assign it a serial port;
- (5) Open a terminal software on the given serial port;
- (6) You would be asked to login – use the default username **root** and the default password **olimex** – after a successful login make sure to change your default login details.

There are two important things to notice when using USB cable for accessing the board:

- **you can't power the board sufficiently from the USB, even if you use externally powered USB;**
- **you still need either external power supply or Li-Po battery;**
- the virtual serial software that enables this debug method loads after u-boot, you can't access u-boot or the custom-made u-boot configuration scripts and that might be a problem in certain cases.

In my case, I use A20-OLinXino-LIME2-e16Gs16M and Windows 10. I insert the micro SD card prepared with latest Olimage image, connect the USB cable between the board and the PC, then I apply the 5V power supply to the board's barrel jack. After a few moments, the board shows as "PI USB to Serial" device in "Windows Device Manager" in the "Ports (COM & LPT)" section. It gets assigned

COM4, as show in the picture on next page:

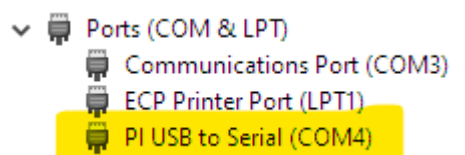


Illustration 9: USB port was assigned COM4 by Windows

Then I start my serial terminal software PuTTY on COM4 with default settings:

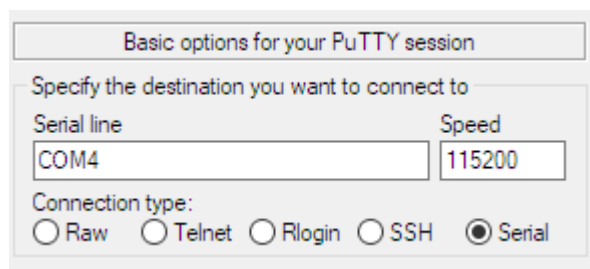


Illustration 10: PuTTY settings

I login with user root and password olimex. Then I change my password with passwd.

6.4. LCD cable

Connection is similar to the HDMI connection but user connects an LCD display via the LCD connector of the board (instead of HDMI monitor on the HDMI connector). Olimage has support for Olimex-made LCDs and touchscreens. It is recommended to use one of the products made by Olimex even if only for research and development purposes to avoid software woes in the beginning.

- 1) the software required loads after u-boot, you can't access u-boot scripts and that might be a limitation in certain cases;
- 2) you can't power the board sufficiently from the USB, you still need external power supply, notice that bigger LCD displays might consume considerable amount of current (like 1A @ 5V or even more), in some cases you might consider either a better power supply or powering the display separately.

This method allows access to the GUI of the Linux image.

If you are using an Olimex-made display and it doesn't get recognized run the display configuration script. This requires another method of accessing the board. The script to enable different output is *olinuxino-display*. Notice that if you don't use LCD it is best idea to keep that video output disabled from the *olinuxino-display* script. Some options for the LCD screens might also be available in *olinuxino-overlay* script, so if you have issues with LCD make sure to also explore the overlay one.

If the touchscreen of the display doesn't work – make sure the resistor matrices are properly set – some boards like the A20 boards are set by default for resistive touchscreen operation by default, if your touch is capacitive, you would need to change the position of a resistor matrix. Notice that **A64** boards support **only capacitive touchscreens**.

6.5. Ethernet cable

Boards with Ethernet connector can be accessed via the network over SSH. The board starts SSH server upon power up. The SSH server requires DHCP enabled in your network. This means that the IP address of the board is given by your network equipment (it is not set static by default). You need to first check your network what IP address was given to the board to access it.

Insert the Olimage SD card into the board, attach Ethernet cable, power the board. Wait a few moments and check what IP address the board was given (via the network equipment, or the software related to it). Finally open SSH connection on the IP address of the board on port 22 via suitable software. Login to access the board.

For example, the A20-OLinuXino-LIME2 board that I use here was given IP address of 192.168.0.112 – so using my personal computer that is in the same network I start PuTTY and establish SSH over address 192.168.0.112 using port 22:

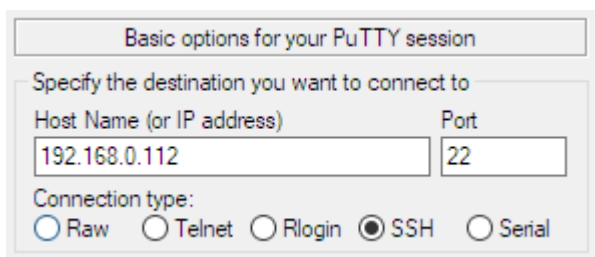


Illustration 11: SSH using PuTTY

Once the connection to the board is established, you would be prompted for username and password.

Important! You should login as user **olimex** with password **olimex** over SSH. For security reasons you can't login as user **root** over the network.

7. The boot process

The boot process is different for different boards and boot mediums. However, most common way when using SD card is pretty simple – the board boots from the SD card that stores everything except some software configuration stored on the EEPROM which gets loaded later during the boot.

If the board has to boot from eMMC memory, instead of the card, the boot process is usually a bit more convoluted and requires initial boot from the SPI flash memory. This is mainly due to lack of support for the newest eMMC chips in the Allwinner bootloader.

7.1 U-boot tools

You can interrupt the Olimage boot in order to stay in u-boot. This allows for some early board configuration (before even reaching the kernel).

The u-boot tools allow for formatting the EEPROM of your board, in case it was blank or if it contains improper data.

In order to interrupt u-boot press a key repeatedly even before seeing the required message:

Hit any key to stop autoboot: 0

You would be now allowed to enter commands after “=>”.

If you miss the timing, simply reboot or power cycle the board and try again.

(!) Not all debug mediums allow for interrupting the boot at such an early stage. That is because some of the debug mediums depend on tools loaded at a later stage of the boot process. For best results use USB-serial debug cable.

To see list of available commands, type:

olinuxino

then to see the current config settings, type:

olinuxino config info

to see list of boards and their corresponding ID codes type:

olinuxino config list

The ID for A20-SOM204 would be either 8991 or 8958 or 10257 (depending on the variant). T2-SOM204 boards have different ID number.

To configure a MAC address you would also need to identify the board's revision (printed on the board itself). The rest two parameters needed – serial number and MAC can be random (e.g. the ones you want). So now use command (replace the ID with the desired one, same for the serial

number and the MAC address!):

olinuxino config write 8958 c 11112222 aa:bb:cc:dd:ee:ff

Remember to save the configuration with:

saveenv

Reboot the board with command:

reset

The EEPROM structure is described in the latest document here:

<https://github.com/OLIMEX/OLINUXINO/tree/master/SOFTWARE/A20/A20-EEPROM-contents>

8. First steps with Olimage

The default root is:

root

The default password is:

olimex

After a first boot it is recommended to perform a software reboot. This ensures any first time start-up configuration is applied. In order to restart the board type:

reboot

Also make sure to change the default password with:

passwd

8.1. Test the Ethernet

By default the board requires DHCP in your network. Before doing anything else connect the board to your network then perform a restart! Check if the network is available after the board restart. The simplest thing is to try to ping something either in your internal or external network. I usually ping either the gateway computer in my network or www.google.com – type:

ping www.google.com

If you can't connect, identify which is the ethX name given to the Ethernet interface with:

ifconfig -a

then run the dhclient on the ethX interface:

dhclient ethX

When you have Internet connection set and enabled make sure to update and upgrade the image:

apt update
apt upgrade

In some cases you might also need to copy the u-boot with:

~#u-boot-install

reboot

8.2. Test the scripts

There are few Olimex-made scripts that might alter the behavior of certain parts of your board. Use with caution if unsure. Setting wrong values with the help of these scripts might cause boot issues and you will need to re-write the SD card with fresh Olimage.

~#olinuxino-display – allows setting different video output; if you don't have an LCD it is best idea to use the "DISABLE" option;

~#olinuxino-overlay – provides easy way to enable or disable different hardware assignments – different I2C, SPI, UART, CAN, PWM, etc; can contain some options of displays;

~# olinuxino-sd-to-emmc – allows booting from eMMC; transfers the Linux file system from the card to the eMMC memory (if your board has one); boot is placed in SPI flash memory;

~# olinuxino-sd-to-sata – allows booting from the SATA disk; transfers the Linux file system from the card to external SATA disk; boot is placed in SPI flash memory.

~# olinuxino-reset – resets the stored configuration in the Linux image, this is useful if you have multiple boards and only one SD card

8.3. Main chip and power management chip sensor support

Some features of the processor and power management chip might be supported under Linux.

You can check power-related readings under /sys/class/power_supply/ and its sub-folders.

For example, the contents of /sys/class/power_supply/axp20x-ac/voltage_now will show the current voltage applied to the board:

~#cat /sys/class/power_supply/axp20x-ac/voltage_now

This will return something like:

5122000

This is the current voltage is milivolts x 1000. So this is 5122mV or 5.122V.

You can also check other things like the temperature of the main chip with:

```
~#cat /sys/class/thermal/thermal_zone0/temp
```

This will return the temperature in degrees Celsius x 1000. To receive the correct temperature divide by 1000.

The power supply management is very useful for getting info or editing the settings of the battery too. You can use it to track voltage available in the Li-Po battery or set maximum current to be used to charge the battery. For example you can check the current charge maximum with:

```
~#cat /sys/class/power_supply/axp20x-battery/constant_charge_current_max
```

and you can set value suitable for your specific battery with:

```
~#echo 1000000 > /sys/class/power_supply/axp20x-battery/constant_charge_current_max
```

8.4. How to compile C files

First install a C compiler (if for some reason the C compiler is missing). This requires Internet connection and super user privileges:

```
$ sudo su
```

type your root username and password then:

```
# sudo apt install gcc
```

then you can compile your c code with:

```
# gcc test.c -o test
```

where test.c is the file with the c code, then give privileges for the program to be executed:

```
# chmod +777 test
```

then run it with

```
# ./test
```

8.5. How to install Python and packages

First install python3. This requires Internet connection and super user privileges:

```
$ sudo su
```

type your root username and password then:

```
# apt install python3
```

```
# apt install python3-venv python3-pip
```

```
# python3 -m pip install --upgrade pip setuptools wheel
```

Now you can install a specific python project for your specific goal with:

```
# pip3 install projectname
```

Browse projects at: <https://pypi.org/>

Some examples, for GPIO try: <https://pypi.org/project/gpio/>

For I2C try smbus2 alternative: <https://pypi.org/project/smbus2/>

For SPI generic spidev: <https://pypi.org/project/spidev/>

After that you create an empty projectname.py file, place code inside, save it and finally execute it with:

```
# python3 projectname.py
```

Olimex used to maintain custom-made pypi packages for Allwinner boards. We no longer update those but you can still use them as basis. They are not guaranteed to work. Notice! These outdated Python packages are not available for every specific Olimex Linux board. There is support for boards with Allwinner chip – A10, A13, A20, and A64.

(!) The customized A10, A13, A20, and A64 Python packages are not actively maintained, there is a good chance they might not work properly. Python packages are provided “AS IS”. Consider using alternative generic I2C, SPI, GPIO packages that should work well with the Olimex boards. Currently there is no package for STMP1 boards.

For example, I use A20-OLinuXino-LIME2 and I go:

```
# pip3 install pyA20Lime2
```

Now you have all the essential tools to access the interfaces of the boards. Either directly using sysfs or through C or python code. GPIO, SPI, I2C and serial ports of A20-OLinuXino-LIME2 via Python package.

8.6. How to toggle a GPIO pin via sysfs

General-purpose-input-outputs (GPIOs) are pins that allow connecting to other electronic devices. These pins can be set a digital inputs or outputs. By default none of them is enabled. You can see the available GPIOs with the command:

ls /sys/class/gpio

In order to enable a GPIO you have to export it and to assign it as input or output.

How do you know the correspondence between physical location on the board and GPIO number in Linux? We first locate the pin on the board, trace it back to the main chip and calculate it according to the formula:

$$\text{gpioNumberLinux} = (\text{portLetterChip position in the alphabet} - 1) * \text{bit-width-max} + \text{pinNumberChip}$$

bit-width-max is usually 32, but it can be 16 (for Olimex boards with Allwinner-made main chip it is usually 32, except for boards with STM32MP where it is 16).

I first check the chip's datasheet to find out I the data width is 16 or 32 (Z). Then I open the schematic of the board and take a look at a specific pin (X) from a specific processor port (Y) - the number for that pin in Linux would be $= (Y-1) \times Z + X$.

For example, I use A20-OLinuxino-LIME2 and the chip has 32-bit data width and the green LED is located named on wire PH2/LED that wire goes to port PH6 at the A20 chip. This means that the corresponding number for that pin in Linux would be $(8-1) \times 32 + 6 = 224 + 6 = 230$. If we want to manipulate the blinking of the green LED we would use GPIO230.

Another example, if we want to attach an external LED to a free pin - I search for free pin, for example PI15. Then I attach LED between PI15 (available at connector "GPIO-2" pin #39) and GND (available on multiple connectors, usually as pin #2). We need to connect negative (cathode) lead of the LED to GND (pin #2) and the positive LED positive (anode) lead through 470 ohm resistor to PI15 (GPIO-2, pin #39). The 470 ohm resistor is necessary to limit the current which will go through the LED.

Now to turn on the LED we have to export PI15 then set it as output and finally set it to high position (1). Doing so will bring 3.3V to the PI15 pin and the power will light the LED. First we need to have superuser rights, so we start with:

sudo su

we will be asked for the password, then the prompt will start with symbo "#", we calculate and export the Linux corresponding pin:

$$\text{gpioNumberLinux} = (9-1) \times 32 + 15 = 256 + 15 = 271$$

echo 271 > /sys/class/gpio/export

now we can check if gpio303 is already available with

```
# ls /sys/class/gpio
```

and we see the gpio271 is already listed, now we have to define it as output:

```
# echo out > /sys/class/gpio/gpio271/direction
```

now GPIO271 would be defined as output and you can change its state to a high with:

```
# echo 1 > /sys/class/gpio/gpio271/value
```

And the LED will light!

To turn it if off type:

```
# echo 0 > /sys/class/gpio/gpio271/value
```

Now let's make GPIO271 an input:

```
# echo in > /sys/class/gpio/gpio271/direction
```

then PI15 to GND via a wire and check the value with:

```
# cat /sys/class/gpio/gpio271/value
```

We see it returns 0. If we disconnect the wire between PI15 and GND and, instead, connect PI15 to 3.3V (it can be found on the same connector GPIO-2 as pin #3) and execute the last cat command again we see it reads 1, which means high position.

Important! The GPIOs typically work in the 0V to 3.3V voltage range. If what you need to measure inputs higher than 3.3V – make sure to lower them down to 3.3V, else you might burn the main chip!

The sysfs approach can seem clunky at start but you can insert it in C code or Python code.

GPIO with C code:

Create a blank file and make sure to Below is an example how to drive GPIOs in C code:

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>

int main() {

char s_out[] = "out";
char s_gpio[] = "271";
char s_0[] = "0";
```

```
char s_1[] = "1";
int fd;

//export GPIO271

if((fd=open("/sys/class/gpio/export",O_WRONLY))<0) {
    printf("can't open export GPIO\r\n");
    return(1);
} else {
    if(write(fd,s_gpio,strlen(s_gpio))<0) {
        printf("can't write to export GPIO\r\n");
        return(1);
    }
    printf("GPIO271 is exported\r\n");
    if(close(fd)<0) {
        printf("can't close export GPIO\r\n");
        return(1);
    }
}

//set as output GPIO271

if((fd=open("/sys/class/gpio/gpio271/direction",O_WRONLY))<0) {
    printf("can't open GPIO direction\r\n");
    return(1);
} else {
    if(write(fd,s_out,strlen(s_out))<0) {
        printf("can't write GPIO direction\r\n");
        return(1);
    }
    printf("GPIO271 is set as output\r\n");
    if(close(fd)<0) {
        printf("can't close GPIO direction\r\n");
        return(1);
    }
}

//toggle GPIO271

if((fd=open("/sys/class/gpio/gpio271/value",O_WRONLY))<0) {
    printf("can't open GPIO value\r\n");
    return(1);
} else {
    while(1) {

        if(write(fd,s_1,strlen(s_1))<0) {
            printf("can't write GPIO value\r\n");
            return(1);
        }
    }
}
```

```

    }

    if(write(fd,s_0,strlen(s_0))<0) {
        printf("can't write GPIO value\r\n");
        return(1);
    }
};

if(close(fd)<0)
    return(1);
}

}

```

Remember that you need to compile the source file to be able to run it. First create a file with nano text editor:

nano test.c

paste the code inside and exit and save, then

gcc test.c -o program

and allow program to be executed:

chmod +777 program

finally execute it with:

#./program

GPIO with Python:

Consider using: <https://pypi.org/project/gpio/>

GPIO with Python (using pyA20Lime2 package, seems to work fine at time of updating this document):

Lets use same setup with external LED connected to PI15 and GND. We need to connect negative (cathode) lead of the LED to GND (GPIO5.pin9) and the positive LED positive (anode) lead through 470 ohm resistor to A20.PI15 (GPIO5.pin7). The 470 ohm resistor is necessary to limit the current which will flow through LED.

```

#!/usr/bin/env python
from pyA20Lime2.gpio import gpio
from pyA20Lime2.gpio import port
from pyA20Lime2.gpio import connector

gpio.init() #Initialize module. Always called first

```

```

gpio.setcfg(port.PI15, gpio.OUTPUT) #Configure LED1 as output
gpio.setcfg(port.PI15, 1)           #This is the same as above

gpio.output(port.PI15, gpio.HIGH)   #Set PI15 High (3.3V) and Light on LED
gpio.output(port.PI15, 1)           #same as above

gpio.output(port.PI15, gpio.LOW)    #Set PI15 Low (0V) Light off
gpio.output(port.PI15, 0)           #same as above

gpio.setcfg(port.PI15, gpio.INPUT)  #Configure PI15 as input
gpio.setcfg(port.PI15, 0)           #Same as above

gpio.pullup(port.PI15, 0)            #Clear pullups
gpio.pullup(port.PI15, gpio.PULLDOWN) #Enable pull-down
gpio.pullup(port.PI15, gpio.PULLUP)  #Enable pull-up

if gpio.input(port.PI15) == 1:
    # this will be executed when input is HIGH i.e. 3.3V
else:
    # this will be executed when input is LOW i.e. 0V

```

8.7. I2C

You can expand your Olimex boards with other devices using standardized interfaces like I2C (TWI). You can detect where your device is connected by using `i2cdetect` command.

For this demo I use A20-OLinuXino-LIME2, LIME2-SHIELD, and MOD-IO (for easier access to I2C pins via the UEXT connector of the SHIELD) - if you have connected expansion board like Olimex MOD-IO to UEXT connector and run `i2cdetect` you will see address 0x58 is active:

```

$ sudo i2cdetect -r -y 2
   0 1 2 3 4 5 6 7 8 9 a b c d e f
00:  -----
10:  -----
20:  -----
30:  -----
40:  -----
50:  ----- 58 -----
60:  -----
70:  -----

```

Then you can drive relays on and off:

```
$ sudo i2cset -y 2 0x58 0x10 0x01
```

this will switch on RELAY1, to switch it off:

```
$ sudo i2cset -y 2 0x58 0x10 0x00
```

if you want to read MOD-IO status you can use i2transfer:

```
$ sudo i2cttransfer -y 2 w1@0x58 0x20 r1
```

We get result:

0x00

Now connect wire from IN1 left side to GND (for instance AIN-2 connector has AGND) and right side to +3.3-12V for instance AIN-2 connector has 3.3V).

if you run again i2cttransfer command the returned value will be 0x01:

```
$ sudo i2cttransfer -y 2 w1@0x58 0x20 r1
```

We get result:

0x01

I2C in C code

I2C demo for A20-OLinuXino-LIME2, LIME2-SHIELD, and MOD-IO (the code is also available on GitHub at LIME2-SHIELD's repository):

main.c

```
#include <stdio.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include "i2c.h"

int main(int argc, char **argv)
{
    //switch on MOD-IO relay 1

    int file;
    unsigned char buffer[2], address;

    address = 0x58;
    buffer[0] = 0x10;
    buffer[1] = 0x01;
    I2C_Open(&file, address);
    I2C_Send(&file, buffer, 2);
```

```
    I2C_Close(&file);

    return 0;
}
```

i2c.c

```
#include <stdio.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>

void I2C_Open(int *file, unsigned char address)
{
    *file = (int)open("/dev/i2c-2", O_RDWR);
    if(*file < 0)
    {
        perror("Failed to open I2C");
        exit(1);
    }
    else
    {
        if(ioctl(*file, I2C_SLAVE, address) < 0)
        {
            perror("Failed to access I2C bus");
            exit(1);
        }
    }
}

void I2C_Close(int *file)
{
    close(*file);
}

void I2C_Send(int *file, char *buffer, int num)
{
    int bytes;
    bytes = write(*file, buffer, num);
    if(bytes != num)
    {
        perror("Failed to send data");
        exit(1);
    }
}
```

```
void I2C_Read(int *file, unsigned char *buffer, int num)
{
    int bytes;
    bytes = read(*file, buffer, num);
    if(bytes != num)
    {
        perror("Failed to read data");
        exit(1);
    }
}
```

i2c.h

```
#ifndef I2C_H
#define I2C_H

void I2C_Open(int *file, unsigned char address);
void I2C_Send(int *file, unsigned char *buffer, int num);
void I2C_Read(int *file, unsigned char *buffer, int num);
void I2C_Close(int *file);

#endif

makefile
CC = gcc
CFLAG = -c -Wall

all: i2c-demo

i2c-demo: i2c.o main.o
    $(CC) i2c.o main.o -o mod-io

main.o: main.c
    $(CC) $(CFLAG) main.c

i2c.o: i2c.c
    $(CC) $(CFLAG) i2c.c

clean:
    rm -rf *.o i2c-demo
```

I2C in Python

It is recommended to use the smbus2 package:

<https://pypi.org/project/smbus2/>

I2C in Python (using pyA20Lime2 package, seems to work fine at time of updating this document)

Next code switches on and off MOD-IO relays. MOD-IO should be connected to UEXT1 of the LIME2-SHIELD. The code is also available at GitHub. Create a file “mod-io-lime2.py” with:

```
touch mod-io-lime2.py
```

then open it for editing with:

```
nano mod-io-lime2.py
```

and place the following contents inside:

```
from pyA20Lime2.i2c import i2c
i2c.init("/dev/i2c-2")

i2c.open(0x58)
i2c.write([0x10, 0x07]) # switch ON all relays
i2c.close()

i2c.open(0x58)
i2c.write([0x20])        # read IN1..4 state
value = i2c.read(1)
i2c.close()
```

When done save the file (CTRL + X, Y, Enter).

Finally execute to turn on all relays:

```
python3 mod-io-lime2.py
```

Later edit the py script with this line changed `i2c.write([0x10, 0x00])` – save and execute again. This will turn the relays off.

8.8. SPI

SPI in Python

It is recommended to use the generic spidev package from here: <https://pypi.org/project/spidev/>

There are instructions on how to use spidev.

SPI in Python (Olimex-made Olimex pyA20Lime2 package DOES NOT WORK WELL, it needs fixing, the read and xfer functions end with stack smashing error).

SPI communication demo in Python (instructions with outdated pyA20Lime2 package):

```
#!/usr/bin/env python
```

```
from pyA20Lime2.spi import spi
```

```
spi.open("/dev/spidev2.0")
```

```
#Open SPI device with default settings
```

```
# mode : 0
```

```
# speed : 100000kHz
```

```
# delay : 0
```

```
# bits-per-word: 8
```

```
#Different ways to open device
```

```
spi.open("/dev/spidev2.0", mode=1)
```

```
spi.open("/dev/spidev2.0", mode=2, delay=0)
```

```
spi.open("/dev/spidev2.0", mode=3, delay=0, bits_per_word=8)
```

```
spi.open("/dev/spidev2.0", mode=0, delay=0, bits_per_word=8, speed=100000)
```

```
spi.write([0x01, 0x02]) #Write 2 bytes to slave device
```

```
spi.read(2) #Read 2 bytes from slave device (read throws error)
```

```
spi.xfer([0x01, 0x02], 2) #Write 2 byte and then read 2 bytes. (xfer throws error)
```

```
spi.close() #Close SPI bus
```

8.9. CAN

Some boards have access to CAN. In this demo we use Olimex LIME2-SHIELD on top of A20-OLinuXino-LIME2.

\$ ifconfig -a

```
can0: flags=128<NOARP> mtu 16
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 10 )
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 59
```

For the CAN interface I install can-utils tools and setup the CAN interface:

\$ sudo apt-get install can-utils

\$ ip link set can0 down

\$ ip link set can0 type can bitrate 100000 triple-sampling on loopback off

\$ ip link set can0 up

Now connect A20-CAN to the CAN network two wire interface.

To send a packet over CAN use:

cansend <can_interface> <packet>

For instance:

\$ cansend can0 5AA#10.10.10

To listen for CAN network messages you can use candump :

\$ candump can0

Now you can log your car CAN networking messages and interpret them. There is plenty of info on the web about the different CAN messages which are exchanged on car CAN bus.

CAN in C code

Kernel code documentation <https://www.kernel.org/doc/Documentation/networking/can.txt>

C code Examples from [GitHub](#)

cansend.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/socket.h>

#include <linux/can.h>
#include <linux/can/raw.h>

int main(int argc, char **argv)
{
    int s;
    struct sockaddr_can addr;
    struct ifreq ifr;
    struct can_frame frame;

    printf("CAN Sockets Demo\r\n");

    if ((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
        perror("Socket");
        return 1;
    }

    strcpy(ifr.ifr_name, "can0" );
    ioctl(s, SIOCGIFINDEX, &ifr);

    memset(&addr, 0, sizeof(addr));
    addr.can_family = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;

    if (bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("Bind");
        return 1;
    }

    frame.can_id = 0x555;
    frame.can_dlc = 5;
    sprintf(frame.data, "Hello");
```

```
        if (write(s, &frame, sizeof(struct can_frame)) != sizeof(struct
can_frame)) {
            perror("Write");
            return 1;
        }

        if (close(s) < 0) {
            perror("Close");
            return 1;
        }

        return 0;
    }
}
```

canreceive.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/socket.h>

#include <linux/can.h>
#include <linux/can/raw.h>

int main(int argc, char **argv)
{
    int s, i;
    int nbytes;
    struct sockaddr_can addr;
    struct ifreq ifr;
    struct can_frame frame;

    printf("CAN Sockets Receive Demo\r\n");

    if ((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
        perror("Socket");
        return 1;
    }

    strcpy(ifr.ifr_name, "can0" );
    ioctl(s, SIOCGIFINDEX, &ifr);

    memset(&addr, 0, sizeof(addr));
```

```
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;

if (bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("Bind");
    return 1;
}

nbytes = read(s, &frame, sizeof(struct can_frame));

if (nbytes < 0) {
    perror("Read");
    return 1;
}

printf("0x%03X [%d] ", frame.can_id, frame.can_dlc);

for (i = 0; i < frame.can_dlc; i++)
    printf("%02X ", frame.data[i]);

printf("\r\n");

if (close(s) < 0) {
    perror("Close");
    return 1;
}

return 0;
}
```

9. What other images are available

You can find older (no longer supported) Olimex images here:

<https://ftp.olimex.com/Allwinner Images/>

There are plenty of community made images, unfortunately we lack the resources to maintain and test all possible images. Feel free to use community-made and community-supported images, but expect no support from Olimex.

10. Frequently asked question

Q: I have troubles compiling the images. I get an error. Can you give me step-by-step instructions on compiling the images?

A: We have provided instructions on how to compile DTS and kernel in next chapter.

Q: I miss a package and I want to add some custom logo at start. Can you compile a custom image for me?

A: No. We lack the personnel and the will to tailor custom images for thousands of customers. It is open-source software effort and you have access to the sources, if you are facing problems with tailoring the images – maybe find a third-party professional or a company specialized in embedded Linux to tailor the images for you.

Q: Why isn't there video acceleration in Olimage Linux images?

A: Mainly because there isn't a good open source driver for the video chip. The reversed engineered one that was used in sunxi images uses binary blobs and breaks the open source spirit of the software.

Q: I have two similar hardware setups with two different Olimex boards, and both boards are supported by Olimage) – one works fine but the other – doesn't. I use the same image and the same micro SD card.

A: After the first boot had been completed with a micro SD, the same card should be used only with the same boards from the same board variant. Else you would need to reset the stored configuration by running olinuxino-reset script.

For example, if you first used the Olimage micro SD card with A20-OLinUxino-LIME2-e16s16M, you should use the same micro SD card only with other A20-OLinUxino-LIME2-e16s16M. If you decide to use it with A20-OLinUxino-MICRO or with A20-SOM204-EVB or with A20-OLinUxino-LIME2 – some problems might appear. It is recommended to either run olinuxino-reset script or re-write the card to perform first time setup.

There is auto-detection and configuration in the Olimage. On first boot the card gets recognized by the data stored in its EEPROM and optimal settings for that specific board get written in the Linux image on the micro SD card, then subsequent boots with the same card will NOT perform auto-detection and re-configuration. Either reset the original configuration data by running the olinuxino-reset script or re-write the card before using it with the second A20 board. The EEPROM contents are described in the latest document here:

<https://github.com/OLIMEX/OLINUXINO/tree/master/SOFTWARE/A20/A20-eeeprom-contents>

11. Instructions on rebuilding DTS and kernel

(!) Rebuilding the DTS and rebuilding the kernel are not recommended for beginners. The algorithm bellow has been tested multiple times here and is provided **AS IS!** Feel free to experiment and test everything but do not expect Olimex to solve why something won't build at your side.

11.1. How to build Device Tree Source (DTS) file for Olimage

1) Acquire Kernel source code:

```
#git clone https://github.com/OLIMEX/linux-olimex.git
```

2) Prepare Kernel sources

- install required packages and tools

```
#sudo apt install build-essential bc kmod flex bison cpio libncurses5-dev fakeroot  
libelf-dev libssl-dev
```

- install toolchain – either arm-linux-gnueabi- or aarch64

- for 32-bit boards (for example – A20-OLinuXino-LIME2) run:

```
#sudo apt-get install gcc-arm-linux-gnueabi- g++-arm-linux-gnueabi-
```

- for 64-bit boards (for example – A64-OLinuXino) run:

```
#sudo apt-get install gcc-aarch64-linux-gnu
```

```
#cd linux-olimex
```

- to build dtb from dts:

```
#sudo make clean  
#sudo make distclean  
#sudo make ARCH=<arch> CROSS_COMPILE=<toolchain> olinuxino_defconfig  
#sudo make ARCH=<arch> CROSS_COMPILE=<toolchain> prepare  
#sudo make ARCH=arch CROSS_COMPILE=arm-linux-gnueabi- dtbs DTC_FLAGS='-@'
```

, where:

ARCH=arm – for 32-bit kernel

or

ARCH=arm64 – for 64-bit kernel

CROSS_COMPILE=arm-linux-gnueabi- – for 32-bit kernel

or

CROSS_COMPILE=aarch64-linux-gnu- – **for 64-bit kernel for example, for 32-bit kernel:**

```
#make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- olinuxino_defconfig
```

Device-Tree sources are located at either:

/arch/arm/boot/dts – **for 32-bit ARM**

or

/arch/arm64/boot/dts/<vendor_name>/ – **for 64-bit ARM for example, for 64-bit sources check:**

/arch/arm64/boot/dts/allwinner

Copy *.dtb files

```
#scp arch/arm/boot/dts/<dtb_name>.dtb user@board
```

3) Install on board

```
#uname -r
```

the result will be similar to:

```
#x.yy.zz-olimax
```

```
#mv <board_dir>/<dtb_name>.dtb /usr/lib/linux-image-x.yy.zz-olimax/.
```

, where

<dtb_name>.dtb – **should be replaced with respective dtb**

linux-image-x.yy.zz-olimax – **should be replaced with properer folder name**

4) Rebuild FIT

run:

```
#/etc/kernel/postinst.d/uboot-fit x.yy.zz-olimax
```

```
#reboot
```

11.2. How to build Linux Kernel for Olimage

1) Get Olimage sources

```
#git clone https://github.com/OLIMEX/linux-olimex.git
```

2) Prepare for building

2.1) Install additional packages and tools required

```
#sudo apt install build-essential bc kmod flex bison cpio libncurses5-dev fakeroot  
libelf-dev libssl-dev
```

2.2) Install arm-linux-gnueabi- or aarch64 toolchain

2.2.1) For 32-bit boards (for example – A20-OLinuXino-LIME2) do:

```
#sudo apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi
```

2.2.2) For 64-bit boards (for example – A64-OLinuXino) run:

```
#sudo apt-get install gcc-aarch64-linux-gnu
```

3) Building the kernel

3.1) Configuring kernel options

```
#cd linux-olimex
```

```
#make ARCH=<arch> CROSS_COMPILE=<toolchain> olinuxino_defconfig
```

, where:

ARCH=arm – for 32-bit kernel

or

ARCH=arm64 – for 64-bit kernel

and

CROSS_COMPILE=arm-linux-gnueabi- – for 32-bit kernel

or

CROSS_COMPILE=aarch64-linux-gnu- – for 64-bit kernel

for example, for 32-bit build:

```
#make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- olinuxino_defconfig
```

3.2) Enable or disable packages and modules

Manually edit .config or use:

```
#make ARCH=<arch> CROSS_COMPILE=<toolchain> menuconfig
```

3.3) Build kernel package

```
#EXTRA_VER=$(date +%Y%m%d-%H%M%S)
```

```
#sudo make ARCH=<arch> CROSS_COMPILE=<toolchain> -j$(nproc) bindeb-pkg  
LOCALVERSION=-olimex KDEB_PKGVERSION=$(make kernelversion)-$EXTRA_VER DTC_FLAGS=-@
```

4) Install kernel packages

4.1) Copy deb packages to board

```
#scp ../*.deb user@board:
```

4.2) Install on target device

```
#ssh user@board
```

```
#user@board:~$ dpkg -i *.deb
```

```
#reboot
```

12. How to send feedback or report a problem

Send an e-mail at support@olimex.com

Make sure to include as much details about the problem as possible. The very nature of the problem, is the issue reproducible, does it occur with more than one board. Include basic information about your hardware setup like the exact name and variant of the board you are using, the way of powering, specification of power supply, any extra hardware attached to the board. Any logs or pictures showing the issue provide valuable information and might speed up the support response and solving the issue.

Software issues can also be directly reported as a new issue at GitHub repositories related to Olimage.

Please notice that Olimage is provided “AS IS” – it is a free open-source software available for download even without a purchase. It’s sources and build instructions are available to everyone. Do not expect custom features added upon request.

Olimex makes no other warranties, express or implied, and hereby disclaims all implied warranties, including any warranty of merchantability and warranty of fitness for a particular purpose.

13. Document revision

Changes to this document.

Revision	Changes	Modified Page#
A, 04.06.21	First release	All
B, 15.07.21	Added info about the support for STM32P1xx boards	1, 3, 4, 5, 11
C, 01.10.21	Added detailed DTS and kernel building instructions	40-43
D, 07.01.22	We no longer build Ubuntu images, changed Ubuntu references to Debian ones.	Multiple
E, 24.02.22	Added notice about temperature sensor Added notice about lack of Python support for STMP1 boards	Multiple
F, 20.12.22	Additional info about LCD DISABLE option, more info about overlays	Multiple
G, 23.12.22	Altered GPIO calculation to include the option for 16-bit width bus	24
H, 09.11.23	Updated links and product names	Multiple
I, 08.07.24	Clarified that Olimex-made Python packages are no longer supported, provided links to some alternatives	Multiple
F, 29.04.25	Added info about iMX8MP boards	4