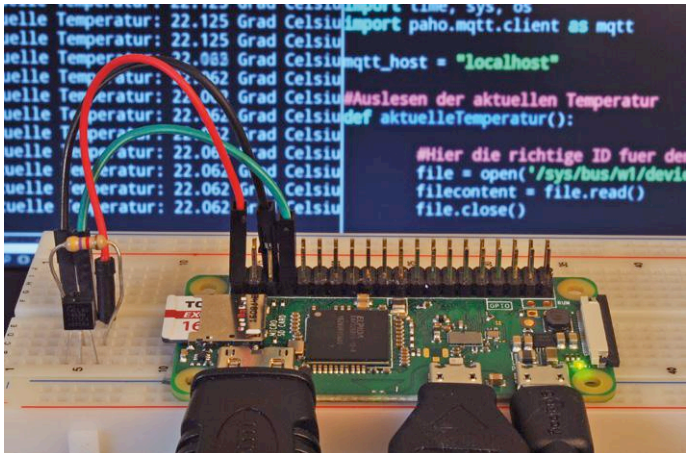
Small but Perfectly Formed:
the Raspberry Pi Zero W
First steps for the newest arrival in the Pi family

The mini version of the Raspberry Pi with the suffix Zero has been with us since the end of 2015. February 2017 saw the advent of the Zero W (W for 'wireless') model, enhanced with WLAN and Bluetooth capabilities.
In this article, we examine how to get started with the new board started, what it does, and the advantages of adding wireless technology to this tiny single-board computer. As a practical example, we'll hook up a temperature sensor and distribute the data by WLAN using the MQTT protocol.



By **Markus Ulsass** (Germany)

At approximately €25 / £10 / $20 (prices vary from country to country) the cost of the new Raspberry Pi Zero W may be double that of its basic predecessor but the higher price is fully compensated by its cordless WLAN and Bluetooth capabilities, which are indispensable for so many projects. Low-cost robotic, smart home and Internet of Things (IoT) projects become feasible, thanks to the compact form factor and very low power consumption of the Zero W. The single-core Broadcom SoC BCM2835 chip is clocked at 1 GHz (512 MB RAM). This makes the Zero W not as speedy as its bigger brother Raspberry Pi 3 but nevertheless more than adequate for most applications.
The British shop Pimoroni [1] sells to customers worldwide; their prices are keen even when you include the postage cost and delivery is snappy at the time of writing.
The Zero W is equipped with Mini HDMI and Micro USB On The Go (OTG) ports; the single board device is supplied with a bare (unpopulated) 2x20-pin (HAT-compatible) GPIO interface. This needs a 40-way header to complete and for that reason, if you don't already have something suitable in your store cupboard, it's worth ordering the corresponding accessory parts for your Pi at the same time (dealers offer these as an adapter kit).

Installing the operating system and what comes next

Having got hold of a Zero W, together with all the necessary adapters and a power supply (see the shopping list panel for the necessary accessories), we now need to install the operating system on a micro SD card with at least 8 GB of memory.

To do this we download the current file Image of *Raspbian Jessie with Pixel* from [2]. For transferring the Image onto an SD card we need a card burner tool such as *Etcher*, which can be downloaded at [3] (**Figure 1**). Incidentally, a key advantage of this program is that it is available for all operating system platforms and the downloaded Image does not have to be unpacked. Having installed and started the program, we next select the appropriate Image and transfer it then to the chosen SD card.
After this we insert the SD card into the Zero W, we use an adapter to connect the monitor (using HDMI) together with a keyboard and a mouse (using USB) and finally connect up the power supply. Take special care here: the connector for this is the right-hand one of the two Micro USB sockets, assuming you have the board horizontal in front of you, with the SD card on the left-hand side (**Figure 2**).
The Raspberry Pi Zero W should now start up in the Pixel desktop of the Raspbian operating systems. And with that, the task of installing an oven-ready Linux operating system on this super-small computer is complete.

Figure 1. Etcher is an SD Card burning tool for loading the Raspbian image.

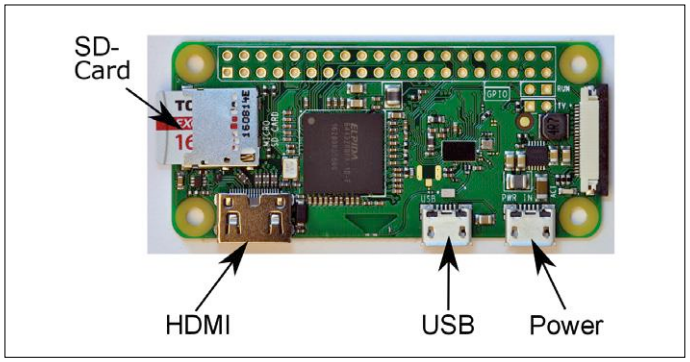


Figure 2. Connections for the Raspberry Pi Zero W, with the SD card at far left.

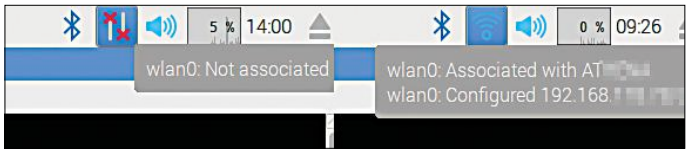


Figure 3. Before (left) and after (right). The Zero W should link up to the WLAN following configuration. Frequently you'll see additional information when you move the mouse over the symbols.

Fine tuning

If you now open a Terminal (*CTRL+T* or click on the Monitor icon in the upper navigation bar), you'll see that the default language, regional settings and keyboard layout assume that you are located in Britain. If you need to change these, type *sudo raspi-config* into the Terminal. This opens the *Raspberry Pi Software Configuration Tool*, in which you use the cursor to select (only if necessary) *4 Localisation Options* and press *Enter*. If, for example, you are in Germany, just select *I1 Change Locale de_DE.UTF-8 UTF-8* with the space bar and use the *Tab* to select *<OK>*. In the following window set *de_DE.UTF-8* as the *Default locale for the system environment* and confirm once more using *<OK>*. You can now restart the Pi Zero W using *sudo reboot*.

Your system's default language is now German but the keyboard layout is still incorrect. Call up *sudo raspi-config* once more, select *4 Localisation Options* and this time select *I3 Change Keyboard Layout*. There you may choose *Generische PC-Tastatur mit 105 Tasten (Intl)*, in the next window *Andere*, then *Deutsch* and confirm with *<OK>*. Move the cursor to *Deutsch – Deutsch (T3)* and confirm once more using the *Enter* key. In the next window confirm again with *<OK>* or *<Nein>* as appropriate and finally return to the main menu. When you leave this using *<Finish>*, you enter a *z* on the Console and should find it set up for the German keyboard layout. Obviously you can follow a similar process for any other languages and regional settings that you wish to specify — the above information is given with Elektor Magazine's wide distribution in Europe in mind.

As the Zero W has a default password of *raspberry* for the user *pi*, it's vital that you change these settings. This is done using *sudo raspi-config* and the entry *1 Change User Password.* Provide a new password and make a written note of this somewhere.

In order to identify your Zero W subsequently on the network unambiguously by name, you should change this (the default is *raspberrypi*). To do this you edit the relevant entry in two text files, using the *nano* text editor. First enter *sudo nano /etc/hostname* and in the text file replace the entry *raspberrypi* with *zerow*. Save this with *CTRL+O, Enter* and finalize with *CTRL+X*. Next you open the second file with *sudo nano /etc/hosts* and search for the last entry with *127.0.0.1*. Here too you again replace *raspberrypi* with *zerow*. Now save it again using *CTRL+O, Enter* and finalize with *CTRL+X*.

External connections

In order to be able to link our Zero W to a local network using Wi-Fi, we now have to enter the data for the WLAN router (SSID, password). To do this click on the symbol with den two vertical lines and red crosses at upper right on the navigation bar (**Figure 3**). Choose the correct access point, enter the password and the Raspberry Pi Zero W should connect to the WLAN.

With a fresh Image it makes sense update the package

Listing 1. Flashing LED Python script.

```
#!/usr/bin/python
#LED_Blink.py
import RPi.GPIO as GPIO #link GPIO library
import time #library required for Sleep

LED = 14
GPIO.setmode(GPIO.BCM) #use BCM-GPIO labels
GPIO.setwarnings(False) #disable warnings
GPIO.setup(LED, GPIO.OUT) #use LED pin as output
PAUSEON = 1.0 #On time
PAUSEOFF = 1.0 #Off time

while True:
    GPIO.output(LED, GPIO.HIGH) #LED on
    time.sleep(PAUSEON) #On time
    GPIO.output(LED, GPIO.LOW) #LED off
    time.sleep(PAUSEOFF) #Off time
```

SHOPPING LIST

- Raspberry Pi Zero W
- Micro SD card (min. 8 GB)
- HDMI to mini HDMI adapter
- Micro USB OTG adapter (e.g. Delock 65296)
- Micro USB power supply (min. 1 A)
- 2 pcs 20-pin header strips (0.1" pitch)
- Breadboard
- Jumper cables
- LED (red)
- Resistor 470 Ω
- Resistor 4.7 kΩ
- Temperature sensor DS18B20

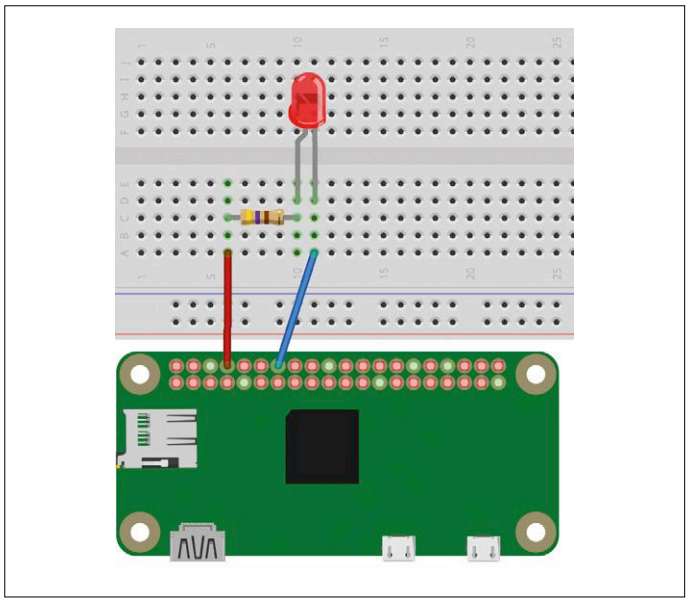


Figure 4. Fritzing sketch for hooking up the LED to the Zero W.

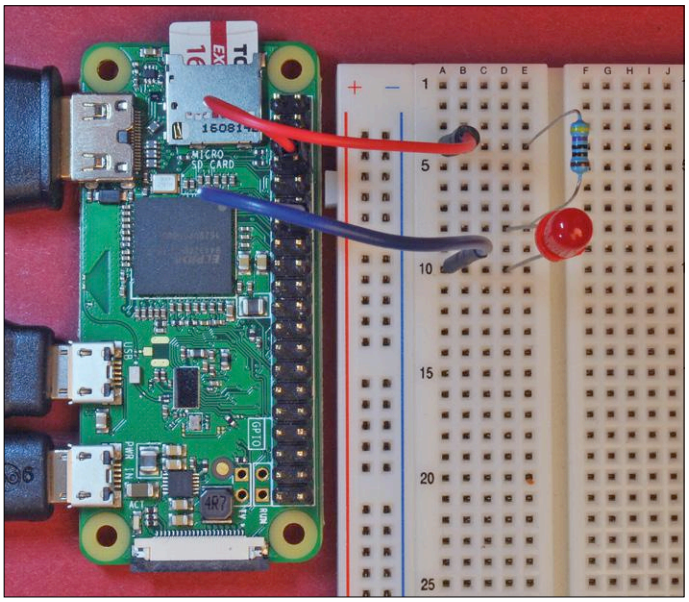


Figure 5. Zero W cabled to the LED and resistor on the breadboard.

sources and the system to the most recent version. We do this with *sudo apt update && sudo apt upgrade*, again in the Terminal. Confirm the question whether to proceed with *Y* and wait a couple of minutes until all of the downloads are complete.

'Hello World' *ersatz* with an LED

With these initial preparations finished, we will skip typing 'Hallo World' and instead get started by connecting some simple hardware. First we need to solder the 40-pin header to the Zero W board, so that we can use some jumper wires and breadboard for controlling an LED.

Before connecting the resistor and the LED we need to shut down the Zero W completely, to avoid any short circuits or other harmful effects that might occur when hooking up hardware direct to the GPIO connections. We take care of this using *sudo shutdown –h now* on the Console or by clicking on the raspberry symbol on the navigation bar, selecting *Shutdown* in the drop-down menu and choosing *Shut Down* in the window that follows.

Forgoing the urge to type 'Hello World', we employ GPIO14 for control purposes along with a 470-Ω resistor and a red LED. You can find the Pinout for the Zero W at [4] for example. We connect GPIO14 to the resistor, attach this to the anode of the LED and link its cathode to one of the Ground Pins on the Pi Zero W. **Figure 4** shows the Fritzing sketch and **Figure 5** demonstrates the assembly as it actually appears.

For a program to make the LED flash we use a small Python script (*LED_Blink.py*). It is printed out in **Listing 1** but you can also download it from the Elektor website at [5].

We can generate the code using once more the Editor *nano* or alternatively using the somewhat more convenient *Geany* (*Menu ➞ Development ➞ Geany*), which is included in the Raspbian operating system by default. Note that in Python you make indentations (as in listings for instance) using the space bar, otherwise you'll see the message IndentationError. After saving the program we call it up on command line in Terminal with *python LED_Blink.py*. The LED should now flash once a second.

We close the program with *CTRL+C*.

Temperature measurement with the DS18B20

A popular and simple-to-connect digital temperature sensor is the DS18B20 from Maxim Integrated. First we shut down the system in order to link the circuitry safely to the breadboard and the Zero W.

The left-hand Pin (with the part number facing upwards) of the DS18B20 is linked to the 3.3 V power supply of the Zero W, the right-hand one to a Ground Pin. A jumper wire goes from the center (data) Pin to GPIO4. Lastly we provide a pull-up resistor between the data Pin and the supply voltage to supply 'parasitic' power to the sensor (see the Fritzing schematic in **Figure 6**).

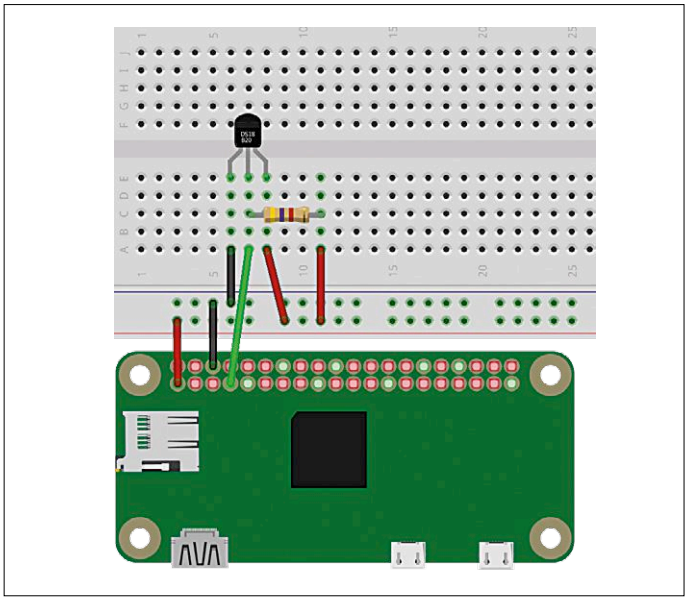


Figure 6. Wire connections for the DS18B20 temperature sensor.

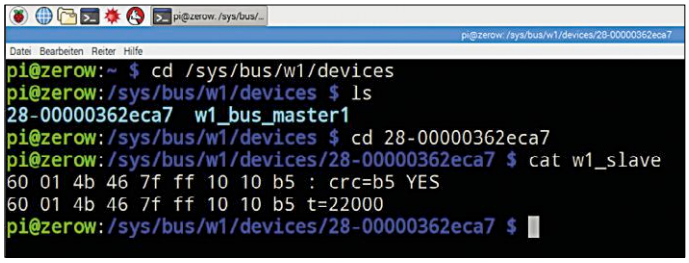


Figure 7. Testing the sensor after connection.

After making these connections and a quick check whether the wiring is correct, we can restart our Zero W. But wait: before we can use the one-wire temperature sensor, we must first integrate (include) it with the system. We do this using *sudo raspi-config* under *P7 1-Wire*, after which we answer *Yes* to the question *Would you like the one-wire interface to be enabled?* We then close the configuration program and must reboot the system.

It's time now to test whether the sensor has been embedded into the system correctly and is connected (available). For this we use the Terminal to switch into the *cd /sys/bus/w1/devices* Folder and list its contents with *ls*. This should make an alphanumeric combination appear that represents our sensor.

We now navigate to *cd <Sensor-ID>* (in this set-up we're using *cd 28-00000362eca7* in the relevant subfolder) and read out the sensor's current data using *cat w1_slave*, which is displayed in two lines (**Figure 7**). Here t=22000 indicates a temperature of 22 degrees Celsius (centigrade). With this now done, the sensor is configured correctly and fully operational.

Sharing our temperature sensor with the rest of the world using MQTT

There will be times when we wish to forward sensor values captured with our Raspberry Pi Zero W to other recipients. For this we employ the streamlined MQTT protocol – a data protocol for machine-to-machine communication. MQTT uses a so-called publish/subscribe (pub/sub) system. With this for instance sensors can 'publish' data on specific topics and users known as Clients can sign up ('subscribe') for these topics and receive the data. The topics are memorized to the data file using included slashes (an example would be *Apartment/Kitchen/Temperature*) and even wildcards are possible. Instead of files, however, we specify different sensors or actuators.

Our own temperature sensor might transmit the temperature values it captures under the topic */home/outdoors/temperature/sensor1*, for which a Client (for example a home automation system like openHAB, fhem or Node-Red) has subscribed.

For MQTT we require an intermediary — called the *Broker* in the MQTT jargon — that organizes the data flow between sender and receiver. A popular Broker is Mosquitto, which we install on our Zero W using *sudo apt install mosquitto mosquitto-clients*.

To check that everything has been installed properly we can test both transmitting and receiving data on the same device. For this we first start up the Broker with *sudo systemctl start mosquitto* on the Console. In order to activate it automatically the next time we boot up we also enter *sudo systemctl enable mosquitto*.

Next we'll do some simulating on the Console with *mosquitto_sub -h localhost -t /sensor1* (in the examples that follow we will save ourselves some writing effort and abbreviate the Topic shown above): first an information subscriber, who will later receive data (mosquitto_sub). First of all nothing happens, because the system is of course waiting for some data. We open a second Terminal window and enter *mosquitto_pub -h localhost -t /sensor1 -m "22"* (**Figure 8**). Doing this is the same as if we were transmitting

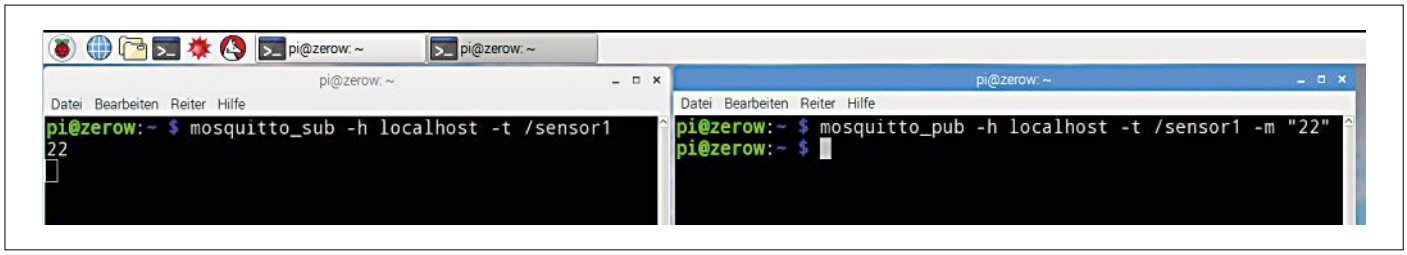


Figure 8. Testing the MQTT function.

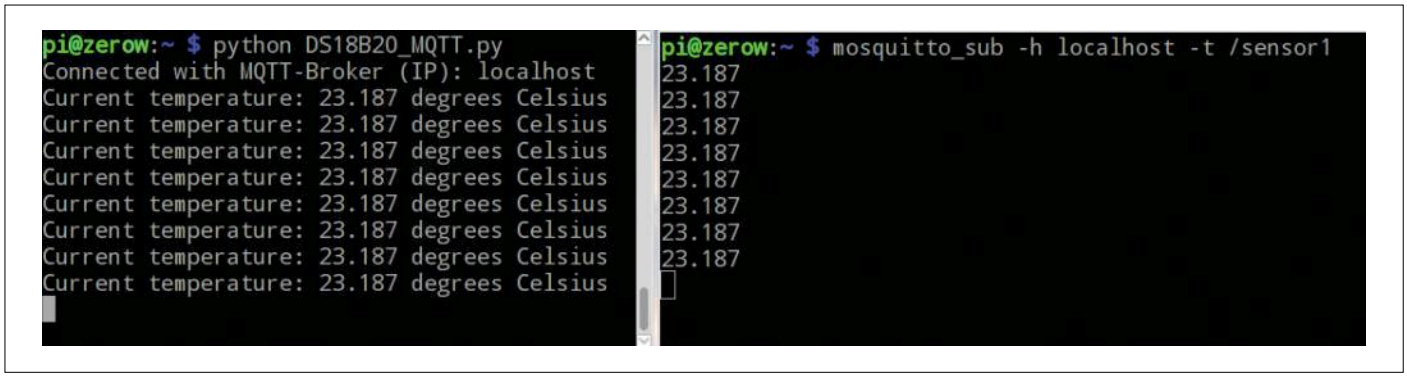


Figure 9. Testing MQTT data transfer with our temperature sensors: local output (on the left), received data using MQTT (on the right). [Celsius = centigrade]

a temperature value (mosquitto_pub) or respectively Publishing it to remain in the MQTT vocabulary. Following this a *22* should appear in the first window. Our system for publishing the value from our temperature sensor is now up and running, and in a final step we can automate it even further.

Transmitting temperatures by MQTT

We employ a Python script for automating data transfer by MQTT. For this we use a Library that we install using *sudo pip install paho-mqtt*.

Our script (**Listing 2**) first imports the required Modules and defines the IP address of the computer on which Mosquitto Broker is running ('localhost' for the local broker, actual IP address for a remote MQTT broker).

The current temperature value is then read out in the Function `currentTemperature()`. For the sensor ID given in the sample code you need to enter the ID of the sensor used of course. The date read out is converted into a suitable value and displayed as the value returned by the Function.

Following this a connection is made to the local (or remote) MQTT broker (mqtt_host). In the *while* loop the temperature value (`temperature) is then polled every two seconds`, displayed on the command line and transmitted to the MQTT broker under the topic */sensor1*.

After we have created the program with an Editor or downloaded it (according to your preference), we open a Terminal and initiate it with *python DS18B20_MQTT.py*. The command line should now report a connection between the MQTT broker and the local Zero W, followed by a display of the current temperature value (**Figure 9**). Now we generate a second Terminal screen and Subscribe ourselves again by MQTT-Client to the Topic */sensor1* on the local computer: *mosquitto_sub -h localhost -t /sensor1*.

Listing 2. Python script for transmitting the temperature by MQTT.

```
#!/usr/bin/python
#DS18B20_MQTT.py
#Import of required module
import time, sys, os
import paho.mqtt.client as mqtt

mqtt_host = "localhost"

#read out current temperature
def currentTemperature():

        #enter correct sensor ID here
        file = open('/sys/bus/w1/devices/28-00000362eca7/w1_slave')
        filecontent = file.read()
        file.close()

        #convert temperature value
        temperaturestring = filecontent.split("\n")[1].split(" ")[9]
        temperaturevalue = float(temperaturestring[2:]) / 1000
        return(temperaturevalue)

def on_connect(client, userdata, flags, rc):

    print("Connected with MQTT Broker (IP): " + mqtt_host)

client = mqtt.Client()
client.on_connect = on_connect

client.connect(mqtt_host, 1883, 60)

client.loop_start()

while True:
    time.sleep(2)

    temperature = currentTemperature()
    print("Current temperature: " + str(temperature) + " degrees
Celsius")
    client.publish("/sensor1", temperature)
```

And that's still not everything...

With the Zero W the Raspberry Pi Foundation has delivered a truly fantastic board, right on time for the fifth birthday of the original Raspberry Pi. Who would have thought, five years ago, that a Linux nanocomputer like this could be possible, in such a tiny form factor and for such an affordable price?

The USB port, blocked on predecessor versions by the keyboard or the WLAN module, is now available for additional functionality. Even the current consumption is very modest and is already almost in the same league as self-powered Internet of Things or Smart Home devices. A comparison chart of the current drawn by various types of Raspberry Pi can be found at [6].

With such amazing benchmark parameters it's not surprising that the board can be found only in limited quantities at present. That was also the case with the first Raspberry Pi and therefore we can only hope that the availability of Zero W boards improves soon. ⏮

(160451)

Web Links

[1] https://shop.pimoroni.com/

[2] www.raspberrypi.org/downloads/raspbian/

[3] https://etcher.io/

[4] https://pinout.xyz/

[5] www.elektormagazine.com/160451

[6] https://blog.adafruit.com/2017/03/03/how-much-power-does-pi-zero-w-use-piday-raspberrypi-raspberry_pi/