# Comparing Raspberries and Oranges

## A good-value allrounder

By **Tam Hanna** (Slovakia)

uilding a project around a Raspberry Pi can work out relatively expensive. The made-in-China Orange Pi family comprises over a dozen different boards which are more appealing in terms of both price and computing power. In this article we take the Orange Pi Lite for a quick spin.

Eben Upton's Raspberry Pi Foundation has for a long time had the hobbyist market in single-board computers sewn up. The processor, which was cheap to obtain as it was being mass produced for use in smartphones, allowed the then manager at Broadcom to develop an embedded computer with which at least initially it was hard to compete on price.

It's often said that Chinese manufacturers always come up with a response to any product sooner or later. Allwinner, a processor manufacturer based in China, was not going to let their competitor Broadcom enjoy this success without a fight, and one answer that employed their devices was the Orange Pi, which came out in 2015. Since then the family has grown, and more than a dozen different models can be found at AliExpress.

An engineer is always keen to try to find ways to reduce costs and increase processing power. The Orange Pi family [1] from Shenzhen Xunlong will appeal to anyone who wants to control external hardware using a Linux-based computer offering a similar level of performance to a Raspberry Pi board. A particularly attractive feature of the Chinese boards is that they offer good Wi-Fi range. And perhaps some people will also prefer to look for an alternative to the Raspberry Pi if they do not find themselves aligned with the political inclinations of the Raspberry Pi Foundation.

### Availability

Although Orange Pi controller boards are available from various distributors all over the world, the true connoisseur will want to buy direct from the manufacturer in China. Shenzhen Xunlong offers a range of kits and accessories on their website, and the products are also available via AliExpress.

The question then arises as to which boards are the most interesting to experiment with. The author has divided up the available controller boards according to the scheme shown in **Figure 1**. It is worth noting that a good proportion of the boards (the 'Zero' family is an exception) use the pinout shown in **Figure 2**, which is by and large compatible with that of the

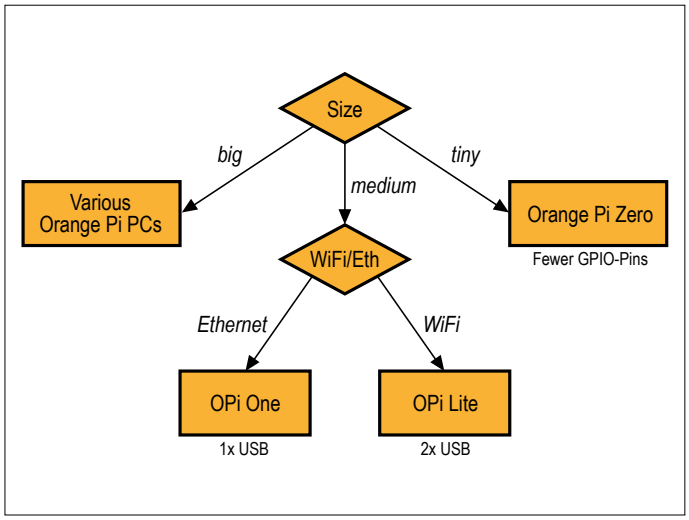


Figure 1. An overview of the family members.

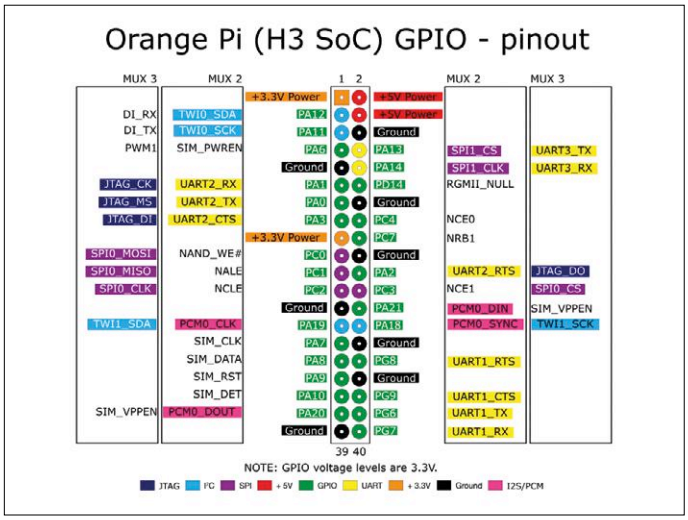


Figure 2. The layout of the 40-way header is in places reminiscent of that on the Raspberry Pi (source: www.orangepi.org).

Raspberry Pi; see also **Figure 3**.
In the following experiments we will use the Orange Pi Lite. Weighing in at some 28.8 grams it is lighter than the Raspberry Pi 3, and it also costs less, at around twelve dollars. Nevertheless it offers two USB ports and a Wi-Fi antenna. The Orange Pi One, on the other hand, has just one USB port instead of two, and Ethernet instead of Wi-Fi, and comes in even cheaper at around nine dollars.

## Configuration

We have to say one thing for the Raspberry Pi Foundation: if you buy their processor boards you will find that support in terms of the software and the ecosystem is excellent. Also, a Raspberry Pi can be powered from any ordinary micro USB charger, which makes it very easy to get started.
The Orange Pi Lite also has a micro USB port but, amusingly enough, cannot be powered from it because of the interface components used: you can find out more about this at [2]. Instead, power at 5 V has to be applied at the power jack or at pins 2, 4 and 6 of the header (pins 2 and 4 to 5 V, pin 6 to ground). This inconvenience is present across the Orange Pi family: if you need to be able to power your computer over the micro USB port, you will have to avoid Shenzhen Xunlong's products.
The next problem is software. The operating system images that can be found on the manufacturer's website at [3] are little more than window dressing. The links sometimes fail to work, and any file you do manage to download will likely be poorly supported.
The Armbian distribution has established itself more or less as a standard among the Orange Pi user community: downloads are available at [4]. Note that the Armbian distribution builds are different for the different members of the Orange Pi family. With our Orange Pi Lite we will be using a Debian-based distribution, but the Ubuntu-based version should work just as well. The differences between the versions are in general only small but they do manifest themselves from time to time and that means that you need to take extra care when testing.
Download the installation image [4a] and write it onto a sufficiently high-capacity SD card. Connect the processor board to an HDMI display, mouse and keyboard, and then apply power. Don't be surprised if it takes a minute or so before you see anything happen: the Orange Pi boards generally take a long time to start up and present an image on the screen.

## Android on board

The internal flash memory of the Orange Pi PC Plus contains a Chinese version of Android that is reportedly quite usable.

## It runs Linux!

Armbian images are classical computer operating systems: when first started up they present the user with a login prompt. The credentials to use at this point vary from image to image: on the Orange Pi Lite you can log in with the user name 'root' and password '1234'.
After the first login the operating system will prompt you to change your password for security reasons and to run an installation wizard. The user name has to be supplied, but the other fields can be left blank by simply repeatedly pressing 'Enter'. After that, you can reboot the system and you will be presented again with Armbian's graphical user interface (see **Figure 4**).
Armbian is Debian, a Linux distribution with a long history which also serves as the basis for the Raspbian distribution used on the Raspberry Pi. It follows, therefore, that at the application level the Raspberry Pi and the Orange Pi are more or less compatible. The first small difference comes in how external hardware is controlled. Where Raspberry Pi uses the WiringPi library, the people of Shenzhen Xunlong bring you WiringOP. The library must be downloaded and compiled before it can be used, which (assuming all goes well) can be done using the following sequence of commands:

```
cd ~
mkdir opigpiolib
git clone https://github.com/zhaolei/WiringOP.git -b
```

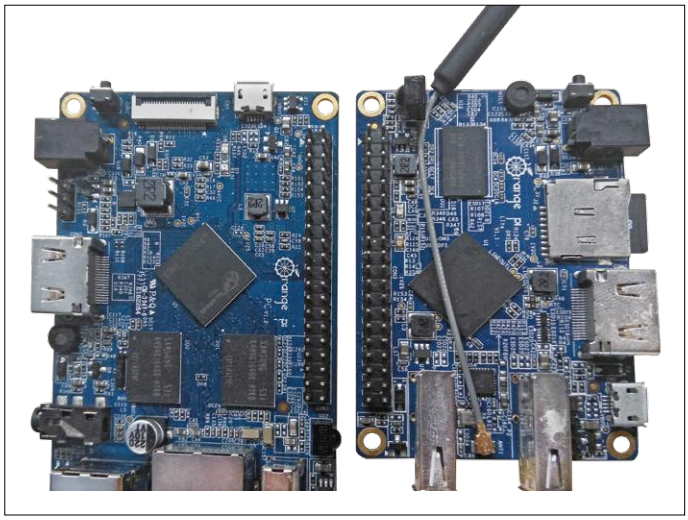

Figure 3. Watch out! The orientation of the headers on the two boards is different.


Figure 4. Just like the Linux we know and love.

```
$ gpio readall
 +-----+-----+----------+------+---+-Orange Pi+---+---+------+----------+-----+-----+
 | BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name     | wPi | BCM |
 +-----+-----+----------+------+---+----++----+---+---+------+----------+-----+-----+
 |     |     |     3.3v |      |   |  1 || 2  |   |   |      | 5v       |     |     |
 |  12 |   8 |    SDA.0 | ALT5 | 0 |  3 || 4  |   |   |      | 5V       |     |     |
 |  11 |   9 |    SCL.0 | ALT5 | 0 |  5 || 6  |   |   |      | 0v       |     |     |
 |   6 |   7 |  GPIO.7  | ALT3 | 0 |  7 || 8  | 0 | ALT3 | TxD3     | 15  | 13  |
 |     |     |       0v |      |   |  9 || 10 | 0 | ALT3 | RxD3     | 16  | 14  |
 |   1 |   0 |     RxD2 | ALT3 | 0 | 11 || 12 | 0 | ALT3 | GPIO.1   | 1   | 110 |
 |   0 |   2 |     TxD2 | ALT3 | 0 | 13 || 14 |   |   |      | 0v       |     |     |
 |   3 |   3 |     CTS2 |   IN | 0 | 15 || 16 | 0 | ALT3 | GPIO.4   | 4   | 68  |
 |     |     |     3.3v |      |   | 17 || 18 | 0 | ALT3 | GPIO.5   | 5   | 71  |
 |  64 |  12 |     MOSI |   IN | 0 | 19 || 20 |   |   |      | 0v       |     |     |
 |  65 |  13 |     MISO |   IN | 0 | 21 || 22 | 0 | ALT3 | RTS2     | 6   | 2   |
 |  66 |  14 |     SCLK |   IN | 0 | 23 || 24 | 0 | ALT4 | CE0      | 10  | 67  |
 |     |     |       0v |      |   | 25 || 26 | 0 | ALT3 | GPIO.11  | 11  | 21  |
 |  19 |  30 |    SDA.1 | ALT4 | 0 | 27 || 28 | 0 | ALT4 | SCL.1    | 31  | 18  |
 |   7 |  21 |  GPIO.21 | ALT3 | 0 | 29 || 30 |   |   |      | 0v       |     |     |
 |   8 |  22 |  GPIO.22 |  OUT | 0 | 31 || 32 | 0 | ALT3 | RTS1     | 26  | 200 |
 |   9 |  23 |  GPIO.23 |  OUT | 1 | 33 || 34 |   |   |      | 0v       |     |     |
 |  10 |  24 |  GPIO.24 |  OUT | 0 | 35 || 36 | 0 | ALT3 | CTS1     | 27  | 201 |
 |  20 |  25 |  GPIO.25 |  OUT | 0 | 37 || 38 | 0 | ALT3 | TxD1     | 28  | 198 |
 |     |     |       0v |      |   | 39 || 40 | 0 | ALT3 | RxD1     | 29  | 199 |
 +-----+-----+----------+------+---+----++----+---+---+------+----------+-----+-----+
 | BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name     | wPi | BCM |
 +-----+-----+----------+------+---+-Orange Pi+---+---+------+----------+-----+-----+
```

Figure 5. The GPIO options on the Orange Pi are rather complicated.

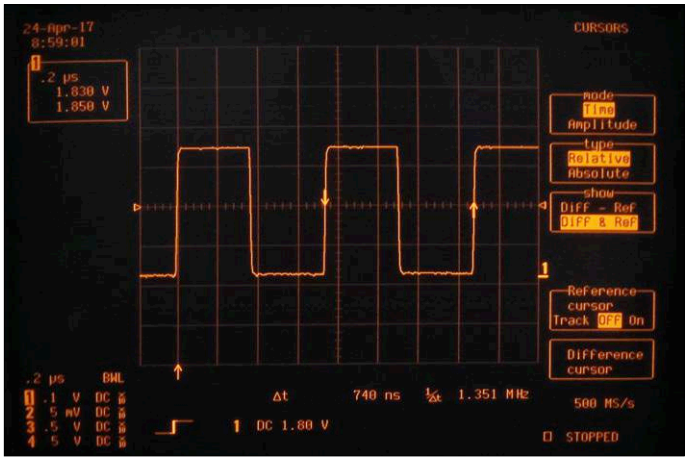


Figure 6. The square wave looks usable.

```
   h3
cd WiringOP
chmod +x ./build
sudo ./build
```

In the next step we look at a small example program that generates a square wave on a GPIO pin. To do this, create a file called 'main.c' with the following contents:

```
#include <wiringPi.h>
int main (void) {
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;) {
    digitalWrite (0, HIGH) ;
    digitalWrite (0,  LOW) ;
    digitalWrite (0, HIGH) ;
    digitalWrite (0,  LOW) ;
 }
  return 0 ;
}
```

The process of compiling this code is largely identical to the familiar process on the Raspberry Pi:

```
$ gcc -Wall main.c -lwiringPi
$ sudo ./a.out
```

## A sensitive flower

Orange Pi boards are extremely sensitive to a supply voltage that is too low: this can lead to crashes when graphics mode is enabled and read failures on the SD card. If you have problems booting up your new computer, one of the first things to check is the voltage drop across the supply cables. Note that the board can draw currents of up to 800 mA!

On some versions of the Orange Pi you will see compilation errors that stem from the fact that the 'pthread' library is not present:

```
$ gcc -Wall main.c -lwiringPi
//usr/local/lib/libwiringPi.so: undefined reference
   to 'pthread_join'
//usr/local/lib/libwiringPi.so: undefined reference
   to 'pthread_create'
//usr/local/lib/libwiringPi.so: undefined reference
   to 'pthread_cancel'
collect2: error: ld returned 1 exit status
```

This in turn results from misconfiguration of the gcc compiler, which is not automatically linking in the pthread library. To solve this it is simply necessary to add another option to the compiler command line as follows:

```
$ gcc -Wall main.c -lwiringPi -pthread
$
```

It should be possible to obtain the layout of the individual pins on the GPIO header using the command '`gpio readall`'. For simplicity we give an example result from this command run on a 40-pin Orange Pi PC Plus in **Figure 5**.

We can now team up the processor board with a duo of instruments: a modulation domain analyzer and a digital storage oscilloscope. **Figure 6** and **Figure 7** show the results you might get.

### Enabling the UART

In our next example we will show how to read an RFID card. The Seeed RFID transponder module, which operates at 125 kHz, is ideal for this task as it only requires a small amount of configuration. It is available from Mouser (order code 713-113020002) for around twelve dollars including antenna. **Figure 8** shows the electronics required to connect up this module: whereas the Raspberry Pi 3 is to some extent 5 V tolerant, this is not the case for the Orange Pi and a (not especially sophisticated) level shifting circuit is required.

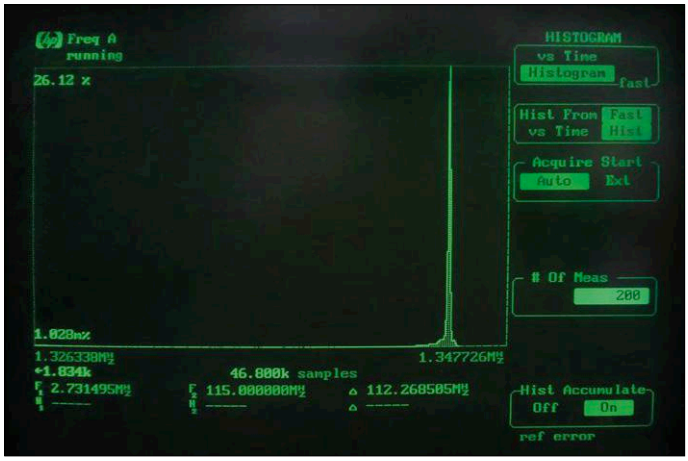


Figure 7. The stability of the waveform, too, leaves not much to carp about.

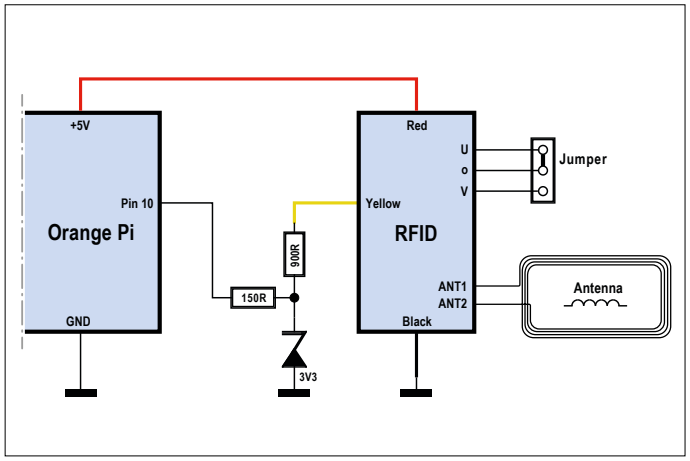


Figure 8. A voltage level shifter is essential.

Allwinner supplies the hardware configurations for its SoCs in the form of a binary file which is loaded into the device by the kernel as part of the boot process to set up the voltage regulators, clock sources and various other subsystems. The UART used in our program example is not normally accessible, as can be seen from a look at the /dev directory where only one UART (ttyS0) is present:

```
$ ls -l

. . .

crw--w---- 1 root tty      4,   9 Apr 12 06:07 tty9
crw--w---- 1 root tty    251,   0 Apr 12 06:08 ttyS0
crw------- 1 root root   248,   0 Apr 12 06:07 tv
```

To get around this problem the first step is to modify the configuration file that is loaded into the hardware during the boot process. The scripts used for the various boards can be found on the SD card in the directory '/boot/bin', which under Armbian looks like this:

```
$ ls
aw-som-a20.bin          bananapiprolcd7.bin  lime-a10-lcd.bin  nanopiair.bin       orangepilite.bin    pcduino2.bin
bananapi.bin            beelinkx2.bin        lime-a10.bin      nanopim1.bin        orangepione.bin     pcduino3.bin
bananapilcd7.bin        cubieboard.bin       lime-a33.bin      nanopim1plus.bin    orangepipc.bin      pcduino3nano.bin
bananapim1plus.bin      cubieboard2.bin      lime.bin          nanopineo.bin       orangepipcplus.bin
bananapim1pluslcd7.bin  cubieboard2dual.bin  lime2-emmc.bin    olinux-som-a13.bin  orangepiplus.bin
bananapim2plus.bin      cubietruck.bin       lime2.bin         orangepi.bin        orangepiplus2e.bin
bananapipro.bin         lamobo-r1.bin        micro.bin         orangepi2.bin       orangepizero.bin
```

Meanwhile, the file that is actually used to initialize the board lies in the directory immediately above this one under the name 'script.bin'. However, this is in fact a symbolic link, and the `realpath` command can tell us the real file to which it corresponds:

```
$ pwd
/boot

$ realpath script.bin
/boot/bin/orangepipcplus.bin
```

These configuration files can exist in two different forms: as well as the binary version that the processor itself understands there is a '.fex' format, which is human-readable. We can convert the binary file (the one for our processor is, as we discovered above, called '/boot/bin/orangepipcplus.bin') to readable form using the following command:

```
$ sudo bin2fex orangepipcplus.bin orangepipcplus.fex
[sudo] password for spotter: . . .
fexc-bin: orangepipcplus.bin: version: 1.2
fexc-bin: orangepipcplus.bin: size: 36392 (84
   sections), header value: 36392
```

The structure of a .fex file is reminiscent of that of a .ini file, and is documented in detail at [5]. Of interest to us at the moment is how the UART that is connected to the RFID reader is configured. Edit the block so that it contains the following information:

```
[uart3]
uart_used = 1
```

```
uart_port = 3
uart_type = 4
uart_tx = port:PA13<3><1><default><default>
uart_rx = port:PA14<3><1><default><default>
uart_rts = port:PA15<3><1><default><default>
uart_cts = port:PA16<3><1><default><default>
```

For comparison, a block that describes a disabled UART looks like this:

```
[uart2]
uart_used = 0
uart_port = 2
. . .
```

However, it is not sufficient simply to modify the .fex file and reboot, as this will not change the behavior of the machine. We first have to convert the edited .fex file back into a .bin file, using the command `fex2bin`.

```
$ sudo fex2bin orangepipcplus.
fex orangepipcplus.bin
```

And now at this point we reboot the machine and admire the fruits of our handiwork: the new serial port appears in the /dev directory

```
crw--w---- 1 root tty
   251,   0 Apr 24 13:34 ttyS0
crw-rw---- 1 root dialout
   251,   3 Apr 24 13:34 ttyS3
crw------- 1 root root
   248,   0 Apr 24 13:34 tv
```

Enabling and configuring a UART is just one of the possibilities opened up by the ability to edit a .fex file. Another example would be changing the processor clock frequency to gain more processing speed or to save power, and more information on this topic can be found in the relevant forums. But meanwhile, let's crack on with reading some RFID data.

**Listing 1. Reading an RFID card over a UART.**

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

#include <wiringPi.h>
#include <wiringSerial.h>

int main ()
{
  int fd ;
  int count ;
  unsigned int nextTime ;

  if ((fd = serialOpen ("/dev/ttyS3", 9600)) < 0)
  {
    fprintf (stderr, "Unable to open serial device: %s\n", strerror (errno)) ;
    return 1 ;
  }

  if (wiringPiSetup () == -1)
  {
    fprintf (stdout, "Unable to start wiringPi: %s\n", strerror (errno)) ;
    return 1 ;
  }

  nextTime = millis () + 300 ;

  for (count = 0 ; count < 256 ; )
  {
    delay (3) ;

    while (serialDataAvail (fd))
    {
      printf (" -> %3d", serialGetchar (fd)) ;
      fflush (stdout) ;
    }
  }

  printf ("\n") ;
  return 0 ;
}
```

```
tamhan@TAMHAN14: ~
$ sudo ./a.out
 ->   2 ->  48 ->  49 ->  48 ->  70 ->  53 ->  67 ->  67 ->  65 ->  51 ->  66 ->
  65 ->  51 ->   3

 ->   2 ->  48 ->  49 ->  49 ->  48 ->  57 ->  49 ->  69 ->  65 ->  67 ->  68 ->
  65 ->  55 ->   3
```

Figure 9. Two different RFID cards duly recognized: press 'Enter' after each scan to split up the lines of data.

## Scanning an RFID card

All we need to make a fully-functional card reader is a program to read serial data delivered from the UART. Its first job is to open the connection to the UART: see the first part of **Listing 1** [6].

The next step is to read in the incoming data. In view of the ever-increasing importance of RFID technologies the author feels a brief digression is justified. There are two basic types of reader. The first group, to which our module belongs, reads all the information it can find on the card and sends this as a serial data stream to whatever it is connected to. Readers in group number two implement a Wiegand interface. This involves a protocol that communicates using pulses on one of two data lines.

## Risk of collision!

When several programs try to access the same serial bus at the same time, collisions can occur. Experience shows that the operating system does not detect this and the individual programs only receive fragments of the data they are expecting.

Despite being limited to data lengths of a mere 26 bits or 31 bits, the Wiegand protocol shows no sign of going away and so remains relevant in embedded control applications. Most RF readers, in fact, are available in both Wiegand and UART editions, with the UART edition being the more sophisticated. For simplicity our program gathers data one byte at a time (see Listing 1) and displays them in the terminal window.
We are now ready for a first test. Connect the card reader to the computer, compile the program in the usual way, and bring a card near to the red reader coil. The display in the terminal window should look something like **Figure 9**.

### Going further

As we have seen, the Orange Pi is a classical embedded controller board that runs the Linux operating system. That means that it supports a range of protocols such as I²C and SPI in much the same way as the Raspberry Pi and its friends.
Anyone who is willing to get to grips with the Linux I²C and SPI APIs (or who is willing to use the WiringOP library) will find it easy to connect a diverse range of peripheral devices, including displays and sensors in all shapes and sizes.

### Conclusion

When the first Orange Pi computer saw the light of day, many had their doubts regarding the expected longevity of the manufacturer. Two years later, and Shenzhen Xunlong is still going strong and has brought many new models to market. Also, practically all of the devices are still readily available.
Using an Orange Pi is particularly beneficial in mass-produced systems: a cost reduction to the tune of 50 % or even more is not something to be sniffed at. In small quantities there is the question of development costs: if you are not an expert Linux user, you will probably be better off with the support of the larger Raspberry Pi community. ⏮

(160456)

### Web Links


[1] www.orangepi.org/
[2] https://www.youtube.com/watch?v=4zICevDOhr8
[3] www.orangepi.org/downloadresources/
[4] www.armbian.com/download/
[4a] https://dl.armbian.com/orangepilite/
[5] http://linux-sunxi.org/Fex_Guide
[6] www.elektormagazine.com/160456