Chipsee

Industrial PC

Android 11 OS on RK3568 User Manual

For RK3568 Products

Content can change at anytime, check our website for latest information of this product.
www.chipsee.com

Contents

Android 11 OS

Android 11 OS on RK3568 User Manual

This is the software manual for RK3568 Chipsee industrial PC. If you've never developed on this hardware with an Android 11 OS, this manual can get you started quickly.

Supported Chipsee PCs: all Chipsee RK3568 based industrial PCs, including but not limited to:

- CS12720-RK3568-050P
- CS10600-RK3568-070P
- CS12800-RK3568-101P
- CS10768-RK3568-121P
- CS19108-RK3568-133P
- CS10768-RK3568-150P
- CS19108-RK3568-156P
- CS12102-RK3568-170P
- CS19108-RK3568-185P
- CS12102-RK3568-190P
- CS12800-RK3568-215P
- CS19108-RK3568-236P
- CS-RK3568-BOX

When you develop software for the Chipsee industrial PC, you can open the hardware document beside this software document, to aid you in wiring your devices.

In this document, main topics are:

- How to connect to the hardware from your workstation.
- How to use the hardware resources such as RS232, RS485 and GPIO, etc.
- How to install a new operating system.

Connect From Workstation

There are 4 ways to connect to your Chipsee industrial PC from your workstation:

1. From **Serial** RS232 port
2. From **USB** Type-C port
3. From **Ethernet**
4. From **Wi-Fi**

Let’s take a look at these connecting methods one by one.

Install Android Platform Tools

If you intend **only** to use serial port RS232 to connect to Chipsee industrial PC, you can skip this part, debugging with serial port **does not require** installing Android Platform Tools.

If you want to control or test your Android 11 based Chipsee industrial PC from your workstation, such as your PC or laptop, one tool is required to be installed on your machine.

Since the Chipsee industrial PC runs an Android OS, like developing on any other Android devices (phones, tablets, TVs, etc), the tools and knowledge apply to developing on Chipsee industrial PC as well. We will need a tool called **Android Platform Tools** to be installed on your workstation. You can refer to Google Android's official developer doc to learn how to do that, here is a link to download this tool: https://developer.android.com/tools/releases/platform-tools. If the link is dead when you're reading this document, you can search for *Android Platform Tools* in your favourite search engine to find out where it has gone to. Then download and install the distribution that matches your workstation's platform (MacOS, Linux or Windows).

Instead of Android Platform Tools, you can also get the **Android Studio**, it includes a SDK Manager that helps you download the SDK tools and platforms we'll need. Android Studio is much more powerful and requires more space and time to install. If you only need the command line tool, you're fine with just the Android Platform Tools alone.

If you have successfully installed the Android Platform Tools, you can confirm that by opening a terminal window and type a command *adb* , you should see multiple lines of instructions.

```
usb:
 attach                  attach a detached USB device
 detach                  detach from a USB device to allow use by other processes
environment variables:
 $ADB_TRACE
     comma-separated list of debug info to log:
     all,adb,sockets,packets,rwx,usb,sync,sysdeps,transport,jdwp
 $ADB_VENDOR_KEYS        colon-separated list of keys (files or directories)
 $ANDROID_SERIAL         serial number to connect to (see -s)
 $ANDROID_LOG_TAGS       tags to be used by logcat (see logcat --help)
 $ADB_LOCAL_TRANSPORT_MAX_PORT max emulator scan port (default 5585, 16 emus)
 $ADB_MDNS_AUTO_CONNECT  comma-separated list of mdns services to allow auto-connect (default adb-tls-connect)

Online documentation: https://android.googlesource.com/platform/packages/modules/adb/+/refs/heads/master/docs/user/adb.1.md

➜  Documentation git:(rk3568-soft) ✗ which adb
/opt/homebrew/bin/adb
```

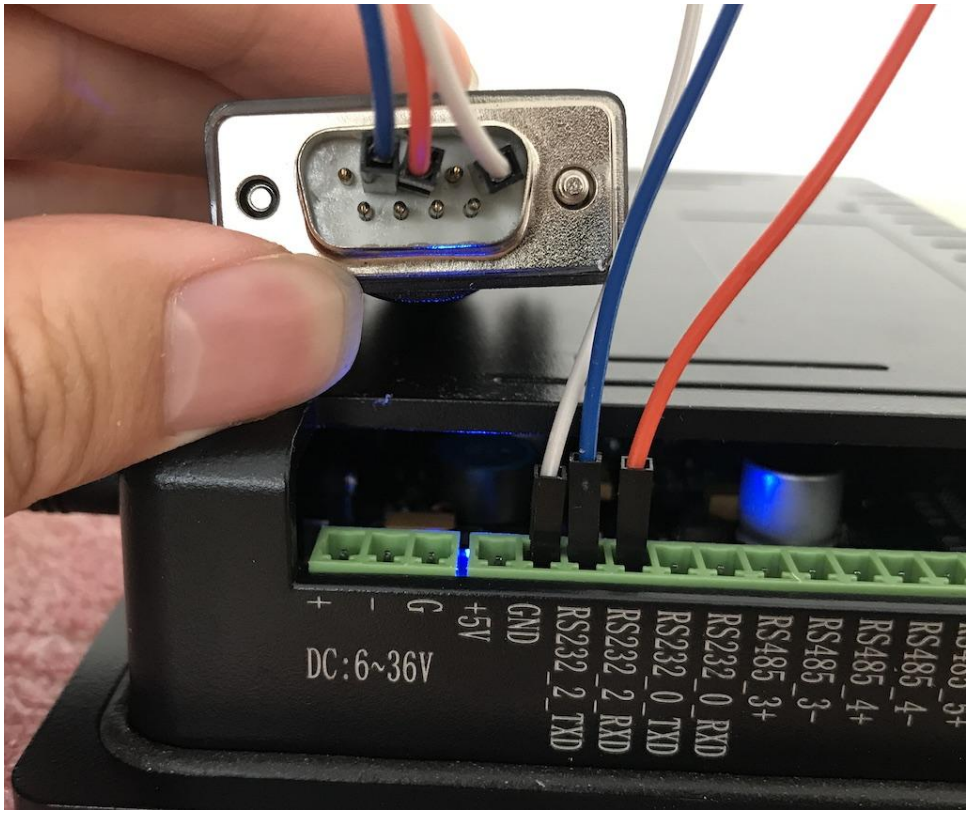
Type adb in Your Terminal to Confirm a Successful Installation of Android Platform Tools

Connect From Serial Port

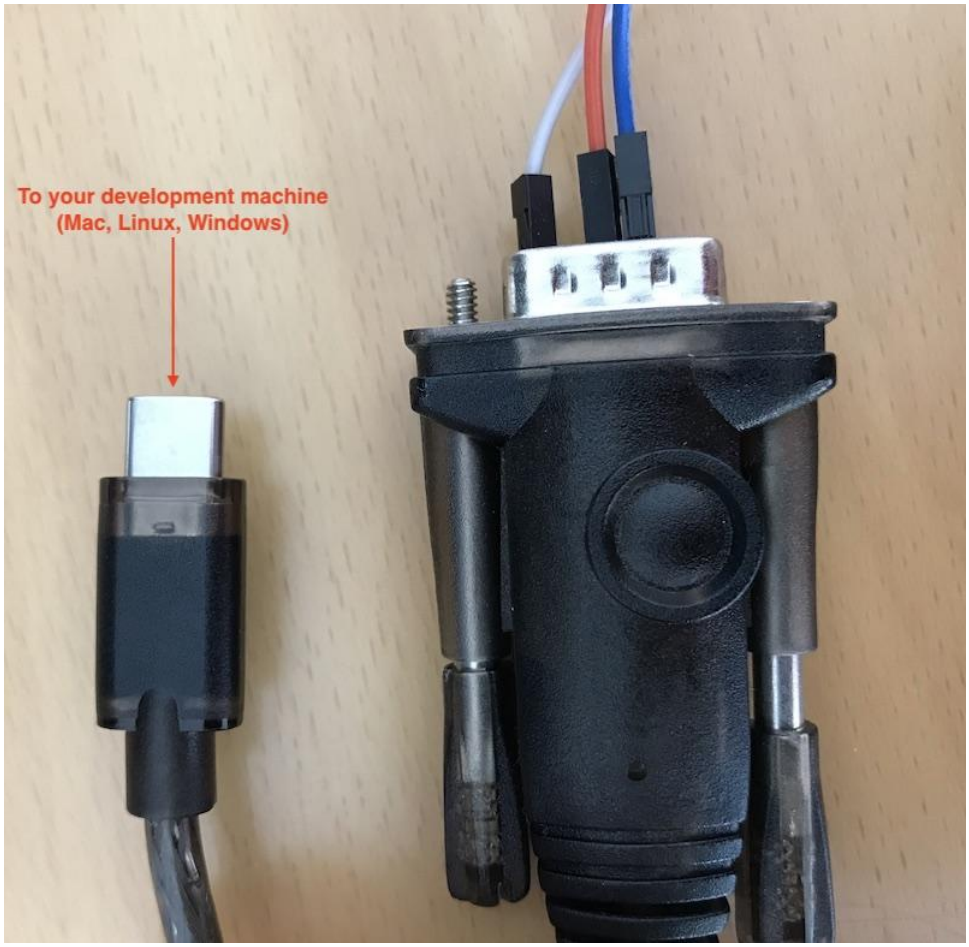
In our prebuilt Android 11 OS, the **RS232_2** serves as a serial debug port on the RK3568 products of all screen sizes. We can connect a RS232 cable from the Chipsee industrial PC to our workstation, allowing us to control the Chipsee industrial PC from our workstation.

To get started, except for the Chipsee industrial PC and your computer, you also need a USB to serial cable, we will use a USB Type-C to DB-9 cable as an example.

I will plug the USB Type-C port to a Mac (Windows and Linux work fine too), and then I would use three female to female dupont wires, to connect 3 of the DB-9 pins to my Chipsee industrial PC 's GND and RS232_2 (RX and TX) pins. Take a look at the image below, it's a 7-inch product, if your product has a different screen size, check out the hardware document to find out which pins are RS232_2 TX and RX. You should connect the DB-9 RX to Chipsee industrial PC's TX, and TX to RX, you should also connect their GNDs (the white wire).



Connect 3 Pins with Dupont Wires



Use a USB to Serial Cable to Connect the Chipsee Industrial PC with Your Workstation

For **MacOS and Linux** users, you will need a program called **screen**. It should be already installed on most MacOS and Linux distributions. Open your terminal and type *screen -v* . If the *screen* program is already installed in your computer, you should see a version number, like that in the image below:

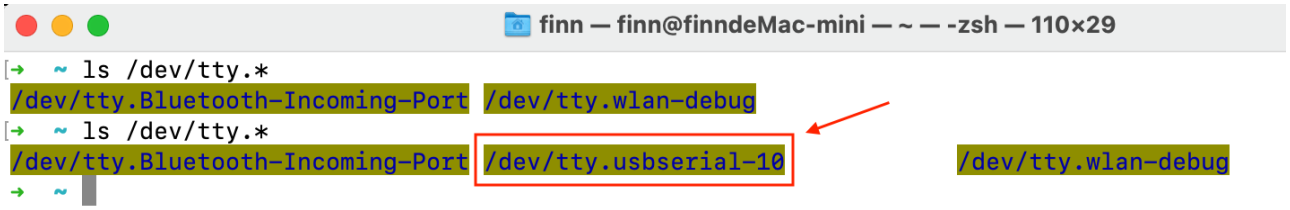
Confirm You Have the Screen Program (MacOS & Linux)

And then, let's check what device our USB to RS232 cable in our operating system is. You should type a command in your terminal to find it out, different cables may appear as different devices in your OS:

```
ls /dev/tty.*
```

You might see many *tty* devices listed in your terminal, and cannot decide which is your USB to RS232 cable. Here is a tip: you can unplug the cable first, type the command to see what is listed in the OS. Then you plug it back in, and test again, to see what has recently

appeared. The difference between the two should be your cable. In the image below, we found the */dev/tty.usbserial-10* is our USB to RS232 cable.

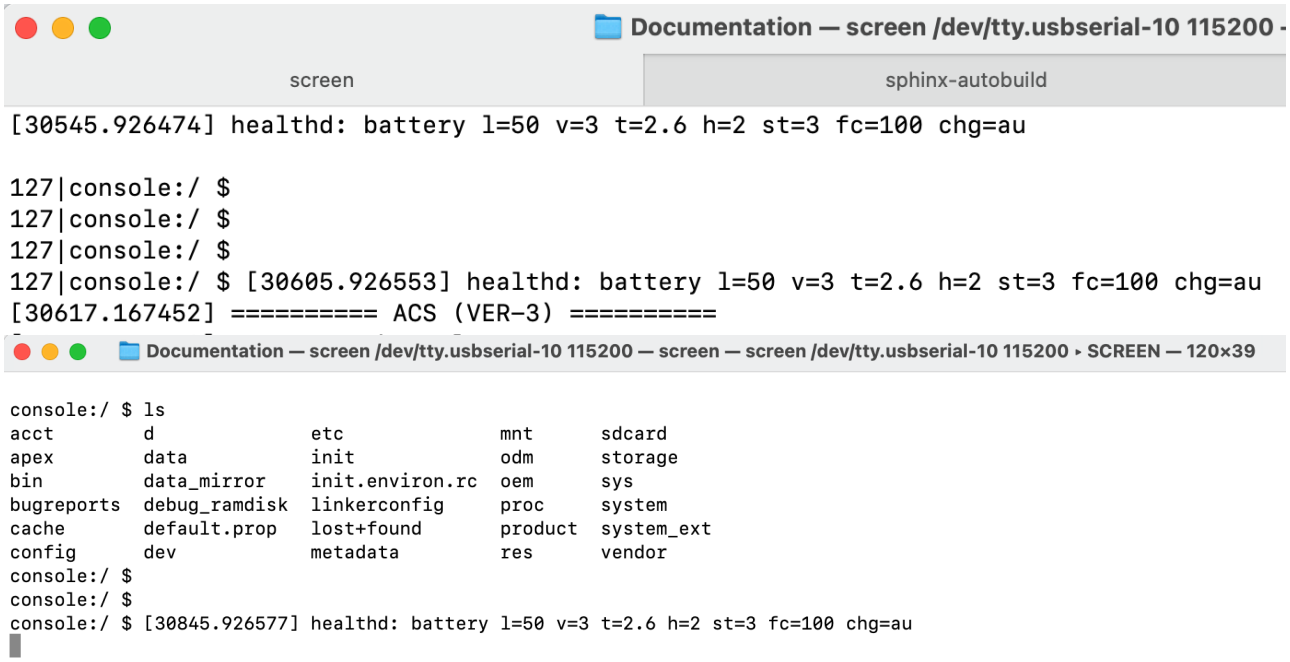


Find Out Which Device is Your USB to Serial Cable

Finally, you can put the device you found to the screen program as a parameter. To connect to your Chipsee industrial PC:

```
screen /dev/tty.usbserial-10 115200
```

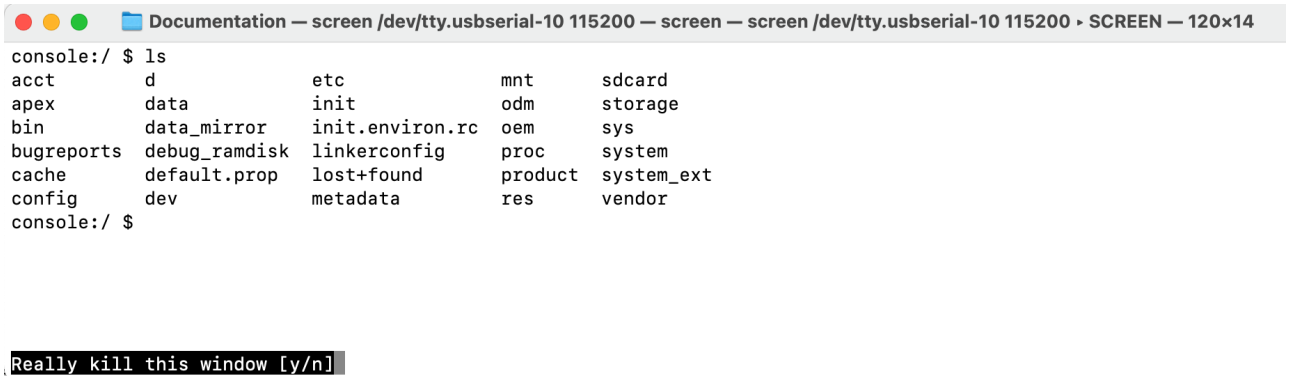
There should be a few seconds of blank screen, then you might see some health checks pop up in your terminal, you can hit Enter or type some commands to see if you can interact with your Chipsee industrial PC. In the image below, the Android Chipsee industrial PC responds to my *ls* command.



The Screen Program Connects Your Workstation to the Industrial PC's Android Console

If you want to **exit** the screen program, you can press *Ctrl+A* then press *K* .

The program will ask you if you want to kill this window, then press *y* to exit the program.

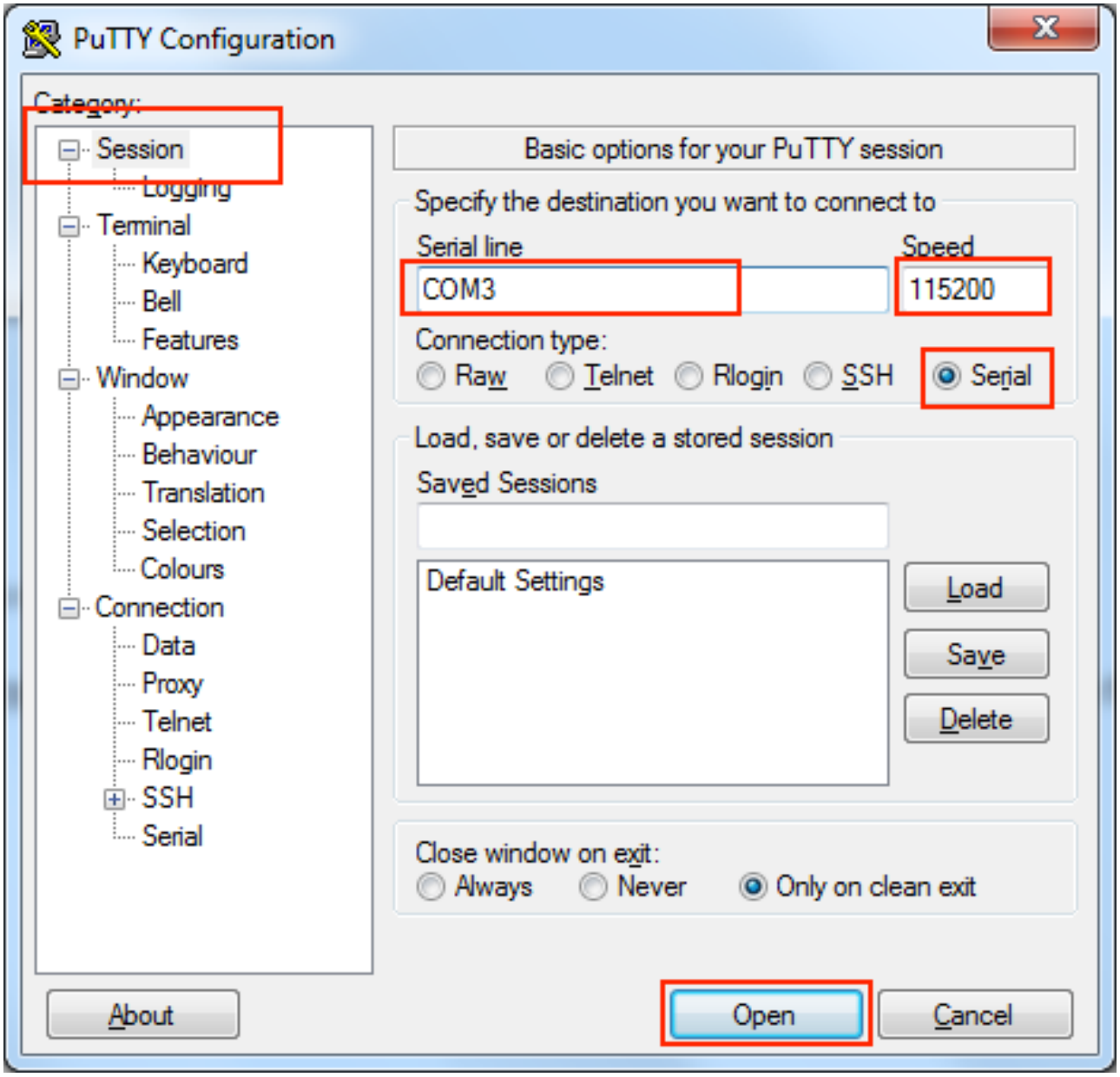


Press "Ctrl + A" Follow by a "K" to Exit the Program

For **Windows** users, you should install a program called **PuTTY**, here is a link to download the software: https://www.putty.org/. If you're comfortable with other clients rather than PuTTY, you can use those as well. There are no hard requirements when you wish to develop on a Chipsee industrial PC. This Chipsee industrial PC is an Android device in the end, that's also one benefit of embracing open technologies – no vendor lock-in, use your preferred developer tools.

With PuTTY in your belt, let's find out which COM port your USB to RS232 cable is using. One tip is to check the Microsoft Windows Device Manager. Check which COM port appears/ disappears when you plug/unplug the USB to serial cable. And that COM port device should be your USB to RS232 cable. When you have found the COM port your serial cable is using, let's say it's "your-com", or "COM3" in this example.

Open the PuTTY program, select Session, choose Serial in the radio buttons, and fill the COM port you found previously in the Microsoft Windows Device Manager, and choose 115200 as baud rate, then click Open.



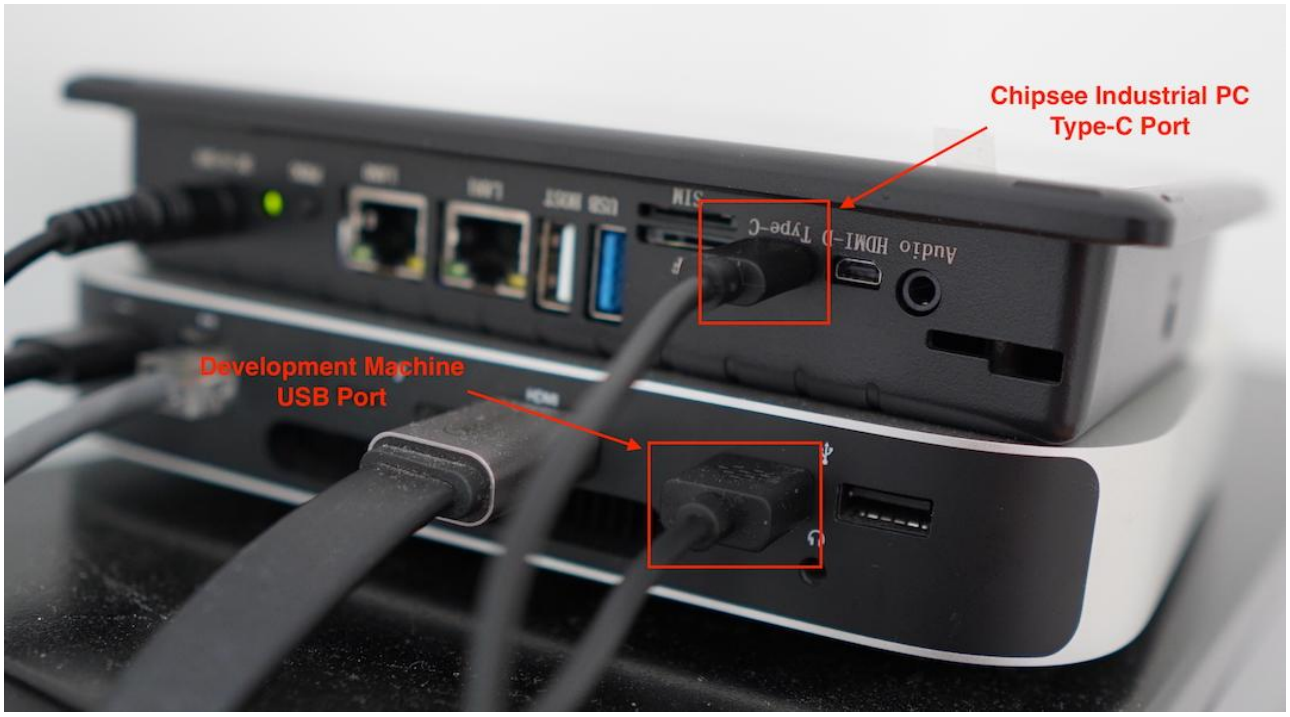
Input your-com in the Serial Line field, in our case COM3

Now you should be able to connect from your workstation to the Chipsee industrial PC through the serial port. You can try to hit Enter, or type some commands, like *ls* , to see if the Chipsee industrial PC replies to your commands.

Connect From USB Type-C Port

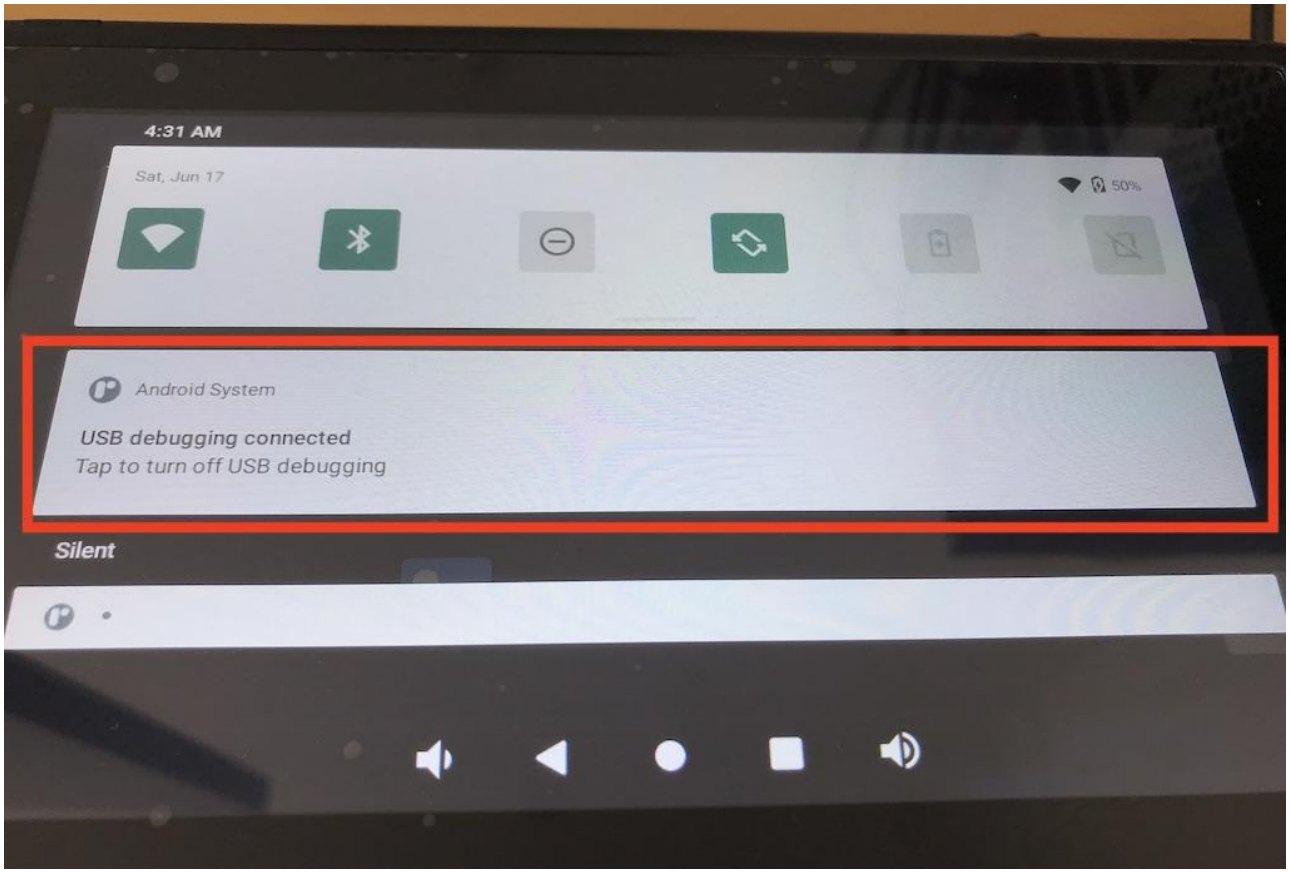
Apart from the serial port, you can also connect to the Chipsee industrial PC through its Type-C port. In this case, make sure you have Android Platform Tools installed on your development computer. Let's take a look at how we can connect through USB.

For either **Windows** or **MacOS/Linux** users, firstly you need a USB Type-C cable. It can be a Type-C to Type-C, or Type-A to Type-C, at least one end needs to be Type-C, because it needs to be plugged into the Chipsee industrial PC's Type-C port. Secondly be sure the Android Platform Tools is installed on your workstation. The image below is a 7 inch RK3568 powered Chipsee industrial PC, your product should have a Type-C port as well.



Connect Your Workstation to Chipsee Industrial PC's Type-C Port

When you plug in the USB cable properly, you can hear a sound, and see there is a pop up notification, telling you USB debugging is connected on your Chipsee industrial PC:



Plugging in the USB Cable Should Trigger a Notification

USB debugging is enabled by default, if yours isn't responding as expected, you may check whether USB debugging is set correctly on your Chipsee industrial PC, USB debugging should be enabled. Also check your USB cable to ensure your USB cable can transmit data, rather than only supports charging.

Now that the cable is connected successfully, let's connect the two with software.

For **MacOS and Linux** users, you can open your terminal, and type a command:

```
adb devices
```

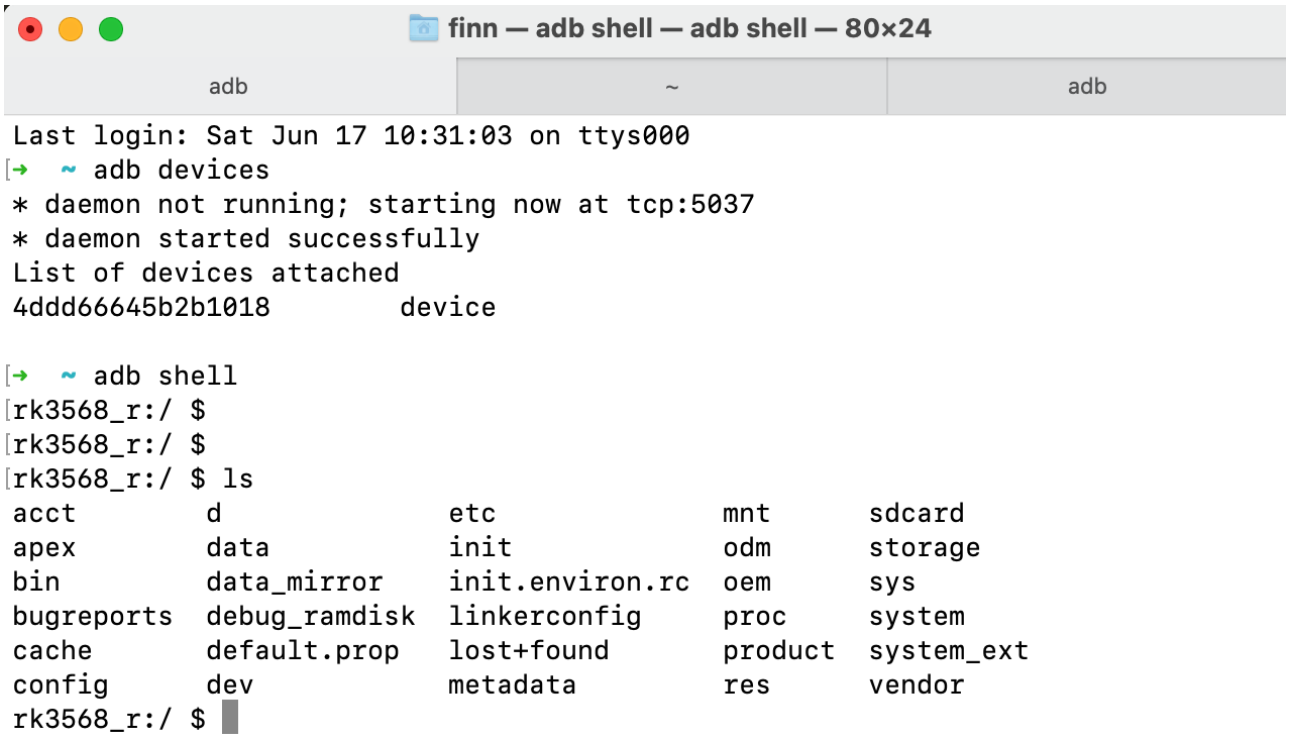
You should see your Chipsee industrial PC listed in the outputs, like in the image below:

4ddd66645b2b1018 is our Chipsee Android Device

And then you can connect to this device with another command:

```
adb shell
```

You might see a **rk3568** followed by a dollar (**$**) sign, indicating you're sending commands to the Chipsee industrial PC's Android system in the ADB shell.



Now We Are Connected From USB Type-C Port

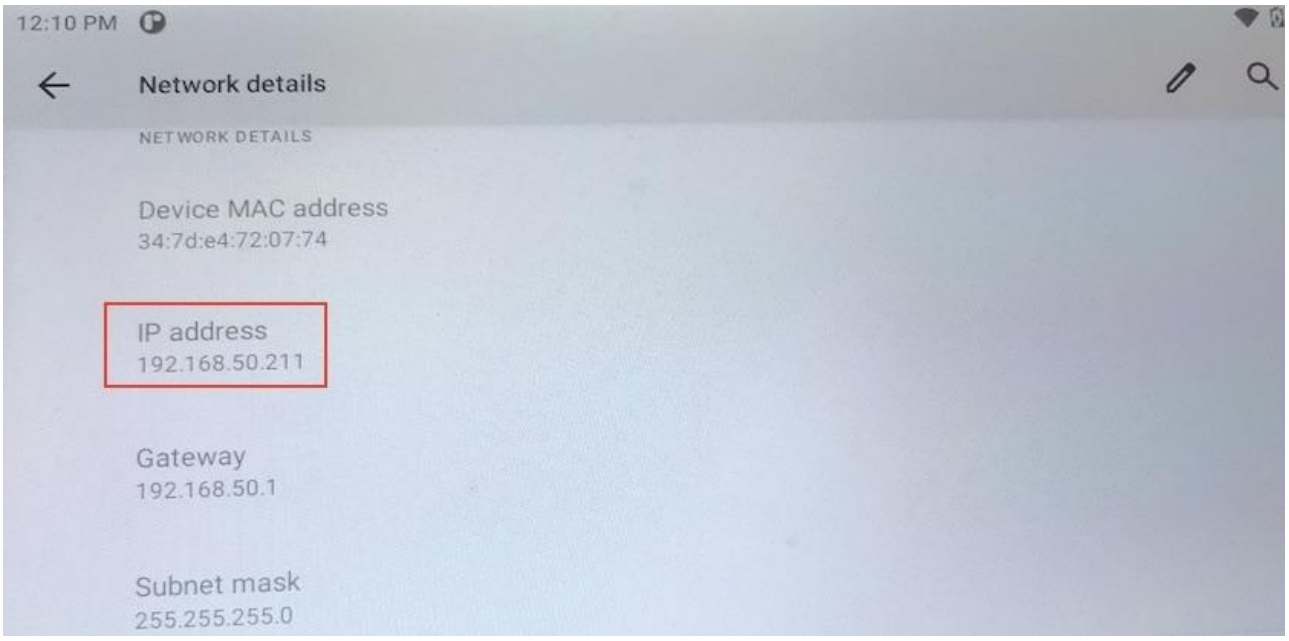
You can type *exit* or hit *Ctrl + D* to exit the ADB shell, to get back to your workstation's terminal.

Connect From Wi-Fi

Debugging from Wifi is enabled by default. Also, please keep USB debugging enabled, even though you're not connecting through the Type-C port. Make sure you have Android Platform Tools installed on your development computer.

To start debugging wirelessly, make sure your Chipsee industrial PC and your workstation are connected to the same Wifi network.

We will need your Chipsee industrial PC's IP address to start connecting, you can check your Chipsee industrial PC's IP address in **Settings - Network & internet - Wi-Fi - your WiFi ssid (Network details) - Advanced - IP address**:



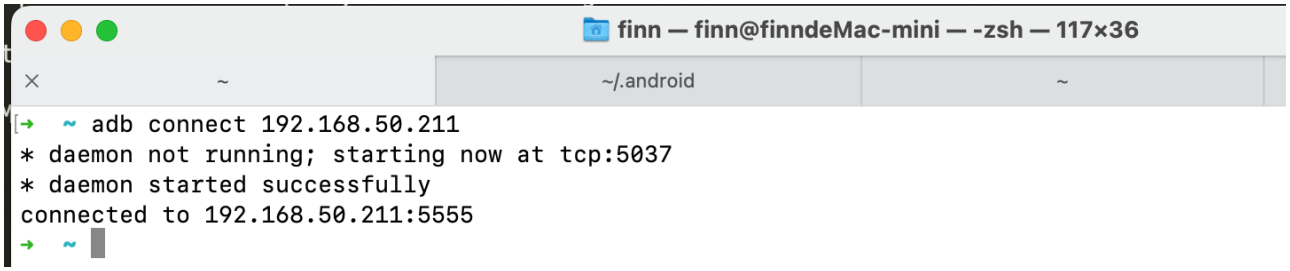
Check Your Android PC's IP Address, in Our Case: 192.168.50.211

For **MacOS and Linux** users, you can then type a command to connect to your Chipsee industrial PC wirelessly in the terminal:

```
adb connect your-ip
```

In our case, *"your-ip"* is 192.168.50.211, yours should be a different value, maybe it starts with *172* or *10.0* or *192* etc, it depends on your office network configuration.

If connected successfully, you should see a **connected to your-ip:5555** message in the terminal.

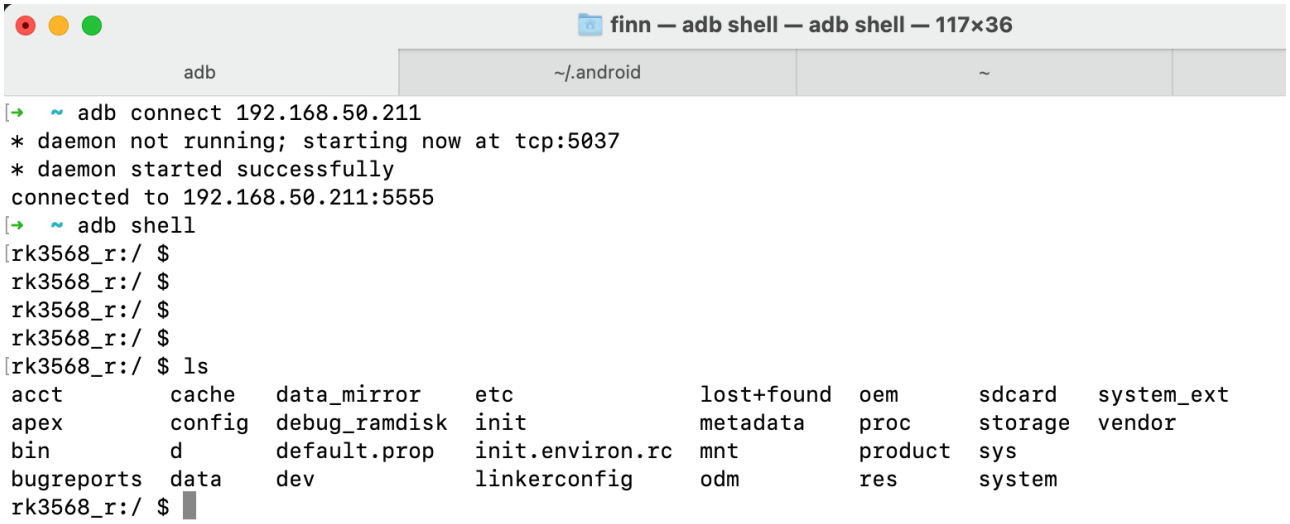


Successfully Connected to Android PC

Let's get into the adb shell by typing:

```
adb shell
```

Then you can type some commands, like *ls*, as shown in the image below. If you experience a lot of input lags, a poor Wifi connection might be one of the reasons, try moving to a place with better Wifi signal.



Use adb shell to Get into the Shell of Your Android PC

To exit the shell, you can either hit *Ctrl + D* or type *exit*.

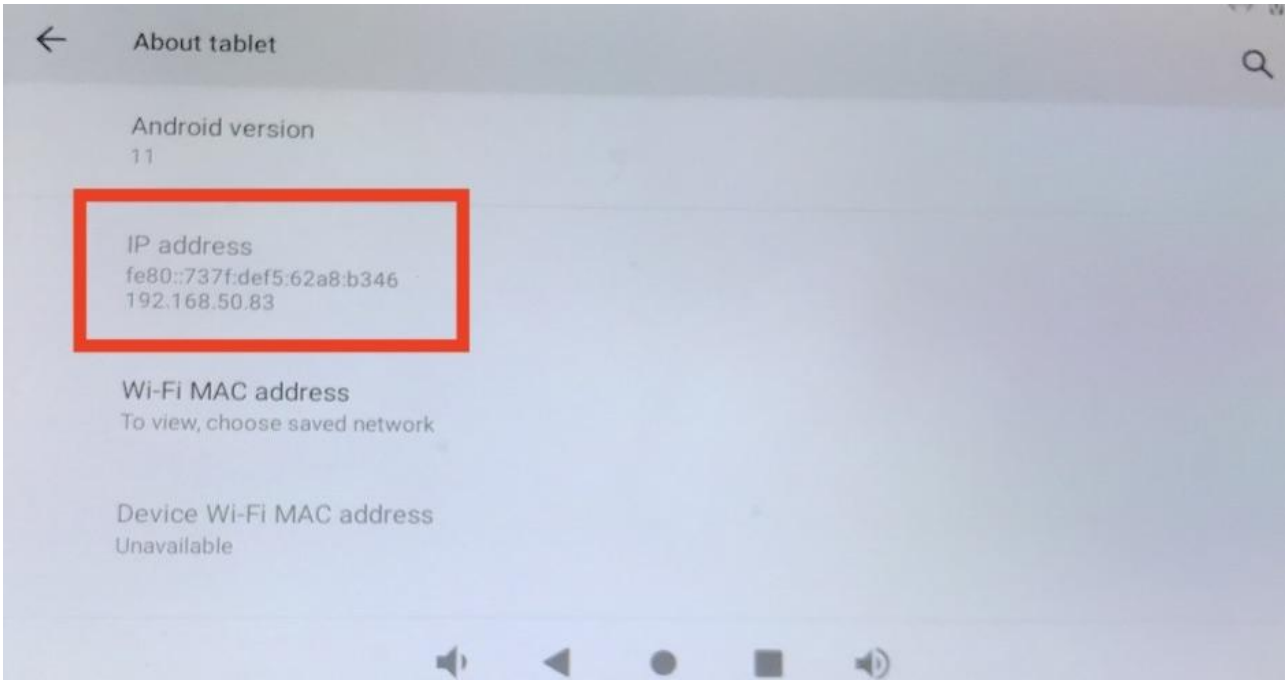
If you have other problems when debugging through Wifi, you can also search for the official Android Developer documents, the Chipsee industrial PC is running a full Android operating system, other knowledge of Android development applies to this device as well.

Connect From Ethernet

Apart from serial port, USB cable and Wifi, connecting and debugging through Ethernet is another choice for developers.

Firstly, plug in an RJ-45 Ethernet cable, to connect your Chipsee industrial PC to the same network of your workstation. The 5 inch or 10.1 inch and above products have only one RJ-45 port. The 7 inch product has two RJ-45 ports, you can use either one.

Then you can find the IP address in **Settings - About tablet - IP address**. We will need this value to connect from our workstation with **adb** later. Let's say it's *your-ip*, in our case it is 192.168.50.83, as shown in the image below. Yours might be a different value, maybe it starts with *172* or *10.0* or *192* etc, it depends on your office network configuration.

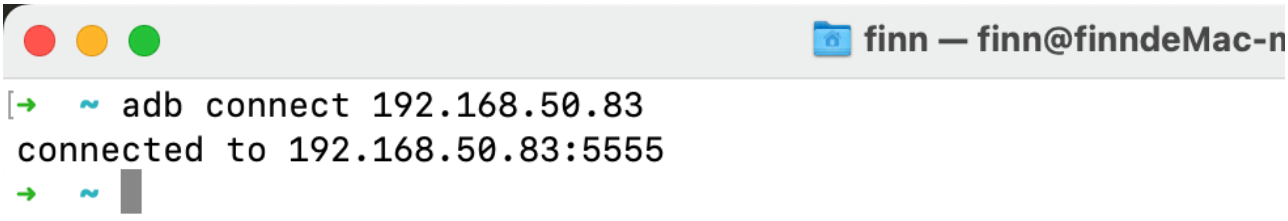


Our IP is 192.168.50.83, Yours Might be Different

For **MacOS and Linux** users, open your terminal, and type a command:

```
adb connect your-ip
```

In our case, "your-ip" is 192.168.50.83:



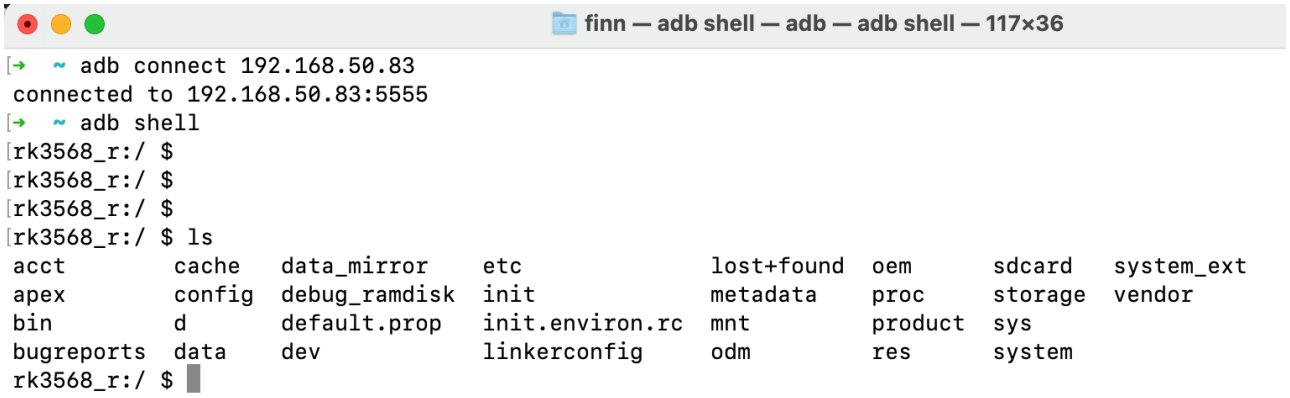
A Successful Connection

If your connection to the Chipsee industrial PC is established, you should see **connected to your-ip:5555**.

Then let's get into the adb shell by:

```
adb shell
```

You can type some commands to interact with your Chipsee industrial PC inside the shell, like *ls* , to check if you have successfully gotten into the Android shell.



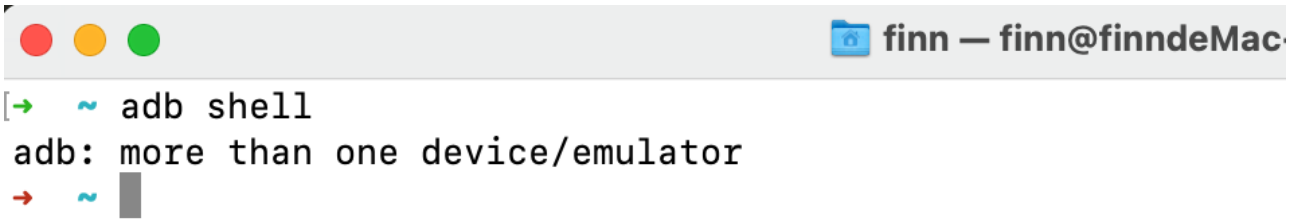
Use adb shell to Get into the Shell of Your Android PC

To exit the shell, you can either hit *Ctrl + D* or type *exit* .

Common Issues for Connecting

You might encounter unexpected behaviours when you're connecting to the Chipsee industrial PC from your workstation, here are some common issues we've seen.

The prompt says: "adb: more than one device/emulator".



The Prompt Says "adb: more than one device/emulator"

When you use *adb shell*, and there are more than one device, the adb might be confused about which to connect. This happens when you've been trying to connect to the Chipsee industrial PC from more than one method. You can confirm this situation by entering *adb devices*, you should see more than one device listed.

More Than One Devices are Listed

In this case, you can add a *-s* to specify which device you want to connect.

```
adb -s 192.168.50.146:5555 shell
```

If you prefer the USB, use:

```
adb -d shell
```

Or prefer TCP/IP:

```
adb -e shell
```

You can also disconnect all devices by:

```
adb disconnect
```

And kill the adb server by:

```
adb kill-server
```

You can also shut down the Chipsee industrial PC with adb:

```
adb shell reboot -p # -p means --poweroff
```

There are far more options you can try with **adb**, to learn more, you can search for **"adb usages"** on the Internet, or check out the official Android Developer documents. Adb is not a Chipsee vender specific tool, you can find a lot of tutorials about adb on the Internet.

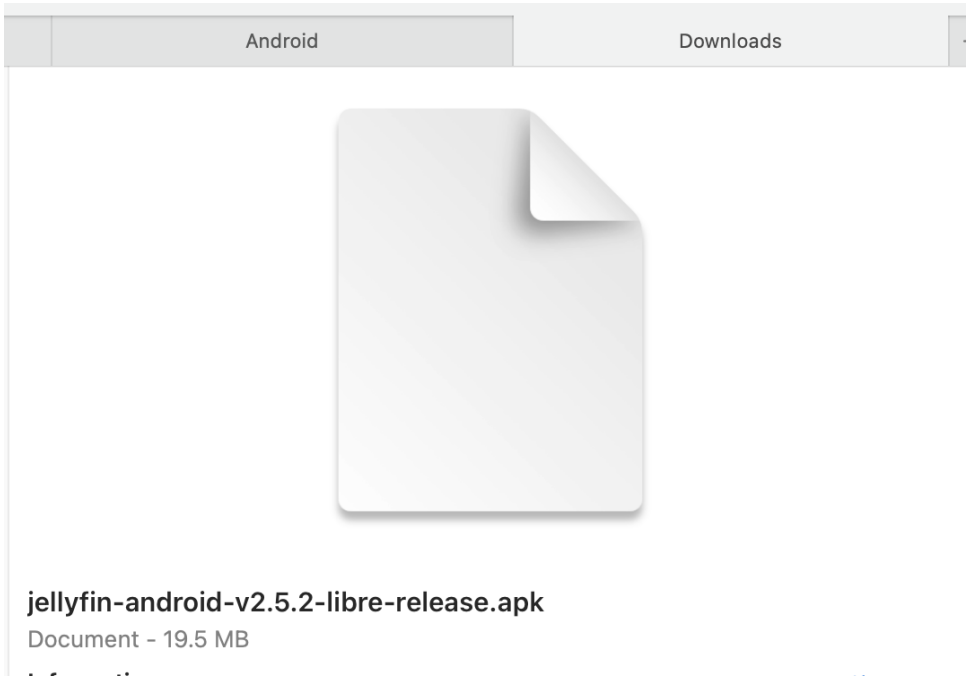
Install an APK

To install an app from your workstation to the Chipsee industrial PC, we'll also need the help of **adb**.

You should download or compile an APK file you want to install on your workstation, put it in a folder you can find, such as Downloads folder.

For **MacOS and Linux** users, you can open a terminal window, connect to your Chipsee industrial PC like explained in the previous chapter, but don't get into shell.

Let's say we have a copy of Jellyfin-Android apk in our Downloads folder:



Prepare Your APK File

Then we can type *adb install path-to-file* to install it to our Chipsee industrial PC's Android system. Take care of the relative file path, you can use an absolute path as well.

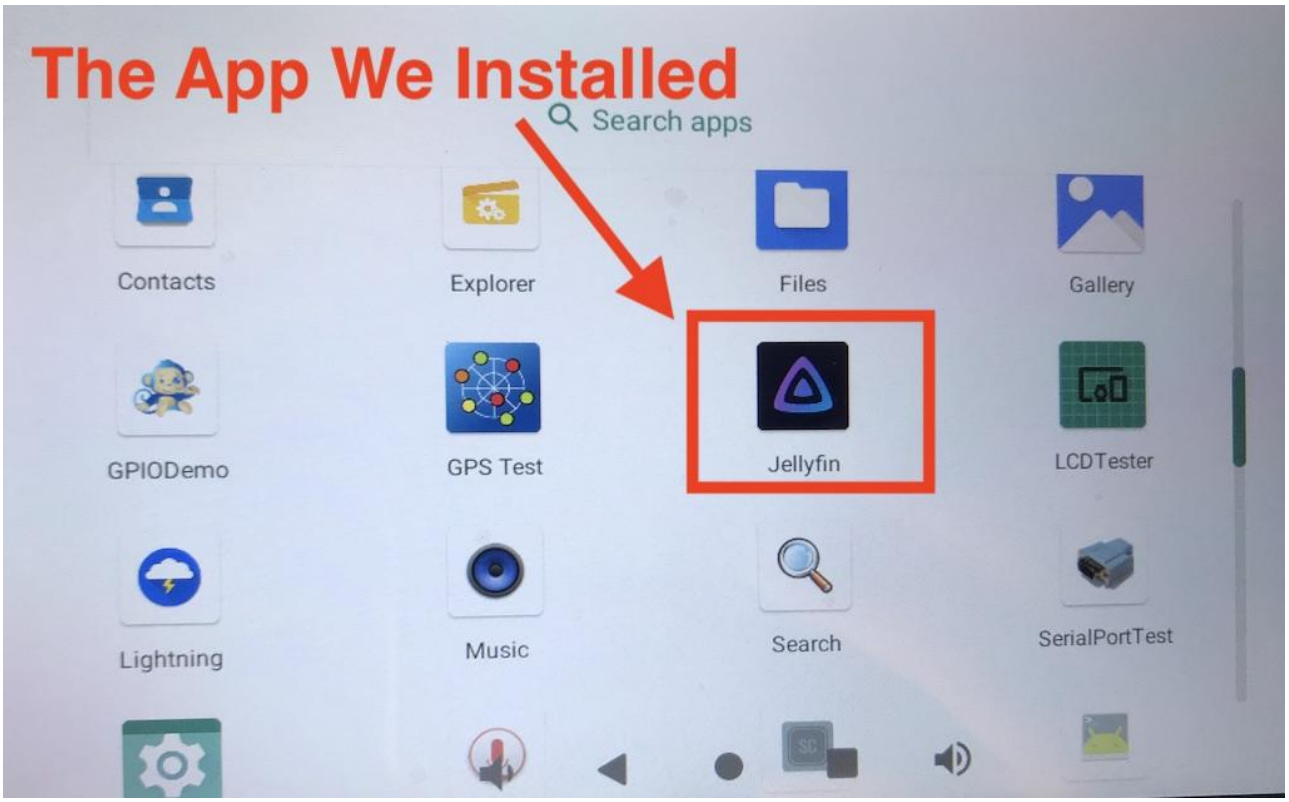
```
adb install Downloads/jellyfin-android-v2.5.2-libre-release.apk
```

It might take a minute to get one app installed, you should see a **success** in the terminal after the installation is finished successfully.

```
finn — finn@bogon — -zsh — 117×36
~
[➜  ~ adb devices
List of devices attached
192.168.50.211:5555     device

[➜  ~ adb install Downloads/jellyfin-android-v2.5.2-libre-release.apk
Performing Streamed Install

Success
➜  ~
```

Installation Finished After 1 Minute

You can check the Chipsee industrial PC from its screen, and confirm your app is installed successfully:



The App We Just Installed is on our Android PC

Hardware Resources in the OS

Now that you have successfully booted the Chipsee industrial PC and connected the Chipsee industrial PC to your laptop/computer, this section will tell you how to control this Chipsee industrial PC from its OS desktop itself, or from your PC.

Network

You can use the native Android network API or the settings menu to change network configuration.

Serial Port RS232 and RS485

The RK3568 based Chipsee industrial PC supports RS232 and RS485, here are the mapping from the port name to the system tree device:

5 inch product

Name	Node	Protocol
RS232_0	/dev/ttyFIQ0	RS232, Serial Debug
RS232_2	/dev/ttyS0	RS232
RS485_3	/dev/ttyS3	RS485
RS485_5	/dev/ttyS4	RS485

Table 230 RS232/485 for **5 inch** product(CS12720-RK3568-050P)

7 inch product and Box product

Name	Node	Protocol
RS232_0	/dev/ttyS0	RS232
RS232_2	/dev/ttyFIQ0	RS232, Serial Debug
RS485_3	/dev/ttyS3	RS485
RS485_4	/dev/ttyS4	RS485
RS485_5	/dev/ttyS5	RS485

Table 231 RS232/485 for **7 inch** product(CS10600-RK3568-070P) and **box** product(CS-RK3568-BOX)

10.1+ inch products

Name	Node	Protocol
RS232_0	/dev/ttyS0	RS232
RS232_2	/dev/ttyFIQ0	RS232, Serial Debug
RS485_3	/dev/ttyS3	RS485
RS485_4	/dev/ttyS4	RS485

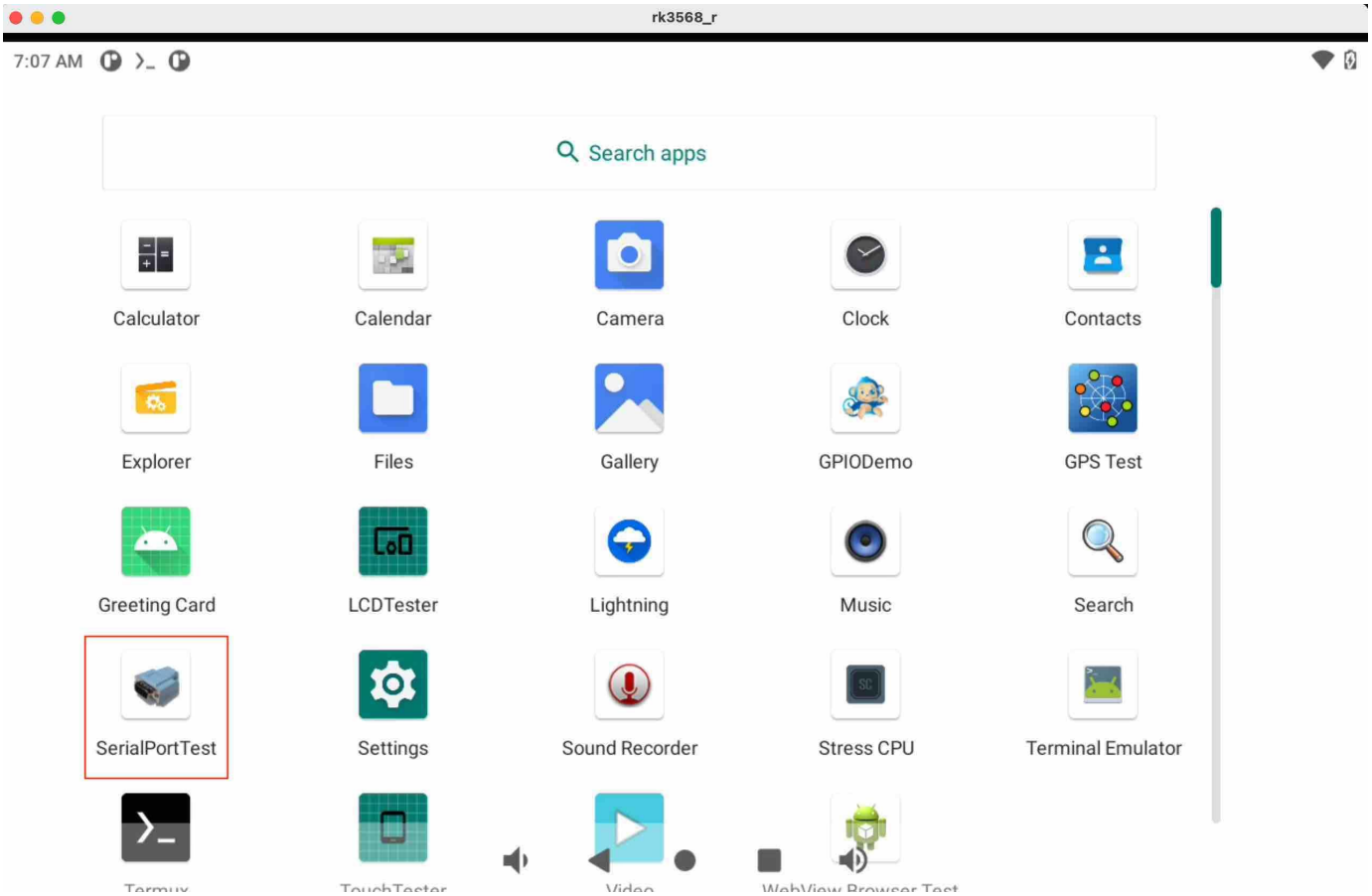
Table 232 RS232/485 for **10.1+ inch** products(CS12800-RK3568-101P, CS10768-RK3568-121P, CS19108-RK3568-133P, CS10768-RK3568-150P, CS19108-RK3568-156P, CS12102-RK3568-170P, CS19108-RK3568-185P, CS12102-RK3568-190P, CS12800-RK3568-215P, CS19108-RK3568-236P)

The 120 Ohm match resistor is already mounted on the RS485 port. RS485 ports are half-duplex, the hardware can switch the Tx/Rx direction automatically. RS232 ports are full-duplex.

We know of 3 methods to interact with RS232 and RS485 in Android OS, sadly Android API does not support native ttyS* RS232 and RS485 devices, so people usually use JNI to add some C code to interact with serial ports, we'll show you what we know, because we are not experts in Android developments, we hope these methods can give software engineer experts like you some inspiration.

Method 1: Google Serialport-Api App

The **first method** is an Android app Google built in 2009, they open sourced their code at: https://code.google.com/archive/p/android-serialport-api/. This app is also installed on the Chipsee industrial PC out of the factory, the app's name is "SerialPortTest". You can learn from the source code if you're building your own serial port app. There are also forks of this app on Github, some improves the app by adding more features like supporting a parity bit or flow control. You can search *android serial api* to find out more on the Internet.

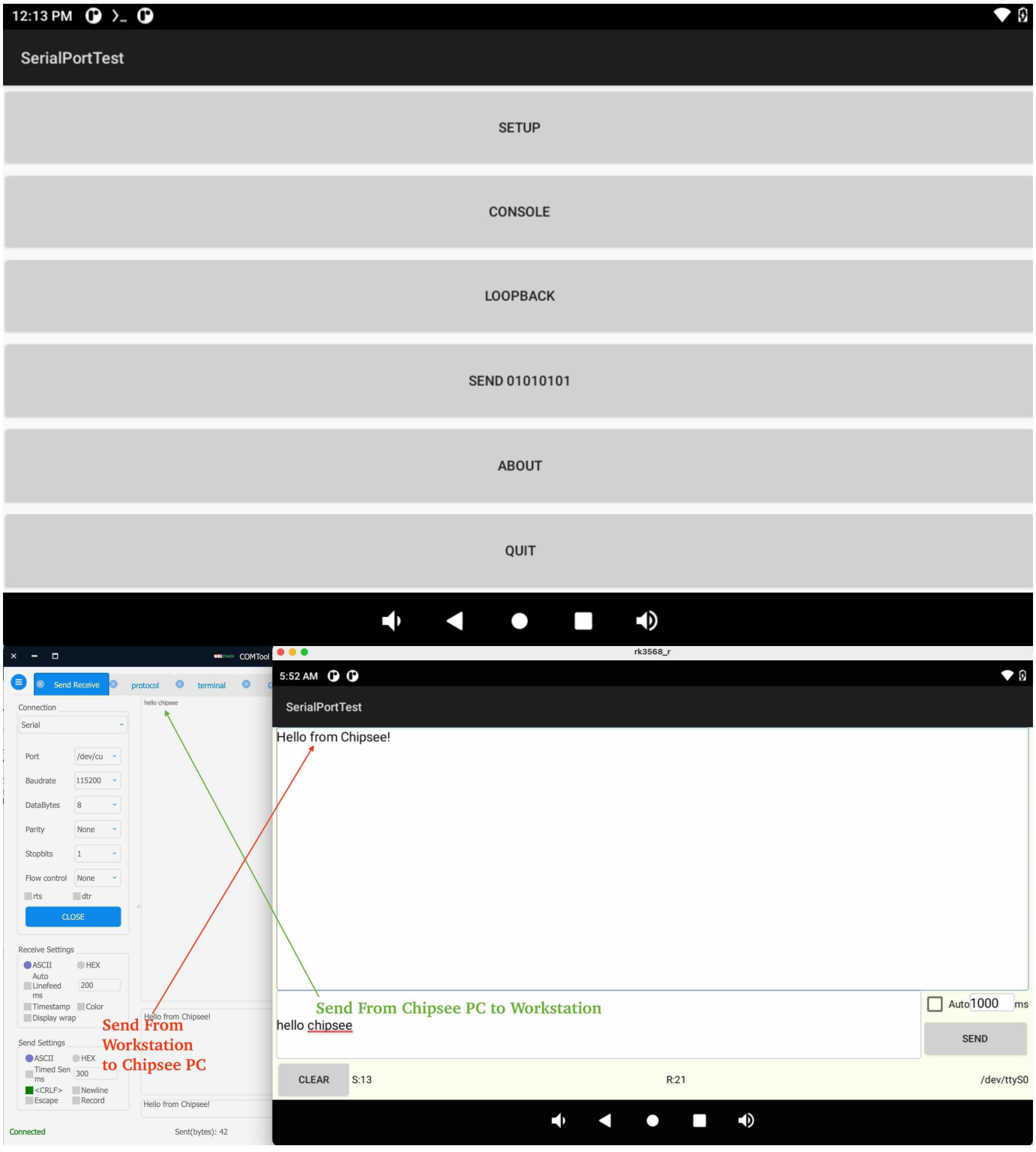


When you first get your Chipsee industrial PC, and looking to develop a program that involves serial ports, it's recommended to first test against the app Google built. For example, you have two Chipsee industrial PCs, you should wire the RS232/RS485 port of two devices, and launch this app on both devices. You then set the baud rate, send messages from one device to another, and see if you can get the correct message on the other device. In this way, you can confirm you're wiring the hardware pins correctly.

Then, if you have developed your app, you can install it on one of the devices, and still use this app on the other device to test your app. In this way, you make sure at least one device

is working correctly, and if things go wrong, you can be confident it should be your code's problem, and then you gradually debug to fix the program.

If you have one device, you can use the approach above to test it with your workstation, there are serial debug tools available on Windows, Mac or Linux. Use a USB to serial RS232/RS485 cable to connect them, beware the serial cable is **NOT** a USB-to-TTL converter, the TTL voltage is not the same as RS232/RS485.



The image above shows we can use Google's app to test serial port. On the left is our workstation running a COMTool, on the right is the Android app, they are connected with a RS232 to USB cable.

Method 2: ADB

The **second method** is using adb, **cat** and **echo** programs. You can wire the device's serial port to another device or your workstation, if you use two devices to communicate with each other, open two adb shells on your terminal app, each window should belong to one of the devices. In the example below, I will let the device talk to my Mac Mini, and use another tool on Mac called COMTool, it's a serial debugger you can download from Github.

At the beginning you may wish to set the baud rate of the Chipsee industrial PC's serial devices, you can use **stty**, the command below will set ttyS0 (our RS232_0) to 115200 baud rate.

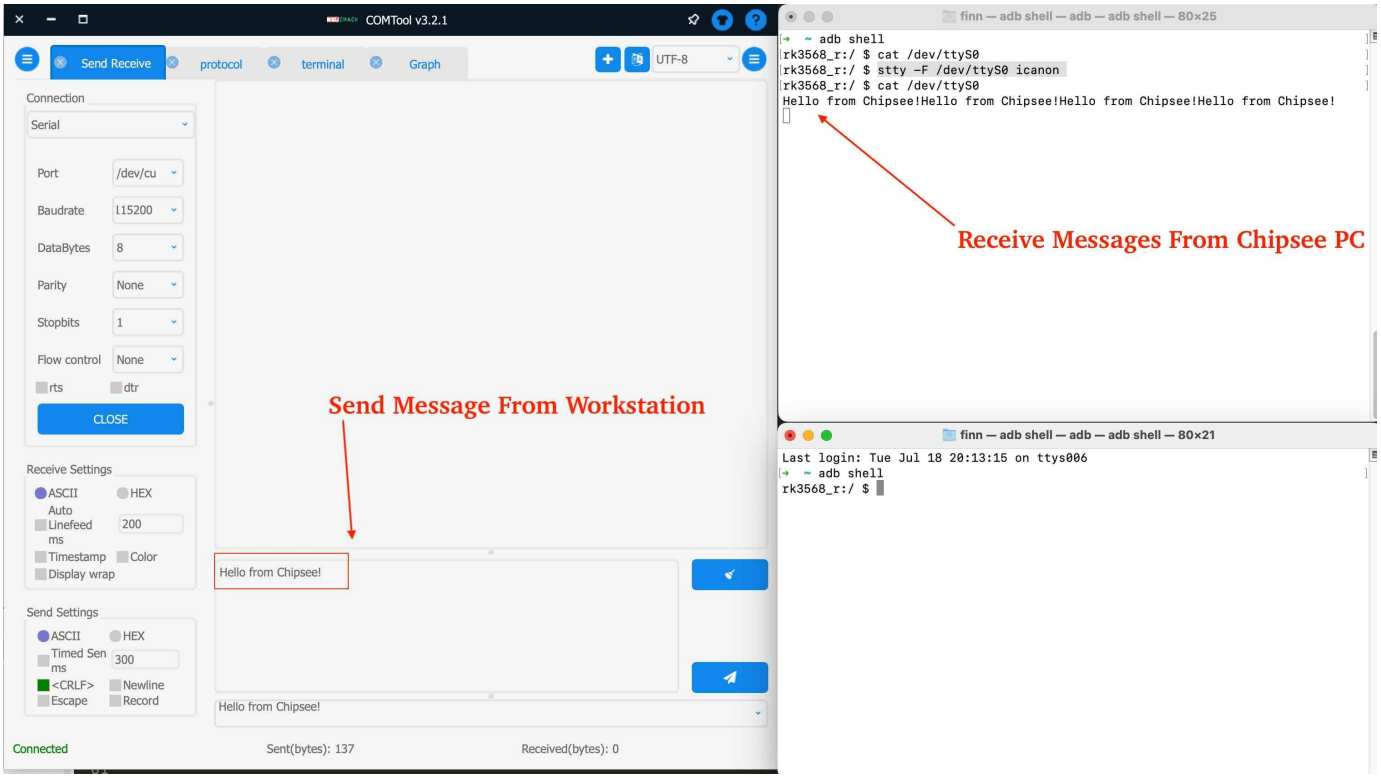
```
stty -F /dev/ttyS0 115200
```

In the first window, you use **cat** program to listen for RS232/RS485 message:

```
cat /dev/ttyS0
```

> **Note**
>
> If you've been running some programs before, like a *pyserial* Python script (yes it's possible on Android!), the cat program might exit immediately. You can use a *stty* command to set most settings back to defaults (canonical), for more information, here is a doc for *stty*: https://man7.org/linux/man-pages/man1/stty.1.html.
>
> ```
> stty -F /dev/ttyS0 icanon
> ```
>
> And then use *cat /dev/ttyS0* again.

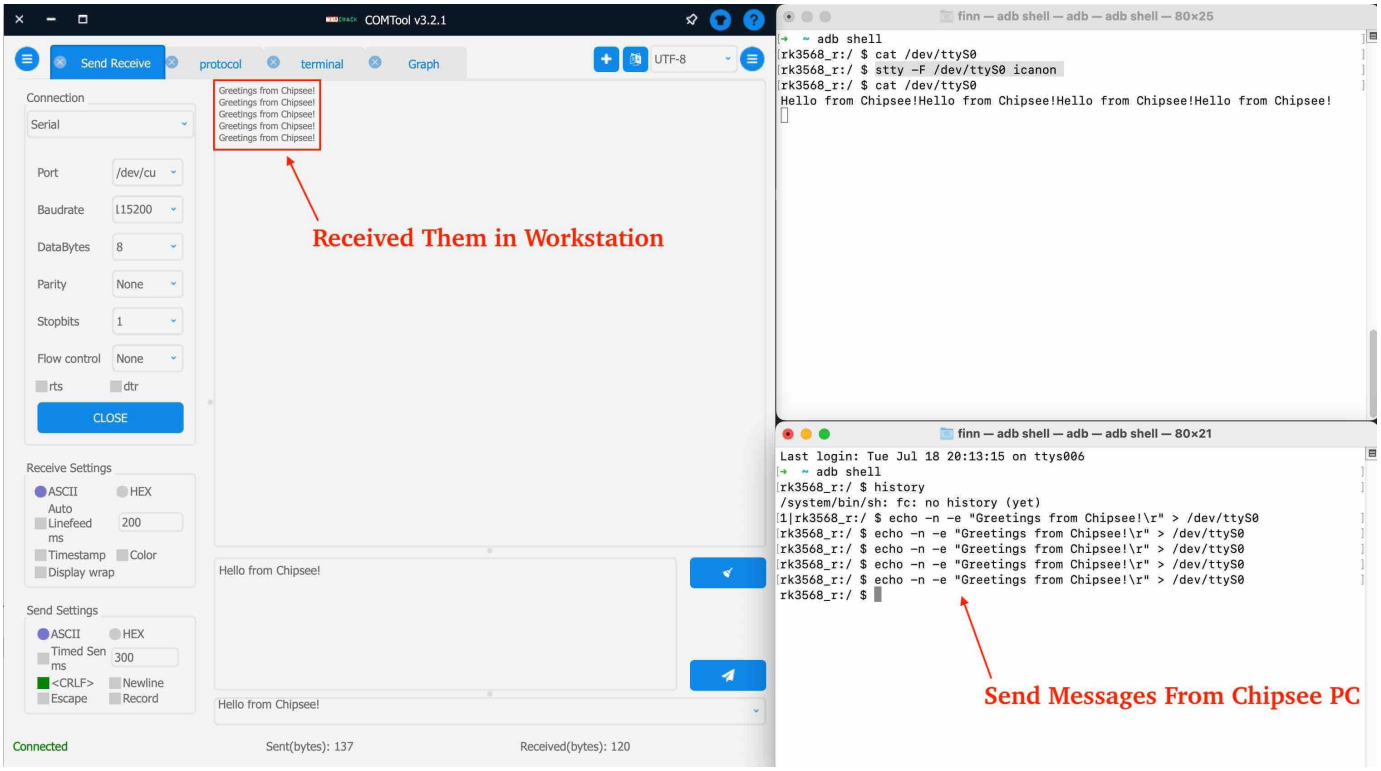
You should see this cat program seems blocking, and it's waiting for messages. Then we use our workstation to send a command through the serial port wire:



The image above shows the Mac's COMTool sent multiple "Hello from Chipsee!" to the Chipsee industrial PC using RS232.

To send message from Chipsee industrial PC and receive them in our workstation, we can use **echo**:

```
echo -n -e "Greetings from Chipsee!\r" > /dev/ttyS0
```



This is one way to send and receive message from ADB, the ttyS* are our serial port RS232/RS485 devices represented by Linux files in Android OS. Hopefully, an Android developer could find this useful for building his Android native apps.

Method 3: Flutter Android

The **third** method is a Flutter app. We made an Android app with Flutter in this repo: https://github.com/printfinn/chipsee_serial_port_flutter.

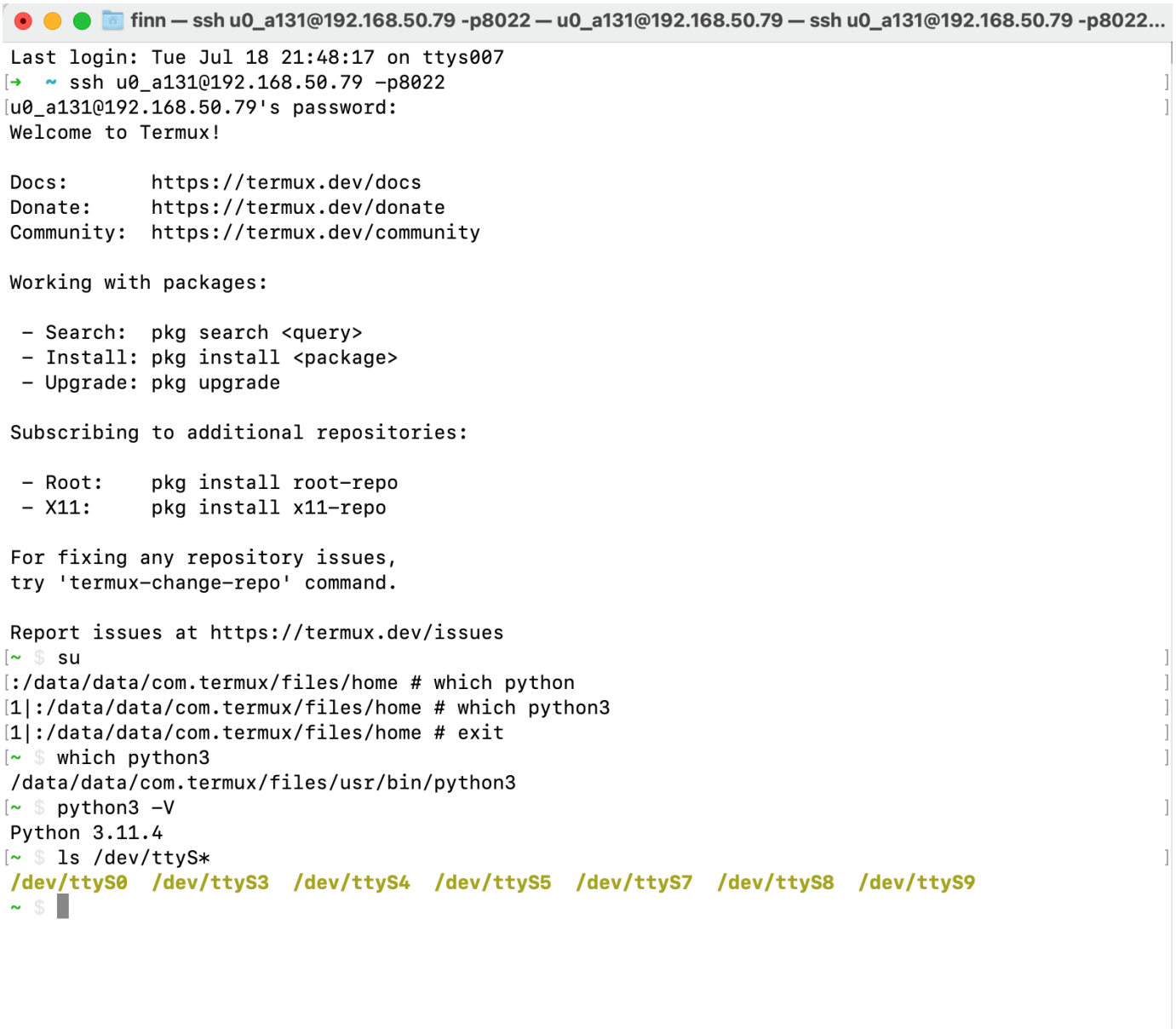
Here is a video of how to read data from serial port: https://www.youtube.com/watch?v=VQrIEv66WFE.

Here is a video of how to write data to serial port: https://www.youtube.com/watch?v=LlE8OHMz-lk&t

Method 4: Termux

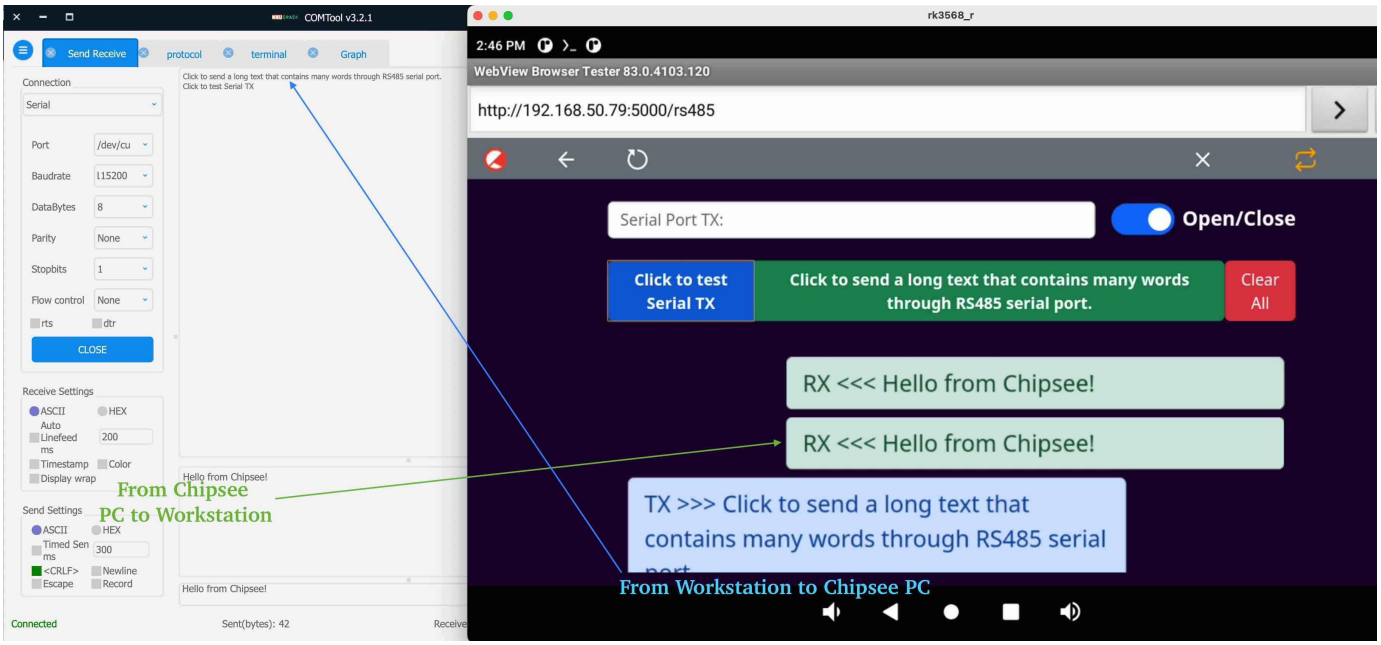
The **fourth** method is a workaround, even if it does not compile to an Android APK, it gives a similar user experience. It uses an Android app called **Termux**, which is an emulator that turns your Android to a Linux environment. With Termux, you can use the programming tools and languages you're familiar with, like Python, JavaScript. In this Termux Linux environment, you still have access to the Android's Chipsee hardware, like the RS232 and RS485 devices, through ttyS* files nodes. Then you can start a program to listen for incoming requests, such as a web server, then open a web page to send requests to this server. There are other solutions as well.

You can read the Termux's official doc to learn how to install your preferred Linux tools, for example, we installed Python 3.11 (in July, 2023):



In the image above, it shows we can login to the Android Chipsee industrial PC through ssh, and we have Python3 installed, we are also able to find our ttyS* devices (our RS232 RS485 devices).

We built a demo Python Flask web app targeting Debian OS, but it can also run on the Termux environment in Android OS, and the RS232, RS485 controlled by the *pyserial* library are fully functional as well. We can then open a browser or a webview Android app to visit the local IP address hosted by the Termux Linux environment on the Android to interact with RS232/RS485.



The image above shows we can communicate through RS485 between our workstation and Chipsee industrial PC. On the left is my workstation Mac, on the right is a screen of our Chipsee industrial PC. The Flask Python web server running in the Termux picks up the messages from ttyS3 (our RS485_3) using *pyserial* library, and then the Python server sends a command through websocket to the webview (think about a webview as a less powerful web browser Android app), then JavaScript in the webview picks up the message and displays them in the Android's screen as a HTML file.

To use the third method in your development, you can find out how Termux can auto boot your server program with *Termux:Boot*; and build a simple Android native APK that integrates a webview to visit your server's web port as soon as the APK is started; then start your app automatically on system boot. In this way, when your Chipsee industrial PC boots, your program is ready to use without having to type commands in Termux.

The source code of this demo app is in Chipsee's Github: https://github.com/Chipsee/chipsee-industrial-pc-web. If you're looking to use the approach above, or to develop your program on Debian Linux, you can refer to the repo, it's a Python program with HTML web pages. Except for serial ports, it has code to control buzzer, gpio, backlight, CAN and implemented a Modbus server/client as well.

Those are the three methods we know of that can control the RS232/RS485 devices on Android.

GPIO

There are 8 GPIOs, 4 Output, and 4 Input, they are all isolated. You can control the output or input pin voltage by feeding the VDD_ISO suite voltage. The pin voltage should be from 5V to 24V. Refer to the tables below for a detailed port definition:

Function	Device Node
IN1	/dev/chipsee-gpio5
IN2	/dev/chipsee-gpio6
IN3	/dev/chipsee-gpio7
IN4	/dev/chipsee-gpio8
OUT1	/dev/chipsee-gpio1
OUT2	/dev/chipsee-gpio2
OUT3	/dev/chipsee-gpio3
OUT4	/dev/chipsee-gpio4

Table 233 GPIO Device Node

- **Set** *OUT1* **to high or low**

```
$ echo 1 > /dev/chipsee-gpio1    # set OUT1 to high

$ echo 0 > /dev/chipsee-gpio1    # set OUT1 to low
```

- **Get** *IN1* **value**

```
$ cat /dev/chipsee-gpio5    # value 1 indicates high, value 0 indicates
low
```

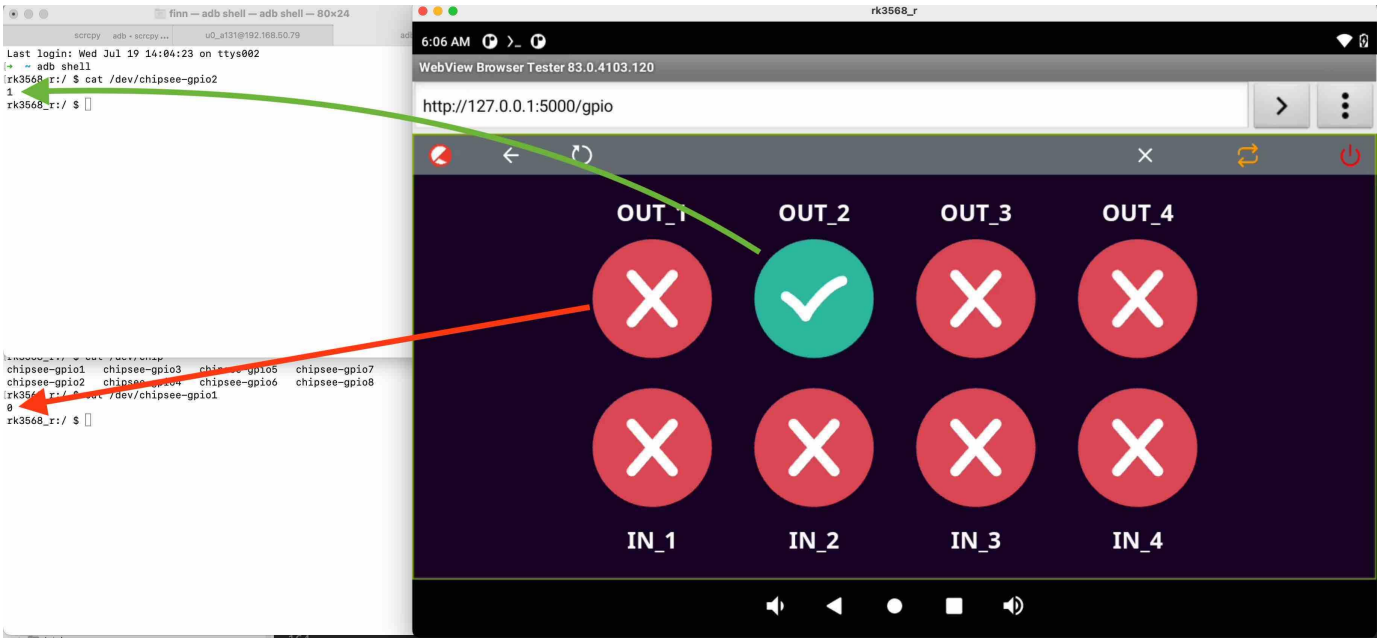
GPIO devices are already exported as 8 Linux files in the Android OS, they are */dev/chipsee-gpio1 ~ /dev/chipsee-gpio8*.

To use GPIO in your program, there are also two methods:

The **first** method is to read the file or write 0/1 to the file to control GPIO in **adb**. Similarly, you can also build Android APK and use Kotlin or Java to read/write those files to control GPIO.

The **second** method is the workaround we addressed in the serial port RS232/RS485 section. We can use Termux to take advantage of a Linux environment and are still able to use the /dev/chipsee-gpio1~8 device nodes.

Here is an demo we made in Python and webview running in Android:

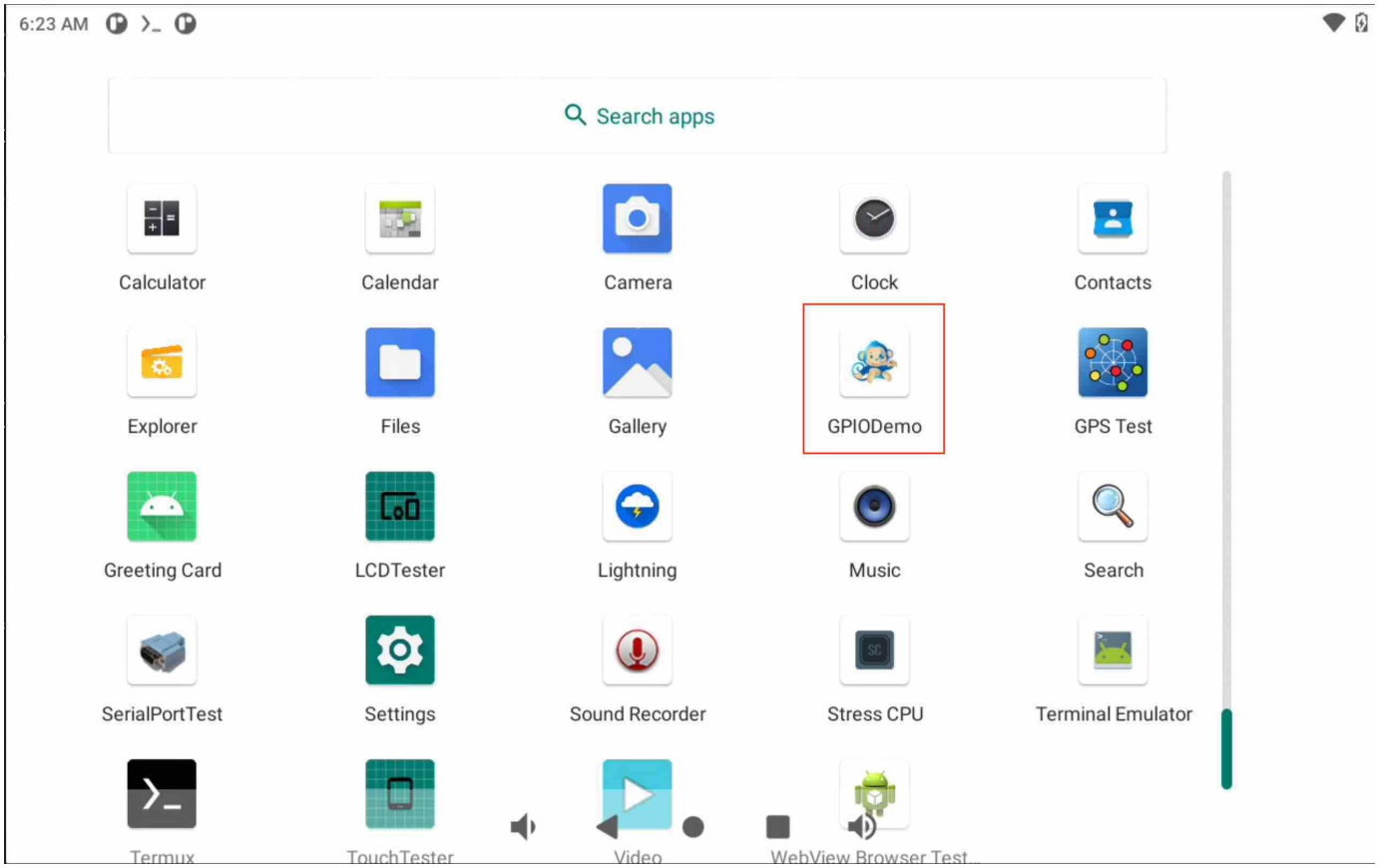


The image above shows 8 GPIO ports in a webview (similar to a web browser, but in a limited Android app), the 4 round icons above are 4 GPIO output pins, among them the OUT_2 is set to high voltage by tapping the screen. We can confirm this in the left top window, when we *cat* the /dev/chipsee-gpio2, it returns a “1”, indicating a high voltage.

The 4 round icons in the bottom are 4 GPIO input pins, because we are not applying voltages to them, they all appear as red cross, meaning they are “0”. If we apply a logic high to any pin, it would turn green.

This GPIO demo is a Python program, like it is discussed in the serial port section, it’s running in the Linux environment on Android OS using Termux. The program starts a Flask web server, listens for requests from the webview. In the webview part, JavaScript polls the Python web server to know if the content in the HTML should be changed, then Python reads/writes contents of 8 /dev/chipsee-gpio* files to know the status of the input pins or make a change to the output pins.

To **test** the GPIO, there is a GPIODemo app made by Chipsee pre installed in your Android OS:



Inside the app, you can test GPIO outputs and inputs, as well as Buzzer.

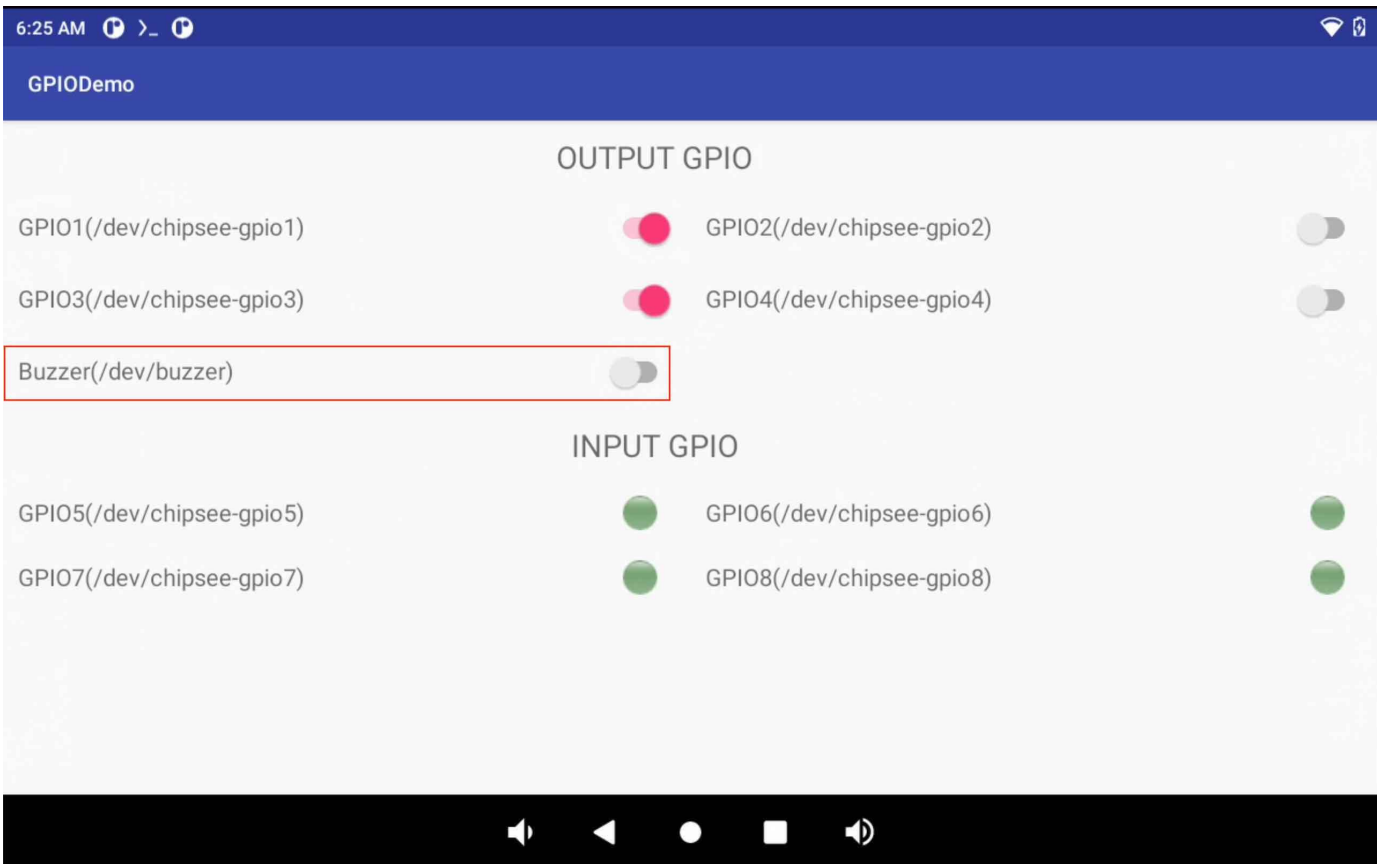
BUZZER

Inside ADB, you can **echo** 0 or 1 to control the buzzer.

The Chipsee industrial PC has one buzzer. We have created one symbol link to */dev/buzzer* . You can control it as follows:

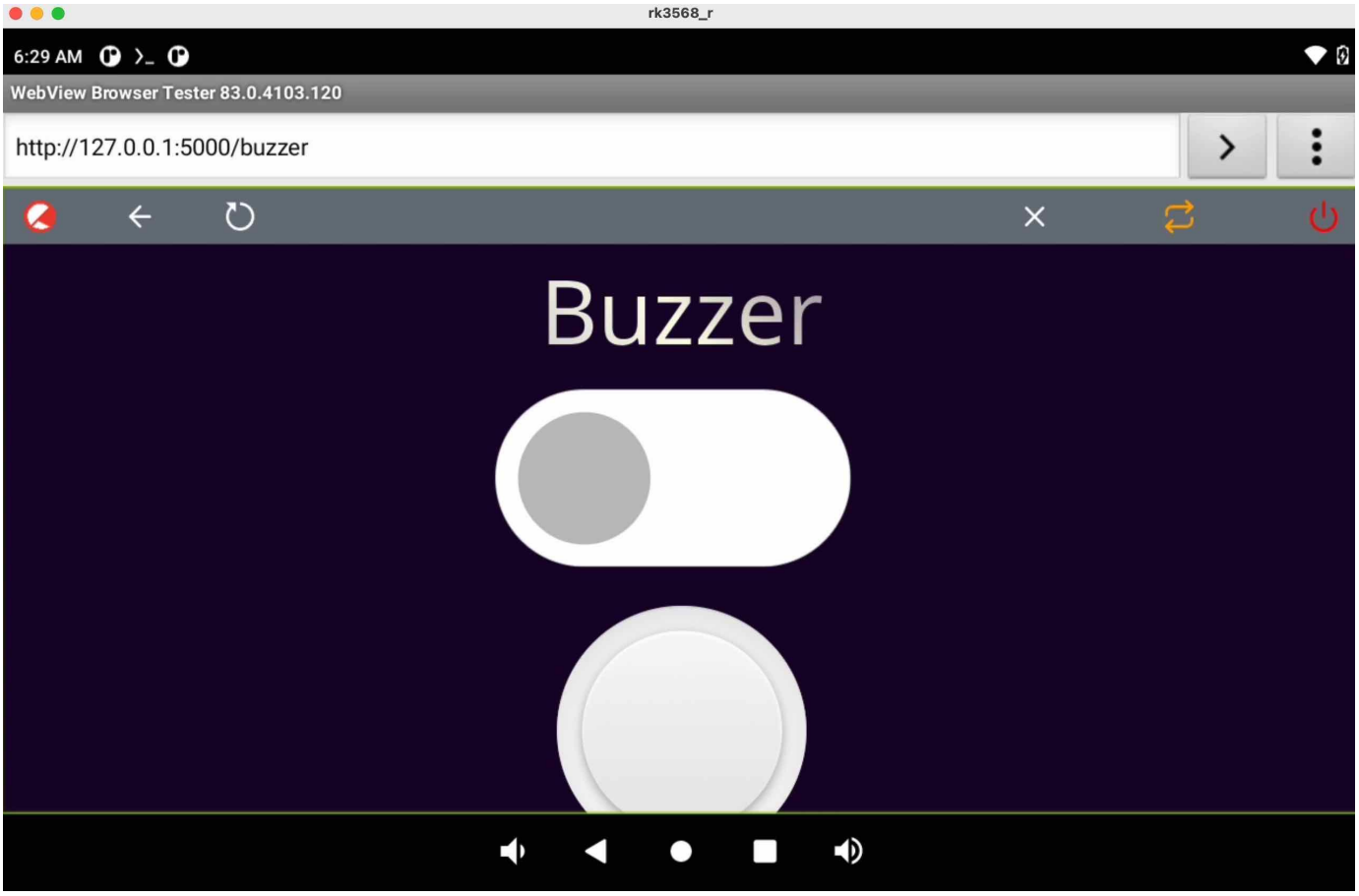
```
$ echo 1 > /dev/buzzer    # enable buzzer

$ echo 0 > /dev/buzzer    # disable buzzer
```

As shown in the GPIO section, you can **test** buzzer in the GPIODemo app.



The red square has a buzzer switch, when you toggle the switch, the internal buzzer will be enabled.

You can also use Termux and Linux plus your favourite programming language to control buzzer:



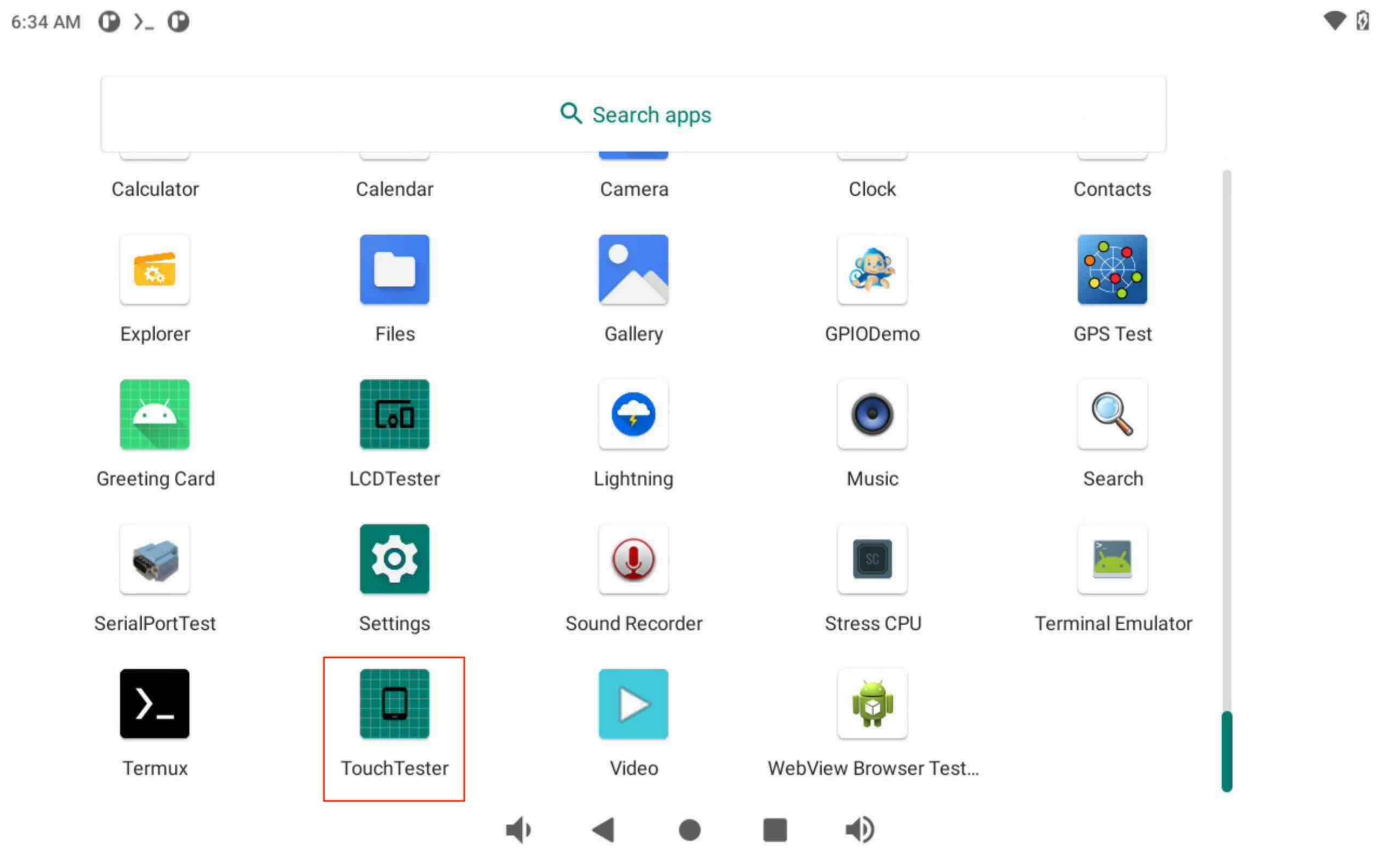
The image above is a Python plus webview demo for controlling buzzer. Same Termux Linux environment as GPIO and Serial Port sections.

Backlight

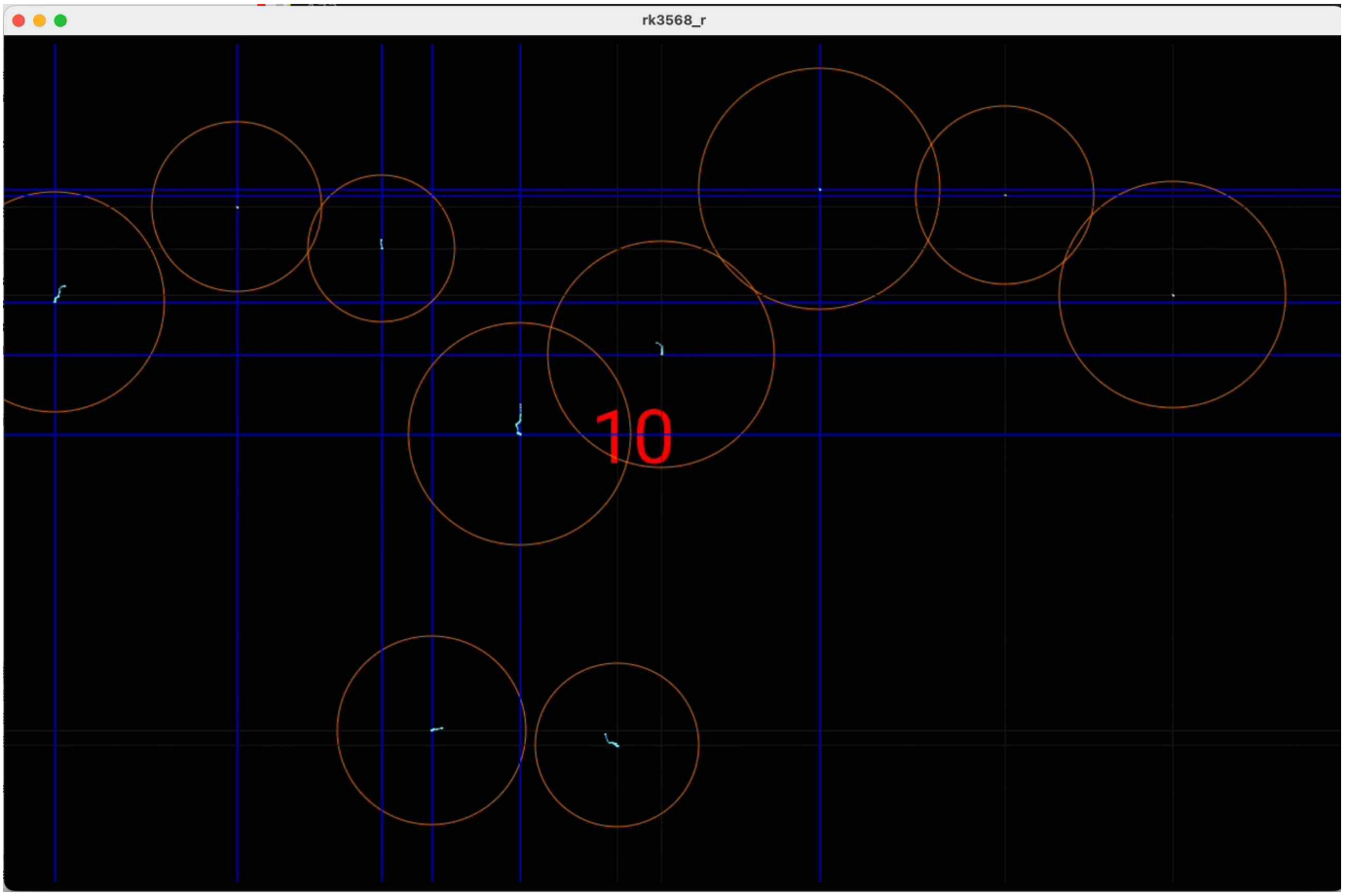
Android OS has native backlight API.

Multitouch Test

You can test the touch screen with the preinstalled TouchTester app:



For different Chipsee industrial PC models, some supports 5-point multitouch, some supports 10-point multitouch.



The image above shows a 10.1 inch Chipsee industrial PC supporting 10 point multitouch.

Flashing OS Image

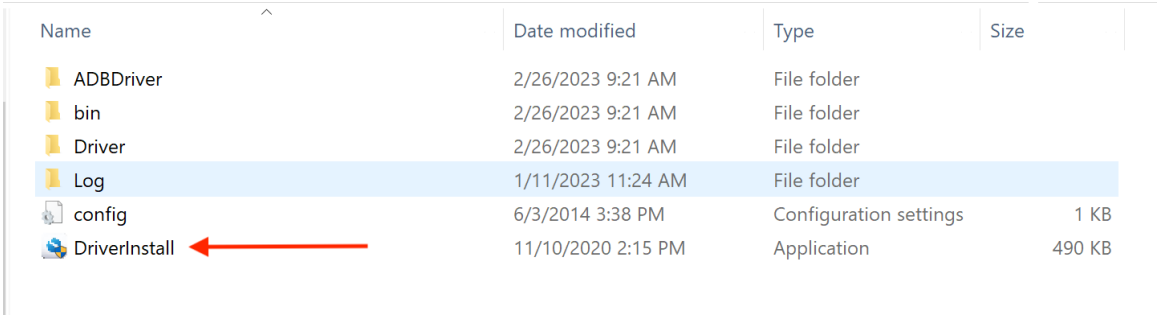
Download Required Tools

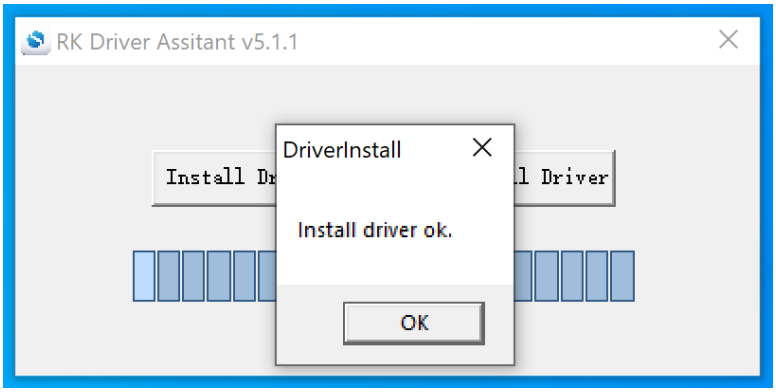
If you want a fresh OS, you can flash your Chipsee industrial PC.

You need two tools to flash the Android 11 OS image to the RK3568 PC. The first is *DriverAssistant_v5.1.1*, the second is *RKDevTool_v2.93*, you can download all of them here.

These tools are Windows executables, please execute them on a Windows machine.

If you've been using a prior version of *DriverAssistant*, click uninstall before installing *DriverAssistant_v5.1.1*.



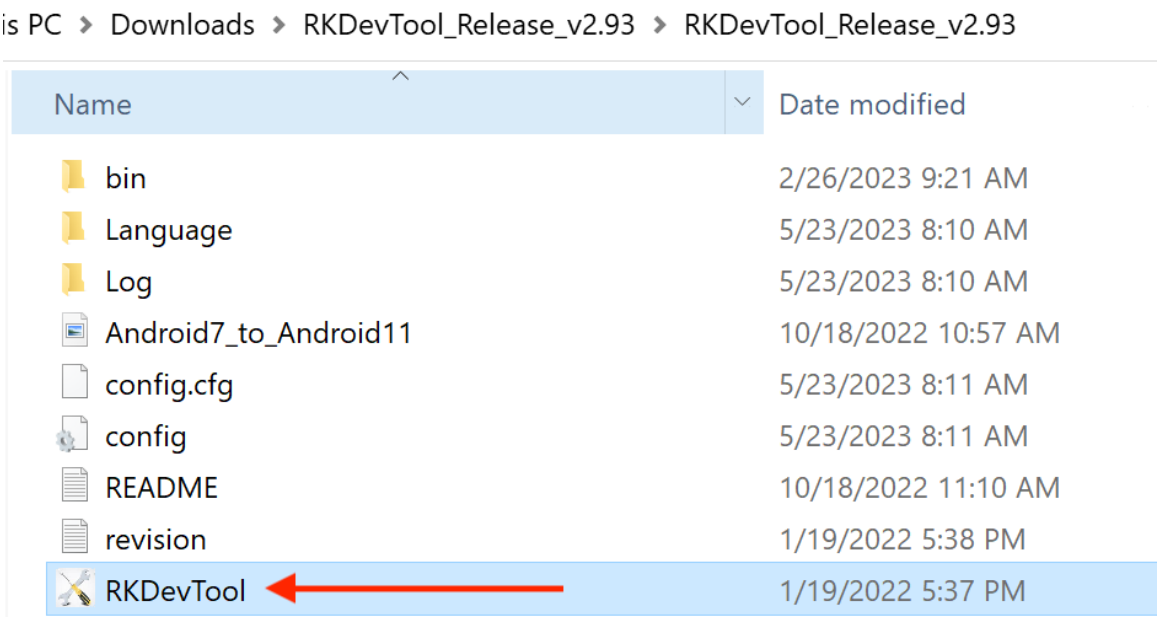


Download Prebuilt OS Images

If you haven't downloaded the prebuilt OS images, you can find one here.

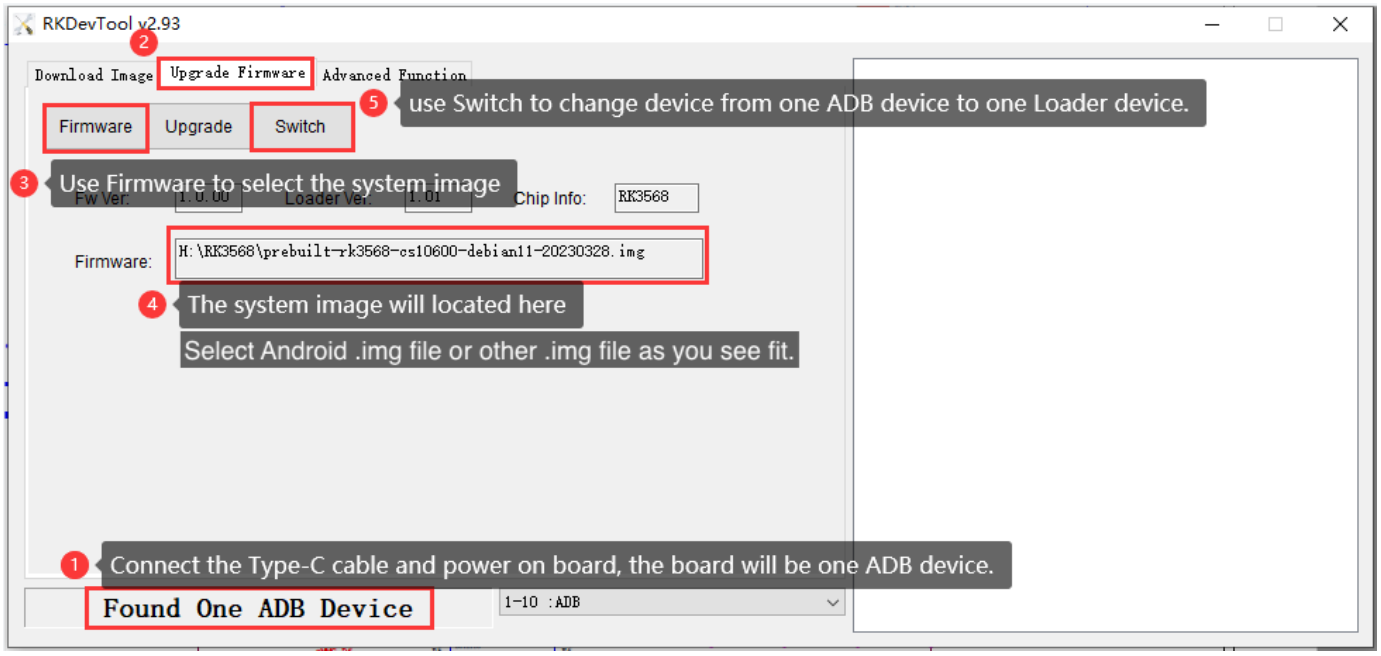
Start Flashing

After installing the DriverAssistant, you can now start to flash an OS image to the RK3568 board with *RKDevTool*. Double click the program to start flashing. The tool has English and Chinese language support.



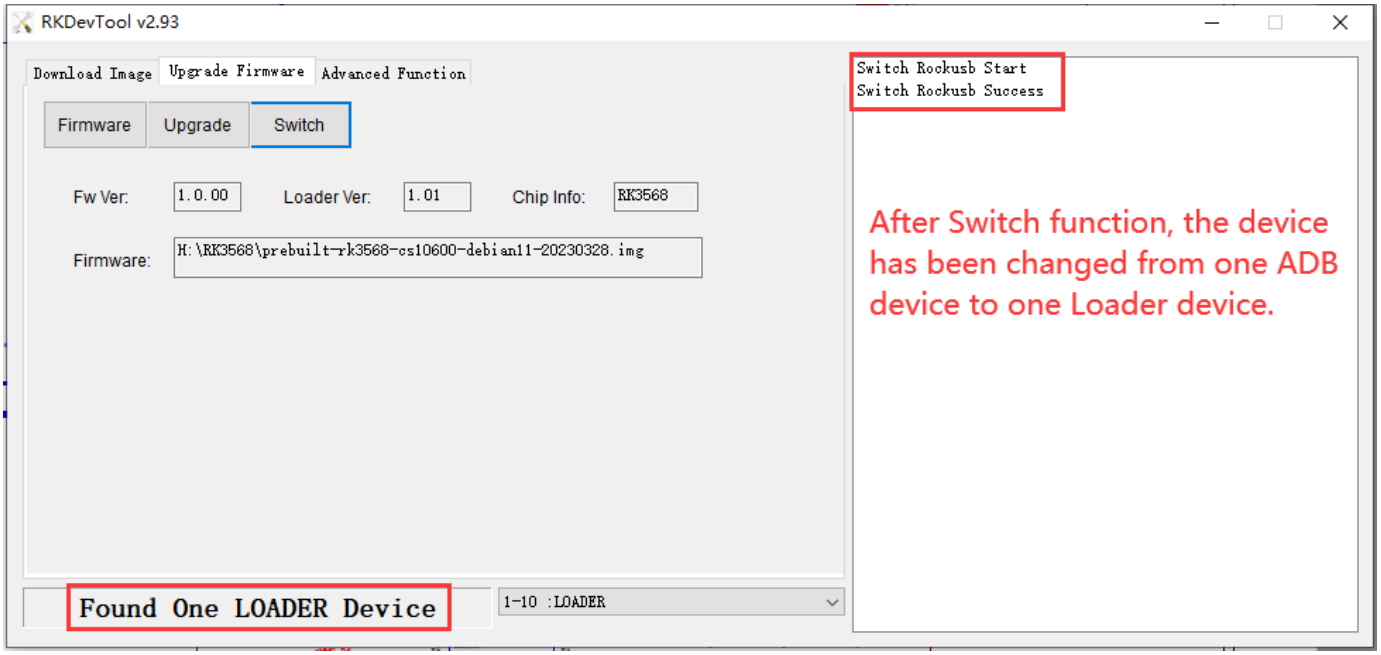
STEP 1:

1. Connect the Type-C cable and power on the board. (If unexpected messages occur at any of the following steps, try plugging the Type-C cable again.)
2. Click **Upgrade Firmware** tab.
3. Click **Firmware** button to select a .img Android 11 image file. The screenshots show a debian11 img file is selected, but this is applicable to other OSes as well.



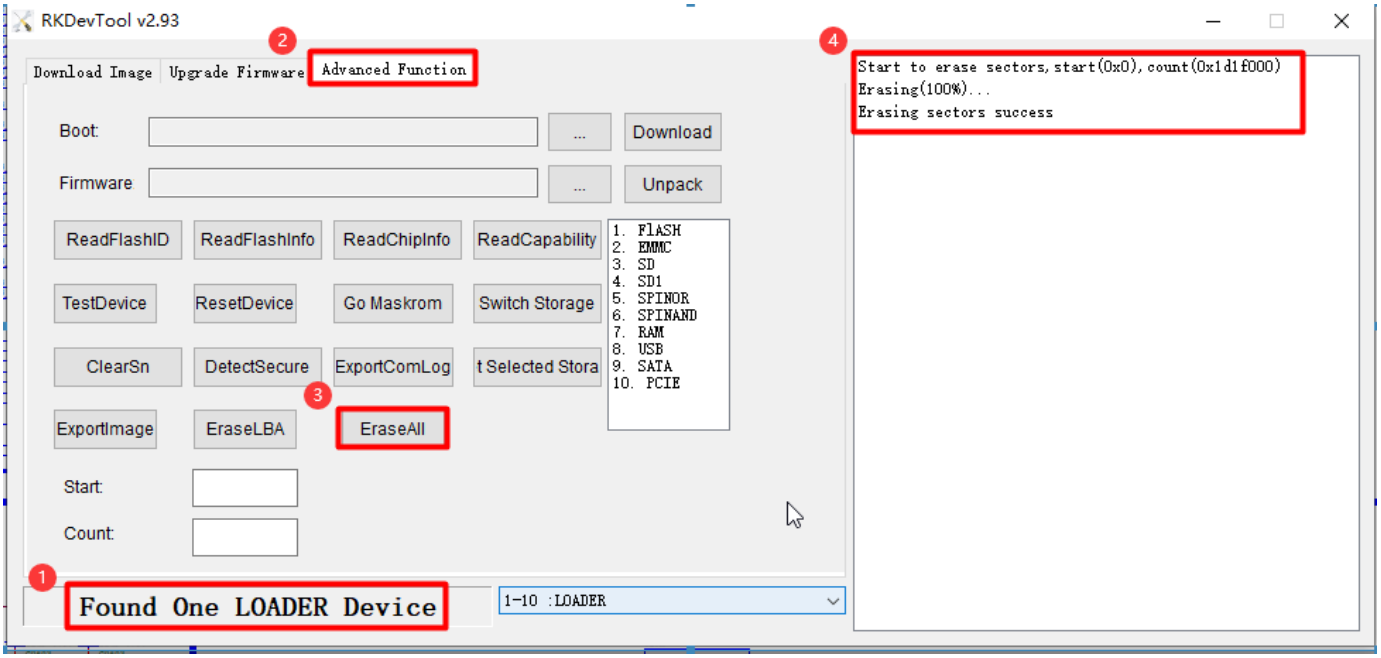
STEP 2:

1. Click **Switch** button to switch the device to a Loader device.



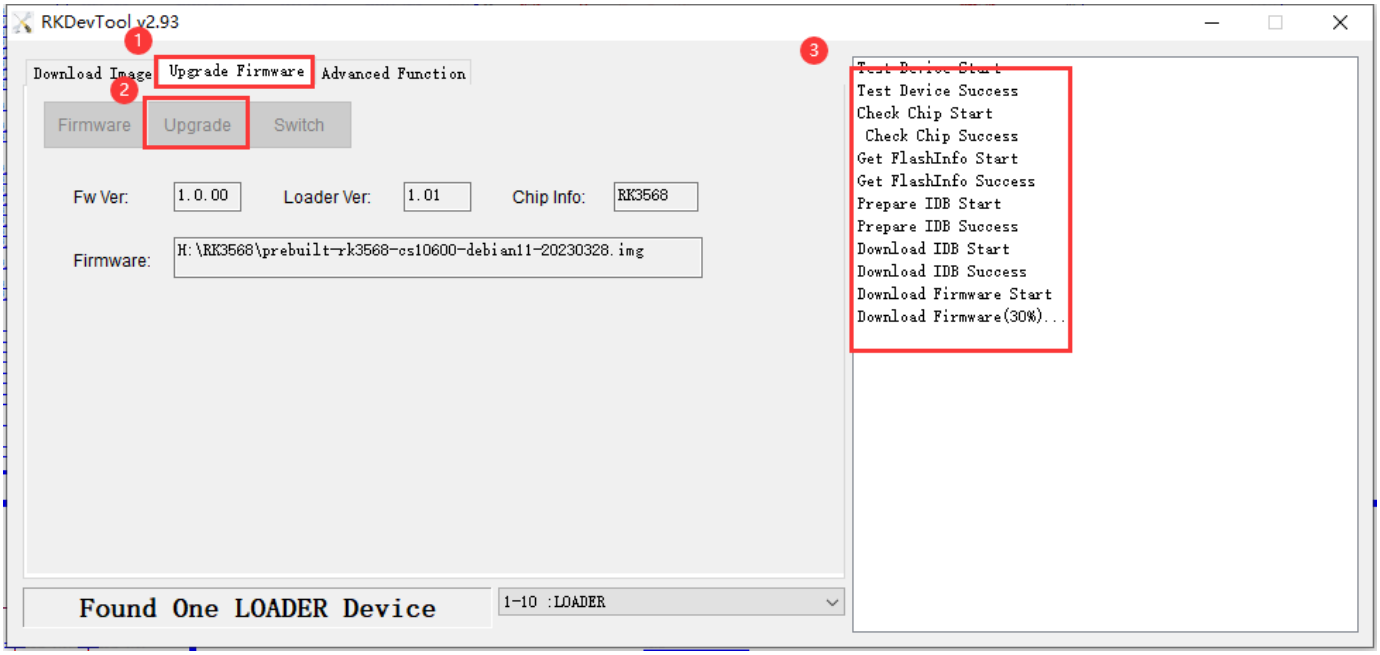
STEP 3:

1. You should see “Found One LOADER Device”.
2. Click **Advanced Function** tab.
3. Click **EraseAll** button.
4. You should see “Erasing sectors success” on the right side logs.



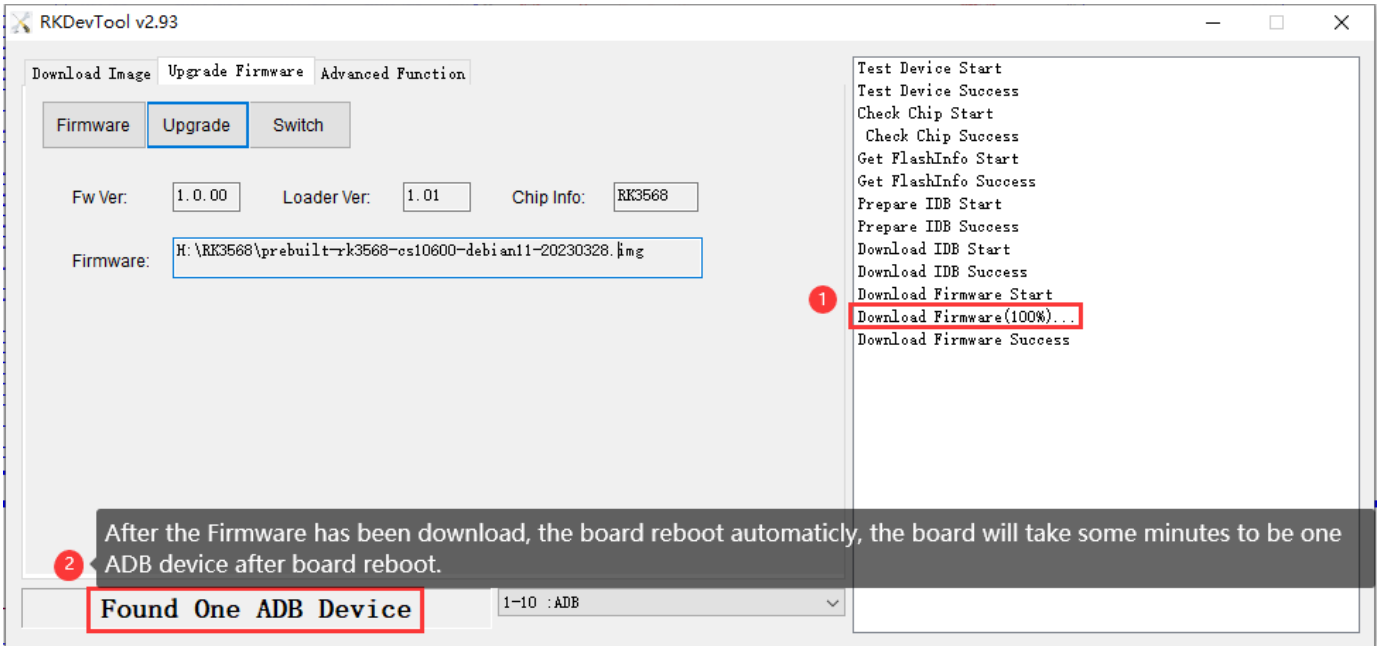
STEP 4:

1. Click **Upgrade Firmware** tab.
2. Click **Upgrade** button.
3. You should see Download Firmware progress on the right side logs.



STEP 5:

1. After the download firmware progress goes to 100%, the board reboots itself automatically.
2. After a few minutes, you should see “Found One ADB Device”.
3. Now your new OS is ready for use.



Video Tutorial for Flashing OS

Method 1: LOADER Mode

Here is a video tutorial we made demonstrating the OS installation process described above in Windows in the **LOADER** mode: https://www.youtube.com/watch?v=ufKDCJ1hpf4

The approach in the video above works best for devices that are still able to boot into the desktop, and when your workstation is a Windows machine. However, if you do not have a *Windows* machine in the room, you can use the approach below to flash an OS, in a Linux or Mac.

Method 2: MASKROM Mode

Apart from flashing in **LOADER** mode, when you're working on a *Linux(X86_64)* workstation or *MacOS(Intel and Apple Silicon)* machine, you can use another approach: **MASKROM** mode, to flash the OS. There is a PROG button on the Chipsee industrial PC, you can press the button before powering up the device, power up and hold the PROG button for 2~4 seconds, then use a X86_64/darwin_64 *upgrade_tool* program in the command line to flash the OS, here is a video we made to teach you how to do that in two minutes: https://www.youtube.com/watch?v=TDIHoQ9AuX4

The approach described in the second video works best for devices that are "bricked" (compared to the first approach), it can help rescue your device if your operating system is broken and cannot boot into the desktop. Even if your device is still functional, you can also use this approach to flash an OS, it works in Windows, Linux as well as MacOS.

The command used in the videos are:

For **Linux** workstation:

```
sudo ./upgrade_tool_linux_x86-64 ld # to list device
sudo ./upgrade_tool_linux_x86-64 uf ./prebuilt-rk3568-xxx.img # to upload
firmware
```

For **MacOS**:

```
./upgrade_tool_darwin64 ld # to list device
./upgrade_tool_darwin64 uf ./prebuilt-rk3568-xxx.img # to upload firmware
```

And that's all it takes.

The **upgrade_tool** used in the video can be download at:

1. `upgrade_tool_x86-64 (For Linux x86)`
2. `upgrade_tool_darwin64 (For MacOS Intel & Apple Silicon)`

We've tested that the MacOS upgrade_tool can execute in M1/Apple Silicon Macs, but you will need to install Rosetta to run this program. For Intel Macs, you do not need Rosetta, you can execute the binary program directly in your terminal.

Also, as noted in the video, do use a **absolute path** to the firmware file or **"./prebuilt-rk3568-xxx.img"**, rather than a relative path (e.g. your current directory contains the img file, and you directly use "upgrad_tool uf prebuilt-rk3568-xxx.img", this will not work). And make sure to use *sudo* in Linux.

Disclaimer

This document is provided strictly for informational purposes. Its contents are subject to change without notice. Chipsee assumes no responsibility for any errors that may occur in this document. Furthermore, Chipsee reserves the right to alter the hardware, software, and/or specifications set forth herein at any time without prior notice and undertakes no obligation to update the information contained in this document.

While every effort has been made to ensure the accuracy of the information contained herein, this document is not guaranteed to be error-free. Further, it does not offer any warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document.

Despite our best efforts to maintain the accuracy of the information in this document, we assume no responsibility for errors or omissions, nor for damages resulting from the use of the information herein. Please note that Chipsee products are not authorized for use as critical components in life support devices or systems.

Technical Support

If you encounter any difficulties or have questions related to this document, we encourage you to refer to our other documentation for potential solutions. If you cannot find the solution you're looking for, feel free to contact us. Please email Chipsee Technical Support at **support@chipsee.com**, providing all relevant information. We value your queries and suggestions and are committed to providing you with the assistance you require.